Fractal Drums

Knut Andre G. Prestsveen

March 11, 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

$$A = 4\pi \lim_{\omega\to\infty} \frac{N(\omega)}{\omega^2}..., \tag{1}$$

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

$$N(\omega) = \frac{A}{4\pi}\omega^2 - \frac{L}{4\pi}\omega + ... \tag{2}$$

in the limit of large ω. Equation **??** 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.

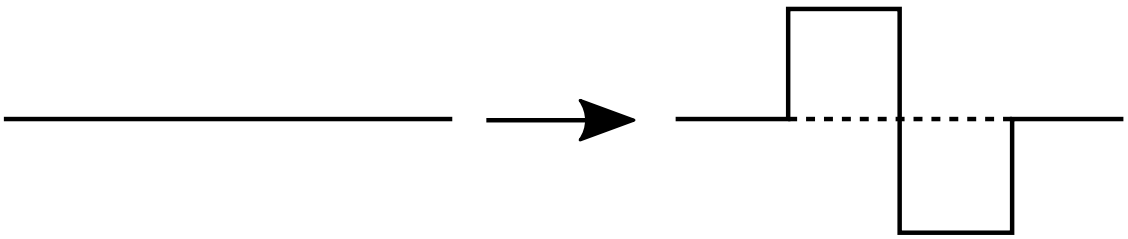

Figure 1: Koch fractal

3.3 The Helmholtz equation - Finite Difference Approximation

The displacement of the drum's surface $u(r,t)$ is given by the wave equation 3, with the boundary conditions $u = 0$ on the boundary $\partial\Omega$. Performing the Fourier transform of the wave equation 3 results in the Helmholtz equation 4, and turns the problem into an eigenvalue problem.

$$-\nabla^2 u = \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2} \tag{3}$$

$$-\nabla^2 U(\mathbf{x},\omega) = \frac{\omega^2}{v^2}U(\mathbf{x},\omega) \tag{4}$$

