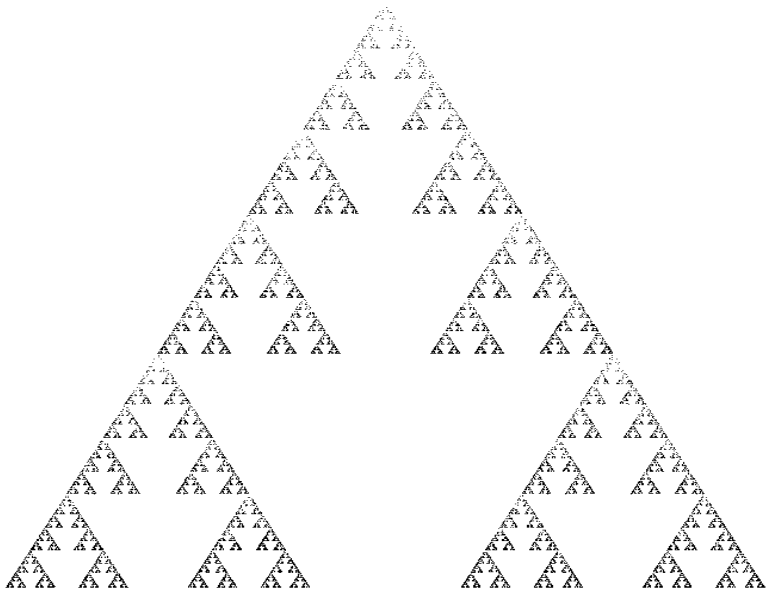
Lecture 33

Iterated function systems (IFS) and the construction of fractal sets (cont'd)

Modified Sierpinski gasket - "Cantor tree"

If we alter the contraction factors of the three affine maps so that the first and third copies touch the second copy, but do not touch each other, we produce the following fractal set. A horizontal line intersecting the set will generally intersect it over a Cantor-like set.



Modified Sierpinski gasket - "Cantor tree"

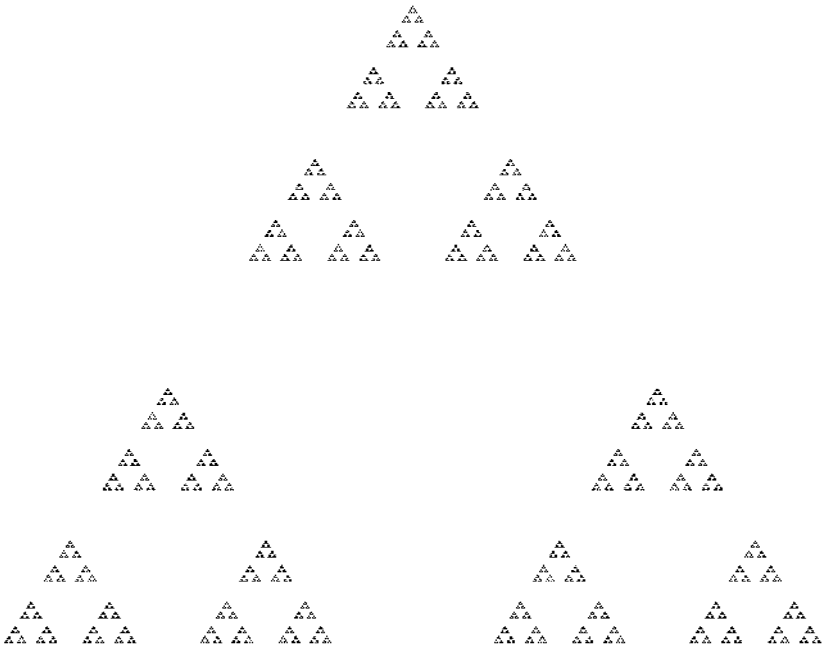
$$f_1(x,y) = \begin{pmatrix} \frac{2}{5} & 0 \\ 0 & \frac{2}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$f_2(x,y) = \begin{pmatrix} \frac{3}{5} & 0 \\ 0 & \frac{3}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{5} \\ \frac{\sqrt{3}}{5} \end{pmatrix}$$

$$f_3(x,y) = \begin{pmatrix} \frac{2}{5} & 0 \\ 0 & \frac{2}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{3}{5} \\ 0 \end{pmatrix}$$

Another modified Sierpinski gasket - "Cantor dust"

If the contraction factors of the three affine maps are adjusted so that there is no intersection of copies, then the result is a "3D Cantor dust".



Modified Sierpinski gasket - "3D Cantor dust"

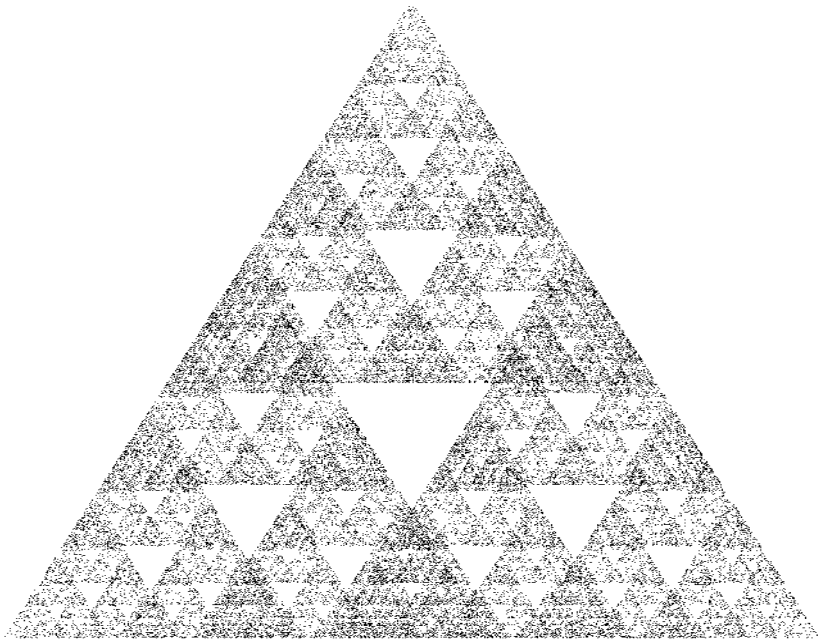
$$f_1(x,y) = \begin{pmatrix} \frac{2}{5} & 0 \\ 0 & \frac{2}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$f_2(x,y) = \begin{pmatrix} \frac{2}{5} & 0 \\ 0 & \frac{2}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{3}{10} \\ \frac{\sqrt{3}}{10} \end{pmatrix}$$

$$f_3(x,y) = \begin{pmatrix} \frac{2}{5} & 0 \\ 0 & \frac{2}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{3}{5} \\ 0 \end{pmatrix}$$

And yet another modified Sierpinski gasket

The contraction factors of the three maps have now been increased beyond $\frac{1}{2}$ so that there is overlapping between the copies $\hat{f}_i(S)$. This overlapping will occur in a self-similar manner throughout the set.



Modified Sierpinski gasket with overlap

$$f_1(x,y) = \begin{pmatrix} \frac{3}{5} & 0 \\ 0 & \frac{3}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

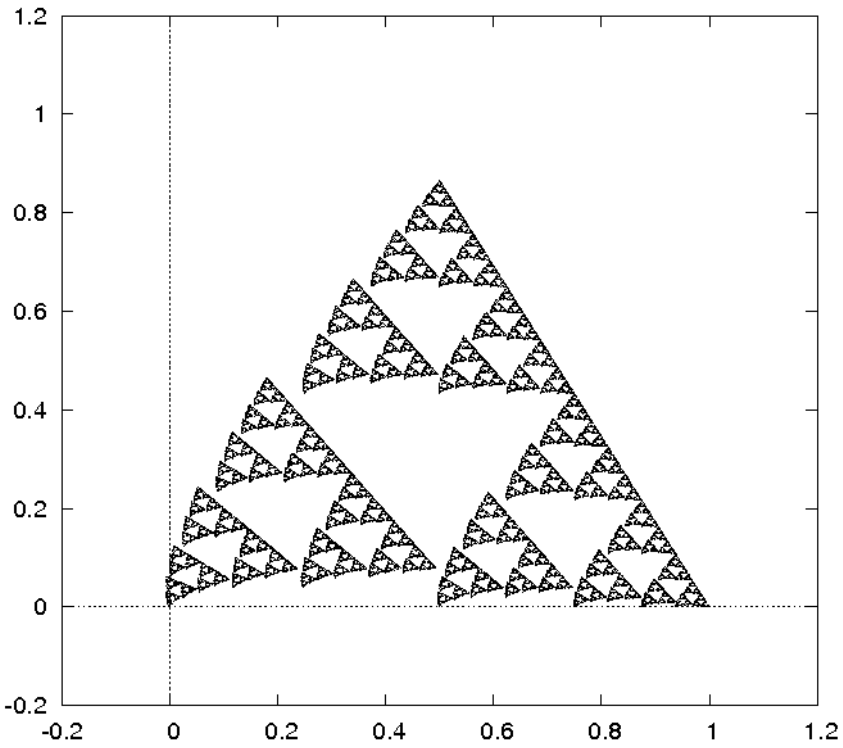
$$f_2(x,y) = \begin{pmatrix} \frac{3}{5} & 0 \\ 0 & \frac{3}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{5} \\ \frac{\sqrt{3}}{5} \end{pmatrix}$$

$$f_3(x,y) = \begin{pmatrix} \frac{3}{5} & 0 \\ 0 & \frac{3}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{2}{5} \\ 0 \end{pmatrix}$$

Modified Sierpinski gasket with rotations

In the following examples, the maps have the general form,

$$f_i(x,y) = r \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$



Modified Sierpinski gasket with one rotation map

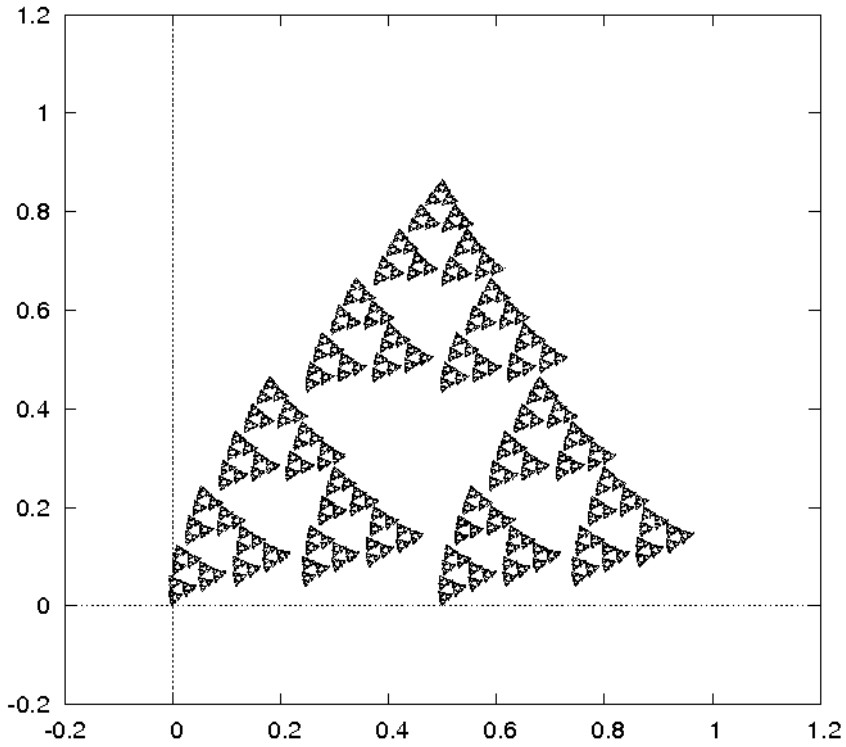
$$f_1(x,y): \quad r = 0.5\,, \quad \theta = \frac{\pi}{20}\,.$$

