Scientific Computing Exercise Set 2

Victoria Peterson - 15476758 & Paul Jungnickel - 15716554 & Karolina Chłopicka - 15716546

***Repository**: https://github.com/PaulJungnickel/ScientificComputing_Set3*

I. Introduction

COMPUTATIONAL effort required to approximate the solution to a system of equations may vary significantly depending on the type of discretization scheme used. Many numerical approaches leverage optimized general-purpose linear algebra operations, while others preserve system-specific symmetries to enhance accuracy and stability. Likewise, sparse matrix representation can play a part in improving computational performance in large-scale systems.

This report approximates the eigenmodes of vibrations in two-dimensional manifolds and diffusive heat spreading using matrix methods. Additionally, comparisons are made between the characteristics of different numerical integration methods on the harmonic oscillator, comparing accuracy and stability. Leveraging both direct and iterative numerical methods in both sparse and dense matrices methods allows for an evaluation of their influence on computational complexity and solution fidelity.

The report begins with Theory (II) for the foundational 2D wave equation, steady-state diffusion, and harmonic oscillator; followed by a Methods (III section which describes the numerical and computational methods/algorithms employed; applied in the Results (IV section, discussing and highlighting key observations from implementations; and ending with the Conclusion (V) to summarize findings.

II. Theory

A. 2D Wave Equation

Vibrations on 2D membranes may be solved with the wave equation in two dimensions to find eigenmodes representing vibration pattern characteristics and eigenfrequencies corresponding to the natural frequency at which the vibrations occur.

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

$u(x, y, t)$ is the displacement of the membrane at position (x, y) at time t, c is the wave propagation speed, and ∇^2 is the Laplacian operator, describing spatial variation of u. In this case, the edges of the membrane are boundaries such that $u(x, y) = 0$. Assuming a separable solution in the form...

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

Substitute into the wave equation...

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

$$v(x, y)\frac{\partial^2 T(t)}{\partial t^2} = c^2 T(t) \nabla^2 v(x, y) \tag{4}$$

Then split time and space into different sides...

$$\frac{1}{T(t)}\frac{\partial^2 T(t)}{\partial t^2} = c^2 \frac{1}{v(x, y)} \nabla^2 v(x, y) \tag{5}$$

Separate the left and right sides...

$$\frac{\partial^2 T(t)}{\partial t^2} = K c^2 T(t) \tag{6}$$

$$\nabla^2 v(x, y) = K v(x, y) \tag{7}$$

Given that ∇^2 is a linear operator, Eq 7 is an eigenvalue problem. Here, the eigenvalues K determine the resonance frequencies $\omega = \sqrt{-K}$ while the eigenmodes describe the shape of vibration at these eigenfrequencies.

Writing the system of equations into a matrix eigenvalue problem of the form $\mathbf{Mv} = K\mathbf{v}$ allows for a solution to be found using numerical solvers.

B. Steady-State Diffusion using Direct Methods

As an expansion on previous diffusion equation iterative solvers such as Jacobi or Gauss-Seidel, a steady state solution of the equation may be found using a direct method by formulating the problem as a linear system and solving it in one step using matrix operations. The steady-state solution of the Laplace equation describes the equilibrium of diffusion given by...

$$\nabla^2 c = 0 \tag{8}$$

C. Harmonic Oscillator

The driven harmonic oscillator describes a single particle inside an harmonic potential that is driven by a time-dependent force $F(t)$. Its phase space is spanned by its displacement x and velocity v. Their time evolution is described by the following set of differential equations:

$$\partial_t x = v$$
$$\partial_t v = \frac{F(t) - kx}{m}$$

where k is a positive constant and m the particle mass.

