Computation of Eigenvalues for Starlike Domains

Robert A. Book

April, 1994

TR94-20
RICE UNIVERSITY

Computation of Eigenvalues for Starlike Domains

by

Robert A. Book

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Arts

APPROVED, THESIS COMMITTEE:

Steven J. Cox, Chairman
Assistant Professor of Computational and
Applied Mathematics

Richard A. Tapia
Noah Harding Professor of Applied
Mathematics

Clint Dawson
Assistant Professor of Computational and
Applied Mathematics

Houston, Texas

April, 1994
Computation of Eigenvalues for Starlike Domains

Robert A. Book

Abstract

In this paper, we present a software tool for the computation of eigenvalues of starlike domains defined by polar boundary functions. We also offer and numerically test a conjecture on the monotonicity of the fundamental eigenvalues of the members of a family of starlike domains.
Acknowledgments

I would like to thank my thesis advisor, Professor Steven J. Cox, for his patience, guidance, and advice throughout the course of this project.

I would also like to express thanks to Professors Richard A. Tapia and Clint Dawson for serving on my thesis committee, and for serving, respectively, as an extraordinary instructor and graduate advisor.

In addition, I would like to thank Dr. Philip T. Keenan for his time and patience in helping me with the PLTMG package, and Professor Fadil Santosa for providing an example of his use of PLTMG for an eigenvalue problem.

I would also like to thank Anthony José Kearsley for his advice and encouragement during the period of my studies at Rice University.

Last, but certainly not least, I would like to express my gratitude to my fiancé, Mandy Greenfield, and to my parents, Ruth and Stephen Book, for their unending support, patience, guidance, and advice both in the course of this undertaking and in all aspects of my life.
Contents

Abstract
Acknowledgments
List of Illustrations
List of Tables

1 Introduction

2 Starlike Domains
   2.1 Definitions
   2.2 Polar Representation of Starlike Domains

3 Software for Computing Eigenvalues
   3.1 Description of the Software Tool
   3.2 Initializing the Domain
   3.3 Finding the Fundamental Eigenvalue
      3.3.1 Choosing the Initial Guess by Domain Monotonicity
      3.3.2 Computing the Eigenfunction and Testing for the
            Fundamental Eigenvalue
      3.3.3 Graphical Output

4 Monotonicity of a Family of Starlike Domains
   4.1 A Family of Polar Starlike Domains
   4.2 A Conjecture on Monotonicity
   4.3 Testing the Conjecture

A Example Functions

B Software Tools
   B.1 Function definitions for the PDE (star.f)
B.2 Generating the initial triangulation (PolarTri) ................. 35
B.3 Modified version of TRIPLT ........................................... 37

C Sounding the Fundamental Frequency 43

Bibliography 44
Illustrations

3.1 Domain used for the sample program run presented in Figures 3.2 and 3.3. ................................................................. 8

3.2 Dialog of user interaction and program output, showing initialization of the triangulation and PDE for the computation of the smallest eigenvalue of the Laplacian on the region bounded by
\[ f(\theta) = 2 + \sin(10\theta) . \] This sample run is continued in the next figure. ................................................................. 9

3.3 Dialog of user interaction and program output (continued). The smallest eigenvalue is found on the first attempt. The corresponding eigenfunction is then plotted. ................................................................. 10

3.4 Initial triangulation for the sample program run. This plot was produced by the first "i" command in Figure 3.2. ................................................................. 12

3.5 Further triangulation produced by the "t" command. This plot was produced by the second "i" command in Figure 3.2. ................................................................. 14

3.6 A three-dimensional plot, showing level curves, of the fundamental eigenfunction \( z = u(x,y) \) of the Laplacian over the domain shown in Figure 3.2. The function is viewed from the point \((0,1,1)\). ................................................................. 18

4.1 The first four members of the one-parameter family of functions
\[ f_n(\theta) = 2 + \sin(n\theta) . \] ................................................................. 21

4.2 Members of the one-parameter family of functions \( f_n(\theta) = 2 + \sin(n\theta) \) for \( n = 5, 10, 12, \) and 15. ................................................................. 22

4.3 The fundamental eigenvalue \( \lambda_1(\Omega_n) \) as a function of \( n \) for the family of domains given by \( f_n(\theta) = 2 + \sin(n\theta) \). The horizontal lines at 0.64 and 5.76 indicate the fundamental eigenvalues for the discs of radius 1 and 3, respectively. ................................................................. 24
3.1 Selected options for the “s” command. The first column shows the
alias for setting the variable in the command line, and the second
shows the variable name used in Bank [1]. ........................................ 13

4.1 Fundamental eigenvalues for the family of domains given by
\( \partial \Omega = f_n(\theta) = 2 + \sin(n\theta) \). Note that the eigenvalues \( \lambda_1(\Omega_n) \) are
monotonically increasing as a function of \( n \). ........................................ 23
Chapter 1

Introduction

When a drum is struck, a sound is produced. The precise sound that is produced by a particular drum depends on many factors, such as the force with which it is struck, the tensile strength of the membrane, and so on, but the basic tone of the drum depends only on the fundamental frequency of vibration of the drumhead – and this frequency is completely determined by the size and shape of the drumhead.

Consider an open, simply connected, bounded region $\Omega \subset \mathbb{R}^2$. We will think of our drum as a membrane of shape $\Omega$ attached at its boundary to a rim of shape $\partial \Omega$, the boundary of region $\Omega$. The vibrations of the membrane may be modeled by the two-dimensional wave equation on this region. The frequencies of vibration are the eigenvalues of the Laplacian, which are given by the solutions to

$$-\Delta u = \lambda u \quad \text{for} \quad (x, y) \in \Omega \quad (1.1)$$

$$u = 0 \quad \text{for} \quad (x, y) \in \partial \Omega$$

where $u(x, y)$ is the displacement of the membrane

$\Delta u = u_{xx} + u_{yy}$ is the Laplacian of $u$

$\partial \Omega$ is the boundary of $\Omega$

$\lambda$ is an eigenvalue of the Laplacian
Values of \( \lambda \) for which the system (1.1) is solvable with \( \lambda \neq 0 \) and \( u(x,y) \neq 0 \) are the \textit{frequencies of vibration} of the membrane of shape \( \Omega \). For any given \( \Omega \), there are infinitely many frequencies (eigenvalues), and they satisfy

\[
0 < \lambda_1(\Omega) < \lambda_2(\Omega) \leq \lambda_3(\Omega) \leq \cdots < \infty
\]  

(Courant and Hilbert, [2]). Note that the first two inequalities are strict; that is, the smallest eigenvalue \( \lambda_1 \) is both distinct and strictly positive. The fundamental frequency of vibration of a membrane of shape \( \Omega \) is thus the smallest positive eigenvalue \( \lambda_1 \) of the Laplacian on \( \Omega \).

