

U.S.N.A. ---Trident Scholar project report; no. 336 (2005)

Normal Mode Analysis of the Chesapeake Bay

by

Midshipman 1/C Grant I. Gillary, Class of 2005
United States Naval Academy
Annapolis Maryland

(signature)

Certification of Advisor(s) Approval

Professor Reza Malek-Madani
Director of Research & Scholarship

(signature)

(date)

Assistant Professor Kevin McIlhany
Physics Department

(signature)

(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research & Scholarship

(signature)

(date)

USNA-1531-2

Abstract

Keywords: Chesapeake, Finite Differences, Normal Modes, FEMLAB®, MATLAB®

The purpose of this project was to find the normal modes for a mathematical model of the Chesapeake Bay geometry. The method used, normal mode analysis, was similar to that of Eremeev et al. [1992a] and Lipphardt et al. [2000]. Normal mode analysis uses a truncated basis set of velocity fields to approximate the flow for a specific body of water. The approach taken in this project uses the three modes described by Lipphardt et al. [2000] for application to Monterey Bay with one mode corresponding to flows with streamline potentials, one mode to flows with velocity potentials and an inhomogeneous mode which takes into account forcing functions at the boundaries. In practice linear combinations of these three normal modes are used to provide a complete picture of the flows in a specific body of water from limited amounts of empirical or model data. The ability to accurately fill in partial empirical velocity fields can be used to provide the military with current data in coastal waters for mission planning or navigation. This approach is also useful for studying the spread of wet life in a body of water.

There is no analytical solution for the normal mode equations with a boundary as complicated as the Chesapeake Bay, which has 11,684 miles of shoreline but is only 189 miles long and 30 miles wide. Therefore, the normal modes have been calculated using a finite differencing method in MATLAB® alongside the finite element based program FEMLAB®. Convergence and accuracy of the solutions were first tested on the square, the circle and the equilateral triangle geometries, then the normal mode equations were solved for a representation of the Chesapeake Bay. This project has produced two useful products: the normal modes of the Chesapeake Bay and open source MATLAB® code that uses the finite difference method.

Acknowledgements

First and foremost I would like to thank my advisors, Professor Reza Malek-Madani and Assistant Professor Kevin McIlhany, for all the time and effort they have put into mentoring me and guiding me through the research process. Assistant Professor Irina Popovici for helping me learn mathematical rigor and fill in gaps in my mathematical background. Lisa Beckett and the CADIG staff for their help and support throughout the year with any of my computing problems. Tom Gross for his work with NOAA on the Quoddy finite element formulation for the Chesapeake Bay. Dr. Kirwan and Dr. Lipphardt for their support during this project and their work on the Monterey Bay.

Most importantly, I would like to thank my friends and family who have consistently supported me even though they had to put up with my strange disappearance for large portions of the past two semesters. Surviving the past year would have been impossible without the significant contributions made to both this project and my life by all those mentioned above, thank you.

Contents

Chapter 1: Introduction	4
Chapter 2: Background	7
Chapter 3: The Finite Difference Method.....	16
Chapter 4: Testing and Analysis of the Dirichlet Finite Differencing Scheme.....	42
Chapter 5: Testing and Analysis of the Neumann Finite Differencing Scheme.....	59
Chapter 6: Vorticity Potentials of the Chesapeake Bay.....	74
Chapter 7: Velocity Potentials of the Chesapeake Bay.....	85
Chapter 8: Analysis of the Vorticity and Velocity Potentials of the Chesapeake Bay.....	95
Chapter 9: Inhomogenous Potentials of the Chesapeake Bay.....	103
Chapter 10: Conclusion.....	109
Endnotes.....	111
Bibliography.....	115
Appendix A: MATLAB® Code.....	117

Chapter 1

Introduction

During the past decade a surge of data has become available concerning coastal waterways and estuaries. This influx is due to numerous methods of data collection which are being implemented: HF or high frequency radar, Lagrangian drifters, synthetic aperture radar, new generation passive remote-sensing platforms and towed arrays which can collect information on current velocity fields of a ship's wake [Lipphardt et al., 2000].¹ This gives scientists a significantly improved picture of current flow throughout many coastal regions. However, the data collection techniques are not always capable of covering all of the areas of interest and so provide only partial data for the regions in which they are implemented. A method to fill in the gaps in this data would greatly increase the velocity field's usefulness in numerous applications.

There have been two studies in recent years [Eremeev et al., 1992b; Lipphardt et al., 2000] which have tested methods for filling in gaps in the velocity field for the data they obtained.^{2,3} In the paper by Eremeev, the data was received from autonomous drifting buoys (ADB) in the Black Sea was used to extrapolate velocity fields for this closed body of water using what was later termed by Lipphardt as normal mode analysis (NMA).^{4,5} Eremeev and his

collaborators found that this process allowed him to model the large scale currents measured by the ADB's with a relatively small number of modes. One set of data required only 76 modes and yet still accounted for 70% of the kinetic energy associated with the observed field. Lipphardt used this same method to fill in gaps of velocity fields in Monterey Bay. They used a 39 by 39 grid model with HF radar data collected at each point on the grid.⁷ This grid spacing corresponds to a minimum spacing between data points of two kilometers. The NMA model used by Eremeev did not account for normal flow through an open boundary such as the Monterey Bay's boundary with the Pacific Ocean.⁸ Lipphardt extended the NMA used by adding a mode to account for the

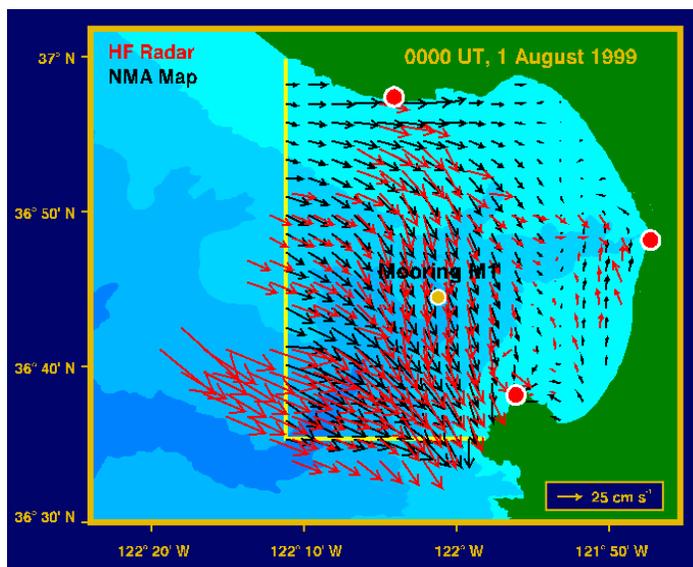


Figure 1.1 Taken from Bruce Lipphardt's website with preliminary NMA results for Monterey Bay.

flow between Monterey Bay and the Pacific.⁹ Combining the NMA of the bay with HF radar observations, they generated velocity fields for the Bay called Nowcasts. Figure 1.1 shows a Nowcast with data taken from the HF radar and combined with NMA. Lipphardt and Kirwan used 12 modes where each had a kinetic energy equal to at least 15% of the mean. These

twelve modes were used to fill out the velocity field obtained by the HF radar.¹⁰

The ability of NMA to efficiently fill in missing current data for coastal regions has many military and civilian applications. As the US Navy begins to increase its focus on the littoral environment, knowledge of surface currents in coastal waterways would significantly improve navigation. Once NMA has been applied to a significant number of waterways, it will become

invaluable. Realistically, when NMA becomes widespread, the Navy could provide its ships with accurate current data for numerous waterways around the world while only taking data from a small percentage of the total area. This would provide an efficient and cost effective method for providing useful data for both navigation and mission planning.

In the civilian context, NMA would provide a way to help track contaminants and keep tabs on wet life in a region. NMA's ability to complete the picture of the surface current is necessary for accurate computation of packet trajectories in any waterway. For example, if another Exxon Valdeze accident occurred, or chemical or biological agents were introduced into a waterway, a complete picture of the currents in that area would allow those in charge to calculate the dispersion of the contaminant and locate the hardest hit areas for both clean up and evacuation. Currents also play an important part in the health of an area's wet life. For the Chesapeake Bay, the waterway with which this project is concerned, NOAA is currently administering a project to restore the oyster population, which was almost fished to extinction during the 18th and 19th centuries. For this project, current data is important for determining placement of reefs and movement of the oyster population. "If reefs are to be a source of spat for shell plantings, and for sustainability of the reef itself then salinity, flow regime and basin morphology will be important considerations. Hydrodynamic models or drifter studies will be useful in determining fate of larvae from any proposed reef site" [Chesapeake Research Consortium 1999].¹¹

Chapter 2

Background

The goal of this project is to produce the normal modes of the Chesapeake Bay as defined in equations (2.12), (2.13) and (2.14) with their proper boundary conditions. Enough normal modes for each equation will be found so that each major portion of the Bay is touched by at least ten of the modes. The secondary goal of this project is to create programs in MATLAB® which can be used by others interested in normal mode analysis to find the modes of other geometries. Since MATLAB® is a readily available product, all programs produced in this

language are portable to any system with MATLAB® installed. In order to reach these goals, a finite differencing scheme was applied to the Chesapeake Bay in order to produce a numerical approximation of the modes. The finite differencing programs were coded in MATLAB®. In order to validate the finite difference solutions, comparisons with an off the shelf finite element program, FEMLAB®, were used.

Background: This section will give background information for understanding how normal mode analysis has been developed with respect to mapping velocity fields in both Lipphardt and

Eremeev.^{12,13} There are two fundamental ideas in the normal mode analysis: basis sets and spectral approximations.

Normal modes are a type of a basis set for a linear vector space. A linear vector space is a set of physical or mathematical objects which adhere to a certain set of laws for addition and multiplication by scalars in the set of real or complex numbers. The five laws for vector addition include: vector addition is closed, associative, commutative, that a zero vector exists such that any vector plus the zero vector yields the same vector and finally, each vector has an additive inverse such that any vector plus its additive inverse results in the zero vector. The five laws for scalar multiplication are: multiplication by a scalar is closed, multiplication by a scalar is distributive across a sum of vectors, scalar multiplication is associative, multiplication by a scalar sum is distributive across the vector and one times any vector is that same vector. A basis set contains the minimum number of vectors that are required to span the space. A set of vectors spans a space if any vector in that space can be described as a linear combination of the vectors in the set. In a basis set, removing any one of the vectors will reduce the span of the set. A common example of a basis set is $\{\hat{i}, \hat{j}, \hat{k}\}$, the unit vectors in the x, y and z directions in \mathbb{R}^3 , the Cartesian coordinate system. Any position in three dimensional space can be described as a combination of these three vectors.

The basis set for a space is not necessarily finite in number and does not have to describe spatial position. A good example of these ideas is the set of trigonometric functions in the Fourier Series expansion. In Fourier Series, sine and cosine functions plus a constant term describe all possible ways in which a periodic function can oscillate. Sine functions describe odd periodic motion. Cosine functions describe even periodic motion. The constant takes into account the average value. Unlike \mathbb{R}^3 , in Fourier finitely many orthogonal vectors, where the

vectors are sine and cosine functions with different frequencies: A smooth function $f(t)$ can be represented as:

$$f(t) = a_0 + \sum_{m=1}^{\infty} a_m \cos(\omega_m t) + \sum_{m=1}^{\infty} b_m \sin(\omega_m t). \quad (0.1)$$

The coefficients describe the amplitude of each mode and the ω 's are the frequencies of the modes. The inner product in Fourier space of any two distinct basis functions is zero. Note that given a function f in (0.1) the appropriate inner product of (0.1) with the basis functions results in the solution for the coefficients a_0 , a_m and b_m .

An exact description of a non-sinusoidal function using Fourier Series requires infinitely many frequencies. By decreasing the number of frequencies used to represent the function one can get a sufficiently accurate approximation of an otherwise unsolvable problem. Fourier Series are the building blocks for constructing solutions to partial differential equations. For example, consider the partial differential equation (2.2) and its boundary conditions:

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2}, \quad (0.2)$$

$$u(0,t) = u(\pi,t) = 0 \text{ and} \quad (0.3)$$

$$u(x,0) = f(x,t).^{13} \quad (0.4)$$

The variable u is defined on the domain $0 \leq x \leq \pi$ and $t \geq 0$. Using the Galerkin method, the boundary conditions (0.3) will be accounted for if we choose a basis set which already satisfies the boundary conditions. The set $\{\sin(nt)\}$ is one such basis set. Therefore, the variable u can be approximated by the $\sin(\omega t)$ term from the Fourier Series expansion.

$$u(x, t) = \sum_{n=1}^{\infty} a_n(t) \sin(nx) \quad (0.5)$$

To illustrate how one computes $a_n(t)$, I will truncate n at three:

$$u(x, t) = a_1(t) \sin(x) + a_2(t) \sin(2x) + a_3(t) \sin(3x). \quad (0.6)$$

Inserting (0.6) into the partial differential equation yields

$$\frac{\partial a_1}{\partial t}(t) \sin(x) + \frac{\partial a_2}{\partial t}(t) \sin(2x) + \frac{\partial a_3}{\partial t}(t) \sin(3x) = -a_1(t) \sin(x) - 4a_2(t) \sin(2x) - 9a_3(t) \sin(3x). \quad (0.7)$$

The spatial dependence of (0.7) can be removed by taking the inner product of (0.7) with a fixed element of the basis set. Here the inner product of $f(t)$ with $\sin(nx)$ is defined by

$$\langle f(x) | \sin(nx) \rangle = \frac{2}{\pi} \int_0^{\pi} f(x) \sin(nx) dx. \quad (0.8)$$

Taking the inner product of (0.7) with each term in the spatial basis set produces the following system of three evolution equations:

$$\frac{\partial a_1}{\partial t} = -a_1(t), \quad (0.9)$$

$$\frac{\partial a_2}{\partial t} = -4a_2(t) \text{ and} \quad (0.10)$$

$$\frac{\partial a_3}{\partial t} = -9a_3(t). \quad (0.11)$$

Equations (0.9)-(0.11) are easily integrated. Normal mode analysis is analogous to the Galerkin method. A spatial basis set consisting of a term for fluid flow with circulation, a term for fluid flow without circulation and forcing at the boundaries is used. Instead of solving for the coefficients of the spatial basis set by taking the inner product with the time evolution equation,

the coefficients of the spatial basis set are computed by comparing the basis set with empirical data.

This project will be primarily based upon the Normal Mode method developed by Eremeev and expanded by Lipphardt in their respective analyses of surface currents in the Black Sea and Monterey Bay.^{14,15} The goal of this project is to compute the normal modes for regions as complex as the Chesapeake Bay using commercial and off-the-shelf software. To do this the following eigenvalue-eigenfunction problems (2.12), (2.13) and (2.14) have been solved in a space Ω , where Ω is \square^3 subject to the boundary conditions (2.15), (2.16) and (2.17).

$$\nabla^2 \Psi_n = -\lambda_n \Psi_n, \quad (0.12)$$

$$\nabla^2 \Phi_m = -\mu_m \Phi_m, \quad (0.13)$$

$$\nabla^2 \Theta(x, y, 0, t) = S_\Theta(t), \quad (0.14)$$

$$\Psi|_{boundary} = 0, \quad (0.15)$$

$$(\hat{n} \cdot \bar{\nabla} \Phi)|_{boundary} = 0, \quad (0.16)$$

$$(\hat{m} \cdot \bar{\nabla} \Theta)|_{boundary} = (\hat{m} \cdot \bar{u}_{model})|_{boundary}. \quad (0.17)$$

Similar to (0.5), the velocity field \bar{u} will be represented in terms of the eigenfunctions (0.12), (0.13) and the inhomogeneous solution to (0.14). NMA can also be equated to a multidimensional equivalent to the Fourier Series expansion described above. While Fourier series models the oscillations of one function, NMA models the possible movements in an entire two-dimensional region with the important feature that the shape of the boundary is accounted for through equations (0.15) and (0.16). In other words, solutions to (0.12) and (0.13) constitute the natural modes of vibration in the relatively complex domain Ω . In this formulation there are three modes: one that describes motion without any divergence (Ψ), one that describes motion

without any vorticity (Φ), and a third mode that accounts for net flow into the region due to tides at its openings (Θ). The purpose of this project is to use MATLAB® and its byproduct FEMLAB® to accomplish the task of computationally deriving the solutions to (0.12)-(0.17).

The three primary modes NMA uses are the Dirichlet mode, the Neumann mode and the inhomogeneous mode. The Neumann and Dirichlet modes can be derived from the representation for an incompressible, non-divergent three dimensional velocity field used by Eremeev. It is known that conservation of mass for an incompressible fluid reduces to the equation $\nabla \cdot \bar{u} = 0$. The key observation by Eremeev is that

$$\bar{u} = \bar{\nabla} \times [(\hat{n}\Psi) + \bar{\nabla} \times (\hat{n}\Phi)] \quad (0.18)$$

is the general solution to the equation $\nabla \cdot \bar{u} = 0$. In (0.18) \bar{u} is the three dimensional velocity field, Ψ is a scalar potential called the stream function, or vorticity mode, and Φ is a scalar potential called the divergence mode or velocity mode. The vector \hat{n} is the field of normals at the boundaries of the surface that is being studied. For the purposes of this project \hat{n} will be in the \hat{k} direction, a unit vector in the z-axis direction. The Chesapeake Bay will be considered to be flat and lying on the x and y plane. The boundary conditions (0.15) and (0.16) can be found by integrating (0.18) over a volume V_0 bounded by the surface S.

$$\int_{V_0} \bar{u} dV = \int_{V_0} (\bar{\nabla} \times \hat{n}\Psi) dV + \int_{V_0} \bar{\nabla} \times [\bar{\nabla} \times (\hat{n}\Phi)] dV \quad (0.19)$$

Eremeev solves this equation for a geometry bounded by two planar surfaces and the coastline and comes up with (0.15) and (0.16). Equation (0.16) is an additional term to Eremeev's formulation introduced by Lipphardt.¹⁷ Eremeev shows that (0.18) is equivalent to:

$$\hat{k} \cdot \bar{u} = -\nabla^2 \Phi \quad \text{and} \quad (0.20)$$

$$\hat{k} \cdot (\bar{\nabla} \times \bar{u}) = -\nabla^2 \Psi \quad (0.21)$$

by taking the dot product of (0.18) with \hat{k} and the curl of (0.18) dotted with \hat{k} . He also shows that the natural method for solving (0.20) and (0.21) is the eigenvalue-eigenfunction expansion. This leads to the examination of equations (0.12)-(0.17), that is, the general solutions for equations (0.20) and (0.21) are of the form:

$$\Psi = \sum_{n=1}^{\infty} A_n \Psi_n \quad (0.22)$$

$$\Phi = \sum_{m=1}^{\infty} B_m \Phi_m \quad (0.23)$$

Part of the usefulness of NMA is that, typically, a small number of modes are needed to approximate the solution to a boundary value problem because the geometry of the boundary is already embedded into the basis function. Consequently, equations (0.22) and (0.23) will only have to be calculated out to small finite values of n and m . Lipphardt found that 12 modes would satisfactorily describe the velocity field for the Monterey Bay.¹⁹

The actual surface velocities of each mode can be represented with respect to the Dirichlet and Neumann modes as follows:

$$(\mathbf{u}_n^D, \mathbf{v}_n^D) = \left(\frac{-\partial \Psi_n}{\partial y}, \frac{\partial \Psi_n}{\partial x} \right) \text{ and} \quad (0.24)$$

$$(\mathbf{u}_m^N, \mathbf{v}_m^N) = \left(\frac{\partial \Phi_m}{\partial x}, \frac{\partial \Phi_m}{\partial y} \right). \quad (0.25)$$

Dirichlet and Neumann modes enforce zero normal flow for both the coastline and intersections with other bodies of water.^{20,21} Eremeev did not consider open boundaries with other bodies of water since the Black Sea does not open into any major water sources.²² Consequently, the vorticity and velocity modes served as an accurate description of the Black Sea's surface currents. Because Monterey Bay is not a closed basin like the Black Sea,

Lipphardt took tidal forces and normal flow into account with an inhomogeneous mode for Monterey Bay's geometry.²³

Lipphardt's model was based on tidal flow and forcing through the open boundary. Consequently, Lipphardt solved for equation (0.14) with boundary condition (0.17).²⁴ In equations (0.14) and (0.17) \hat{m} is the unit outward normal vector at the boundary. The function $S_{\ominus}(t)$ is the net flow per unit area at the bay's boundary with the ocean,

$$S_{\ominus}(t) = \frac{\oint \hat{m} \bar{u}_{model}}{\iint dxdy}. \quad (0.26)$$

The vector \bar{u}_{model} is a known velocity field at the open boundary. The velocity field for this mode is simply the gradient of the inhomogeneous potential function.

$$(u^i, v^i) = \nabla \Theta(x, y, 0, t) \quad (0.27)$$

By combining the three boundary conditions with the three modes, Lipphardt derived a set of equations that would take into account all possible surface currents in the bay.²⁵

$$u(x, y, 0, t) = \sum_{n=1}^N A_n(0, t) u_n^D(x, y) + \sum_{m=1}^M B_m(0, t) u_m^N(x, y) + u^i(x, y, 0, t) \quad (0.28)$$

$$v(x, y, 0, t) = \sum_{n=1}^N A_n(0, t) v_n^D(x, y) + \sum_{m=1}^M B_m(0, t) v_m^N(x, y) + v^i(x, y, 0, t) \quad (0.29)$$

By numerically matching the coefficients of these functions with data received from HF radar Lipphardt was able to calculate the amplitudes of each normal mode.²⁶

The vector field that results from this computation is complete for the boundaries in which the modes have been calculated and can be used with the empirical data to fill in gaps in the velocity field. At this point both the initial conditions and the spatial dependence are known completely. The time dependent portion of this system is independent of the normal modes

themselves. This time dependent portion could be solved in a manner similar to the Galerkin method described above if a time evolution equation for the modes in the Chesapeake could be developed. Equations (0.28) and (0.29) are the final output of normal mode analysis for a body of water.

For geometries such as the Monterey Bay, Black Sea and the Chesapeake Bay, it is necessary to use numerical methods for solving equations (0.12), (0.13) and (0.14). Lipphardt used a FORTRAN library implementation of the Arnoldi method and a double-precision version of the generalized minimum residual method for sparse matrices included in the SPARSKIT FORTRAN library.²⁷ This project will deal with two numerical methods. The first method is based on the finite difference method and was the primary method used in this project. The second method is the finite element method which is the basis of FEMLAB, a MATLAB byproduct employed in this project for the purpose of comparing the two different numerical methods for geometries without analytical solutions.

Chapter 3

The Finite Difference Method

In this chapter the finite difference method and its applications are examined. First, the general finite difference scheme is derived. The execution of the finite difference method in MATLAB® is then explained. Both Neumann and Dirichlet boundary conditions are covered through the general finite difference approximation and a method for moving boundary points on the grid. Lastly, the discretization of the equations for the inhomogenous potential is formulated along with its implementation in MATLAB®.

The Finite Difference Method: In the finite difference method all the derivatives in equations (0.12), (0.13) and (0.14) are replaced by discrete approximations to the continuous equation. For example, since

$$\Psi_x(a, b) = \lim_{\Delta x \rightarrow 0} \frac{\Psi(a + \Delta x, b) - \Psi(a, b)}{\Delta x}, \quad (0.30)$$

$\Psi_x(a, b)$, the first derivative of the vorticity potential with respect to x , is replaced by

$$\frac{\Psi(a + \Delta x, b) - \Psi(a, b)}{\Delta x}. \quad (0.31)$$

Similarly, Ψ_{xx} and Ψ_{yy} then can be approximated by the so-called centered finite difference, that is,

$$\Psi_{xx}(a, b) = \frac{\Psi(a + \Delta x, b) - 2\Psi(a, b) + \Psi(a - \Delta x, b)}{(\Delta x)^2}, \quad (0.32)$$

$$\Psi_{yy}(a, b) = \frac{\Psi(a, b + \Delta y) - 2\Psi(a, b) + \Psi(a, b - \Delta y)}{(\Delta y)^2}, \quad (0.33)$$

which can be verified using Taylor series. By combining these two equations, an approximation for the Laplacian operator is produced.

$$\nabla^2 \Psi = -2\Psi(a, b) \left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} \right) + \frac{\Psi(a, b + \Delta y) + \Psi(a, b - \Delta y)}{(\Delta y)^2} + \frac{\Psi(a + \Delta x, b) + \Psi(a - \Delta x, b)}{(\Delta x)^2} \quad (0.34)$$

The domain Ω is then partitioned into a rectangular grid (x_i, y_j) and all derivatives in (0.12)-(0.14) are replaced at (x_i, y_j) by their corresponding finite difference terms. This process converts an equation like (0.12) to a large sparse system of algebraic equations which was solved numerically in MATLAB®.

The method applied in this paper uses a square grid where $\Delta x = \Delta y = h$. The approximation of the Laplacian then becomes,

$$\nabla^2 \Psi = \frac{1}{h^2} (-4\Psi(a, b) + \Psi(a, b + \Delta y) + \Psi(a, b - \Delta y) + \Psi(a + \Delta x, b) + \Psi(a - \Delta x, b)). \quad (0.35)$$

The eigenvalue-eigenfunction equations (0.12)-(0.14) can be approximated on a discrete square grid by replacing the Laplacian with the finite difference equation.

$$-\lambda \Psi = \frac{1}{h^2} (-4\Psi(a, b) + \Psi(a, b + \Delta y) + \Psi(a, b - \Delta y) + \Psi(a + \Delta x, b) + \Psi(a - \Delta x, b)). \quad (0.36)$$

The Ψ term is defined at all points on the grid. Applying this scheme creates a system of equations with each equation corresponding to the solution to one point on the grid. For example, if a rectangle were grided with two nodes in the x-direction and four nodes in the y-direction and then turned into a system of equations, it would produce this corresponding matrix.

$$\frac{-1}{h^2} \begin{bmatrix} -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \Psi_3 \\ \Psi_4 \\ \Psi_5 \\ \Psi_6 \\ \Psi_7 \\ \Psi_8 \end{bmatrix} = \lambda \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \Psi_3 \\ \Psi_4 \\ \Psi_5 \\ \Psi_6 \\ \Psi_7 \\ \Psi_8 \end{bmatrix}$$

No boundary conditions have been applied to the matrix in Figure 3.1. The matrix in Figure 3.1 used to discretely approximate the Laplacian operator is known as a differentiation matrix. It can

Figure 3.1: Differentiation matrix for 2nd order finite difference method.

easily be seen that this matrix is extremely sparse. MATLAB®’s “eig” and “eigs” functions can simultaneously solve this set of equations and produce the eigenvalues and eigenvectors. These two functions are not serial solvers. “Eig” and “eigs” require that the entire matrix be held in memory and input as one variable. Consequently, the size of the matrix becomes very important when thinking about system and program memory limits.

The size of the differentiation matrix increases at the rate of the number of nodes within the geometry squared. Consequently a square with ten nodes in the x-direction and ten nodes in the y-direction would have one hundred-nodes within its geometry and a differentiation matrix with ten-thousand elements. For a square with one-hundred nodes per side, there are ten-thousand total nodes making a solution matrix with one-hundred million elements. If each element in this differentiation matrix required eight bytes of memory, the total matrix would require eight-hundred megabytes of ram to be kept in memory. The large memory requirement is one major drawback of using a simultaneous solver. As the number of nodes used increases

the solution approaches the actual solution asymptotically causing an increase in the number of nodes to require vastly more memory for a diminishing increase in the accuracy of the solution. The memory problem can be solved by using MATLAB®'s sparse command. The sparse command removes all zeroes from the matrix and replaces the matrix with a vector of the value and position of each non-zero element in the matrix. MATLAB® allows the implementation of most normal matrix functions inherent in MATLAB® when using sparse matrices. For a sparse matrix the command "eigs" rather than "eig" must be used. Using sparse matrices, the number of elements in the differentiation matrix increases as five times the number of nodes within the geometry rather than the number of nodes squared.

Creating Geometries: The section above described how a particular group of nodes can be turned into a system of equations and solved to some approximation of the continuous solution, but it did not state how the relationships between the nodes are constructed. An orderly method for the storage and retrieval of the nodal relationships is necessary. The logical connections between nodes and the physical positions of elements in a matrix simplify the choice of a storage method. Using a square discretization where distances between nodes in the x and y directions are equal allows the position of numbers in a storage matrix to correspond exactly to their relationship in the finite differences equations. Figure 3.2 is an example of how the geometry of a triangle might be coded into a matrix. In the storage matrix ones correspond to nodes that are within the geometry and zeroes correspond to nodes which are outside of the geometry. The fact that this is a poor approximation to a triangle can be easily seen. Aliasing and the problems involved will be discussed later in this paper.

Every geometry discussed in this paper is created on a square matrix such as the one in Figure 3.2. The length of each side is set at one. For the purpose of finding exact solutions in test cases, the sides of the geometry are proportioned according to the length of the sides of the

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 3.2: Square Storage Matrix

storage matrix. Storing the geometry in a matrix also allows the creation of geometries using MATLAB®’s programming capabilities so that it becomes much easier to resize each shape for a different number of nodes.

A method for the construction of the differentiation matrix from a storage matrix such as the one above will now be examined. First a differentiation matrix which includes every node in the storage matrix is developed. This can be accomplished using the command `Toeplitz` in MATLAB®. Next the shape of the geometry must be transferred to the solution matrix. The way in which the storage matrix is searched has a significant effect on the meaning of the Ψ ’s in the matrix. For example, in the program used in this project the storage matrix is searched using a Raster method starting from the top left corner and continuing down the rows until the final row is reached.

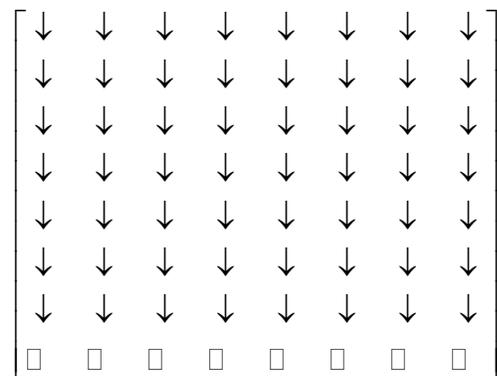


Figure 3.3: Raster Search Scheme

The search then moves to the top of the next column and continues on through the rest of the matrix. In the differentiation matrix one row refers to one node in the geometry. The position of the node corresponding to each consecutive row in the differentiation matrix has the same order as the search scheme. Therefore the

the first row in the differentiation matrix corresponds to the top left node in the geometry. As one travels down the rows of the differentiation matrix one also travels down the rows of the storage matrix until the final row in the storage matrix is reached. At this point one continues to

travel down the rows of the solution matrix but in the storage matrix the column is increased by one and the row value returns to one before continuing down the rows of the storage matrix. During this search the value at each node in the storage matrix is examined. If the value at the node is one, then nothing is done to the corresponding values in the differentiation matrix. If the value at the node is zero then the row and column of the solution matrix corresponding to that node are removed. By removing both the row and column all instances of the node are removed and the differentiation matrix begins to incorporate the shape of the geometry.

Expected Error Using the Finite Difference Method: There are two primary types of error involved in this project. The first type of error involves the finite differencing method itself. This can include aliasing effects in the approximation of non-rectangular shapes on a square grid as well as truncation error in the approximation itself. The second type of error is based upon the assumptions and approximations made by Eremeev and Lipphardt in their creation of the method.^{28,29} Both of these types of error provide bounds upon how accurate the normal mode solutions can become.

Truncation error in second order central differencing schemes has been thoroughly studied and can be found in most introductory texts on finite-difference methods. The truncation error for the central differencing approximation of the Laplacian can be found by doing a Taylor series expansion of the scalar function being studied for steps of $\pm\Delta x$ and $\pm\Delta y$,

$$\Psi(x + \Delta x) = \Psi(x, y) + \Delta x \Psi_x + \frac{1}{2} \Delta x^2 \Psi_{xx} + \frac{1}{6} \Delta x^3 \Psi_{xxx} + \frac{1}{24} \Delta x^4 \Psi_{xxxx} \quad (0.37)$$

$$\Psi(x - \Delta x) = \Psi(x, y) - \Delta x \Psi_x + \frac{1}{2} \Delta x^2 \Psi_{xx} - \frac{1}{6} \Delta x^3 \Psi_{xxx} + \frac{1}{24} \Delta x^4 \Psi_{xxxx} \quad (0.38)$$

with similar equations for steps of $\pm\Delta y$. When the four Taylor series expansions are added together all of the odd terms except for the first term cancel out. Since all computations in this paper are computed on a square grid it will now be assumed that $\Delta x = \Delta y = h$.

$$\Psi(x+h) + \Psi(x-h) + \Psi(y+h) + \Psi(y-h) = 4\Psi + h^2(\Psi_{xx} + \Psi_{yy}) + \frac{1}{12}h^4(\Psi_{xxxx} + \Psi_{yyyy}) \quad (0.39)$$

Observing (0.39), it can be seen that the second term on the right hand side is the Laplacian operator. Therefore that term can be replaced with the right hand side of equation (0.12).

Replacing the Laplacian operator and rearranging,

$$\lambda\Psi = \frac{1}{h^2}(-4\Psi + \Psi(x+h) + \Psi(x-h) + \Psi(y+h) + \Psi(y-h)) + \frac{1}{12}h^2(\Psi_{xxxx} + \Psi_{yyyy}) \dots \quad (0.40)$$

For the central differencing approximation the h^2 term and all the terms after it are truncated in the approximation. Therefore, the truncation error is $O(h^2)$ and should be on the order of,

$$\frac{1}{(n-1)^2} \quad (0.41)$$

where n is the number of nodes per side of the storage matrix and $h = 1/(n-1)$. When n is very large, $1/(n-1) \approx 1/n$. This error is inherent in the system and cannot be overcome without changing the method itself.

The use of a simple square grid to discretize the domain introduces error into the boundaries of the geometries being studied. This error is called aliasing and is directly correlated to the method in which points on the geometry are sampled. It is impossible using a square grid to approximate exactly the shape of an equilateral triangle, a circle, or for that matter any non-rectangular shape. There is zero probability of nodes on the outer edges of the geometry corresponding exactly to the boundary of the geometry itself. This is simply the limitations of a

square grid and can only be decreased by sampling with an increasing number of nodes. Consequently, for any non-rectangular geometry the boundary will be an approximation of the true shape with maximum deviation from exact boundary of $2\sqrt{h}$.

The second type of error concerns the physical assumptions made by Eremeev and Lipphardt.^{30,31} The first source of error is the use of only two dimensions in the study of the modes. Because of this two-dimensional approach, areas of the Bay where the water is shallow will be given greater weight in the computation than is necessary. Water touching a stationary surface is assumed to have zero velocity tangential to the surface due to friction. The viscosity of the water then decreases the tangential velocity of the water farther from the surface in the direction normal to the surface. Therefore, the shallower the area of the Bay the more the bottom friction will affect velocity of the surface currents. Another possible area of error is the constantly changing boundary of the Bay due to tides. However, this variation is extremely small when compared to a geometry which is approximately 200 miles long. Since only the lowest modes of this 200 mile long geometry are being studied, there is very little probability that the wavelengths of any of the modes in question will be significantly affected by this variation. Thus the primary focus of this project was the numerical calculations of the normal modes and their corresponding error rather than the implementation of the modes to actual data.

Dirichlet Boundary Conditions: The Dirichlet boundary condition $\Psi = 0$ applies to the streamline potential in equation (0.12), $\nabla^2\Psi_n = -\lambda_n\Psi_n$. For a discrete geometry the $\Psi = 0$ condition must be satisfied at every boundary node. The method used to make the boundary values zero is the same as that required to create the geometry itself. The rows and columns in the differentiation matrix corresponding to the nodes on the boundary of the geometry are removed. Therefore, the boundary nodes do not actually appear in the final differentiation

matrix for a geometry with Dirichlet boundary conditions. The removal of the rows and columns from the differentiation matrix imposes the boundary condition on each boundary node by completely removing that node from the possible four other equations which would use it in their finite difference scheme. In the finite difference equations for the surrounding nodes removing the node from all equations has the same effect as setting $\Psi = 0$.

Ames Method for Dirichlet Boundary Conditions: The two primary contributors of error to the finite difference method have already been stated as truncation error and aliasing at the boundaries. In order to help differentiate between these two types of error a method for moving boundary nodes was implemented. This method was taken from Numerical Methods for Partial Differential Equations by Ames.³² The only place where Ames's method differs from the normal finite differencing method is at the boundaries. At the boundary proportionality constants are used to effectively move the boundary nodes to the position of the actual boundary. This movement requires exact knowledge of the boundary and therefore is a very useful method for testing how much error in a solution is a result of the Taylor series approximation and how much is a result of aliasing at the boundaries. Ames's method assumes that the actual boundary falls somewhere in between the boundary nodes and the first set of interior nodes in the geometry.

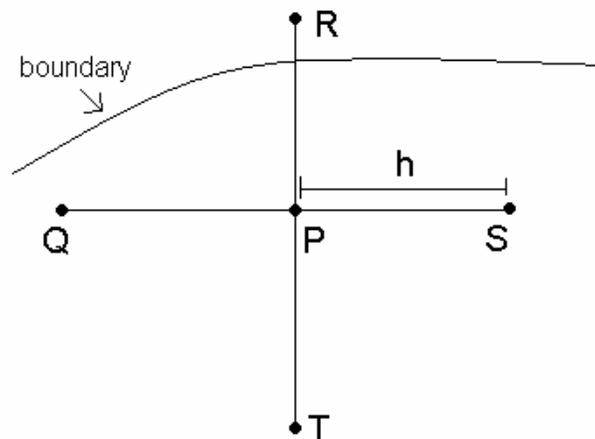


Figure 3.4: Pictured is a five point computational molecule on a square grid. The boundary falls in between the boundary node R and the interior node P.

Figure 3.4 is a five point computational molecule, the five nodes used in the Taylor Series approximation to the Laplacian operator, set on a square grid with distance h between nodes. There is a small error in the position of boundary node R. By multiplying the distance h between boundary node R and the central node P by a scaling factor, node R can be shifted down far enough on the y-axis of the molecule to coincide exactly with the boundary of the geometry in question. In Figure 3.5 the variable n is a scaling factor for the line PR.

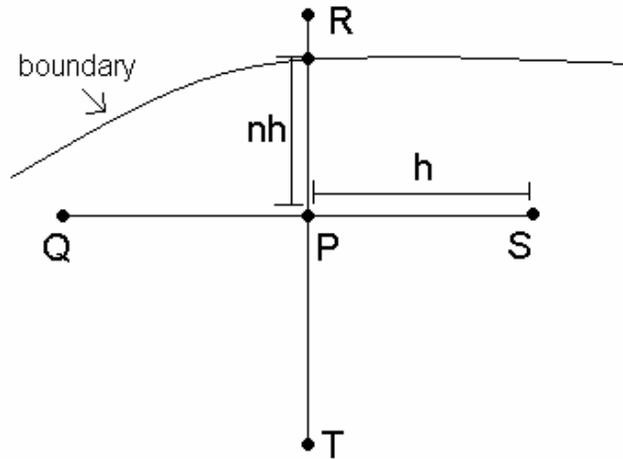


Figure 3.5: Five point computational molecule with boundary node R shifted in the y-direction by a scaling factor of n .

In order to calculate the difference equations with the scale factor for Ames's method a slightly different approach to using the Taylor series expansion was used. Rather than immediately approximating the derivative, the second order Taylor series expansion was applied to approximate the value of the potential at each of the four nodes around the central node in the five point computational molecule,

$$\Psi(x, y) = \Psi_p + x(\Psi_x)_p + y(\Psi_y)_p + \frac{x^2}{2}(\Psi_{xx})_p + xy(\Psi_{xy})_p + \frac{y^2}{2}(\Psi_{yy})_p \quad (0.42)$$

$$\Psi(0, nh) = \Psi_p + nh(\Psi_y)_p + \frac{(nh)^2}{2}(\Psi_{yy})_p + \dots \quad (0.43)$$

$$\Psi(0, -h) = \Psi_P - h(\Psi_y)_P + \frac{h^2}{2}(\Psi_{yy})_P + \dots \quad (0.44)$$

$$\Psi(h, 0) = \Psi_P + h(\Psi_x)_P + \frac{h^2}{2}(\Psi_{xx})_P + \dots \quad (0.45)$$

$$\Psi(-h, 0) = \Psi_P - h(\Psi_x)_P + \frac{h^2}{2}(\Psi_{xx})_P + \dots \quad (0.46)$$

Solving the four linear equations, (0.43)-(0.46), in matrix form produces two equations, one for the second derivative of Ψ with respect to y and one for the second derivative of Ψ with respect to x.

$$(\Psi_{yy})_P = \frac{2}{h^2} \left[\frac{1}{n(n+1)} \Psi_R + \frac{1}{(n+1)} \Psi_T - \frac{1}{n} \Psi_P \right] + O(h^2) \quad \text{and} \quad (0.47)$$

$$(\Psi_{xx})_P = \frac{1}{h^2} [\Psi_Q + \Psi_S - 2\Psi_P] + O(h^2). \quad (0.48)$$

Equation (0.48) is the exact formulation used in the finite difference method for the second order approximation of the second derivative. This formulation makes sense considering no nodes in the molecule have been shifted in the x-direction. Equation (0.48) has shifted from equation (0.47) by a set of scaling factors proportional to n. These factors approximate the downward shift of boundary node R by a factor n to the exact boundary of the geometry. In the equation for the second derivative with respect to y the value for Ψ_R is set to zero since it is a boundary node for Dirichlet boundary conditions. Adding these two equations together and replacing the second derivatives with the right hand side of equation (0.12) gives the discrete approximation to the streamline potential at node P,

$$-\lambda\Psi_P = \frac{1}{h^2} \left[\Psi_Q + \Psi_S + \frac{2}{(n+1)} \Psi_T - 2\left(\frac{n+1}{n}\right) \Psi_P \right] + O(h^2) \quad (0.49)$$

Different formulations of equations x and y must be produced to account for all combinations of shifts in all four cardinal directions. There are fourteen possible combinations. The derivation of these equations follows exactly from the derivation for the shift in one boundary node in the negative y -direction derived previously.