III. Method

A. Discretization of Laplace operator

We obtain a discretized version of the problem by considering evenly spaced points $(x_i, y_j) = (i\Delta x, j\Delta x) \in S$ in space. Here, S can be any shape that is a subset of some larger $N \times N$ grid. To second order in Δx, the Laplacian of a scalar field on the surface is:

$$\nabla^2 v_{i,j} = \frac{v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{i,j}}{\Delta x^2} \quad (9)$$

This results in a system of N^2 linear equations which can be written as a $N^2 \times N^2$ matrix and solved for the N^2 grid points. We always specify $v_{i,j} = 0, \forall (x_i, y_j) \notin S$, so we only need to solve $|S|$ equations and can remove all outside elements from the formula (9). This results in an $|S| \times |S|$ system of equations. The shapes considered in this report are squares:

$$(x_i, y_j) \in S \Leftrightarrow max(|x_i|, |y_j|) \leq L$$

$L \times 2L$ rectangles:

$$(x_i, y_j) \in S \Leftrightarrow max(|2x_i|, |y_j|) \leq L$$

and circles of radius L:

$$(x_i, y_j) \in S \Leftrightarrow \sqrt{x_i^2 + y_j^2} \leq L$$

Algorithm 1 shows the process of generating the Laplacian from a boolean mask that specifies which points of the grid are in the shape. Each $(x_i, y_j) \in S$ is assigned a unique index given by some order (e.g. upper left $\rightarrow$ lower right) and the terms of Eq 9 for neighbors in S written in the appropriate locations.

Figure 1 shows the resulting Laplacian matrices for a square and a circle in a very coarse 5×5 grid.

Algorithm 1 Construct Laplacian Matrix

```
procedure LAPLACIAN(N, S)
    L ← |S| × |S| matrix
    for (x_i, y_j) ∈ S do
        id_{i,j} ← order of x_i, y_j in S
        L[id_{i,j}, id_{i,j}] ← −4
        for r, s ∈ {±1} do
            if (x_{i+r}, y_{j+s}) ∈ S then
                L[id_{i,j}, id_{i+r,j+s}] ← 1
            end if
        end for
    end for
    L ← (L/N)^2 L
    return L
end procedure
```

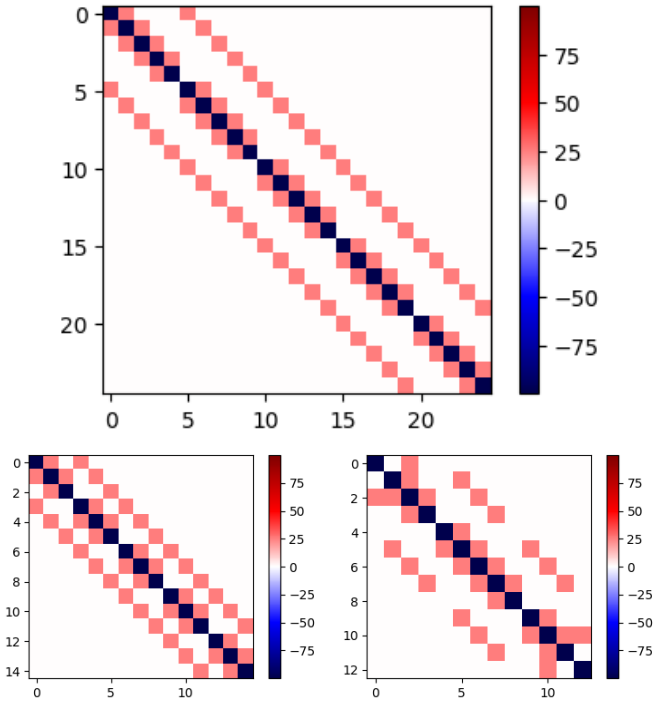


Fig. 1: Shape of the Laplacian $\mathcal{L}$ for 5x5 discretization with a square (top), rectangular (left) and spherical (right) shape.

B. Numerical solutions to the eigenvalue problem

Several methods are available to solve eigenvalue problems. We used the implementations provided by the scipy python library: `eig()` for finding all eigenvalues and the much more efficient `eigs()` for finding the eigenvalues with the smallest magnitude. `eigs()` also supports using a sparse matrix representation, which is very useful, since the Laplacian $\mathcal{L}$ has at most $4|S| << |S|^2$ nonzero entries.

C. Numerical solution of the Steady State Diffusion equation

The Laplacian is again produced as described in section III-A. The goal is still to solve the system of linear equations, but this time the right side of the equation is constant. In this case the `linsolve` method from the `scipy` library is appropriate.