We are interested in the mapping

\[
\Omega \mapsto \lambda_1(\Omega)
\]

i.e., the map that takes any given open, simply connected bounded region to its fundamental frequency of vibration.

Now, suppose we constrain the region \( \Omega \) to be of area \( A \) and consider the minimization problem

\[
\min_{|\Omega|=A} \lambda_1(\Omega)
\]

According to Cox and Ross [3], it was suggested by Rayleigh and proved independently by Faber (1923) and Krahm (1925) that the minimizer \( \Omega_{\text{min}} \) is a disc of the appropriate radius, \( r = \sqrt{\frac{A}{\pi}} \), and the minimum value \( \lambda_1(\Omega_{\text{min}}) \) is approximately \( \frac{5.76}{r^2} \).

Over the same collection of possible sets \( \Omega \), the corresponding maximization problem does not have a solution, since a set \( \Omega_0 \in \{ \Omega : |\Omega| = A \} \) with \( \lambda_1(\Omega_0) = l \) may be
found for any arbitrarily large value of $l$. However, a solution can be found if the class of regions $\Omega$ is properly restricted. It has been proven (Cox and Ross [3]) that if $\Omega$ is restricted to be a convex domain of finite perimeter, then the maximization problem has a solution. However, although the maximum value of $\lambda_1$ can be bounded, neither the maximizer $\Omega_{\text{max}}$ nor the maximum value $\lambda_1(\Omega_{\text{max}})$ is known analytically, nor has anyone devised a method for computing it numerically. Cox and Ross [3] also showed that a maximum exists if $\Omega$ is constrained to be a starlike domain, to contain a given disk, and to have a perimeter $|\partial \Omega|$ not exceeding a particular value. Again, neither the maximizer nor the maximum value is known analytically or numerically.

In general, it is difficult to calculate $\lambda_1(\Omega)$ analytically for an arbitrary domain $\Omega$. In this paper, we present a software tool that can be used to compute numerically the first eigenvalue $\lambda_1(\Omega)$ where $\Omega$ is any starlike domain. We also offer a conjecture and a numerical verification of the monotonicity of $\lambda_1(\Omega)$ for a restricted class of starlike domains $\Omega$.

It is hoped that the work described herein will help lead to a solution of the maximization problem for $\Omega$ restricted to the class of starlike domains. By providing a reliable function evaluator for $\lambda_1(\Omega)$, we hope to make it possible to approximate numerically the maximizer and the maximum value once a suitable optimization procedure is implemented.
Chapter 2

Starlike Domains

One class of $\Omega$ for which a solution to the maximization problem is known to exist is the collection of starlike domains. This is the class with which we will be concerned for the remainder of this paper.

2.1 Definitions

We define a starlike domain as follows:

**Definition 2.1** Let $\Omega$ be a subset of $\mathbb{R}^2$ bounded by a simple closed curve $\partial \Omega$. Then $\Omega$ is a starlike domain if there exists a point $x \in \Omega$, such that for every point $y \in \partial \Omega$, we have $tx + (1-t)y \in \Omega$ for all $t \in (0, 1]$.

The point $x$ is called a center of $\Omega$.

Note that we have left the domain of $t$ half-open to allow for the fact that even if $y \in \partial \Omega$, it might be that $y \not\in \Omega$. This is true, for example, in the case that $\Omega$ is an open set. It should be clear that if $y \in \Omega \cap \partial \Omega$, then the condition holds if $t = 0$ as well.

Note also that the collection of convex sets is a subcollection of the collection of starlike domains. To see this, recall the definition of a convex set in $\mathbb{R}^2$:
Definition 2.2 Let \( S \subset \mathbb{R}^2 \). Then \( S \) is a **convex set** if for and two points \( x, y \in S \), we have \( tx + (1 - t)y \in S \) for all \( t \in [0, 1] \).

It can be seen that a convex set is simply a starlike domain for which any point \( x \) in the set may serve as a center.

### 2.2 Polar Representation of Starlike Domains

One reason starlike domains are useful for the problem of interest is that it is easy to specify their boundaries. As can be seen from Definition 2.1, the most prominent feature of a starlike domain is that a line segment drawn between a center and any point on the boundary is completely contained within the domain. If we locate the origin of a polar coordinate system at a center \( x \) of a starlike domain \( \Omega \), then any point \( y \) on the boundary may be expressed in polar coordinates as \((|y - x|, \theta)\). If \( f(\theta) = |y - x| \) is a continuous function, then the boundary is given in polar coordinates by

\[
\partial \Omega = \{(r, \theta) : r = f(\theta), \ 0 \leq \theta < 2\pi\} \tag{2.1}
\]

In practice, it is generally easiest to specify \( \Omega \) by the polar function \( f(\theta) \) that defines its boundary \( \partial \Omega \).

Note that while it is likely in practice that most domains of interest will be bounded by continuous, \( 2\pi \)-periodic functions of \( \theta \), it is not necessary to require either of these to be the case. If the function is not \( 2\pi \)-periodic, we may simply choose a half-open interval of length \( 2\pi \) (usually \([0, 2\pi)\)) with which to define \( \partial \Omega \). If
the function is not continuous but is piecewise continuous, then at points where \( f(\theta) \) is discontinuous, we include in \( \partial \Omega \) every point on the radial line segment connecting the upper and lower limits of \( f \) at \( \theta \). That is,

\[
\partial \Omega = \{ (r, \theta) : 0 \leq \theta < 2\pi, \lim_{\theta_0 \uparrow \theta} f(\theta_0) \leq r \leq \lim_{\theta_0 \downarrow \theta} f(\theta_0) \} \tag{2.2}
\]

Note that at values of \( \theta \) for which \( f \) is continuous, \( r \) takes on only the single value \( \lim_{\theta_0 \uparrow \theta} f(\theta_0) = \lim_{\theta_0 \downarrow \theta} f(\theta_0) = f(\theta) \).
Chapter 3

Software for Computing Eigenvalues

This chapter describes a software tool for computing eigenvalues of starlike domains. The basis for this tool is the PLTMG (Piecewise Linear Triangular Multigrid) package by Bank [1]. PLTMG is a library of Fortran subroutines that allows a user to solve planar boundary value problems using adaptive finite element discretizations based on piecewise linear triangular grids. The package also includes graphics support, as well as a run-time module allowing for interactive solution of problems.

3.1 Description of the Software Tool

The present author developed an add-on package for PLTMG that provides for the solution of (1.1) when the domain $\Omega$ is expressed as a piecewise continuous polar function, as in (2.2). The add-on package allows the user to install easily several polar functional forms at compile time and then select the function, including any necessary parameters, as well as the fineness of the initial triangulation grid, at run time. A template for installing new functions is given in Appendix A. The package also includes a modification of PLTMG's subroutine TRIPLT, which allows the user to test
whether the eigenvalue that has been found is the first eigenvalue. The modifications
and additions to the PLTMG package are described in Appendix B.