One important note is that the actual boundary node does not appear in the equation since its value is zero. The only effect the boundary node has on the finite difference equations is to change the scaling factors in front of the interior nodes. The removal of the boundary node in the differentiation matrix becomes very important when more than one interior node uses a boundary node in its finite difference equation. If a node is moved in the x -direction to touch the exact boundary then it no longer rests on the vertical line connecting it to other nodes on the square grid. Because of the shift in the x -direction, another interior node cannot use the shifted boundary node as a term in its difference equation as part of the approximation of the y -derivative. The boundary node no longer resides on the y -axis of the computational molecule for that interior node. For example, in Figure 3.6 if node Q is moved to point V then node U will no longer be able to use node Q in its finite difference equation without cross terms.

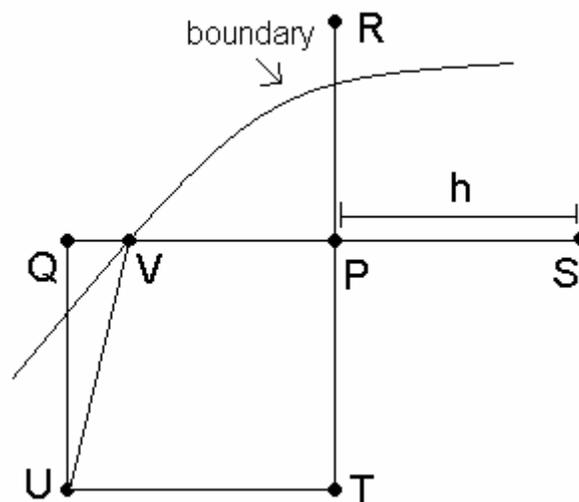


Figure 3.6: If node Q is moved to point V it node longer resides on the y -axis of node U .

Since the shift in the boundary node only appears in scaling factors in front of the interior nodes it can be moved in both the x-direction and the y-direction without adverse affects to the positions of the nodes.

The actual application of Ames's method requires two primary deviations from the normal finite difference calculation. The first deviation is the calculation of scaling factors in both the x-direction and the y-direction for every point on the boundary. In this project two matrices were used for the storage of the scaling factors. One matrix held all of the scaling factors in the x-direction and the other matrix held all of the scaling factors in the y-direction. The scaling factors were obtained by computing the distance between the boundary nodes and the actual boundary in both the x-direction and the y-direction. Once the distance between the two boundaries was calculated, the distance was then divided by the distance between two nodes on the square grid. This produces a scaling factor between zero and one. Finally, in the creation of the differentiation matrix the computer must decide whether the boundary node is on the x-axis or y-axis in order to choose the correct scaling factor for the difference equation.

The second primary difference between the implementation of Ames's method and the normal finite difference method is the scaling factor. Because the boundary nodes are zero and therefore do not appear in the equation the rows and columns of each of the boundary nodes are still removed. However, based on the combination of boundary nodes in each molecule one of fourteen different finite difference equations will be used along with a specific scaling factor for each boundary node.

Neumann Boundary Conditions: The next mode for the normal mode analysis method is the mode defined by Neumann boundary condition. This method requires a more complicated approximation for the boundary conditions than the Dirichlet and presents two significant

problems. The first problem is that approximating a derivative on the boundary of the geometry requires points outside of the geometry itself. The normal mode formulation gives no information about what happens just outside the geometry. Consequently, points outside of the geometry cannot be used in the formulation. The second problem is that the normal derivatives of non-rectangular geometries very rarely correspond with nodes on the square grid.

A method to remove the dependence on nodes outside of the geometry and to approximate the normal of a non-rectangular boundary has been developed. The geometry is still created by removing rows and columns. If the normal derivative is in either the x or y direction, then a constraint for the normal derivative can easily be found. This constraint uses a central differencing approximation for the normal derivative.

$$\left. \frac{\partial \Phi}{\partial x} \right|_{\text{boundary}} \approx \frac{\Phi(x+h) - \Phi(x-h)}{2h} = 0 \quad (0.50)$$

Rearranging equation (0.50) yields,

$$\Phi(x+h) = \Phi(x-h) \quad (0.51)$$

The approximation in (0.51) both removes the necessity of using a point outside of the boundary and sets the normal derivative equal to zero. The new equation for a node on the boundary with the normal in the x direction can be found by replacing the node outside of the geometry with the node that has the same value but is inside the geometry,

$$\lambda \Phi = \frac{1}{h^2} (-4\Phi + 2\Phi(x+h) + \Phi(y+h) + \Phi(y-h)). \quad (0.52)$$

Equation (0.52) does not cover the corners of the square. The exact solution to the square geometry with Neumann boundary conditions provides the motivation for the approximation of the corners. The exact solution to the Neumann square is the following function of cosines,

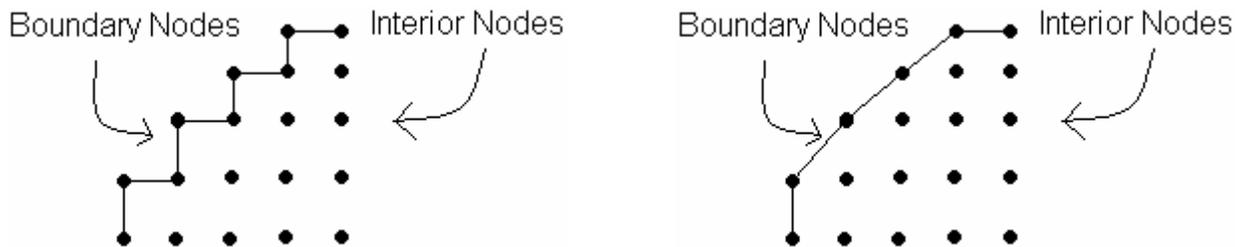
$$\Phi(x, y) = \cos\left(\frac{n\pi x}{L_x}\right) \cos\left(\frac{m\pi y}{L_y}\right). \quad (0.53)$$

It is equation (0.53) from which the motivation for the finite difference approximation to the corners is taken. Trying to determine the normal to a piecewise point in a curve, such as a corner, is not an easy task. The derivative at each of the corners on the square provides the necessary approximation. At each corner in the square the derivatives in both the x and y directions are equal to zero. At a corner both points outside of the geometry can be approximated with equation (0.52). The finite difference approximation at the corner is:

$$\lambda\Phi = \frac{1}{h^2}(-4\Phi + 2\Phi(x+h) + 2\Phi(y+h)). \quad (0.54)$$

On a square grid every curve and line is approximated by many corners. Consequently, with the corners and the vertical and horizontal boundaries covered this method can be expanded to solve for the circle, the equilateral triangle or the Chesapeake Bay.

For the Neumann method there are two ways to look at the boundary. The first way likens the boundary to a set of stairs. In this formulation each boundary node is connected to the adjacent boundary nodes by one of the points in its corresponding five point molecule.



Figures 3.7-3.8: Two different ways of viewing the boundary for a square grid.

Figure 3.7 is the stair step way of looking at the boundary. For the Neumann boundary conditions this has both concave and convex corners corresponding to twelve different finite difference approximations for the boundary condition derived in the same manner as equation (0.54). With the stair step all of the boundary nodes on concave corners have no nodes in their

computational molecule that are outside of the geometry. They have two on the boundary and two on the interior. The diagonally connected boundary was used instead of the stair step boundary for the computation of Neumann modes. With the diagonally connected boundary there are only eight finite difference approximations for the boundary condition and each boundary node has at least one node in its computational molecule outside of the geometry.

The calculation of Neumann modes applies some variation of equations similar to (0.54) to the boundary nodes. All of the interior nodes use the standard finite difference approximation for the Laplacian. Because of the derivative boundary condition Neumann modes include the boundary points in their differentiation matrix. This means that a Neumann mode will have a larger differentiation matrix that must be solved for the same grid size as the Dirichlet equivalent. The application of the derivative boundary conditions also increases the degree of asymmetry in the differentiation matrix. Asymmetric matrices tend to have some imaginary eigenvalues and are also more difficult to compute than symmetric matrices.

Ames's Method for Neumann Boundary Conditions: The method for moving boundary points described in the section on Dirichlet boundary conditions can also be applied to Neumann boundary conditions. Ames's method for Neumann boundary conditions is exactly the same as that for Dirichlet with respect to the shift of boundary nodes to the exact boundary of the geometry. To correctly apply the boundary conditions the normal derivative must also be included in the calculations and consequently the normal to the boundary must also be included. Figure 3.9 shows the five point computational molecule with two boundary nodes. The scaling factors are n and m . The angles of the normal to the boundary at R and S with respect to the x -axis are ϕ and θ . These two normals can be incorporated into the directional derivative to

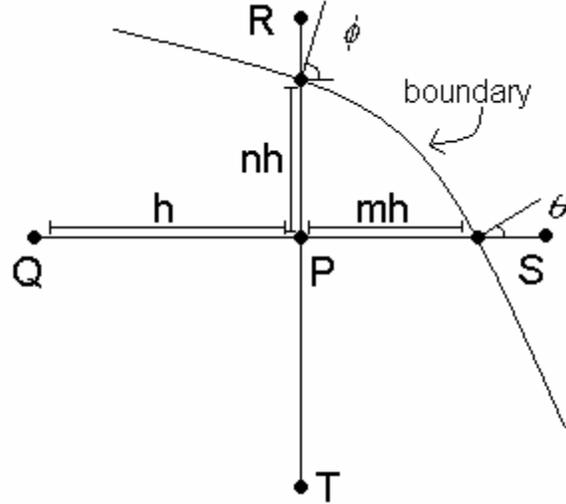


Figure 3.9: Ames method for Neumann boundary conditions

approximate the normal derivative at both boundary nodes R and S. The linearized normal derivative, also called the directional derivative is:

$$\left. \frac{\partial \Phi}{\partial n} \right|_{\phi} = \frac{\partial \Phi}{\partial x} \cos(\phi) + \frac{\partial \Phi}{\partial y} \sin(\phi). \quad (0.55)$$

In this equation $\left. \frac{\partial \Phi}{\partial n} \right|_{\phi}$ is evaluated at ϕ , the angle between the normal to the boundary and the x-axis at boundary node R. In order to derive a Taylor series approximation for the directional derivative, the derivative of the Taylor series evaluated at point R must be taken with respect to both x and y.

$$\frac{\partial \Phi(x, y)}{\partial x} = (\Phi_x)_P + x(\Phi_{xx})_P + y(\Phi_{xy})_P + \dots \quad (0.56)$$

$$\frac{\partial \Phi(x, y)}{\partial y} = (\Phi_y)_P + x(\Phi_{xy})_P + y(\Phi_{yy})_P + \dots \quad (0.57)$$

Equations (0.56) and (0.57) are derivatives of the two-dimensional Taylor series expansion truncated after the derivative of the third term. Inserting these two equations into the equation for the directional derivative provides a discrete approximation for the directional derivative.

$$\begin{aligned} \left. \frac{\partial \Phi(x, y)}{\partial n} \right|_{\phi} = & \left[(\Phi_x)_p + x(\Phi_{xx})_p + y(\Phi_{xy})_p \right] \cos(\phi) + \dots \\ & \left[(\Phi_y)_p + x(\Phi_{xy})_p + y(\Phi_{yy})_p \right] \sin(\phi) \end{aligned} \quad (0.58)$$

Evaluating equation (0.58) at the scaled boundary node R, $\Phi(0, nh)$, yields:

$$\left. \frac{\partial \Phi(0, nh)}{\partial n} \right|_{\phi} = \left[(\Phi_x)_p + y(\Phi_{xy})_p \right] \cos(\phi) + \left[(\Phi_y)_p + y(\Phi_{yy})_p \right] \sin(\phi) \quad (0.59)$$

Using this technique at boundary node S produces another equation similar to (0.59). The velocity potential at the interior nodes is the normal finite difference approximation. For the normal finite difference approximation, there are four equations but five unknowns. Because the normal derivative of the potential at the boundary node has been used instead of the value of the potential at the boundary, a cross term has come into the equations for both the normal derivatives at boundary nodes R and S. Consequently, a fifth equation must be developed so that the derivatives can be solved.

The easiest way to produce a fifth equation is to take a point on the boundary close to the central node of the computational molecule and find the finite difference approximation for the normal derivative of that node at the boundary. Adding an extra node on the boundary increases the information that the differentiation matrix has concerning the boundary and normal derivatives. In this case it adds a cross term to the calculations. More boundary nodes could be added to go out to farther orders. However, the corresponding linear equations to be solved and

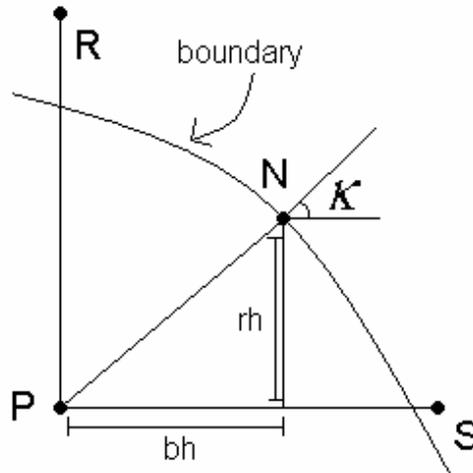


Figure 3.10: Node N has been inserted into the boundary.

the resulting solutions become exponentially more complicated. In Figure 3.10 a new node N has been inserted into the boundary. Its position in the computational molecule is (bh, rh) where b and r are scaling factors. The angle κ is the angle between the x-axis and the normal to the boundary at node N. In order to find a discrete equation for the normal derivative at node N equation (0.58) must be evaluated at (bh, rh) ,

$$\left. \frac{\partial \Phi(bh, rh)}{\partial n} \right|_{\phi} = \left[(\Phi_x)_p + bh(\Phi_{xx})_p + rh(\Phi_{xy})_p \right] \cos(\phi) + \dots \quad (0.60)$$

$$\left[(\Phi_y)_p + bh(\Phi_{xy})_p + rh(\Phi_{yy})_p \right] \sin(\phi)$$

All of the terms in the approximation to the discrete directional derivative are now included since node N lies on a diagonal in between the x-axis and the y-axis. Combining equation (0.60) with the two equations for the interior nodes and the two equations for the boundary nodes yields five equations to solve for the five unknowns (3.32)-(3.36). Just as in Ames's method for Dirichlet boundary conditions the value of the boundary condition is zero so that the $\frac{\partial \Phi}{\partial n}$ term is removed from each of the boundary node equations leaving only terms for the interior nodes. Therefore,

the normal derivative terms for nodes R, S and N will not actually be input into the differentiation matrix and will only effect the final equations through coefficients. The five finite difference equations to be solved are:

$$\left[(\Phi_x)_p + rh(\Phi_{xy})_p \right] \cos(\phi) + \left[(\Phi_y)_p + rh(\Phi_{yy})_p \right] \sin(\phi) = 0 \quad (0.61)$$

$$\left[(\Phi_x)_p + bh(\Phi_{xx})_p \right] \cos(\phi) + \left[(\Phi_y)_p + bh(\Phi_{xy})_p \right] \sin(\phi) = 0 \quad (0.62)$$

$$\left[(\Phi_x)_p + bh(\Phi_{xx})_p + rh(\Phi_{xy})_p \right] \cos(\phi) + \left[(\Phi_y)_p + bh(\Phi_{xy})_p + rh(\Phi_{yy})_p \right] \sin(\phi) = 0 \quad (0.63)$$

$$\Phi_p - h(\Phi_y)_p + \frac{h^2}{2}(\Phi_{yy})_p = \Phi(0, -h) \quad (0.64)$$

$$\Phi_p - h(\Phi_x)_p + \frac{h^2}{2}(\Phi_{xx})_p = \Phi(-h, 0) \quad (0.65)$$

The solution to these five linear equations for Φ_{xx} and Φ_{yy} contains too many terms to be efficiently solved and programmed. Rather, the calculation of their solution is shifted to the implementation of the MATLAB® program itself.

The implementation of Ames's method for Neumann boundary conditions in MATLAB® builds upon the implementation of Ames's method for Dirichlet boundary conditions. The shift for the two boundary nodes R and S remains exactly the same. The changes required to complete the computation are: the addition of node N, calculation of the normal angles at each boundary point and the computation of Φ_{xx} and Φ_{yy} using terms from the known information about the normal angles and distance between the nodes.

The creation of node N and the calculation of the normal angles produce seven new storage matrices for the computation of Ames's method, assuming there are two boundary nodes in the molecule. The first three matrices come from the creation of node N. Node N is created

by choosing a point on the boundary that lies within that specific computational molecule. The method of choosing the position of N is dependent upon the specific geometry that is being studied and can be any point on the boundary of the geometry within the computational molecule. A point that is on the boundary within the molecule but as far as possible from the other boundary nodes provides the most information about the boundary. Once the position of N has been computed two matrices must be created one for the x-position of node N and one for the y-position. The x-position and y-position of the node are stated with respect to the central node in the computational molecule and in this project were represented as scaling factors with respect to h. The third calculation for node N uses knowledge of the boundary to find the normal angle. These three matrices provide all the information that is required about node N.

The final four matrices contain the normal angles for the other two boundary nodes in the computational molecule. Before these can be calculated the shift in the x-direction and the y-direction to the boundary at each node must be found. Once the shifts are known, then the normal angles when the node is shifted in both the x-direction and the y-direction can be calculated. The shift of the node combined with the normal angle corresponding to that shift gives all required information about the boundary nodes R and S.

When all the normal angles and nodal shifts are known, all of the coefficients of the derivatives in equations (0.61) through (0.65) are also known. By combining like terms and moving all non-derivative term to the right hand side of the equation, a matrix of coefficients is created:

$$\begin{bmatrix}
\cos(\phi) & \sin(\phi) & nh \cos(\phi) & 0 & nh \sin(\phi) \\
\cos(\theta) & \sin(\theta) & mh \sin(\theta) & mh \cos(\theta) & 0 \\
\cos(\kappa) & \sin(\kappa) & h[r \cos(\kappa) + b \sin(\kappa)] & bh \cos(\kappa) & rh \sin(\kappa) \\
0 & -h & 0 & 0 & h^2/2 \\
-h & 0 & 0 & h^2/2 & 0
\end{bmatrix}
\begin{bmatrix}
(\Phi_x)_P \\
(\Phi_y)_P \\
(\Phi_{xy})_P \\
(\Phi_{xx})_P \\
(\Phi_{yy})_P
\end{bmatrix}
=
\begin{bmatrix}
0 \\
0 \\
0 \\
(\Phi)_T - (\Phi)_P \\
(\Phi)_Q - (\Phi)_P
\end{bmatrix}$$

(0.66)

The five by five matrix on the left side of equation (0.66) is composed of coefficients to the derivatives of Φ which are known but unique to each computational molecule containing a boundary node. By multiplying the left hand side of both sides of equation (0.66) by the inverse of the coefficient matrix a solution for the derivatives of Φ can be found. Once each coefficient has been evaluated for a specific computational molecule, MATLAB® can also solve this equation by inverting the five by five matrix providing a numerical solution for Φ_{xx} and Φ_{yy} . MATLAB® can efficiently compute the inverse of a five by five matrix. While the inversion does exact a computational toll it is more efficient than attempting to symbolically solve and program the solution to the equation above.

Ames's Method for Neumann boundary conditions requires significantly more work than it does for Dirichlet. An extra node must be fabricated, normal angles for each boundary node must be computed and the numerical solution for Φ_{xx} and Φ_{yy} must be calculated. At the same time all of the interior nodes are approximated by the exact same finite differencing method that was used in each of the previous examples.

The Inhomogenous Mode: In the Galerkin method for spectral approximation, each term in the basis set must individually comply with the boundary conditions of the domain in question. The first two modes, vorticity and velocity, both have zero normal flow at their boundaries. However, in order to incorporate normal flow through the boundaries of the domain a term must

be used which allows the expansion to satisfy all the behavior of the system at the boundary. The inhomogenous mode provides for this necessity. It uses model data or experimental data to take into account normal flow through boundaries with other bodies of water. The equations for the inhomogenous mode are:

$$\nabla^2 \Theta(x, y, 0, t) = S_{\Theta}(t), \quad (0.67)$$

$$(\hat{m} \cdot \bar{\nabla} \Theta)|_{boundary} = (\hat{m} \cdot \bar{u}_{model})|_{boundary} \quad \text{and} \quad (0.68)$$

$$S_{\Theta}(t) = \frac{\oint \hat{m} \bar{u}_{model}}{\iint dxdy}. \quad (0.69)$$

The Θ term is the inhomogenous potential while the $S_{\Theta}(t)$ term is the line integral on the boundary of the geometry of the normal flow through the boundary divided by the area of the geometry. The term \hat{m} is the normal to the boundary and \bar{u}_{model} is the surface current at the boundary taken from an outside model or observations.

The first point to note about the inhomogenous mode is that it is not an eigenvalue equation, rather it is a Poisson equation where the Laplacian is set equal to a constant term. . Because the equation for the inhomogenous mode is a Poisson's equation it does not have any eigenvalues associated with it and consequently has only one particular solution. The constant term in the Poisson's equation, $S_{\Theta}(t)$, is only constant with respect to the spatial domain. The variable $S_{\Theta}(t)$ is continuously changing with respect to time taking into account the changing normal flow at the boundaries coming from the empirical data or model. The changes in $S_{\Theta}(t)$ with respect to time require that for each time slice a new mode must be calculated or a method for guessing future forcing terms must be in place. In contrast it is only the coefficients of the vorticity and velocity modes which change leaving the basis set itself as a constant.

The discrete approximation of the inhomogenous mode uses the same general finite differencing method as the Neumann mode since both have derivative boundary conditions. The Laplacian is approximated using the Taylor series expansion and truncating after the second order terms,

$$\Delta\Theta = \frac{1}{h^2} \left[\Theta(x+h) + \Theta(x-h) + \Theta(y+h) + \Theta(y-h) - 4\Theta \right] \quad (0.70)$$

The approximation to the normal derivative for the inhomogenous mode is slightly different from that used for the Neumann mode since it is equal to the normal velocity at the boundary.

$$\left. \frac{\partial\Phi}{\partial n} \right|_{boundary} = \frac{\Phi(x+h) - \Phi(x-h)}{2h} = (\hat{m} \cdot \bar{u}_{model})|_{boundary} \cdot \quad (0.71)$$

Rearranging equation (0.71) produces,

$$\Phi(x+h) = \Phi(x-h) + 2h(\hat{m} \cdot \bar{u}_{model})|_{boundary} \cdot \quad (0.72)$$

Inserting equation (0.72) into the discretization of the Poisson's equation yields,

$$\frac{1}{h^2} \left(-4\Phi + 2\Phi(x+h) + \Phi(y+h) + \Phi(y-h) \right) = S_{\Theta}(t) - \frac{2}{h} (\hat{m} \cdot \bar{u}_{model})|_{boundary} \cdot \quad (0.73)$$

Equation (0.73) is the discretization of the inhomogenous mode including boundary conditions. Just as for the Neumann mode there are a number of different possibilities and combinations of derivatives depending upon the number and arrangement of boundary nodes in the computational molecule. No matter what approximation for the boundary condition is used the right hand side of equation (0.73) is only dependent upon the empirical data or model that is being used.

The implementation in MATLAB® of the formulation for the inhomogenous potential requires the computation of a number of different variables: a matrix containing the normal velocity of the data or model at boundary nodes in the geometry corresponding to open sources with other bodies of water, the area of the geometry and the line integral of the normal flow

through the boundary. The procedure for putting the model or data normal flow through a boundary into a form that can be used with the finite differencing method thus far described is completely dependent upon the form in which the data or model data is presented. In this project velocity data from Quoddy, a finite element model of the Chesapeake Bay which uses the Navier Stokes equations and some empirical data to model three dimensional fluid flow, was used to get the normal flow at the boundary. The surface of the Quoddy data has 9700 nodes. At each node there is a value for both the velocity in the x-direction, \bar{u} , and velocity in the y-direction, \bar{v} . In MATLAB® the “griddata” command can be used to fit a three dimensional surface to data with irregular grid points and interpolate it to an evenly spaced grid. “Griddata” was used to take the model velocity from Quoddy and put it into matrix form corresponding to the number of nodes that were used on the Chesapeake Bay. From there the boundary nodes corresponding to the open boundaries were calculated. Using this information and the matrix created with "griddata" the x-velocity and the y-velocity data at the correct boundaries nodes was extracted. Next the normal angle was used to find the velocity normal to each of the boundary points at open sources.

The first calculation in the creation of the $S_{\circ}(t)$ term is the line integral of the normal velocity along the boundary of the geometry. The integral takes a very simple form. At each boundary node the normal velocity from the previously calculated values was multiplied by the distance between adjacent nodes, h , and summed over the entire boundary,

$$\oint \hat{m} \cdot \bar{u}_{model} \approx \sum_{c=1}^C h (\hat{m} \cdot \bar{u}_{model})_c \quad (0.74)$$

The variable c is the index of each boundary node and C is the total number of boundary nodes. Next the area of the geometry was calculated. Each of the interior nodes for a square grid is comprised of a box that is length h on each side. Therefore the area per node is h^2 . On the

boundary there are two other possibilities. The first possibility is that the boundary is a straight boundary and the area per node is $1/2h^2$. The second possibility is that the boundary node is a corner and the area per node is $1/4h^2$. Summing over all of the nodes in the geometry using the corresponding areas for boundary nodes provides the total area of the geometry. Dividing the line integral in equation (0.74) by the total area provides the source term $S_{\ominus}(t)$.

In the actual calculation of the \ominus potential the source term $S_{\ominus}(t)$ becomes a vector that is the same length as the differentiation matrix, its number of elements is equal to the total number of nodes in the geometry. Since $S_{\ominus}(t)$ is a scalar each element in its vector is exactly same. However, once the boundary conditions have been applied to the discrete approximation of the Laplacian, the value of the vector becomes: $S_{\ominus}(t) - 2/h(\hat{m} \cdot \bar{u}_{model})|_{boundary}$. The variations in this vector are constrained elements corresponding to boundary nodes on the geometry which have a non-zero normal derivative.

From here the differentiation matrix is applied in the exact same manner as for the Neumann mode. This differentiation matrix multiplied times a vector of the potentials at each node such as in Figure 3.1 is set equal to the vector, $S_{\ominus}(t) - 2/h(\hat{m} \cdot \bar{u}_{model})|_{boundary}$. The discrete equation is: $Ax = b$ where A is the differentiation matrix, x is the vector of potentials and b is the source term vector. The solution for the potential vector can be found by multiplying each side of the equation by the inverse of the differentiation matrix: $A^{-1}Ax = A^{-1}b$. Therefore, the solution for the inhomogenous mode is: $x = A^{-1}b$. Solving for the inhomogenous mode at each time step in this manner allows the sources in the Chesapeake Bay to be taken into account.

Chapter 4

Testing and Analysis of the Dirichlet Finite Difference Scheme

The Dirichlet boundary conditions were solved in two steps: first the algorithm was tested on three different shapes: the square, the circle and the equilateral triangle then the accuracy of the test geometries were compared to theory, the eigenvectors were tested for orthogonality and convergence of the eigenvalues was tested. The data from this analysis was then compared to the solution from both Ames's method and FEMLAB®. Once the analysis was complete the algorithm was applied to the Chesapeake Bay and error estimates were made about the accuracy of the solutions using convergence and comparison with the solution from FEMLAB®. In order to test the finite differencing method, it is necessary to apply it to geometries where the solutions are well known. This type of analysis will be done for each of the modes. In the first section, three shapes are used to test the accuracy of the method for Dirichlet boundary conditions, where the value is zero at the boundaries. Four error estimates were used for the test geometries: convergence of the first eigenvalue, accuracy of the first hundred eigenvalues with respect to theory, the orthogonality of the eigenvectors and

comparisons between the eigenvalues of the finite differencing method and the eigenvalues retrieved from FEMLAB®.

The Square: The first geometry to which the finite differencing method was applied was the square. Because a square can be approximated exactly on a square grid, the convergence of the solution in this geometry should have been the closest to the ideal convergence, the truncation error, as is possible. The solution to (0.12), $\nabla^2\Psi_n = -\lambda_n\Psi_n$, for the square can be found exactly through a simple derivation. First, separation of variables is done by separating Ψ into its x and y components, $\Psi(x,y) = \Gamma(x)\Pi(y)$, giving

$$\frac{1}{\Gamma(x)} \frac{\partial^2\Gamma(x)}{\partial x^2} + \frac{1}{\Pi(y)} \frac{\partial^2\Pi(y)}{\partial y^2} = -\lambda \quad (0.75)$$

In equation (0.75) λ is not a function of either x or y. Since λ is constant and both terms vary independently of each other, the terms on the left side of equation (0.75) must be equal to a negative constant.

$$\frac{1}{\Gamma(x)} \frac{\partial^2\Gamma(x)}{\partial x^2} = -g^2 \quad (0.76)$$

$$\frac{1}{\Pi(y)} \frac{\partial^2\Pi(y)}{\partial y^2} = -f^2 \quad (0.77)$$

Here f and g are arbitrary constants. Both (0.76) and (0.77) are oscillatory equations. Therefore a good guess of their solutions would be a sine function.

$$\Gamma(x) = \sin(gx) \quad (0.78)$$

$$\Pi(y) = \sin(fy) \quad (0.79)$$

Equation (0.78) must equal zero at $x = 0$ and at $x = 1$. Equation (0.79) must be equal to zero at $y = 0$ and at $y = 1$. Application of these boundary conditions sets the constants f and g equal to:

$$f = \frac{n\pi x}{L_x} \quad (0.80)$$

$$g = \frac{m\pi y}{L_y} \quad (0.81)$$

The complete solution is a combination of the solutions to the two one-dimensional equations (0.78) and (0.79). The eigenfunction is then,

$$\Psi(x, y) = \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right). \quad (0.82)$$

In these equations L_x and L_y are the lengths of each side of the rectangle and are equal to one in the geometry being investigated. The variables n and m can be any integer from one to infinity. The eigenvalues of the equation can be found by substituting Ψ into equation (0.12). The resulting eigenvalues are:

$$\lambda = \pi^2 \left(\frac{n^2}{L_x^2} + \frac{m^2}{L_y^2} \right). \quad (0.83)$$

The second-order finite differencing method was used to solve the unit square geometry. The first square had ten nodes on each side. The number of nodes per side was increased by ten until there were one-hundred nodes per side. Next the square was discretized in increments of one-hundred nodes per side until the square geometry had five-hundred nodes per side. A five-hundred by five-hundred node square corresponded to a solution matrix with sixty-two billion five-hundred million elements. The first two eigenmodes and vector fields for the square geometry have been plotted in figures 5-8. The solutions were exactly as expected. The first mode had one large gyre. As the mode increased, the size of the gyres decreased and the number of gyres increased. Modes with axes of symmetry such as mode two had a multiplicity of states. For example, mode three for the square is just mode two shifted ninety degrees.

Square Dirichlet Potential Mode 1 (300X300)

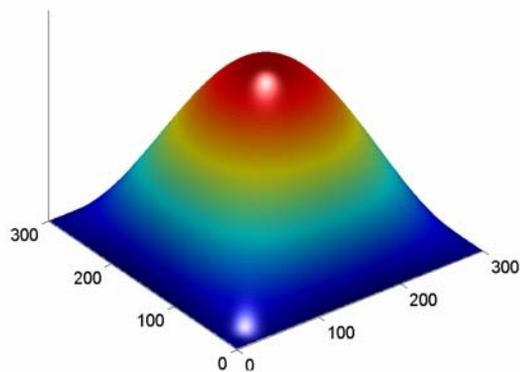


Figure 4.1

Square Dirichlet Potential Mode 2 (300X300)

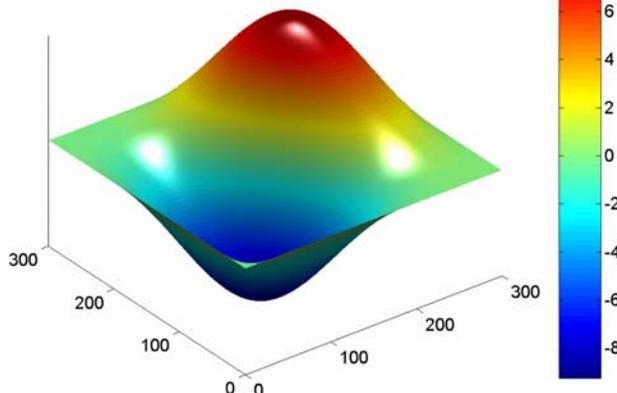
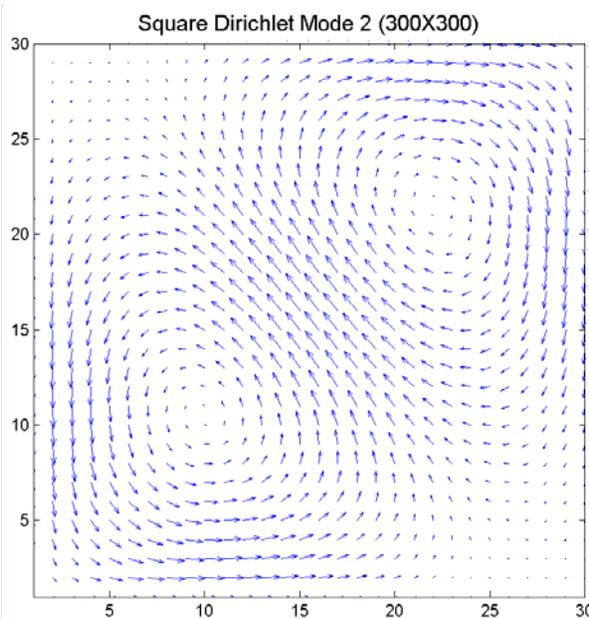
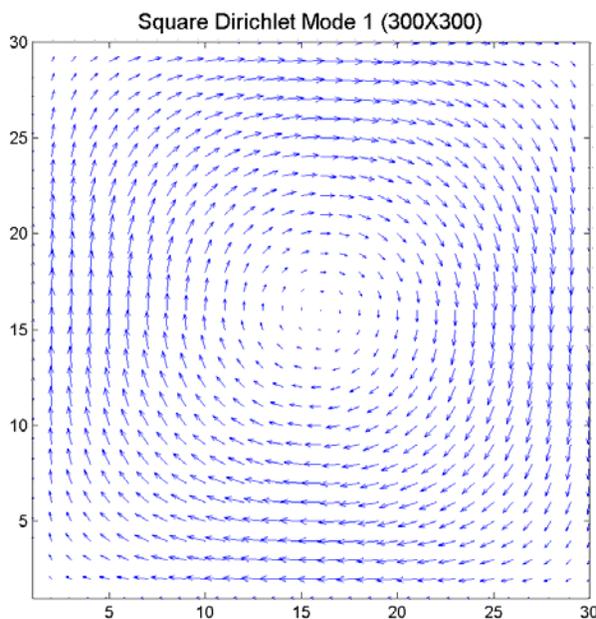


Figure 4.2



Figures 4.3-4.4: Both vector fields were constructed from the respective potentials in figures 5-6. A central differencing approximation of (0.24) was used. Only 1:10 of the rows and columns from the actual graph have been sampled

The Circle: The finite differencing method for Dirichlet boundary conditions was tested on the Circle. The exact solution to the circle follows much the same method as the solution to the square geometry. The primary difference is that polar coordinates are used. Applying separation of variables in polar coordinates to equation (0.12) with $\Psi(r, \theta) = R(r)\Theta(\theta)$ produces,

$$\frac{1}{R} \left(r^2 \frac{\partial^2 R}{\partial r^2} + 2r \frac{\partial R}{\partial r} \right) + r^2 \lambda = \frac{1}{\Theta} \frac{\partial^2 \Theta}{\partial \theta^2} \quad (0.84)$$

Since the value of each side of (0.84) is independent of the other side of the equation, both sides must be equal to a constant. Therefore,

$$\left(r^2 \frac{\partial^2 R}{\partial r^2} + 2r \frac{\partial R}{\partial r} \right) + R(r^2 \lambda - n^2) = 0 \quad (0.85)$$

$$\frac{\partial^2 \Theta}{\partial \theta^2} - n^2 \Theta = 0 \quad (0.86)$$

Separation of variables shows that the eigenvalues depend solely on the radial variable. The angular variable merely requires that the function itself be continuous in the $\hat{\theta}$ direction. Equation (0.85) is Bessel's equation whose solutions are Bessel functions of the first and second kind, $J(\lambda r)$ and $Y(\lambda r)$ respectively. Bessel functions of the second kind can be ignored since they approach $-\infty$ at $r=0$ which is not a physically acceptable solution for the potential. The eigenvalues of the circular geometry are the zeros of the Bessel function, $J(\lambda r)$, where $J(\lambda r)=0$ at $r=0.5$. Values for Bessel functions and their zeros were calculated by Abramowitz et al. [1964].³⁴

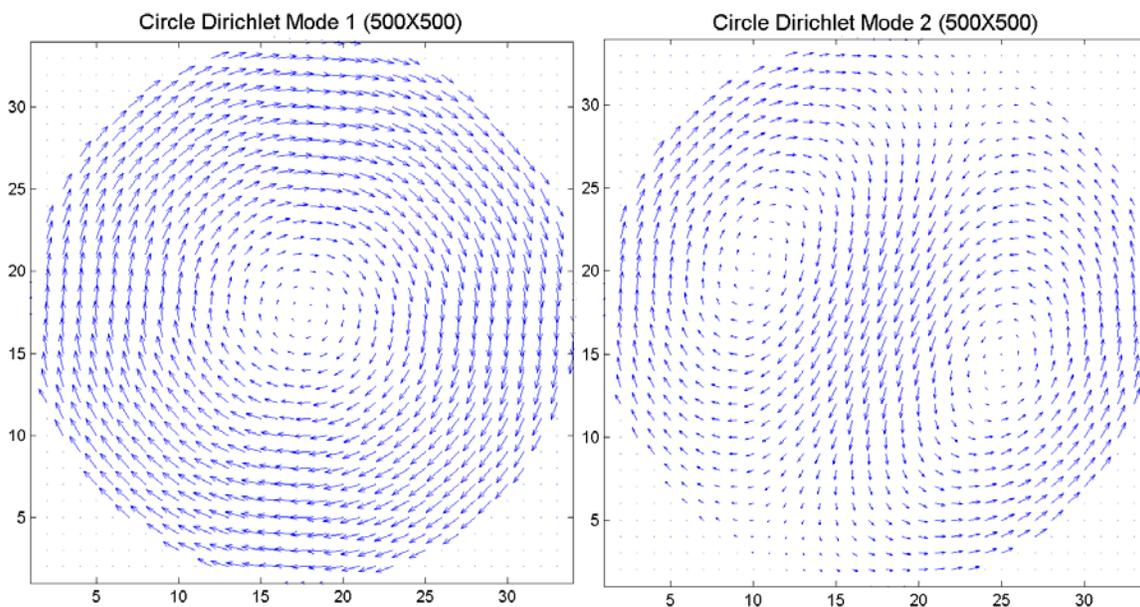
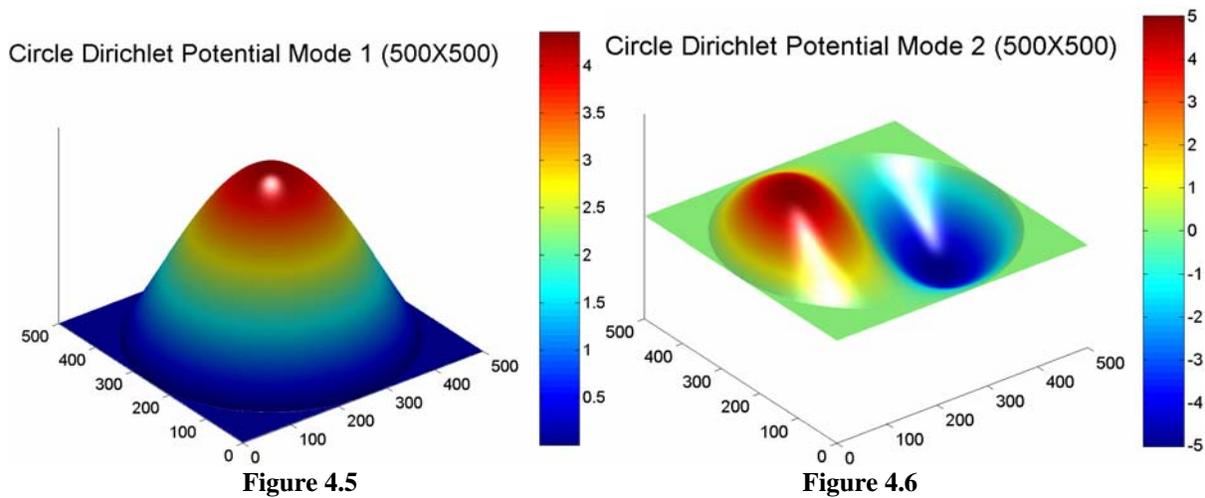
$$\Theta(\theta) = \cos(n\theta) \quad (0.87)$$

The complete solution for the circle with Dirichlet boundary conditions is,

$$\Psi = \cos(n\theta) J(\lambda r). \quad (0.88)$$

The test sizes of the finite differencing method for the circle were done in the same manner as the square geometry. The increase in nodes was based upon the number of nodes on the side of the unit square on which the circle was inscribed. One extra run was done for six-

hundred nodes on each side of the unit-square. The number of nodes internal to the geometry in each of the test runs were: 60, 276, 648, 1184, 1876, 2724, 3720, 4872, 6180, 7668, 31064, 70168, 124980, 195496 and 281760. Figures 10-13 show the potentials and vector fields for the first two eigenmodes of the circle. A total of one hundred eigenmodes were solved for each of the test runs. Because the circle was inscribed on a unit square the radius of the circle was $1/2$



Figures 4.7-4.8: Vectors fields created from their respective potentials and (0.24). Approximately 1:14 rows and columns from the actual graph have been sampled.

and the diameter was 1. This decrease in size from the unit circle caused the wavelength of the potential to decrease. A decrease in wavelength corresponds to an increase in the eigenvalues so that the eigenvalues of the circle ranged from 23.13 for the first mode to 1731.7 for mode one-hundred.