This circular domain is defined with a boundary $c(x, y) = 0$, radius of 2, and initial source condition of $c(0.6, 1.2) = 1.0$ in Algorithm 2 where the linear system is defined with the source condition and solved using `scipy.linalg.solve()` or `scipy.sparse.linalg.spsolve()` to find c.

Algorithm 2 Laplacian Concentration Initialization

procedure STEADYSTATECONCENTRATION(x, y, N, S, $\mathcal{L}$)
  $\mathcal{L} \leftarrow |S| \times |S|$ matrix
  $n \leftarrow$ num of shape indices
  Initialize b as a zero vector of size n
  $(src_x, src_y) \leftarrow (0.6, 1.2)$
  $x_{diff} \leftarrow |x[S] - src_x|$
  $y_{diff} \leftarrow |y[S] - src_y|$
  $src\_idx \leftarrow \min(x_{diff} + y_{diff})$
  $b[src\_idx] \leftarrow 1$
  Set $\mathcal{L}[src\_idx] \leftarrow 0$
  Set $\mathcal{L}[src\_idx, src\_idx] \leftarrow 1$
  return $sparse\_laplacian$
end procedure

In this algorithm, the initial Laplacian Matrix created by Algorithm 1 and the shape mask are used to create Matrix M and vector b respectively. b corresponds to the cells of the grid which are located within the circular boundary such that all values are 0 except for at the source point, where it is 1. Likewise, the Laplacian matrix M retains only the values within the circle to enforce boundaries and the row containing the source point is set to 0 except at the diagonal, which is set to 1.

D. The Leapfrog method

The common way for solving an ordinary differential equation with multiple variables is to evaluate the time derivative of all variables at some time points and extrapolate their development. The leapfrog method is used to integrate ODEs containing the second time derivative of a variable. Since the change in velocity only depends on the displacement, and vice-versa, leap-frog alternates between updating the velocity and displacement, using the other updated value. More formally, time is discretized with step size Δt. Each time step, v is updated first at time $t+\frac{1}{2}\Delta t$. Then x is updated using the previous result for v (Eq 27, 28 from assignment, rearranged):

$$v_{i+\frac{1}{2}} = v_{i-\frac{1}{2}} + \Delta t \frac{F(i\Delta t) - kx_i}{m} \tag{10}$$

$$x_{i+1} = x_i + \Delta t v_{i+\frac{1}{2}} \tag{11}$$

This requires only the initial values $x_0, v_{-\frac{1}{2}}$.

IV. Results and Discussion

A. 2D Wave Equation

We solved the wave equation on the two-dimensional elastic material. We are interested in the eigenmodes and the eigenfrequencies of the system. The famous question related to this problem is: *Can you hear the shape of the drum*?

We solved the eigenvalue problem for a square, a rectangle, and a circle shape of membrane. The `scipy.sparse.linalg.eigs()` solver was used. The main focus was on observing the eigenvectors corresponding to the two smallest eigenvalues. In Figure 2 the square shape was used, in Figure 3 a rectangle shape, and finally on Figure 4 a circle shape. We can observe that the underlying shape influences the behavior of the eigenvector.

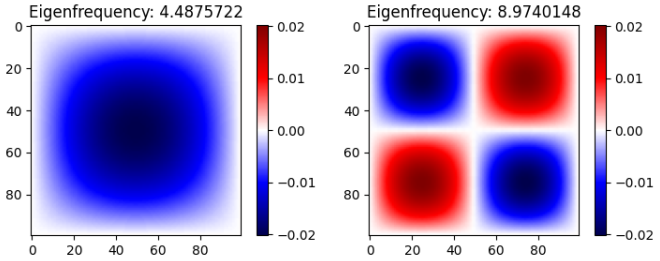


Fig. 2: Eigenvectors for some of the smallest eigenvalues for square shape with side L. Plots are labeled with their frequencies.

