Scientific Computing: Set III

Kasper Nicholas (10678859) & Steven Raaijmakers (10804242)

I. Introduction

A famous question related to the wave equation problem is whether the membrane shape can be identified uniquely from its eigenvalue spectrum. This has been extensively researched in recent history, with both numerical and practical experiments and was shown to be false. In this report we wish to further substantiate these claims with our own numerical experiments using three different membrane shapes of varying size. Contrary to the two previous reports we solve the wave equation by means of direct methods and not iterative methods. This requires the rewriting of the equation in matrix notation. Moreover, the time-dependent equation and solution are split into a time-dependent and space-dependent function. We first solve the system without its time-dependent part and thereafter combine the two for a complete assessment of the system.

The second part of the report focuses on the use of direct methods for solving the diffusion equation. The results of the numerical experiments are subsequently presented and finally discussed in order to asses the validity of our results with respect to the recent conclusion that membranes of different shapes cannot be identified uniquely from its eigenvalue spectrum. Moreover, we wish to assess the accuracy and efficiency of direct methods for solving partial differential equations in matrix notation.[1]

II. Theory

A. The Two-Dimensional Time-Dependent Wave Equation

The time-dependent wave equation with two spatial dimensions is governed by the following partial differential equation (PDE):

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u, \tag{1}$$

where u denotes the solution of the wave and c is a constant that determines the wave velocity. The partial derivatives of x and y are represented by ∇. The boundary condition of our wave equation is $u = 0$, irregardless of the shape of the boundary. The solution to this PDE can be separated in two independent functions, one dependent on the time t and one dependent on spatial dimensions x and y:

$$u(x, y, t) = v(x, y)T(t). \tag{2}$$

The complete solution u, dependent on all three variables, cannot always be separated in this manner. We specifically look for a u of this form due to its several advantageous properties when the solution is obtained with computational methods. These properties, however, are discussed in the next section in more detail. The derivation of such a separated solution with the rewriting of (1) in accordance with the separated solution in (2) and get:

$$\frac{1}{T(t)} \frac{\partial^2 T(t)}{\partial t^2} = c^2 \frac{1}{v(x, y)} \nabla^2 v(x, y), \tag{3}$$

where the t-dependencies are moved to the left and all the x- and y-dependencies are moved to the right. Hence, both sides of the equation must equal some constant K independent of x, y, and t. For the left-hand side this results in:

$$\frac{\partial^2 T(t)}{\partial t^2} = K c^2 T(t). \tag{4}$$

If $K < 0$ we obtain an oscillating solution:

$$T(t) = A \cos(c\lambda t) + B \sin(c\lambda t), \tag{5}$$

where $\lambda^2 = -K$. If $K > 0$, the solution either grows or decays exponentially and thus leads to trivial solutions that are of little interest. Moreover, if $K = 0$ the solution is a constant or a linear function of x and y, indicating $v = 0$ due to the specified boundary condition of (1). Hence, we can assume $\lambda > 0$ without a loss of generality.

Conversely, the right-hand side of (3) with a constant K is:

$$\nabla^2 v(x, y) = K v(x, y). \tag{6}$$

This can be rewritten in the form $Mv = Kv$, where M is a matrix, v an vector of $v(x, y)$ at the grid points and K is a scalar constant. Boundary conditions are $v(x, y) = 0$ on the boundary, i.e. the boundary is fixed. We now recognize (6) as an eigenvalue problem. The solution give the eigenmodes $v(x, y)$ and eigenfrequencies λ.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Fig. 1: A square grid for $N = 4$.

B. Two-Dimensional Time-Independent Diffusion

The time-independent diffusion equation with two spatial dimension is governed by the following PDE:

$$\nabla^2 c = 0, \tag{7}$$

where ∇ again denotes the partial space derivatives with respect to x and y, and c is the concentration at (x, y). This is also known as the Laplace equation. As for the wave equation, we wish to rewrite this PDE as a matrix equation of the type $M \cdot c = b$. In other words, a matrix is constructed for ∇^2. The boundary condition of the equation is $c = 0$ and a source is present with $c = 1$.