We will describe the use of this tool using a sample program dialog in which we
find the first eigenvalue for the starlike domain $\Omega$ with its boundary $\partial\Omega$ defined by
the polar function $f(\theta) = 2 + \sin(10\theta)$. A plot of this function is given in Figure 3.1.
The sample program dialog is shown in Figures 3.2 and 3.3.

![A star with 10 lobes defined by $f(\theta) = 2 + \sin(10\theta)$](image)

**Figure 3.1** Domain used for the sample program run presented in Figures 3.2 and 3.3.

### 3.2 Initializing the Domain

The first several commands serve the purpose of initializing the triangulation, gen-
erating the triangle tree, and finding an initial solution. In Figure 3.2, we see that
You may use any of the following polar functions to define the domain:

1. \( f_1(\theta) = 2 + \sin(\theta) \)
2. \( f_2(\theta) = 2 + \sin(2\theta) \)
3. \( f_3(\theta) = 2 + \sin(3\theta) \)
4. \( f_4(\theta) = 2 + \sin(4\theta) \)
5. \( f_5(\theta) = 2 + \sin(5\theta) \)
6. \( f_6(\theta) = 1 + (\sin(2\theta))^2 \)
7. \( f_7(\theta) = 1 + \theta^2 \)
8. \( f_8(\theta) = r \) (A circle of radius \( r \))
9. \( f_9(\theta) = 2 + \sin(n\theta) \) (A star with \( n \) lobes.)

Enter the number of your choice:
9

Enter the number of lobes for the domain:
10

Experience shows that a good initial number of triangles is 8 times the number of lobes.

Enter the initial number of triangles:
80

Domain Boundary: Star with 10 lobes
Initial triangles: \( \#T = 80 \)

pltmg edition 6.2

* commands
  t triangulate domain  k create skeleton  d write data file
  s solve p. d. e.  f plot function  l read data file
  g graph output  i plot input  j journal file
  r reset title  u call usrcmd  x exit
  *k

create skeleton

oflag = 0 ntr = 80 nvr = 81 ncr = 80 nbr = 160
+i
plot input
* t

triangulate domain

oflag = 0 nt = 3383 nv = 1905 nc = 425 nb = 425
+i
plot input
*e ir=3,li=3,a=1,p=0,r=0,l=0

solve p. d. e.

  level 1 continuation nv =****

 lambda  rho  lambda dot  rho dot  determinant eigenvalue
  0 1 0.000000E+00 0.000000E+00 0.100000E+01 0.000000E+00 0.357076**** 0.631081E-03

iflag = 0 aflag = 0 yflag = 0 cflag = 0

Figure 3.2 Dialog of user interaction and program output, showing initialization of the triangulation and PDE for the computation of the smallest eigenvalue of the Laplacian on the region bounded by \( f(\theta) = 2 + \sin(10\theta) \). This sample run is continued in the next figure.
* s p=1,1=5
solve p. d. e.  level 1 continuation  nv =****

lambda  rho  lambda dot  rho dot  determinant  eigenvalue
0 1 0.50000E+01 0.000E+00 0.100E+01 0.000E+00 -0.906e+++ -0.611E-04

find limit / bifurcation point
0 1 0.40439E+01 0.000E+00 0.100E+01 0.000E+00 0.939e+++ 0.831E-04
0 1 0.45949E+01 0.000E+00 0.100E+01 0.000E+00 0.218e+++ 0.771E-06
0 1 0.46001E+01 0.000E+00 0.100E+01 0.000E+00 0.296e+++ 0.490E-08
0 1 0.46004E+01 0.000E+00 0.100E+01 0.000E+00 0.165e+++ -0.613E-07
0 1 0.48002E+01 0.000E+00 0.100E+01 0.000E+00 -0.183e+++ -0.305E-04
0 1 0.47003E+01 0.000E+00 0.100E+01 0.000E+00 -0.187e+++ -0.152E-04
0 1 0.46005E+01 0.000E+00 0.100E+01 0.000E+00 0.156e+++ -0.487E-07
0 1 0.46504E+01 0.000E+00 0.100E+01 0.000E+00 -0.130e+++ -0.763E-05
0 1 0.46254E+01 0.000E+00 0.100E+01 0.000E+00 -0.748e+++ -0.383E-05
0 1 0.46129E+01 0.000E+00 0.100E+01 0.000E+00 -0.397e+++ -0.195E-05

iflag = 0  aflag = 0  yflag = 0  cflag = 0
*s p=2

solve p. d. e.  level 1 continuation  nv =****

lambda  rho  lambda dot  rho dot  determinant  eigenvalue
0 0 0.46129E+01 0.000E+00 0.631E+00 0.955E-06 -0.397e+++ -0.195E-05

iflag = 0  aflag = 0  yflag = 0  cflag = 0
*s r=1

solve p. d. e.  level 1 continuation  nv =****

lambda  rho  lambda dot  rho dot  determinant  eigenvalue
0 5 0.45999E+01 0.602E-02 -0.528E-02 -0.155E+00 0.497e+++ 0.479E-07
0 9 0.45999E+01 0.503E+00 -0.295E-03 -0.142E+01 0.410e+++ 0.201E-07
0 5 0.45999E+01 0.100E+01 -0.249E-03 -0.200E+01 0.410e+++ 0.231E-07

iflag = 0  aflag = 0  yflag = 0  cflag = 0
* if ifun=20
plot function

Eigenfunktion has minimum value:  0.
and maximum value:  0.965914

The current value of lambda is the smallest positive eigenvalue.
* if ifun=0, nx=0, ny=1, nz=1, ls=2, d=2
plot function
*di0
write data file
*x
exit

**Figure 3.3** Dialog of user interaction and program output (continued). The smallest eigenvalue is found on the first attempt. The corresponding eigenfunction is then plotted.
the user is presented with a choice of nine different functions, two of which are really one-parameter families of functions. In this example, the user selects function "9," a star with \( n \) lobes, then chooses \( n = 10 \). A plot of this function is shown in Figure 3.1. The user is then asked for the number of finite-element triangles to use in the initial triangulation. Before being asked, however, the user is informed that previous experience with this family of functions indicates that the number of triangles equal to eight times the number of lobes is adequate. (The reason for this is that the boundary function is evaluated, on average, once for each triangle. The number of function evaluations must be sufficient to pick out all the essential features of the boundary.)

The program then automatically produces an initial triangulation of the domain with the requested number of triangles, and then enters a command mode, in which the user may solve for an eigenvalue, produce plots, and so on. From this point on, all user commands are indicated by the "\*" prompt in Figures 3.2 and 3.3. Notice that the first \* is on a line by itself. This indicates that the user typed <Return> with no command, to produce a list of the available commands.

