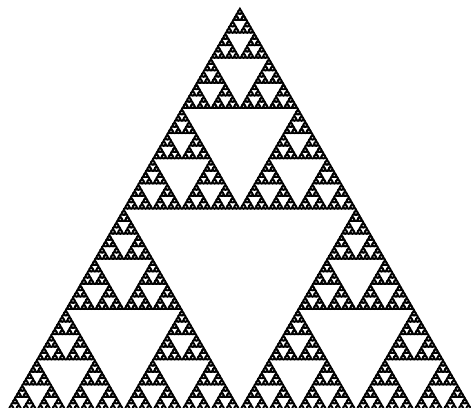


Three fractal problems

Here are three potential research problems based in fractal geometry. In addition to being very geometric, the solutions of all these problems involves a fair amount of discrete mathematics and/or graph theory. Thus, these problems fit nicely into the overall theme of our REU.

I would say that the projects get progressively more involved. For each problem, I've indicated a mathematical level and, since all these problems have a computational aspect, a *Mathematica* level. Of course, other programming tools can be used but Patrick and I both happen to work with *Mathematica* and it is available here at UNCA.

The Sierpinski triangle



A fractal triangle with overlap

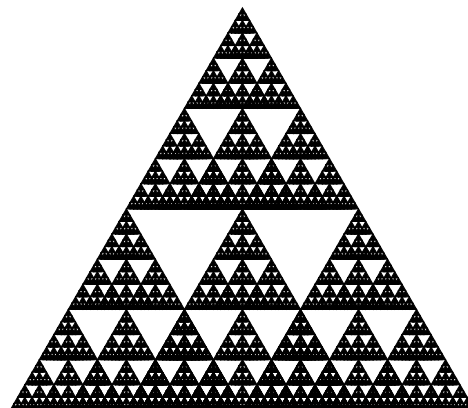


Figure 1: Two major fractal examples.

The problems involve extending known concepts from fractal geometry that are well understood for the Sierpinski triangle, shown on the left in figure 1, to the more complicated fractal shown on the right in figure 1. The Sierpinski triangle (let's call it S , from now on) consists of three copies of itself scaled by the factor $1/2$. As we will learn, this implies that its fractal dimension is $\log(3)/\log(2) \approx 1.58$. The fractal on the right (let's call it T) consists of four copies each scaled by the factor $1/2$, suggesting that its dimension might be $\log(4)/\log(2) = 2$. It certainly doesn't look like a plain old two dimensional set, though. The problem is that there is substantial overlap between the pieces and this complicates virtually all complications involving T or fractals like T . Recent work on fractals with overlap shows that the dimension of T is $\log(2 + \sqrt{2})/\log(2) \approx 1.77$. What else can we say about T and other fractals with overlap?

Comments: There's more to this than just generating a cool picture. As we will learn, the description of S in terms of a cellular automaton leads to a recursive formula for the number of ones to a certain level that ultimately allows us to compute the fractal dimension of S . It would be nice to accomplish the same thing for T . It would be even better if we can accomplish this for some family of fractal sets generalizing T . Figure 4, for example, shows the first 625 rows of Pascal's triangle mod 5; the non-zero elements are shaded black and the zeros are white. Much of it appears gray, since the black and white are intermingled to some extent.

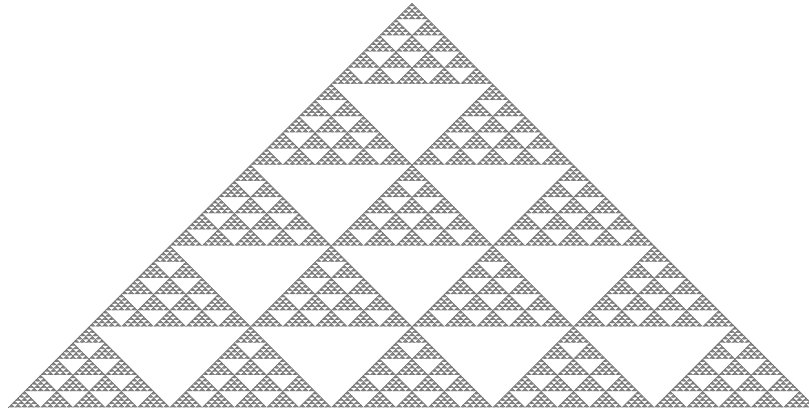


Figure 4: Non-zero elements of Pascal's triangle mod 5.