C. Finite Difference Methods in Matrix Notation

In order to numerically implement PDE's and use direct methods to obtain their solutions we are required to reconsider our spatial grid of x and y as a finite difference mesh. In Fig. 1 this mesh is shown for a four by four grid. Each point on the grid is connected to four neighbours: up, down, left and right. The outer points, however, have only three neighbours, or even two if they are in the corner of the grid. Interior grid points are indicated by:

$$(x_i, y_j) = (i\Delta x, j\Delta y), \qquad i, j = 1, ..., N.$$

We consider $\Delta x = \Delta y = L/N$, where $N = 4$ in the example of Fig. 1. If we consider the spatial component of the wave equation in (6), we can replace the derivatives by the centered difference approximation at each interior mesh point to obtain the finite difference equation:

$$\begin{aligned} &\frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{(\Delta x)^2} + \\ &\frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{(\Delta x)^2} \approx \nabla^2 v(x, y), \end{aligned} \tag{8}$$

where $v_{i,j}$ is an approximation to the true solution $v(x_i, y_j)$ for $i, j = 1, ..., n$ and represents one of the given boundary values if i or j is 0 or $n+1$. Simplifying and writing out the resulting 16 equations for $N = 4$ explicitly gives:

$$\begin{aligned} &-4v_{1,1} + v_{0,1} + v_{2,1} + v_{1,0} + v_{1,2} \\ &-4v_{2,1} + v_{1,1} + v_{3,1} + v_{2,0} + v_{2,2} \\ &\qquad\qquad \vdots \\ &-4v_{3,4} + v_{2,4} + v_{4,4} + v_{3,3} + v_{3,5} \\ &-4v_{4,4} + v_{3,4} + v_{5,4} + v_{4,3} + v_{4,5}, \end{aligned}$$

which can be rewritten in matrix form as $M \cdot v$. Note that we leave out the factor $1/(\Delta x)^2$ for the time being to indicate the logic of subsequent matrix entries. This factor is multiplied with the matrix later thereafter. The corresponding matrix is $N * N = 4 * 4$. It is constructed by iterating through each point on our grid and checking if it has neighbours on the grid. If this is the case, the corresponding neighbour's index from Fig. 1 is used to determine the column index in the current of the current cell and is denoted as 1. In other words, each row represents one of the grid cells, and with the column entries we can determine which are its neighbours by means of Fig. 1.

We further note in our equations that some points have neighbours with $i > 4$ or $j > 4$. Because they fall outside the considered grid for $N = 4$ we consider these to be zero as the solution does not exist there. The equations for the example in matrix notation is:

$$\frac{1}{(\Delta x)^2} \begin{pmatrix} -4 & 1 & 0 & 0 & 1 & 0 & \cdots & \cdots & 0 \\ 1 & -4 & 1 & 0 & 0 & 1 & 0 & & \vdots \\ 0 & 1 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 1 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 1 \\ 0 & 1 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & \cdots & 0 & 1 & 0 & 0 & 1 & -4 \end{pmatrix}, \tag{9}$$

and v is denoted by:

$$(v_{1,1} \quad v_{1,2} \quad \cdots \quad v_{3,4} \quad v_{4,4})^T. \tag{10}$$

We have now defined our eigenvalue problem as $M \cdot v = Kv$. For more complex problems, i.e. $N >> 4$, we would still only have a maximum of five entries in this matrix, making it symmetric and very sparse and making subsequent numerical computations faster and cheaper in terms of time and storage. By means of numerical solvers in Python the eigenvalues K and

corresponding eigenvectors v can be obtained. It should be noted that this example and the corresponding matrix notation is valid for square grids. If we change the shape of the considered grid the matrix can become a-symmetric and less sparse, and thus complicating numerical computations.

III. Methods

In the two previous reports we discussed various iterative or indirect methods to numerically implement the discretized PDE's. In this report, however, we wish to focus on direct methods. This requires the PDE's to be formulated with matrix notation as is shown with a small example in the previous section.