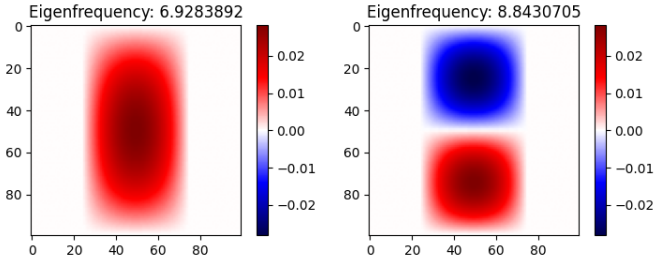


Fig. 3: Eigenvectors for some of the smallest eigenvalues for rectangle shape with sides L and 2xL. Plots are labeled with their frequencies.

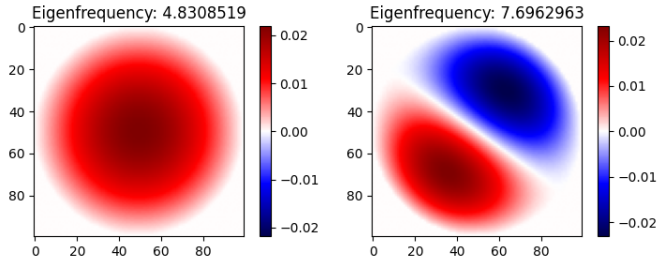


Fig. 4: Eigenvectors for some of the smallest eigenvalues for circle shape with diameter L. Plots are labeled with their frequencies.

Given that the Laplacian matrix has a lot of zero entries, the calculation of eigenvalues can be further sped up, by introducing the sparse matrices. Using the `scipy.sparse.linalg.eigs()` we computed the eigenvalues and eigenvectors for different number of discretization steps (N). We did it for both a dense matrix and a sparse matrix and compared the time needed for calculation. The results are presented in the Figure 5. The time needed to compute a dense matrix is considerably longer than the time needed for a sparse matrix. This difference is increasing with N. This result is not surprising, given that the use of sparse matrices requires significantly less memory.

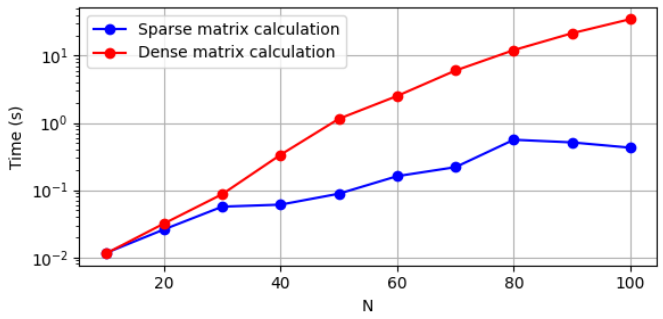


Fig. 5: Comparison in time for calculation, for `scipy.sparse.linalg.eigs()` between sparse and dense matrices. The comparison was done for the circle domain and averaged over multiple runs.

Furthermore we investigated the influence of the number of discretization steps N on the spectrum of eigenfrequencies. The results are presented on Figure 6. For each eigenfrequency, the frequency of occurrence is measured and plotted. The same was repeated for $N \in [20, 30, 40, 50]$. All results were normalized, for easier comparison. There is no significant change in the spectrum of eigenfrequencies when the number of discretization steps change from 20 to 50. For all of them, we observe the peak around two. This eigenvalue is most frequently appearing for all numbers of discretization steps because the Laplacian matrix has mostly -4 on the diagonal. The reason why the peak can be observed in the same place for all N is because the eigenfrequencies are normalized by N. Otherwise, the peak would shift to the right with increasing N.

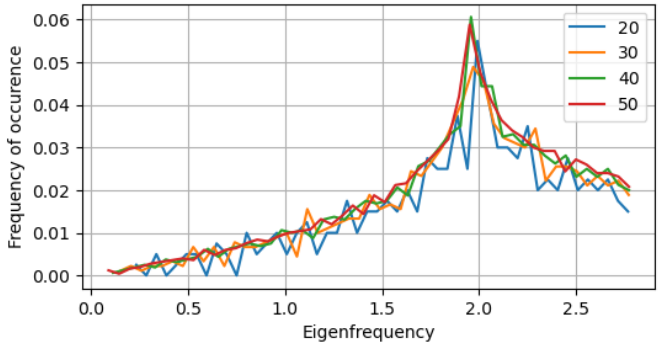


Fig. 6: Eigenfrequencies depending on the number of discretization steps (N) for the square shape. The eigenfrequencies are normalized by dividing by N.

