

The finite type algorithm

The algorithm that allows us compute the box-counting dimension of a self-similar set in the presence of overlap in certain special cases is called the *finite type algorithm*. We now present a careful description of this algorithm and the potential computational research problem that arises.

Illustrative example

First, we need to develop a little useful notation associated with an IFS $\{f_i\}_{i=1}^m$ with ratio list $\{r_i\}_{i=1}^m$, invariant set E , and similarity dimension s . A string with symbols chosen from $\{1, \dots, m\}$ is simply a finite or infinite sequence with values in $\{1, \dots, m\}$. A finite string α will be denoted $\alpha = i_1 \cdots i_k$. Given a positive integer k , let J_k denote the set of all strings of length k with values chosen from the set $\{1, \dots, m\}$. Let $J_* = \bigcup_{k=1}^{\infty} J_k$ denote the set of all such finite strings. Given $\alpha = i_1 \cdots i_k \in J_k$, let $f_\alpha = f_{i_1} \circ \cdots \circ f_{i_k}$ and $r_\alpha = r_{i_1} \cdots r_{i_k}$ (this will just be r^k in our special case). Then J_k induces a k^{th} level decomposition of E given by

$$E = \bigcup_{\alpha \in J_k} f_\alpha(E).$$

These strings can be thought of as addresses for locations in the Sierpinski triangle. We can visualize the addresses easily enough, as in figure 1.

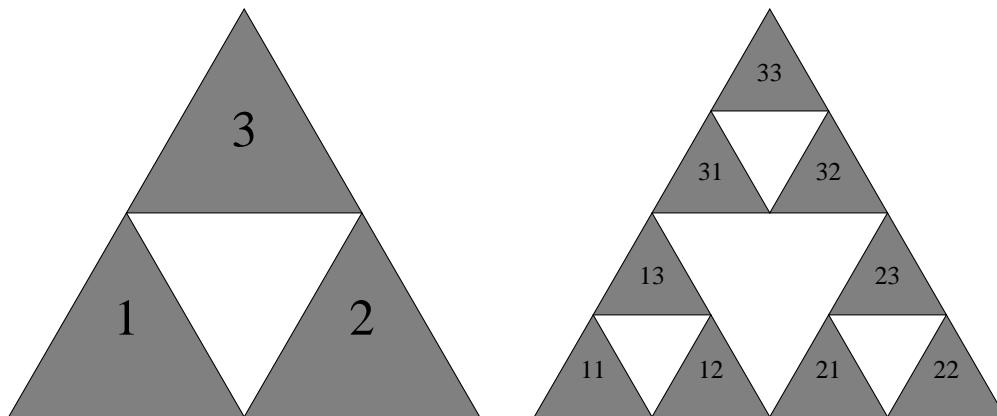


Figure 1: Addresses in the Sierpinski triangle

In the presence of OSC, the addresses and the pieces in the cover induced by the IFS are in one-to-one correspondence. This need not be the case in the presence of overlap. In fact, it is possible

that the intersection of two pieces at one level precisely form a piece at the next level. This is essentially the situation in which the finite type algorithm works. We can illustrate this with my favorite example shown in figure 2.

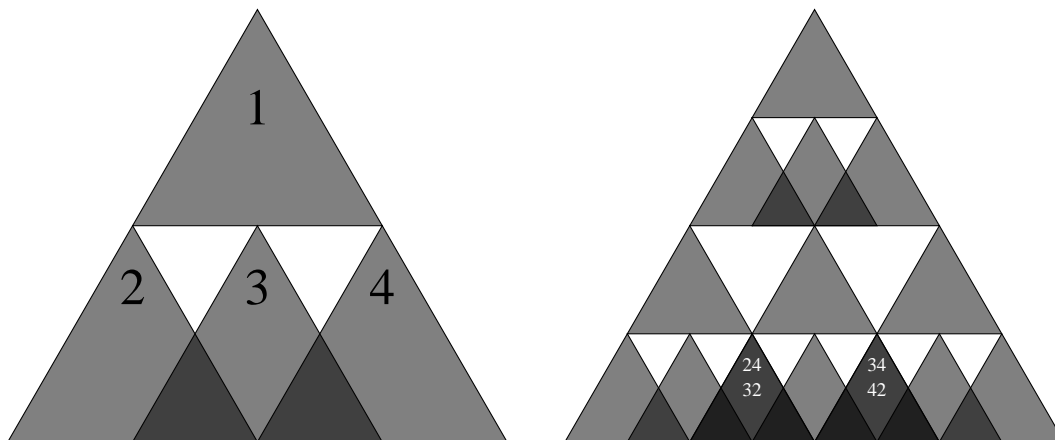


Figure 2: Addresses in the triangle with overlap

Note that the intersection between the level one pieces with addresses 2 and 3 is exactly the level two piece that could be address 24 or 32. If we want to reliably count the number of pieces in the cover using addresses, this needs to be taken into account. Also note that the triangles labeled 2 and 3 fit into the overall set differently. The triangle labeled 2 intersects one other set on the right, while the triangle labeled 3 intersects a triangle on either side. We'll be able to use this to make a precise definition of "type" of set used in the cover. We'll then cover the whole set with subsets of various types and keep track of the way the types decompose using a matrix. As we've seen several times now, the growth rate will be determined by the dominant eigenvalue of this matrix.