The Equilateral Triangle: The last geometry that was examined for the Dirichlet mode was the equilateral triangle. The exact derivation of the solution to the equilateral triangle was published just recently by [McCartin 2003].³³ The eigenfunction for the equilateral triangle is,

$$\begin{aligned} \Psi_s^{m,n} = & \sin\left[\frac{\pi l}{3r}(u+2r)\right] \cos\left[\frac{\pi(m-n)}{9r}\right](v-w) + \sin\left[\frac{\pi m}{3r}(u+2r)\right] \cos\left[\frac{\pi(n-l)}{9r}\right](v-w) \\ & + \sin\left[\frac{\pi n}{3r}(u+2r)\right] \cos\left[\frac{\pi(l-m)}{9r}\right](v-w) \end{aligned} \quad (0.89)$$

The origin is at the center of the triangle where

$$u = \frac{1}{2\sqrt{3}} - y, \quad (0.90)$$

$$v = \frac{\sqrt{3}}{2} \left(x - \frac{1}{2}\right) + \frac{1}{2} \left(y - \frac{1}{2\sqrt{3}}\right) \text{ and} \quad (0.91)$$

$$w = \frac{\sqrt{3}}{2} \left(\frac{1}{2} - x\right) + \frac{1}{2} \left(y - \frac{1}{2\sqrt{3}}\right). \quad (0.92)$$

The exact eigenvalues of the equilateral triangle are,

$$\lambda = \frac{8\pi^2}{9} [l^2 + m^2 + n^2] \text{ and} \quad (0.93)$$

$$l + m + n = 0. \quad (0.94)$$

For the finite differences solution, the equilateral triangle was oriented on the storage matrix by setting one side of the triangle against the top of the storage matrix. The top of the triangle had a height of $\sqrt{3}/2$ leaving a significant portion of empty space in the storage matrix.

Equilateral Triangle Dirichlet Potential Mode 1 (700X700)

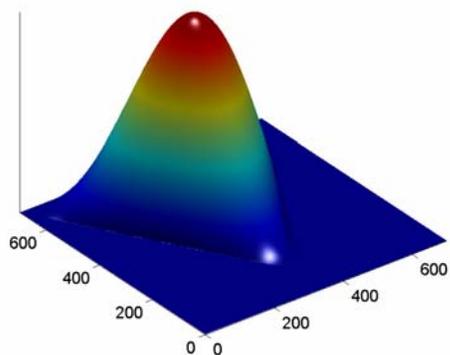


Figure 4.8

Equilateral Triangle Dirichlet Potential Mode 2 (700X700)

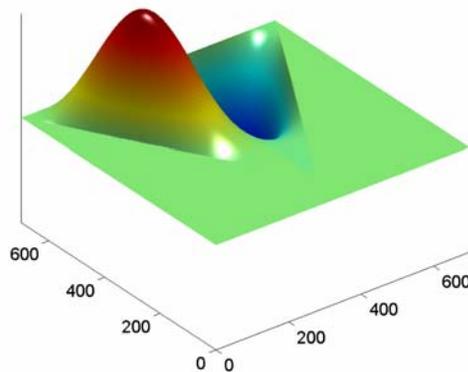
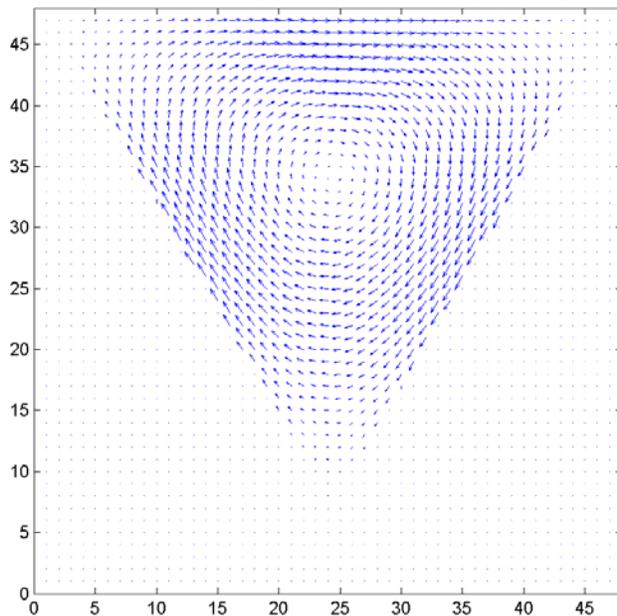
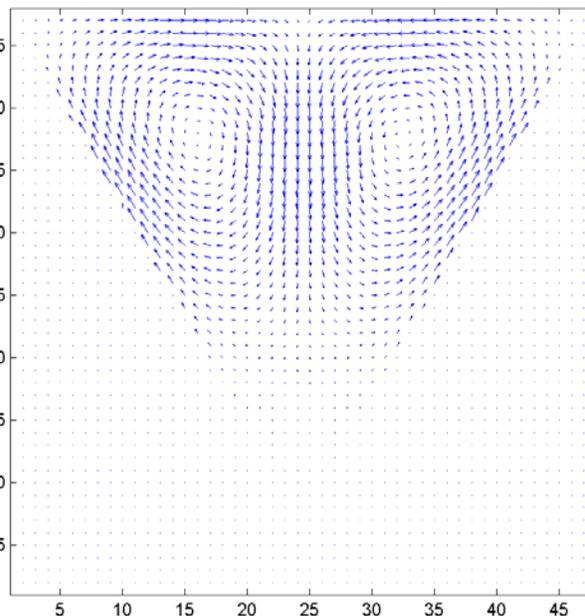


Figure 4.9

Equilateral Triangle Dirichlet Mode 1 (700X700)



Equilateral Triangle Dirichlet Mode 2 (700X700)



Figures 4.10-4.11: Vectors fields created from their respective potentials and (0.24). Approximately 1:15 rows and columns from the actual graph have been sampled.

The equilateral triangle geometry was tested in the same way as the square and circle except that one extra run was completed on a matrix that was 700 by 700 nodes. The number of nodes internal to the geometry for each test run was 144, 346, 633, 1008, 1470, 2017, 2652, 3374, 4181, 5076, 17022, 38522, 68683, 107504 and 154984. The modes for the triangle showed similar gyres and degeneracy as both the square and the circle for the first few modes and then diverged as the eigenvalues increased. The eigenvalues for the equilateral triangle ranged from 52.6 for the lowest eigenvalue to 3314.8 for mode one-hundred.

Analysis of Dirichlet Test Geometries: After the eigenvalues and eigenvectors of the test shapes had been computed their solutions were evaluated in four ways; for convergence against theoretical values, for their accuracy over all one-hundred eigenvalues computed against theoretical values, for their orthogonality and by comparison with the solutions found using FEMLAB®. The first analysis completed was the convergence analysis for the first eigenvalue in each of the geometries. The relative error of the first eigenvalue for an increasing number of nodes was computed against the theoretical value. The relative error in the first eigenvalue was then plotted versus the number of internal nodes in that geometry for each run. A fit was then done to the points on the plot to see the rate of convergence and compare it to the theoretical prediction of the truncation error. For the circle and equilateral triangle convergence plots were also done for the same number of nodes but using Ames's method. The increased rate

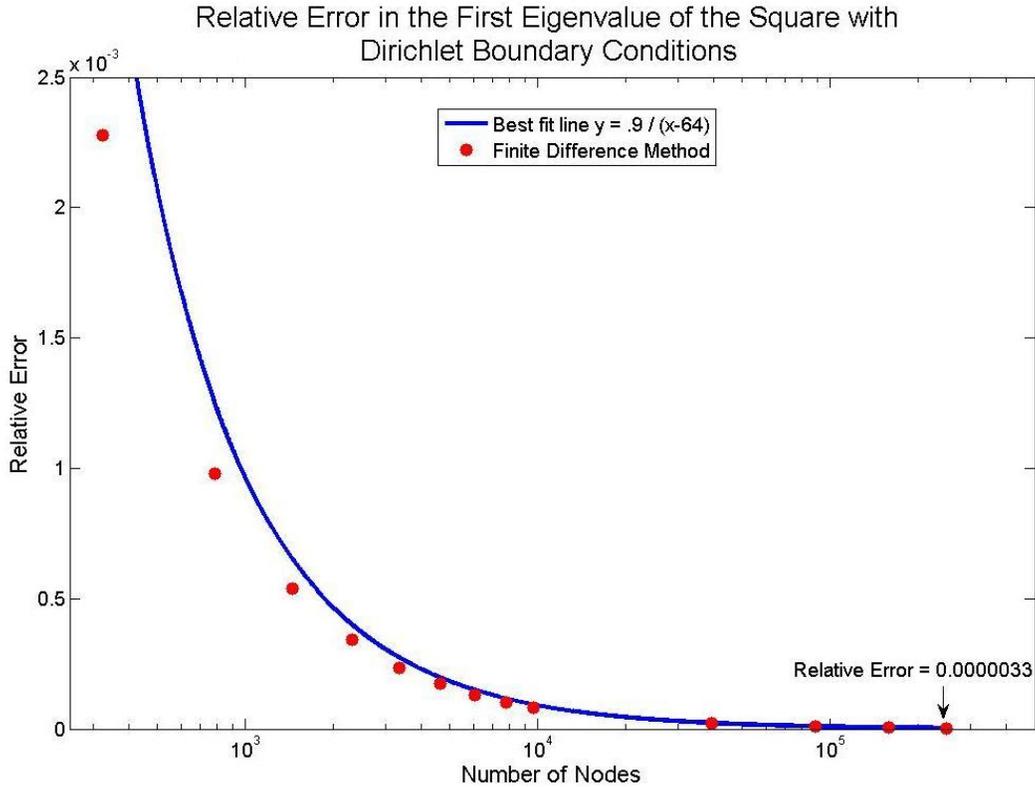


Figure 4.12: Graph of relative error in the finite difference approximation versus theory.

of convergence for these plots was used to substantiate the amount of error input by aliasing at the boundaries of the non-rectangular geometries. The convergence of the first eigenvalue for the square has been plotted in Figure 4.12. The convergence of the first eigenvalue for the square corresponded almost exactly to the $1/(n-1)$ truncation error predicted by the Taylor series error analysis. The only slight deviation was for the first two runs which were more accurate than predicted by the previous analysis.

The convergence for the first eigenvalue of the circular geometry provided a few more interesting properties. The convergence for the circle has been plotted in Figure 4.13. The first and most noticeable deviation from the square were the oscillations in the convergence of the eigenvalue. For example, the accuracy of the eigenvalue decreased as the nodes increased from ninety per side to one-hundred. These oscillations come from aliasing effects in the creation

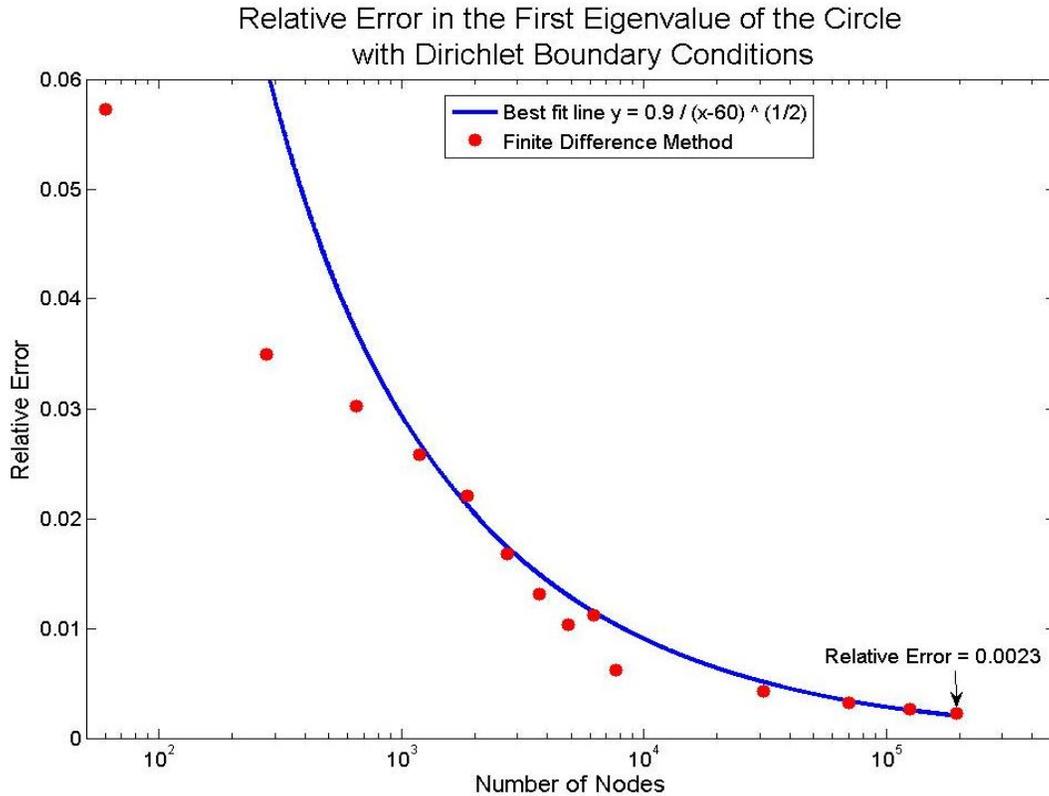


Figure 4.13: Graph of relative error in the finite difference approximation versus theory.

of the geometry, the wavelength of the eigenvalue resonates with certain specific orientations of nodes after which it gradually falls away. Since the accuracy of the approximation of the boundary can vary between zero and $1/\sqrt{2} h$ it is possible for the approximation of a geometry to actually become worse as the number of nodes is increased. However, the oscillations are linearly dependent on the distance between nodes and therefore must approach zero as the distance between the nodes approaches zero. The second feature of the convergence plot is that the rate of convergence had decreased drastically for the first eigenvalue from $1/n$ to $1/\sqrt{n}$. The main difference between the square geometry and the circular geometry was the error built into the boundary of the circle. Consequently, the decrease in the rate of convergence must

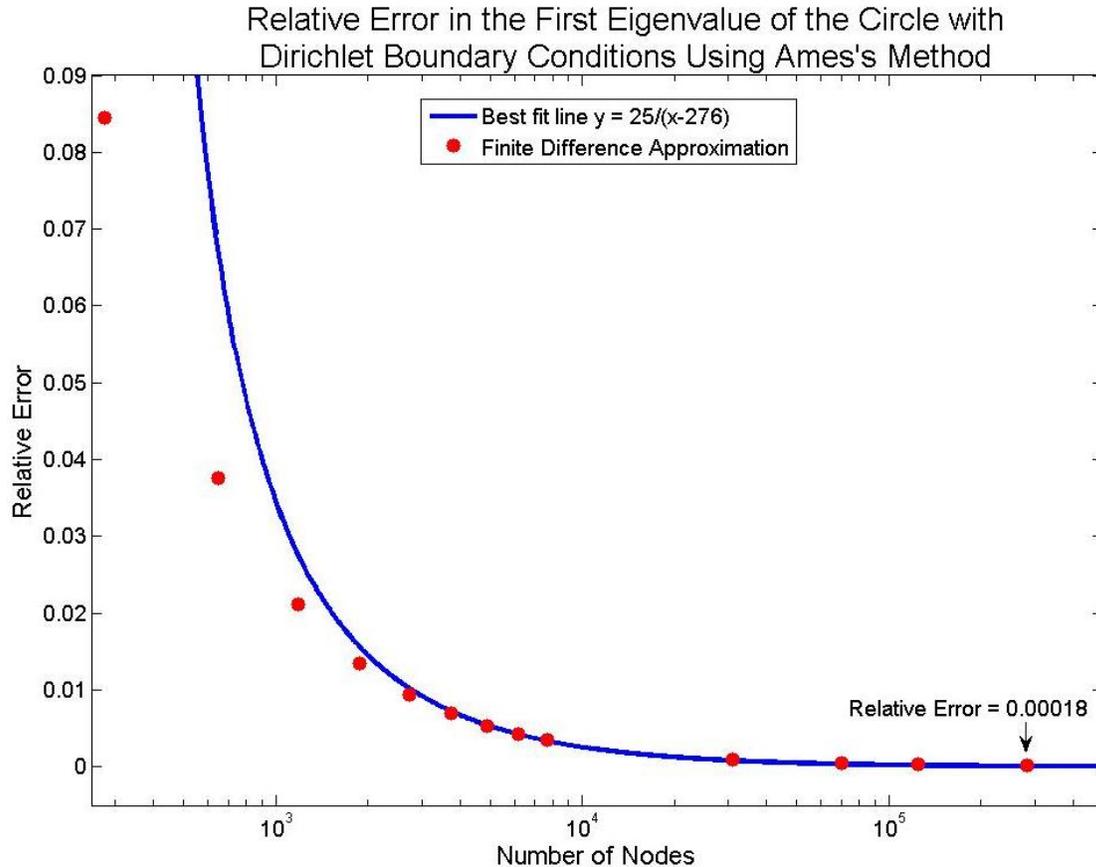


Figure 4.14: Relative Error in the first eigenvalue of the circle versus theory.

have been primarily caused by errors in the boundary. These errors were expected and could possibly be negated by moving the nodes at the boundary onto the true boundary for the circular geometry.

In Figure 4.14 the convergence for Ames's method was plotted in the exact same manner as the normal finite differencing method. As can be seen the accuracy of the first eigenvalue of the solution to the circle using Ames's method has a convergence rate of $1/(n-1)$. The increase in convergence rate shows that much of the difference in error between the square and circle came from errors at the boundary of the domain. It should also be noted that the oscillations in the first eigenvalue have disappeared proving that the oscillations for the normal finite differencing method were due to changes in the placement of boundary nodes.

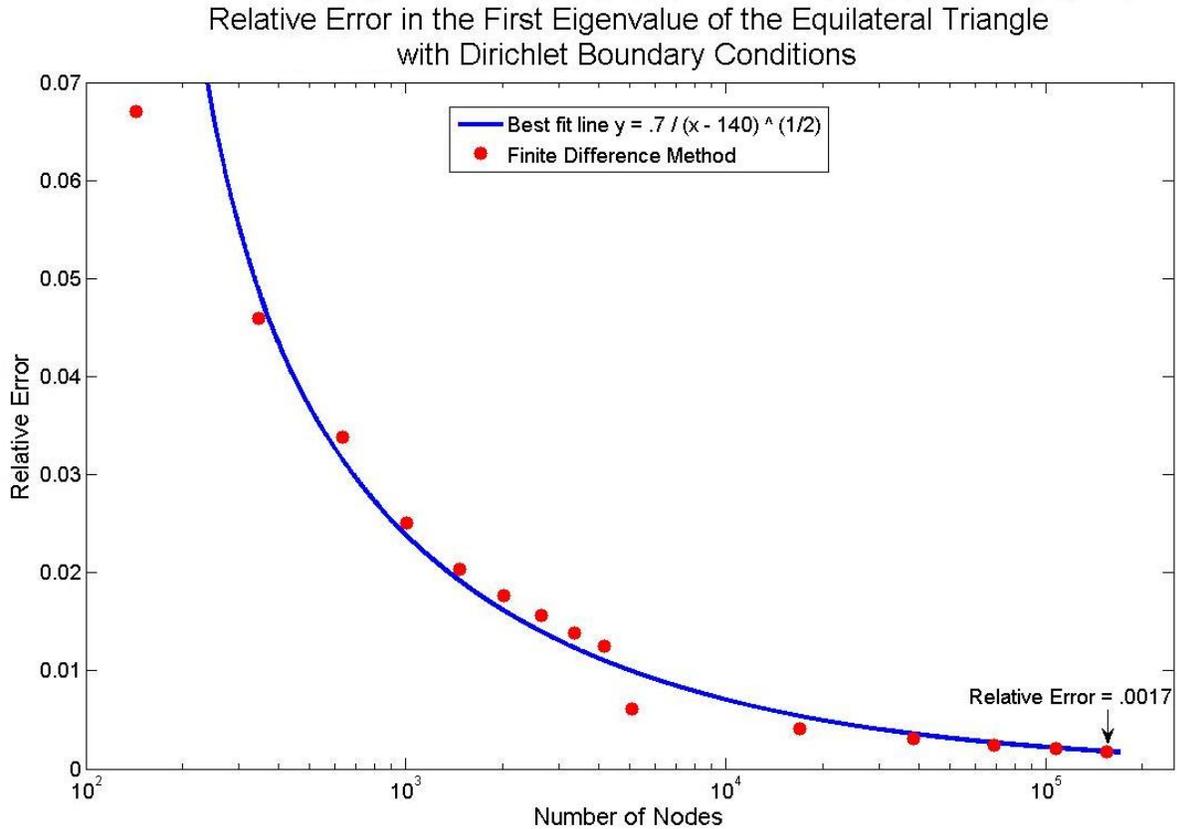


Figure 4.15: Graph of relative error in the finite difference approximation versus theory.

In Figure 4.15 the convergence of the first eigenvalue of the equilateral triangle has been plotted. The convergence of the first eigenvalue for the equilateral triangle shows features similar to the circle. First there is still some slight oscillation in the accuracy of the first eigenvalue. The final accuracy of the eigenvalue for the equilateral triangle and the convergence were both of the same order as the normal finite difference method for the circle. There has also been a significant decrease in the rate of convergence on the same order as the circle.

In Figure 4.16 the convergence of the first eigenvalue of the equilateral triangle using Ames's method has been plotted. The rate of convergence of the equilateral triangle was nearly right on the convergence of the square. However, the final relative error of the first eigenvalue

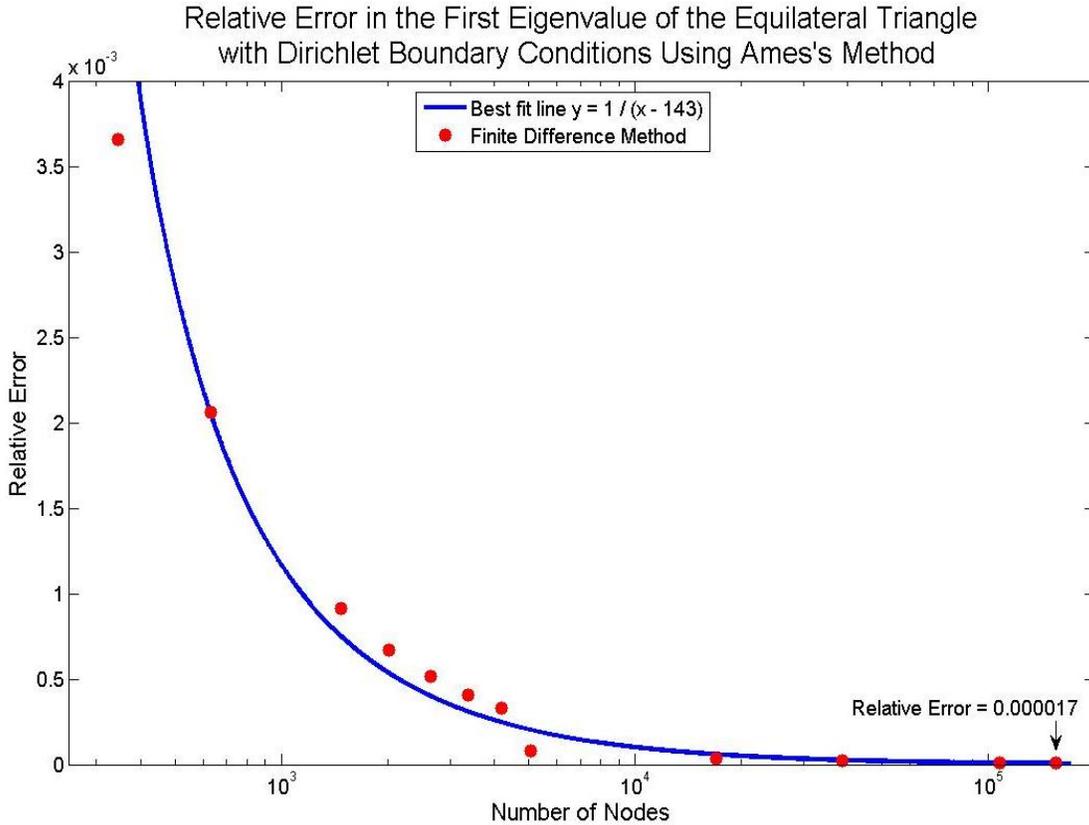
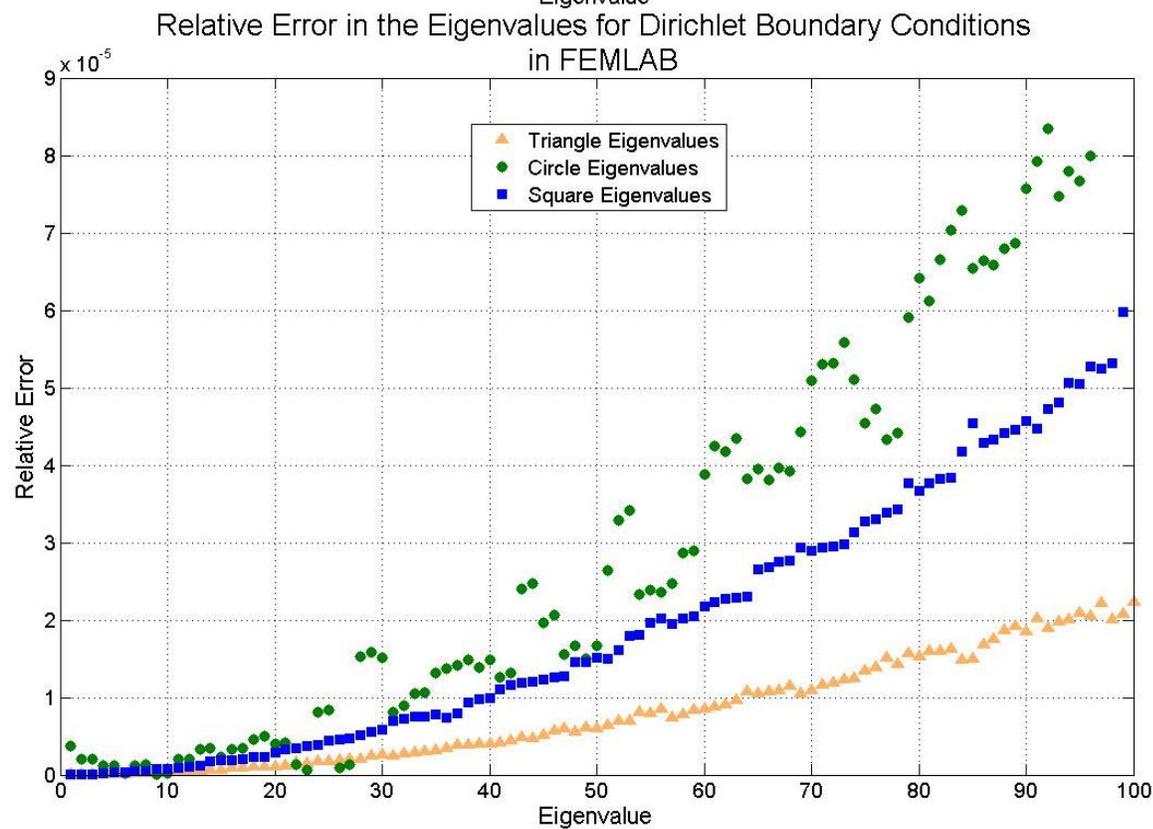
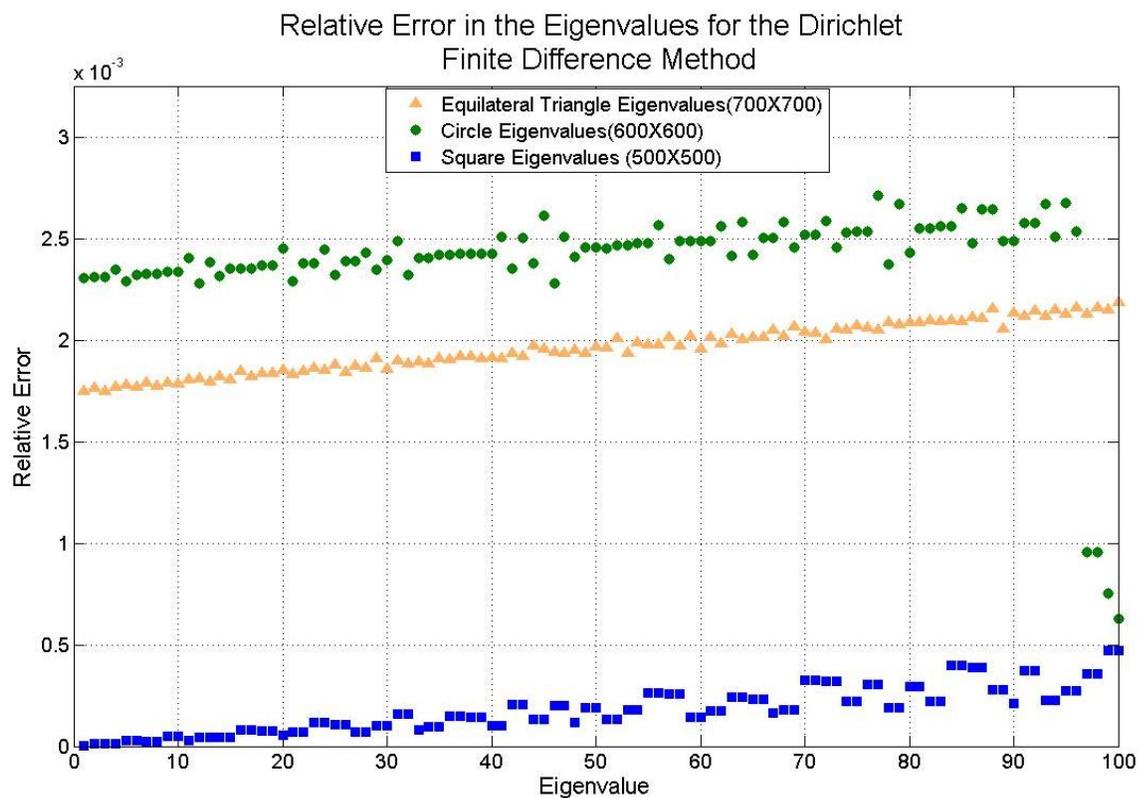


Figure 4.16: Convergence of the first eigenvalues versus theory of the equilateral triangle for Ames's method.

was over two orders of magnitude smaller than it was for the normal finite differencing method. While Ames's method removed the oscillations from the convergence of the circle, the oscillations for the equilateral triangle with Ames's method mirrored the oscillations from the normal finite difference method.

The relative error compared with theory of all one-hundred eigenvalues for each test shape was calculated for the normal finite differencing method. The relative error of all three shapes was plotted in Figure 4.17. For the square, circle and equilateral triangle the accuracy of each eigenvalue decreased linearly for each higher frequency. The total increase in relative error for each of the geometries between the first and hundredth eigenvalue was on the order of 5×10^{-4} . The difference in relative error between all the shapes for similar eigenvalues was less



Figures 4.17-4.18: Error in the first hundred eigenvalues for the test shapes with the finite difference method in MATLAB® and the finite element method in FEMLAB®.

than one order of magnitude. For the circle the last three eigenvalues have erroneously decreased error. No explanation for this occurrence has been forthcoming.

The relative error in the FEMLAB® solutions for the Dirichlet test shapes have been plotted in Figure 4.18. The FEMLAB® solutions showed an increase in accuracy compared to the finite differencing method of two orders of magnitude. As the eigenvalues increased the relative error in the eigenvalues increased exponentially. Because FEMLAB® used a triangular mesh to solve the geometries the equilateral triangle provided the least relative error and the slowest increase in error as the eigenvalues increased. Using the same line of reasoning and looking at the square as two triangles with their bases touching shows why the square has the next smallest relative error. The significant accuracy of the FEMLAB® solution made it a good tool for comparison with the finite differencing method for the Chesapeake Bay where no analytical solution was available for comparison.

The last error test used was the degree of orthogonality between two different potential solutions. Since the Dirichlet modes were supposed to describe unique behavior in the currents of the Chesapeake Bay, the inner product of each mode with another mode in the dimension should be zero. When computing eigenvectors of a matrix MATLAB® automatically sets the dot product of each eigenvector with itself to be one. Therefore, the dot product will be the inner product defined for the space of normal modes. The metric for this space will be one if the two eigenmodes describe exactly the same behavior and zero if the two eigenmodes describe orthogonal behavior. Because the first eigenmode was the most accurate for each of the test geometries it was used to test the orthogonality of all the other modes. For each of the test shapes and runs the dot product of the first mode was taken with each of the other modes. In every case the dot product of the first eigenvalue with itself was one. The square geometry had

inner products all from 10^{-15} – 10^{-17} which represents essentially all completely orthogonal behavior. The inner product for modes of the circle ranged from 10^{-14} – 10^{-19} and the equilateral triangle had the same range as the square. All of the modes for each test shape showed the expected orthogonality throughout all one-hundred eigenmodes. The orthogonality did not change significantly from the first run with few internal nodes to the runs with the most nodal resolution in any of the geometries. While the orthogonality did not change significantly over the different resolutions for the test geometries, deviations were seen for the Neumann boundary condition and more complicated geometries.

Chapter 5

Testing and Analysis of the Neumann Finite Difference Scheme

The Neumann boundary conditions were solved in two steps: first the algorithm was tested on three different shapes: the square, the circle and the equilateral triangle. Both the finite differencing method and FEMLAB® were used to solve each geometry for their eigenmodes. Ames's method was also used to find the eigenmodes of the circle. Secondly, the accuracy of the finite difference solutions for square and circle test geometries were examined by comparing the eigenvalues to theory, the eigenvectors were tested for orthogonality and convergence of the eigenvalues was tested. Because there is currently no known analytical solution for the equilateral triangle with Neumann boundary conditions it could not be examined versus theory. Therefore, the equilateral triangle's convergence was compared to its highest resolution and its error for each of the hundred eigenvalues was compared to the FEMLAB® solution. The data for the square and circle from this analysis was also compared to the solution from FEMLAB®.

The Square: The analytical solution to the square geometry with Neumann boundary conditions follows much the same path as the solution for Dirichlet boundary conditions. The separation of variables is the exact same as for Dirichlet. The only difference in the separation of variables

solution is that a good guess for a solution to the square with Neumann boundary conditions is a cosine function rather than a sine. Therefore,

$$\Gamma(x) = \cos(gx) \quad (0.95)$$

$$\Pi(y) = \cos(fy) \quad (0.96)$$

The derivative of equation (0.95) with respect to x must equal zero at $x=0$ and at $x=1$. The derivative of equation (0.96) with respect to y must be equal to zero at $y=0$ and at $y=1$. After the application of the boundary conditions f and g have the same value as for the Dirichlet solution. The only difference is that n and m can have any integer value from zero to infinity since $\cos(0) \neq 0$. The solution for the square with Neumann boundary conditions is,

$$\Phi(x, y) = \cos\left(\frac{n\pi x}{L_x}\right) \cos\left(\frac{m\pi y}{L_y}\right) \quad (0.97)$$

The equation for the eigenvalues is exactly the same as for Dirichlet boundary conditions except that both n and m integers can be zero valued,

$$\lambda = \pi^2 \left(\frac{n^2}{L_x^2} + \frac{m^2}{L_y^2} \right). \quad (0.98)$$

The finite differencing method was applied to the square geometry with grids starting at ten nodes on each side and increasing by ten until there were one-hundred nodes per side. The number of nodes was then increased by increments of one-hundred until there were five-hundred nodes per side. This corresponded to grids with 64, 324, 784, 1444, 2304, 3364, 4624, 6084, 7744, 9604, 39204, 88804, 158404 and 248004 internal nodes for each of the runs. The potential for the Neumann mode was plotted in the exact same manner as the Dirichlet potential. As can be seen from the plots the derivative boundary condition allows freedom of the mode along the boundary, producing a flag waving effect. The Neumann boundary condition also allows for a

Square Neumann Potential Mode 2 (400X400)

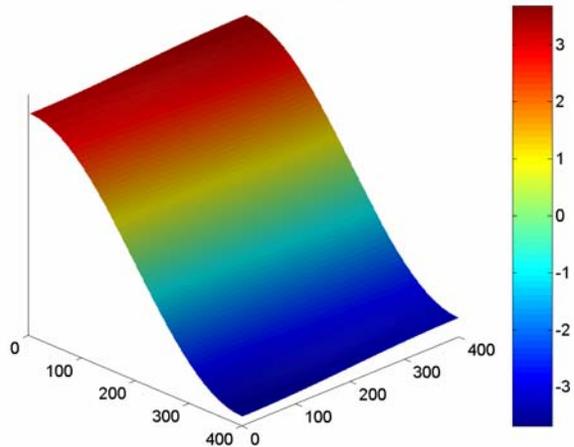


Figure 5.1

Square Neumann Potential Mode 3 (400X400)

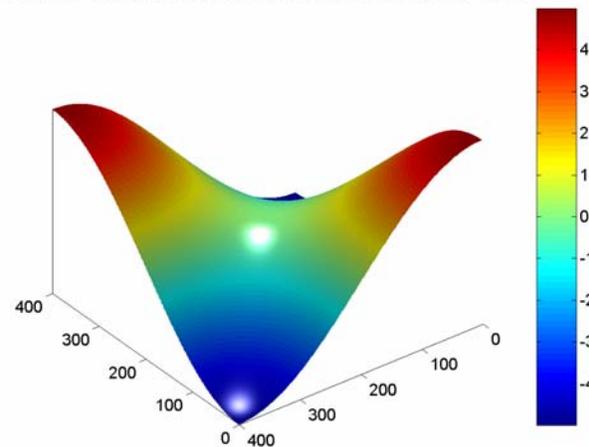
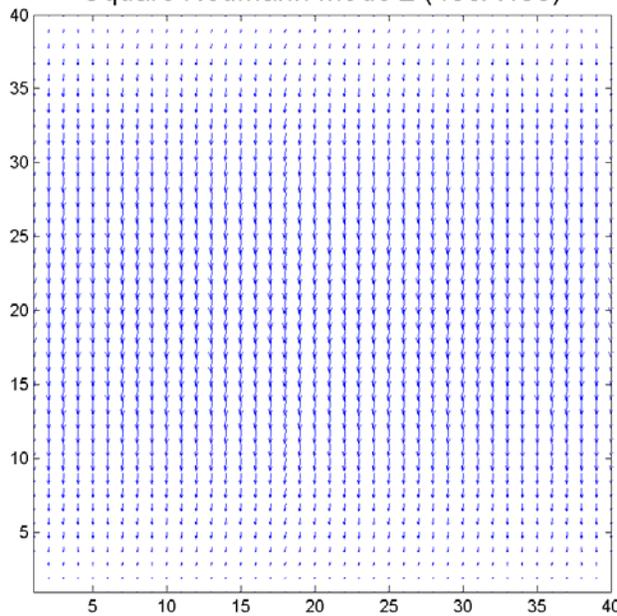
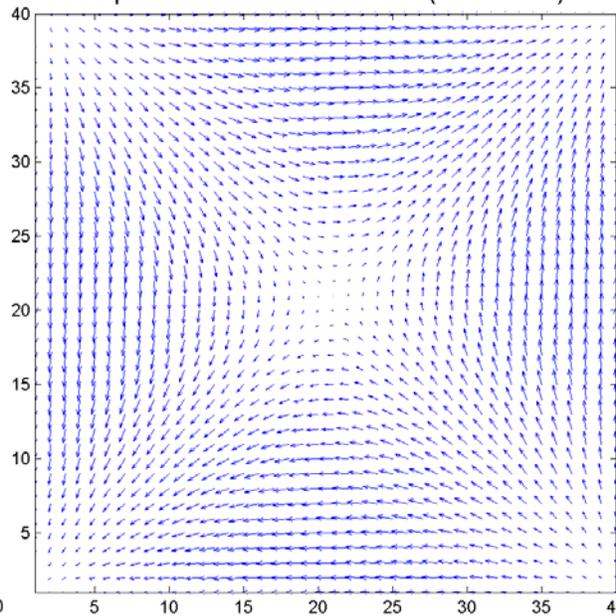


Figure 5.2

Square Neumann Mode 2 (400X400)



Square Neumann Mode 3 (400X400)



Figures 5.3-5.4: Vector fields created from their respective potentials and equation (0.25). To create these plots only 1:10 of the original rows and columns were used.

potential solution that is flat across the entire geometry, which corresponds to the first mode of the square and an eigenvalue of zero. Therefore, the second and third modes of the square have been plotted using “surf.” The quiver plot of the actual mode was calculated from the potential using a central differencing scheme with equation (0.25). This produces flows that are characterized by their lack of gyres. Mode one was a straight flow from one boundary to the other.

