ENGI2211: Solving Laplacian Eigenfunctions and their associated features (Junya Ogawa)

Part 1: Solving Laplacian Eigenfunctions in one dimension (OneDLEs)

The code OneDLEs solves a one-dimensional Laplacian eigenvalue problem. More specifically, it solves a one-dimensional elastic string with length l subdivided into n divisions, with uniform density ρ, tension T and eigenvalues λ. The string can be modelled via the wave equation [1] and then its constant can be equated to the eigenvalue λ.

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\rho}\frac{\partial^2 y}{\partial x^2} = \lambda \frac{\partial^2 y}{\partial x^2}$$

To numerically solve this problem, a finite difference method for Partial Differential Equations (PDEs) was employed to obtain a set of values for the wave function y. The variable h was first defined as the ratio l/n. Secondly, the code creates a blank array y filled with zeroes, the tridiagonal matrix M and the array f which was evaluated to be $-h^2$ on all columns. The PDE was then solved for y using MATLAB's backslash operator.

After which, the eigenvalues were evaluated via adapting some lines of code from a MATLAB blog [2] which features code that solves Laplacian eigenvalue problems on 2-dimensional shapes. The diagonal matrix M replaces the delsq(G) in the blog as M represents the finite difference Laplacian and the eigenvalues were obtained using MATLAB's eigs function. The following eigenvalues were obtained for different values of n and l as shown below, as compared to their theoretical values given by the formula below [3] to 3 significant figures:

	Computed Values				Theoretical values			
	λ_1	λ_2	λ_3	λ_4	λ_1	λ_2	λ_3	λ_4
L = 1, n = 20	8.14	32.4	72.4	127	9.87	39.5	88.8	158
L = 1, n = 40	8.95	35.7	80.2	142	9.87	39.5	88.8	158
L = 2, n = 20	2.04	8.10	18.1	31.7	2.47	9.87	22.2	39.5
L = 2, n = 40	2.24	8.94	20.1	35.5	2.47	9.87	22.2	39.5

Table 1: Table of results for OneDLEs contrasted with theoretical results

$$\lambda_k = \left(\frac{k\pi}{l}\right)^2, \qquad k \in \mathbb{N}$$

Upon analysis, it would seem that the computational values seem somewhat accurate in comparison to the theoretical values. As n increased, the accuracy of the eigenvalues increased. For example, for the case of $L = 2$, the computational value of λ_1 got closer (from 2.0357 to 2.2370) to its theoretical value of 2.4674 as n increased from 20 to 40. This is because as n increases, there are more nodes in which to do calculations from on the finite difference method matrices. Thus, this allows the code to solve the eigenvalue problem more accurately (i.e., obtain greater and more accurate readings of y) and consequently obtain better eigenvalues.

Part 2: Solving Laplacian Eigenfunctions in 2 Dimensions (TwoDLEs)

The code TwoDLEs expands upon the concept of solving eigenvalues with an extension to 2 dimensions. More specifically, it solves the Laplacian eigenfunction problem on a 2-dimensional rectangle plate with lengths l_x by l_y, subdivided by n and m in the x and y axes respectively. Another rectangle of dimensions l_{xx} by l_{yy} was cut out from one of its corners, then rotated clockwise by theta and flipped on its diagonal based on its turn.

To solve this problem, code from [2] was adapted. Steps were taken to first determine if either l_{xx} or l_{yy} were 0 (to form a rectangle with no cut-outs; lines 4-11) but if not, the plate vertices were flipped dependent on its turn (as illustrated on the handout; lines 13-19). The vertices were then translated vertically and horizontally (lines 25-40), followed by a multiplication with the clockwise rotation matrix by the angle theta (lines 42-46) such that the bottom left corner ended up at the origin.

This was followed by largely the same code from [2], with a few edits. Firstly, the mesh grid was flipped dependent on certain conditions (lines 52-56) and the subdivision lengths h_x and h_y were determined by l_x/n and l_y/m respectively (lines 49-50). The number of inner points variable (inpoints) was also edited to count the number of ones in the in logical (lines 60-61). This was then followed by editing the G matrix for Neumann boundary conditions via extending the border indices within the G matrix in all 4 directions towards their respective adjacent zero rows/columns (lines 70-77). Finally, the discrete Laplacian A was edited such that it was now $delsq(G)/(h_x h_y)$ to reflect the fact that $h_x \neq h_y$ in some cases.

