Scientific Computing Exercise Set 3

Sjoerd Terpstra (11251980) & Youri Moll (10714235)

***Abstract*—This report presents numerical solvers for the two-dimensional time dependent wave and time independent diffusion equations as well as the leapfrog method applied to a harmonic oscillator. The wave equation is applied to a drum for a square, rectangular and circular domain. A numerical solution existing of two independent parts is proposed: a function of time and a function of space. The resulting spatial dependent equation is solved using a direct solver to find the eigenmodes of the system, and the time dependence is added by an oscillating function. The time independent diffusion equation, or Laplace equation, is solved for a circular domain with a single source point. This equation is rewritten as a linear system and solved using direct methods. The harmonic oscillator, driven by Hooke's law, is solved by the leapfrog method, a finite difference scheme. A sinusoidal driving force is also added to this system.**

I. Introduction

THIS reports contains an application of a two-dimensional wave equation to a drum. Steady state solutions are found for the diffusion equation and the leapfrog method is applied to a harmonic oscillator.

The wave equation is applied to three shapes of a drum: square, rectangular and circular. The investigated solution is formed by a function of time and a function of space, both independent of each other. This allows the determining of the *eigenmodes* or resonating frequencies of the drum separate from the time dependent solution. The time dependence can then be applied to these eigenmodes. This is done through first writing the wave equation in matrix-vector form and then finding the eigenvalues and eigenvectors that are characteristic for each eigenmode for each shape of the drum.

Steady state solutions for the Laplace diffusion equation are found through direct methods. This is done in a similar fashion as with the wave equation. The Laplace equation is written in matrix form, leading to a linear system of equations. Finding the steady state is then done through solving the set of resulting linear equations.

The leapfrog method is a central difference approximation using forward difference schemes for each separate differential equation in a set. In general, central difference schemes suffer from performance issues because the next timestep is unknown, making it necessary to solve a set of linear equations for the implicit part. This problem is avoided in the leapfrog method as it addresses sets of differential equations that have a specific form, namely where the differential equations depend on each other instead of themselves. In the leapfrog method the differential equations are updated in sequence of each other i.e. for two partial differential equations, each at a half time step apart from the other, each using the latest obtained solution from the other. In this report the example of a harmonic oscillator is used where the change in position and the change in velocity depend on each other instead of themselves.

II. Theory

A. Eigenmodes of drums or membranes of different shapes

Waves in two dimensions are described by the two-dimensional wave equation

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

with $u(x,y,t)$ the amplitude of the wave at coordinates (x,y) and time t, and c a positive coefficient.

The examined solutions are made up of two independent functions; one depending only on time, one only dependent on position

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

To make Equation 1 more easily solvable, Equation 2 can be used to separate Equation 1 in two separate equations coupled by a constant K. Substitute Equation 2 in Equation 1 and move all components depending on time to the left, and all spatial components to the right,

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

Since both sides depends on a different set of independent variables, both sides are constant. This leads to the following two equations

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

$$\nabla^2 v(x,y) = K v(x,y), \tag{5}$$

with K a constant.

There are three different scenarios for Equation 4 and Equation 5: $K > 0$, $K = 0$ or $K < 0$. $K > 0$ gives

an exponential solution, which is in-congruent with our boundary conditions of zero at the edges of our domains. The solution is constant for $K = 0$. For $K < 0$ the solution is an oscillation

$$T(t) = A\cos(c\lambda t) + B\sin(c\lambda t), \tag{6}$$

with $\lambda^2 = -K$. Taking $A = 1$, $B = 1$, the boundary conditions are $-\pi/2 \leq T(t) \leq \pi/2$. At $t = 0$, the solution is $T(0) = \cos 0 + \sin 0 = 1$. Thus $u(x, y, t = 0) = v(x, y)$.

Equation 5 can be seen as an eigenvalue problem with K the eigenvalue, $v(x, y)$ mapped to $\boldsymbol{v}$ as the eigenvector and ∇^2 as a matrix $\mathbf{M}$.