A. The Two-Dimensional Wave Equation

The wave equation is first numerically implemented for two spatial dimensions as it is represented in (6). The example in the previous section on the matrix notation of finite difference equations allows us to extrapolate this easily to larger N problems. We consider a square grid for the number of discretization steps $N = 50$ and length $L = 1$. The size of the steps are again equal to:

$$\Delta x = \frac{L}{n} \quad \text{and} \quad \Delta y = \frac{L}{m}. \tag{11}$$

The matrix can now be calculated for $n = m = N = 50$ with Algorithm 1.

Algorithm 1 *get_m(n,m)*

```
Initialize square matrix of size n * m.
for a cell (i) in range(n*m):
    if cell has right neighbour:
        M[i][i+1]=1.
    if cell has left neighbour:
        M[i][i-1]=1.
    if cell has top neighbour:
        M[i][i-m]=1.
    if cell has right neighbour:
        M[i][i+m]=1.
    M[i][i]=-4.
return M
```

Note that the diagonal entries are always set to -4, irregardless of neighbour configurations. Furthermore

The function *scipy.linalg.eig()* is initially implemented for solving the eigenvalue problem $M \cdot v = Kv$ due to its ability to solve for non-symmetric and non-Hermitian square matrices. This is important when different grid shapes are implemented and the subsequent matrix changes while still allowing for the eigenvalues and eigenvectors to be found. A subsequent implementation of the function scipy.sparse.linalg.eigs() allows for the solving of eigenvalue problems for sparse matrices.

B. Eigenmodes of drums or membranes of different shapes

We are not only interested in the wave equation on square grids. Instead we wish to also compute the equation in rectangular grids with width L and height $2L$. The matrix notation for this grid can still be calculated with algorithm 1. Moreover, we wish to also consider circular shapes and thus require a new method to obtain the matrix notation of the wave equation due to the change in boundary. This is done, for $L = 1$, in algorithm 2. The function *get_circle()* first creates a discrete mask for a circle on a square grid with width and height N. Thereafter it checks whether the neighbours of each cell are within this mask. Only then is this translated to a matrix M with the current cell index, again with the numbering system according to Fig. 1, equal to -4 and neighbouring cells within the circular grid equal to 1. Both the circular and rectangular boundary matrices and corresponding eigenvalue problems are solved with *scipy.linalg.eig()* because we are dealing with non symmetric and less sparse matrices than in the square grid case.

Algorithm 2 *get_circle(N,L)*

```
Initialize a mask of (N, N).
Find middle of grid M = (N-1)/2.
for point (x, y) in (N,N):
    if grid point is within L/2 from M:
        Add point (x,y) to mask.
Initialize square matrix M of size N^2
for point (x, y) in mask:
    Detetermine neighbours of point (x, y)
    if neighbour is within (N, N):
        Add neighbour point to candidates.
    if candidates c(x, y) are in the mask:
        M[x*N+y][c.x*N+c.y]=1.
    M[x*N+y][x*N+y]=-4.
return mask, M
```

C. Adding the Time-Dependent Function

In order to consider the time-dependent wave equation we are required to calculate the time-dependent function $T(t)$ according to (2). We take the constants A, B and c to be equal to 1. After the space-dependent solution $v(x, y)$ is determined according to methods discussed in this section we use the obtained λ values to complete the corresponding $T(t)$. In order to calculate the solution

over time we take 100 values $t \in [0, 2\pi]$. For an increasing t the resulting grid is animated. This concludes the implementation of the wave equation with the time-dependent function $T(t)$ and space-dependent function $v(x, y)$