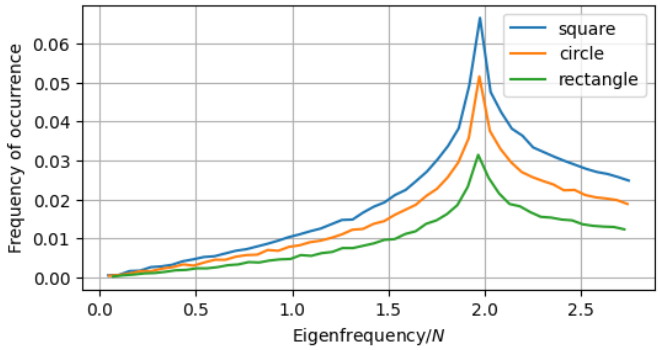


Fig. 7: Spectrum of the Eigenfrequencies for the three drum shapes given $N = 100$ discretization steps. Note that the sparse eigen-solver used did not output the $|S|$ largest out of $|S|^2$ total eigenvalues.

Lastly, the influence of the L value on the spectrum of eigenfrequencies was investigated. We tracked minimum eigenfrequency while varying the L value. The reason only the minimum is tracked is that the same behavior was observed for the entire spectrum of eigenfrequencies. We track the minimum for all domain shapes. The results are shown on the Figure 8. For all shapes the minimum eigenfrequency decreases when the L value is increased.

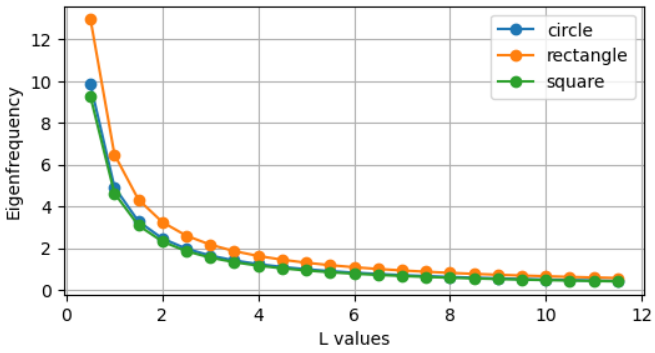


Fig. 8: Minimum eigenfrequency depending on L value, for different shapes.

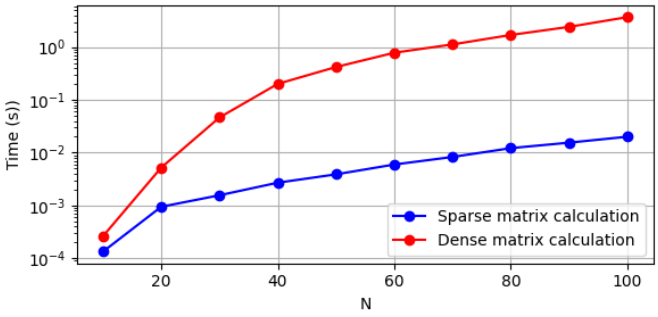


Fig. 10: Average log runtime time in seconds for a range of grid granularities N for both the SciPy Dense linear algebra solver `scipy.linalg.solve` and the Sparse linear algebra solver `scipy.sparse.linalg.spsolve`

B. Steady-State Diffusion using Direct Methods

The Matrix equation $\mathbf{Mc} = \mathbf{b}$ of the discretized steady-state equation Equation 8 allows for the solution of a circular domain using the `scipy.sparse.linalg.spsolve()` method. The steady-state found using this direct method returns a concentration distribution within the circular domain in Figure 9, expressing a peak at the source location with a gradual decrease toward the boundaries.