$v(x, y)$ is formed by a grid of size N_x by N_y with discretized points (i, j), with a value at each grid point denoting the membrane amplitude at that point. Each grid point can be denoted by a number row by row, as seen in Figure 1 for a 3 by 3 grid. This grid can be translated to column vector $\boldsymbol{v}$ which each entry denoting the corresponding grid point

$$\boldsymbol{v} = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \end{pmatrix}^T. \tag{7}$$

The matrix $\mathbf{M}$ has size $N_x^2 \text{x} N_y^2$, and each row in the

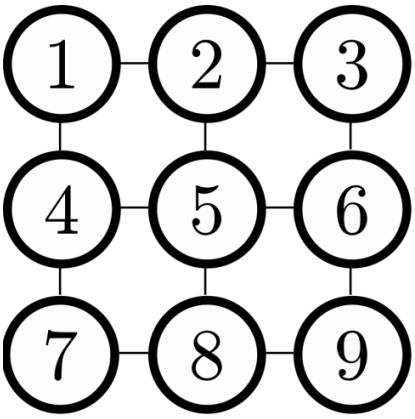

Fig. 1: A 3 by 3 grid for a square domain. Each grid points is numbered from left to right and top to bottom.

matrix corresponds to the update stencil for each grid point (the factor multiplication of $(\frac{L}{N})^{-2}$ is omitted at this point and will be reintroduced later)

$$v_{i,j} = v_{i-1,j} + v_{i+1,j} + v_{i,j-1} + v_{i,j+1} - 4v_{i.j}. \tag{8}$$

This update stencil originates from a finite difference scheme of the ∇^2 operator. Applying this stencil for each grid point results in Equation 9. Whenever an index of $v_{i,j}$ falls outside of the boundary (e.g. $v_{-1,0}$), the value of $v_{i,j}$ is set to be zero at that position.

$$\mathbf{M} = \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \tag{9}$$

This matrix can be easily adjusted for any rectangular domain, by changing the dimension $N_x^2 \text{x} N_y^2$ appropriately. For a circular domain, the matrix gets slightly more complex. Assume a circle with radius $L/2$, and every point inside this circle belongs to the domain. Then place the circle in a square grid with length L, and discretization N_x, N_y, with $N \equiv N_x = N_y$. First construct the matrix $\mathbf{M}$ with dimensions $N^2 \text{x} N^2$ similar to Equation 9. Then check for each row if that row corresponds to a point inside the circle. If not, then set the whole row to zero, except for the value at the position corresponding to the grid point; set this to 1 instead of the standard -4.

As a last step in constructing $\mathbf{M}$ for any domain size, is multiplying it by $(\frac{L}{N})^{-2}$.

The eigenvalues K and eigenvectors $\boldsymbol{v}$ are then obtained by solving

$$\mathbf{M}\boldsymbol{v} = K\boldsymbol{v}. \tag{10}$$

Each element in the resulting $\boldsymbol{v}$ corresponds again to a grid point of the domain. By reshaping this vector to a two-dimensional matrix, a amplitude image is obtained.

B. Direct methods for solving steady state problems

The time independent two dimensional diffusion equation is the Laplace equation

$$\nabla^2 c(x, y) = 0, \tag{11}$$

with $c(c, y)$ the concentration at x, y.

This equation can be directly solved using a similar approach to solving the two dimensional wave equation. For ∇^2 a matrix $\mathbf{M}$ can be constructed. This matrix is similar to the matrix for the circular domain in subsection II-A, since the matrix there also described the ∇^2 operator and the domain can be similarly discretized.

There is one difference with the matrix from subsection II-A; this problem has a source point placed in the domain. The source point should have a constant concentration in the system i.e. not change in the solution

compared to the initial conditions. Thus the source point is encoded in $\mathbf{M}$ with a diagonal value of 1 at the position corresponding to the grid point instead of -4, and the rest of the row zero's (so that the concentration does not change according to its neighbours). The same boundary conditions are taken into account as well, with the sinks with concentration 0 at the boundaries.

The initial condition $\boldsymbol{b}$ is generated in the same fashion as $\boldsymbol{v}$ from subsection II-A. Each element corresponding to the grid position is set to the initial concentration in that position i.e. 0 for an initial concentration zero and 1 for an initial concentration one. For example for a 3 by 3 grid (Figure 1) with one source point at grid position 2

$$\boldsymbol{b} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T. \tag{12}$$

The system can then be solved for the solution $\boldsymbol{c}$ and initial condition $\boldsymbol{b}$ as follows,

$$\mathbf{M}\boldsymbol{c} = \boldsymbol{b}. \tag{13}$$

C. The leapfrog method - efficient time integration

Newton's second law states that the force f on a object is proportionate to the mass m and acceleration a of the object: $F = ma$. This law can be split in two first order differential equations and in one dimension becomes

$$\frac{\partial x(t)}{\partial t} = v(x,t) \tag{14}$$

$$\frac{\partial v(x,t)}{\partial t} = \frac{F(x(t))}{m}, \tag{15}$$

with $v(x,t)$ the velocity.