$$f_2(x,y): \quad r = 0.5\,, \quad \theta = 0\,.$$

$$f_3(x,y): \quad r = 0.5\,, \quad \theta = 0\,.$$

Modified Sierpinski gasket with rotations

$$f_i(x,y) = r \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$



Modified Sierpinski gasket with two rotation maps

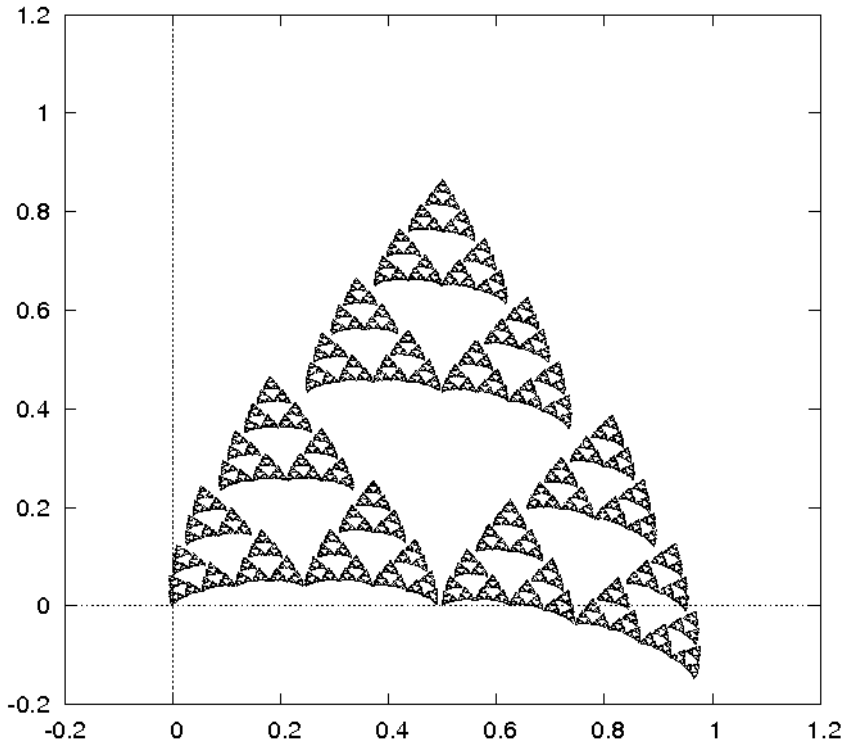
$$f_1(x,y): \quad r = 0.5\,, \quad \theta = \frac{\pi}{20}\,.$$

$$f_2(x,y): \quad r = 0.5\,, \quad \theta = 0\,.$$

$$f_3(x,y): \quad r = 0.5\,, \quad \theta = \frac{\pi}{20}\,.$$

Modified Sierpinski gasket with rotations

$$f_i(x,y) = r \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$



Modified Sierpinski gasket with two rotation maps

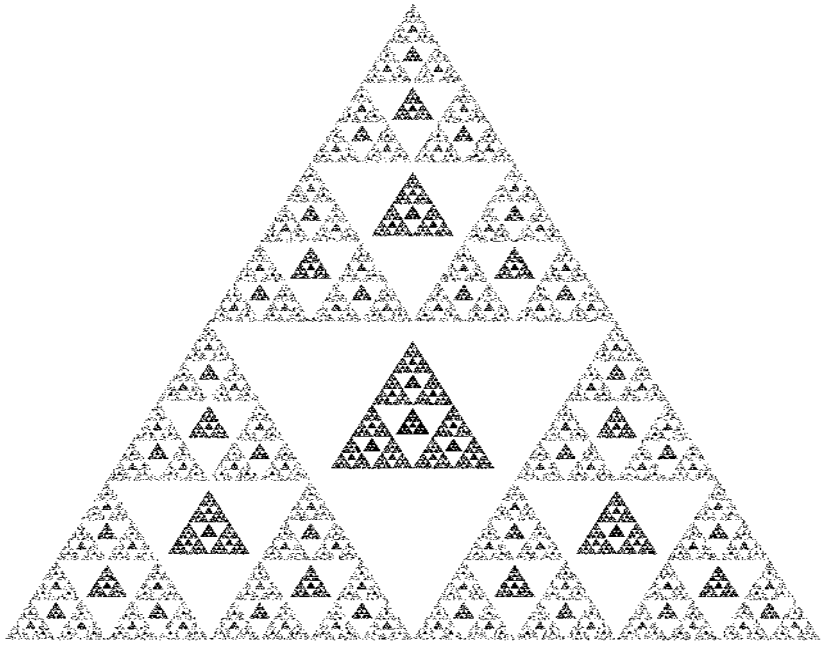
$$f_1(x,y): \quad r = 0.5\,, \quad \theta = \frac{\pi}{20}\,.$$

$$f_2(x,y): \quad r = 0.5\,, \quad \theta = 0\,.$$

$$f_3(x,y): \quad r = 0.5\,, \quad \theta = -\frac{\pi}{20}\,.$$

Sierpinski gasket with an additional map

We now add a fourth map to the Sierpinski gasket IFS. This fourth map is a contraction map with fixed point at the center of the gasket. The contraction factor is $\frac{1}{5}$ – small enough to map the entire triangle into the formerly empty space. Note that these copies are propagated to all "formerly empty" spots.



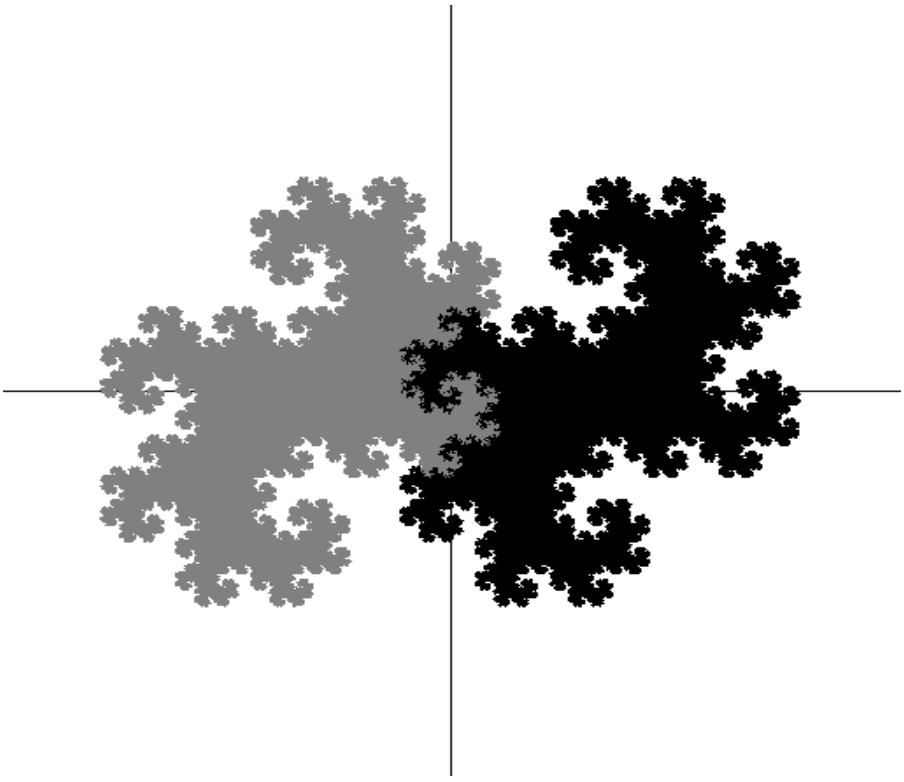
Sierpinski gasket with extra map in middle

$$f_1(x,y) = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad f_2(x,y) = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{\sqrt{3}}{4} \end{pmatrix}$$

$$f_3(x,y) = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} \qquad f_4(x,y) = \begin{pmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{5} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{2}{5} \\ \frac{\sqrt{3}}{4} - \frac{1}{5} \end{pmatrix}$$

"Twin dragon" attractor in $\mathbb{R}^2$

This is a well-known set that is the attractor of a two-map IFS in the plane. The fixed points of the two maps lie on the real line.



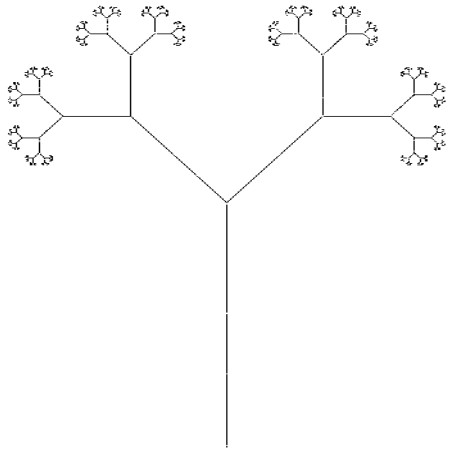
$$f_1(x,y) = \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad f_2(x,y) = \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Both affine maps involve (i) a contraction of $\dfrac{1}{\sqrt{2}}$ and (ii) a rotation by $\dfrac{\pi}{4}$ followed by a translation.

The twin-dragon attractor tiles the plane $\mathbb{R}^2$. There are no holes in this set. The two copies of this set $f_1(A)$ and $f_2(A)$ have been shaded differently so that they can be viewed more easily as contracted (by $1/\sqrt{2}$) and rotated (by $\pi/4$) copies of A.

Tree-like sets

The following 3-map IFS shows how tree-like objects can be generated. The map f_3 generates the main stem. The other two maps take the stem, translate it upwards and then rotate it in opposite directions. Of course, these maps operate not only on the stem but on the entire set. For this reason, the self-similar tree-like attractor set is produced.



Simple tree

$$f_1(x,y) = \begin{pmatrix} 0.353 & -0.353 \\ 0.353 & 0.353 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.5 \end{pmatrix}$$

Contraction factor $r = 0.5$, rotation $\pi/4$.

$$f_2(x,y) = \begin{pmatrix} 0.353 & 0.353 \\ -0.353 & 0.353 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.5 \end{pmatrix}$$

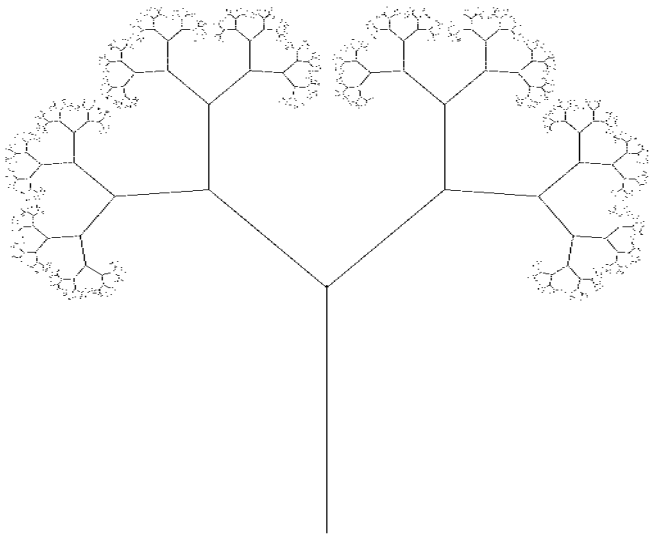
Contraction factor $r = 0.5$, rotation $-\pi/4$.

$$f_3(x,y) = \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.55 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.5 \end{pmatrix}$$

"Terminator": Squash onto y-axis.