The Circle: The analytical solution to the circle with Neumann boundary conditions follows almost exactly the same path as the circle with Dirichlet conditions. First, separation of variables was used to derive,

$$\left(r^2 \frac{\partial^2 R}{\partial r^2} + 2r \frac{\partial R}{\partial r} \right) + R(r^2 \lambda - n^2) = 0 \text{ and} \quad (0.99)$$

$$\frac{\partial^2 \Theta}{\partial \theta^2} - n^2 \Theta = 0. \quad (0.100)$$

The solutions to these two equations are Bessel functions of the first kind. However, the eigenvalues of the solution are the zeros of the first derivative of the Bessel function. Mathematica was used to find the zeros of the normal derivative of the Bessel function.

The geometry was then solved using both the finite differencing method and FEMLAB®. Runs were done in MATLAB® with resolutions of 60, 276, 648, 1184, 1876, 2724, 3720, 4872, 6180, 7668, 31064, 70168, 124980, 195496 and 281760 internal nodes for each of the runs. The potentials and vector fields for the second and third modes have been plotted in figures 5.5-5.9 using the same techniques as for the square geometry. The vector fields for the second and third modes are very similar to the vector fields for the square. Mode two shows the straight flow from one end of the geometry to the other while mode three is divided into four quarters with straight flows along the boundary in each quarter.

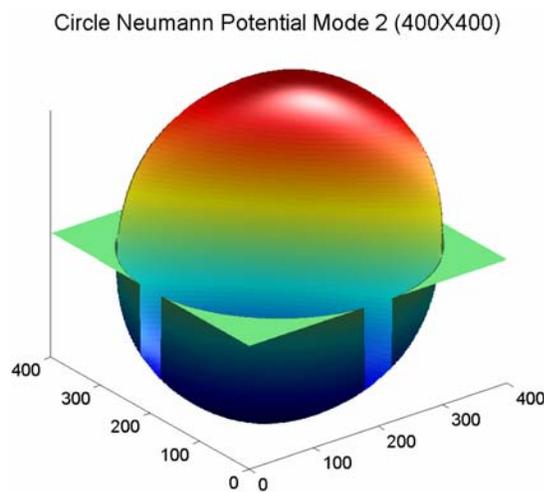


Figure 5.6

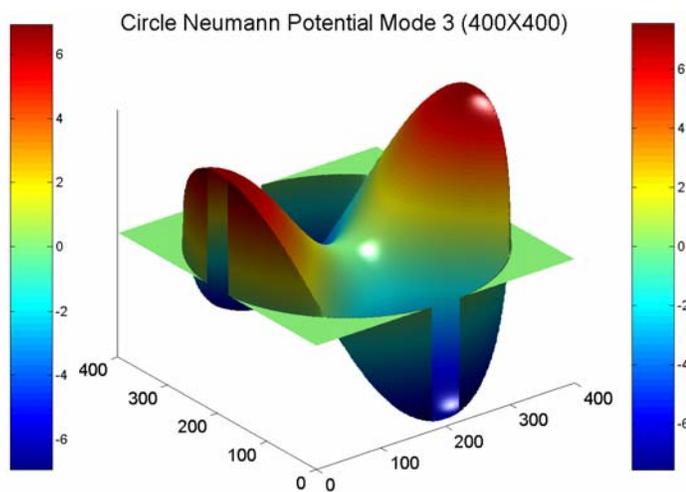
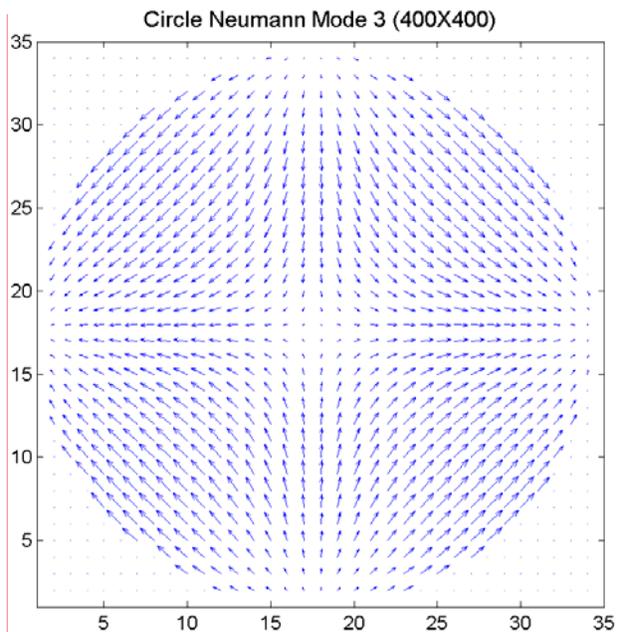
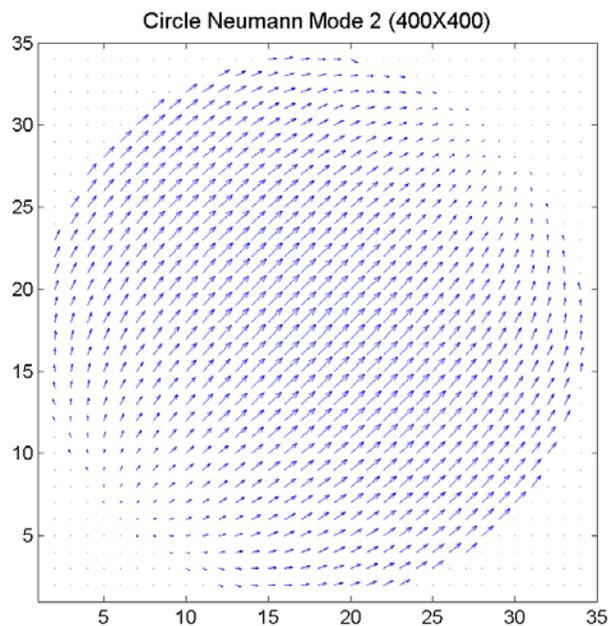


Figure 5.7



Figures 5.8-5.9: Vector fields created from their respective potentials and equation (0.25). To create these plots only 1:11 of the original rows and columns were used.

Equilateral Triangle Neumann Potential Mode 2 (600X600)

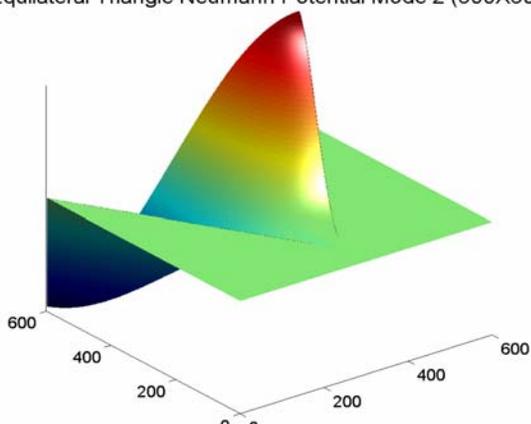


Figure 5.10

Equilateral Triangle Neumann Potential Mode 3 (600X600)

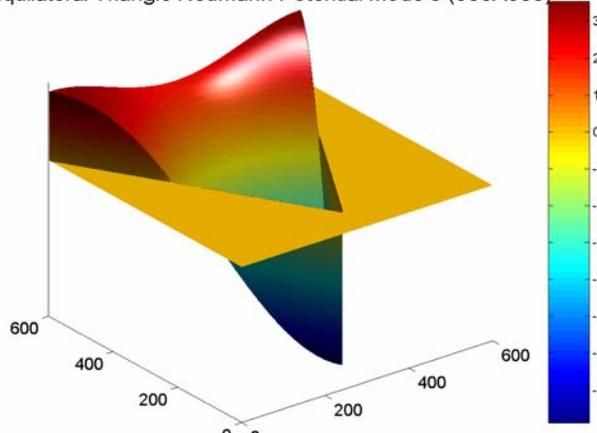
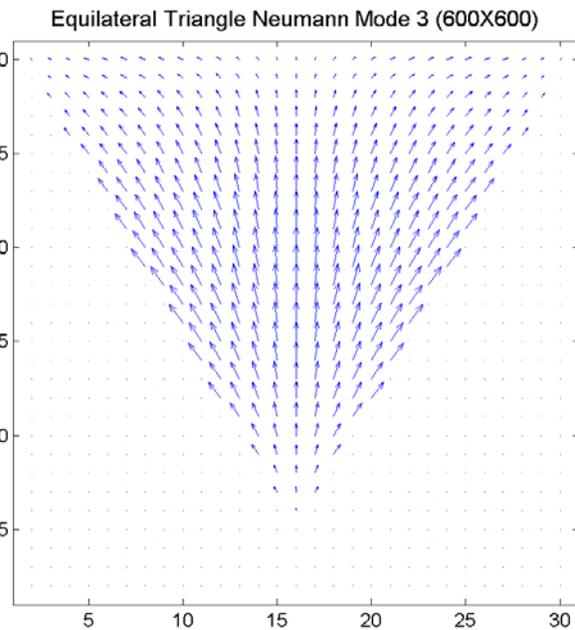
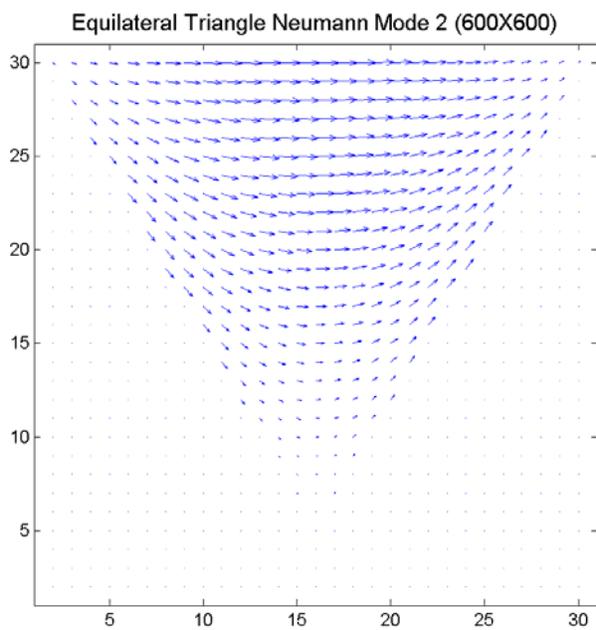


Figure 5.11



Figures 5.12-5.13: Vector fields created from their respective potentials and equation (0.25). To create these plots only 1:20 of the original rows and columns were used.

The Triangle: As of now, there are no published analytical solutions to the equilateral triangle for Neumann boundary conditions. Consequently, no comparison to theoretical solutions could be applied. The equilateral triangle was solved using both the finite difference method in MATLAB® and the finite element method in FEMLAB®. The equilateral triangle was solved on grids with resolutions of 144, 346, 633, 1008, 1470, 2017, 2652, 3374, 4181, 5076, 17022, 38522, 68683, 107504 and 154984 nodes internal to the geometry. The solutions have been plotted in figures 5.10-5.13.

Analysis of Neumann Test Shapes: After the eigenmodes of the square, circle and equilateral triangle were evaluated numerically, the convergence of the second eigenvalue for each of the test shapes was examined. The second eigenvalues was chosen because the first eigenvalue for any geometry with Neumann boundary conditions is zero. The convergence of the second eigenvalue for the square was plotted in Figure 5.14. The number of internal nodes was equivalent to the runs completed for the square with Dirichlet boundary conditions. As can be seen the approximation for the square with Neumann boundary conditions provided accuracy as good as or better than the Dirichlet conditions. It also coincided with the predicted theoretical error based the expansion used in the Taylor Series approximation. Convergence similar to the Dirichlet approximation was expected. Both the normals and boundary of the Neumann square were input exactly into the finite differencing method allowing for the error in the second eigenvalue to be dominated by the truncation error from the Taylor series expansion.

The convergence of the circle with Neumann boundary conditions for each of the test runs was calculated and plotted in figure 5.15. The convergence of the eigenvalues showed the same oscillations that were present in the runs for the circle with Dirichlet boundary conditions. Again, the likely cause was the ability for certain wavelengths of the eigenvalues to resonate

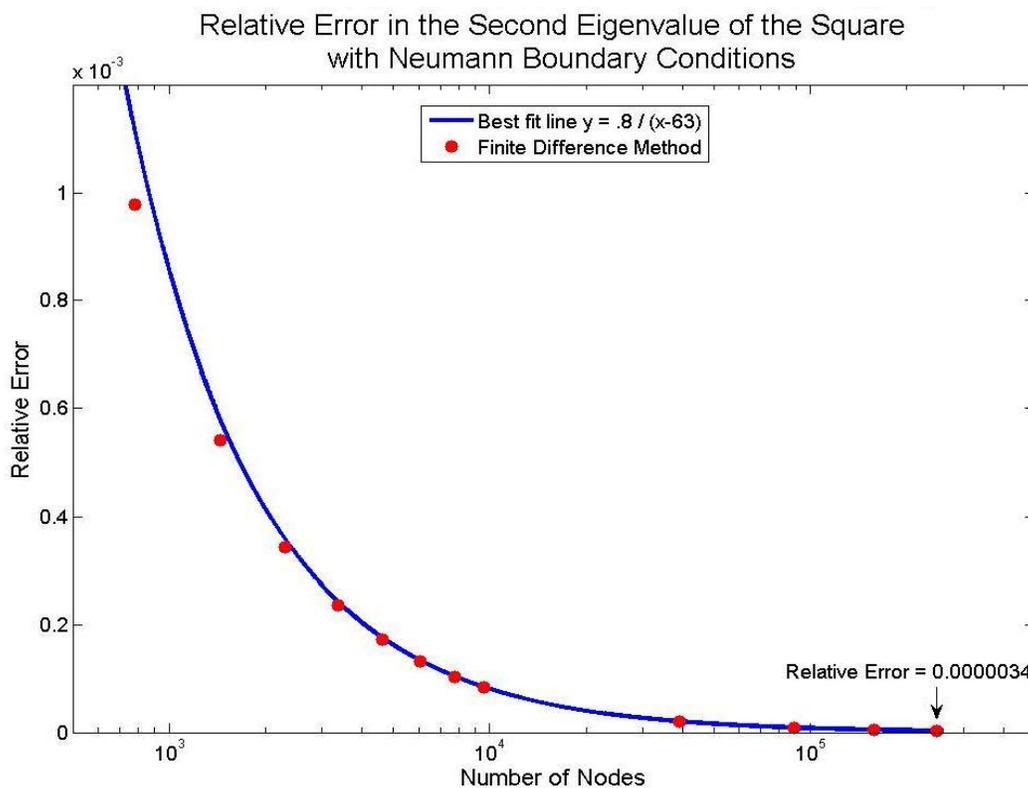


Figure 5.14: Graph of relative error in the second eigenvalue of the finite difference approximation versus theory.

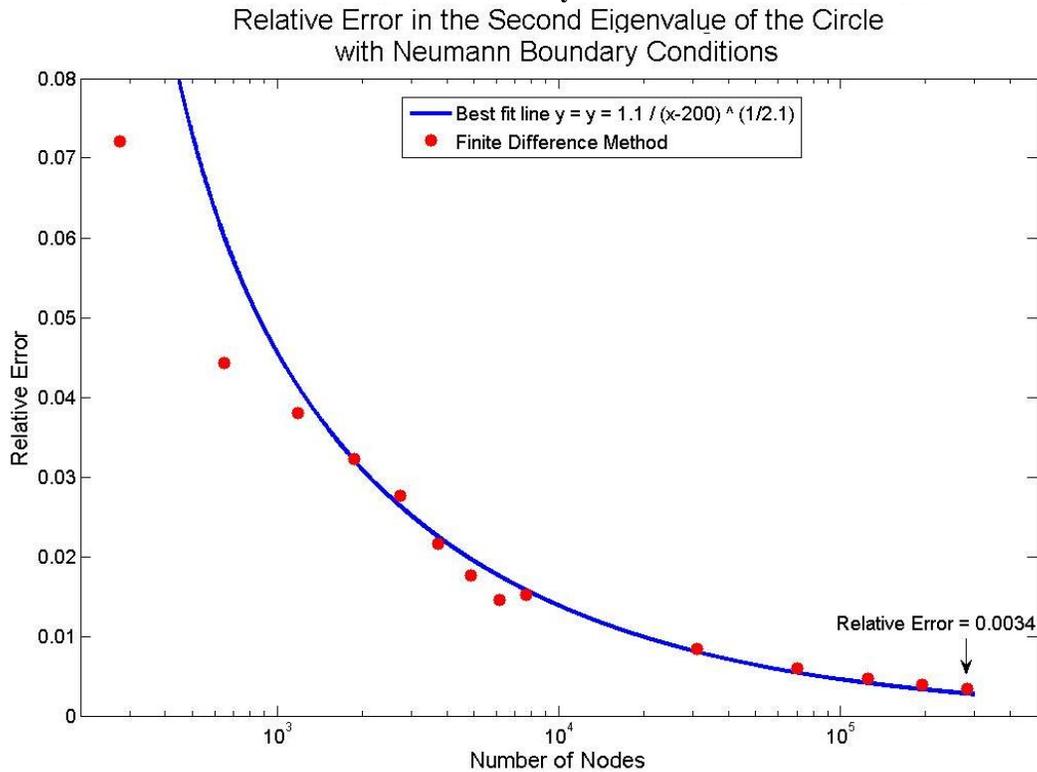


Figure 5.15: Graph of relative error in the second eigenvalue of the finite difference approximation versus theory.

with the boundary created using a square grid. The oscillations in the square grid around the exact boundary allowed slightly different wavelengths to fit inside of the geometry causing the error to oscillate as the average position of the grid points moved around the boundary. The error in the second eigenvalue was converging towards the theoretical value. However its

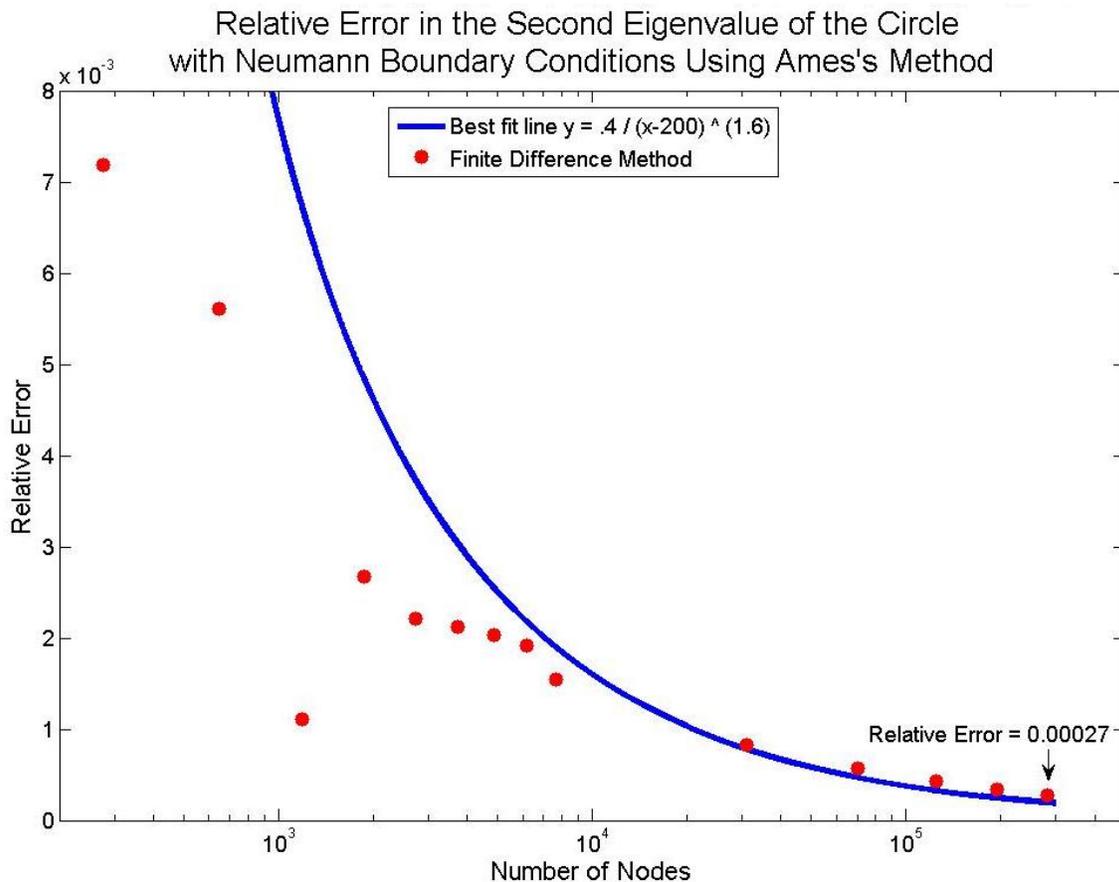


Figure 5.16: Graph of relative error in the second eigenvalue of the Ames finite difference approximation versus theory.

accuracy was significantly less than that for the square. The primary explanation is that for Neumann boundary conditions not only were the positions of the boundary nodes approximated for the geometry, the normals for the boundary conditions were approximate as well.

The calculation of the circle with Neumann boundary conditions using Ames's method was plotted in figure 5.16. The oscillations of the previous solutions for the circle were dampened into one small hump in the convergence. The residual hump may have occurred due

to the fact that the directional derivative only deals with the x and y derivatives and therefore only approximates the true normal derivative. The rate of convergence has increased slightly from the normal finite difference method. However, while the convergence has not increased

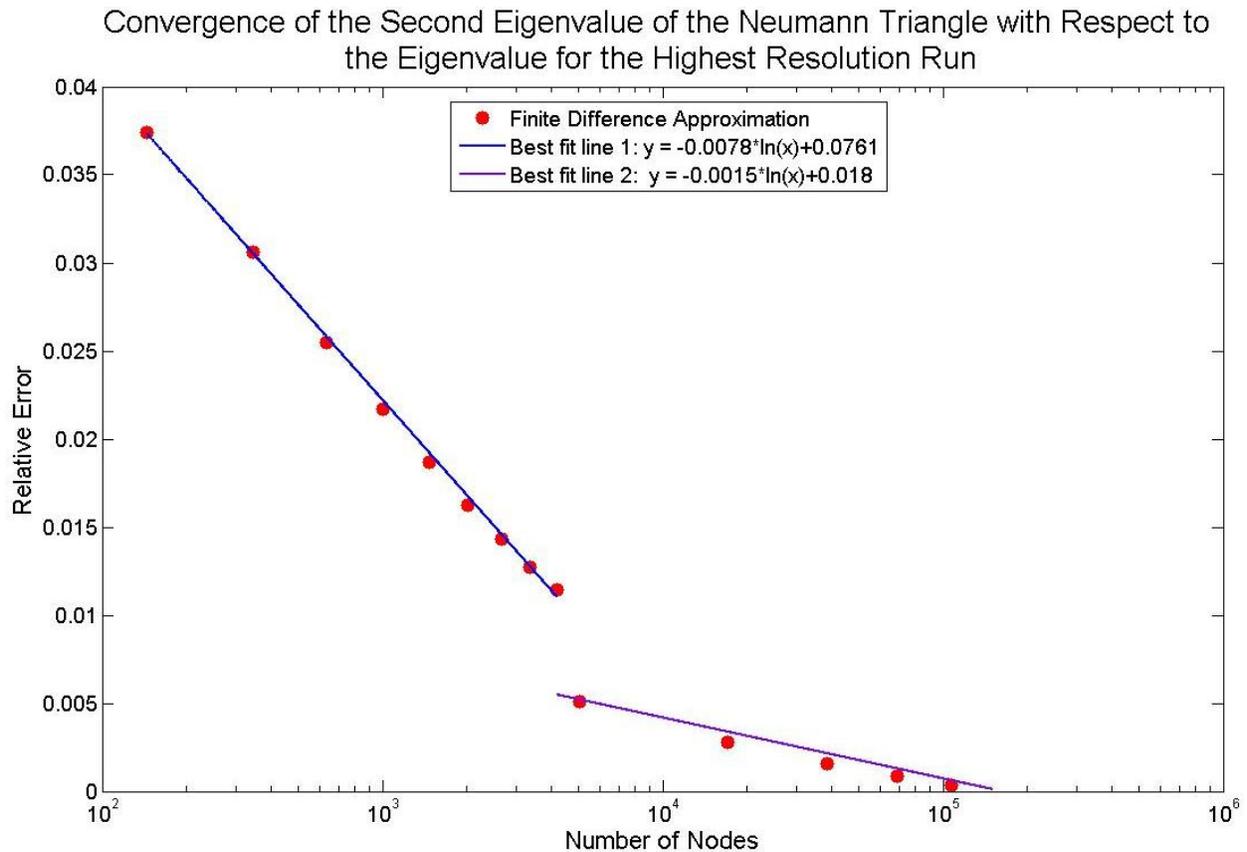


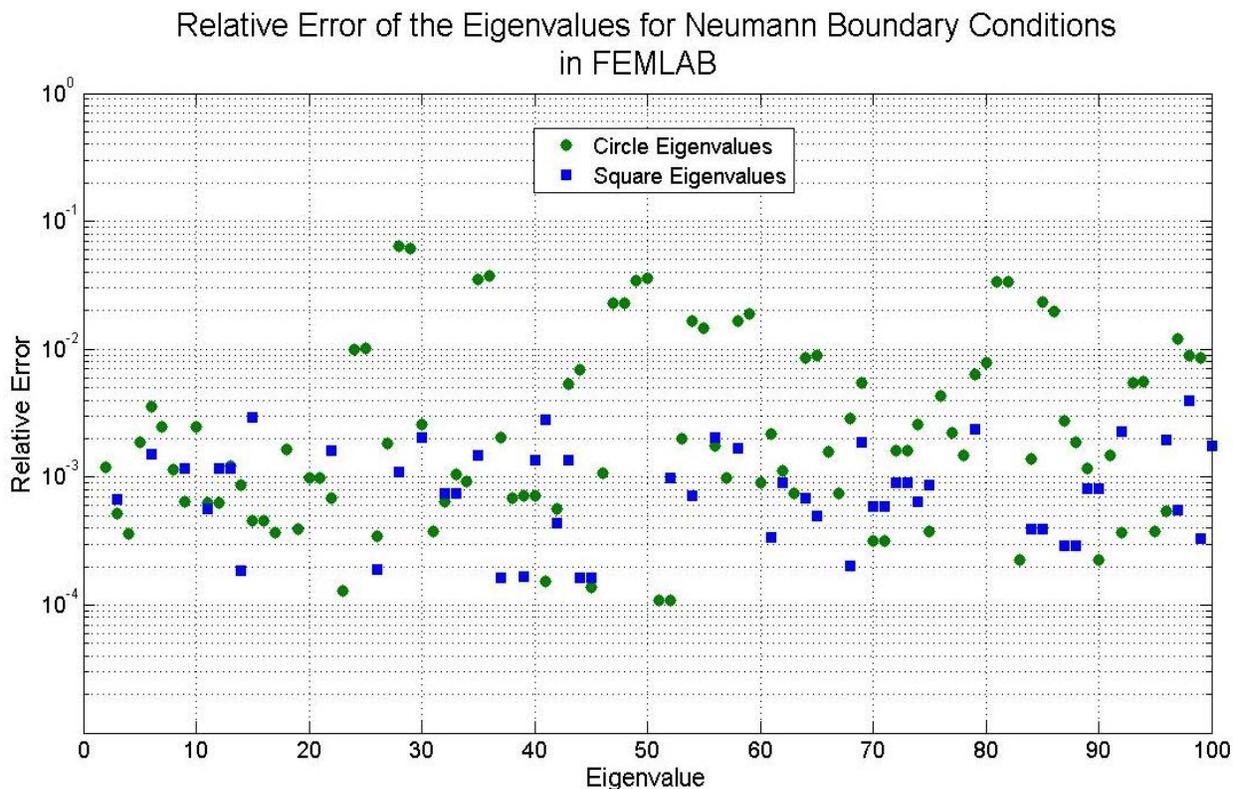
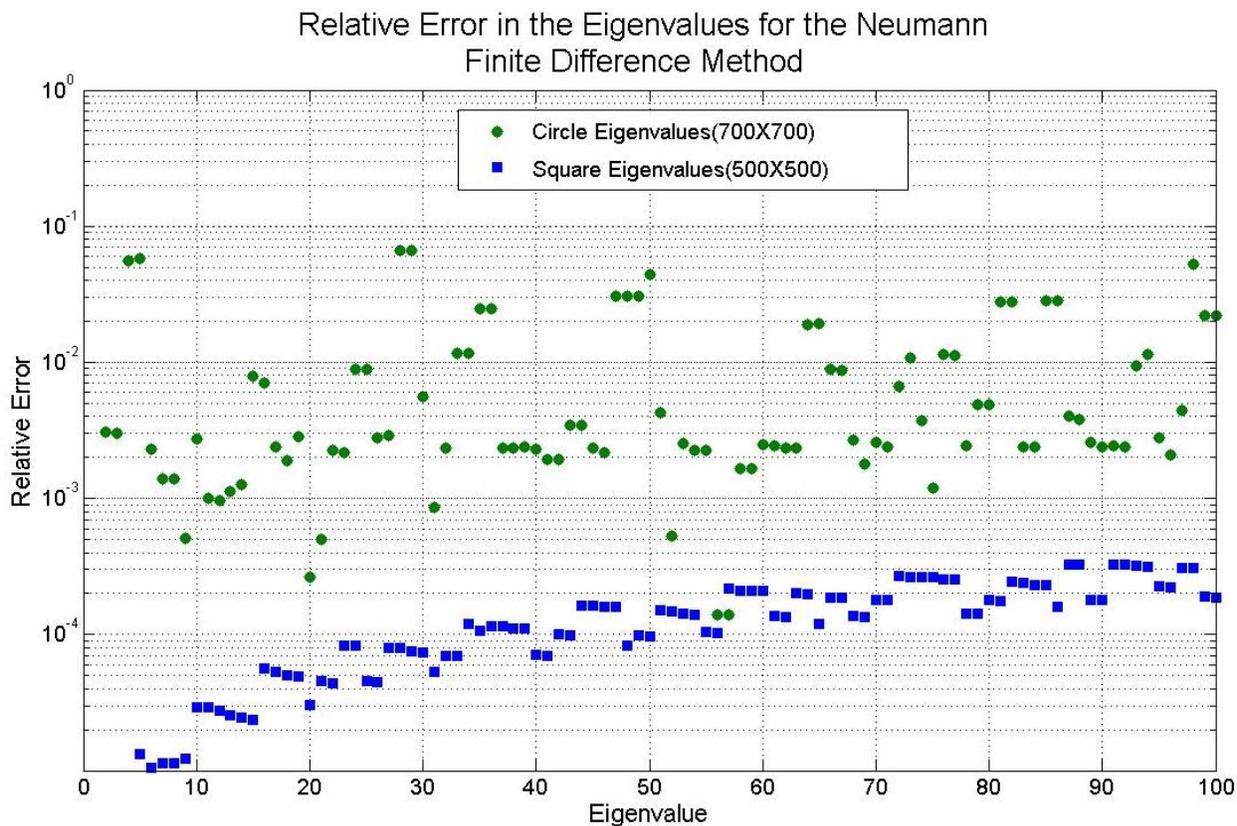
Figure 5.17: Graph of relative error in the second eigenvalue of the finite difference approximation versus the eigenvalues at the highest resolution

drastically, the initial solution was one order of magnitude more accurate for the same number of internal nodes and this greater accuracy continued throughout the test runs.

The convergence of the equilateral triangle with Neumann boundary conditions has been plotted in Figure 5.17. Because there was no analytical solution available for comparison the convergence was plotted with respect to the second eigenvalue of the run with the highest resolution, 154984 internal nodes. The convergence of the second eigenvalue was split into two distinct zones. Between the zones there was a rather large discontinuity with two very different rates of convergence. The jump occurred between the runs with 3374 and 4181 internal nodes.

The first zone shows much faster convergence while the second zone takes a significant jump in accuracy but has a much slower rate of convergence. No explanation for the discontinuity has been forthcoming. However, both sections are converging just at different rates about the discontinuity. The relative convergence shows that the eigenvalues are not erratic and are moving towards an answer.

The relative error of the eigenvalues of the highest resolution runs for the circle and square were also compared to theory. The relative error for both the eigenvalues of the square and circle has been plotted in Figure 5.18. The solutions to the square were very accurate from the lowest to the highest eigenvalues. As the eigenvalue number increased the error in the square eigenvalues for the finite differencing method appeared to be leveling off to an asymptote. This asymptote appeared to be approximately five-hundredths of a percent. The eigenvalues for the circle were much more erratic. There is a strong line of eigenvalues with errors on the order of five-tenths of a percent while others range from just under six percent error to one-thousandth of a percent error. This erratic behavior was most likely due to coupling between the gridded shape of the circle, the approximated normals of the circle and the eigenmodes that were calculated. The deviations from the exact shape and normals may have had enough effect that their interaction with each specific eigenmode overrode the relative error for the eigenvalues. The relative error versus theory for Ames's method for the circle was also examined. The relative error for Ames's method had a similar although slightly smaller amount of erratic behavior. However, the largest band of error was shifted down one order of magnitude to coincide with the error asymptote that the eigenvalues of the square were approaching. The relative error of the eigenvalues calculated using FEMLAB® have been computed and plotted in Figure 5.19. The general erratic behavior shown for the finite difference method was increased for the FEMLAB®



Figures 5.18-5.19: Relative Error in the first hundred eigenvalues of the square and circle versus theory for the finite difference method in MATLAB® and the finite element method in FEMLAB®

solution to include the square as well. The general accuracy of the eigenvalues of the square was increased by half an order of magnitude while that for the circle was decreased by half an order of magnitude so that both relative error plots occupied the same area of the plot. Figures 5.18 and 5.19 show a significant increase in erratic behavior and a decrease in accuracy from the Dirichlet solutions. However, the accuracy of the eigenvalues was still generally under one percent and did not show drastic increases at higher eigenvalues.

The relative error for the first hundred eigenvalues for the equilateral triangle was examined by taking the relative error of the finite difference solution against the FEMLAB® solution and plotting it in Figure 5.20. The first and last eigenvalues showed on the order of ten percent error. There was also a significant number of eigenvalues in a band below one percent error showing that in the case of the equilateral triangle, for which FEMLAB® should be very

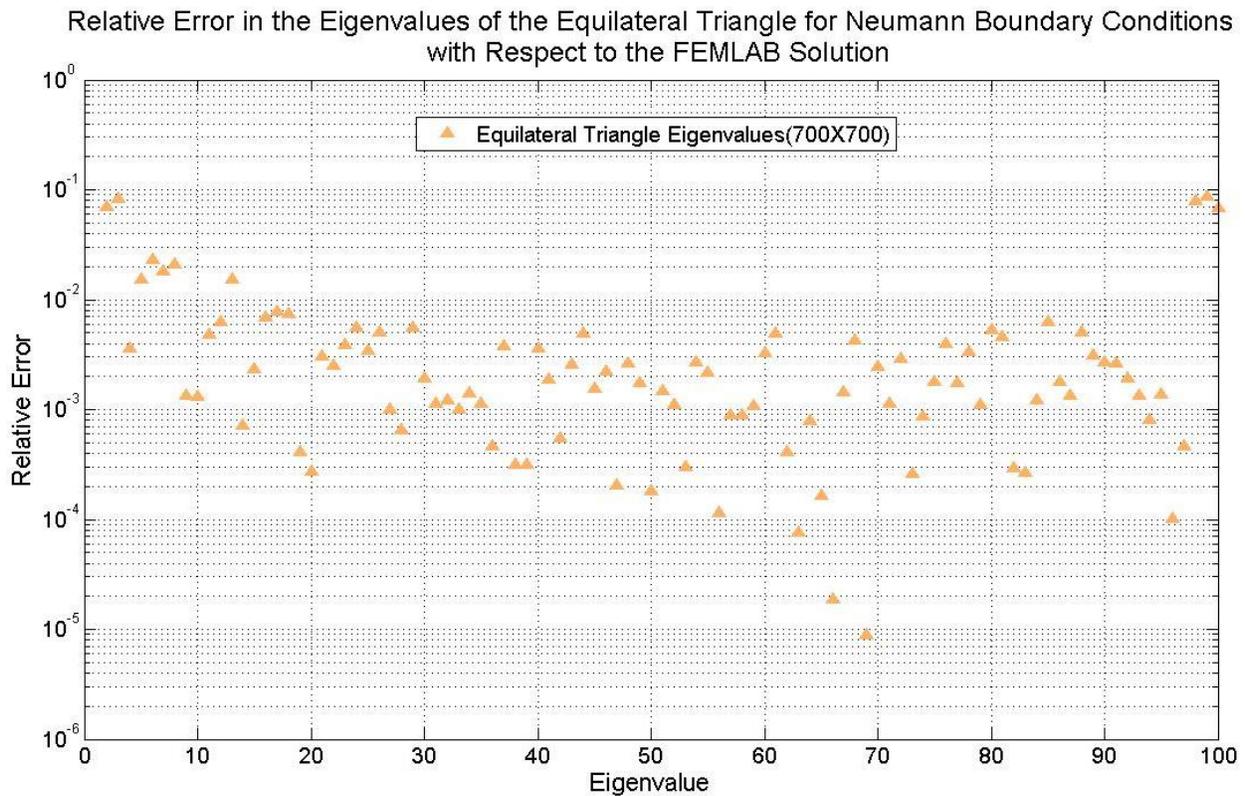


Figure 5.20: Relative error in the first hundred eigenvalues of the equilateral triangle with respect to the FEMLAB® solution.

accurate since it uses triangular elements, both the finite differencing solution and the finite element solution coincided.

The last test of error completed for the test geometries using finite difference solutions with Neumann boundary conditions was the orthogonality of the eigenmodes. The method and theory for testing the orthogonality was exactly the same as for the Dirichlet eigenmodes. In the space covered by the Neumann and Dirichlet potentials, orthogonal eigenmodes are eigenmodes whose dot product is equal to zero. The modes were tested against the first eigenmode of each geometry. The orthogonality of the modes decreased significantly from the eigenmodes for the Dirichlet boundary conditions. For the square, slightly over one quarter of the eigenvalues were greater than 10^{-7} with a maximum value on the order of 10^{-3} . The rest of the eigenmodes had dot products less than 10^{-15} . The dot products for the circle erroneously increased for the final test run with the highest resolution so that the minimum value was 10^{-9} . However, the next most resolved run had only fourteen modes with dot products less than 10^{-7} . The rest of the eigenmodes for the circle ranged from 10^{-11} to 10^{-17} . The equilateral triangle had a similarly erroneous decrease in orthogonality for the highest resolution run. Overall its eigenmodes were the least orthogonal of all of the test shapes. Fifty-three of the eigenmodes had dot products greater than 10^{-6} with a maximum value for one eigenmode on the order of 10^{-1} and an average on the order of 10^{-3} . The other forty-seven eigenmodes had dot products between 10^{-10} and 10^{-16} . All three test geometries showed primarily orthogonal eigenmodes. The equilateral triangle proved to be the least orthogonal geometry with one mode having approximately ten percent of the behavior of the lowest eigenmode. Just as in the Dirichlet solutions the orthogonality of the eigenmodes did not show significant changes over the range of grid resolutions.

The essentially constant orthogonality of the eigenmodes across resolutions for both Dirichlet and Neumann boundary conditions on the test geometries highlights the difference between the type of error that the eigenvalues show and the type of error that the orthogonality shows. Having the correct eigenvalue requires a very accurate boundary since the wavelengths that fit into the geometry are entirely dependent upon the shape and area. Consequently, as the boundary was confined to decreasing areas the eigenvalues were forced to converge as long as the MATLAB® solved the differentiation matrix correctly. The test geometries also showed that not only did the eigenvalues converge they also converged to the correct theoretical eigenvalue when theoretical values were available. The orthogonality, however, was not in comparison to theoretical values. Rather, the orthogonality compared approximate solutions of eigenmodes with exactly the same boundary. Therefore, a significant increase in orthogonality should not have been expected over different resolutions if orthogonality was a fundamental attribute of the equation being solved. The orthogonality then showed how well the finite differencing approximation in MATLAB® constrained the eigenmodes. For the Dirichlet boundary conditions the orthogonality showed that the eigenmodes were constrained very tightly by the applied approximation. However, the constraints for the Neumann boundary conditions were lacking when compared to the Dirichlet solutions. This leads to the possible conclusion that the approximation of the normal derivatives in the differentiation scheme leaves some degrees of freedom of the Neumann eigenmodes unconstrained that should not be free.

Chapter 6

Vorticity Potentials of the Chesapeake Bay

Once the method used for solving Dirichlet boundary conditions was validated it was applied to the Chesapeake Bay. The first step that was taken was the creation of a storage matrix in MATLAB® for the shape of the Chesapeake Bay. Two different outlines for the Chesapeake Bay were tested. The first outline was the exact outline of the Chesapeake Bay taken from a picture of the bathymetry of the Bay.³⁵ This picture was touched up in Photoshop. Five of the rivers were removed from the Western edge of the bay: the Potomac, Rappahannock, York and James. In order to put the Photoshop version of this image into a version compatible with the MATLAB® program it was necessary to change the color scheme to gray scale and then import it into MATLAB®. MATLAB® was then used to convert the color of each pixel in the picture from Photoshop to either a one or a zero.

Once the storage matrix was produced the Bay was solved for Dirichlet boundary conditions using the same method as for the three test geometries. Because the Chesapeake Bay was a better fit to a rectangle, the width of the bottom of the bay was set to one while the length of the bay was the ratio of the number of nodes on along the length of the storage matrix divided by the number of nodes along the width of the storage matrix; the Chesapeake Bay was inscribed

on a rectangle with a base of length one and a height of length 3.386. The exact version of the Bay was solved for nine different resolutions where the number of internal nodes in the geometry was increased for each run.