Definitions

We now attempt to quantify these ideas precisely enough to yield an algorithm. We suppose that we have an iterated function system $\{f_i\}_{i=1}^m$ of similarities with a common contraction ratio r . We do not assume OSC, but we do suppose we have an open set U called a conditioning set. In the presence of OSC, we cover the invariant set K using sets of the form $f_\alpha(K)$, where $\alpha = (i_1, i_2, \dots, i_n)$ is a word of length n chosen from the alphabet $\{1, \dots, m\}$. Counting these sets, yields the dimensional estimate. Now, we will count the functions f_α directly. As we will see, it is entirely possible that $f_\alpha = f_\beta$, for distinct words α and β ; the major challenge is to systematically account for this possibility.

For each $n \in \mathbb{N}$, let Λ_n denote the set of all functions f_α , where $\alpha \in J_n$, the set of all words of length n . The observation above implies that some functions in Λ_n may have multiple addresses chosen from J_n . Thus, the number of elements in Λ_n may be strictly smaller than the number of elements of J_n . Part of our algorithm will associate a unique address with each function. Two functions f_α and f_β in Λ_n are said to be neighbors if $f_\alpha(U) \cap f_\beta(U) \neq \emptyset$. Given a function f in Λ_n , the neighborhood $\mathcal{N}(f)$ of f is the set of all neighbors of f in Λ_n . Two functions f and g (arising at any levels in the construction) are said to be of the same type if there is a similarity φ such that $f = \varphi \circ g$ and

$$\{f' : f' \in \mathcal{N}(f)\} = \{\varphi \circ g' : g' \in \mathcal{N}(g)\}.$$

In the event that there are only finitely many types, the IFS is said to be of finite type. In this case, a matrix based counting scheme will work.

A few points of clarification are in order. While the definition above is purely algebraic, there is a fairly simple corresponding geometric interpretation. Frequently, the functions f_α are uniquely determined by their action on the conditioning set U . In this case, we can typically tell if two functions f and g are of the same type geometrically. Certainly, the images of U under the functions in $\mathcal{N}(f)$ must form an arrangement geometrically similar to the images under the function in $\mathcal{N}(g)$. Furthermore, the orientations of U under f and under g must be similar.

Application to our favorite example

We now turn to the original example to illustrate another issue. The functions involved may be visualized by their effect on the conditioning set, as shown in figure 2 on the left. This shows that the conditioning set (which we take to be of type 1) breaks up into sets of four types: another type 1, a type 2 that intersects another triangle on the right, a type 3 that intersects triangles on either side, and a type 4 that intersects another triangle on the left. Of course, the precise algorithm states this in terms of functions, but the geometry of those functions is plain to see.

Now, suppose we iterate to the next level. It's clear that, as expected, the type 1 triangle decomposes as the type 1 from level zero. Notice, however that the overlap between the types 2 and 3 triangles cause potential confusion. There is a triangle that could be indexed by either $\alpha = (2, 4)$ or $\beta = (3, 2)$. Is this triangle a descendant of the type 2 or the type 3 triangle from the previous level? Ultimately, it really doesn't matter, as long as we have a consistent way of deciding. That is we want triangles of particular types to decompose into triangles of subtypes consistently. One way to achieve this is to assign addresses to sub-triangles using the smallest possible address in the lexicographical ordering. Thus, the 24 - 32 triangle will be considered to be a type 3 descendant of the type 2 triangle from the level one decomposition. Similarly the 34 - 42 triangle will be a type 3 descendant of the type 3 triangle from the level one decomposition. With this convention, we obtain the following decomposition.

$$\begin{aligned} T_1 &\rightarrow T_1 + T_2 + T_3 + T_4 \\ T_2 &\rightarrow T_1 + T_2 + 2T_3 \\ T_3 &\rightarrow T_1 + 2T_3 \\ T_4 &\rightarrow T_1 + T_3 + T_4 \end{aligned}$$

This has the following matrix representation.

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix};$$

Now we compute the eigenvalues of A .

Eigenvalues[A]

$$\{2 + \sqrt{2}, 1, 2 - \sqrt{2}, 0\}$$

Thus, the dimension of the set is $\log(2 + \sqrt{2}) / \log(2) \approx 1.77$.

The major computational issue

As we see, it is of *crucial* importance that we can tell when two triangles overlap, as opposed to intersecting at a single vertex. Thus, we must account for numerical error, if we perform the computations in floating point arithmetic. We do this, in part, by treating points that are close together (say within ε of one another) as the same. This can only work up to a certain depth, however. Suppose, for example, that v_1 and v_2 are vertices chosen from our initial, polygonal conditioning set and $d = |v_1 - v_2|$ represents the distance between them. Then we need $r^n d > \varepsilon$ in order to distinguish these vertices, where r is the unique contraction ratio associated with the IFS. This gives us a simple upper bound on how many times we can iterate the IFS and still distinguish vertices. There's still more to it, though. At each level of the iteration, we accumulate a bit of numerical error in the process. This can limit us even further. Perhaps we could call this parameter δ and fit that into our equation as well.

Is this all we need to account for? Examination of some examples might help; we would want to include a couple of examples anyway. Consider the following IFS, for example.

$$\begin{aligned} f_1(x) &= \frac{1}{3}x \\ f_2(x) &= \frac{1}{3}x + \frac{2}{3} \\ f_3(x) &= \frac{1}{3}x + \frac{1}{3^k} \end{aligned}$$

Note that f_3 depends on the integer parameter k . The larger k is, the closer f_3 is to f_1 , in some sense and I feel this would need to be taken into account as well.

Somewhere in this mess is an inequality or a small collection of inequalities involving the number of iterations performed by the algorithm that must be satisfied in order trust the result. Find it and illustrate with examples.