Tree-like sets (cont'd)

A slight modification of the previous IFS – the maps f_1 and f_2 no longer have pure rotations, but shear transformations.



Less simple tree

$$f_1(x,y) = \begin{pmatrix} 0.4 & -0.433 \\ 0.433 & 0.4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.5 \end{pmatrix}$$

Shear transformation

$$f_2(x,y) = \begin{pmatrix} 0.4 & 0.433 \\ -0.433 & 0.4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.5 \end{pmatrix}$$

Shear transformation

$$f_3(x,y) = \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.55 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.47 \end{pmatrix}$$

"Terminator": Squash onto y-axis.

The epitome of fractal attractors – at least in 1986

This is Prof. Michael Barnsley's celebrated "spleenwort fern," the attractor of a four-map IFS.

Barnsley's Spleenwort Fern

$$f_1(x,y) = \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.16 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.50 \\ 0.0 \end{pmatrix}$$

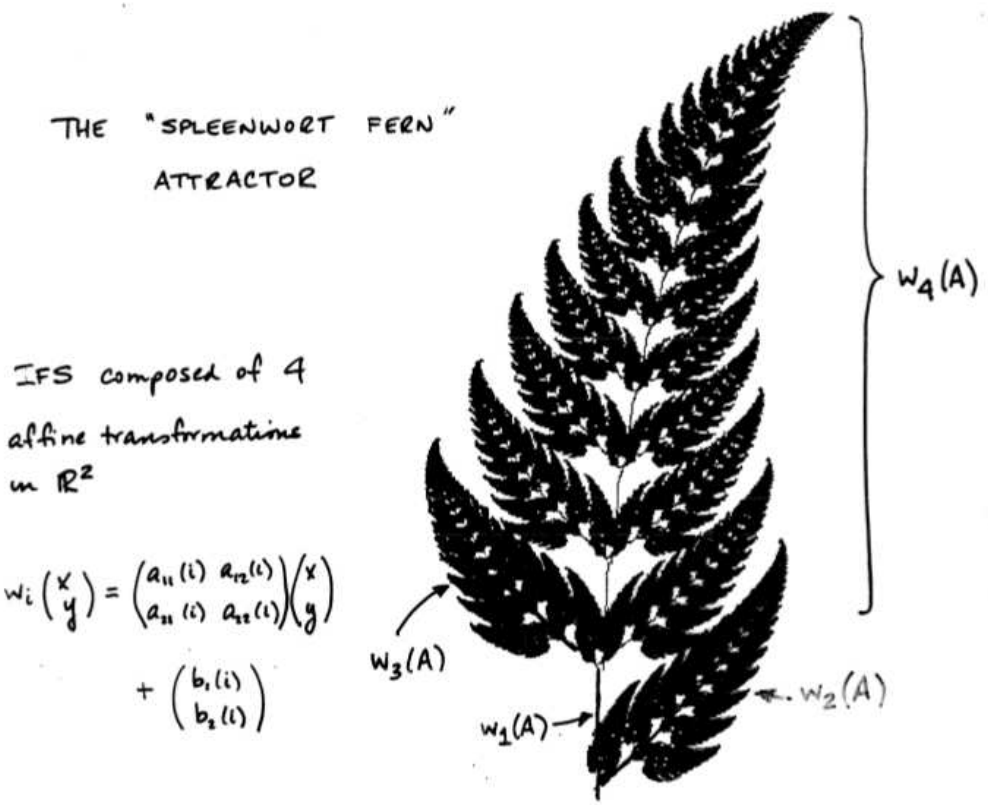
"Terminator": Squash onto y-axis to make stem.

$$f_2(x,y) = \begin{pmatrix} 0.20 & -0.26 \\ 0.23 & 0.22 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.40 \\ 0.05 \end{pmatrix}$$

$$f_3(x,y) = \begin{pmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.57 \\ -0.12 \end{pmatrix}$$

$$f_4(x,y) = \begin{pmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.08 \\ 0.18 \end{pmatrix}$$

In the figure below, the action of the IFS maps, denoted as w_i, is shown, in order to give the reader an idea of how the smaller and smaller copies are produced in a geometric cascade as we move upwards toward the peak of the fern. This cascade is produced by map w_4. Map w_1, the "terminator", produces the stem. Map w_2 produces the lower right copy of the fern and map w_3 flips the fern and produces the lower left copy. Some playing around of the map parameters is necessary in order to ensure that the bottoms of the stems of the two copies produced by w_2 and w_3 touch the main stem – otherwise, there would be gaps that propagate throughout the fern. For the same reason, it is important that the upward translation produced by w_4 also touch the main stem.



i	$a_{11}(i)$	$a_{12}(i)$	$a_{21}(i)$	$a_{22}(i)$	$b_1(i)$	$b_2(i)$
1	0	0	0	0.16	0.50	0
2	0.20	-0.26	0.23	0.22	0.40	0.05
3	-0.15	0.28	0.26	0.24	0.57	-0.12
4	0.85	0.04	-0.04	0.85	0.08	0.18

Using IFS to model the real world?

Professor Barnsley's beautiful discovery gave rise to the question – and the tremendous amount of research it inspired – of how well IFS could be used to generate sets that approximated real-world objects. This eventually led to the idea of **fractal image coding** – a way of representing photos and videos as attractors of special types of IFS. Early in this research (mid 1980's and early 1990's), fractal image coding was shown to be an effective method of **image compression**, i.e., reducing the amount of computer memory required to store a representation of an image, from which it could be regenerated. (JPEG compression has, and continues to be, a standard method of image compression, although much more powerful methods of compression exist today. JPEG compression, by the way, is based on the method of Fourier series representation of functions. It is actually a version of the **discrete cosine transform** for discrete – in this case, digital – data sets.) We'll return to these subjects shortly.

Methods to generate fractal attractor sets of IFS

We have not yet addressed one important matter regarding IFS and their attractor sets: Once we have a set of N contraction maps $\mathbf{f} = \{f_1, f_2, \cdots, f_n\}$, how do we produce a picture of the attractor A of the parallel IFS operator $\hat{\mathbf{f}}$ which they define, i.e., the set which possesses the following self-similarity property,

$$A = \bigcup_{i=1}^{N} \hat{f}_i(A)\,? \tag{1}$$

In what follows, we describe briefly two principal methods that can be used to construct the fractal attractor set A of an N-map IFS in $\mathbb{R}^n$.

Random iteration algorithm: In the research literature, this method is known as the "Chaos Game." (See the book, *Fractals Everywhere*, by M.F. Barnsley.) This method, the main ideas of which were described in a question in Problem Set No. 4, was used to generate the pictures of IFS attractors presented earlier. We start with a seed point $x_0 \in \mathbb{R}^n$ and perform the following random iteration procedure:

$$x_{n+1} = f_{\sigma_n}(x_n)\,, \quad n \geq 0\,. \tag{2}$$

At each step n, the index σ_n is chosen randomly and independently from the set of indices $\{1, 2, \cdots, N\}$. If the seed point x_0 happens to lie in A, then all future x_n will also be in A. However, just to be safe, one should wait for a sufficient number of iterations, say 10 or, better yet, 50, before plotting the points x_n. If $x_0 \in A$, then, because of the contractivity of the f_i maps, future iterates x_n will be attracted to A.

Because the maps f_i are chosen at random, the iterates x_n will be travelling or (or at least near) the attractor A in a random way. It is necessary to plot a sufficiently large number of x_n so that all

regions of the attractor A are visited.

There still remains the question of what **probabilities** to employ in the selection of the maps f_i. For many of the "standard" IFS attractors, e.g., von Koch curve, Sierpinski gasket, good results will be obtained if the maps are chosen with equal probability, i.e., the probability p_i of choosing index $i \in \{1, 2, \cdots, N\}$ is $p_i = \frac{1}{N}$, $1 \leq i \leq N$. For more complicated sets such as the Spleenwort fern, however, it is necessary to employ nonuniform probabilities. To see this, note that the "terminator" map f_1 (or w_1) for the Spleenwort fern maps the entire attractor A onto the tiny stem situated at the base of the fern. On the other hand, map f_4 (or w_4) is responsible for generating about 90% of the fern in terms of the geometric cascade. As such, it is important that a much higher probability be assigned to map w_4. A rough rule of thumb is that the probability p_i of choosing map f_i should be in some way related to the ratio of the area of the copy $f_i(A)$ to the area of A.

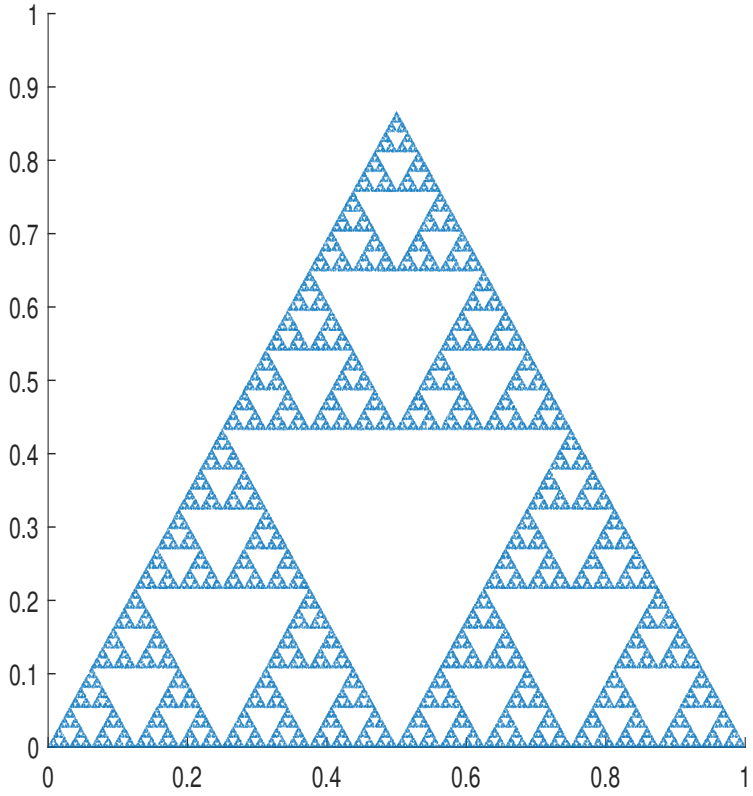
The idea of probabilities p_i being associated with the IFS maps leads to another important concept, that of **invariant measures** that "live" on their attractors A. Unfortunately, there is no time to discuss this topic in this course.

On the next page is presented a very simple MATLAB program, "**ifs2d.m**", to plot the attractor of an N-map IFS in $\mathbb{R}^2$. The program reads the IFS parameters, i.e., the elements of the matrix $\mathbf{A}$ and vector $\mathbf{b}$ defining each affine IFS map, from a file – in this case, the file is called "**ifs2d.dat**". A sample data file is given below the program. Its three lines contain the components of the three maps which define the IFS with Sierpinski gasket as attractor, given earlier. The output generated from this data is shown on the next page.

