Fractal Drums

Knut Andre G. Prestsveen

March 10, 2020

1 Abstract

A numerical study of vibrations of fractal shaped drums. The 10 lowest vibration eigenmodes for a drum with a fractal shaped boundary were computed using finite diferences, and the scaling of the integrated density of states were compared with the Weyl-conjecture and results for drums with smooth boundary.

2 Introduction

For any drum, it's size is related to the pitch of the sound it makes, and the larger the area, the lower the pitch. The question in mind for this study, is if the shape of the drum also affects the tone so that information about the drum's shape can be inferred from the tone, and this study spesifically studied drums shaped like quadratic Koch fractals.

The study was carried out numerically, and program code for generating the fractal and a system lattice, representing the drums boundary and skin respectively, were written. The eigenmodes of the drums vibration were then computed with finite differences, and to see if the Weyl-conjecture could could be used to infer information about the fractal boundary, the integrated density of states of the fractal drum were compared to that of a drum with smooth boundary and same area.

3 Theory

3.1 The Weyl-conjecture

The shape of the drum cannot be completely inferred from it's sound alone, but some information can be extracted. Firstly the area of the drum A can be found from

$$A = 4 * \pi..., \tag{1}$$

where $N(\omega)$ is the integrated density of states (IDOS), i.e. the number of eigenfrequencies smaller than the frequency ω. Also Hermann Weyl conjectured that the second term in the asymptotic of $N(\omega)$ gives the length of the drums perimeter L, so that

$$N(\omega) = \frac{A}{4 * \pi}... \tag{2}$$

in the limit of large ω. Equation 2 is the Weyl-conjecture for the IDOS, and is proven correct under the assumption of a smooth boundary. This study tries to see what happens when the boundary is fractal.

3.2 Quadratic Koch Fractal

The quadratic koch fractal is recursively generated by starting out with a square, dividing each of the sides into four equal pieces, and displace the two middle pieces one side length, one of them up the other down. This process is repeated recursively until reached desired recursion depth. The process is illustrated in 1.

Figure 1: Koch fractal

3.3 The Helmholtz equation - Finite Difference Approximation

The displacement of the drum's surface $u(r,t)$ is given by the wave equation **??**, with the boundary conditions $u = 0$ on the boundary $\partial\Omega$. Performing the Fourier transform of the wave equation **??** results in the Helmholtz equation **??**, and turns the problem into an eigenvalue problem.

$$waveeq \tag{3}$$

$$HHeq \tag{4}$$

The eigen values, and vectors of equation **??** were calculated using finite differences, with both the stantard five-point stensil and the nine-point stensil approximation to the laplacian operator. The discretization of equation **??** becomes

$$fivepoint \tag{5}$$

and

$$ninepoint \tag{6}$$

for the five point and nine point approximations respectively.

4 Code

The program code is written in the julia language, the setup of the system is split into several files, which are all included in `setup.jl`. The main script is `main.jl`, which includes `setup.jl` and solves the eigenvalue problem using the `eigs` routine for sparse matrices from the ARPACK library.

4.1 Fractal creation

The generation of the fractal is just a straight forward implementation of the recursive method described in 3.2. The code was split into three functions: firstly there is `generate_side` which takes the start and end point of a side and returns the "transformed" side and `generate_corners` which calls `generate_side` for every side in the current fractal shape and for every level recursively, returning the alle the positions of the fractals corners. Finally there is `fill_edges` which places points on the sides between the corners, according to the specified grid constant.

4.2 Determation of inside points

The determination of which points are inside is done with a breadth first like search, starting in the middle point which is known to be inside the fractal. Another method which was implemented starts in a point, scans towards the edge in one direction and uses the orientation of the curve to determin if the point was inside or not. The latter requieres that the fractals points are ordered correctly, was considerably slower, and while it could have been sped up considerably by determining every point along each scan in one go, this was not done since the former method seemed more elegant. The lattice is first initialized with every element being `OUTSIDE`, and the breadth search code is shown in listing 4.2.

```
# --snip--

points = [left, right, down, up]
println("begin search")
for point in points
    x, y = point
    if lattice[x, y] == OUTSIDE
        # New INSIDE point
        lattice[x, y] = INSIDE
```

```
            # Add nearest neighbours to points
            for nn in (-1, 1)
                   push!(points, (x + nn, y))
                   push!(points, (x, y + nn))
            end
        else
            continue
        end
    end

    # --snip--
```

4.3 Datastructures

The drum's surface is represented by a lattice, which in the code is a 2D array containing the enum type `Location`, which can take the values `INSIDE`, `OUTSIDE` and `BORDER`. Each points `Location` value is determined by the process described in section 4.2, and the positions of the points on the lattice are corresponding to the `CartesianIndices` of the array.