The user's first few commands (those shown in Figure 3.2) initialize the system in preparation for solving the PDE. The "k" command creates a skeleton by superimposing the initial triangulation over the domain. The "i" command then produces a plot of the initial triangulation (shown in Figure 3.4). Note that the boundary edges of each "triangle" are curved to approximate the actual boundary of the domain.
Figure 3.4 Initial triangulation for the sample program run. This plot was produced by the first "1" command in Figure 3.2.
The user then enters "t" to further triangulate the domain, and plots the fully triangulated domain with the "i" command. This plot is shown in Figure 3.5.

The user then initializes the PDE with the command

\texttt{s ir=3,li=3,a=1,p=0,r=0,l=0}

Here "s" is the command to solve the PDE, and the assignments following it indicate various options that may be set depending on what type of solution is desired. For the problem of finding the first eigenvalue, the above options are appropriate for the initial call to the solver routine. A brief summary of the options above, with the settings used in our example, is presented in Table 3.1. For a more complete discussion of these options, the user is referred to Bank [1].

<table>
<thead>
<tr>
<th>User setting</th>
<th>Variable name in PLTMG</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ir</td>
<td>IREFN</td>
<td>Refinement level (Logical height of the triangle tree.)</td>
</tr>
<tr>
<td>li</td>
<td>LEVEL1</td>
<td>Level of the triangle tree to use for the initial mesh.</td>
</tr>
<tr>
<td>a</td>
<td>IADAPT</td>
<td>Use \texttt{a=1} for first call; \texttt{a=0} subsequently.</td>
</tr>
<tr>
<td>p</td>
<td>IPROB</td>
<td>Continuation option: Use \texttt{p=0} for initial call; \texttt{p=1} to search for an eigenvalue; \texttt{p=2} to switch branches after finding eigenvalue.</td>
</tr>
<tr>
<td>r</td>
<td>RTRGT</td>
<td>Value of (\rho): Use \texttt{r=0} initially; then \texttt{r=1} after switching branches.</td>
</tr>
<tr>
<td>l</td>
<td>RLTRGT</td>
<td>Target value of (\lambda): Use \texttt{l=0} for initial call, then set slightly higher than the suspected first eigenvalue. See text for more information.</td>
</tr>
</tbody>
</table>

\textbf{Table 3.1} Selected options for the "s" command. The first column shows the alias for setting the variable in the command line, and the second shows the variable name used in Bank [1].
Figure 3.5  Further triangulation produced by the "t" command. This plot was produced by the second "i" command in Figure 3.2.
3.3 Finding the Fundamental Eigenvalue

The user now attempts to find the fundamental eigenvalue. The call to “s” is changed to “p=1,” and “1” is chosen to be an initial guess for \( \lambda_1 \).

3.3.1 Choosing the Initial Guess by Domain Monotonicity

We choose our initial guess using the principle of domain monotonicity (see Cox and Ross [3]). This principle states that for two open, connected, bounded domains \( \Omega_1 \) and \( \Omega_2 \),

\[
\Omega_1 \subset \Omega_2 \Rightarrow \lambda_1(\Omega_1) \geq \lambda_1(\Omega_2) \tag{3.1}
\]

Note from Figure 3.1 that \( \Omega \) contains a disc of radius 1 and is itself contained in a disc of radius 3. (This can also be seen from the expression for \( f(\theta) \).) If we denote these discs \( \Omega_1 \) and \( \Omega_3 \) respectively, we have

\[
\Omega_1 \subset \Omega \subset \Omega_3 \Rightarrow \lambda_1(\Omega_1) \geq \lambda_1(\Omega) \geq \lambda_1(\Omega_2) \tag{3.2}
\]

\[
\Rightarrow \frac{2.4^2}{1^2} \geq \lambda_1(\Omega) \geq \frac{2.4^2}{3^2} \tag{3.3}
\]

\[
\Rightarrow 5.76 \geq \lambda_1(\Omega) \geq 0.64 \tag{3.4}
\]

Therefore, we have a rough idea of where \( \lambda_1(\Omega) \) should be. Since PLTMG uses a secant method to find eigenvalues, we need to choose a value of 1 that is close to, but greater than, \( \lambda_1 \). In this case, the user chose 1=5. This resulted in finding an eigenvalue of \( \lambda \approx 4.6129 \).
If this choice had not resulted in an eigenvalue, another search would have had to be made. It would have made sense in this case to choose a larger value, for example, \( l = 5.77 \). If this had not resulted in an eigenvalue, it would have meant that there were two or more eigenvalues in the interval \((0, 5.77)\), and the search would have to proceed “backwards” toward 0.

### 3.3.2 Computing the Eigenfunction and Testing for the Fundamental Eigenvalue

Though this choice did result in an eigenvalue, it remains to be seen whether the eigenvalue found is in fact the smallest eigenvalue. To verify this, we will compute the corresponding eigenfunction \( u(x, y) \) and check to see that it is of one sign (i.e., either always nonnegative or always nonpositive) on the interior of \( \Omega \) (we know from the boundary condition that it must be 0 on \( \partial \Omega \)). Because of the boundary condition, this is equivalent to checking that the either the maximum or minimum value of \( u(x, y) \) for \((x, y) \in \Omega \) is equal to 0. We first use “s p=2” to switch branches at the bifurcation point, and then use “r=1” to compute the corresponding eigenfunction with unit \( L^2(\Omega) \) norm.

Now that the eigenfunction has been computed, we need to check to see if it is of one sign throughout the interior of \( \Omega \). Although the PLTMG package provides no facility to check this, the present author’s modification to the TRIPLT subroutine allows the user to retrieve the maximum and minimum computed values of \( u(x, y) \) and
to check automatically whether the condition for the first eigenvalue is met. This is accomplished with a new option to the "f" command. The user enters "f ifun=20" (or simply, "f20"), and the program responds with the maximum and minimum computed values and the fact that we have indeed found the smallest eigenvalue.

If the eigenvalue found had not been the smallest, it would have been necessary to look for a smaller one. The best way to do this would be to re-initialize the PDE with the command "s ir=3,l1=3,a=1,p=0,r=0,l=0" and then continue as before, choosing as the initial guess for "1" a value slightly below the previously computed eigenvalue.

3.3.3 Graphical Output

Once the eigenfunction is computed, we may use the graphics facility of PLTMG to plot the eigenfunction. The command the user typed to generate the plot is

\[ f \text{ ifun}=0, \text{ nx}=0, \text{ ny}=1, \text{ nz}=1, \text{ ls}=2, \text{ d}=2 \]

There are many different plotting options, which are described in the documentation for PLTMG (Bank [1]). In this case, we chose to make a three-dimensional plot of the eigenfunction, (ifun=0), viewed from the point \((0,1,1)\) \((\text{nx}=0,\text{ny}=1,\text{nz}=1)\), showing level curves \((\text{ls}=2)\). The final option \((\text{d}=2)\) specifies that postscript output is desired; to view the plot on the screen one would substitute "\text{d}=1." The plot produced by this command is shown in Figure 3.6.
Figure 3.6  A three-dimensional plot, showing level curves, of the fundamental eigenfunction $z = u(x, y)$ of the Laplacian over the domain shown in Figure 3.2. The function is viewed from the point $(0, 1, 1)$. 