A table was then obtained, given the base parameters of $l_x = 1, l_y = 2, n = 20$ and $m = 40$:

l_{xx}	l_{yy}	BCs	theta	Turn	λ_1	λ_2	λ_3	λ_4
0	0	“Dirichlet”	0	Right	12.3	19.7	32.0	41.6
0	0	“Neumann”	0	Right	12.3	19.8	32.1	41.7
0	0	“Dirichlet”	90	Right	12.3	19.7	32.0	41.6
0.6	1.6	“Dirichlet”	0	Right	57.9	65.3	74.9	84.9
0.6	1.6	“Dirichlet”	90	Right	57.9	65.3	74.9	84.9
0.6	1.6	“Dirichlet”	180	Right	57.9	65.3	74.9	84.9
0.6	1.6	“Dirichlet”	270	Right	57.9	65.3	74.9	84.9
0.6	1.6	“Dirichlet”	180	Left	57.9	65.3	74.9	84.9
0.6	1.6	“Dirichlet”	270	Left	57.9	65.3	74.9	84.9

Table 2: Table of obtained eigenvalues of TwoDLEs for given parameters.

A few things can be noted from the results of the table. Firstly, it would seem that neither theta nor turn has no bearing on the eigenvalues yielded. This is because the plate remains functionally the same even during rotation or flipping, with no changes to its dimensions. Secondly, it seems that the Neumann boundary conditions only very slightly increased the eigenvalues, from an eigenvalue set of 12.3, 19.7, 32.0, 41.6 to a set 12.3, 19.8, 32.0, 41.7. This indicates that the implementation of the Neumann code was not particularly effective, as it should be noted that the first eigenvalue of the eigenvalue problem under Neumann boundary conditions should equal to zero [3]. This would then follow by the Dirichlet boundary condition’s first, second then third eigenvalue, giving an eigenvalue set of 0, 12.3, 19.7, 32.0.

The effect of the base parameters (l_x, l_y, n, m) were then observed to see their effect on the eigenvalues. Firstly, it would seem that whenever the length parameters were increase, the eigenvalues would decrease and vice versa. This could be explained by the following formula [4] to analytically calculate the eigenvalues of a l_x by l_y rectangular plate:

$$\lambda_{p,k} = \pi^2 \left(\frac{p^2}{{l_x}^2} + \frac{k^2}{{l_y}^2} \right), \qquad p, k \in \mathbb{N}$$

Furthermore, it would seem that by increasing n or m, the eigenvalues approached the values as determined by the above formula. The reasoning behind is also largely similar to that in Part 1, except the shape has been extended to 2 dimensions and increasing n or m creates a finer mesh in lines 52-57 of the code, which allows the programme to extract more nodes for computation of the eigenvalues and thus gain a greater accuracy.

Part 3: Obtaining Eigenvalue Features of given shapes in 2 Dimensions (EigFeatures)

The code EigFeatures calculates the topological features F1, F2, F3 and F4 using a shape's eigenvalues to be able to determine how well eigenvalues are able to recognise shapes. More specifically, it calculates the features [5] as shows:

$$F1 = \left\{\frac{\lambda_1}{\lambda_2}, \frac{\lambda_1}{\lambda_3}, \frac{\lambda_1}{\lambda_4}, \dots \frac{\lambda_1}{\lambda_n}\right\}$$

$$F2 = \left\{\frac{\lambda_1}{\lambda_2}, \frac{\lambda_2}{\lambda_3}, \frac{\lambda_3}{\lambda_4}, \dots \frac{\lambda_{n-1}}{\lambda_n}\right\}$$

$$F3 = \left\{\frac{\lambda_1}{\lambda_2} - \frac{d_1}{d_2}, \frac{\lambda_1}{\lambda_3} - \frac{d_1}{d_3}, \frac{\lambda_1}{\lambda_4} - \frac{d_1}{d_4}, \dots \frac{\lambda_1}{\lambda_n} - \frac{d_1}{d_n}\right\}$$

$$F4 = \left\{\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{2\lambda_1}, \frac{\lambda_4}{3\lambda_1}, \dots \frac{\lambda_{n+1}}{n\lambda_1}\right\}$$

Where the set $\{d_1, d_2, d_3, \dots d_n\}$ denotes the first n eigenvalues of a disk [5]. F3 represents the distance from the shape on a domain to the disk with eigenvalues $\{d_1, d_2, d_3, \dots d_n\}$ [5].

To achieve this, code from TwoDLEs was recycled, with the exception of now generating vertices creating shapes such as a 4 x 6 rectangle (lines 8-10), base 4 and hight 3 right triangle (lines 11-13), diamond/rhombus with horizontal diagonal 4 and vertical diagonal 6 (lines 14-16), a 4-square (lines 17-19), and a disk with centre at origin and radius 4 (lines 21-29). The code also automatically calculated the first 20 eigenvalues $\{d_1, d_2, d_3, \dots d_{20}\}$ from a unit circle at origin for use in the feature F3 (lines 42-59). The first 20 eigenvalues of the main shape were then calculated (lines 33-40, 65-69) and its values used in the formulae mentioned above via iterative processes (lines 71-95) to obtain 4 arrays with 20 values each. n and m were also kept at a constant 40 and 60 respectively.