The eigen values, and vectors of equation 4 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 4 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 an integer value representing the points location. Every point outside the fractal is set to `OUTSIDE_POINT = -1` and points on the border are set to `BORDER_POINT = 0`. The points inside are numbered by positive integers, which corresponds to their inner index in the later contructed laplacian matrix. 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`, where the latter is a custom enum type. 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
```

5 Results and discussion

5.1 Determin inside comparison

The breadth first search is much more efficient than the naive traverse search, and can hangle much larger lattices. The traverse search can probably be improved upon to be comparable to the breadth search, but as the implementations are now the former method takes seconds where the latter method spends minutes. The breadth first search is efficient enough so that it never is a bottle neck for the computation.

5.2 Solutions to EV

Table of eigenvalues (at least 10), ref to plots The highes fractal level my computer and code was able to handle was level 5, and table 5.2 shows the 10 lowest eigenvalues at different levels and grid resolutions. The limitation of the computation is the `eigs` routine from Arpack, which is the most time and memory consuming part of the code for higher levels, and also plotting of the eigenstates takes quite some time at for high levels and grid resolutions. The correspunding eigenstates, but for a level 4 fractal, are shown in figure 5

level=5, gridres=1	level=4, gridres=1	level=3, gridres=4
2.1172036732026874e-5	0.00033779615002478065	0.0008493414391754176
4.761205170693104e-5	0.000758455377648222	0.0018882176645521438
4.761205170693165e-5	0.0007584553776482244	0.0018882176645521613
4.946353369683679e-5	0.0007878141802701458	0.0019595798224599993
4.999806796113937e-5	0.0007964587750657883	0.0019832717170220697
5.416492483189804e-5	0.0008640818302252861	0.0021717340723473185
5.4164924831898765e-5	0.0008640818302252886	0.0021717340723473276
7.421330675523835e-5	0.0011835480338816295	0.0029687326189671865
8.512772404960623e-5	0.0013571443191111746	0.003396846897119718
9.011148145688274e-5	0.0014365248772246742	0.0035942764169037233

5.3 Largest value of L, memory

As mentioned, the highest level I was able to compute the eigenvalues at, was level 5, and that was only with a low grid resolution. For higher levels the process is simply killed because it runs out of memory, and my computer has $8GB$ of memory, but in reality $7.66Gb$ is available according to htop. Simply storing the laplacian matrix does almost not use any memory at all, because it is very sparse and only contains two different integer values. It is stored as a `SparseArray`, and the `eigs` solver from Arpack is a sparse solver. For level 5, grid resolution 1, the laplacian matrix requires only $4e-5MB$, which is a negigible amount of memory compared to the computers RAM.

This means that the thing preventing computation of higher levels is Arpack's `eigs`, and therefore it is not easy to think of anything that can be done to compute for higher levels, except for simply using a more powerfull computer with a lot of memory. However, figure 4 reveals that the laplacian matrix is not only very sparse, but also symmetric, and this can possibly be exploited to save some computer resources.