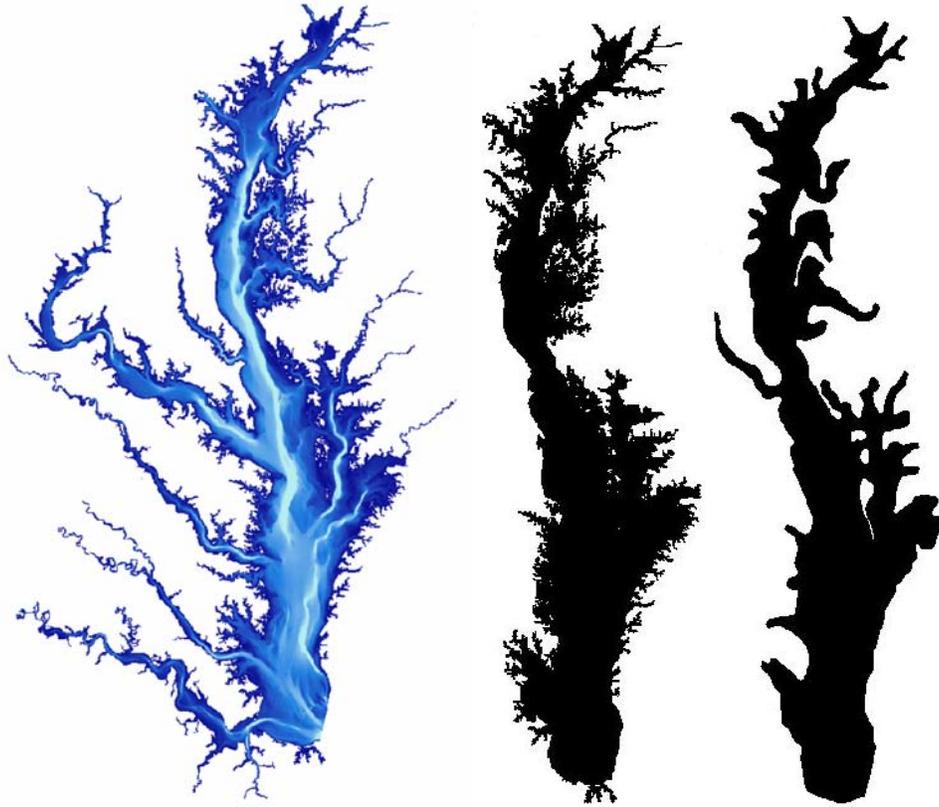


Figure 6.1: The picture on the left was taken from the Chesapeake Information Management Systems website, http://www.chesapeakebay.net/maps/pub_maps.cfm. The figure in the middle is the black and white approximation of the bay that was used in the calculations. The figure on the right is the version of the Chesapeake Bay created from the Quoddy boundary nodes.

The number of nodes for each run was: 9670, 22484, 40175, 62805, 89872, 111515, 158493, 247817 and 356883. The eigenmodes of the Chesapeake Bay corresponded to the general expectations for a geometry with a number of large areas separated by smaller connecting areas. The first eigenvalue filled the largest open area in the Bay. As the eigenvalues increased the wavelength became smaller and could resolve the smaller sections of the Bay. The modes where one frequency resonated with a specific portion of the bay have been plotted in figures 6.2-6.9. Each section of the Bay was touched by at least ten of the 100 modes that were calculated

providing good coverage for the entire geometry. The eigenvalues for the exact version of the Chesapeake Bay ranged from 84.1 to 1725.9.

The other boundary that was evaluated for the Chesapeake Bay was the boundary from Quoddy. The coordinates for each of the boundary points in the Quoddy domain were taken and input into FEMLAB®. FEMLAB® has a CAD based vector drawing system which was used to create a solid boundary from the Quoddy points. This boundary was first solved in FEMLAB®, then the JPEG image of the geometry from FEMLAB® was input into MATLAB® in the same manner as the picture of the bathymetry. Nine different runs were done for the Quoddy boundary in MATLAB® with resolutions of 10235, 16344, 23876, 32873, 44838, 67702, 97334, 132605 and 173154 internal nodes. The less resolved Quoddy boundary was computed for two primary reasons. First, the geometry could be input into FEMLAB® with good accuracy and in a reasonable amount of time. Creation of a vector model out of the more resolved boundary was attempted but a satisfactory vector model could not be completed. Secondly, Quoddy provided a model from which source terms could be taken to solve for the inhomogenous modes. Therefore in order for the inhomogenous modes to be useful the Dirichlet and Neumann modes had to be solved on the Quoddy boundary as well. Selected eigenmodes of the finite difference solution to both Chesapeake Bay boundaries have been plotted below. The first set of plots is for the high resolution boundary while the second set of plots is for the Quoddy boundary.

Dirichlet Mode 1 (400X1403)

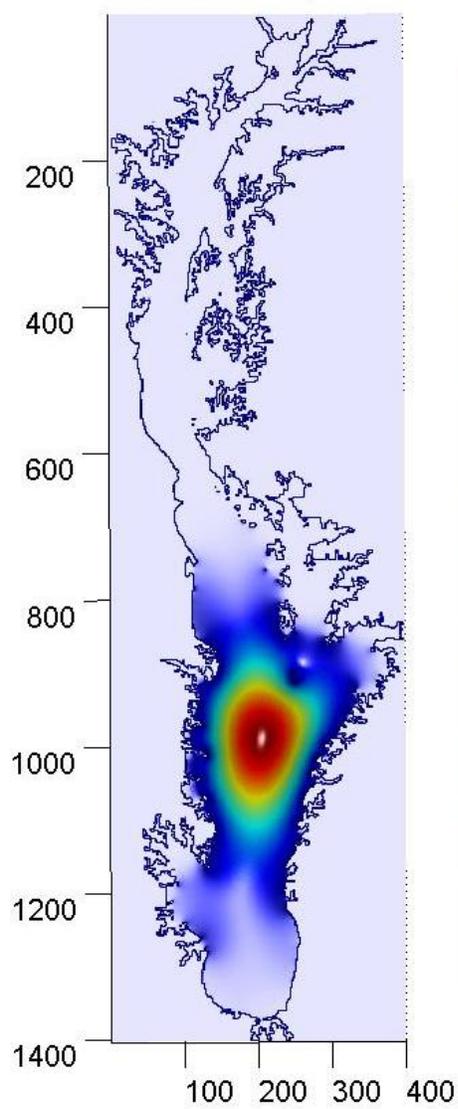


Figure 6.2

Dirichlet Mode 4 (400X1403)

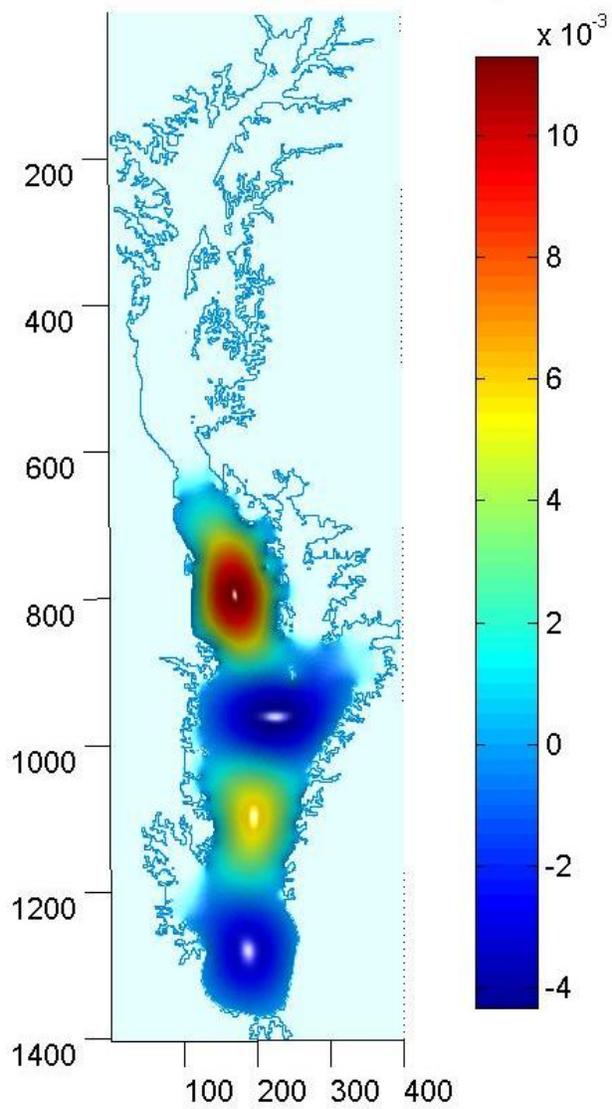


Figure 6.3

Dirichlet Mode 8 (400X1403)

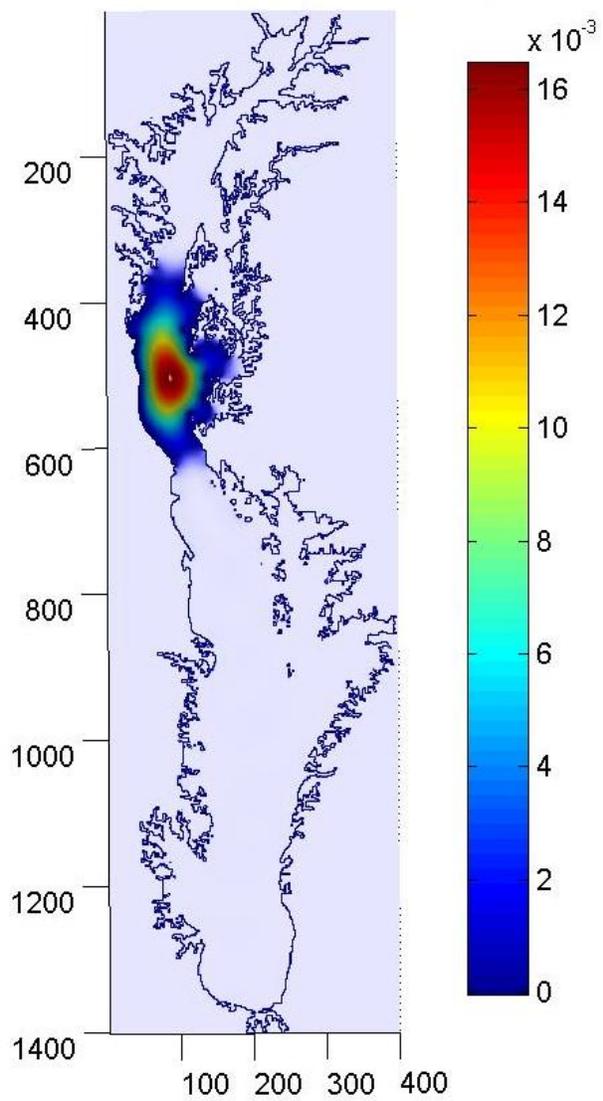


Figure 6.4

Dirichlet Mode 12 (400X1403)

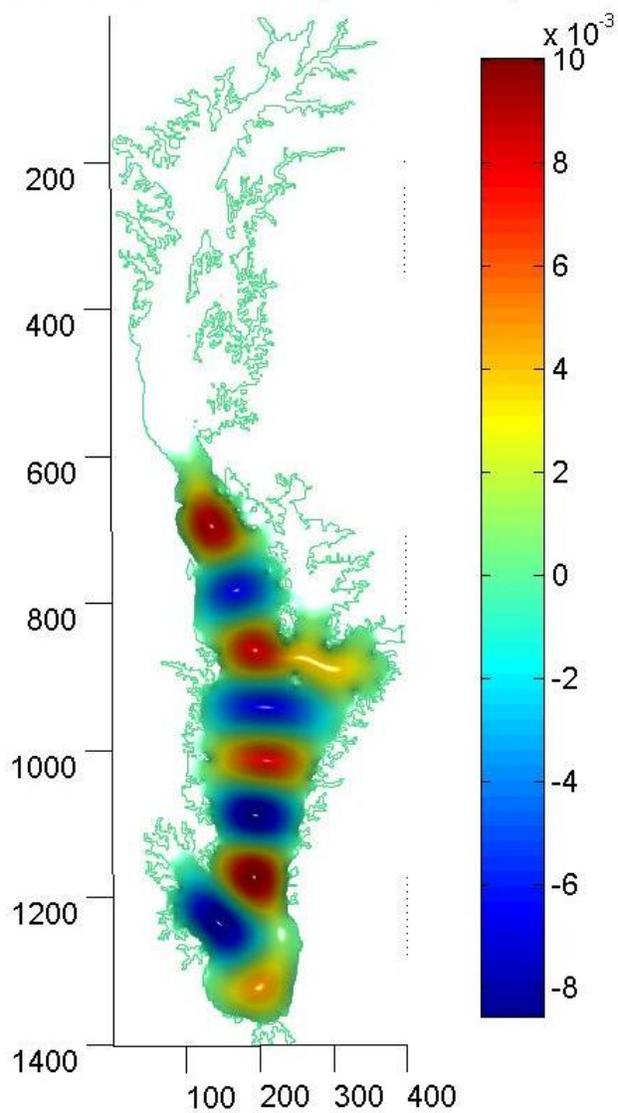


Figure 6.5

Dirichlet Mode 14 (400X1403)

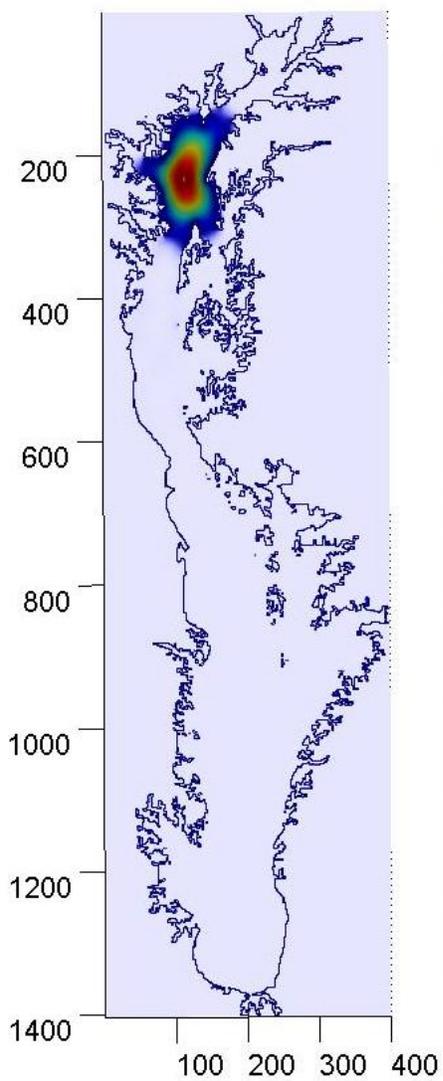


Figure 6.6

Dirichlet Mode 49 (400X1403)

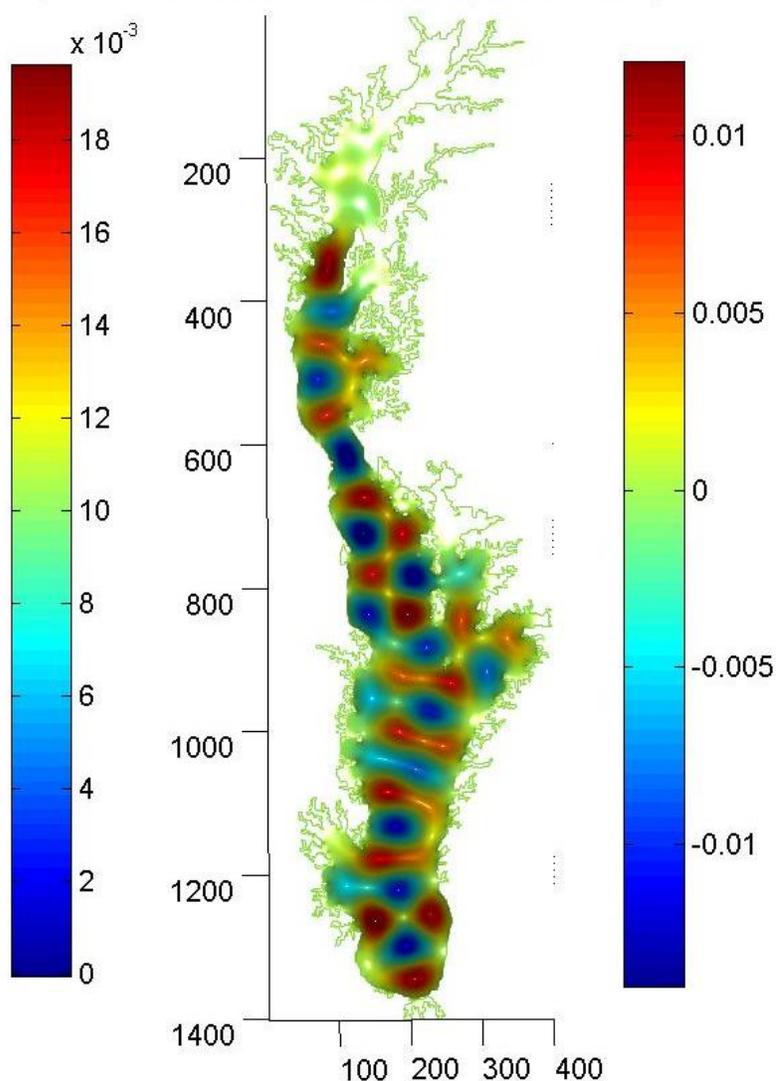


Figure 6.7

Dirichlet Mode 80 (400X1403)

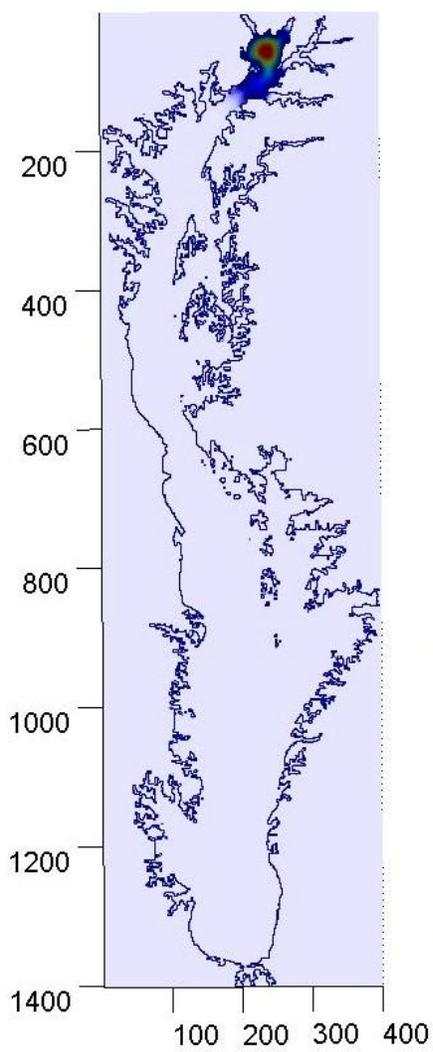


Figure 6.8

Dirichlet Mode 95 (400X1403)

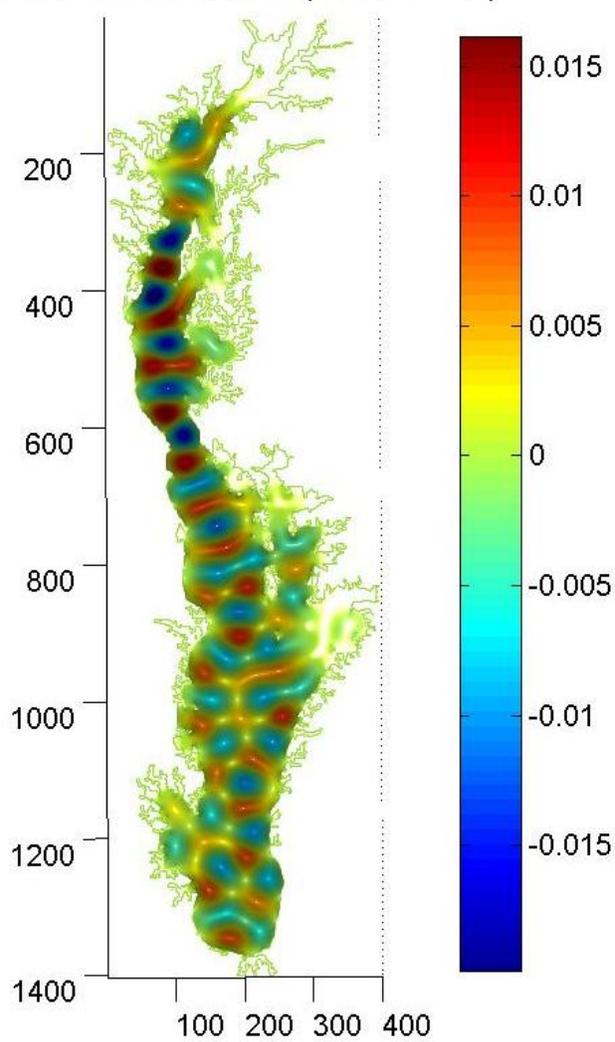


Figure 6.9

Dirichlet Mode 1 (350X1185)

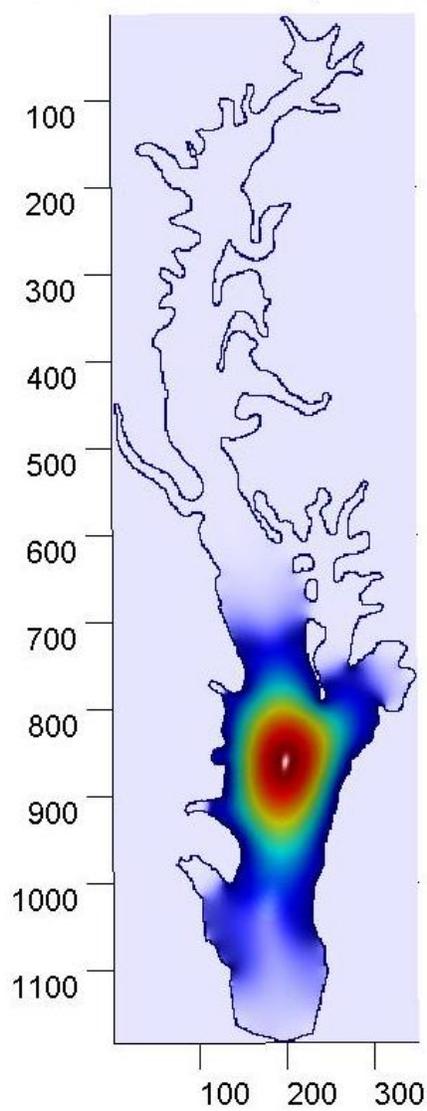


Figure 6.10

Dirichlet Mode 4 (350X1185)

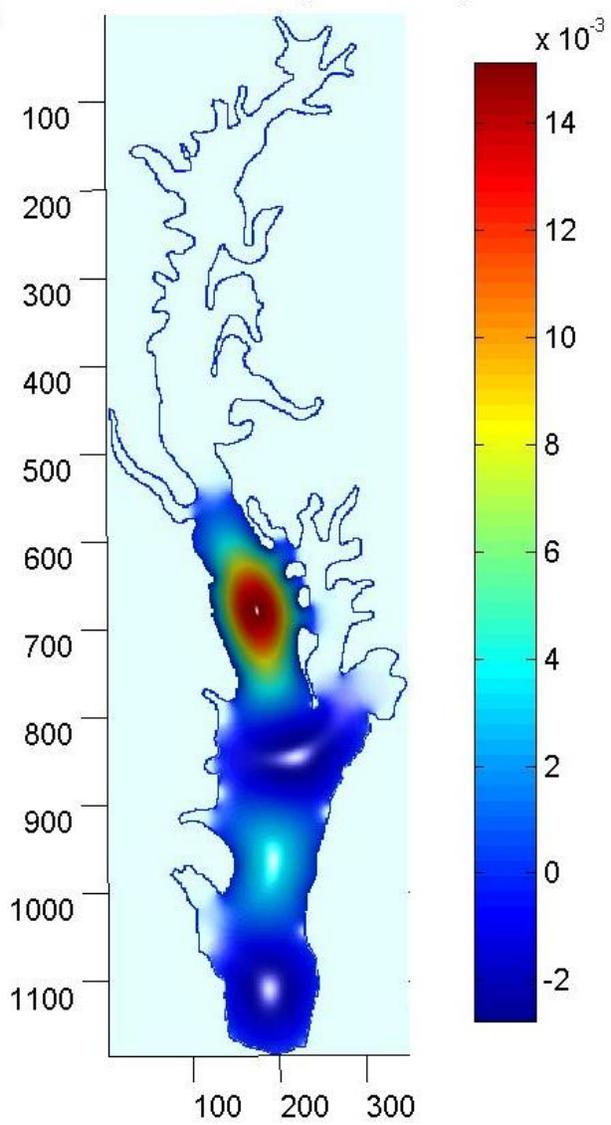


Figure 6.11

Dirichlet Mode 7 (350X1185)

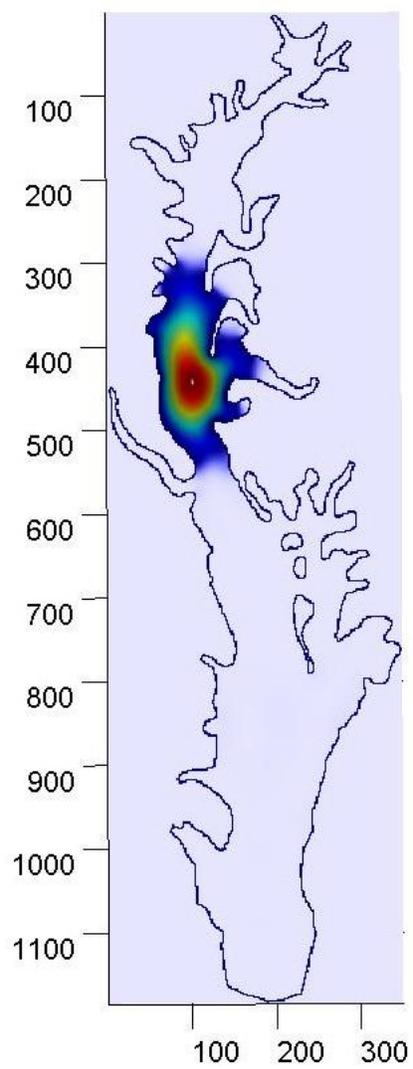


Figure 6.12

Dirichlet Mode 10 (350X1185)

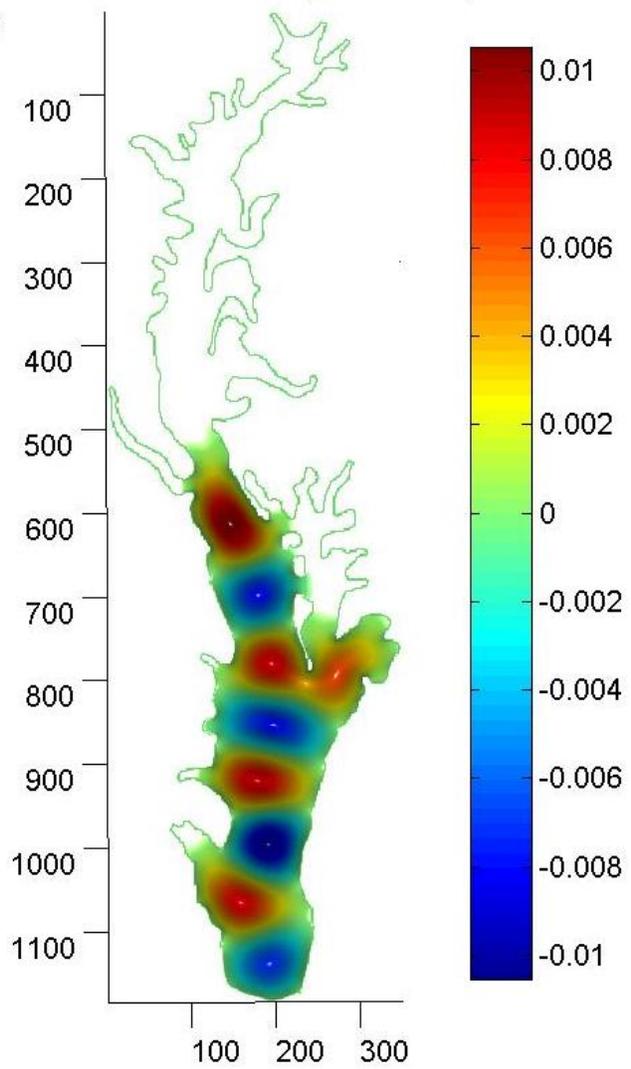


Figure 6.13

Dirichlet Mode 12 (350X1185)

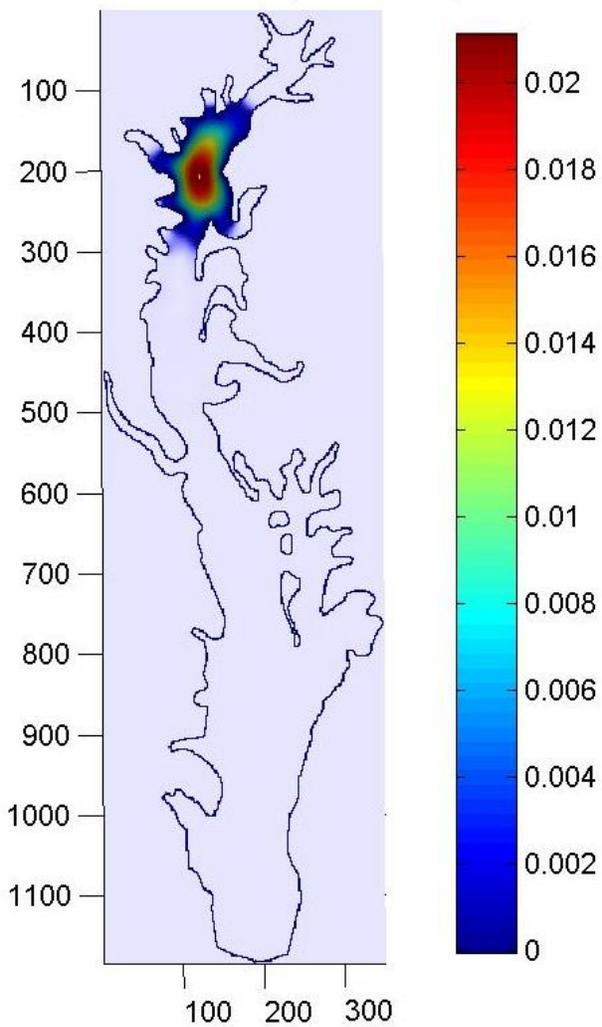


Figure 6.14

Dirichlet Mode 35 (350X1185)

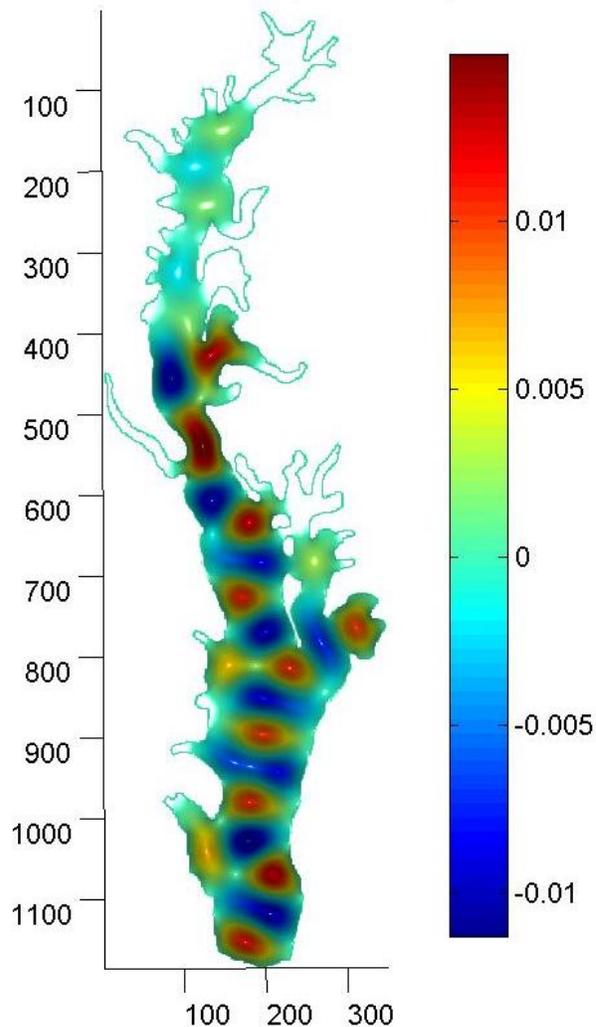


Figure 6.15

Dirichlet Mode 52 (350X1185)

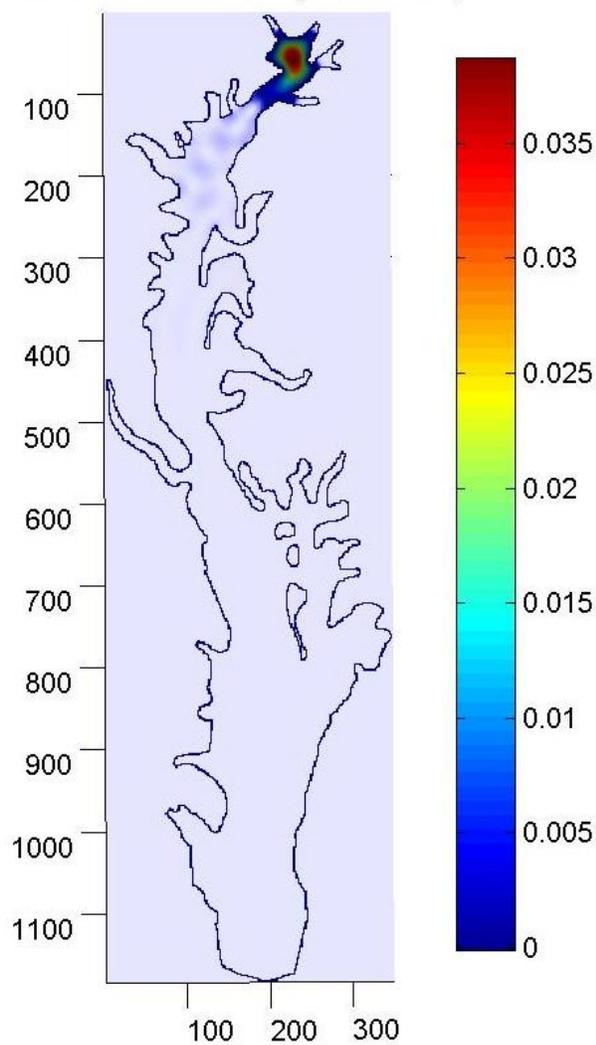


Figure 6.16

Dirichlet Mode 98 (350X1185)

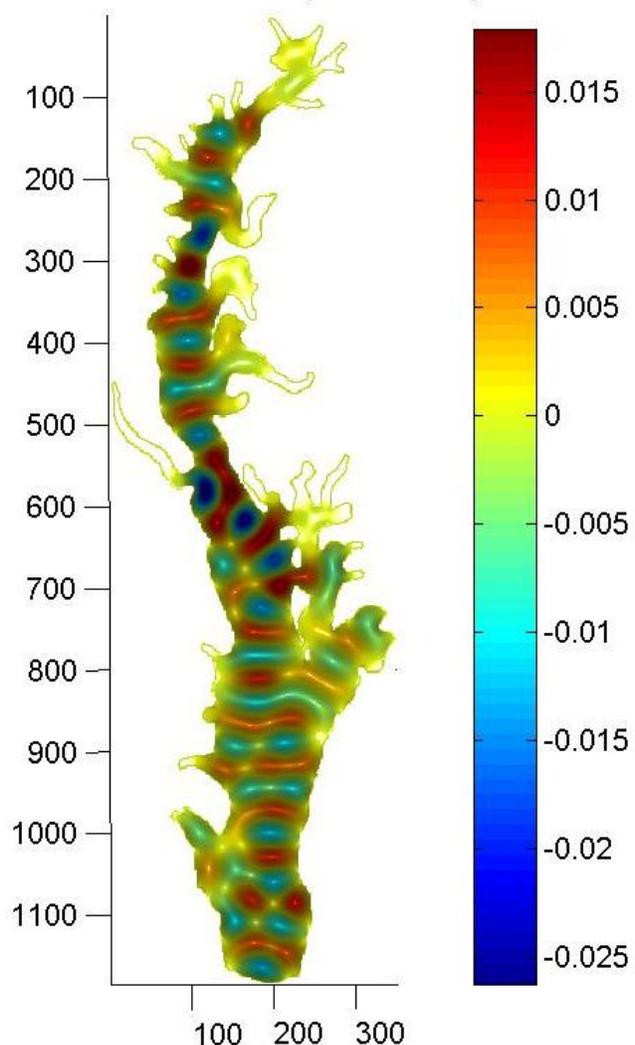


Figure 6.17

Chapter 7

Velocity Potentials of the Chesapeake Bay

After the eigenmodes of the Chesapeake Bay with Dirichlet boundary conditions were computed, the finite differencing method for Neumann boundary conditions was used to solve for the velocity potentials of the bay. Both of the different Chesapeake Bay geometries were solved for the velocity potential. The exact same resolutions were used as in the solutions for Dirichlet boundary conditions for the finite difference method. For the highly resolved boundary the runs were completed with 9670, 22484, 40175, 62805, 89872, 111515, 158493, 247817 and 356883 internal nodes. For the boundary from the Quoddy mesh, runs were completed with 10235, 16344, 23876, 32873, 44838, 67702, 97334, 132605 and 173154 internal nodes. The eigenvalues for the highly resolved solution ranged from 45.0 to 1048.4. The eigenvalues for the Quoddy boundary solution ranged from 27.0 to 882.3. The Quoddy boundary which was input into FEMLAB® was then solved using the finite element method. The eigenvalues from the finite element solution for the Quoddy boundary ranged from 0.00 to 610.0.

The eigenmodes of the Neumann solutions quickly covered the entire Bay. At least ten eigenmodes covered each section of the geometry. The first eigenmode for both the highly resolved boundary and the Quoddy boundary was a hump in the largest section of the Bay. The hump appeared to be shifted over from the first eigenmode of the Dirichlet solution so that the

boundary resided on the zero derivative line along the hump. The higher eigenmodes were able to resolve smaller sections of the Bay than the Dirichlet eigenmodes. Eigenmode fifty-eight for the Quoddy boundary was chosen to illustrate this point. The eigenvalue of this mode resonates with the Patuxent River, the longest protrusion on the Western side of the Bay, so that the primary oscillation of eigenmode fifty-eight was on the river itself. The flag waving effect seen in the eigenmodes of the square were very pronounced in eigenmodes fifteen and thirty-four for the Quoddy boundary. In these eigenmodes the Bay appears to have waves traveling up and down their entire length. Selected eigenmodes of the finite difference solution to both Chesapeake Bay boundaries have been plotted below. The first set of plots is for the high resolution boundary while the second set of plots is for the Quoddy boundary.