D. Direct method for Steady State Diffusion

In the previous sets we formulated iterative methods to find to steady state of the diffusion equation. In this section we elaborate on a direct method, where a matrix M is constructed for ∇^2 in (7). This matrix adheres the linear system $M \cdot c = b$ and is calculated according to the method in algorithm 3. As parameter values we take $N = 100$, $L = 1$ and the radius for the circular mask equal to 2. The source of the diffusion is located at $(0.6, 1.2)$, with $c(x, y) = 1$ here and zero elswhere. For a theoretical example with a square grid we wish to refer to the system in Fig. 1. The vector b is derived by reshaping the grid with the diffusion source noted as 1 and all other points are denoted by 0, i.e. b has a length of N^2. Once we have obtained the sparse matrix M and sparse vector b we can finally calculate the concentration $c(x, y)$ in the steady state of our diffusive system. This is done by feeding M and b into the solver function *scipy.sparse.linalg.spsolve()*, specifically used for solving sparse linear systems.

Algorithm 3 Diffusion with circular domain.

```
Create a matrix M with get_m(N.N).
Create a circular mask with get_circle(N,N).
Determine the diffusion source on new grid at (N/2, N/2) equal to 1.
for each point (i, j) on grid:
    if source[i][j] is 1:
        M[i*N+j]=0.
        M[i*N+j][i*N+j]=1.
    if mask[i][j] is 0:
        M[i*N+j]=0.
        M[i*N+j][i*N+j]=1.
b is obtained by reshaping the source grid matrix to a vector.
c is obtained with a sparse linear system solver.
```

IV. Results

A. Eigenmodes of drums or membranes of different shapes

We have calculated the eigenvectors and eigenvalues for various systems. For all these systems we take $L = 1$, and $N = 50$. In other words, we used 50 discretization steps to represent a continuous length of 1. We have used a system of a square shape (LxL), a system of a rectangle (Lx2L) and a system of a circle with radius L.

In the following figures the dark blue color represent the part of the drum that is below the part that is colored yellow:

- The ten smallest eigenvalues for the square system can be seen in Fig. 2. We see that most eigenvalues have two different eigenvectors.
- The five smallest eigenvalues for the system containing a rectangle with side length L and 2L are shown in Fig. 4.
- For the system concerning a circle with radius L the eigenvalues and eigenvectors are shown in Fig. 4. The figure clearly shows the boundaries of the circle.

Since matrix M is sparse for all our systems, we can use Scipy's sparse eigenvalue and eigenvector solver besides the regular solver. The difference in speed between these two methods for solving the eigenvalue problem of our various systems is shown in Fig. 5. This figure shows us that the sparse solver is faster for all our systems.

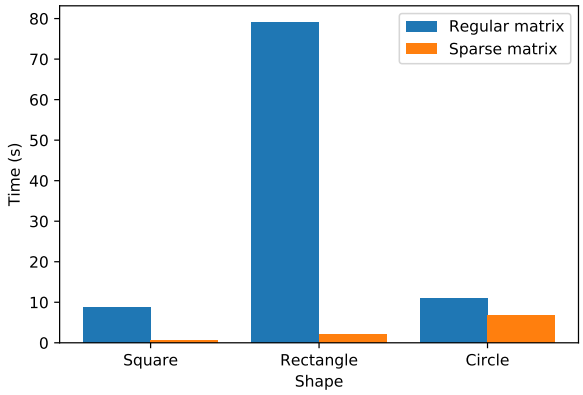

Fig. 5: Computation times for $N = 50$, $L = 1$ and varying grid shapes.

For various L we have computed the eigenfrequencies for our earlier mentioned systems. The results are shown in Fig. 6, Fig. 8 and Fig. 7. These figures are showing us that for higher values of L the eigenfrequencies are getting more clustered and their overall value is decreasing. However, we see that the eigenfrequencies for various L are almost equal for the different systems.

The influence of N on the eigenfrequencies of the square system is plotted in Fig. 9. The figure shows us that for higher N we obtain more and higher eigenfrequencies.