The files **ifs2d.m** and **ifs2d.dat** have been posted at the course site on LEARN.

```
%  ifs2d.m
%  routine to produce plots of attractors sets of 2D
%  iterated function systems with probabilities
%  using the random iteration algorithm, or "Chaos Game"

%  nmaps IFS maps w(i) = A(i)*x + b(i), i=1..nmaps, with probabilities p(i)

%  the attractor A of an IFS is presented as a scatter plot
%  in the region [xmin,xmax] X [ymin,ymax]
%  (the user may have to adjust these values accordingly)

nmaps=3;
niter=100000;
xmin=0.0;
xmax=1.0;
ymin=0.0;
ymax=1.0;
kndx=zeros(1,1,'uint8');

% these vectors will store the x and y coordinates of the points
% generated on the attractor

xx=zeros(niter,1);
yy=zeros(niter,1);

% these vectors store the IFS parameters

a11=zeros(nmaps,1);
a12=zeros(nmaps,1);
a21=zeros(nmaps,1);
a22=zeros(nmaps,1);
b1=zeros(nmaps,1);
b2=zeros(nmaps,1);
p=zeros(nmaps,1);

%  input IFS parameters from a data file

[a11,a12,a21,a22,b1,b2,p] = textread('ifs2d.dat','%f %f %f %f %f %f %f',nmaps);

%  start with (0,0) and iterate first IFS map until points get close to its
%  fixed point (which is a point on the attractor)

x=0.0;
y=0.0;
for i=1:50
    x1=a11(1)*x + a12(1)*y + b1(1);
    y1=a21(1)*x + a22(1)*y + b2(1);
```

```
    x=x1;
    y=y1;
end

% now continue random iteration of IFS over points on attractor
% here, the IFS maps are chosen with equal probabilities
% this program can be modified so that the ith IFS map is chosen
% with probability p(i)

for i=1:niter

    xran=rand;
    kndx=round(nmaps*xran+0.5);
    kk(i)=kndx;
    x1=a11(kndx)*x + a12(kndx)*y + b1(kndx);
    y1=a21(kndx)*x + a22(kndx)*y + b2(kndx);
    x=x1;
    y=y1;
    xx(i)=x;
    yy(i)=y;

end

% plot the attractor

figure(1)
scatter(xx,yy,0.01),xlim([xmin xmax]),ylim([ymin  ymax]);



SAMPLE FILE  "ifs2d.dat"   (Sierpinski gasket)


  0.5  0.0  0.0  0.5  0.0   0.0    0.3333
  0.5  0.0  0.0  0.5  0.25  0.433  0.3333
  0.5  0.0  0.0  0.5  0.5   0.0    0.3333
```



Output from MATLAB program **ifs2d.m** using data from **ifs2d.dat** from previous page: The Sierpinski gasket attractor.

Deterministic algorithm: Recall how the IFS "parallel operator" $\hat{\mathbf{f}}$ associated with a set of N contraction maps f_i acts on sets to produce sets: For a set $S \in D$ (where $D \subset \mathbb{R}^k$ is our region of interest),

$$\hat{\mathbf{f}}(S) = \bigcup_{i=1}^{N} \hat{f}_i(S)\,. \tag{3}$$

In other words, $\hat{\mathbf{f}}$ produces N contracted copies of S. Also recall that for any $S \in D$, the iterates of S under the action of $\hat{\mathbf{f}}$ converge to the attractor/fixed point A of the IFS:

$$\lim_{n\to\infty} \hat{\mathbf{f}}^n(S) = A = \hat{\mathbf{f}}(A) = \bigcup_{i=1}^{N} \hat{f}_i(A)\,. \tag{4}$$

We can start the deterministic algorithm with a seed point $x_0 \in \mathbb{R}^n$, which defines our set S_0. The action of the IFS parallel operator on S_0 is to produce a set S_1 consisting of N (or possibly less) points, i.e.,

$$S_1 = \hat{\mathbf{f}}(S_0) = \hat{\mathbf{f}}(x_0) = \bigcup_{i=1}^{N} f_i(x_0)\,. \tag{5}$$

Applying the IFS parallel operator $\hat{\mathbf{f}}$ on S_1 will produce a set S_2 consisting of N^2 (or possibly less) points, i.e.,

$$\begin{aligned} S_2 = \hat{\mathbf{f}}(S_1) &= \bigcup_{j=1}^{N} f_j \left(\bigcup_{i=1}^{N} f_i(x_0) \right) \\ &= \bigcup_{j=1}^{N} \bigcup_{i=1}^{N} (f_j \circ f_i)(x_0)\,. \end{aligned} \tag{6}$$

The reader should see the pattern. N applications of the IFS parallel operator $\hat{\mathbf{f}}$ will produce a set S_n consisting of (at most) N^n points,

$$S_n = \{ (f_{i_1} \circ f_{i_2} \circ \cdots \circ f_{i_n})(x_0)\,,\ i_1, i_2, \cdots, i_n \in \{1, 2, \cdots, N\} \}\,. \tag{7}$$

If the point $x_0 \in A$, the attractor of the IFS, then all points in S_1, S_2, etc., will lie in A. So if you happen to know a point in A, it's good to use it. But even if x_0 is not in A, the contractivity of the f_i will bring the points in S_n closer and closer to A as n increases. What is more important, however, is to generate a sufficient number of points to ensure that most of the set A is visited by them.

The above algorithm corresponds to taking all possible N^n paths from x_0 down an N-tree to the N^n points which comprise the set S_n. This can be done by means of a recursive calling of a subroutine, provided that the computer language you are using supports recursion.

Lecture 34

IFS and the construction of fractal sets (cont'd)

Using IFS attractors to approximate sets, including natural objects

We return to the idea of using IFS attractors to approximate sets, in particular, sets that look like natural objects. As motivation, we revisit Prof. Michael Barnsley's "spleenwort fern" attractor, shown in the previous lecture and presented again below.

Spleenwort Fern – the attractor of a four-map IFS in $\mathbb{R}^2$.

As mentioned later in that lecture, with the creation of these fern-type attractors in 1984 came the idea of using IFS to approximate other shapes and figures occuring in nature and, ultimately, images in general. The IFS was seen to be a possible method of **data compression**. A high-resolution picture of a shaded fern normally requires on the order of one megabyte of computer memory for storage. Current compression methods might be able to cut this number by a factor of ten or so. However, as an attractor of a four map IFS with probabilities, this fern may be described totally in terms of only 28 IFS parameters! This is a staggering amount of data compression. Not only are the storage requirements reduced but you can also send this small amount of data quickly over communications lines to others who could then "decompress" it and reconstruct the fern by simply iterating the IFS "parallel" operator $\hat{\mathbf{f}}$.

However, not all objects in nature – in fact, very few – exhibit the special self-similarity of the spleenwort fern. Nevertheless, as a starting point there remains the interesting general problem to determine to determine how well sets and images can be approximated by the attractors of IFS. We pose the so-called **inverse problem** for geometric approximation with IFS as follows:

> Given a "target" set S, can one find an IFS $\mathbf{f} = \{f_1, f_2, \cdots, f_N\}$ whose attractor A approximates S to some desired degree of accuracy in an appropriate metric "D" which measures distances between sets?

At first, this appears to be a rather formidable problem. How does one start? By selecting an initial set of maps $\{f_1, f_2, \cdots, f_N\}$, iterating the associated parallel operator $\hat{\mathbf{f}}$ to produce its attractor A and then comparing it to the target set S? And then perhaps altering some or all of the maps in some ways, looking at the effects of the changes on the resulting attractors, hopefully zeroing in on some final IFS?

If we step back a little, we can come up with a strategy. In fact, it won't appear that strange after we outline it, since you are already accustomed to looking at the self-similarity of IFS attractors, e.g., the Sierpinski triangle in this way. Here is the strategy.

Given a target set S, we are looking for the attractor A of an N-map IFS $\mathbf{f}$ which approximates it well, i.e.,

$$S \approx A\,. \tag{8}$$

By "$\approx$", we mean that the S and A are "close" – for the moment "visually close" will be sufficient. Now recall that A is the attractor of the IFS $\mathbf{f}$ so that

$$A = \bigcup_{k=1}^{N} \hat{f}_k(A)\,. \tag{9}$$

Substitution into Eq. (8) yields

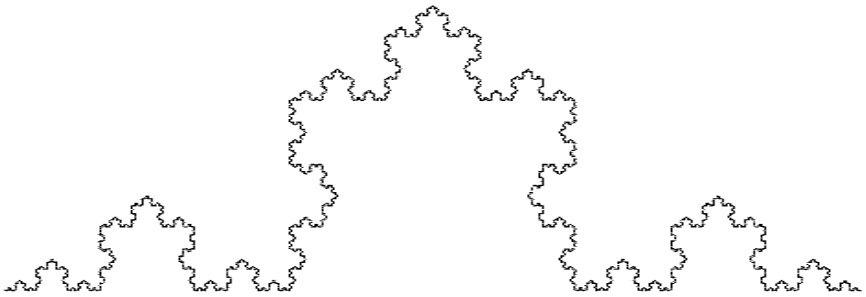
$$S \approx \bigcup_{k=1}^{N} \hat{f}_k(A)\,. \tag{10}$$

But we now use Eq. (8) to replace A on the RHS and arrive at the final result,

$$S \approx \bigcup_{k=1}^{N} \hat{f}_k(S)\,. \tag{11}$$

In other words, in order to find an IFS with attractor A which approximates S, **we look for an IFS, i.e., a set of maps $\mathbf{f} = \{f_1, f_2, \cdots f_n\}$, which, under the parallel action of the IFS operator $\hat{\mathbf{f}}$, map the target set S as close as possible to itself. In this way, we are expressing the target set S as closely as possible as a union of contracted copies of itself.**

This idea should not seem that strange. After all, if the set S is self-similar, e.g., the attractor of an IFS, then the approximation in Eq. (11) becomes an equality. **In fact, we were already using this idea when trying to find IFS associated with self-similar fractal sets such as the Cantor set, the von Koch curve and the Sierpinski triangle.** In those cases, the sets we were analyzing were perfect fractals for which the approximation in Eq. (11) became an equality since they corresponded exactly to attractors of IFS. For example, recall the von Koch curve pictured below.



von Koch curve C

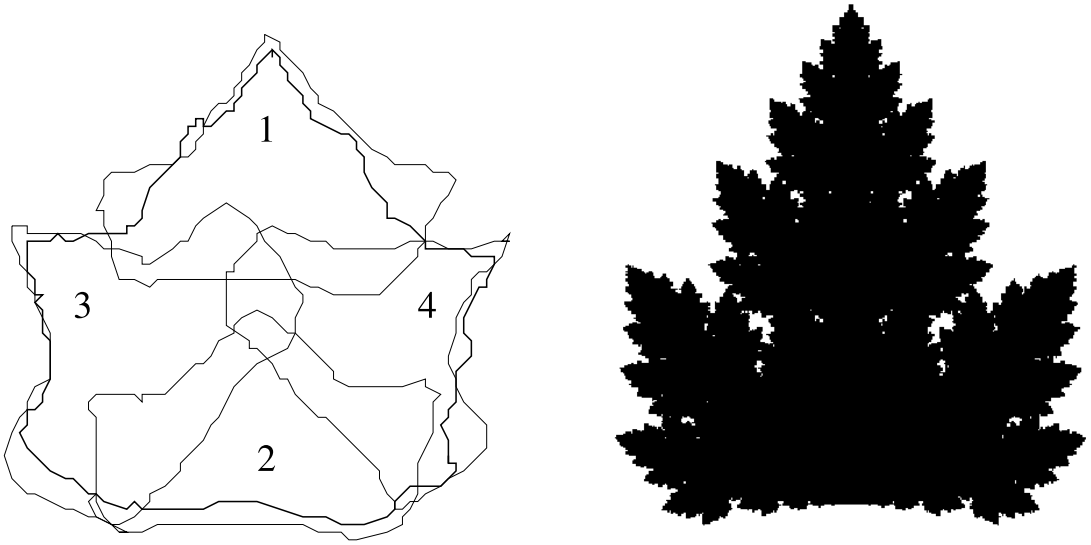
Recall that we viewed the von Koch curve C as a union – and not an approximation – of four contracted copies of itself, i.e.,

$$C = \bigcup_{k=1}^{4} \hat{f}_k(C)\,. \tag{12}$$

Not only that, but we were able to determine, with little difficulty, the affine transformations f_k that mapped the entire curve C into each of the four copies.