A technique called the leapfrog method is deployed to solve these equations numerically. With this technique, both differential equations are solved not in the same time-step, but in succession of each other. This is possible, because both equations do not dependent on their own derivative. Both equations can then be discretized as follows

$$\frac{x_{n+1} - x_n}{\Delta t} = v_{n+1/2} \tag{16}$$

$$\frac{v_{n+3/2} - v_{n+1/2}}{\Delta t} = \frac{F(x_{n+1})}{m}, \tag{17}$$

whereby the velocity is updated in between the updates of the position.

The system to study is a simple one-dimensional harmonic oscillator, described by Hooke's law

$$F(x) = -kx. \tag{18}$$

Substitute Equation 18 in Equation 17 and simplify to obtain the iterative scheme

$$x_{n+1} = v_{n+1/2}\Delta t + x_n \tag{19}$$

$$v_{n+3/2} = \frac{-kx_{n+1}}{m}\Delta t + v_{n+1/2} \tag{20}$$

It is possible to add an external time-dependent sinusoidal force

$$F(t) = \sin(\omega t), \tag{21}$$

by adding this force to the total force in Equation 15 together with Equation 18. The iterative scheme for the velocity then becomes

$$v_{n+3/2} = \frac{\sin(\omega t_{n+1}) - x_{n+1}}{m}\Delta t + v_{n+1/2}. \tag{22}$$

The iterative scheme for the position stays the same as Equation 19.

III. Method

All models are implemented in Python 3, using the Numpy and Scipy libraries.

For the drum we define $L = 1$ divided in N intervals. This means that the square drum is divided in N by N, the circular drum in a circle with radius $1/2$ inside a square grid, again divided in N by N and the rectangular drum is formed by N by $2N$ intervals. If not mentioned otherwise in the results, the following default values are assumed: square domain $N = 100$, rectangular domain $N = 80$, circular domain $N = 100$.

After the grid is initiated, first matrix M from Equation 9 is constructed for the wave equation. The eigenvalues and eigenvectors are then obtained by solving Equation 10. Solving this is done using `scipy.linalg.eig` and `scipy.sparse.linalg.eigs`. The former is a solver for dense matrices and the latter for sparse matrices. A speed comparison is done between those two methods. Whenever not specifically mentioned, the system is solved using the sparse solver. This is because M is a sparse matrix, so a higher speed of solving is expected when using a sparse solver. `scipy.linalg.eig` finds all the eigenvalues of the system, while `scipy.sparse.linalg.eigs` only finds a specified number of eigenvalues. For this, two arguments are supplemented to this function: `which="SM"`, this means that the function will find the smallest eigenvalues, and `k` which denotes the amount of eigenvalues that need to be found. If not specified in the Results section, `k` is set to 6.

The boundary conditions are implemented as fixed with wave amplitude 0 at the outside of the edges of the grid or outside the circular area of the grid for the circular drum.

For the diffusion equation, the matrix described in subsection II-B is first constructed alongside the initial condition vector. The constant c is taken as 0 for all simulations. Then the set of linear equations is solved with one of the following

two library functions: `scipy.linalg.solve` and `scipy.sparse.linalg.spsolve`. Both of these methods both solve the linear system. However the method from the sparse package is, again, more geared towards spare matrices. A comparison between the performance of these two methods will be presented in the following section.

The implementation of the leapfrog scheme iterates N times with $N = \frac{total\ time}{dt}$. The mass m is always taken as 1. For each iteration, the next position value is first calculated, then the next velocity value. This process is repeated until the total time is elapsed and the process terminates and results are returned.

IV. Results

The first, third, fifth and seventh eigenmodes for the square, rectangular and circular drum can be found in Figures Figure 2, Figure 3 and Figure 4 respectively. Note that, as the eigenfrequency corresponding to the eigenmode increases, the amount of minima and maxima also increases. Three mp4 files are uploaded alongside this report showing the time evolution of the first, third and seventh smallest eigenfrequency for a square drum.

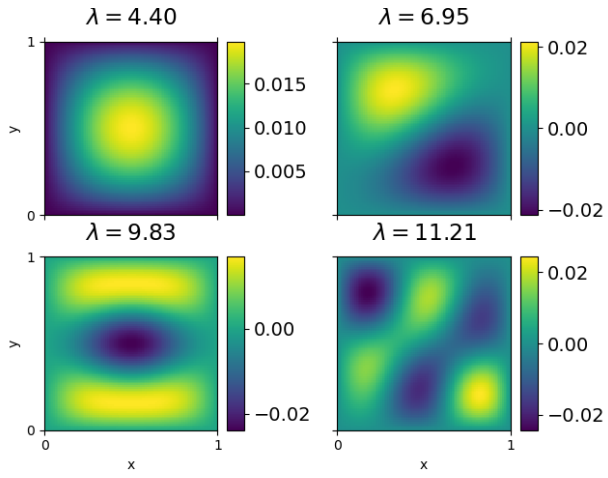

Fig. 2: The first, third, fifth and seventh smallest eigenfrequencies for a square drum of length 1.