To give an example of the feature values, in the rectangular base case, the values of F1 dropped from 0.520 to 0.0641 with an initial sharp drop to 0.326 but remained relatively steady later. F2 featured erratically fluctuating values that seemed to trend upwards from 0.520 towards 1 with an initial sharp increase towards 0.889. F3 contained values between 0.124 and -0.0707, which also seemed to fluctuate but these values also tended towards 0 as the number of eigenvalues increased. Finally, the F4 trend seemed similar to that of F1, except starting at 1.922 and decreasing towards 0.780.

To compare this with the other shapes created, it seems that a lot of these established patterns remained consistent irregardless of shape. For example, the feature set F1 always showed an overall decreasing trend, F2 an upward trend towards 1, F3 a fluctuation towards a near-zero value, and F4 a scaled version of F1. For the different shapes, F1 had a range of 0.075 to 0.5 for triangle, 0.45 to 0.075 for diamond, 0.520 to 0.075 for square and 0.06 to less than 0.5 for the circle. For F2, values started from less than 0.5 for triangle, diamond and circle, but greater than 0.5 for square. F3 fluctuated to a value less than 0.02 for the triangle, less than 0.01 for the diamond, towards 0 for the square and towards -0.0015 for the circle. For F4, triangle had a range of marginally less than 2 to greater than 0.6, a range of less than 0.8 to near 2.2 for diamond, marginally less than 0.8 to 2 for square and 0.8 to greater than 2 for circle. It should also be noted that the for the feature sets, the square and the rectangle were the most similar. The feature set that stood out was of circle's F3 as it had a very small magnitude throughout. This could be because the F3 feature set is a difference of the features vs. a disk's features. As the features are expected to be scale invariant (i.e., they do not change with size, [3]), this could explain the relatively small values.

Part 4: Using the Kmeans algorithm to sort features from EigFeatures (kmeansAcc)

The code kmeansAcc calculates the accuracy of using the kmeans algorithm to sort a set of features to their respective shapes. More specifically, it sorts a selected number (FN) of features from each feature set and sorts them to what the algorithm thinks the shape will be and assigns an index (1 or 2) to it via the following method [6]:

1) Compute sum of squared distance between data points and all centroids.
2) Assign each data point to closest centroid.
3) Compute centroids for clusters by taking the data point average of each cluster.
4) Initialisation via shuffling and selecting number of clusters (K) for centroids without replacement.
5) Reiteration until centroids do not change.

To achieve this, 100 different shapes were prepared: first, 30 different squares were prepared from a side length of 1 to 30. The feature sets were then saved in a 30 x 80 matrix (F1234sq) via running the EigFeatures code in an iterative loop (not shown), where the rows contain the data via size and the columns via their feature set. 70 different right triangles of lengths 2 to 71 were also prepared similarly, with their data saved on a 70 x 80 matrix (F1234tri). The data was then loaded onto the programme and appended to form a 100 x 80 matrix (X), the F1234sq matrix lying on the F1234tri matrix. The columns were then extracted to form 4 matrices of size 100 x FN from each feature set and appended together to form a new smaller matrix (x) of size 100 x 4FN. The kmeans algorithm was then run for 2 clusters and the resulting vector (idx) was compared against a constructed real index with 30 ones and 70 twos. The values were logically checked for equivalency and counted, and the resulting number was divided by a 100 to give a value for accuracy.

The results produced by this code seemed highly irregular. For example, the code only yielded accuracy results of 0 or 1 irregardless of the KN value. When further inspection was conducted, it seemed that when the accuracy was 0, in virtually all instances this was when the algorithm had switched 2 and 1 as indexes such that 2 went first then 1, ensuring that in virtually all instances the accuracy was 1. This could be because of the innate problems of generating eigenvalues for 100 different shapes, especially those with wildly differing sizes. It seemed that within the two F1234sq and F1234tri matrices, the feature values had remained consistent when the iteration index (used to generate the size) had reach a value of about 10. This could be a computational limitation of either MATLAB or the current computer being used to run the code as usually these problems do not happen.

Sources

[1] Georgia Tech University, "The Wave Equation," n.d.. [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Waves/waveq.html. [Accessed 27 April 2023].

[2] C. Moler, "Can One Hear the Shape of a Drum? Part 1, Eigenvalues," 6 August 2012. [Online]. Available: https://blogs.mathworks.com/cleve/2012/08/06/can-one-hear-the-shape-of-a-drum-part-1-eigenvalues/. [Accessed 27 April 2023].

[3] N. Saito, "MAT 207B Lectures 26, 27, 28," University of California (Davis), Davis, 2018.

[4] N. Saito, “MAT 280: Laplacian Eigenfunctions: Theory, Applications, and Computations. Lectures 12+13,” University of California (Davis), Davis, 2007.

[5] M. B. H. Rhouma, M. A. Khabou and L. H. Hermi, Advances in Imaging and Electron Physics, Amsterdam: Elsevier Inc., 2011.

[6] I. Dabbura, “K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks,” 17 September 2018. [Online]. Available: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. [Accessed 27 April 2023].