For more realistic and perhaps "less fractal" sets, the basic idea is illustrated in the figure below. At the left, a leaf – enclosed with a solid curve – is viewed as an approximate union of four contracted copies of itself. Each smaller copy is obtained by an appropriate contractive IFS map f_i. If we restrict

ourselves to affine IFS maps in the plane, i.e. $f_i(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, then the coefficients of each matrix $\mathbf{A}$ and associated column vector $\mathbf{b}$ – a total of six unknown coefficients – can be obtained from a knowledge of where three points of the original leaf S are mapped in the contracted copy $\hat{f}_i(S)$. We then expect that the attractor A of the resulting IFS $\hat{\mathbf{f}}$ lies close to the target leaf S. The attractor A of the IFS is shown on the right.



Left: Approximating a leaf as a "collage", i.e. a union of contracted copies of itself. **Right:** The attractor A of the four-map IFS obtained from the "collage" procedure on the left.

This procedure was first reported in a 1985 paper by Prof. Michael Barnsley and students from the School of Mathematics, Georgia Institute of Technology entitled, "Solution of an inverse problem for fractals and other sets," published in 1986 in the Proceedings of the National Academy of Sciences of the USA, Vol. 83, pp. 1975-1977 (April 1986). A copy of this paper is attached at the end of this day's lecture notes.

Once again, we actually employed this procedure when we looked at perfect fractal sets such as the von Koch curve and the Sierpinski triangle a lecture or so ago. In these cases, it was very easy to see the self-similarity of the set, i.e., how it could be expressed as a union of N contracted copies of itself. And the affine mappings $\hat{f}_k$ that mapped the set S into the contracted copies $\hat{f}_k(S)$ were relatively easy to determine.

In general, however, the determination of optimal IFS maps by looking for approximate geometric self-similarities in a set is a very difficult problem with no simple solutions, **especially if one wishes**

to automate the process. Fortunately, we can proceed by another route by realizing that there is much more to a picture than just geometric shapes. There is also **shading** in an image. For example, a real fern has veins which may be darker than the outer extremeties of the fronds. Thus it is more natural to think of a picture as defining a function: At each point or pixel (x, y) in a photograph or a computer display (represented, for convenience, by the region $X = [0, 1]^2$) there is an associated grey level or **greyscale value** $u(x, y)$, which may assume a finite nonnegative value. (In practical applications, i.e. digitized images, each pixel can assume one of only a finite number of discrete values.) This leads to the consideration of an IFS-type method which operates on **image functions**: Given an image function $u(x, y)$ that we wish to approximate by the attractor of an IFS-type method, find a set of maps – involving both the spatial coordinates (x, y) as well as the greyscale values $y = u(x, y)$ – which approximates the function $u(x, y)$ as a union of geometrically-contracted and greyscale-modified copies of itself. This is the idea of **fractal image coding**, which will be discussed in the next section.

For a "gentle" introduction to fractal image coding, the reader is invited to consult the instructor's article, *A Hitchhiker's Guide to "Fractal-Based" Function Approximation and Image Compression*, a slightly expanded version of two articles which appeared in the February and August 1995 issues of the UW Faculty of Mathematics Alumni newpaper, *Math Ties*. It may be downloaded from the instructor's webpage.

Iterated function systems for functions: "Fractal transforms" and "fractal image coding"

Note: The following section is taken from ERV's article, *A Hitchhiker's Guide to 'Fractal-Based' Function Approximation and Image Compression.* This material was presented in class, and in these notes, for information only.

As mentioned near the end of the previous lecture, an image or picture is much more than a set of geometric shapes. as being more than merely geometric shapes. There is also shading. As such, it is more natural to think of a picture as defining a function: At each point or pixel (x, y) in a photograph – assumed to be black-and-white for the moment – there is an associated "grey level" $u(x, y)$ which assumes a finite and nonnegative value. (Here, $(x, y) \in X = [0, 1]^2$, for convenience.) For example, consider Figure 1 below, a standard test case in image processing studies named "Boat". The image is a 512×512 pixel array. Each pixel assumes one of 256 shades of grey (0 = white, 255 = black). From the point of view of continuous real variables (x, y), the image is represented as a piecewise constant function $u(x, y)$. If the grey level value of each pixel is interpreted as a value in the z direction, then the graph of the *image function* $z = u(x, y)$ is a surface in $\mathbb{R}^3$, as shown on the right. The red-blue spectrum of colours in the plot is used to characterize function values: Higher values are more red, lower values are more blue.

Our goal is to set up an IFS-type approach to work with non-negative functions $u : X \to \mathbb{R}^+$ instead of sets. Before writing any mathematics, let us illustrate schematically what can be done. For ease of presentation, we consider for the moment only one-dimensional images, i.e. positive real-valued functions $u(x)$ where $x \in [0, 1]$. An example is sketched in Figure 2(a). Suppose our IFS is composed of only two contractive maps f_1, f_2. Each of these functions f_i will map the "base space" $X = [0, 1]$ to a subinterval $\hat{f}_i(X)$ contained in X. Let's choose

$$f_1(x) = 0.6x, \qquad f_2(x) = 0.6x + 0.4. \tag{13}$$

For reasons which will become clear below, it is important that $\hat{f}_1(X)$ and $\hat{f}_2(X)$ are *not* disjoint - they will have to overlap with each other, even if the overlap occurs only at one point.

The first step in our IFS procedure is to make two copies of the graph of $u(x)$ which are distorted to fit on the subsets $\hat{f}_1(X) = [0, 0.6]$ and $\hat{f}_2(X) = [0.4, 1]$ by "shrinking" and translating the graph in

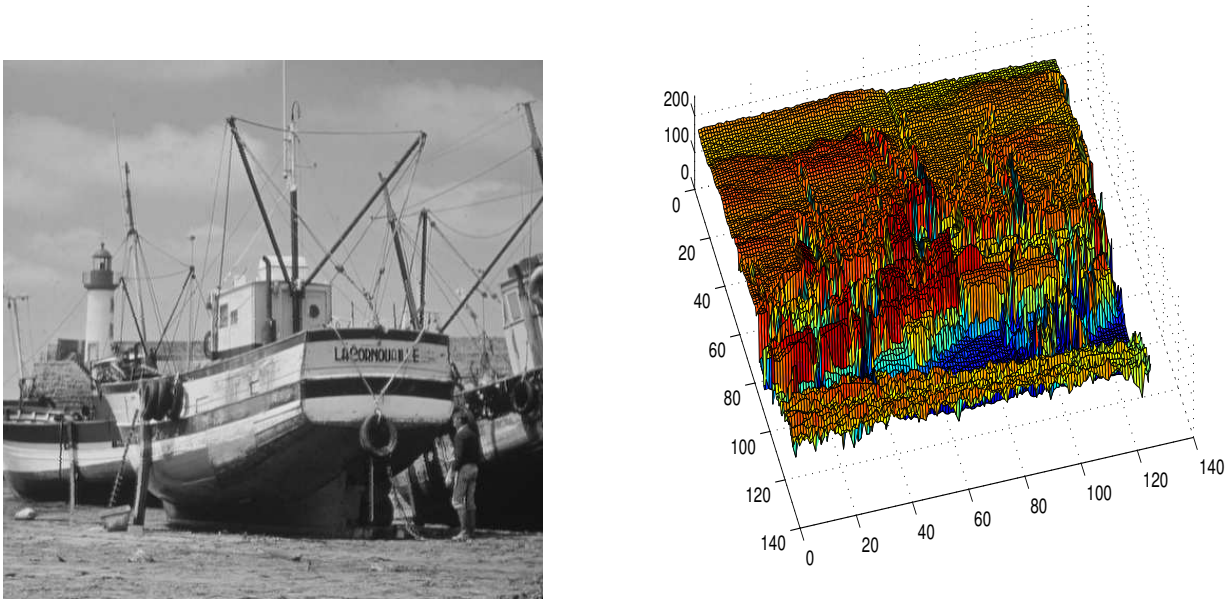


Figure 1. Left: The standard test-image, *Boat*, a 512×512-pixel digital image, 8 bits per pixel. **Right:** The *Boat* image, viewed as a non-negative image function $z = u(x, y)$.

the x-direction. This is illustrated in Figure 2(b). Mathematically, the two "component" curves $a_1(x)$ and $a_2(x)$ in Figure 2(b) are given by

$$a_1(x) = u(f_1^{-1}(x)) \quad x \in \hat{f}_1(X), \qquad a_2(x) = u(f_2^{-1}(x)) \quad x \in \hat{f}_2(X), \tag{14}$$

It is important to understand this equation. For example, the term $f_1^{-1}(x)$ is defined only for those $x \in X$ at which the inverse of f_1 exists. For the inverse of f_1 to exist at x means that one must be able to get to x under the action of the map f_1, i.e., there exists a $y \in X$ such that $f_1(y) = x$. But this means that $y = f_1^{-1}(x)$. It also means that $x \in \hat{f}_1(X)$, where

$$\hat{f}_1(X) = \{f_1(y)\,,\ y \in X\}\,. \tag{15}$$

Furthermore, note that since the map $f_1(x)$ is a contraction map, it follows that the function $u_1(x)$ is a **contracted** copy of $u(x)$ which is situated on the set $\hat{f}_1(X)$. All of the above discussion also applies to the map $f_2(x)$.

We're not finished, however, since some additional flexibility in modifying these curves would be desirable. Suppose that are allowed to modify the y (or grey level) values of each component function $a_i(x)$. For example, let us

1. multiply all values $a_1(x)$ by 0.5 and add 0.5,

2. multiply all values $a_2(x)$ by 0.75.

The modified component functions, denoted as $b_1(x)$ and $b_2(x)$, respectively, are shown in Figure 2(c). What we have just done can be written as

$$\begin{aligned} b_1(x) &= \phi_1(a_1(x)) = \phi_1(u(f_1^{-1}(x))) \quad x \in \hat{f}_1(X), \\ b_2(x) &= \phi_2(a_2(x)) = \phi_2(u(f_2^{-1}(x))) \quad x \in \hat{f}_2(X), \end{aligned} \tag{16}$$

where

$$\phi_1(y) = 0.5y + 0.5, \qquad \phi_2(y) = 0.75y, \quad y \in \mathbb{R}^+. \tag{17}$$

The ϕ_i are known as *grey-level maps*: They map (nonnegative) grey-level values to grey-level values.