Additionally, I read about a similar problem *REF?* where they had defined the stensil to use the diagonal neigbouring points, and indexed the lattice as a chess board. This allowed them to split the computation in two

concurrent parts, as odd and even indexed points could be computed completely independent of each other. This could also perhaps implemented for this problem.

5.4 IDOS

According to the wayl conjecture, the integrates density of states is supposed to scale as ω^d, where d is th borders fractal dimension. In the limit of a smooth perimeter, the Koch fractal has $d = 1.5$, so we expect our estimates for d to approach this value for higher level fractals. The plot in **??** shows the regression curves for ω^d, and the estimates for d at each level is listed in table 5.4. Note that the reason there are so few eigenfrequencies for level 5 is that the number of values I was able to compute was limited by my computers memory.

level	gridres	d
3	4	1.3126325672146508
4	1	1.3736896123492528
5	1	1.4994142369831907

5.5 Compare 9p to 5p stensil

An implementation of a nine point stensil approximation to the laplacian was also implemented...

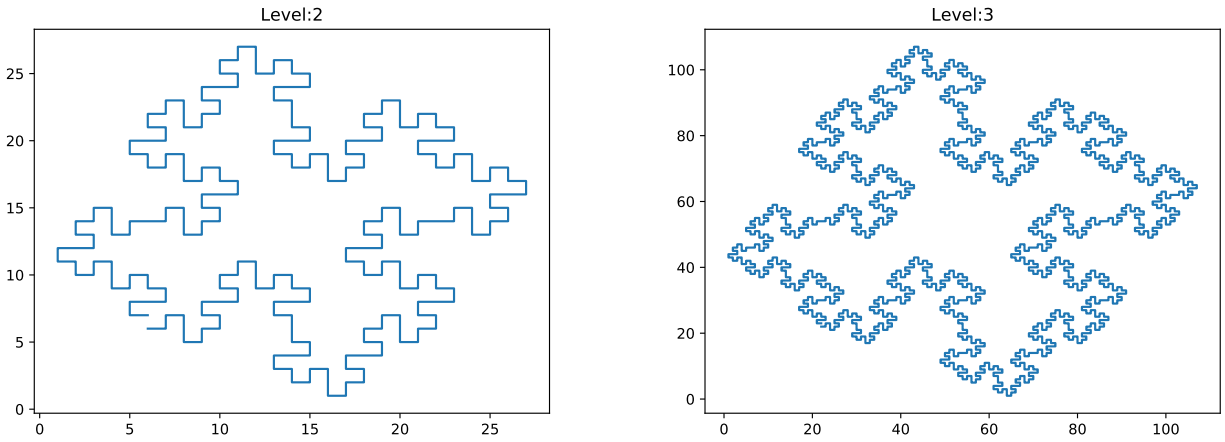

Figure 2: Quadratic koch fractals of recursion depth 2 and 3, generated by the code described in 4.1

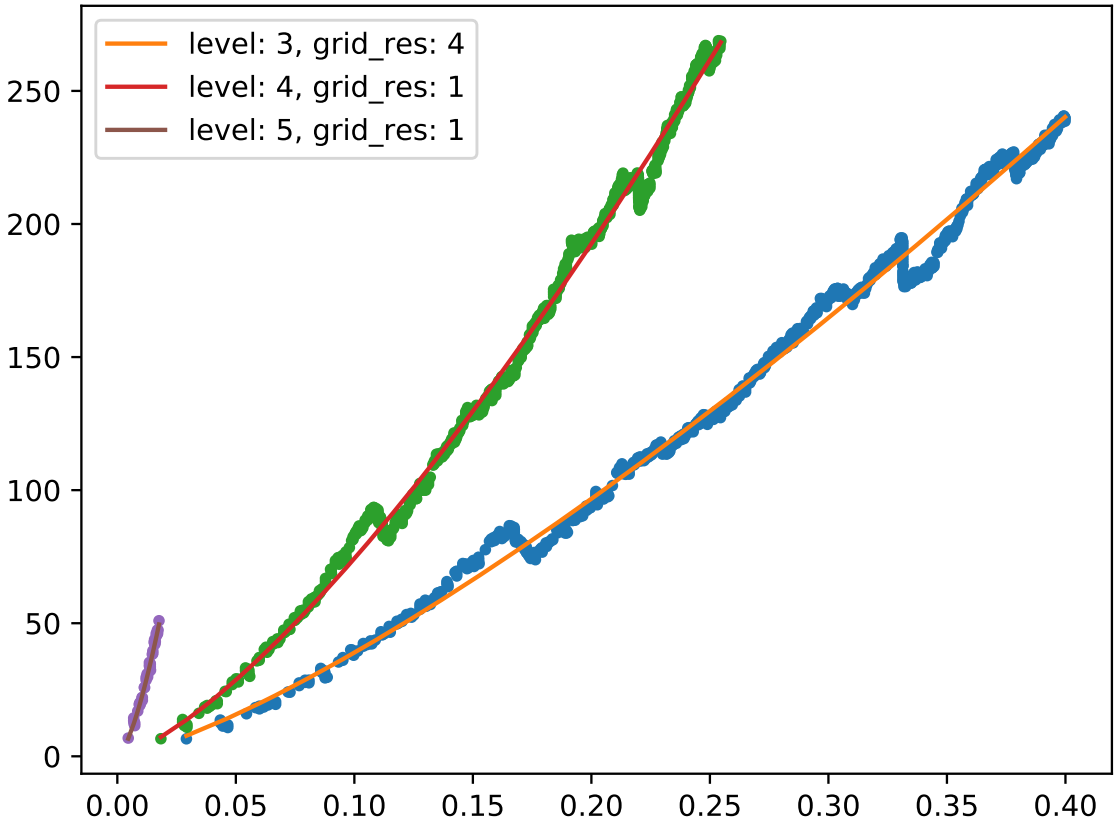

Figure 3: Regression curves for the scaling of the eigenfrequencies.

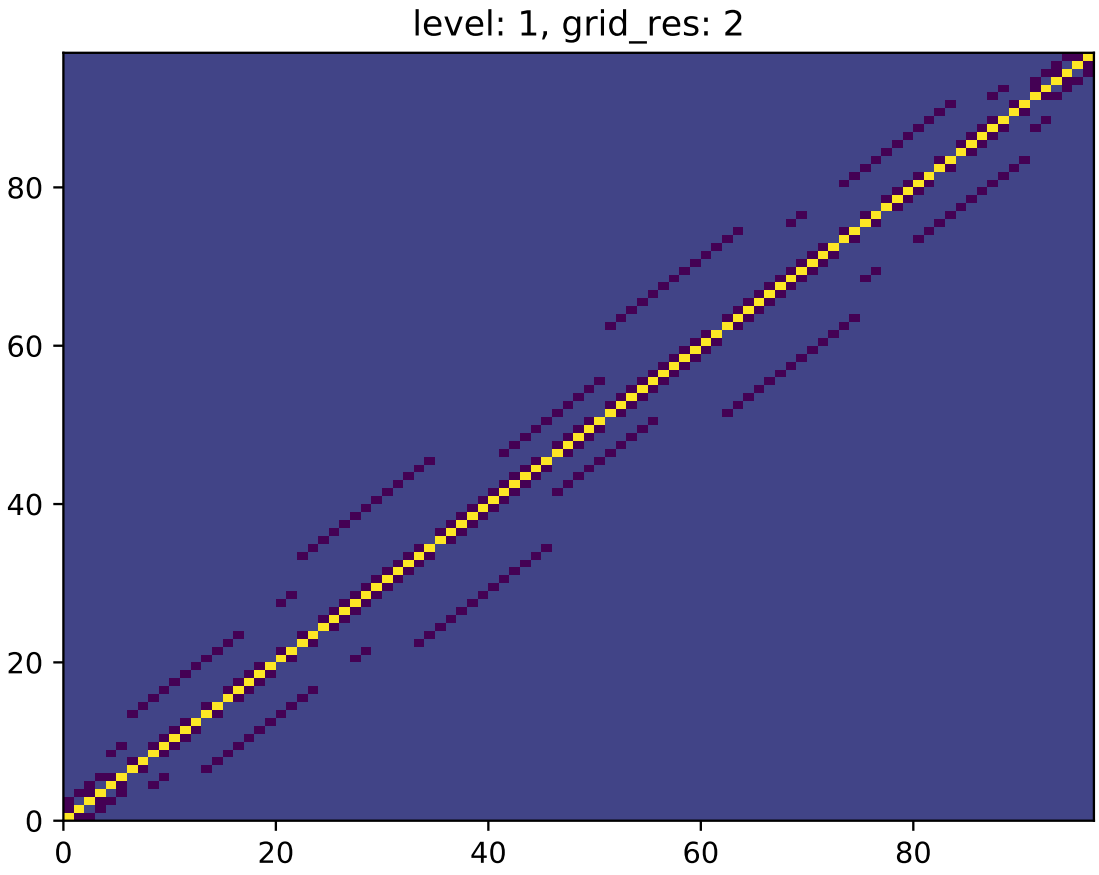

Figure 4: The laplacian matrix for recursion depth 1 and grid res 2.

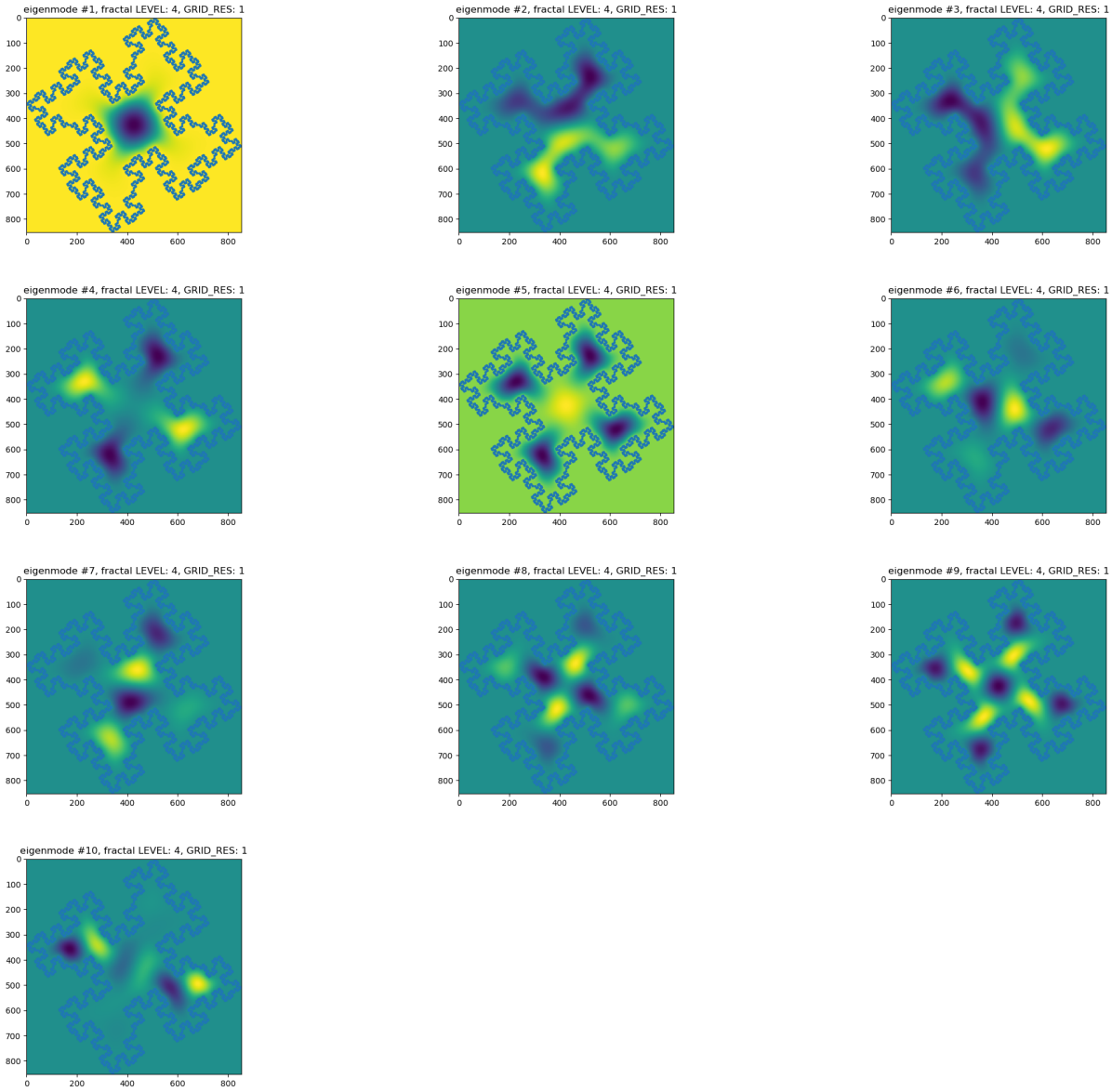

Figure 5: The 10 lowest eigenmodes, 2D plots.