Domain Boundary: Star with 10 lobes
Chapter 4

Monotonicity of a Family of Starlike Domains

In this chapter, we offer a conjecture on the monotonicity of fundamental eigenvalues for a family of starlike domains and present numerical evidence to support this conjecture.

4.1 A Family of Polar Starlike Domains

The principle of domain monotonicity (see Cox and Ross [3]) states that for two open, connected, bounded domains $\Omega$ and $\Omega'$,

$$\Omega' \subset \Omega \Rightarrow \lambda_1(\Omega') \geq \lambda_1(\Omega)$$

(4.1)

Consider the one-parameter family of polar functions

$$f_n(\theta) = 2 + \sin(n\theta)$$

(4.2)

If $n$ is an integer, then $f_n(\theta)$ is a $2\pi$-periodic function of $\theta$. We will consider the family of starlike domains $\Omega_n$ bounded by functions $f_n$, where $n$ is a positive integer.

It is easy to see that for all $n$, $\Omega_n$ contains $D_1$, the disc of radius 1 centered at the origin, and is contained in a disc $D_3$ of radius 3 centered at the origin. By domain monotonicity, we have

$$\lambda_1(D_1) \leq \lambda_1(\Omega_n) \leq \lambda_1(D_3)$$

(4.3)
We also note that all domains $\Omega_n$ in the family occupy the same area, regardless of the value of $n$. To see this,

$$|\Omega_n| = \int_{\Omega_n} dy \, dx$$

$$= \int_{0}^{2\pi} \int_{0}^{f_n(\theta)} r \, dr \, d\theta$$

$$= \int_{0}^{2\pi} \frac{1}{2} (f_n(\theta))^2 \, d\theta$$

$$= \int_{0}^{2\pi} \frac{1}{2} (2 + \sin(n\theta))^2 \, d\theta$$

$$= \frac{9\pi}{2} \text{ for all positive integers } n$$

Since every domain in the family has the same area, we can compare the eigenvalues of the domains in a meaningful way.

### 4.2 A Conjecture on Monotonicity

Several members of the polar family $f_n(\theta)$ are plotted in Figures 4.1 and 4.2. Keeping in mind that the boundary condition is such that $u = 0$ on $\partial \Omega$, notice that as $n$ increases, $\Omega_n$ gradually comes to resemble a disc of radius 1, in the sense that the freedom of motion of a membrane of shape $\Omega_n$ constrained to be 0 on $\partial \Omega$ becomes more and more restricted until it resembles that of a membrane in the shape of the unit disc constrained to be 0 at the boundary.

Therefore, we conjecture that

$$n_1 < n_2 \Rightarrow \lambda_1(\Omega_{n_1}) < \lambda_1(\Omega_{n_2}) \quad (4.4)$$
Figure 4.1 The first four members of the one-parameter family of functions $f_n(\theta) = 2 + \sin(n\theta)$. 
Figure 4.2  Members of the one-parameter family of functions $f_n(\theta) = 2 + \sin(n\theta)$ for $n = 5, 10, 12,$ and $15.$
4.3 Testing the Conjecture

Using the software described in Chapter 3, we test the conjecture by numerically calculating the fundamental eigenvalue \( \lambda_1(\Omega_n) \) for all integers \( n \in 1, 2, \ldots, 15 \). The results, shown in Table 4.1, clearly display the monotonicity predicted. A plot of the results is shown in Figure 4.3.

<table>
<thead>
<tr>
<th>Number of lobes ( n )</th>
<th>Fundamental Eigenvalue ( \lambda_1(\Omega_n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.2933</td>
</tr>
<tr>
<td>2</td>
<td>1.9669</td>
</tr>
<tr>
<td>3</td>
<td>2.7035</td>
</tr>
<tr>
<td>4</td>
<td>3.3114</td>
</tr>
<tr>
<td>5</td>
<td>3.7119</td>
</tr>
<tr>
<td>6</td>
<td>3.99965</td>
</tr>
<tr>
<td>7</td>
<td>4.2029</td>
</tr>
<tr>
<td>8</td>
<td>4.3672</td>
</tr>
<tr>
<td>9</td>
<td>4.4942</td>
</tr>
<tr>
<td>10</td>
<td>4.5999</td>
</tr>
<tr>
<td>11</td>
<td>4.6897</td>
</tr>
<tr>
<td>12</td>
<td>4.7671</td>
</tr>
<tr>
<td>13</td>
<td>4.8300</td>
</tr>
<tr>
<td>14</td>
<td>4.8879</td>
</tr>
<tr>
<td>15</td>
<td>4.9397</td>
</tr>
</tbody>
</table>

Table 4.1 Fundamental eigenvalues for the family of domains given by \( \partial \Omega = f_n(\theta) = 2 + \sin(n\theta) \). Note that the eigenvalues \( \lambda_1(\Omega_n) \) are monotonically increasing as a function of \( n \).
Figure 4.3  The fundamental eigenvalue $\lambda_1(\Omega_n)$ as a function of $n$ for the family of domains given by $f_n(\theta) = 2 + \sin(n\theta)$. The horizontal lines at 0.64 and 5.76 indicate the fundamental eigenvalues for the discs of radius 1 and 3, respectively.
Appendix A

Example Functions

The following is a FORTRAN file containing the subroutine which asks the user to choose a boundary function, as well as a set of example functions. This file may be used as a model to add other functions that may be desired.

Subroutine PolarFun(title,NT, VX,VY,XM,YM,ITNODE,NV,NB,NC,IBNDRY)
Character*80 title
Integer NT
Real VX(*), VY(*), XM(*), YM(*)
Integer ITNODE(4,*), NV, NB, NC, IBNDRY(5,*)
Real f1,f2,f3,f4,f5,f6,f7,circle,star
Character*10 IntStr
Character*12 RealStr
Character*3 dumstr

C-- This Subroutine allows the user to choose which polar function to
C use to define the boundary of the domain. It then calls PolarTri
C to define the initial triangulation. All variables are as
C returned from PolarTri, except title, which is the title for the
C plots. All variables above are output to the calling program.