Mathematical level

This is the most elementary of the problems on this list. Elementary does not mean easy, by the way. It simply means that there is not a lot of prerequisite material required to get started on the problem. Indeed, we can learn most of what is needed about cellular automata in an afternoon; you could get started right after that. The required fractal geometry would take a couple of days before you could really start on the dimension computation.

It's not out of the question that a clever student could solve the main problem fairly quickly. On the other hand, as far as I know, there is no appropriate generalization to do this problem. As we will learn, there are only 256 so called *elementary cellular automata*. These are one-dimensional, two-state cellular automata whose update rule is a simple linear combination of x_i and its two nearest neighbors. I'm virtually certain that none of these will generate T . I think it likely that some reasonable generalization will generate T , however.

Mathematica level

This problem will likely call for a fair amount of experimentation. *Mathematica* is a great tool for this purpose. In fact, the first eight rows of Pascal's triangle mod 2, with the ones highlighted can be generated using the following one line command.

```
Grid[CellularAutomaton[90, {{1}, 0}, 7]] /.
  0 -> Style[0, LightGray]
```

Thus, I would anticipate this problem involving a fair amount of *Mathematica*.

Topic 2: Computational aspects of finite-type computations

As mentioned in the introduction, the fractal dimension of S is $\log(3)/\log(2)$. More generally, the fractal dimension of a set that consists of N copies of itself scaled by the factor $r < 1$ is $\log(N)/\log(1/r)$, *provided* that the copies don't overlap too much. This computation (and the precise definition of overlap) is made explicit in a 1946 paper of Moran, but the fundamental ideas go back even before that. Computation of fractal dimension in the presence of overlap is much more difficult, however. Probably, the definitive paper is "Hausdorff dimension of self-similar sets with overlap" by Sze-Man Ngai and Yang Wang that appeared only in 2000. This defines a certain class of self-similar sets, generated by so-called *finite-type iterated function systems*, that admit a procedure to determine the fractal dimension. It's important to emphasize that this is a procedure, as opposed to a simple formula. That is, there is an algorithm that produces a matrix; the fractal dimension is then computed in terms of the eigenvalues of that matrix. The algorithm is fairly complex and tedious; it cries out for a computer implementation. In fact, I've done this so (after my `FiniteTypeIFS` package has been installed) the dimension of T may be computed as follows.

```
Needs["FractalGeometry`FiniteTypeIFS`"];
A = {{1/2, 0}, {0, 1/2}};
finiteTypeIFS = {{A, {1/4, Sqrt[3]/4}},
  {A, {0, 0}}, {A, {1/4, 0}}, {A, {1/2, 0}}};
conditionSet = {{0, 0}, {1, 0}, {1/2, Sqrt[3]/2}};
{r, T, V} = FiniteTypeInfo[finiteTypeIFS, N[conditionSet]];
spectralRadius = Max[Abs[Eigenvalues[T]]];
dimension = -Log[spectralRadius] / Log[r]
```

$$\frac{\text{Log}\left[2 + \sqrt{2}\right]}{\text{Log}[2]}$$

Very roughly, the algorithm works by decomposing the larger set into smaller sets of certain types, then iteratively decomposing these into even smaller sets of potentially different types, and continuing until no new types are generated. (Of course, we need to define "type"). We can estimate how quickly the number of pieces grow by analyzing the way the various types decompose; this leads to a computation of the fractal dimension.

The problem

There are sufficient conditions that guarantee that some iterated function systems will be of finite-type but there is no general, a priori way to know how deep the iteration will need to be performed. Furthermore, the computations described above are performed with floating point approximations that lead to round off error. Thus, if the iteration is performed too deeply, the results will be no good. To accept results from the program, we need a bound on the number of iterations that can be performed before things go awry. What is this bound, in terms of the generic iterated function system of finite-type?

Baby version: Prove that the finite-type computation for T works. This should not be too hard but understanding it is a good step towards understanding the general problem.

Related problem: Suppose we apply the finite-type algorithm to an iterated function system that is not of finite-type and stop the process artificially. Can we draw any conclusions about the dimension of the attractor?