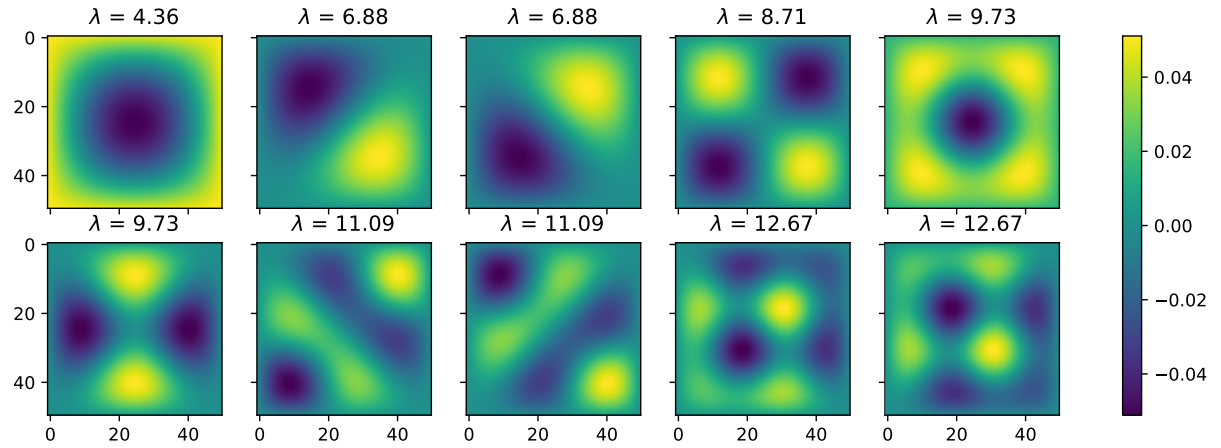

Fig. 2: The ten smallest eigenfrequencies and corresponding eigenvectors for square (50x50).

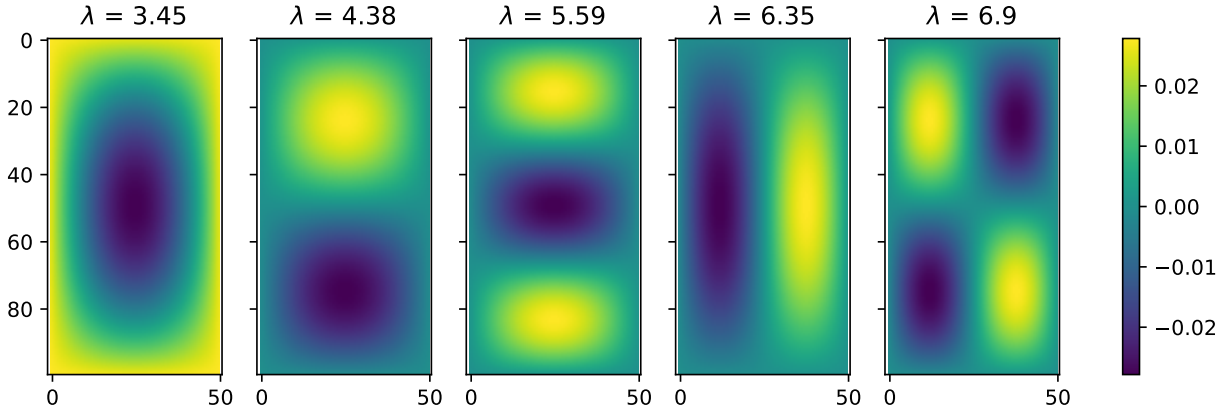

Fig. 3: The five smallest eigenfrequencies and corresponding eigenvectors for rectangle (50x100).

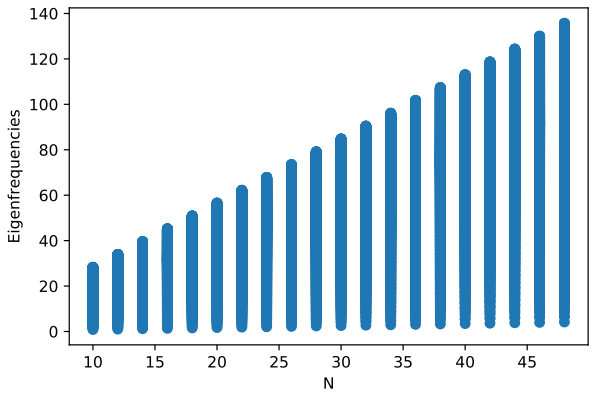

Fig. 9: Eigenfrequencies for varying N on a square grid with $L = 1$.