We now use the component functions b_i in Figure 2(c) to construct a new function $v(x)$. How do we do this? Well, there is no problem to define $v(x)$ at values of $x \in [0, 1]$ which lie in only one of the two subsets $\hat{f}_i(X)$. For example, $x_1 = 0.25$ lies only in $\hat{f}_1(X)$. As such, we define $v(x_1) = b_1(x) = \phi_1(u(f_1^{-1}(x)))$. The same is true for $x_2 = 0.75$, which lies only in $\hat{f}_2(X)$. We define $v(x_2) = b_2(x) = \phi_2(u(f_2^{-1}(x)))$.

Now what about points that lie in *both* $\hat{f}_1(X)$ and $\hat{f}_2(X)$, for example $x_3 = 0.5$? There are two possible components that we may use to define our resulting function $v(x_3)$, namely $b_1(x_3)$ and $b_2(x_3)$. How do we suitably choose or combine these values to produce a resulting function $v(x)$ for x in this region of overlap?

To make a long story short, this is a rather complicated mathematical issue and was a subject of research, in particular at Waterloo. There are many possibilities of combining these values, including (1) adding them, (2) taking the maximum or (3) taking some weighted sum, for example, the average. In what follows, we consider the first case, i.e. *we simply add the values*. The resulting function $v(x)$ is sketched in Figure 3(a). The observant reader may now be able to guess why we demanded that the subsets $\hat{f}_1([0, 1])$ and $\hat{f}_2([0, 1])$ overlap, touching at least at one point. If they didn't, then the union

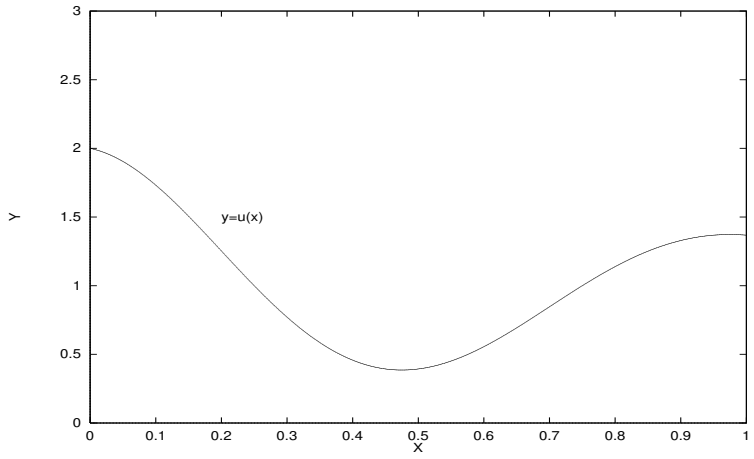


Figure 2(a): A sample "one-dimensional image" $u(x)$ on [0,1].

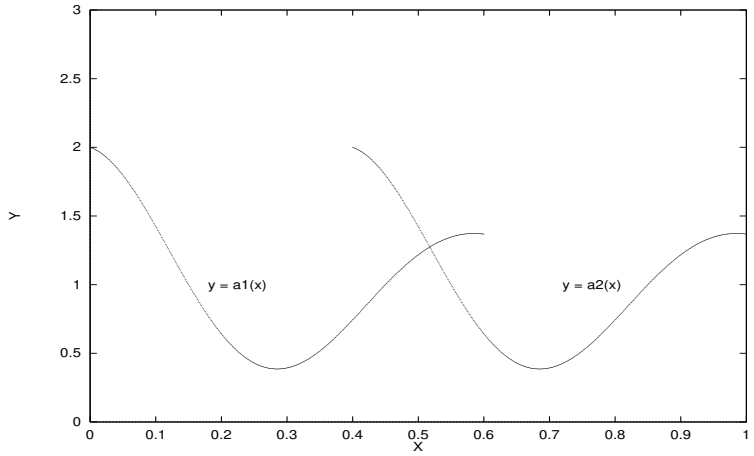


Figure 2(b): The component functions given in Eq. (14).

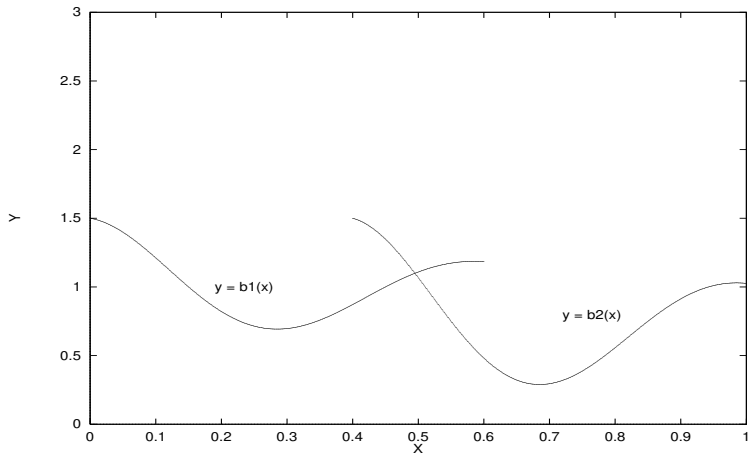


Figure 2(c): The modified component functions given in Eq. (16).

$\hat{f}_1(X) \cup \hat{f}_2(X)$ would have "holes", i.e. points $x \in [0,1]$ at which no component functions $a_i(x)$, hence $b_i(x)$, would be defined. (Remember the Cantor set?) Since want our IFS procedure to map functions on X to functions on X, the resulting function $v(x)$ *must* be defined for all $x \in X$.

The 2-map IFS $\mathbf{f} = \{f_1, f_2\}$, $f_i : X \to X$, along with associated grey-level maps $\Phi = \{\phi_1, \phi_2\}$, $\phi_i : \mathbb{R}^+ \to \mathbb{R}^+$, is referred to as an **Iterated Function System with Grey-Level Maps** (IFSM), $(\mathbf{f}, \Phi)$. What we did above was to associate with this IFSM an operator T which acts on a function u (Figure 2(a)) to produce a new function $v = Tu$ (Figure 3(a)). Mathematically, the action of this operator may be written as follows: For any $x \in X$,

$$v(x) = (Tu)(x) = \sum_{i=1}^{N}{}' \phi_i(u(f_i^{-1}(x))). \tag{18}$$

The prime on the summation signifies that for each $x \in X$ we sum over only those $i \in \{1, 2\}$ for which a "preimage" $f_i^{-1}(x)$ exists. (Because of the "no holes" condition, it guaranteed that for each $x \in X$, there exists at least one such i value.) For $x \in [0, 0.4)$, i can be only 1. Likewise, for $x \in (0.6, 1]$, $i = 2$. For $x \in [0.4, 0.6]$, i can assume both values 1 and 2. The extension to a general N-map IFSM is straightforward.

There is nothing preventing us from applying the T operator to the function v, so let $w = Tv = T(Tu)$. Again, we take the graph of v and "shrink" it to form two copies, etc.. The result is shown in Figure 3(b). As T is applied repeatedly, we produce a sequence of functions which converges to a function $\bar{u}$ in an appropriate metric space of functions, which we shall simply denote as $\mathcal{F}(X)$. In most applications, one employs the function space $L^2(X)$, the space of real-valued square-integrable functions on X, i.e.,

$$L^2(X) = \left\{ f : X \to \mathbb{R}\,,\ \| f \|_2 \equiv \left[\int_X |f(x)|^2 dx \right]^{1/2} < \infty \right\}. \tag{19}$$

In this space, the distance between two functions $u, v \in L^2(X)$ is given by

$$d_2(u, v) = \| u - v \|_2 = \left[\int_X |u(x) - v(x)|^2 \, dx \right]^{1/2}. \tag{20}$$

The function $\bar{u}$ is sketched in Figure 3(c). (Because it has so many jumps, it is better viewed as a histogram plot.)

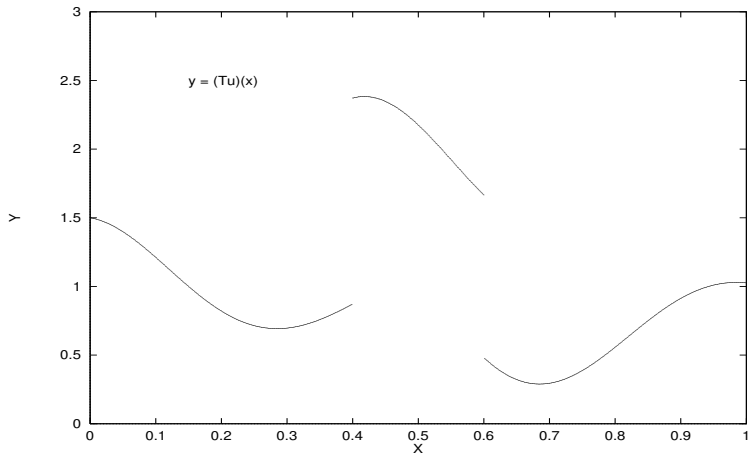


Figure 3(a): The resulting "fractal transform" function $v(x) = (Tu)(x)$ obtained from the component functions of Figure 2(c).

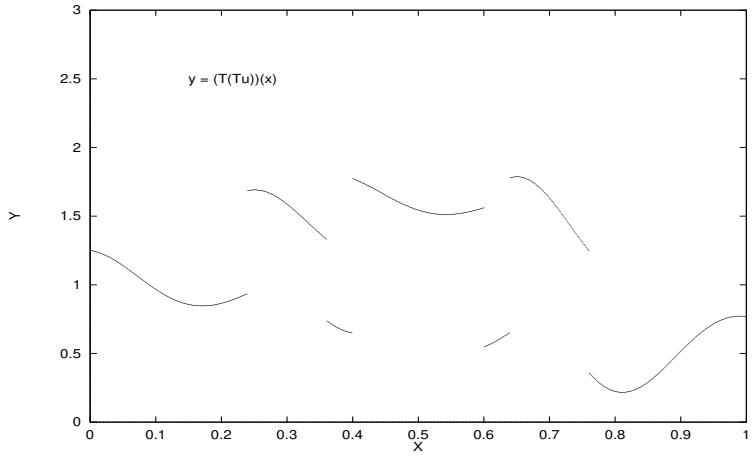


Figure 3(b): The function $w(x) = T(Tu)(x) = (T^{\circ 2}u)(x)$: the result of two applications of the fractal transform operator T.

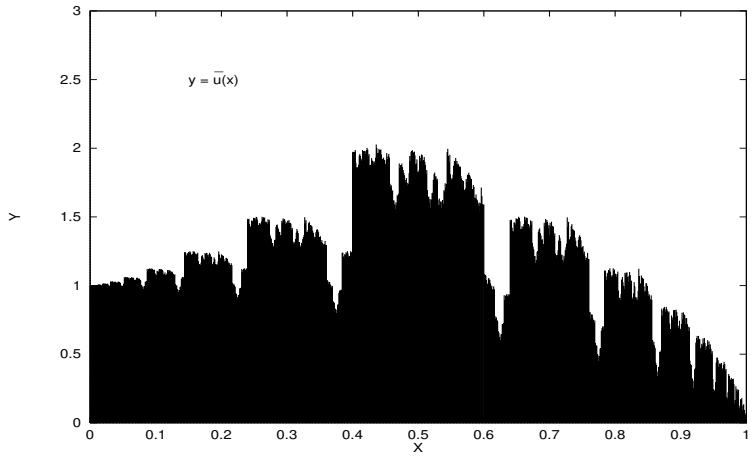


Figure 3(c): The "attractor" function $\bar{u} = T\bar{u}$ of the two-map IFSM given in the text.