Mathematical level

This requires a bit more knowledge than problem 1. We'll need to discuss fractal dimension and the finite-type algorithm in detail; we should be able to do this in less than a couple of weeks. At its core though, this is a problem in numerical analysis. With each level of iteration, distinct points in the approximation are closer to one another. We need to know when these points are distinct, in spite of round-off error. These types of statements are made in terms of inequalities, i.e. the epsilonics that you learn in a real analysis class. I think the ideal student will have studied some real analysis.

On the other hand, I have no doubt that such an inequality exists. The problem should be quite doable and should yield a nice result that will accompany a paper I'm writing on the topic.

Mathematica level

While you might want to experiment with the algorithm using *Mathematica*, this is likely to be much more of a "sit down and think" sort of problem. You can approach it without having to delve into too much *Mathematica*.

Topic 3: Differential equations on T

Suppose that we made an actual Sierpinski triangle - or at least a very fine approximation to a Sierpinski triangle. What physical properties would it have? Suppose, for example, that it vibrated perpendicularly to the plane that contains it at rest. A fundamental mode of vibration of S is shown in figure 5.

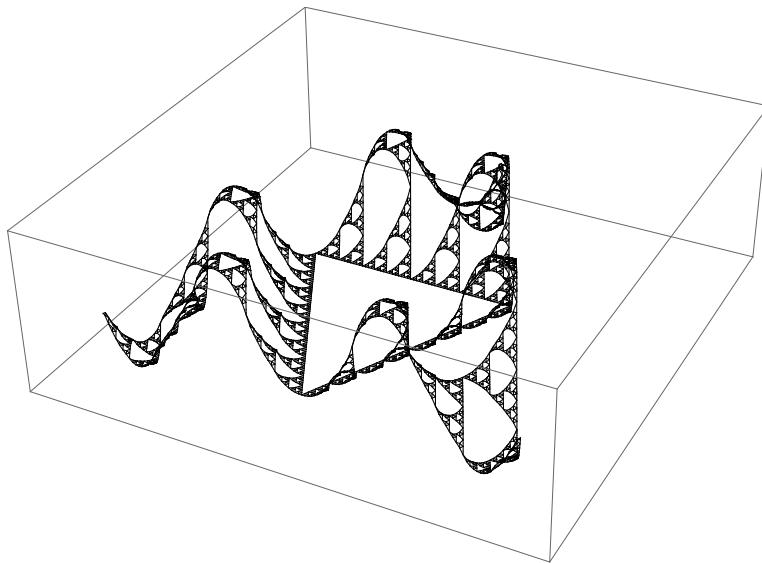


Figure 5: A vibrational mode of S

The problem of fractal vibration belongs to the growing field of differential equations on fractals. You can view actual vibrations and heat diffusion in these animations:

<http://facstaff.unca.edu/mcmcclur/class/FractalGeometry/SierpinskiDrivenVibration.gif>
<http://facstaff.unca.edu/mcmcclur/class/FractalGeometry/SierpinskiPluckedVibration.gif>
<http://facstaff.unca.edu/mcmcclur/class/FractalGeometry/SierpinskiHeatKernelColor.gif>

In order to generate these images, we need versions of the heat equation $u_t = k\Delta u$ and the wave equation $u_{tt} = c^2\Delta u$ that are valid on fractals. Since t represents the continuous quantity time, the key issue is the Laplacian Δ , which is just the sum of the second derivatives of the spatial variables in normal PDEs. Figure 5 and the animations are all based on a recently developed numerical approach to the fractal Laplacian.

The problem

Can the fractal Laplacian techniques that work so well on S be applied to T ?

Comment: I'm not sure if the answer is yes or no. I think it is likely that we can generate some experimental results but analytic proofs might be very difficult.

Mathematical level

Higher than problem one but not quite so high as problem two. After learning the basics of iterated function systems (fractal dimension is not so important here), we'd discuss partial differential equations from a numerical perspective and the generalization of the Laplacian to S . We should be able to get through most of the material in a couple of weeks but the interested student will likely have a bit more independent study to do.

Mathematica level

This project would certainly require a lot of detailed *Mathematica* programming. The ideal student will have a desire to develop these programming skills, as well as an interest in the problem.