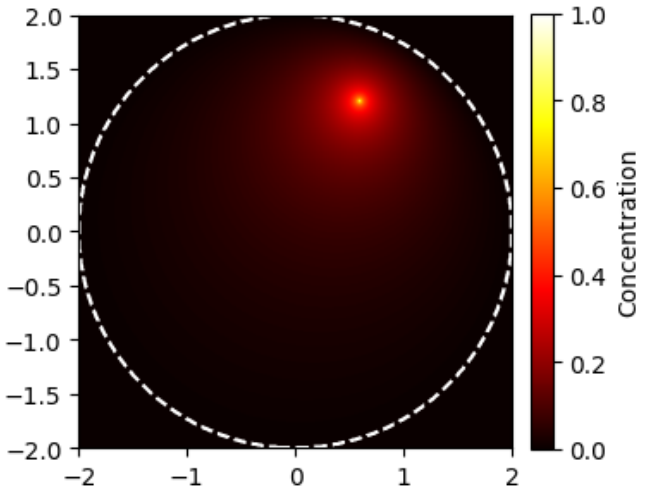


Fig. 9: Steady State Diffusion for circular drum with with $R = 2$ radius, $N = 100$ grid steps, and a source at $(0.6, 1.2)$

Using a dense matrix solver `scipy.linalg.solve` and a sparse matrix solver `scipy.sparse.linalg.spsolve` in Figure 10 once again shows that sparse matrices are more efficient to calculate and see a magnitude lower calculation time.

C. Harmonic Oscillator

In our experiments the initial state is always a static particle with displacement of one length unit and zero velocity. Figure 11 shows the development of the amplitudes of oscillation over long simulation times. Our implementation of the leapfrog method is compared to the implementations `RK23`, `RK45`, `DOP853` of the Runge-Kutta method of order 3, 5, and 8 respectively. For all Runge-Kutta methods, the amplitude decays to zero, showing that these methods are not stable on the problem. Surprisingly, `DOP853` diverges faster than the lower order `RK45` method. Not shown here is the amplitude of the `LSODA` method, which increased exponentially. In contrast to these methods, the leapfrog method keeps a constant amplitude for all tested time intervals.

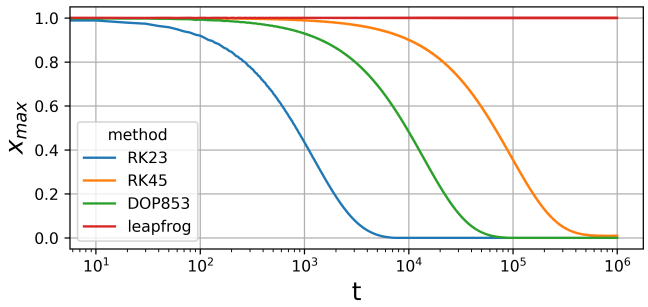
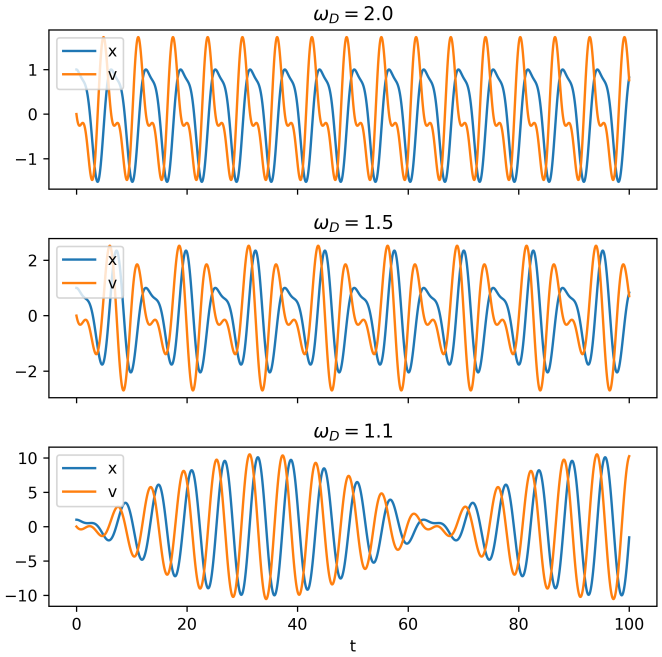
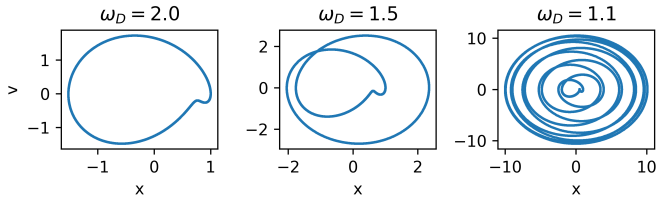