After our experiments with the time-independent part we add back in the time component to our algorithm to see how the shapes are changing over time. The result can be found in the attachments for the three smallest eigenvalues of the square system.

B. Direct method for Steady State Diffusion

Fig. 10 shows the steady state concentration $c(x, y)$ for a domain of a circular disk that has a radius of 2. A source, where the concentration is 1, is located at point $(0.6, 1.2)$. The concentration can be seen to spread in a circular manner and more so towards the center of the domain than towards the boundaries.

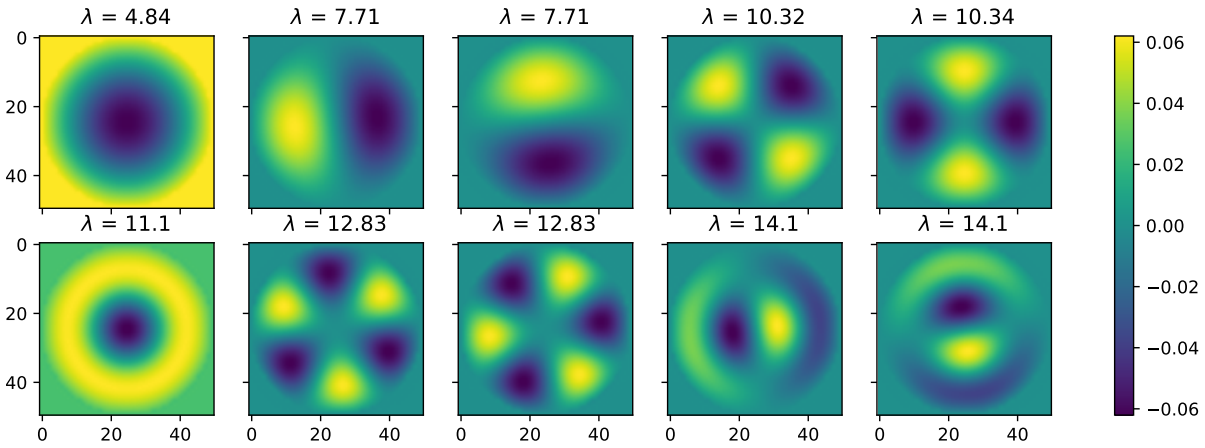

Fig. 4: The ten smallest eigenfrequencies and corresponding eigenvectors for circle (50x50, radius=50).

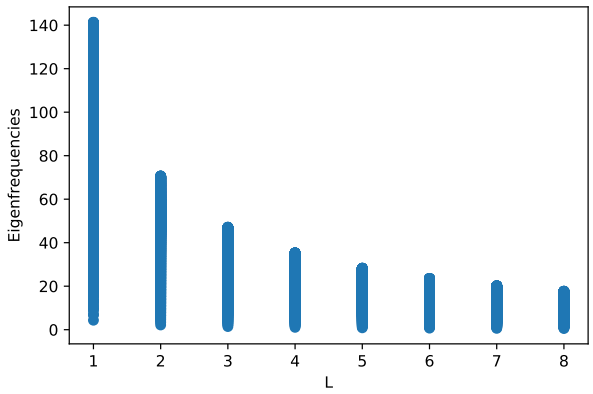

Fig. 6: Eigenfrequencies for varying L on a square grid with $N = 50$.