Neumann Mode 1 (400X1403) Neumann Mode 4 (400X1403)

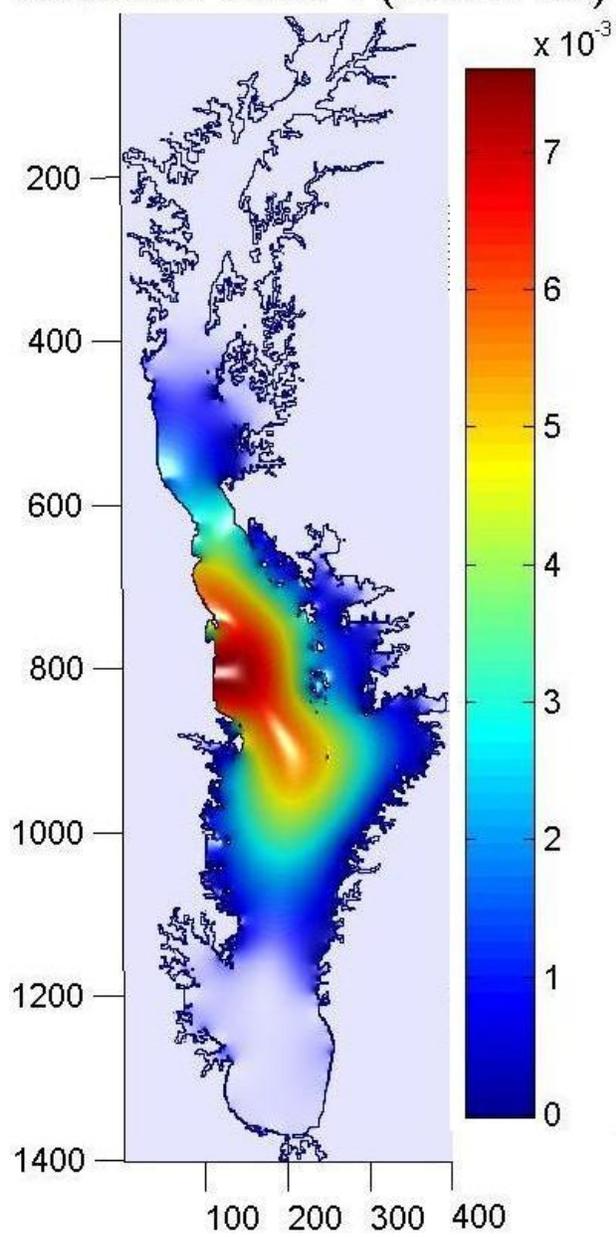


Figure 7.1

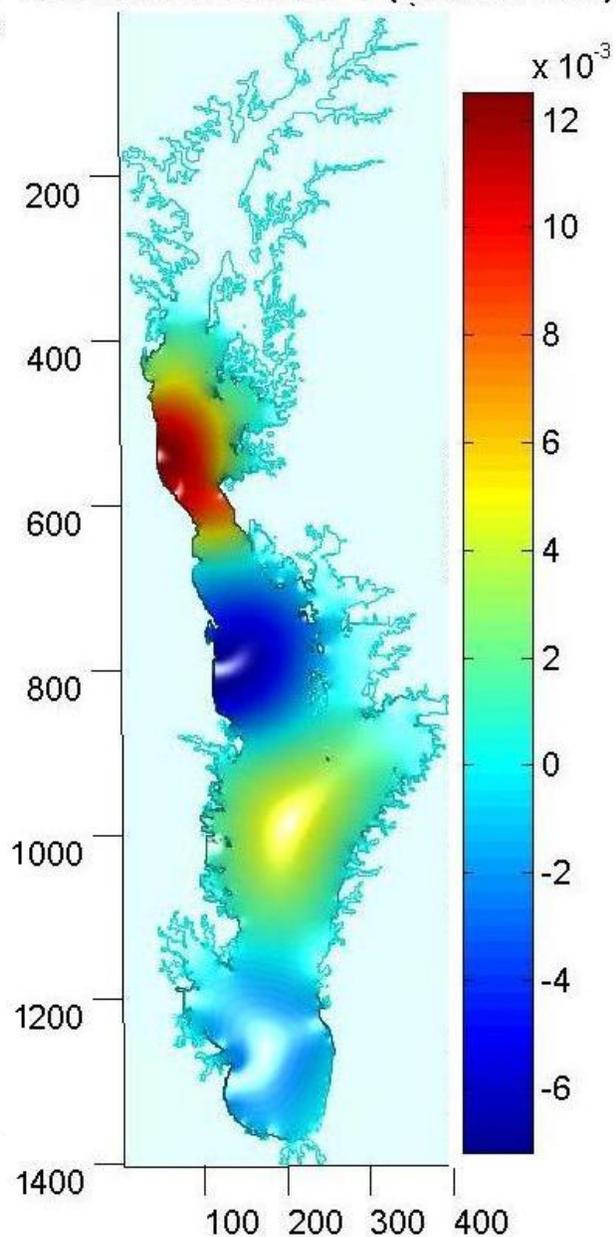


Figure 7.2

Neumann Mode 15 (400X1403) Neumann Mode 34 (400X1403)

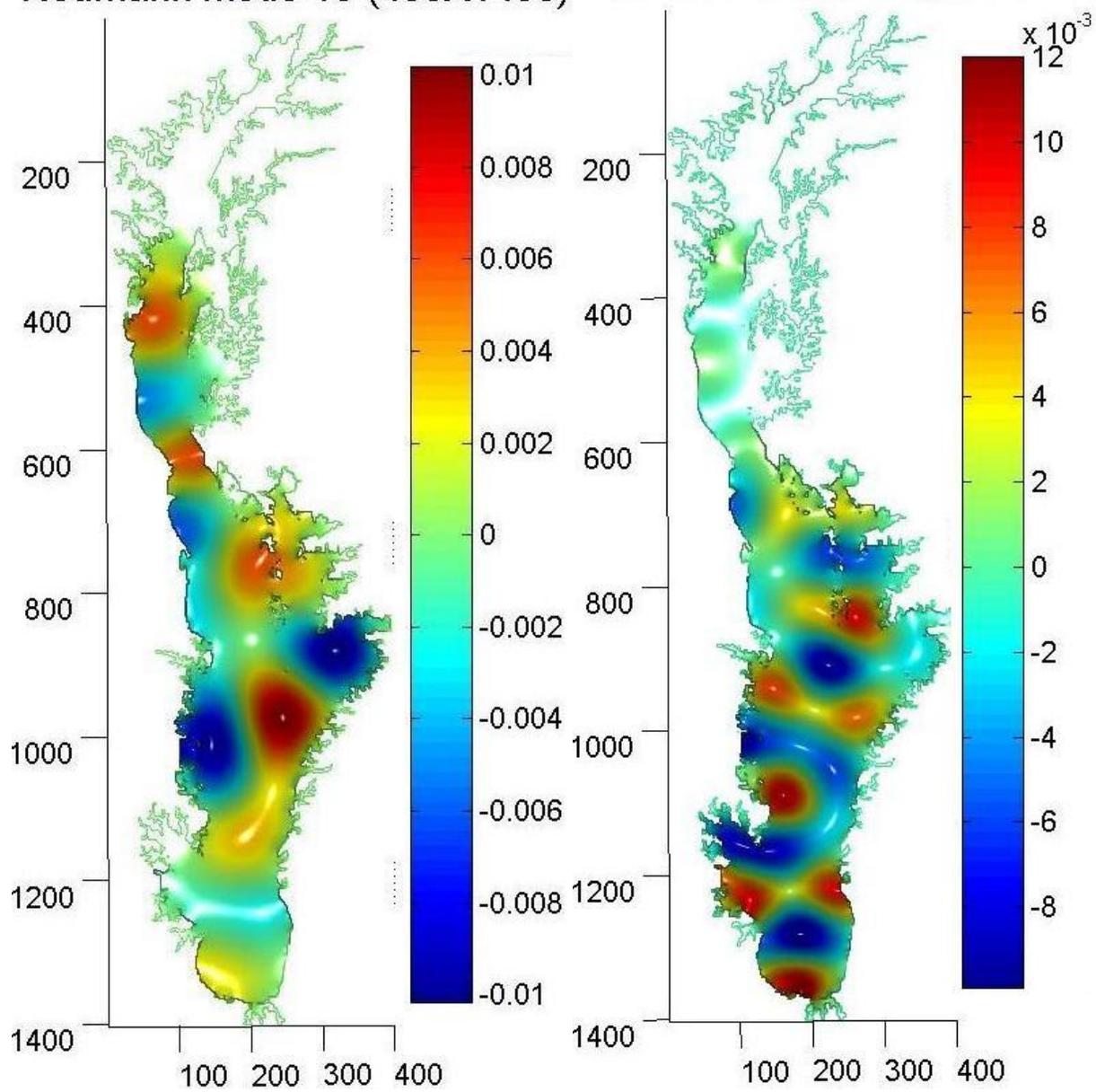


Figure 7.3

Figure 7.4

Neumann Mode 48 (400X1403) Neumann Mode 58 (400X1403)

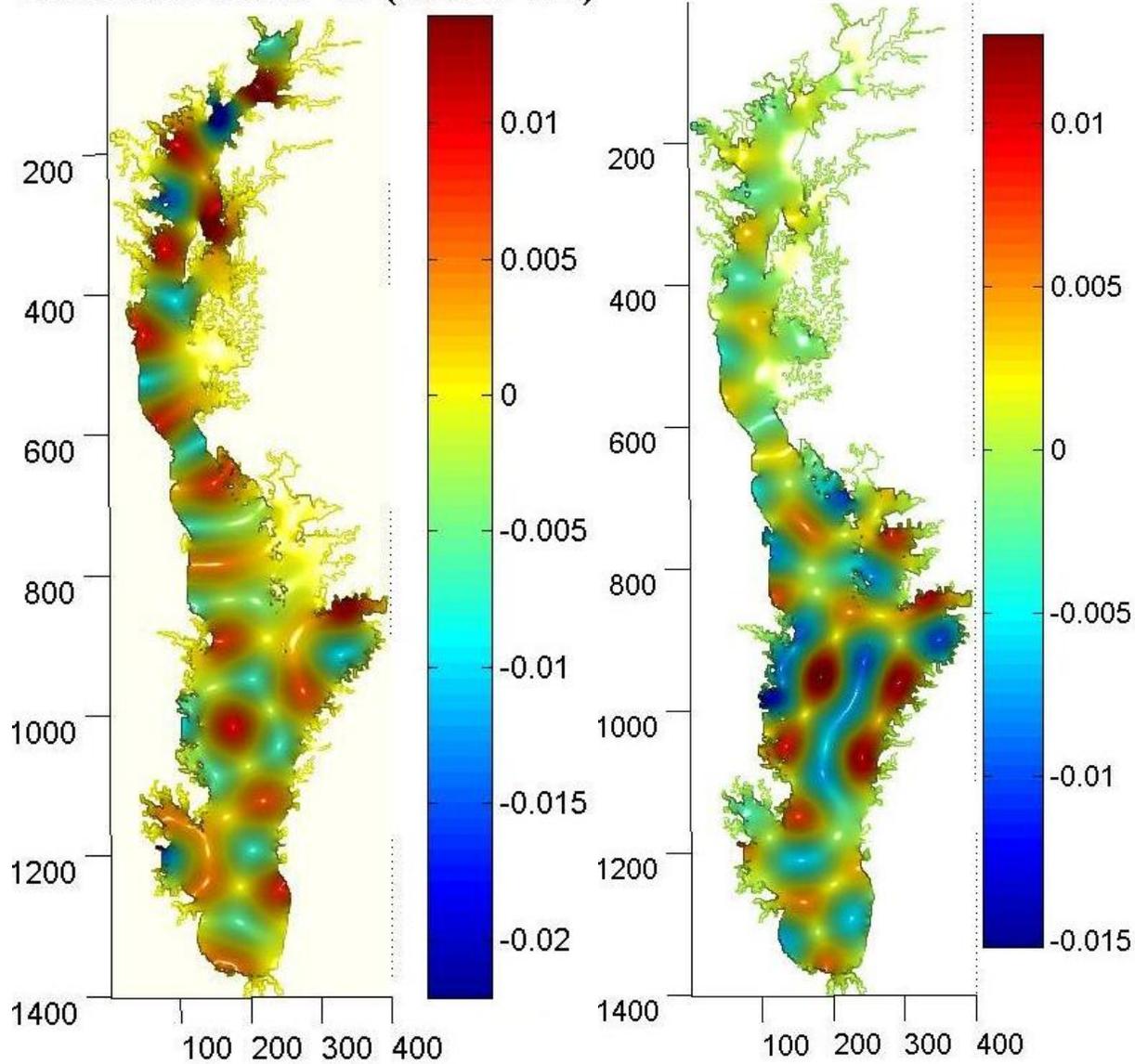


Figure 7.5

Figure 7.6

Neumann Mode 91 (400X1403) Neumann Mode 100 (400X1403)

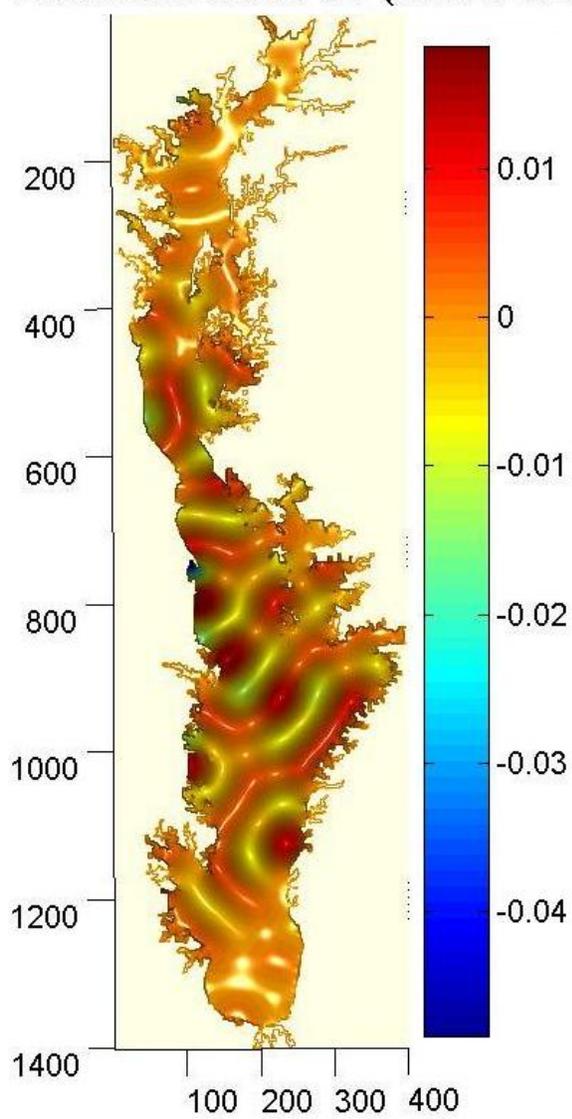


Figure 7.7

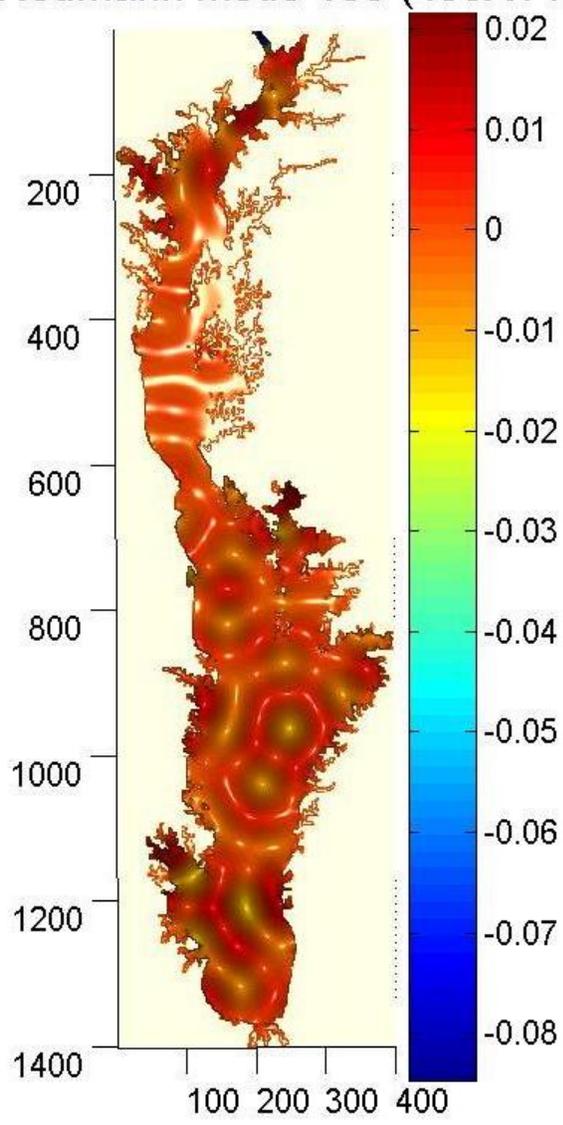


Figure 7.8

Neumann Mode 1 (350X1185)

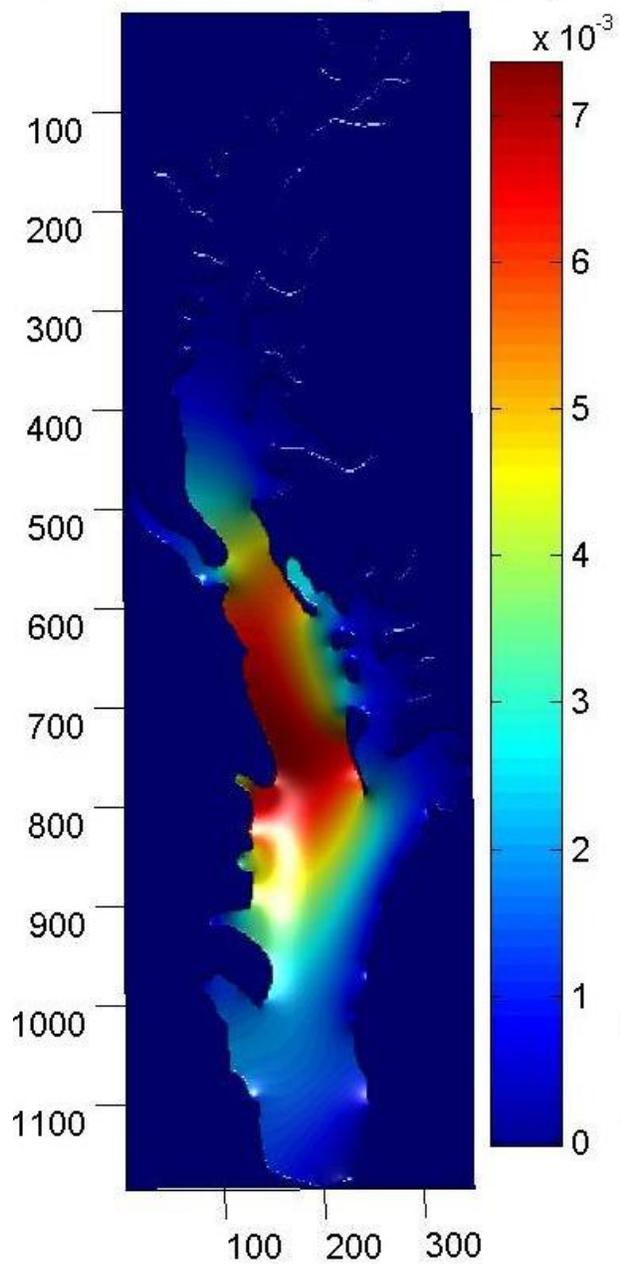


Figure 7.9

Neumann Mode 4 (350X1185)

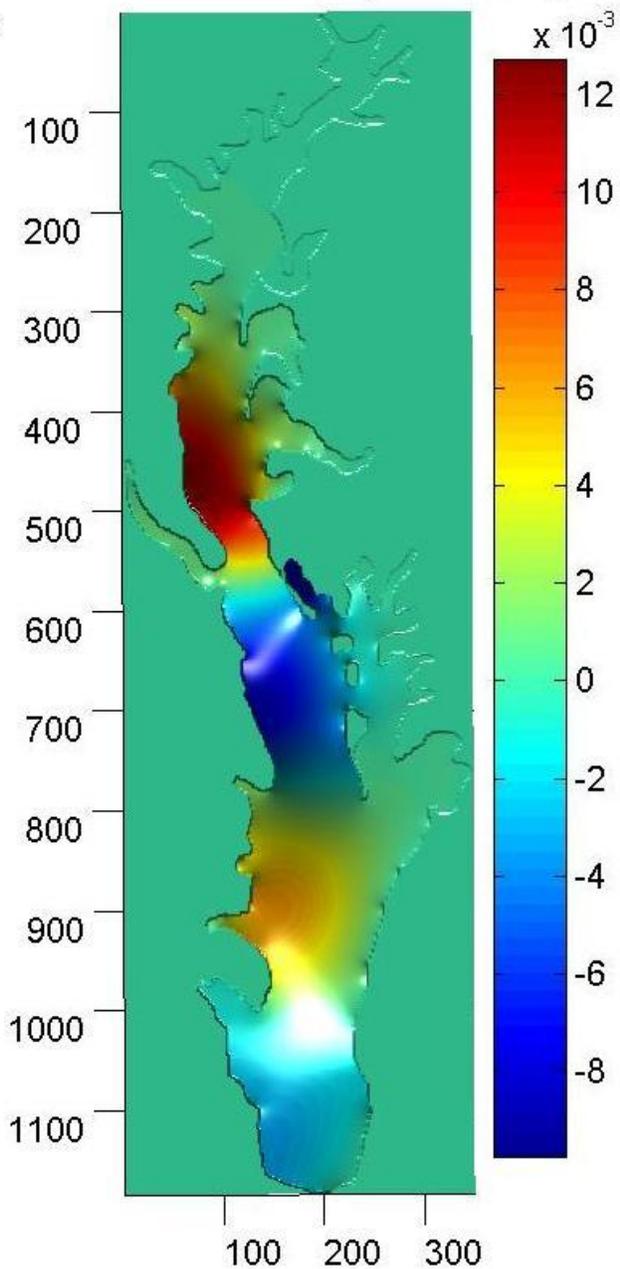
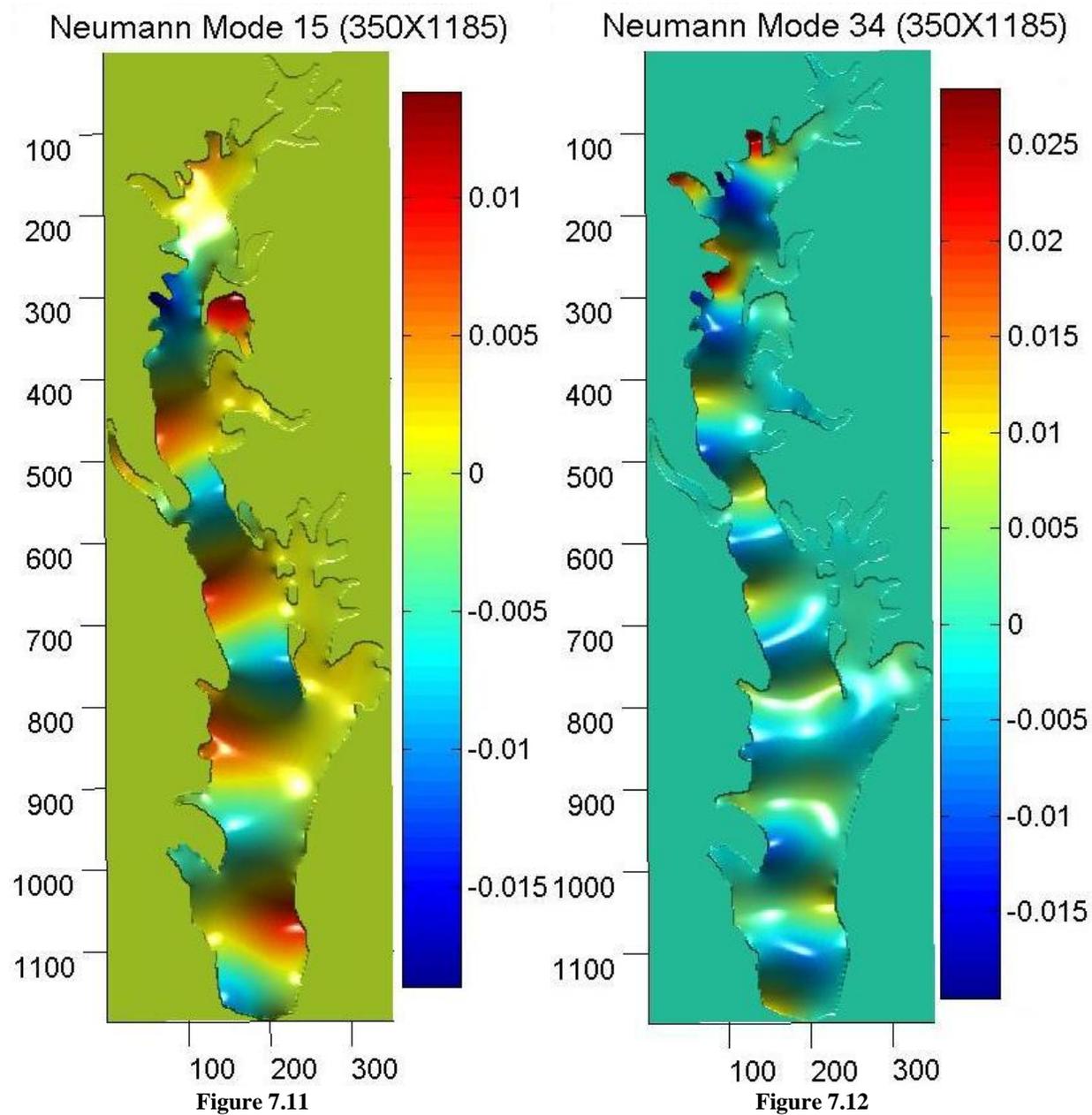


Figure 7.10



Neumann Mode 48 (350X1185) Neumann Mode 58 (350X1185)

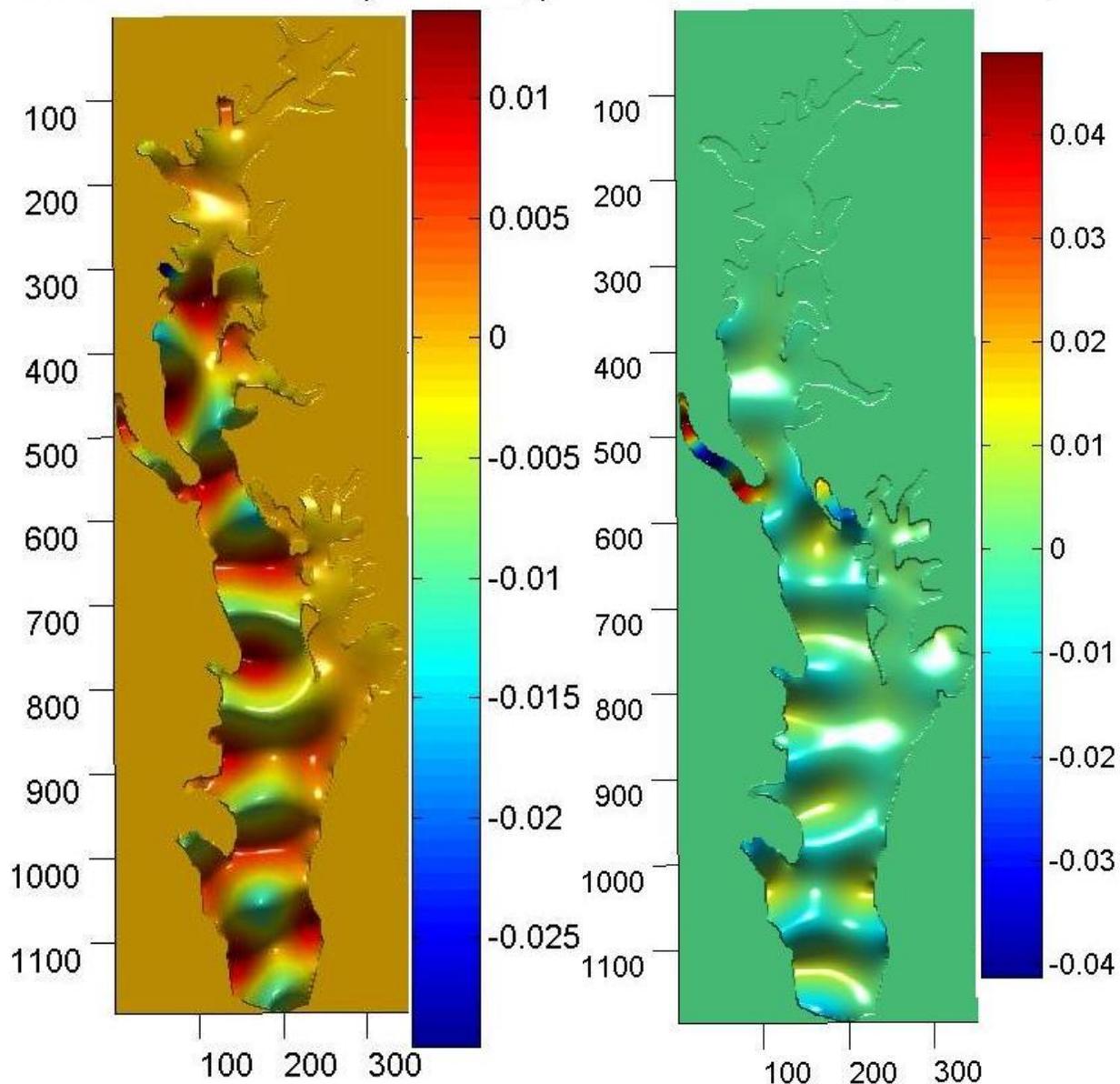


Figure 7.13

Figure 7.14

Neumann Mode 91 (350X1185)

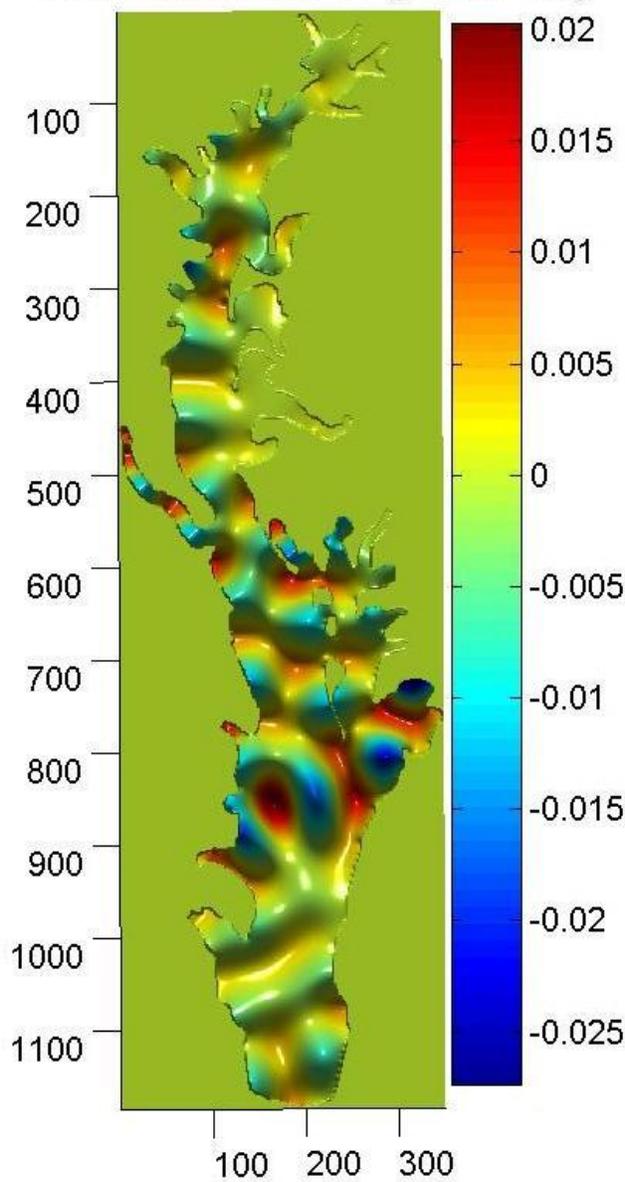


Figure 7.15

Neumann Mode 100 (350X1185)

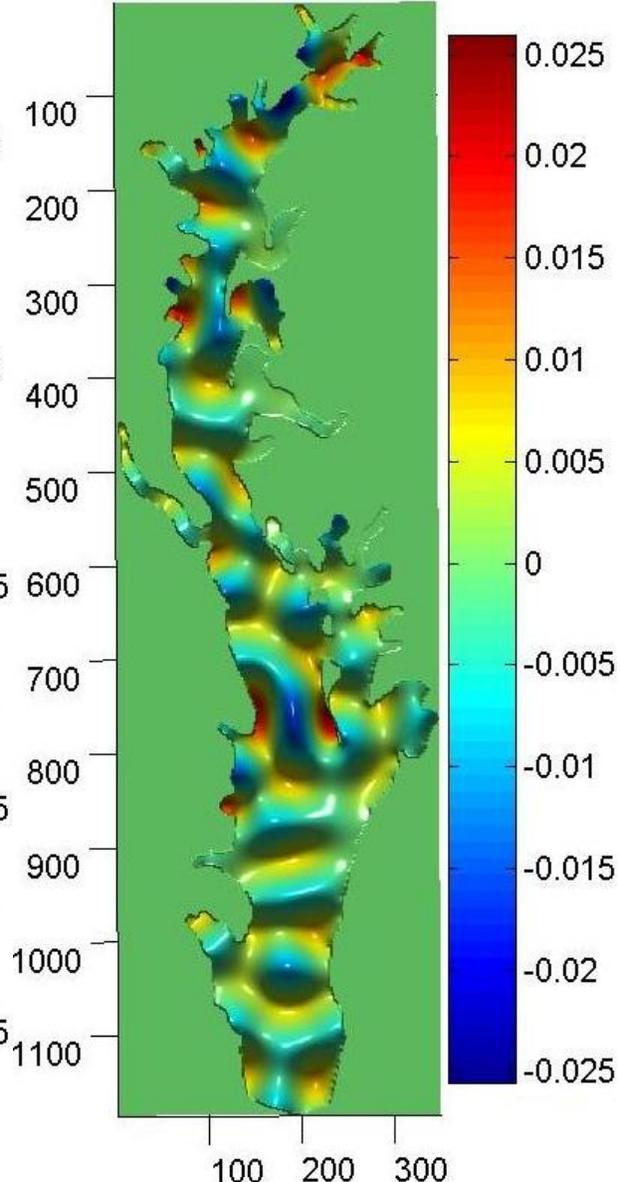


Figure 7.16

Chapter 8

Analysis of the Vorticity and Velocity Potentials of the Chesapeake Bay

Once the Neumann and Dirichlet eigenmodes for both of the Chesapeake boundaries were completed, an error analysis was done each of the solutions. For all the Chesapeake solutions there were no theoretical values with which to compare. Consequently, three different methods were used to examine the error of the eigenmodes. Just as in the Neumann equilateral triangle the convergence of the first eigenvalue was computed with respect to the highest resolution. For the Quoddy boundary the eigenvalues from the finite difference method were compared to the eigenvalues computed using FEMLAB®. In this graph the relative error with respect to FEMLAB®, was plotted for both the Dirichlet and Neumann boundary conditions. Because it was not input into FEMLAB®, a comparison between the eigenvalues for the finite difference solution on the high resolution boundary could not be completed. The final error test used was the orthogonality with respect to the first eigenvalue for each boundary and boundary condition.

The relative error of the first eigenvalue with respect to the highest resolution solution for both the Dirichlet and Neumann boundary conditions on the exact boundary for the Chesapeake

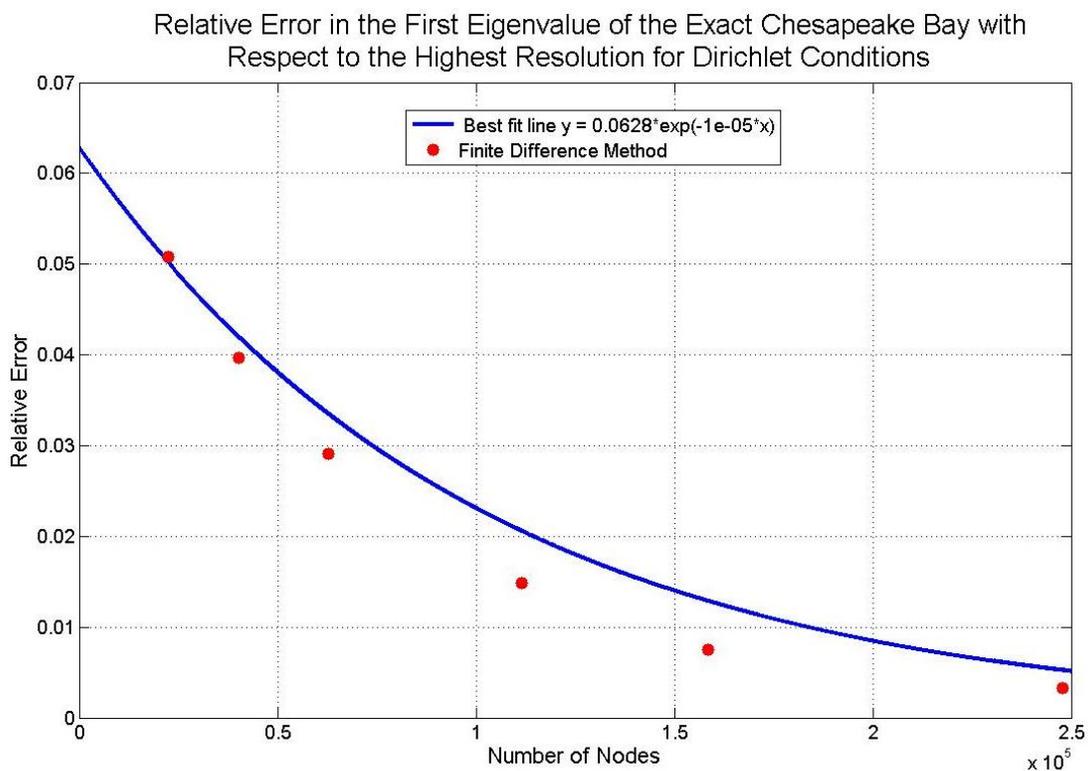


Figure 7.1: Relative error with respect to the highest resolution finite difference solution for Dirichlet boundary conditions.

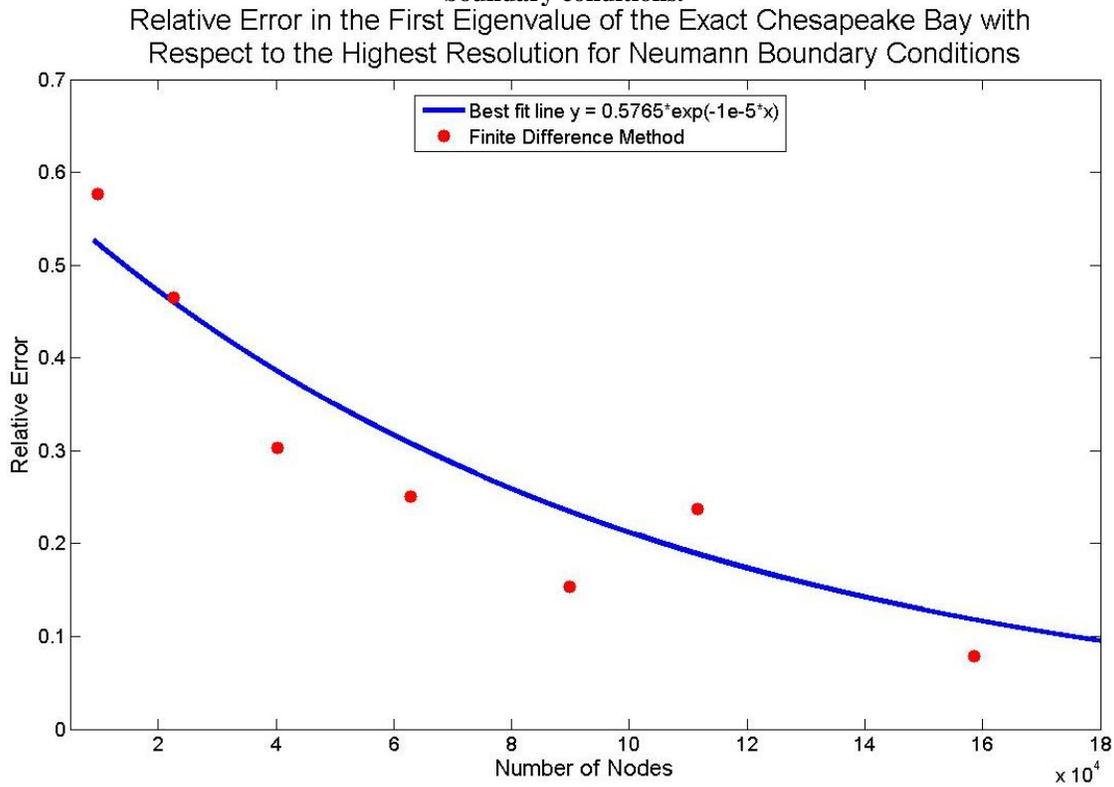


Figure 7.2: Relative error with respect to the highest resolution finite difference solution for Neumann boundary conditions.

Bay have been plotted in Figures 7.1 and 7.2 respectively. The best fit lines of the convergence plots in this section were all computed in Excel and then imported to MATLAB® to be plotted. Both plots showed convergence to the eigenvalue at the highest resolution. There was some significant oscillation about the best fit line, however, the fit showed a decently well behaved function converging as the grid became increasingly fine. The convergence for both types of boundary conditions changed significantly from the test geometries when applied to the Chesapeake Bay. The convergence no longer went as $1/(n-1)$, instead the convergence was exponentially decreasing. The term in the exponential was very small for both solutions at about 10^{-6} .