Integer fnumb
C --Function number choice
Real dummy
C --Dummy variable for function parameter
External IntStr, RealStr
C --Functions to convert dummy to a string
External f1,f2,f3,f4,f5,f6,f7,circle,star
C --Function names

Print *,','
Print *,'You may use any of the following polar functions',

+ to define the domain:
Print *,
Print *, 1. f1(theta) = 2 + sin(theta) ',
Print *, 2. f2(theta) = 2 + sin(2*theta) ',
Print *, 3. f3(theta) = 2 + sin(3*theta) ',
Print *, 4. f4(theta) = 2 + sin(4*theta) ',
Print *, 5. f5(theta) = 2 + sin(5*theta) ',
Print *, 6. f6(theta) = 1 + (sin(2*theta))^2 ',
Print *, 7. f7(theta) = 1 + theta*(2*PI-theta) ',
Print *, 8. f8(theta) = r (A circle of ',
+ ' radius r')
Print *, 9. f9(theta) = 2 + sin(n*theta) (A star with ',
+ ' n lobes.)'
Print *,
Print *, 'Enter the number of your choice:
Read *, fnumb
Print *,

If (fnumb .EQ. 8) Then
C --This first call to circle will query the user for the radius
dummy = circle(0.0)
Print *,
Elseif (fnumb .EQ. 9) Then
C --This first call to star will query the user for the number of lobes
dummy = star(0.0)
Print *,
Print *, 'Experience shows that a good initial number of ',
+ 'triangles is 8 times the number of lobes.'
Print *,
Endif

100 Print *, 'Enter the initial number of triangles:
Read *, NT
If (NT .LT. 2) Then
    Print *, 'You need to start with at least 2 triangles.'
    Go To 100
Endif

Go To (1,2,3,4,5,6,7,8,9), fnumb

1 title = 'Domain Boundary: f1(t) = 2 + sin(t)'
Call PolarTri(f1,NT, VX,VY,XM,YM,ITNODE,NV,NB,NC,IBNDRY)
Return
title = 'Domain Boundary:  f2(t) = 2 + sin(2t)'
Call PolarTri(f2, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

3 title = 'Domain Boundary:  f3(t) = 2 + sin(3t)'
Call PolarTri(f3, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

4 title = 'Domain Boundary:  f4(t) = 2 + sin(4t)'
Call PolarTri(f4, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

5 title = 'Domain Boundary:  f5(t) = 2 + sin(5t)'
Call PolarTri(f5, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

6 title = 'Domain Boundary:  f6(t) = 1 + (sin(2t))^2'
Call PolarTri(f6, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

7 title = 'Domain Boundary:  f7(t) = 1 + t*(2*PI-t)'
Call PolarTri(f7, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

8 title = 'Domain Boundary:  Circle of radius ' // RealStr(dummy)
Call PolarTri(circle, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

9 dumstr = IntStr(int(dummy))
title='Domain Boundary:  Star with '//'dumstr//' lobes'
Call PolarTri(star, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)
Return

200 Return
End

C=============================================================================
Real Function f1(theta)
Real theta

f1 = 2+sin(theta)

End
C=============================================================================
Real Function f2(theta)
Real theta

f2 = 2+sin(2*theta)

End

Real Function f3(theta)
Real theta

f3 = 2+sin(3*theta)

End

Real Function f4(theta)
Real theta

f4 = 2+sin(4*theta)

End

Real Function f5(theta)
Real theta

f5 = 2+sin(5*theta)

End

Real Function f6(theta)
Real theta

f6 = 1+(sin(2*theta))**2

End

Real Function f7(theta)
Real theta, PI
Data PI/3.141592653589793e0/
f7 = 1 + theta*(2*PI-theta)

End

Real Function circle(theta)
Real theta,r
Save r
Data r/0.0/

C    --This part is only executed on the first call to this function.
    If (r .eq. 0.0) then
        Print *, 'Enter radius of the circle: '
        Read *, r
    Endif

C    --This part evaluates the function, and is executed every time.
    circle = r

End

C===============================================================================

Real Function star(theta)
    Real theta
    Integer n
    Save n
    Data n/0/

C    --This part is only executed on the first call to this function.
    If (n .eq. 0) then
        Print *, 'Enter the number of lobes for the domain: '
        Read *, n
        star = n
    C    --This time, we return the number of lobes
        Return
    Endif

C    --This part evaluates the function, and is executed every time.
    star = 2+sin(n*theta)

End
Appendix B

Software Tools

This appendix contains three elements of the software tool described in this paper. The first is the FORTRAN file (star.f), which contains the function definitions required for PLTMG. They define the PDE and the boundary conditions, and give instructions for defining the domain and the triangulation.

The second is the subroutine PolarTri, which generates an initial triangulation given a polar boundary function. PolarTri is called by the subroutine gdata in star.f.

The third is a modified version of the subroutine triplt, the original of which is part of the PLTMG package. The present author's modification (clearly delineated in the comments) adds the capability to test whether the current eigenvalue is the smallest one, by checking the sign of the maximum and minimum values of the eigenfunction over the closure of $\Omega$.

B.1 Function definitions for the PDE (star.f)

C Star.f
C PLTMG routines for wave equation on starlike domains.
C By Robert A. Book, Rice University, 3-24-94
C Derived from yy.f by Fadil Santosa
C
C-----------------------------------------------
piecewise linear triangle multi grid package

edition 6.2 -- june, 1992

problem name -- star

solves \(-\Delta u = \lambda u\) in a starlike domain
\quad u = 0 \quad \text{on boundary}

real function a1xy(x,y,u,ux,uy,rl,itag,itype)

implicit real (a-h,o-z)
implicit integer (i-n)

go to (1,2,3,2,2),itype
1 a1xy=ux
   return
2 a1xy=0.0e0
   return
3 a1xy=1.0
   return
end

real function a2xy(x,y,u,ux,uy,rl,itag,itype)

implicit real (a-h,o-z)
implicit integer (i-n)

go to (1,2,2,3,2),itype
1 a2xy=uy
   return
2 a2xy=0.0e0
   return
3 a2xy=1.0
   return
end

real function fxy(x,y,u,ux,uy,rl,itag,itype)

implicit real (a-h,o-z)
implicit integer (i-n)

if(itype.eq.1) fxy=-rl*u
if(iotype.eq.2) fxy=-r1
if(iotype.eq.3) fxy=-0.0
if(iotype.eq.4) fxy=-0.0
if(iotype.eq.5) fxy=-u
return
end

c
real function gxy(x,y,u,rl,itag,itype)
c
    implicit real (a-h,o-z)
    implicit integer (i-n)
c
gxy=0.0e0
return
end

c
real function uxy(x,y,itag,itype)
c
    implicit real (a-h,o-z)
    implicit integer (i-n)
c
uxy=0.0e0
return
end

c
real function p1xy(x,y,u,ux,uy,rl,itag,itype)
c
    implicit real (a-h,o-z)
    implicit integer (i-n)
c
if(iotype.eq.1) p1xy=u*u
if(iotype.eq.2) p1xy=2.0*u
if(iotype.eq.3) p1xy=0.0e0
if(iotype.eq.4) p1xy=0.0e0
if(iotype.eq.5) p1xy=0.0e0
return
end

c
real function p2xy(x,y,dx,dy,u,ux,uy,rl,itag,itype)
c
    implicit real (a-h,o-z)
    implicit integer (i-n)
c
p2xy=0.0e0
return
end

real function qxy(x,y,u,ux,uy,rl,itag,itype)