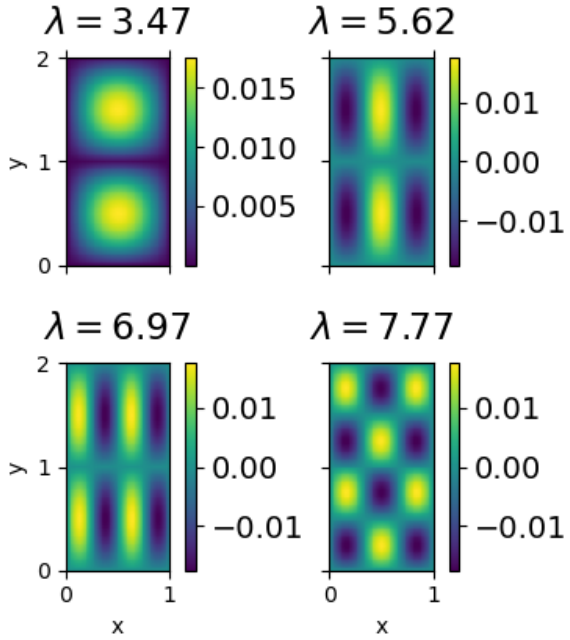

Fig. 3: The first, third, fifth and seventh smallest eigenfrequencies for a rectangular drum of width 1 by length 2.

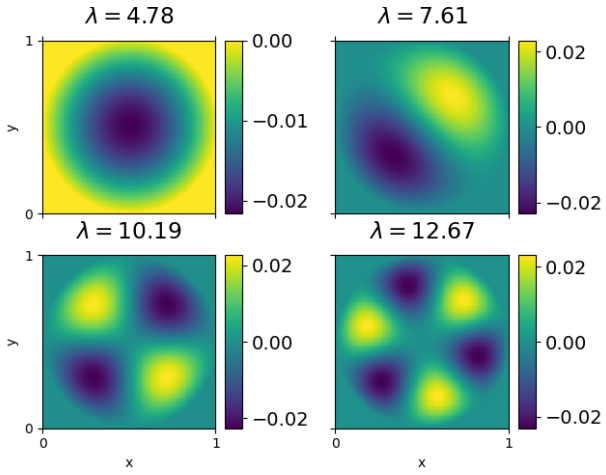

Fig. 4: The first, third, fifth and seventh smallest eigenfrequencies for a circular drum with radius $1/2$.

The spectrum of the eigenfrequencies of the square, rectangular and circular drum depending on the size of the domain L are shown in Figures Figure 5, Figure 6 and Figure 7 respectively. For all three domains, the 40 smallest eigenfrequencies decrease with the domain size L. The eigenfrequencies seem to be significantly lower for the rectangular domain compared with both the square and circular domain. Note that the shape of the drum does not seem to have as much effect on this as the size of the drum.

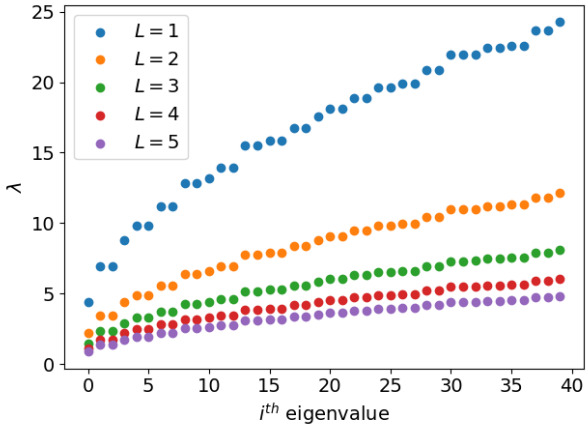

Fig. 5: The 40 smallest eigenfrequencies for the spatial solution of a square drum with $N = 100$ for varying lengths L.

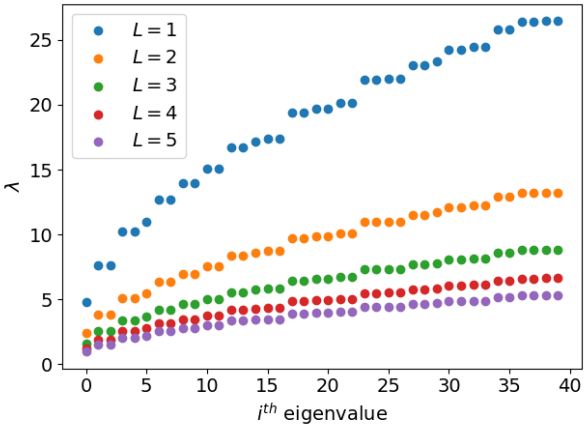

Fig. 7: The 40 smallest eigenfrequencies for the spatial solution of a circular drum with $N = 100$ for varying diameters L.