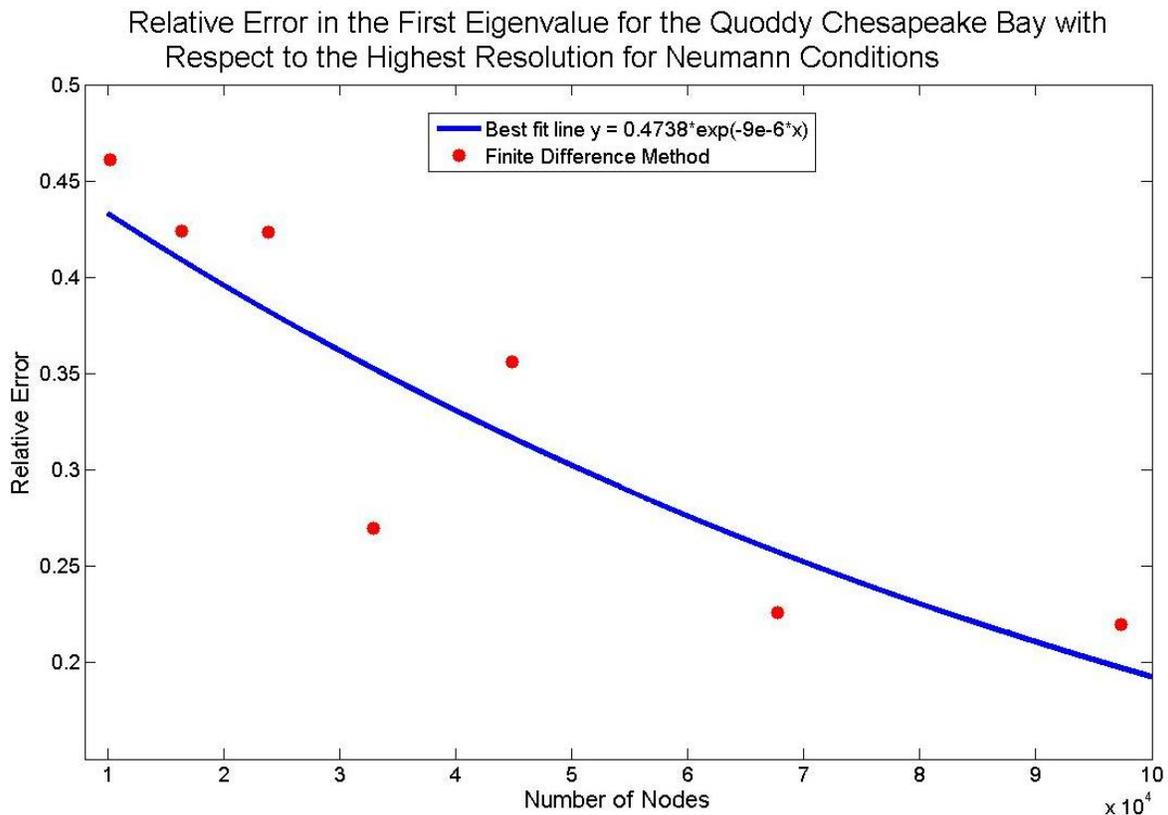
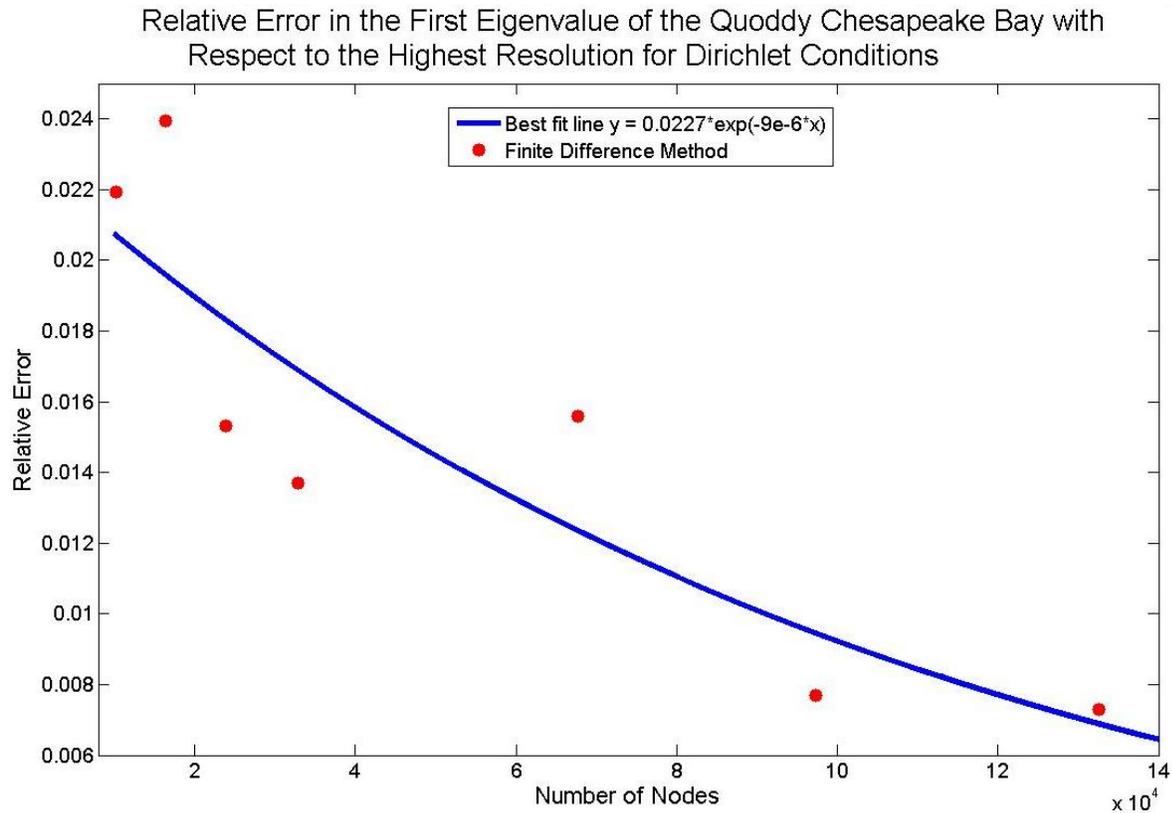


Figure 7.3: Relative error for Neumann boundary conditions versus the finite difference solution at the highest resolution on the Quoddy version of the Chesapeake Bay



Figures 7.4: Relative error for Dirichlet boundary conditions versus the finite difference solution at the highest resolution on the Quoddy version of the Chesapeake Bay

The convergence graphs for the Quoddy boundary for Dirichlet and Neumann boundary conditions have been plotted in Figures 7.3 and 7.4 respectively. In each of these plots there was one outlying eigenvalue removed. Both plots looked very similar to the graphs of convergence for the exact boundary. The magnitude of the exponent for the rate of convergence at -9×10^{-6} was exactly the same as convergence for the exact boundary with Dirichlet conditions. The overall relative error was consistent throughout both of the plots with the Dirichlet solutions having less relative error but converging at almost the same rate as the Neumann.

The next error analysis applied was a comparison between both the Dirichlet and Neumann solutions on the Quoddy boundary to the solution computed in FEMLAB®. Comparisons were computed for the lowest hundred eigenvalues of the highest resolution solution. This error analysis has been plotted in Figure 7.5. In this plot the eigenvalues of the

finite difference solution with Dirichlet boundary conditions was consistently three percent off of the FEMLAB® solution throughout almost the entire set of eigenvalues. The difference between the two solutions saw a drastic increase for the last seven eigenvalues in the Dirichlet solution. This posed two possibilities. One was that the finite difference solution failed at eigenvalue ninety-four and the other was that the FEMLAB® solution failed. Based on the test geometries it was probable that the FEMLAB® solution was failing. Looking at the convergence graphs versus theory for the three test geometries, while the accuracy of the eigenvalues for the FEMLAB® solution was in general an order of magnitude better it also fell off exponentially while the decrease in accuracy for the finite difference method was linear. The fact that the relative difference between the two solutions was three percent until the last few eigenvalues makes it much more likely that the error was from the FEMLAB® solution. This can be supported by the fact that the last eight eigenvalues for the FEMLAB® solution were 1608, 7224, 11676, 15100, 20274, 405899, 1142244 and 680962. The drastic change from 1608 to 680962 over the span of eight eigenvalues for the FEMLAB® solution makes the previous hypothesis very plausible. The relative error for the Neumann boundary conditions showed significant deviation throughout the entire span of eigenvalues. The FEMLAB® eigenvalues corresponded to the Neumann eigenvalues that had been present in the test solutions. It started with an eigenvalue of zero and then increased to 25.3 by the tenth eigenvalue with the first four eigenvalues below five. The small eigenvalues were expected because the bay occupied a rectangle that was larger in area than the unit square used for the test geometries. The increase in the size of the box containing the geometry also increased the area enough that the small separated partitions still produced eigenvalues smaller than those for the test geometries. The finite difference solution produced a

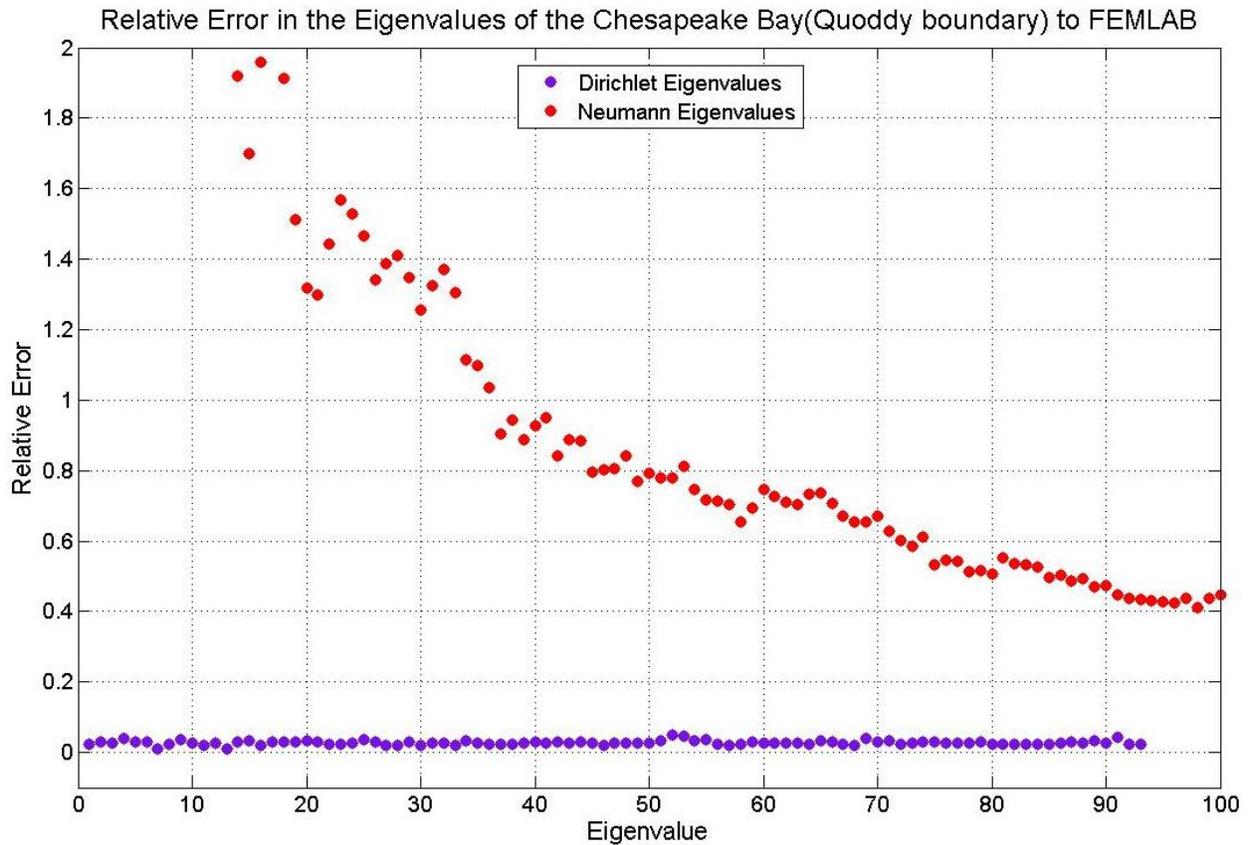


Figure 7.5: Relative error of the first hundred eigenvalues for Neumann and Dirichlet boundary conditions on the Quoddy boundary with respect to the FEMLAB® solution

first eigenvalue of 27.0. A constant potential solution should have been produced since a flat potential always has a gradient of zero. In order to test if the finite difference method was simply missing the first solutions the eigenvalues of the FEMLAB® were cut off at eigenvalue eleven which was approximately 27. The remaining eigenvalues were then compared with the finite difference solution. The first eigenvalue which had been matched showed a reasonably small error. However, after the first few eigenvalues the error increased significantly. There was not enough correlation between the shifted FEMLAB® solution and the finite difference solution to make the hypothesis plausible.

The last test applied to the Chesapeake solutions was orthogonality. The method and application were exactly the same as they explained and applied to the test geometries. For the solutions with Dirichlet boundary on both the Quoddy and exact boundaries the orthogonality

ranged between 10^{-15} and 10^{-17} . These orthogonality measures are exactly in line with the Dirichlet solutions for each of the test geometries. The orthogonality for the solutions with Neumann boundary conditions deviated significantly from the orthogonality computed for the test geometries. For the Quoddy boundary the orthogonality ranged from 10^{-2} to 10^{-7} . The orthogonality for the exact boundary with Neumann conditions ranged from 10^{-3} to 10^{-5} . For the test geometries certain modes exhibited orthogonality values within a few orders of magnitude of one. However, for the Chesapeake Bay this behavior completely took over. This behavior was likely tied to the approximation used for the normal derivatives at the corners. As the geometries became more complex the boundary gained an increasing number of corners to approximate the large number of curves. Since the approximation for the normal derivative at the corners was the solution for the square, as the geometry was perturbed farther from the square the approximation lost its validity and no longer produced correct eigenmodes.

The error analysis of the Chesapeake Bay told two very different stories. The modes with Dirichlet boundary conditions proved reliable through all of the error estimators. The eigenvalues converged, it compared well with the FEMLAB® solutions and the eigenvectors were nearly completely orthogonal. The Dirichlet solutions even proved more accurate for high eigenvalues than the FEMLAB® solution. This was not surprising considering the exponential increase in error over the one-hundred eigenvalues for the FEMLAB® solutions on the test geometries. The mode with Neumann boundary conditions on the other hand still appears to be an essentially open problem. The comparison with FEMLAB®, failure to find a constant potential solution and the lack of orthogonal behavior in the computed modes provided no evidence that the finite difference solutions were trustworthy. The FEMLAB® solution may

prove to be accurate. However, a much more thorough error analysis would have to be applied to the FEMLAB® solution before its eigenmodes could be validated.

Chapter 9

Inhomogenous Potentials of the Chesapeake Bay

The final mode in the normal mode analysis method was the inhomogenous mode. This mode takes into account flow through open boundaries in the geometry. For the inhomogenous mode solutions were only computed on the Quoddy geometry so that data from the Quoddy model could be used to produce the boundary conditions for the Bay. The data that was obtained for the Quoddy model was for 1999. The velocity fields were spaced in hour increments and began at 0000 01FEB1999. The inhomogenous modes were computed for fort-eight separate time slices from 0000 01FEB1999 until 0000 03FEB1999. Because the equation for the inhomogenous mode is a Poisson equation, only one mode was produced for each computation. This also means that the computed potentials are simply modes rather than eigenmodes. The normals for the open boundaries were taken from the lines used by FEMLAB® in its CAD drawing system to connect the nodes taken from the Quoddy boundary.

The computed potentials have been plotted in Figures 8.1-8.8. Two features tend to dominate all of the inhomogenous modes, the boundary with the Atlantic Ocean and the Potomac River. The boundary with the Atlantic Ocean in each of the modes plotted is the large negative dip in the potential at the Southeast corner of the Bay. The Potomac River is the large hump in

the potential on the Western section of the Bay. The flows from the other three rivers the Rappahannock, York and James were drowned out by the inflow from the Potomac and the outflow at the mouth of the Bay. The hour increments pictured below show the Bay through one change in the tides. As the potentials increase in time the flow through the mouth decreases until it reverses itself in the last plot. Interestingly, when the Atlantic begins to pour in through the mouth of the Bay the potential at the mouth of the Potomac flattens out. The flat potential indicates very little flow through the mouth of the Potomac River. The inflow from the Atlantic appears to have stopped up the Potomac River.

Source Mode
0700 01FEB1999(199X674)

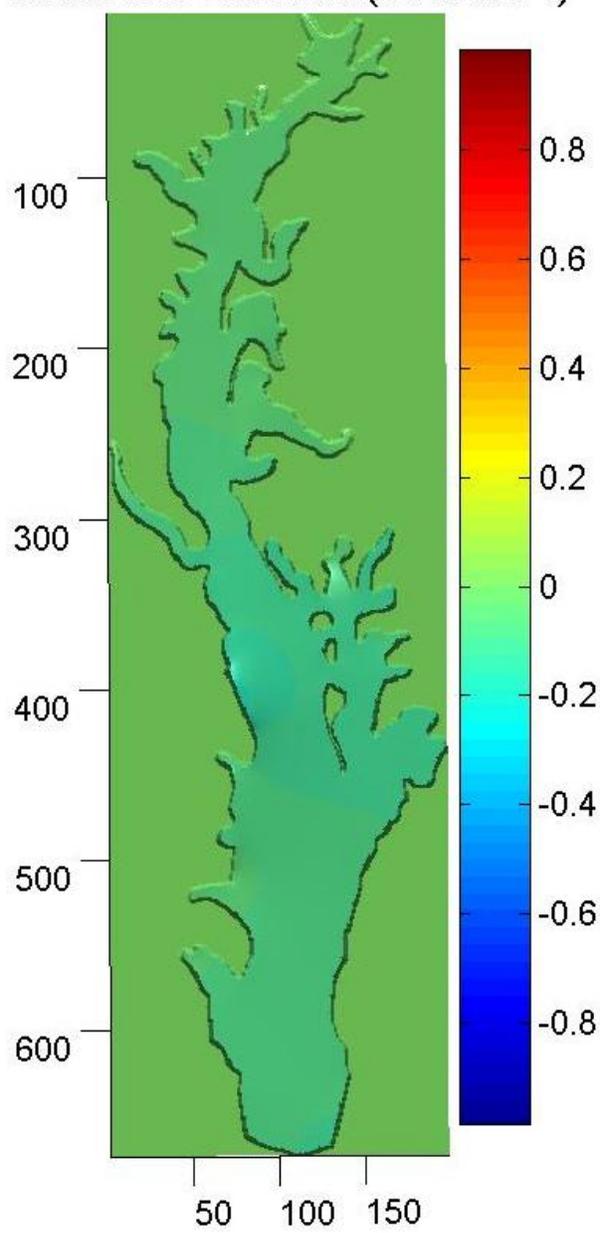


Figure 8.1

Source Mode
0800 01FEB1999(199X674)

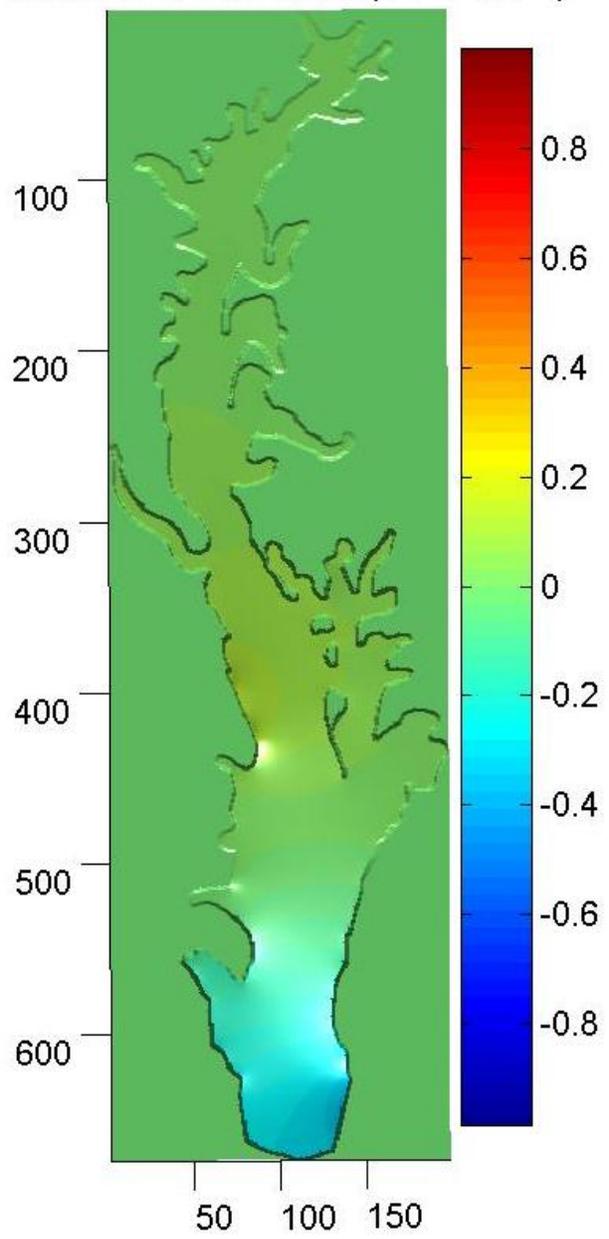


Figure 8.2

Source Mode
0900 01FEB1999(199X674)

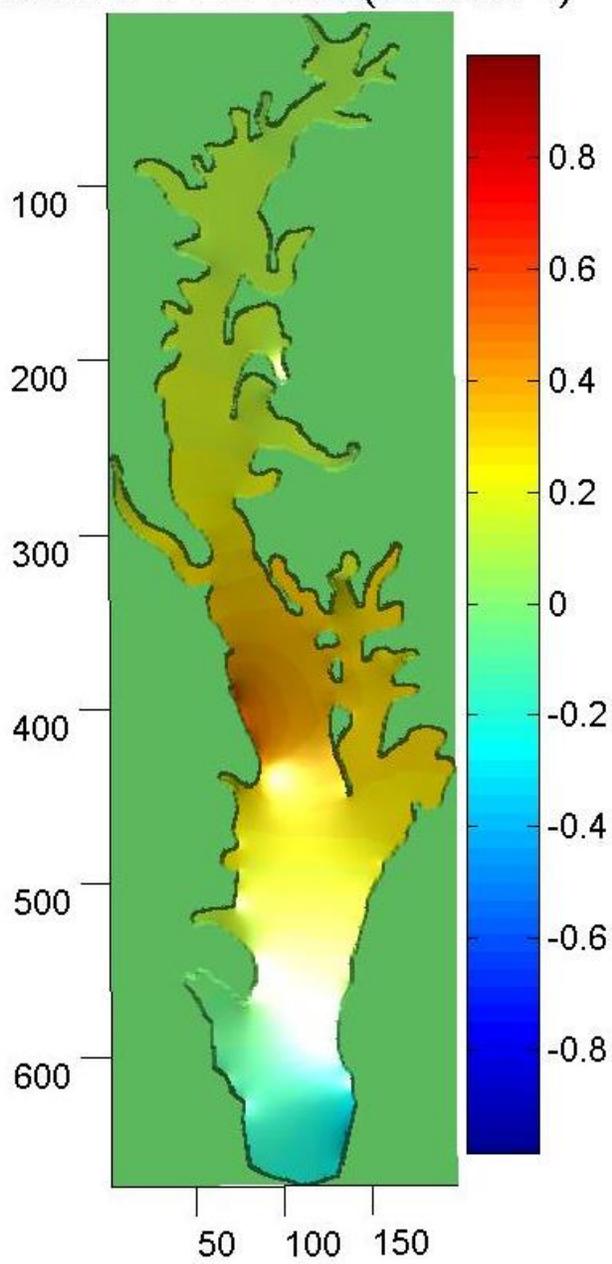


Figure 8.3

Source Mode
1000 01FEB1999(199X674)

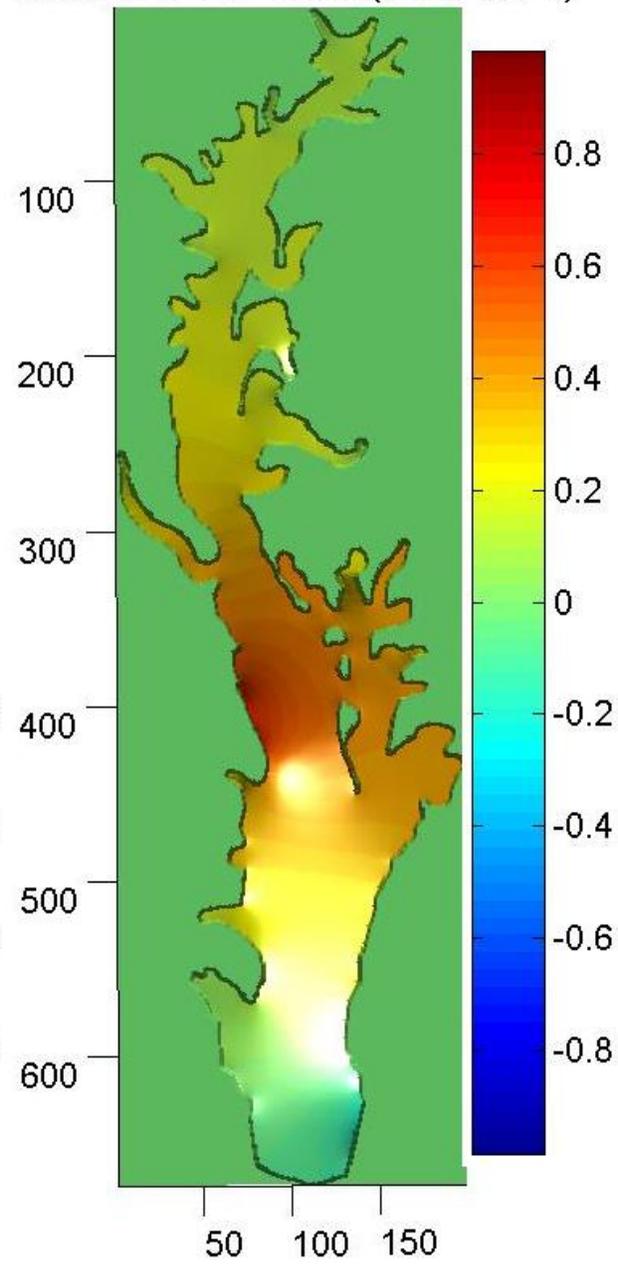


Figure 8.4

Source Mode
1100 01FEB1999(199X674)

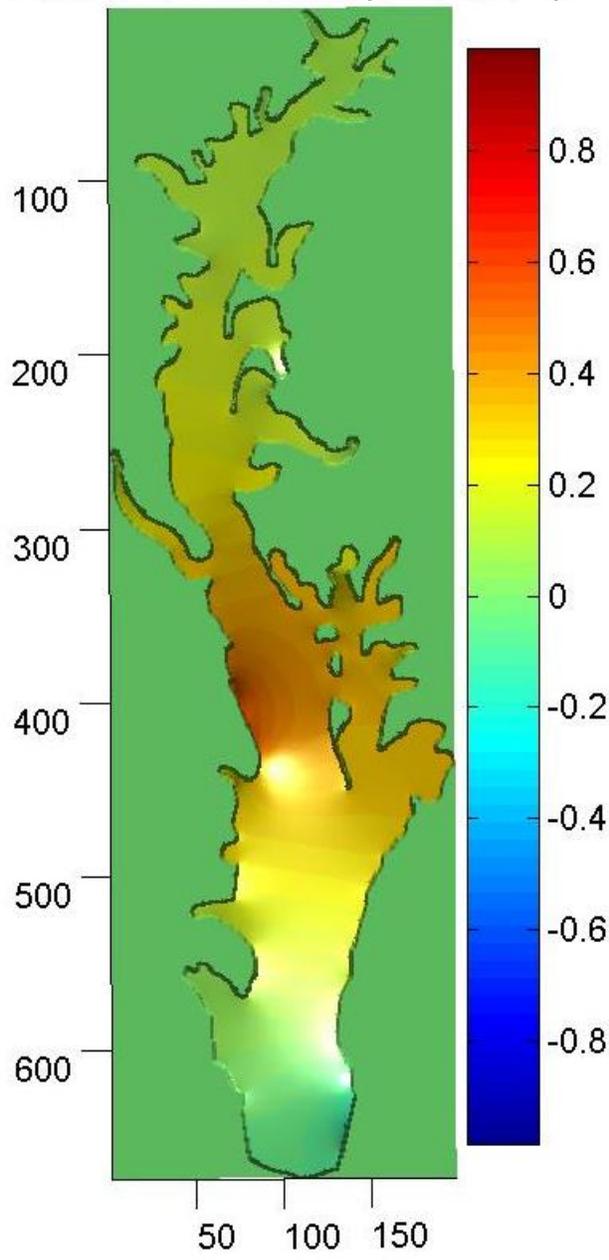


Figure 8.5

Source Mode
1200 01FEB1999(199X674)

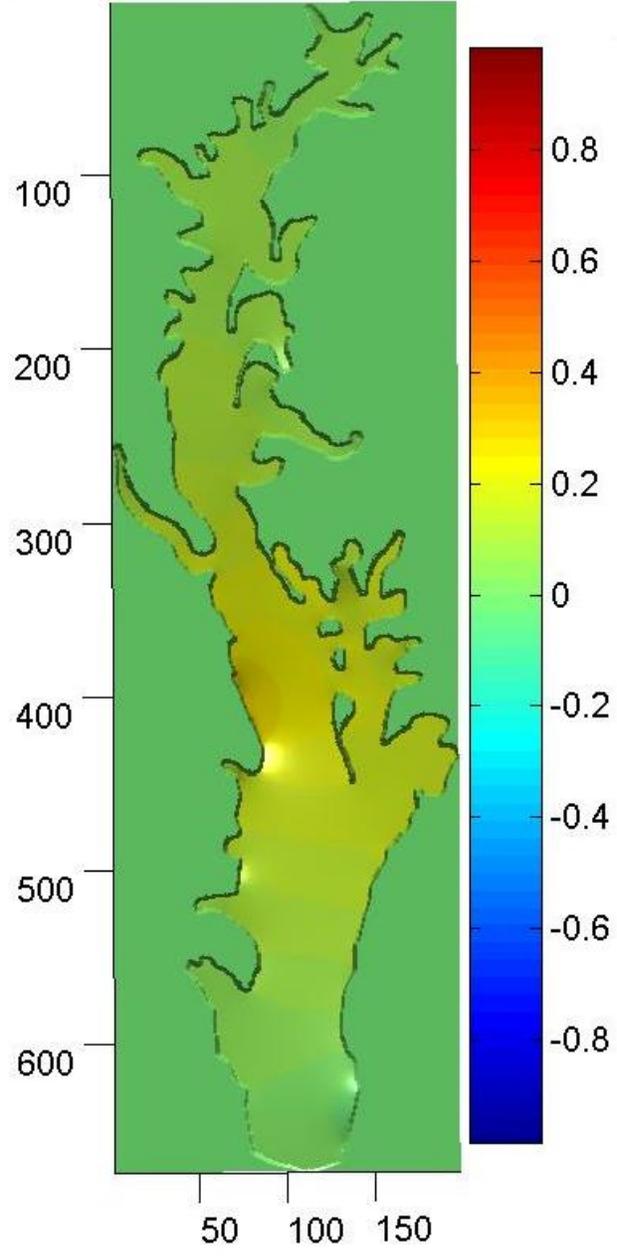


Figure 8.6

Source Mode
1300 01FEB1999(199X674)

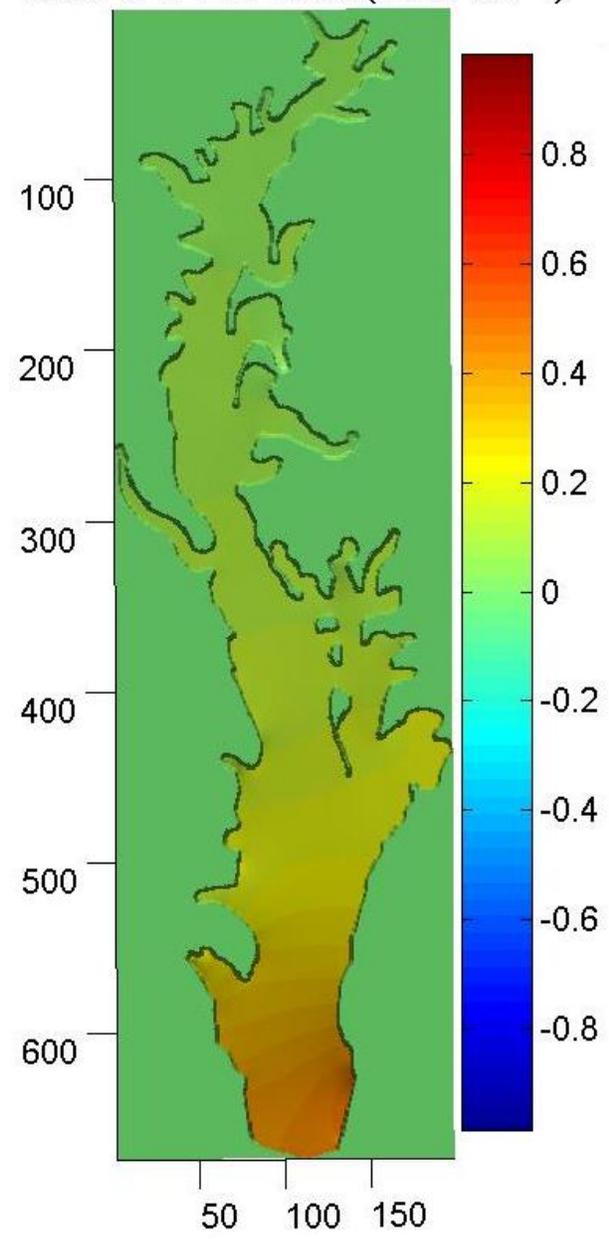


Figure 8.7

Source Mode
1400 01FEB1999(199X674)

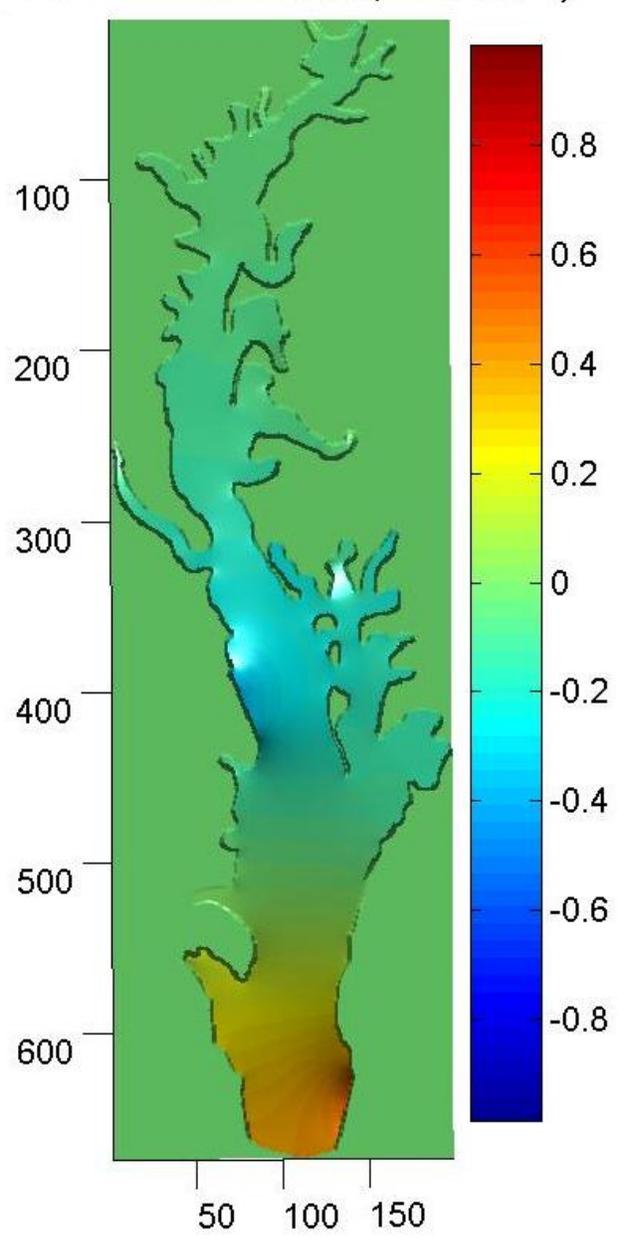


Figure 8.7

Chapter 10

Conclusion

In this project finite differencing methods were developed to solve the Helmholtz eigenvalues equation with both Dirichlet and Neumann boundary conditions. The general Dirichlet method, Ames's method and FEMLAB® were tested on three different shapes whose solutions are well known: the square, circle and equilateral triangle. The same computations were done for the Neumann boundary conditions. The validity of the solutions was tested using both theory and other error indicators. The convergence of the first eigenvalue for each of these geometries was graphed and then fitted with a curve based on its rate of convergence, the eigenvalues of the highest resolution runs were compared with the solutions from FEMLAB® and the orthogonality of the eigenmodes themselves was computed. The solutions for the test geometries complied with the theoretical predictions and the other error indicators. The solutions for Neumann boundary conditions converged more slowly, had more random behavior and the orthogonality of a large number of eigenmodes was much greater than that for the Dirichlet solutions. Once the methods had been tested, the general Dirichlet and Neumann methods were applied to two versions of the Chesapeake Bay. One hundred modes were calculated for each of the versions at a number of different resolutions. Both the vorticity

potentials and the velocity potentials covered the entirety of the Bay with at least ten modes. Similar error analysis was applied to the Bay as was applied to the test geometries. The error analysis showed that the Dirichlet finite difference method provided trustworthy solutions along a large number of eigenvalues. The Neumann finite difference method was not validated by the error analysis. There were significant deviations from the FEMLAB® solution and the orthogonal behavior of the eigenmodes underwent a large decrease. The likely cause of this error was the approximation of the normal derivative at the corners. While time was spent researching alternative methods for approximating the normal derivative at the corners, no satisfactory method was completed. The FEMLAB® solution to the Chesapeake Bay for Neumann boundary conditions may still be valid and could possibly be combined with the finite difference method with Dirichlet boundary conditions if more error analysis were focused on FEMLAB®'s solution. The last computations completed were for the inhomogenous mode using model data from Quoddy for forty-eight different time slices. These computations showed that a significant amount of the forcing function for the Bay's modes were concentrated around the mouth of the Bay and the Potomac River. Error analysis was not completed for the inhomogenous mode. For future analysis, a possible error metric for the inhomogenous mode is the degree to which the modes comply with the boundary conditions. This would require computation of the vector field from the potential. The boundary of the vector field would then be dotted with calculated normals along the boundary and compared to the actual boundary conditions used.

Endnotes

¹ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3425.

² Ereemeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., O.V. Melnichenko, S.V. Kochergin, and R.R. Stanichnaya, *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 2, Characteristics of the large-scale circulation of the Black Sea*, (*J. Geophys. Res.*, 97, 9743-9753, 1992b), 9733.

³ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3425.

⁴ Ereemeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., O.V. Melnichenko, S.V. Kochergin, and R.R. Stanichnaya, *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 2, Characteristics of the large-scale circulation of the Black Sea*, (*J. Geophys. Res.*, 97, 9743-9753, 1992b), 9733.

⁵ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3428

⁶ Ereemeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., O.V. Melnichenko, S.V. Kochergin, and R.R. Stanichnaya, *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 2, Characteristics of the large-scale circulation of the Black Sea*, (*J. Geophys. Res.*, 97, 9743-9753, 1992b), 9747.

⁷ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3428.

⁸ Ereemeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., O.V. Melnichenko, S.V. Kochergin, and R.R. Stanichnaya, *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 2, Characteristics of the large-scale circulation of the Black Sea*, (*J. Geophys. Res.*, 97, 9743-9753, 1992b), 9744.

⁹ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3428.

¹⁰ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3432.

¹¹ Chesapeake Research Consortium. Chesapeake Bay Oyster Restoration Consensus of a Meeting of Scientific Experts Virginia Institute of Marine Science Wachapreague, Virginia. June 1999. 20 Jan. 2004. <http://www.vims.edu/vimsnews/CBOysRestor.pdf>.

¹² Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3425.

¹³ Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods*, (*J. Geophys. Res.*, 97, 9733-9742, 1992a), 9733.

¹⁴ Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods*, (*J. Geophys. Res.*, 97, 9733-9742, 1992a), 9733.

¹⁵ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3425.

¹⁶ Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods*, (*J. Geophys. Res.*, 97, 9733-9742, 1992a), 9736.

¹⁷ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3428.

¹⁸ Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods*, (*J. Geophys. Res.*, 97, 9733-9742, 1992a), 9736.

¹⁹ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3441.

²⁰ Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods*, (*J. Geophys. Res.*, 97, 9733-9742, 1992a), 9737.

²¹ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3428.

²² Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., O.V. Melnichenko, S.V. Kochergin, and R.R. Stanichnaya, *Reconstruction of oceanic flow characteristics from quasi-Lagrangian*

data, 2, *Characteristics of the large-scale circulation of the Black Sea*, (*J. Geophys. Res.*, 97, 9743-9753, 1992b), 9744.

²³ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3428.

²⁴ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3429.

²⁵ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3429.

²⁶ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3430.

²⁷ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3429.

²⁸ Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods*, (*J. Geophys. Res.*, 97, 9733-9742, 1992a), 9733.

²⁹ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3425.

³⁰ Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods*, (*J. Geophys. Res.*, 97, 9733-9742, 1992a), 9736.

³¹ Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis*, (*J. Geophys. Res.*, 105, 3425-3450, 2000), 3430.

³² Ames, William F. Numerical Methods for Partial Differential Equations. (New York: Barnes & Noble, Inc., 1969), 30.

³³ Abramowitz, Milton and Irene A. Stegun ed. Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables. New York: Dover, 1964.

³⁴ McCartin, Brian J., Eigenstructure of the Equilateral Triangle, Part I: The Dirichlet Problem, SIAM Review, 45, 267-287, 2003.

³⁵ Chesapeake Bay Program. Chesapeake Bay Bathymetry. March 2002. 10 Oct. 2004.
<http://www.chesapeakebay.net/pubs/maps/2003-138.pdf>

Bibliography

- Abramowitz, Milton and Irene A. Stegun ed. Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables. New York: Dover, 1964.
- Ames, William F. Numerical Methods for Partial Differential Equations. New York: Barnes & Noble, Inc., 1969.
- Chesapeake Bay Program. Chesapeake Bay Bathymetry. March 2002. 10 Oct. 2004. <http://www.chesapeakebay.net/pubs/maps/2003-138.pdf>
- Chesapeake Research Consortium. Chesapeake Bay Oyster Restoration Consensus of a Meeting of Scientific Experts Virginia Institute of Marine Science Wachapreague, Virginia. June 1999. 20 Jan. 2004. <http://www.vims.edu/vimsnews/CBOysRestor.pdf>.
- Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 1, Approach and mathematical methods, J. Geophys. Res.*, 97, 9733-9742, 1992a.
- Eremeev, V.N., L.M. Ivanov, A.D. Kirwan Jr., O.V. Melnichenko, S.V. Kochergin, and R.R. Stanichnaya, *Reconstruction of oceanic flow characteristics from quasi-Lagrangian data, 2, Characteristics of the large-scale circulation of the Black Sea, J. Geophys. Res.*, 97, 9743-9753, 1992b.
- Farlow, Stanley J. Partial Differential Equations for Scientists and Engineers. New York: Dover, 1993.
- Gottlieb, David and S.A. Orszag. Numerical Analysis of Spectral Methods: Theory and Applications. Philadelphia: Society for Industrial and Applied Mathematics, 1997.
- Higdon, R.L., "Modeling the global climate system on high-performance computers," SIAM News, 33.1 (Jan. 2000). <http://www.siam.org/siamnews>.
- Jones, C.K.R.T., "A dynamics group looks at the ocean," SIAM News, 33.2 (Mar. 2000). <http://www.siam.org/siamnews>.
- Lewis, J.K., I. Shulman, and A.F. Blumberg, *Assimilation of doppler radar current data into numerical ocean models, Cont. Shelf Res.*, 18, 541-559, 1998.
- Lipphardt, B.L., A.D. Kirwan Jr., C.E. Grosch, J.K. Lewis, and J.D. Paduan, *Blending HF radar and model velocities in Monterey Bay through normal mode analysis, J. Geophys. Res.*, 105, 3425-3450, 2000.