implicit real (a-h,o-z)
implicit integer (i-n)

go to (1,2,3,4),itype
1 qxy=u
return
2 qxy=ux
return
3 qxy=uy
return
4 qxy=float(itag)
return
end

subroutine usrcmd

implicit real (a-h,o-z)
implicit integer (i-n)
integer
  indx(2)
character*6
  table(2)
character*2
  alias(2)
common /atest1/ip(100),rp(100)
common /atest2/iu(100),ru(100)
save table,alias,indx,ni,nr

data ni,nr/1,0/
data table( 1),alias( 1),indx( 1)/'domain',d',i/

cold=iu(1)
call reset(table,alias,indx,ni,nr,iu,ru)

inew=max0(1,iu(1))
inew=min0(6,inew)
if(iold.ne.inew) ip(30)=-1
return
end

subroutine gdata(title,vx,vy,xm,ym,itnode,ibndry,jb,ip,rp,iu,ru)
implicit real (a-h,o-z)
implicit integer (i-n)
integer
  itnode(4,*),ibndry(5,*),jb(*),ip(100),iu(100)
real
  vx(*),vy(*),xm(*),ym(*),rp(100),ru(100)
character*80
  title
save ispd,iprob

data ispd,iprob/1,8/
Parameter (NT=20)

if(ip(30).eq.1) then
  iu(1)=1
endif

ip(5)=ispd
ip(7)=iprob

Call PolarFun(title,NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)

10 Format (A2, A76)
Print *, ' ',
Print 10, ', ',title
Print *, 'Initial triangles: NT =', NT
Print *, ' ',

ip(1)=nt
ip(2)=nv
ip(3)=nc
ip(4)=nb
ip(5)=ispd
ip(6)=nv
ip(7)=iprob

c
pi=3.141592653589793e0
B.2 Generating the initial triangulation (PolarTri)

Subroutine PolarTri(f, NT, VX, VY, XM, YM, ITNODE, NV, NB, NC, IBNDRY)

Real f
External f
Integer NT

C --Input parameters:
C f = f(theta), polar function describing the boundary of the domain.
C NT = Number of triangles to make

Real VX(*), VY(*), XM(*), YM(*)
Integer ITNODE(4,*), NV, NB, NC, IBNDRY(5,*)

C --Output parameters:
C VX, VY = x and y coordinates of triangle vertices (dim=NT+1)
C XM, YM = x and y coordinates of midpoints of boundary arcs (dim=NT)
C ITNODE = which vertices go with which triangles (see Users' guide, p. 17)
C ITNODE(1-3,i) point to the coordinates of the vertices of triangle i;
C i.e., if ITNODE(n,i)=k, then the n-th vertex of triangle i has
C (cartesian) coordinates (VX(k),VY(k)). ITNODE(4,i) is an arbitrary
C label. (We label triangles from 1 to NT counterclockwise)
C NV = Number of vertices total (for us, NV=NT+1)
C NB = Number of boundary arcs (for us, NB=NT always)
C NC = Number of curved boundary arcs (for us, NC=NB)
C IBNDRY = which points are on which boundary arcs (see p. 17-18)
C IBNDRY(1-2,i) point to the coordinates of the endpoints of the
C boundary arcs (as in ITNODE). IBNDRY(3,i) points to the "midpoint"
C of the arc; i.e., if IBNDRY(3,i)=k then the cartesian coordinates
C of the midpoint are (XM(k),YM(k)). IBNDRY(4,i) indicates the
C boundary condition imposed on arc i. IBNDRY(5,i) is an arbitrary
C label [like ITNODE(4,i)].

Real theta, ft, PI
Integer i

C --Local Variables

PI = 4*ATAN(1.0)
C--In our scheme, every triangle has a boundary arc
   NB = NT
C ...And there is vertex per triangle, plus the center
   NV = NT+1
C ...And all arcs are curved
   NC = NB

C--Set up initial points at origin and first point on boundary
   VX(1) = 0
   VY(1) = 0
   VX(2) = f(0.0)
   VY(2) = 0

C--Find remaining points along boundary
   Do i=1,NT-1
   C    --Find vertex
         theta = (2*PI/NT)*i
         ft = f(theta)
         VX(i+2) = ft*cos(theta)
         VY(i+2) = ft*sin(theta)
   Enddo

C--Set up ITNODE table with vertices of each triangle
   Do i=1,NT
   C    --Every triangle has the origin as a vertex
       ITNODE(1,i) = 1
   C    --Every triangle has two consecutive outer points as vertices
       ITNODE(2,i) = i+1
       ITNODE(3,i) = i+2
   C    --Make the "arbitrary label" the sequential number of the triangle
       ITNODE(4,i) = i
   Enddo
   C    --Correct the third vertex of the last triangle
       ITNODE(3,NT) = 2

C--Find "midpoint" of each arc and set up IBNDRY table
   Do i=1,NB
   C    --Find coordinates of "midpoint"
       theta = (2*PI/NT)*(i-0.5)
       ft = f(theta)
       XM(i) = ft*cos(theta)
       YM(i) = ft*sin(theta)
   C    --Enter pointers to arc endpoints (VX,VY) in IBNDRY
IBNDRY(1,i) = i+1
IBNDRY(2,i) = i+2

C --Enter pointer to "midpoint" (Xm,Ym) in IBNDRY
   IBNDRY(3,i) = i

C --Set Dirichlet boundary conditions on each arc
   IBNDRY(4,i) = -1

C --Set arbitrary label to sequential number of arc
   IBNDRY(5,i) = i

Enddo

C --Correct the second endpoint of the last arc
   IBNDRY(2,NB) = 2

End

B.3 Modified version of TRIPLT

piecewise linear triangle multi grid package

    edition 6.2 -- march, 1992

    Revised by R.A. Book, April 1994

subroutine triplt(title,qxy,smin,smax,ip,w)

    implicit real (a-h,o-z)
    implicit integer (i-n)

    integer
       +   ip(100),jp(25),jp1(25)
    real
       +   w(*),q(3,3),t(25),tl(25)
    character*80

    title
    external qxy

    i      jp(i)  t(i)  tl(i)
    1      nt    xshift   xshift
    2      nv    yshift   yshift
    3      nc    scale   scale
c 4    icont    zratio    zratio