In order to determine the influence of discretization factor N on the spectrum of eigenfrequencies, the eigenspectrum for $N = 50$ is shown in Figure 8. As can be seen in the figure, the spectrum is highly similar to that of $N = 100$, meaning that it is viable that the spectrum is independent of N.

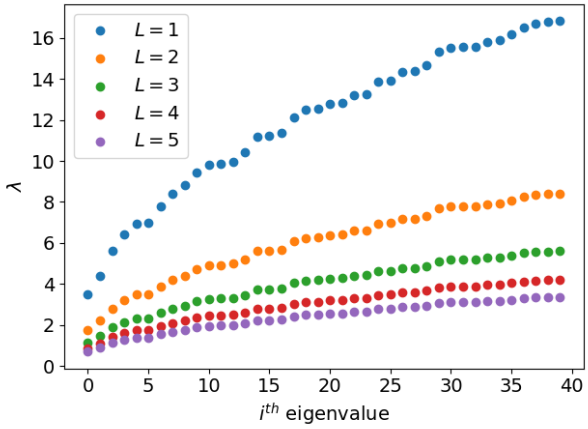

Fig. 6: The 40 smallest eigenfrequencies for the spatial solution of a rectangle drum with $N = 80$ for varying lengths L and widths $2L$.

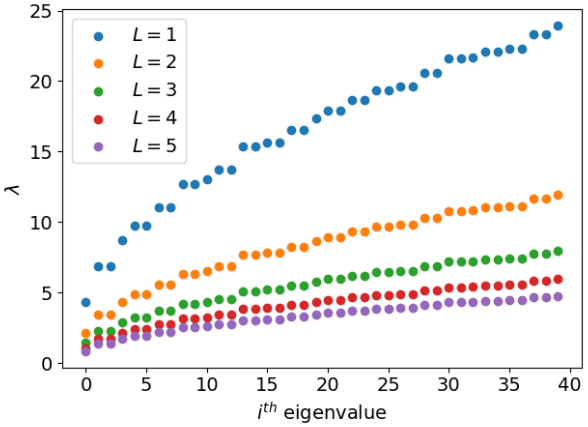

Fig. 8: Square for comparing eig values for lower discretization, $N = 50$

The time to solve the system is greatly reduced by using a sparse matrix solver, as is clear from Figure 9. The execution time for both the `scipy.linalg.eigs` and `scipy.sparse.linalg.eigs` methods seem to increase exponentially with N.

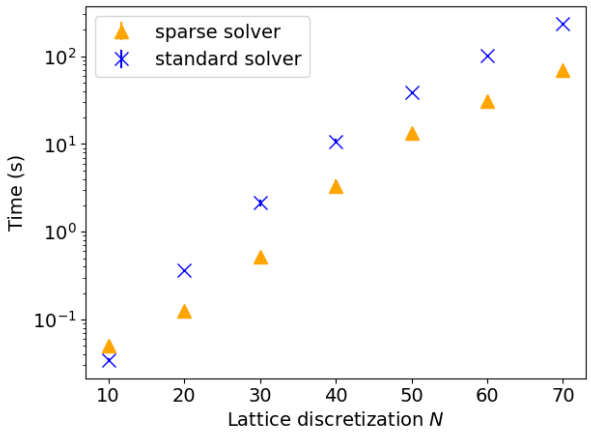

Fig. 9: Speed comparison for solving Equation 10 for a square drum of length 1 and discretization $N = 100$ using a standard solver (`scipy.linalg.eig`) and a sparse solver (`scipy.sparse.linalg.eigs`). This is done for ten repetitions, and the errorbars show the standard deviation (however the standard deviation is so small for most points, that it is not visible.

The solution of the diffusion of one source point at $(0.6, 1.2)$ in a circular domain is plotted in Figure 10. Intersection plot in Figure 11 shows the concentration of the diffusing substance in the x and y direction from the source point. As can be seen, the concentration seems to decrease in a quadratic fashion.