- Lipphardt, B.L. Preliminary Research Results. 20 Jan. 2004.
<http://newark.cms.udel.edu/~brucel/prelimresults.html>.
- Lynch, D. and Werner F. *Three-Dimensional Hydrodynamics on Finite Elements Part II: Non-Linear Time-Stepping Model*. Inter. Jour. Numer. Meth Fluids, 12, 507-533, 1991.
- Malek-Madani, Reza. Advanced engineering mathematics with Mathematica and MATLAB®. Reading: Addison Wesley, 1998.
- McCartin, Brian J., *Eigenstructure of the Equilateral Triangle, Part I: The Dirichlet Problem*, SIAM Review, 45, 267-287, 2003.
- Trefethen, Lloyd N. Spectral Methods in MATLAB®. Philadelphia: SIAM, 2000.
- Wilson , Nancy. Map of the Chesapeake Bay. 26 Apr. 1999. 20 Jan. 2004.
<http://www.vims.edu/physical/projects/organic/themap.htm>.

Appendix A

MATLAB® Code

The Following programs solve for Stream Functions, the solutions to the eigenvalue equation with Dirichlet boundary conditions.

```
%{
This program computes the differentiation matrix with Dirichlet
boundary conditions. It requires the input of "bay" which is a matrix
containing the square grid approximation of the geometry using ones
and zeros. The eigenvalues and eigenvectors of the differentiation
matrix are computed using eigs, the sparse eigensolver in MATLAB®.
%}

[length width] = size(bay);
nodes = width * length;
perpdelta = -((width-1))^2;
invbay = (bay ./ bay) - 1;
[I,J] = find(invbay);
Z = I + (J - 1) .* length;
gapvector = int32(Z);
vect = ones(nodes,1);
A = perpdelta .* spdiags([vect vect -4*vect vect vect], ...
    [-length -1 0 1 length ],nodes, nodes);
A(gapvector,:) = [];
A(:,gapvector) = [];
[eigenvector,eigenvalues] = eigs(A,[],100,'SM');
lambda = diag(eigenvalues);
lamb = sort(lambda);
plot(lamb, 'g.')
```

```

%{
This program calculates the differentiation matrix for Ames's formulation
of the Dirichlet boundary condition. The program "circledistance" is used
to calculate the movex and movey matrices which contain the shift of each
boundary node proportional to h. The bay, movex and movey variables come
from "circledistance."
%}

clear all
clkbegin = clock;

circledistance

[length width] = size(bay);
size = width * length;
perpdelta = -(width-1)^2;
numeigvals = 100;

%{
Toetemplate is a vector used as a template for each row of the
differentiation matrix. The five non-zero values correspond to the five
point molecule used in the finite differencing method.
%}
firstzeros = length - 2;
secondzeros = size - firstzeros - 3;
zerone(1:firstzeros) = 0;
zertwo(1:secondzeros) = 0;
toetemplate = [zertwo 1 zerone 1 -4 1 zerone 1 zertwo];
toetemplate = int8(toetemplate);
%get correct size of matrix and position of gaps for specific geometry
invbay = (bayone ./ bayone) - 1;
[I,J] = find(invbay);
Z = I + (J - 1) .* length;
gapvector = int32(Z);
%variables used in the calculation of the differentiation matrix. "A" is
%the differentiation matrix.
A = [];
baycounter = 1;
intcounter = 1;
nullcounter = 0;
toeplace = size;
%This section creates the differentiation matrix from bay, movex and movey.
while intcounter <= size
switch double(bay(intcounter))
case 0

```

```

    nullcounter = nullcounter + 1;
case 1
    if baytwo(intcounter + 1) == 1 & baytwo(intcounter - 1) == 1 & baytwo(intcounter + width)
    == 1 & baytwo(intcounter - width) == 1
        B = double(toetemplate(toeplace:(toeplace + size - 1)));
    elseif baytwo(intcounter) == 1
        B(1:size) = 0;
        if baytwo(intcounter + 1) == 2 & baytwo(intcounter - 1) == 2
            yshiftone = movey(intcounter + 1);
            yshifftwo = movey(intcounter - 1);
            B(intcounter + 1) = B(intcounter + 1) + 2 / (yshiftone * (yshiftone + yshifftwo));
            B(intcounter - 1) = B(intcounter - 1) + 2 / (yshifftwo * (yshiftone + yshifftwo));
            B(intcounter) = B(intcounter) - 2 / (yshiftone * yshifftwo);
        elseif baytwo(intcounter + 1) == 2

            yshiftone = movey(intcounter + 1);
            B(intcounter + 1) = B(intcounter + 1) + 2 / (yshiftone * (yshiftone + 1));
            B(intcounter - 1) = B(intcounter - 1) + 2 / (yshiftone + 1);
            B(intcounter) = B(intcounter) - 2 / yshiftone;
        elseif baytwo(intcounter - 1) == 2
            yshifftwo = movey(intcounter - 1);
            B(intcounter + 1) = B(intcounter + 1) + 2 / (yshifftwo + 1);
            B(intcounter - 1) = B(intcounter - 1) + 2 / (yshifftwo * (yshifftwo + 1));
            B(intcounter) = B(intcounter) - 2 / yshifftwo;
        elseif baytwo(intcounter + 1) == 1 & baytwo(intcounter - 1) == 1
            B(intcounter + 1) = B(intcounter + 1) + 1;
            B(intcounter - 1) = B(intcounter - 1) + 1;
            B(intcounter) = B(intcounter) - 2;
        end
    if baytwo(intcounter + width) == 2 & baytwo(intcounter - width) == 2
        xshiftone = movex(intcounter + width);
        xshifftwo = movex(intcounter - width);
        B(intcounter + width) = B(intcounter + width) + 2 / (xshiftone * (xshiftone + xshifftwo));
        B(intcounter - width) = B(intcounter - width) + 2 / (xshifftwo * (xshiftone + xshifftwo));
        B(intcounter) = B(intcounter) - 2 / (xshiftone * xshifftwo);
    elseif baytwo(intcounter + width) == 2
        xshiftone = movex(intcounter + width);
        B(intcounter + width) = B(intcounter + width) + 2 / (xshiftone * (xshiftone + 1));
        B(intcounter - width) = B(intcounter - width) + 2 / (xshiftone + 1);
        B(intcounter) = B(intcounter) - 2 / xshiftone;
    elseif baytwo(intcounter - width) == 2
        xshifftwo = movex(intcounter - width);
        B(intcounter + width) = B(intcounter + width) + 2 / (xshifftwo + 1);
        B(intcounter - width) = B(intcounter - width) + 2 / (xshifftwo * (xshifftwo + 1));
        B(intcounter) = B(intcounter) - 2 / xshifftwo;
    elseif baytwo(intcounter + width) == 1 & baytwo(intcounter - width) == 1

```

```

        B(intcounter + width) = B(intcounter + width) + 1;
        B(intcounter - width) = B(intcounter - width) + 1;
        B(intcounter)      = B(intcounter) - 2;
    end
end
B(gapvector) = [];
B = B .* perpdelta;
C = sparse(B);
A = [A;C];
clear B;
otherwise
    error('Case not valid.');
```

```

end
    intcounter = intcounter + 1;
    toeplace = toeplace - 1;
end
clkend = clock;
time1 = clkend - clkbegin;
save tmpfile size width length time1 A numeigvals bay
clear all
load tmpfile
clkbegin = clock;
%Calculate the eigenvectors and eigenvalues of the differentiation matrix
%"A". The smallest one hundred eigenvalues and their corresponding
%eigenvectors are chosen using numeigvals defined above.
[eigenvector,eigenvalues] = eigs(A,[],numeigvals,'SM');
lambda = diag(eigenvalues);
lamb = sort(lambda);
subplot(2,1,1)
plot(lamb, 'g.')
clkend = clock;
time2 = clkend - clkbegin
```

The following programs compute the velocity potentials, solutions to the eigenvalue equation with Neumann boundary conditions.

```

%This program uses only uses convex corners. The inner nodes which do not
%connect to any nodes outside of the geometry have been changed to inner
%nodes. This program computes the differentiation matrix for Neumann
%boundary conditions where the normal derivative is approximated
%as the corner of a square.
clkbegin = clock;
%how many eigenvalues to compute with eigs
```

```

numeigvals = 100;
[length width] = size(bay);
nodes = width * length;
troop = 1;
while troop <= length
    trop = 1;
    while trop <= width
        if bay(troop, trop) < .9
            bay(troop, trop) = 0;
        else
            bay(troop, trop) = 1;
        end
        trop = trop + 1;
    end
    troop = troop + 1;
end
%Perpdelta = -1/h^2 where h is the distance between nodes
perpdelta = -(width-1)^2;
%this program will add Neumann BC's to a shape that is set up for Dirichlet
bay = findingtheboundary_circ(bay);
%create Toeplitz template
firstzeros = length - 2;
secondzeros = nodes - firstzeros - 3;
zerone(1:firstzeros) = 0;
zertwo(1:secondzeros) = 0;
toetemplate = [zertwo 1 zerone 1 -4 1 zerone 1 zertwo];
toetemplate1 = [zertwo 1 zerone 0 -4 2 zerone 1 zertwo];
toetemplate2 = [zertwo 1 zerone 2 -4 0 zerone 1 zertwo];
toetemplate3 = [zertwo 0 zerone 1 -4 1 zerone 2 zertwo];
toetemplate4 = [zertwo 2 zerone 1 -4 1 zerone 0 zertwo];
toetemplate5 = [zertwo 0 zerone 0 -4 2 zerone 2 zertwo];
toetemplate6 = [zertwo 2 zerone 0 -4 2 zerone 0 zertwo];
toetemplate7 = [zertwo 0 zerone 2 -4 0 zerone 2 zertwo];
toetemplate8 = [zertwo 2 zerone 2 -4 0 zerone 0 zertwo];
toetemplate9 = [zertwo 2 zerone 0 -4 0 zerone 0 zertwo];
toetemplate10 = [zertwo 0 zerone 0 -4 0 zerone 2 zertwo];
toetemplate11 = [zertwo 0 zerone 2 -4 0 zerone 0 zertwo];
toetemplate12 = [zertwo 0 zerone 0 -4 2 zerone 0 zertwo];
toetemplate = sparse(toetemplate);
toetemplate1 = sparse(toetemplate1);
toetemplate2 = sparse(toetemplate2);
toetemplate3 = sparse(toetemplate3);
toetemplate4 = sparse(toetemplate4);
toetemplate5 = sparse(toetemplate5);
toetemplate6 = sparse(toetemplate6);
toetemplate7 = sparse(toetemplate7);

```

```

toetemplate8 = sparse(toetemplate8);
toetemplate8 = sparse(toetemplate9);
toetemplate8 = sparse(toetemplate10);
toetemplate8 = sparse(toetemplate11);
toetemplate8 = sparse(toetemplate12);
%get correct size of matrix and position of gaps for specific geometry
invbay = (bay ./ bay) - 1;
[I,J] = find(invbay);
Z = I + (J - 1) .* length;
gapvector = int32(Z);
A = [];
intcounter = 1;
nullcounter = 0;
toeplace = nodes;
%Computes the differentiation matrix
xpos = 1;
while xpos <= width
    ypos = 1;
    while ypos <= length
        switch double(bay(intcounter))
        case 0
            nullcounter = nullcounter + 1;
        case 1 %Area in water but not on boundary
            B = toetemplate(toeplace:(toeplace + nodes - 1));
            B = full(B);
            B(gapvector) = [];
            B = B .* perpdelta;
            B = sparse(B);
            A = [A;B];
            clear B;
        case 2 %Neumann boundary
            %decide whether to apply the boundaries to the x or y derivative
            if xpos == 1 & ypos == 1
                B = toetemplate5(toeplace:(toeplace + nodes - 1));
            elseif xpos == width & ypos == 1
                B = toetemplate6(toeplace:(toeplace + nodes - 1));
            elseif xpos == 1 & ypos == length
                B = toetemplate7(toeplace:(toeplace + nodes - 1));
            elseif xpos == width & ypos == length
                B = toetemplate8(toeplace:(toeplace + nodes - 1));
            elseif xpos == 1
                if bay(ypos + 1, xpos) ~= 0 & bay(ypos - 1, xpos) ~= 0
                    B = toetemplate3(toeplace:(toeplace + nodes - 1));
                elseif bay(ypos - 1, xpos) == 0 & bay(ypos + 1, xpos) == 2
                    B = toetemplate5(toeplace:(toeplace + nodes - 1));
                elseif bay(ypos + 1, xpos) == 0 & bay(ypos - 1, xpos) == 2

```

```

    B = toetemplate7(toeplace:(toeplace + nodes - 1));
elseif bay(ypos + 1, xpos) == 0 & bay(ypos - 1, xpos) == 0
    B = toetemplate10(toeplace:(toeplace + nodes - 1));
end
elseif xpos == width
    if bay(ypos + 1, xpos) ~= 0 & bay(ypos - 1, xpos) ~= 0
        B = toetemplate4(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos - 1, xpos) == 0 & bay(ypos + 1, xpos) == 2
        B = toetemplate6(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos + 1, xpos) == 0 & bay(ypos - 1, xpos) == 2
        B = toetemplate8(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos + 1, xpos) == 0 & bay(ypos - 1, xpos) == 0
        B = toetemplate9(toeplace:(toeplace + nodes - 1));
    end
elseif ypos == 1
    if bay(ypos, xpos + 1) ~= 0 & bay(ypos, xpos - 1) ~= 0
        B = toetemplate1(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos + 1) == 2 & bay(ypos, xpos - 1) == 0
        B = toetemplate5(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos - 1) == 2 & bay(ypos, xpos + 1) == 0
        B = toetemplate6(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos + 1) == 0 & bay(ypos, xpos - 1) == 0
        B = toetemplate12(toeplace:(toeplace + nodes - 1));
    end
elseif ypos == length
    if bay(ypos, xpos + 1) ~= 0 & bay(ypos, xpos - 1) ~= 0
        B = toetemplate2(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos + 1) == 2 & bay(ypos, xpos - 1) == 0
        B = toetemplate7(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos - 1) == 2 & bay(ypos, xpos + 1) == 0
        B = toetemplate8(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos + 1) == 0 & bay(ypos, xpos - 1) == 0
        B = toetemplate11(toeplace:(toeplace + nodes - 1));
    end
elseif bay(ypos + 1, xpos) ~= 0 & bay(ypos - 1, xpos) ~= 0 ...
    & bay(ypos, xpos + 1) ~= 0 & bay(ypos, xpos - 1) ~= 0
    B = toetemplate(toeplace:(toeplace + nodes - 1));
elseif bay(ypos - 1, xpos) == 0 & bay(ypos, xpos - 1) == 0
    B = toetemplate5(toeplace:(toeplace + nodes - 1));
elseif bay(ypos - 1, xpos) == 0 & bay(ypos, xpos + 1) == 0
    B = toetemplate6(toeplace:(toeplace + nodes - 1));
elseif bay(ypos + 1, xpos) == 0 & bay(ypos, xpos - 1) == 0
    B = toetemplate7(toeplace:(toeplace + nodes - 1));
elseif bay(ypos + 1, xpos) == 0 & bay(ypos, xpos + 1) == 0
    B = toetemplate8(toeplace:(toeplace + nodes - 1));
elseif bay(ypos + 1, xpos) == 0 & bay(ypos - 1, xpos) == 0

```

```

    if bay(ypos, xpos + 1) ~= 0
        B = toetemplate10(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos - 1) ~= 0
        B = toetemplate9(toeplace:(toeplace + nodes - 1));
    end
elseif bay(ypos, xpos + 1) == 0 & bay(ypos, xpos - 1) == 0
    if bay(ypos + 1, xpos) ~= 0
        B = toetemplate12(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos - 1, xpos) ~= 0
        B = toetemplate11(toeplace:(toeplace + nodes - 1));
    end
elseif bay(ypos, xpos + 1) ~= 0 & bay(ypos, xpos - 1) ~= 0
    if bay(ypos + 1, xpos) > 0
        B = toetemplate1(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos - 1, xpos) > 0
        B = toetemplate2(toeplace:(toeplace + nodes - 1));
    end
elseif bay(ypos + 1, xpos) ~= 0 & bay(ypos - 1, xpos) ~= 0
    if bay(ypos, xpos + 1) > 0
        B = toetemplate3(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos - 1) > 0
        B = toetemplate4(toeplace:(toeplace + nodes - 1));
    end
end
end
B = full(B);
B(gapvector) = [];
B = B .* perpdelta;
B = sparse(B);
A = [A;B];
clear B;
otherwise
    error('Case not valid.');
```

```

end
intcounter = intcounter + 1;
toeplace = toeplace - 1;
ypos = ypos + 1;
end
xpos = xpos + 1;
end
clkend = clock;
time1 = clkend - clkbegin;
save tmpfile nodes width length time1 A numeigvals bay
clear all
load tmpfile
clkbegin = clock;
%Compute the eigenvectors and eigenvalues of the differenation matrix "A"
```

```
[eigenvector,eigenvalues] = eigs(A,[],numeigvals,'SM');
lambda = diag(eigenvalues);
lamb = sort(lambda);
plot(lamb, 'g.')
clkend = clock;
time2 = clkend - clkbegin
```

```
%{
This program computes the velocity potentials for a geometry using Ames's
method where both boundaries and normal derivatives are shifted using the
exact boundary for the geometry.
%}

clear all
clkbegin = clock;
%This program computes the distances to move the boundary nodes and the
%correct normal angles for the boundary nodes.
circledistanceNeumann
[length width] = size(bay);
nodes = width * length;
stepsize = 1 / (width - 1);
numeigvals = 100;
%create Toeplitz template
firstzeros = length - 2;
secondzeros = nodes - firstzeros - 3;
zerone(1:firstzeros) = 0;
zertwo(1:secondzeros) = 0;
toetemplate = [zertwo 1 zerone 1 -4 1 zerone 1 zertwo];
toetemplate = int8(toetemplate);
%Get correct size of matrix and position of gaps for specific geometry
invbay = (bayone ./ bayone) - 1;
[I,J] = find(invbay);
Z = I + (J - 1) .* length;
gapvector = int32(Z);
A = [];
baycounter = 1;
intcounter = 1;
nullcounter = 0;
toeplace = nodes;
%enter phi coefficient values for eigenvalue matrix to be solved
while intcounter <= nodes
switch double(bay(intcounter))
case 0
    nullcounter = nullcounter + 1;
case 1
```

```

if baytwo(intcounter + 1) == 1 & baytwo(intcounter - 1) == 1 & baytwo(intcounter + width)
== 1 & baytwo(intcounter - width) == 1
    B = -(1 / stepsize)^2 * double(toeplace(toeplace:(toeplace + nodes - 1)));
else
    B(1:nodes) = 0;
    %The matrix C is used to calculate the value of Uxx and Uyy for all
    %the possible combinations of boundary points.
    C = zeros(5);
    %The vector point represents the vector containing the neighborhood
    %around the center node.
    point = zeros(5, 1);
    %There are four if statements for the neumann condition. Each of
    %the four points in the neighborhood can have either a value of one
    %or zero corresponding to four ifs. Since one point must be on the
    %boundary the point N must be created. Its position is independent
    %of the boundary points.
    if baytwo(intcounter + 1) == 2
        yangle = yalpha(intcounter + 1);
        yshift = movey(intcounter + 1);
        C(1, 1) = cos(yangle);
        C(1, 2) = sin(yangle);
        C(1, 3) = yshift * stepsize * cos(yangle);
        C(1, 4) = 0;
        C(1, 5) = yshift * stepsize * sin(yangle);
        clear yangle yshift
    elseif baytwo(intcounter + 1) == 1
        C(1, 1) = 0;
        C(1, 2) = stepsize;
        C(1, 3) = 0;
        C(1, 4) = 0;
        C(1, 5) = stepsize^2 / 2;
        point(1) = 1;
    end
    if baytwo(intcounter + width) == 2
        xangle = xalpha(intcounter + width);
        xshift = movex(intcounter + width);
        C(2, 1) = cos(xangle);
        C(2, 2) = sin(xangle);
        C(2, 3) = xshift * stepsize * sin(xangle);
        C(2, 4) = xshift * stepsize * cos(xangle);
        C(2, 5) = 0;
        clear xangle xshift
    elseif baytwo(intcounter + width) == 1
        C(2, 1) = stepsize;
        C(2, 2) = 0;
        C(2, 3) = 0;

```

```

    C(2, 4) = stepsize^2 / 2;
    C(2, 5) = 0;
    point(2) = 1;
end
if baytwo(intcounter - 1) == 2
    yangle = yalpha(intcounter - 1);
    yshift = movey(intcounter - 1);
    C(4, 1) = cos(yangle);
    C(4, 2) = sin(yangle);
    C(4, 3) = yshift * -stepsize * cos(yangle);
    C(4, 4) = 0;
    C(4, 5) = yshift * -stepsize * sin(yangle);
    clear yangle yshift
elseif baytwo(intcounter - 1) == 1
    C(4, 1) = 0;
    C(4, 2) = -stepsize;
    C(4, 3) = 0;
    C(4, 4) = 0;
    C(4, 5) = stepsize^2 / 2;
    point(4) = 1;
end
if baytwo(intcounter - width) == 2
    xangle = xalpha(intcounter - width);
    xshift = movex(intcounter - width);
    C(5, 1) = cos(xangle);
    C(5, 2) = sin(xangle);
    C(5, 3) = xshift * -stepsize * sin(xangle);
    C(5, 4) = xshift * -stepsize * cos(xangle);
    C(5, 5) = 0;
    clear xangle xshift
elseif baytwo(intcounter - width) == 1
    C(5, 1) = -stepsize;
    C(5, 2) = 0;
    C(5, 3) = 0;
    C(5, 4) = stepsize^2 / 2;
    C(5, 5) = 0;
    point(5) = 1;
end
%Quad tells where the point N is with respect to the molecule.
quad = [1 1];
if baytwo(intcounter - 1) == 2
    quad(1) = -1;
end
if baytwo(intcounter - width) == 2
    quad(2) = -1;
end

```

```

nangle = nalpha(intcounter);
nxshift = quad(2) * nxlambd(intcounter);
nyshift = quad(1) * nylambd(intcounter);
C(3, 1) = cos(nangle);
C(3, 2) = sin(nangle);
C(3, 3) = stepsize * (nyshift * cos(nangle) + ...
                    nxshift * sin(nangle));
C(3, 4) = nxshift * stepsize * cos(nangle);
C(3, 5) = nyshift * stepsize * sin(nangle);
clear nangle nxshift nyshift
%This section solves the matrix equation to find the finite
%differences equation for a specific boundary node.
D = C^-1;
D = D(4, :) + D(5, :);
D = -D;
B(intcounter)      = -(D(1) * point(1) + D(2) * point(2) + ...
                    D(4) * point(4) + D(5) * point(5));
B(intcounter + 1)  = D(1) * point(1);
B(intcounter + width) = D(2) * point(2);
B(intcounter - 1)  = D(4) * point(4);
B(intcounter - width) = D(5) * point(5);
end
B(gapvector) = [];
E = sparse(B);
A = [A;E];
clear B C D point E;
otherwise
    error('Case not valid.');
```

```

end
intcounter = intcounter + 1;
toeplace = toeplace - 1;
end
clkend = clock;
time1 = clkend - clkbegin;
clkbegin = clock;
[eigenvector,eigenvalues] = eigs(A,[],numeigvals,'SM');
lambda = diag(eigenvalues);
lamb = sort(lambda);
subplot(2,1,1)
plot(lamb, 'g.')
clkend = clock;
time2 = clkend - clkbegin
```

The following programs compute the inhomogenous potentials which are solutions to the Poisson equation. These potentials take into account normal flow through the boundaries with other bodies of water.

```
%{
This program computes the inhomogenous potentials for a geometry. The
boundary conditions used are load into the variable normalflow. Sthetaprog
is used to compute the area of the geometry and the line integral along the
boundary to account for the total number of sources.
%}
sourcesolns = [];
for n = 1:2
clkbegin = clock;

load ./chesapeakebaycalcs/normaldata
normalflow = normaldata(:,n);
clear normaldata
load ./chesapeakebaycalcs/quoddynoriver674X199map
[length width] = size(bay);
nodes = width * length;
bay = findingtheboundary_circ2(bay);
%initialize length scale for both perpendicular and diagonal finite difference
%computations
stepsize = 1/(width-1);
%this program will add Neumann BC's to a shape that is set up for Dirichlet
Sthetaprog
Stheta = Stheta * stepsize ^2;
%create Toeplitz template
firstzeros = length - 2;
secondzeros = nodes - firstzeros - 3;
zerone(1:firstzeros) = 0;
zertwo(1:secondzeros) = 0;
toetemplate = [zertwo 1 zerone 1 -4 1 zerone 1 zertwo];
toetemplate1 = [zertwo 1 zerone 0 -4 2 zerone 1 zertwo];
toetemplate2 = [zertwo 1 zerone 2 -4 0 zerone 1 zertwo];
toetemplate3 = [zertwo 0 zerone 1 -4 1 zerone 2 zertwo];
toetemplate4 = [zertwo 2 zerone 1 -4 1 zerone 0 zertwo];
toetemplate5 = [zertwo 0 zerone 0 -4 2 zerone 2 zertwo];
toetemplate6 = [zertwo 2 zerone 0 -4 2 zerone 0 zertwo];
toetemplate7 = [zertwo 0 zerone 2 -4 0 zerone 2 zertwo];
toetemplate8 = [zertwo 2 zerone 2 -4 0 zerone 0 zertwo];
toetemplate = sparse(toetemplate);
```

```

toetemplate1 = sparse(toetemplate1);
toetemplate2 = sparse(toetemplate2);
toetemplate3 = sparse(toetemplate3);
toetemplate4 = sparse(toetemplate4);
toetemplate5 = sparse(toetemplate5);
toetemplate6 = sparse(toetemplate6);
toetemplate7 = sparse(toetemplate7);
toetemplate8 = sparse(toetemplate8);
%get correct size of matrix and position of gaps for specific geometry
invbay = (bay ./ bay) - 1;
[I,J] = find(invbay);
Z = I + (J - 1) .* length;
gapvector = int32(Z);
%get solution matrix size
A = [];
intcounter = 1;
nullcounter = 0;
toeplace = nodes;
%Compute the differentiation matrix for the Poisson's equation with
%Neumann boundary conditions.
xpos = 1;
while xpos <= width
    ypos = 1;
    while ypos <= length
        switch double(bay(intcounter))
        case 0
            nullcounter = nullcounter + 1;
        case 1 %Area in water but not on boundary
            B = toetemplate(toeplace:(toeplace + nodes - 1));
            B = full(B);
            B(gapvector) = [];
            B = sparse(B);
            A = [A;B];
            clear B;
        case 2 %Neumann boundary
            %Decide whether to apply the boundaries to the x or y derivative
            %as well as add the source value to the vector on the right hand
            %side of the equation. The first line (99) computes the sources
            %while the other lines apply the approximation to the normal
            %derivative. Different portions of the "toetemplate" matrices are
            %used to make the programming more efficient.
            Stheta(intcounter) = Stheta(intcounter) - normalflow(intcounter) ./ stepsize;
            if xpos == 1 & ypos == 1
                B = toetemplate5(toeplace:(toeplace + nodes - 1));
            elseif xpos == width & ypos == 1
                B = toetemplate6(toeplace:(toeplace + nodes - 1));

```

```

elseif xpos == 1 & ypos == length
    B = toetemplate7(toeplace:(toeplace + nodes - 1));
elseif xpos == width & ypos == length
    B = toetemplate8(toeplace:(toeplace + nodes - 1));
elseif xpos == 1
    if bay(ypos + 1, xpos) ~= 0 & bay(ypos - 1, xpos) ~= 0
        B = toetemplate3(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos - 1, xpos) == 0 & bay(ypos + 1, xpos) == 2
        B = toetemplate5(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos + 1, xpos) == 0 & bay(ypos - 1, xpos) == 2
        B = toetemplate7(toeplace:(toeplace + nodes - 1));
    end
elseif xpos == width
    if bay(ypos + 1, xpos) == 2 & bay(ypos - 1, xpos) == 2
        B = toetemplate4(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos - 1, xpos) == 0 & bay(ypos + 1, xpos) == 2
        B = toetemplate6(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos + 1, xpos) == 0 & bay(ypos - 1, xpos) == 2
        B = toetemplate8(toeplace:(toeplace + nodes - 1));
    end
elseif ypos == 1
    if bay(ypos, xpos + 1) == 2 & bay(ypos, xpos - 1) == 2
        B = toetemplate1(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos + 1) == 2 & bay(ypos, xpos - 1) == 0
        B = toetemplate5(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos - 1) == 2 & bay(ypos, xpos + 1) == 0
        B = toetemplate6(toeplace:(toeplace + nodes - 1));
    end
elseif ypos == length
    if bay(ypos, xpos + 1) == 2 & bay(ypos, xpos - 1) == 2
        B = toetemplate2(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos + 1) == 2 & bay(ypos, xpos - 1) == 0
        B = toetemplate7(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos - 1) == 2 & bay(ypos, xpos + 1) == 0
        B = toetemplate8(toeplace:(toeplace + nodes - 1));
    end
elseif bay(ypos, xpos + 1) ~= 0 & bay(ypos, xpos - 1) ~= 0
    if bay(ypos + 1, xpos) ~= 0
        B = toetemplate1(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos - 1, xpos) ~= 0
        B = toetemplate2(toeplace:(toeplace + nodes - 1));
    end
elseif bay(ypos + 1, xpos) ~= 0 & bay(ypos - 1, xpos) ~= 0
    if bay(ypos, xpos + 1) ~= 0
        B = toetemplate3(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos, xpos - 1) ~= 0

```

```

        B = toetemplate4(toeplace:(toeplace + nodes - 1));
    end
    %this segment deals with the corners
    elseif bay(ypos - 1, xpos) == 0 & bay(ypos, xpos - 1) == 0
        B = toetemplate5(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos - 1, xpos) == 0 & bay(ypos, xpos + 1) == 0
        B = toetemplate6(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos + 1, xpos) == 0 & bay(ypos, xpos - 1) == 0
        B = toetemplate7(toeplace:(toeplace + nodes - 1));
    elseif bay(ypos + 1, xpos) == 0 & bay(ypos, xpos + 1) == 0
        B = toetemplate8(toeplace:(toeplace + nodes - 1));
    end
    B = full(B);
    B(gapvector) = [];
    B = sparse(B) .* -1/stepsize^2;
    A = [A;B];
    clear B;
otherwise
    error('Case not valid.');
```

```

end
intcounter = intcounter + 1;
toeplace = toeplace - 1;
ypos = ypos + 1;
end
xpos = xpos + 1;
end
clkend = clock;
time1 = clkend - clkbegin;
clkbegin = clock;
%Solve for the potential from the differentiation matrix "A" and the source
%vector "Stheta"
Stheta(gapvector) = [];
solution = A\Stheta;
cup = A * Stheta;
clkend = clock;
time2 = clkend - clkbegin
sourcesolns = cat(3, sourcesolns, solution);
save sourcesolns sourcesolns bay width length nodes n
clear all
load sourcesolns sourcesolns n
end
```

The following programs are auxiliary programs used in the above three sections.

Findingtheboundary_circ.m

```

function [baytwo] = findingtheboundary(bay)
%{
This program places twos around the current "bay" matrix to denote the
boundary. This program does not include convex corners.
%}
wc = 1;
[bl,bw] = size(bay);
baytwo = bay;
while wc <= bw
    lc = 1;
    while lc <= bl
        if ((wc == 1 & lc == 1)&bay(2,2) == 1)|((wc == 1 & lc == bl)&bay(lc-1,2) == 1) ...
            |((wc == bw & lc == 1)&bay(2, wc - 1) == 1)|((wc == bw & lc == bl)&bay(lc - 1,wc -1)
== 1)
            baytwo(lc,wc) = 2;
        elseif (lc ~= 1)&(lc~=bl)&(wc~=bw)&(wc == 1) & (bay(lc, 2) == 1)
            baytwo(lc, wc) = 2;
        elseif (lc ~= 1)&(lc~=bl)&(wc~=1)&(wc == bw) & (bay(lc, bw - 1) == 1)
            baytwo(lc, wc) = 2;
        elseif (lc~=bl)&(wc~=1)&(wc~=bw)&(lc == 1) & (bay(2, wc) == 1)
            baytwo(lc, wc) = 2;
        elseif (lc ~= 1)&(wc~=1)&(wc~=bw)&(lc == bl) & (bay(lc - 1, wc) == 1)
            baytwo(lc, wc) = 2;
        elseif (lc ~= 1)&(lc~=bl)&(wc~=1)&(wc~=bw)&bay(lc, wc) == 0 & (bay(lc + 1,wc) == 1
| bay(lc - 1, wc)...
            == 1 | bay(lc, wc + 1) == 1 | bay(lc, wc - 1) == 1)
            baytwo(lc, wc) = 2;
        end
        lc = lc + 1;
    end
    wc = wc + 1;
end

```

Circledistance.m

```

%Program begins with the input of the circle shape from there use  $x^2 + y^2$ 
% = .5^2 to calculate the distance to the edge of the circle in both the x
% and y direction. The matrices computed in this program are used in the
% computation of the differentiation matrix for Ames method for Dirichlet
% boundary conditions.
% Create the circular geometry on square grid.
circle
bayone = bay;

```

```

findingtheboundary_circ
baytwo = bay;
bay = bayone;
%Actual error in the approximation of the boundary in the x and y direction
xerror = zeros(xnode - 1);
yerror = zeros(ynode - 1);
xspot = zeros(xnode - 1);
yspot = zeros(xnode - 1);
stepsize = 1 / (xnode - 2);
imax = xnode - 1;
jmax = ynode - 1;
i = 1;
while i <= imax
    j = 1;
    while j <= jmax
        if baytwo(j, i) == 2
            xposition = (i - 1) * stepsize - .5;
            yposition = (j - 1) * stepsize - .5;
            actualx = sqrt(abs((1/2)^2 - yposition^2));
            actualy = sqrt(abs((1/2)^2 - xposition^2));

            xspot(j, i) = xposition;
            yspot(j, i) = yposition;
            xerror(j, i) = abs(abs(actualx) - abs(xposition));
            yerror(j, i) = abs(abs(actualy) - abs(yposition));
            invxerror(j, i) = stepsize - xerror(j, i);
            invyerror(j, i) = stepsize - yerror(j, i);
        end
        j = j + 1;
    end
    i = i + 1;
end
movex = abs(invxerror) ./ stepsize;
movey = abs(invyerror) ./ stepsize;

```

Triangledistance.m

```

%Program begins with the input of the triangle shape and then calculates
%error in the x and y directions at each boundary node. The corresponding
%matrices are used to create the differentiation matrices for Ames method
%for Dirichlet boundary conditions.
%Create the triangle geometry:
triangle_new
bayone = bay;
%Add a boundary layer of twos to the geometry matrix

```

```

findingtheboundary
baytwo = bay;
bay = bayone;
%Actual error in the approximation of the boundary in the x and y direction
xerror = zeros(xnode);
yerror = zeros(ynode);
%Matrices used to check the solutions.
xspot = zeros(xnode);
yspot = zeros(ynode);
xie = [];
yie = [];
stepsize = 1 / (xnode - 1);
imax = xnode;
jmax = ynode;
i = 1;
while i <= imax
    j = 1;
    while j <= jmax
        if baytwo(j, i) == 2
            %Set error on the diagonal sides of the equilateral triangle
            xposition = (i - 1) * stepsize;
            yposition = -((j - 1) * stepsize - 1);
            if xposition <= 1/2
                actualx = yposition / gradient;
                actualy = gradient * xposition;
            elseif xposition > 1/2
                actualx = (yposition - sqrt(3))/-gradient;
                actualy = -gradient * xposition + sqrt(3);
            end
            actualy2 = 0;
            xspot(j, i) = xposition;
            yspot(j, i) = yposition;
            xerror(j, i) = actualx - xposition;
            if yposition > 0
                yerror(j, i) = actualy - yposition;
            else
                %If the node is on the bottom horizontal boundary
                yerror(j, i) = 0 - yposition;
            end
            invxerror(j, i) = stepsize - abs(xerror(j, i));
            invyerror(j, i) = stepsize - abs(yerror(j, i));
        end
        j = j + 1;
    end
    i = i + 1;
end

```

```
movex = abs(invxerror) ./ stepsize;
movey = abs(invyerror) ./ stepsize;
```

Circledistance_neumann.m

```
%{
This program calculates information required for Ames's method for the
Neumann boundary condition. The shifts in the x and y direction for each
boundary node are calculated as well as the corresponding normal angles.
The position and normal angle of an extra node on the boundary, node N, are
also calculated.
%}
circle
baytwo = findingtheboundary_circ(bay)
%actual error in the approximation of the boundary in the x and y directions
xerror = zeros(xnode - 1);
yerror = zeros(xnode - 1);
xspot = zeros(xnode - 1); % matrix used for testing purposes
yspot = zeros(xnode - 1); % matrix used for testing purposes
xalpha = zeros(xnode - 1); % normal angle if x is moved
yalpha = zeros(xnode - 1); % normal angle if y is moved
nxlambda = zeros(xnode - 1); % x position of node N
nylambda = zeros(xnode - 1); % y position of node N
nalpha = zeros(xnode - 1); % normal angle for node N
stepsize = 1 / (xnode - 2);
imax = xnode - 1;
jmax = ynode - 1;
i = 1;
while i <= imax
    j = 1;
    while j <= jmax
        xposition = (i - 1) * stepsize - .5;
        yposition = (j - 1) * stepsize - .5;
        %this if statement finds the normals and positions of the boundary
        %points
        if baytwo(j, i) == 2
            % x value of the point if the y value is held constant when
            % moved to the boundary
            actualx = sqrt(abs((1/2)^2 - yposition^2));
            if xposition < 0
                actualx = -actualx;
            end
            % y value of the point if the x value is held constant when
            % moved to the boundary
            actualy = sqrt(abs((1/2)^2 - xposition^2));
```

```

if yposition < 0
    actually = -actually;
end
xspot(j, i) = xposition; %test matrix
yspot(j, i) = yposition; %test matrix
xerror(j, i) = abs(abs(actualx) - abs(xposition));
yerror(j, i) = abs(abs(actually) - abs(yposition));
invxerror(j, i) = stepsize - xerror(j, i);
invyerror(j, i) = stepsize - yerror(j, i);
xalpha(j, i) = atan2(yposition, actualx);
yalpha(j, i) = atan2(actually, xposition);
end
    %This if statement finds the normal angle from the x-axis and the
    %position of the point N found at the intersection
    %between the boundary of the circle and the line running through
    %point P and the origin.
if baytwo(j, i) == 1 & ((baytwo(j + 1, i) == 2 | ...
baytwo(j - 1, i) == 2 | baytwo(j, i + 1) == 2 | baytwo(j, i - 1) == 2))
    nalpha(j, i) = atan2(yposition, xposition);
    hypoteneuse = .5 - sqrt(xposition^2 + yposition^2);
    if abs(nalpha(j,i)) > pi
        calcangle = pi - abs(nalpha(j,i));
    else
        calcangle = abs(nalpha(j,i));
    end
    nxlambdaj(j, i) = cos(calcangle) * hypoteneuse;
    nylambdaj(j, i) = sin(calcangle) * hypoteneuse;
end
j = j + 1;
end
i = i + 1;
end
nxlambdaj = abs(nxlambdaj) ./ stepsize;
nylambdaj = abs(nylambdaj) ./ stepsize;
movex = invxerror ./ stepsize;
movey = invyerror ./ stepsize;

```

Sthetaprogram.m

```

%This program computes this area within the geometry and the line integral
%of the sources around the boundaries. The output is "Stheta which is the
%initial vector used in the computation of the inhomogenous potentials.
Stheta = zeros(nodes, 1);
Sintegral = 0;
area = 0;

```

```

squaresize = stepsize^2;
j = 1;;
while j <= length
    i = 1;
    while i <= width
        %Computes the line integral of normalflow
        Sintegral = Sintegral + normalflow(j, i) * stepsize;
        if bay(j, i) == 1
            area = area + squaresize;
        end
        if bay(j, i) == 2
            %This section computes the area for boundary nodes
            if (i == 1 & j == 1) | (i == 1 & j == length) | (i == width & j == 1) | (i == width & j ==
length)
                area = area + squaresize / 4;
            elseif i == 1 & bay(j + 1, i) ~= 0 & bay(j - 1, i) ~= 0
                area = area + squaresize / 2;
            elseif i == 1 & (bay(j + 1, i) == 0 | bay(j - 1, i) == 0)
                area = area + squaresize / 4;
            elseif i == width & bay(j + 1, i) ~= 0 & bay(j - 1, i) ~= 0
                area = area + squaresize / 2;
            elseif i == width & (bay(j + 1, i) == 0 | bay(j - 1, i) == 0)
                area = area + squaresize / 4;
            elseif j == 1 & bay(j, i + 1) ~= 0 & bay(j, i - 1) ~= 0
                area = area + squaresize / 2;
            elseif j == 1 & (bay(j, i + 1) == 0 | bay(j, i - 1) == 0)
                area = area + squaresize / 4;
            elseif j == length & bay(j, i + 1) ~= 0 & bay(j, i - 1) ~= 0
                area = area + squaresize / 2;
            elseif j == length & (bay(j, i + 1) == 0 | bay(j, i - 1) == 0)
                area = area + squaresize / 4;
            end
        end
        i = i + 1;
    end
    j = j + 1;
end
Stheta(:, :) = Sintegral / area;

```