For the alternative method of lattice creation, a slightly different representation was used. The lattice was then an array of the custom type `Point`, which contains the membervariables `x::Int`, `y::Int` and `location::Location`. The definitions of `Point` and `Location` is shown in listing 4.3.

```
@enum Location INSIDE OUTSIDE BORDER


struct Point
    x::Int
    y::Int
    location::Location
end
```

Creation of laplacian matrix

The negative laplacian matrix construction is also quite straight forward. It constructs the matrix row by row by looping through every lattice point inside the fractal, setting the diagonal elements to 4 and the nearest neighbour elements to -1 if the neighbour also is an inside point. The definition of the function setting up the five point matrix is shown in listing 4.3, and the nine point version is completely analogous.

```
function five_point_laplacian(N, lattice, points_inside)
    println("Making laplacian matrix")
    lap_matrix = spzeros(N, N)
    for (idx, p) in enumerate(points_inside)
        x, y = p
        lap_matrix[idx, idx] = 4
        for nn in (-1, 1)
            if lattice[x+nn, y] == INSIDE
                nn_idx = findfirst(p -> p== (x+nn, y), points_inside)
                lap_matrix[idx, nn_idx] = -1
            end
            if lattice[x, y+nn] == INSIDE
                nn_idx = findfirst(p -> p== (x, y+nn), points_inside)
                lap_matrix[idx, nn_idx] = -1
            end
        end
    end
    return lap_matrix
end
```

Figure 2: Quadratic koch fractals of recursion depth 2 and 3, generated by the code described in 4.1

5 Results

Cool plots, and hopefully something clever about IDOS.

5.1 Determin inside comparison

Compare runtimes I guess

5.2 Solutions to EV

Table of eigenvalues (at least 10), ref to plots

5.3 Largest value of L, memory

L = 4, refer to possible suggestions in later section

5.4 IDOS

5.5 Compare 9p to 5p stensil

Compare some numbers, ev. runtimes

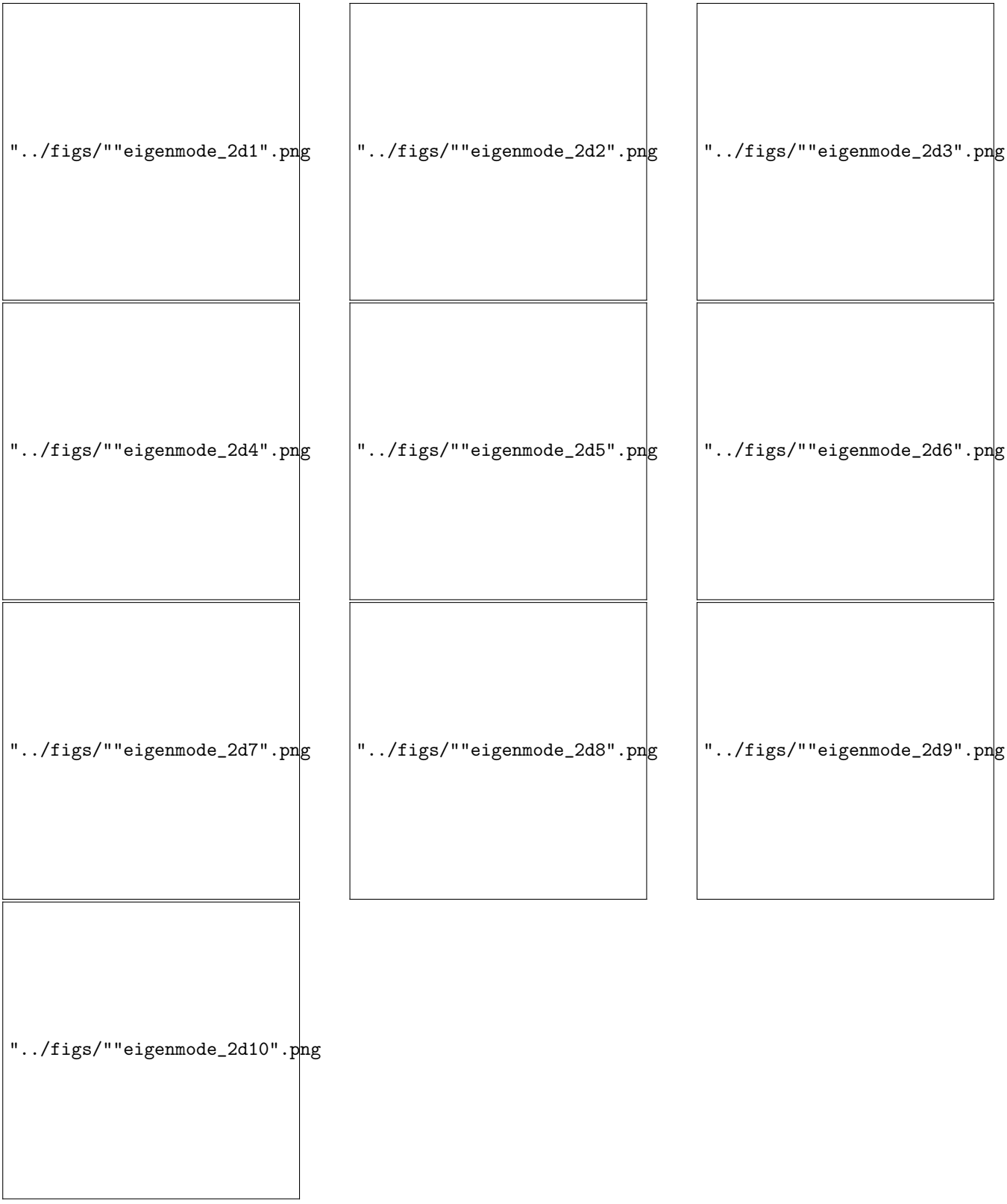

Figure 3: The 10 lowest eigenmodes, 2D plots.