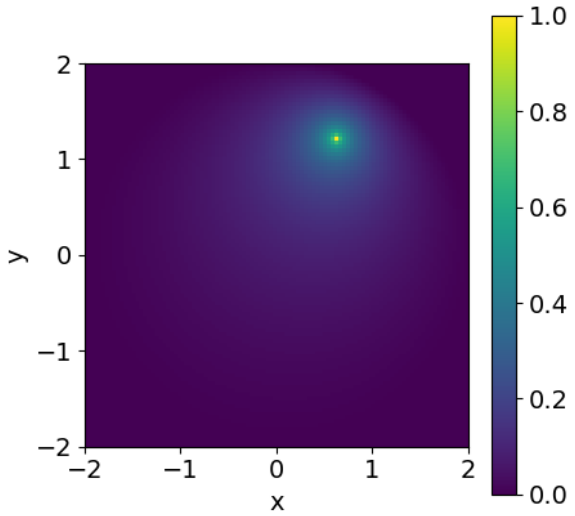

Fig. 10: Steady state of the diffusion in a circular domain with a radius of 2 with discretization factor $N = 100$. A single source with a value of 1 is present at $(0.6, 1.2)$.

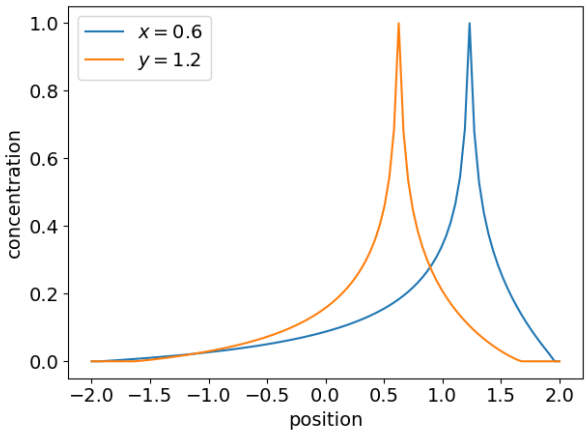

Fig. 11: Solution to the Laplace equation for a circular domain with radius 2 with a single source point of 1 at $(0.6, 1.2)$ with discretization $N = 100$. Shows is an intersection in the y-direction that goes through $x = 0.6$ and an intersection through $y = 1.2$).

Then, in Figure 12 a comparison is made between the speed of the sparse solver and the standard solver for solving a system of linear equations. As expected, the sparse solver performs better due to the fact that this is a sparse matrix problem, for which the sparse solver is highly optimised.

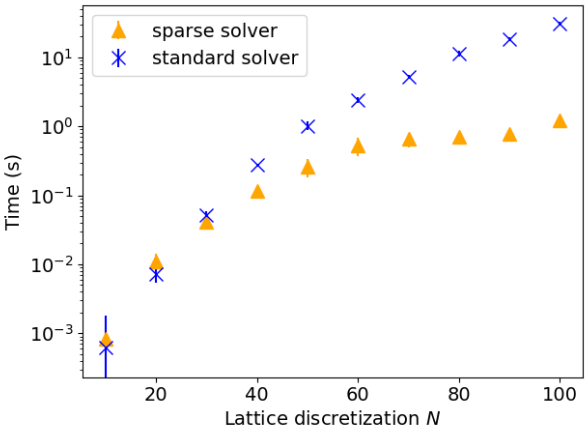

Fig. 12: Speed comparison for solving Equation 13 for the Laplace equation with a circular domain with diameter 4 and a single source of 1 present at $(0.6, 1.2)$ and discretization $N = 100$ using a standard solver (`scipy.linalg.solve`) and a sparse solver (`scipy.sparse.linalg.spsolve`). This is done for ten repetitions, and the errorbars show the standard deviation (however the standard deviation is so small for most points, that it is not visible.

The harmonic oscillators generated through the leapfrog method for varying values for the spring con-

stant k can be found in figures 13 ($k = 0.2$), 14 ($k = 0.5$), 15 ($k = 1$) and 16 ($k = 2$) with $\Delta t = 0.01$.

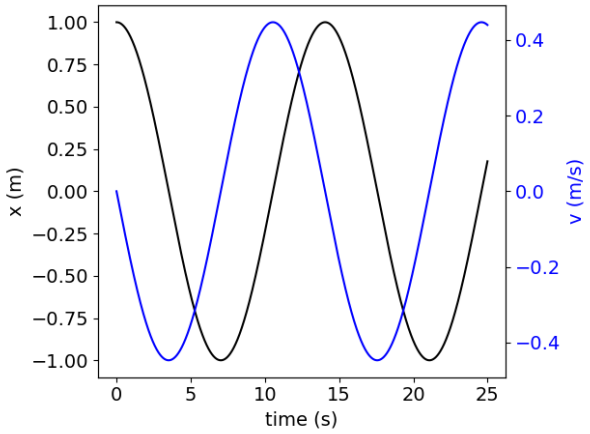

Fig. 13: A harmonic oscillator governed by Hooke's law solved by the leapfrog method with spring constant $k = 0.2$ and timestep $\Delta t = 0.01$.