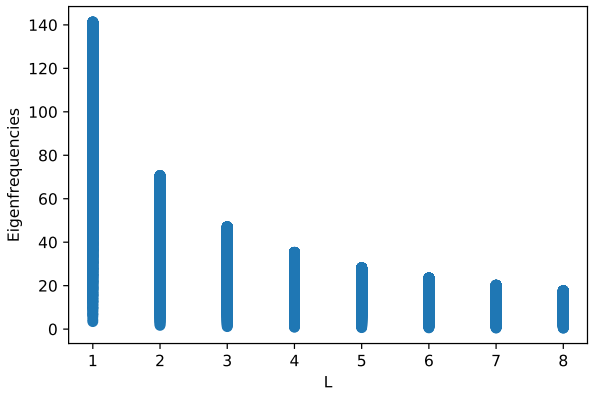

Fig. 8: Eigenfrequencies for varying L on a rectangular grid with $N = 50$.

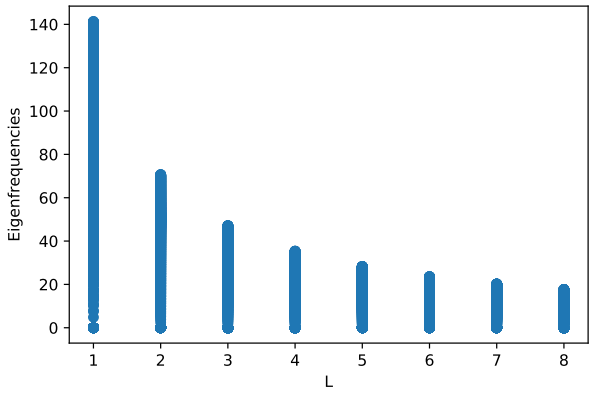

Fig. 7: Eigenfrequencies for varying L on a circular grid with $N = 50$.

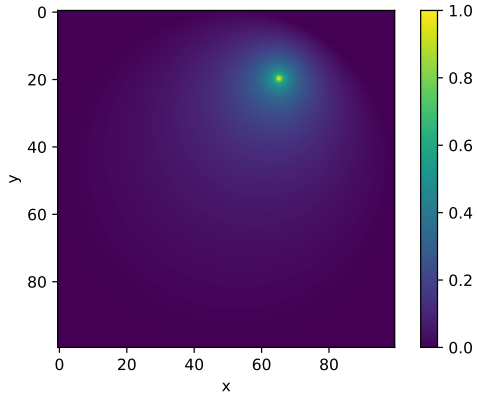

Fig. 10: Steady state concentration $c(x, y)$ for source at $(0.6, 1.2)$

V. Discussion

The results of our experiments concerning the eigenmodes of drums are showing us various things. First we can see in Fig. 2, Fig. 3 and Fig. 4 that the various shapes have different smallest eigenvalues and corresponding eigenvectors.

Second, Fig. 6, Fig. 8 and Fig. 7 are showing us the eigenfrequencies are dependent of size L (F), which means one is be able to identify the size of a drum by its eigenfrequencies. However, these plots are also showing us that the eigenfrequencies are almost equal for the various shapes of systems used. Hence, we conclude that it is not possible to identify the shape of a drum by its eigenfrequencies, and therefore one cannot hear the shape the shape of the drum.

Furthermore, we have seen using the sparse solver is indeed an improvement over using the regular solver (Fig. 5). Using the sparse solver, solving the square system takes the least amount of time, which is explained by M being the most sparse for this system. Solving the eigenvalue problem for the rectangle system takes more time which is due to M having a bigger size. Solving the circle system using the sparse solver wins time as well, however it is significantly lower since M is less sparse than in the other two systems. Also, we noticed using a different solver than the regular one (sp.linalg.eig) results in slightly modified eigenvectors which we cannot explain yet. Further work would consist of investigating this behaviour.

Also the influence of N on the eigenfrequencies of the square system are being investigated (Fig. 9). We see that the number of eigenfrequencies are growing for bigger N, while their range is also increasing. This is a logical consequence since M has shape (N^2, N^2) and thus results in more eigenfrequencies.

For the steady state system solved with a direct method we end up with the plot in Fig. 10. In this plot the diffusion is spreading towards the centre, since the source is close to the boundaries of a circular disk.

References

[1] M Heath. *Computing: An introductory survey*. McGraw-Hill, 1998.