In general, under suitable conditions on the IFS maps f_i and the grey-level maps ϕ_i, the operator T associated with an IFSM $(\mathbf{w}, \Phi)$ is contractive in the space $\mathcal{F}(X)$. Therefore, from the Banach Contraction Mapping Theorem, it possesses a unique "fixed point" function $\bar{u} \in \mathcal{F}(X)$. This is precisely the case with the 2-map IFSM given above. Its attractor is sketched in Figure 3(c). Note that from the fixed point property $\bar{u} = T\bar{u}$ and Eq. (18), the attractor $\bar{u}$ of an N-map IFSM satisfies the equation

$$\bar{u}(x) = \sum_{i=1}^{N}{}' \phi_i(\bar{u}(f_i^{-1}(x))), \qquad . \tag{21}$$

In other words, the graph of $\bar{u}$ satisfies a kind of "self-tiling" property: it may be written as a sum of distorted copies of itself.

Before going on, let's consider the three-map IFSM composed of the following IFS maps and associated grey-level maps:

$$\begin{aligned} f_1(x) &= \frac{1}{3}x, & \phi_1(y) &= \frac{1}{2}y, \\ f_2(x) &= \frac{1}{3}x + \frac{1}{3}, & \phi_2(y) &= \frac{1}{2}, \\ f_3(x) &= \frac{1}{3}x + \frac{2}{3}, & \phi_3(y) &= \frac{1}{2}y + \frac{1}{2}, \end{aligned} \tag{22}$$

Notice that $\hat{f}_1(X) = [0, \frac{1}{3}]$ and $\hat{f}_2(X) = [\frac{1}{3}, 1]$ overlap only at one point, $x = \frac{1}{3}$. Likewise, $\hat{f}_2(X)$ and $\hat{f}_3(X)$ overlap only at $x = \frac{2}{3}$. The fixed point attractor function $\bar{u}$ of this IFSM is sketched in Figure 4. It is known as the "Devil's Staircase" function. You can see that the attractor satisfies a self-tiling property: If you shrink the graph in the x-direction onto the interval $[0, \frac{1}{3}]$ and shrink the in y-direction by $\frac{1}{3}$, you obtain one piece of it. The second copy, on $[\frac{1}{3}, \frac{2}{3}]$, is obtained by squashing the graph to produce a constant. The third copy, on $[\frac{2}{3}, 1]$, is just a translation of the first copy by $\frac{2}{3}$ in the x-direction and $\frac{1}{2}$ in the y-direction. (Note: The observant reader can complain that the function graphed in Figure 6 is **not** the fixed point of the IFSM operator T as defined in Eq. (22): The value $v(\frac{1}{3})$ should be $\frac{3}{2}$ and not $\frac{1}{2}$, since $x = \frac{1}{3}$ is a point of overlap. In fact, this will also happen at $x = \frac{2}{3}$ as well as an infinity of points obtained by the action of the f_i maps on $x = \frac{1}{3}$ and $\frac{2}{3}$. What a mess! Well, not quite, since the function in Figure 7 and the true attractor differ on a countable infinity of points. Therefore, the the L^2 distance between them is zero! The two functions belong to the same *equivalence class* in $L^2([0, 1])$.)

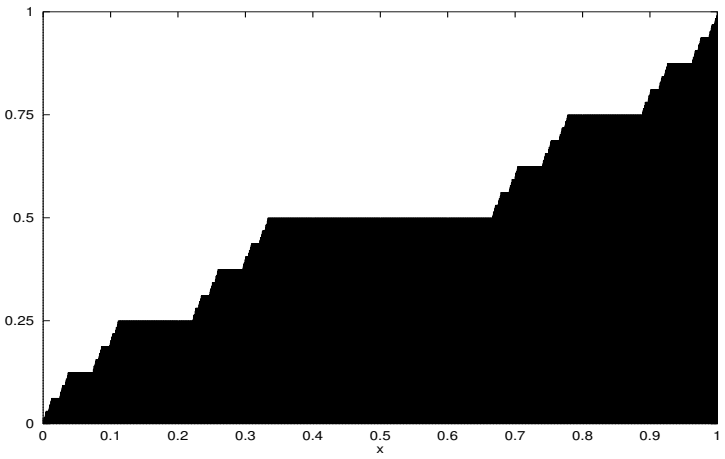


Figure 4: The "Devil's staircase" function, the attractor of the three-map IFSM given in Eq. (22).

Now we have an IFS-method of acting on functions. Along with a set of IFS maps f_i there is a corresponding set of grey-level maps ϕ_i. Together, Under suitable conditions, the determine a unique attracting fixed point function $\bar{u}$ which can be generated by iterating operator,T, defined in Eq. (vTu). As was the case with the "geometrical IFS" earlier, we are naturally led to the following **inverse problem for function (or image) approximation**:

> Given a "target" function (or image) v, can we find can we find an IFSM $(\mathbf{f}, \Phi)$ whose attractor $\bar{u}$ approximates v, i.e.,
>
> $$u \approx v\,? \tag{23}$$

We can make this a little more mathematically precise:

> Given a "target" function (or image) v and an $\epsilon > 0$, can we find an IFSM $(\mathbf{f}, \Phi)$ whose attractor $\bar{u}$ approximates v to within ϵ, i.e. satisfies the inequality $\| v - \bar{u} \| < \epsilon$?

Here, $\| \cdot \|$ denotes an appropriate norm for the space of image functions considered.

For the same reason as in the previous lecture, the above inverse problem may be reformulated as follows:

> Given a target function v, can we find an IFSM $(\mathbf{f}, \Phi)$ with associated operator T, such that
>
> $$u \approx Tu\,? \tag{24}$$

In other words, we look for a fractal transform T that maps the target image u as close as possible to itself. Once again, we can make this a little more mathematically precise:

> Given a target function u and an $\delta > 0$, can we find an IFSM $(\mathbf{f}, \Phi)$ with associated operator T, such that
>
> $$\| u - Tu \| < \delta \, ? \tag{25}$$

This basically asks the question, "How well can we 'tile' the graph of u with distorted copies of itself (subject to the operations given above)?" Now, you might comment, it looks like we're right back where we started. We have to examine a graph for some kind of "self-tiling" symmetries, involving both geometry (the f_i) as well as grey-levels (the ϕ_i), which sounds quite difficult. The response is "Yes, in general it is." However, it turns out that an enormous simplification is achieved if we give up the idea of trying to find the best IFS maps f_i. Instead, we choose to work with a **fixed** set of IFS maps f_i, $1 \leq i \leq N$, and then find the "best" grey-level maps ϕ_i associated with the f_i.

> **Question:** What are these "best" grey-level maps?
>
> **Answer:** They are the ϕ_i maps which will give the best "collage" or tiling of the function v with contracted copies of itself using the fixed IFS maps, w_i.

To illustrate, consider the target function $v = \sqrt{x}$. Suppose that we work with the following two IFS maps on [0,1]: $f_1(x) = \frac{1}{2}x$ and $f_2(x) = \frac{1}{2}x + \frac{1}{2}$. Note that $\hat{f}_1(X) = [0, \frac{1}{2}]$ and $\hat{f}_1(X) = [\frac{1}{2}, 1]$. The two sets $\hat{f}(X)$ overlap only at $x = \frac{1}{2}$.

(**Note:** It is very convenient to work with IFS maps for which the overlapping between subsets $\hat{f}_i(X)$ is minimal, referred to as the "nonoverlapping" case. In fact, this is the usual practice in applications. The remainder of this discussion will be restricted to the nonoverlapping case, so you can forget all of the earlier headaches involving "overlapping" and combining of fractal components.)

We wish to find the best ϕ_i maps, i.e. those that make $\| v - Tv \|$ small. Roughly speaking, we would like that

$$v(x) \approx (Tv)(x), \qquad x \in [0, 1], \tag{26}$$

or at least for as many $x \in [0,1]$ as possible. Recall from our earlier discussion that the first step in the action of the T operator is to produce copies of v which are contracted in the x-direction onto the subsets $\hat{f}_i(X)$. These copies, $a_i(x) = v(f_i^{-1}(x))$, $i = 1, 2$, are shown in Figure 5(a) along with the target $v(x)$ for reference. The final action is to modify these functions $a_i(x)$ to produce functions $b_i(x)$ which are to be as close as possible to the pieces of the original target function v which sit on the subsets $\hat{f}_i(X)$. Recall that this is the role of the grey-level maps ϕ_i since $b_i(x) = \phi_i(a_i(x))$ for all $x \in \hat{f}_i(X)$. Ideally, we would like grey-level maps that give the result

$$v(x) \approx b_i(x) = \phi_i(v(f_i^{-1}(x))), \qquad x \in \hat{f}_i(X). \tag{27}$$

Thus if, for all $x \in \hat{f}_i(X)$, we plot $v(x)$ vs. $v(f_i^{-1}(x))$, then we have an idea of what the map ϕ_i should look like. Figure 5(b) shows these plots for the two subsets $\hat{f}_i(X)$, $i = 1, 2$. In this particular example, the exact form of the grey level maps can be derived: $\phi_1(t) = \frac{1}{\sqrt{2}}t$ and $\phi_2(t) = \frac{1}{\sqrt{2}}\sqrt{t^2+1}$. I leave this as an exercise for the interested reader.

In general, however, the functional form of the ϕ_i grey level maps will not be known. In fact, such plots will generally produce quite scattered sets of points, often with several $\phi(t)$ values for a single t value. The goal is then to find the "best" grey level curves which pass through these data points. But that sounds like **least squares**, doesn't it? In most such "fractal transform" applications, only a straight line fit of the form $\phi_i(t) = \alpha_i t + \beta_i$ is assumed. For the functions in Figure 5(b), the "best" affine grey level maps associated with the two IFS maps given above are:

$$\begin{aligned} \phi_1(t) &= \frac{1}{\sqrt{2}}t, \\ \phi_2(t) &\approx 0.35216t + 0.62717. \end{aligned} \tag{28}$$

The attractor of this 2-map IFSM, shown in Figure 5(c), is a very good approximation to the target function $v(x) = \sqrt{x}$.

In principle, if more IFS maps w_i and associated grey level maps ϕ_i are employed, albeit in a careful manner, then a better accuracy should be achieved. The primary goal of IFS-based methods of image compression, however, is not necessarily to provide approximations of arbitary accuracy, but rather to provide approximations of acceptable accuracy "to the discerning eye" with as few parameters as

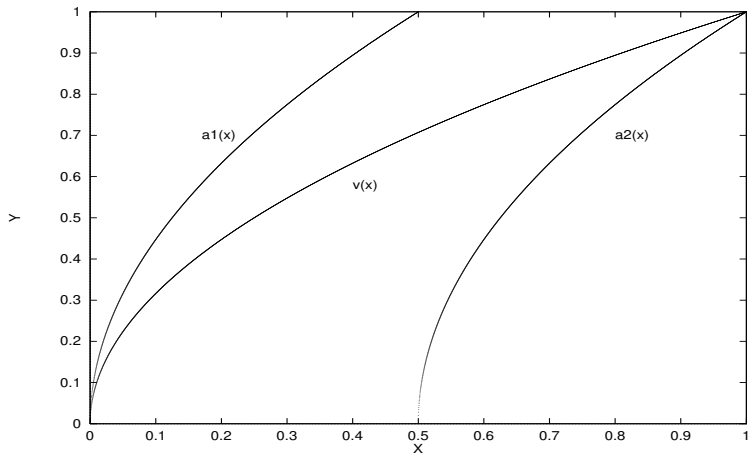


Figure 5(a): The target function $v(x) = \sqrt(x)$ on [0,1] along with its contractions $a_i(x) = v(w_i^{-1}(x))$, $i = 1, 2$, where the two IFS maps are $w_1(x) = \frac{1}{2}x$, $w_2(x) = \frac{1}{2}x + \frac{1}{2}$.