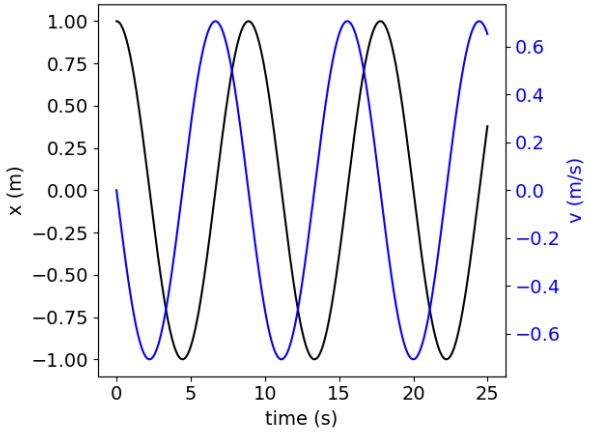

Fig. 14: A harmonic oscillator governed by Hooke's law solved by the leapfrog method with spring constant $k = 0.5$ and timestep $\Delta t = 0.01$.

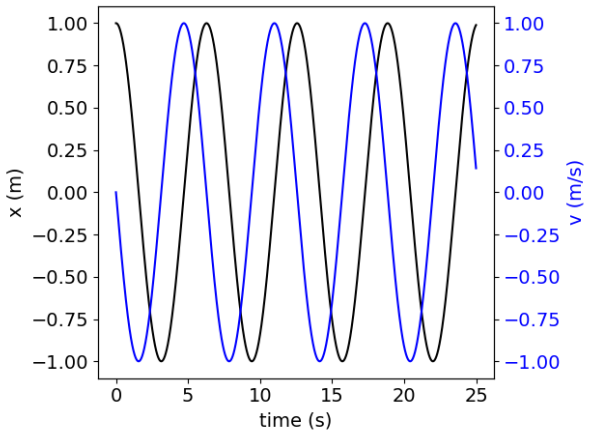

Fig. 15: A harmonic oscillator governed by Hooke's law solved by the leapfrog method with spring constant $k = 1$ and timestep $\Delta t = 0.01$.

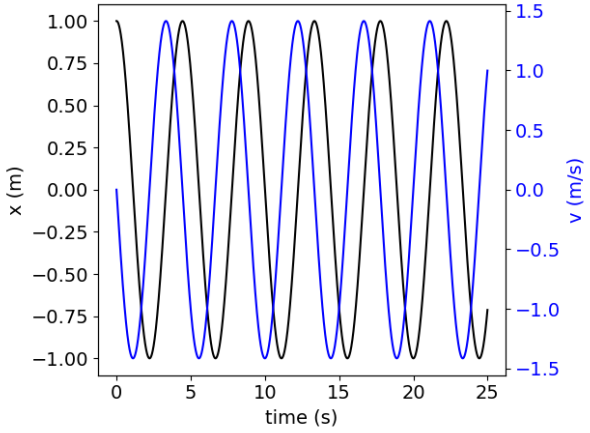

Fig. 16: A harmonic oscillator governed by Hooke's law solved by the leapfrog method with spring constant $k = 2$ and timestep $\Delta t = 0.01$.

In Figure 17 the harmonic oscillator ($k = 2$, $\Delta t = 0.01$) with an added sinusoidal force with frequency $\omega = 2.5$ can be seen.

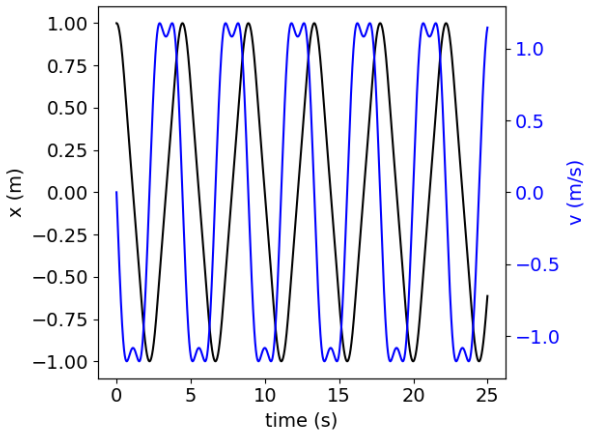

Fig. 17: A harmonic oscillator governed by Hooke's law with an added external sinusoidal force with frequency $\omega = 2.5$ solved by the leapfrog method with spring constant $k = 2$ and timestep $\Delta t = 0.01$.

Lastly, Figure 18 shows a phase plot of the harmonic oscillator ($k = 2$, $\Delta t = 0.01$) with an added sinusoidal force with varying frequencies ω.