4 5    ncolor

c 6    ifun    lambda    lambda

6 7    llen    eps    eps

c 8    tlen    xl    xl from t

8 9    mntag    xr    xr from t

9 10    mag    yb    yb from t

10 11    ix    yt    yt from t

11 12    iy    rmag    1.0e0

12 13    nx    fmin    fmin

13 14    ny    fmax    fmax

14 15    nz    rmag from t

15 16    nshade    size    size

16 17    mxcolr    xs1    xs1

17 18    maplen    ys1    ys1

18 19    iscale    xs2

19 20    lines    ys2

20 21    numbrs    good    good

21 22    i3d    fair    fair

22 23    isclsw    poor    poor

23 24    worst    worst

24 25    average    average

25

array pointers...in the order that they
occur in the w array

c
jtime=ip(81)
jhist=ip(82)
jpath=ip(83)
jusr=ip(84)
iuu=ip(85)
iudot=ip(86)

iu0=ip(87)
iu0dot=ip(88)
iur=ip(89)
iul=ip(90)
itri=ip(91)
ivert=ip(92)

iee=ip(93)

ibb=ip(94)

iz=ip(95)

c

c
additional pointers
do 1 i=1,25
     t1(i)=0.0e0
     t(i)=0.0e0
     jp(i)=0
     jpl(i)=0
1   t(13)=smin
t(14)=smax
maxt=ip(39)
maxv=ip(40)
mxt=2*maxv
nc=ip(3)
mxcolr=max0(2,ip(72))
ired=iz
igreen=ired+mxcolr
iblue=igreen+mxcolr
icxy=iblue+mxcolr
ivx=icxy+3*nc
ivy=ivx+maxv
ivz=ivy+maxv
itnode=ivz+mxt
itedge=itnode+3*mxt
itag=itedge+3*mxt
label=itag+mxt
iord=label+mxt
jord=iord+mxt
nblo= jord+mxt
list= nblo+mxt

these are overlapping work arrays:

c level/err/tlist/iorder in tplot/cplot/bblock/torder
c nblo/tlist/z/map/rad/rad in torder/bblock/plinit/dsinit/cplot/tplot
c cen/list in bblok/torder
c
irad=nblo
ilevel=jord
itlist=jord
ierr=jord
ivw=jord
jp(8)=mxt
izz=nblo
map=nblo
c
icen=list
llen=ip(16)-list+1
if(llen.lt.maxt) go to 10
jp(7)=llen

initialize data structures

call dsinit(ip,w(itri),w(ivert),w(jusr),w,
+    jp,t,w(itnode),w(itedge),w(itag),w(ilevel),
1    w(label),w(map),w(ivx),w(ivy),w(icxy),w(ivz),w(iee))

initialize q, compute element order is necessary

call torder(jp,w(itnode),w(itedge),w(iord),w(jord),w(nblock),
+    w(list),w(itlist),w(ivx),w(ivy),w(icen),q,iflag)

ip(34)=iflag
if(iflag.ne.0) return

ifun=jp(6)
ivu=iuu
if(ifun.eq.1.or.ifun.eq.13) ivu=iudot
if(ifun.eq.2.or.ifun.eq.14) ivu=iur
if(ifun.eq.3.or.ifun.eq.15) ivu=iul
if(ifun.eq.4.or.ifun.eq.16) ivu=ibb
if(ifun.eq.6) then
    nt=jp(1)
do 2 i=1,nt
2    w(ierr+i-1)=w(ivz+i-1)
endif

call plinit(jp,w(itnode),w(itedge),w(itag),w(ilevel),w(ierr),
+    w(ivx),w(ivy),w(ivz),w(ivw),w(icxy),w(ivu),w(izz),q,t,tl,
1    jpl,qxy)

C=================================================================
C---New Routine to test for the smallest eigenvalue

IF (ip(66) .EQ. 20) THEN
   PRINT *, 'Eigenfuntion has minimum value: ',t(13)
   PRINT *, ',', and maximum value: ',t(14)
   PRINT *, ','
IF (sign(1.,t(13))*sign(1.,t(14)) .NE. -1.) THEN
  PRINT *, 'The current value of lambda is the smallest ',
  'positive eigenvalue.'
ELSE
  PRINT *, 'Look for a smaller eigenvalue.'
ENDIF
PRINT *, ''
RETURN
ENDIF

C=====================================================================
call clrmap(w(ired),w(igreen),w(ibleue),jp)
c
call pltut1(jp(18),w(ired),w(igreen),w(ibleue))
call title0(title,0)
jp(34)=0
c
triangle plots
c
if(ifun.ge.7.and.ifun.le.11) then
  if(ifun.ge.9) then
    call legnd2(jp,tl)
  else
    call legnd1(jp,tl)
  endif
  call tplot(jp1,w(itnode),w(itedge),w(ilevel),w(label),
     + w(itag),w(ivx),w(ivy),w(icx),q,tl,w(irad))
  call legnd0(tl)
  call tplot(jp,w(itnode),w(itedge),w(ilevel),w(label),
     + w(itag),w(ivx),w(ivy),w(icx),q,t,w(irad))
endif
c
color plot
c
if(ifun.lt.7) then
  call legnd4(jp,tl,w(ierr))
  call cplot(jp1,w(itnode),w(itedge),w(itag),w(iord),
     + w(ierr),w(ivx),w(ivy),w(ivz),w(icx),w(iuu),
     1 w(label),w(irad),q,tl,qxy)
  call legnd0(tl)
  call cplot(jp,w(itnode),w(itedge),w(itag),w(iord),
     + w(ierr),w(ivx),w(ivy),w(ivz),w(icx),w(iuu),
     1 w(label),w(irad),q,t,qxy)
endif

c
c    gradient plot

c     if (ifun.gt.11) then
        call legnd3(jp,tl)
        call vplot(jp,w(itnode),w(itedge),w(ivu),w(label),
        + w(itag),w(ivx),w(ivy),w(ivz),w(ierr),w(icxy),
        1 q,tl,w(irad),qxy)
        call legnd0(tl)
        call vplot(jp,w(itnode),w(itedge),w(ivu),w(label),
        + w(itag),w(ivx),w(ivy),w(ivz),w(ierr),w(icxy),
        1 q,t,w(irad),qxy)
    endif

    call pltut1(-1,w(ired),w(igreen),w(iblue))

C-- Before returning, record fmin and fmax where they will not get lost.
IP(98)=T(13)
IP(99)=T(14)

return

ip(34)=-1
return
end
Appendix C

Sounding the Fundamental Frequency

Once the first eigenvalue, or fundamental frequency of vibration, has been found using the tools described in this paper, it is possible to use Matlab (with the appropriate hardware) to produce a sound at the fundamental frequency. A Matlab function to accomplish this is shown below.

Note that this function does not necessarily accurately reproduce the true sound of a vibrating drum or membrane of shape $\Omega$, since it does not take into account the harmonics (corresponding to higher eigenvalues) which are present in a real vibrating drum. However, this function can be used to give some idea of the relative difference in the fundamental eigenvalues of the Laplacian on different domains, and, with suitable adjustments for relative amplitudes, be adapted to simulate harmonics as well.

function y = play(freq)

% This function plays a sound at frequency 'freq' for approximately
% one second. The duration of the sound may be changed by setting the
% value of 'duration' below. (The default duration=1000 corresponds to
% approximately one second.) No value is returned.
% By Robert Book, Rice University, 4-28-94

duration = 1000 ;
w = sin(freq*(0:0.1:duration)) ;
sound(w) ;
Bibliography