Fig. 11: time-development of the amplitude of oscillation with four different numerical methods



(a)



(b)

Fig. 12: Time development (a) and phase diagram (b) of displacement and velocity of the driven harmonic oscillator with a resonance frequency of 1 and driving frequency $w_D \in \{1.1, 1.5, 2.0\}$

Adding a driving force $f(t) = \sin(w_D t)$ to the harmonic oscillator, we observe periodic motion with longer cycles, as shown in Figure 12. Higher amplitudes are reached when the driving and spring force act in the same direction. Near the resonance frequency, the two forces are synchronized for longer time intervals, leading to longer cycles with higher maximum amplitudes. In the limit $w_D \to w_{res}$, the two forces always act in the same direction, so the cycle length and amplitude grow infinitely large.

V. Conclusion

Circling back to the initial question: Can we hear the shape of the drum? More specifically, can the membrane shape be uniquely identified from its eigenvalue spectrum? Even though we do observe some differences between different shapes it is impossible to uniquely distinguish them, only based on the eigenfrequencies spectrum.

Beyond spectral analysis, the results highlight the importance of using sparse matrices in large-scale computations as the resulting Laplacian matrix for system with N discretization steps is N^4. The use of sparse matrices reduces memory overhead and improves computation speed especially in the discretized finite difference systems represented by symmetric diagonal matrices, thus making them important in exploring diffusive systems.

Additionally, the importance of method selection for simulations becomes apparent in the simulation of the harmonic oscillator, where the leapfrog method demonstrates improved stability compared to the general purpose Runga-Kutta methods.

VI. Appendix

A.

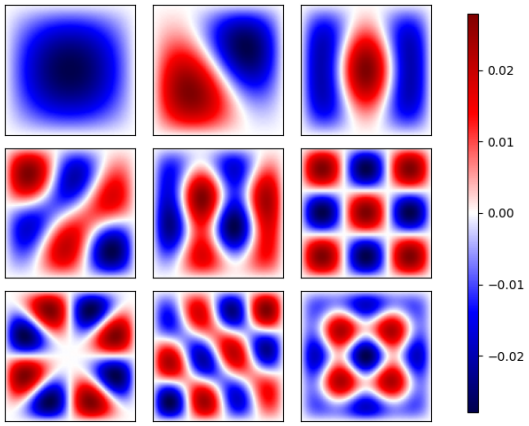


Fig. 13: Eigenmodes plotted for every other mode in an ascending range of eigenfrequencies for a square shape with side $L = 1.0$.

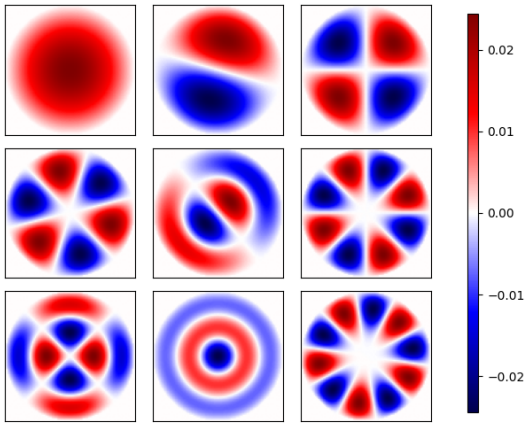


Fig. 14: Eigenmodes plotted for every other mode in an ascending range of eigenfrequencies for a circle shape with diameter $L = 1.0$.