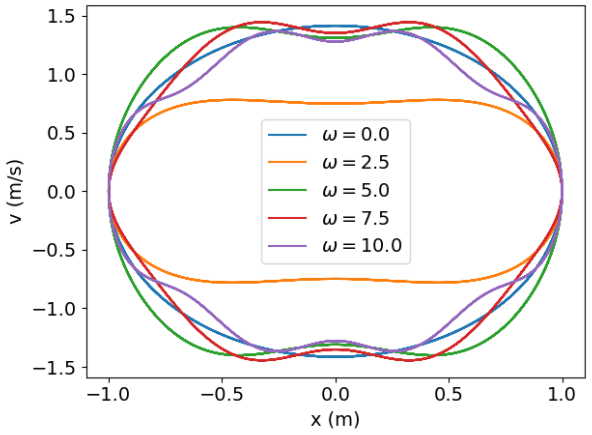

Fig. 18: A phase plot of a harmonic oscillator governed by Hooke's law with an added external sinusoidal force with varying frequencies ω solved by the leapfrog method with spring constant $k = 2$ and timestep $\Delta t = 0.01$.

V. Discussion

The first observation is that the eigenmodes, or stationary waves, get more complex (number of minima and maxima) as the corresponding eigenvalue increases. Note that in the results section the first, third, fifth and seventh eigenmodes are shown. This is because some consecutive eigenmodes appear to be reflections of one another in the height of the amplitude or the z-axis.

The time dependent solution of the vibrating of the drum gives a stationary waves with oscillating amplitudes in time, as seen in the supplied animations.

Furthermore, it has been shown that the spectrum of eigenfrequencies does depend on L for all shapes, as opposed to the number of discretization steps. However, using less discretization steps results in a loss of accuracy. The value of the smallest eigenvalues seem to get lower when increasing the domain size N. This might be explained by the fact that the matrix $\mathbf{M}$ is multiplied by a factor $(L/N)^{-2}$. This means that if the number of discretization steps N is kept constant, that increasing L, lead to a smaller multiplication factor. Since this is a constant factor, the eigenvalues K resulting from solving Equation 10 should also decrease by increasing L.

For the steady state of the diffusion equation, the concentration seem to decays in a quadratic fashion from the source point. This is as expected because in two dimensions, the diffusing substance can propagate as a circle proportionate to r^2. Following a similar argument, the decay in concentration is expected to be cubic in three dimensions.

The sparse matrix solver has been shown to perform better for both the wave- and diffusion equations. Especially for solving the Laplace equation, the sparse solver outperformed the standard solver by far for the larger lattice discretization sizes N. However, the documentation of these sparse solvers mentions that they become very computationally expensive for dense matrices. Because of this, the sparse solvers are not suitable for all problems and it is therefore up to the programmer to determine which solver to use.

The harmonic oscillator solved with the leapfrog method shows a connection between the spring constant k and the period of the oscillator. It has been observed that by increasing k, the period gets shorter. This is explained by the fact that according to Hooke's law, the force increases linearly with a higher k. A higher force results in a higher velocity and thus decreasing the period of the harmonic oscillator.

The leapfrog method is quick, and intuitively it makes sense that it performs better than a naive approach where the implicit part is solved though solving a set of linear equations. However, it would have been interesting to measure the difference in performance between the leapfrog method and the naive approach in order to quantify the speedup.

Lastly, for the sinusoidal force added to the harmonic oscillator that is solved using the leapfrog scheme, it has been found that the resulting shape becomes more and more stretched in the x-direction as the ω value gets closer to the original frequency. This is due to

the fact that the sinusoidal force starts resonating with the original frequency causing periodic amplifications and compression of the oscillating behavior. For higher ω values the phase graphs show oscillating behaviour around an elliptic shape, with an increasing amount of periods, with increasing ω.

VI. Conclusion

In this report, numerical schemes for solving the two dimensional wave equation, diffusion equation and a harmonic oscillator have been presented. For the wave equation, harmonic frequencies that produce a stationary wave have been found. This was combined with an independent time dependent function to produce oscillating solutions.

The diffusion equation has been shown to be solvable through a direct method which can be used to find the steady state of diffusion models. For both the wave- and diffusion equations, a performance comparison was done between a sparse and dense matrix solver. Since these are sparse matrix problems, the sparse solver performs better, especially for larger discretization factors.

Lastly, a harmonic oscillator governed by hook's law has been implemented using the leapfrog method. The influence of the spring constant on the frequency has been shown. Then, a sinusoidal force has been added to the oscillator and the effect of varying the frequency of this sinusoidal force on the resulting oscillator has been examined.