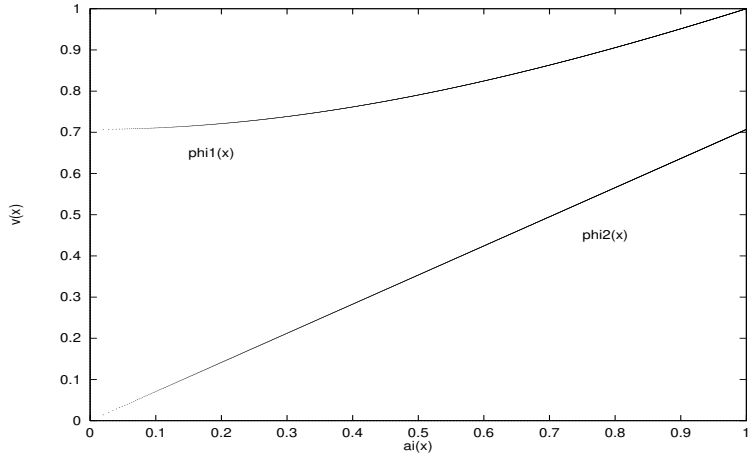


Figure 5(b): Plots of $v(x)$ vs $a_i(x) = v(w_i^{-1}(x))$ for $x \in w_i(X)$, $i = 1, 2$. These graphs reveal the grey level maps ϕ_i associated with the two-map IFSM.

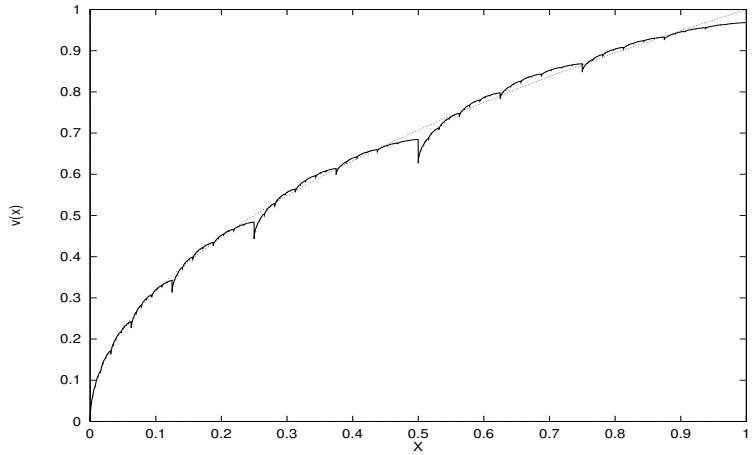


Figure 5(c): The attractor of the two-map IFSM with grey level maps given in Eq. (28).

possible. As well, it is desirable to be able to compute the IFS parameters in a reasonable amount of time.

"Local IFSM"

That all being said, there is still a problem with the IFS method outlined above. It works fine for the examples that were presented but these are rather special cases – all of the examples involved monotonic functions. In such cases, it is reasonable to expect that the function can be approximated well by combinations of spatially-contracted and range-modified copies of itself. In general, however, this is not guaranteed to work. A simple example is the target function $u(x) = \sin \pi x$ on [0,1], the graph of which is sketched in Figure 6 below.

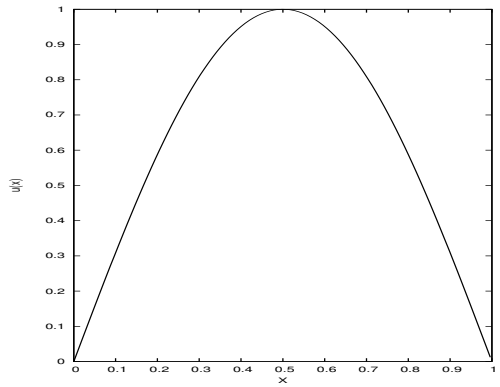


Figure 6: Target function $u(x) = \sin \pi x$ on [0,1]

Suppose that we try to approximate $u(x) = \sin \pi x$ with an IFS composed with the two maps,

$$f_1(x) = \frac{1}{2}x \qquad f_2(x) = \frac{1}{2} + \frac{1}{2}\,. \tag{29}$$

It certainly does not look as if one could express $u(x) = \sin \pi x$ with two contracted copies of itself which lie on the intervals $[0, 1/2]$ and $[1/2, 1]$. Nevertheless, if we try it anyway, we obtain the result shown in Figure 7. The best "tiling" of $u(x)$ with two copies of itself is the constant function, $\bar{u}(x) = \frac{2}{\pi}$, which is the mean value of $u(x)$ over $[0, 1]$.

If we stubbornly push ahead and try to express $u(x) = \sin \pi x$ with four copies of itself, i.e., use the four IFS maps,

$$f_1(x) = \frac{1}{4}x\,, \quad f_2(x) = \frac{1}{4}x + \frac{1}{4}\,, \quad f_3(x) = \frac{1}{4}x + \frac{1}{2}\,, \quad f_4(x) = \frac{1}{4}x + \frac{3}{4}\,, \tag{30}$$

then the attractor of the "best four-map IFS" is shown in Figure 8. It appears to be a piecewise constant function as well.

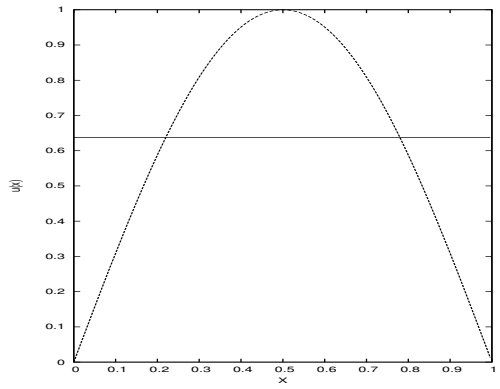


Figure 7: IFSM attractor obtained by trying to approximate $u(x) = \sin \pi x$ on [0,1] with two copies of itself.

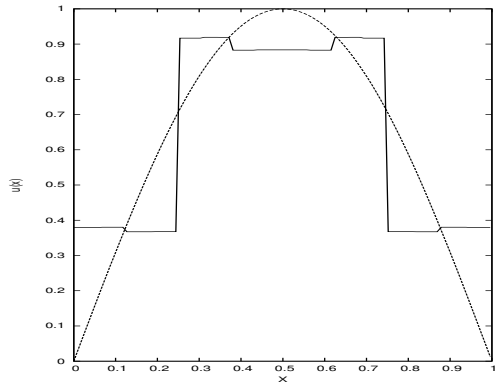


Figure 8: IFSM attractor obtained by trying to approximate $u(x) = \sin \pi x$ on [0,1] with four copies of itself.

Of course, we can increase the number of IFS maps to produce better and better piecewise constant approximations to the target funcion $u(x)$. But we really don't need IFS to do this. A better strategy, which follows a method A significant improvement, which follows a method introduced in 1989 by A. Jacquin, then a Ph.D. student of Prof. Barnsley, is to break up the function into "pieces", i.e., consider it as a collection of functions defined over subintervals of the interval X. Instead of trying to express a function as a union of copies of spatially-contracted and range-modified copies of itself, the modified method, known as "local IFS," tries to express each "piece" of a function as a spatially-contracted and range-modified copie of larger "pieces" of the function, not the entire function. We illustrate by considering once again the target function $u(x) = \sin \pi x$. It can be viewed as a union of two monotonic functions which are defined over the intervals $[0, 1/2]$ and $[1/2, 1]$. But neither of these "pieces" can, in any way, be considered as spatially-contracted copies of other monotone functions extracted from $u(x)$. As such, we consider $u(x)$ as the union of four "pieces," which are supported on the so-called

"range" intervals,

$$I_1 = [0, 1/4]\,, \quad I_2 = [1/4, 1/2]\,, I_3 = [1/2, 3/4]\,, \quad I_4 = [3/4, 1]\,. \tag{31}$$

We now try to express each of these pieces as spatially-contracted and range-modified copies of the two larger "pieces" of $u(x)$ which are supported on the so-called "domain" intervals,

$$J_1 = [0, 1/2] \quad J_2[1/2, 1]\,. \tag{32}$$

In principle, we can find IFS-type contraction maps which map each of the J_k intervals to the I_l intervals. But we can skip these details. We'll just present the final result. Figure 9 shows the attractor of the IFS that produces the best "collage" of $u(x) = \sin \pi x$ using this 4 domain block/2 range block method. It clearly provides a much better approximation than the earlier four-IFS-map method.

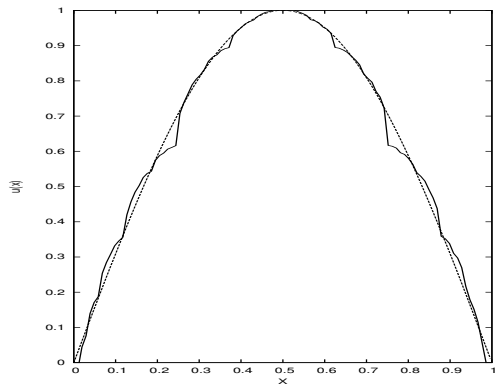


Figure 9: IFSM attractor obtained by trying to approximate $u(x) = \sin \pi x$ on [0,1] with four copies of itself.

Fractal image coding

We now outline a simple block-based fractal coding scheme for a greyscale image function, for example, 512×512 pixel *Boat* image shown back in Figure 1(a).

In what follows, let X be an $n_1 \times n_2$ pixel array on which the image u is defined.

- Let $\mathcal{R}^{(n)}$ denote a set of $n \times n$-pixel **range** subblocks R_i, $1 \le i \le N_{\mathcal{R}^{(n)}}$, which cover X, i.e., $X = \cup_i R_i$.

- Let $\mathcal{D}^{(m)}$ denote a set of $m \times m$-pixel **domain** D_j, $1 \le j \le N_{\mathcal{D}^{(m)}}$, where $m = 2n$. (The D_i are not necessarily non-overlapping, but they should cover X.) These two partitions of the image are illustrated in Figure 10.

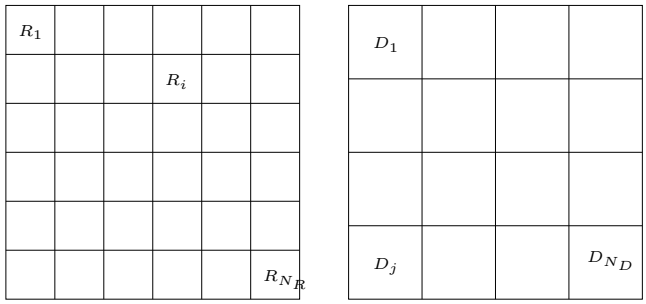


Figure 10: Partitioning of an image into range and domain blocks.

- Let $w_{ij} : D_j \to R_i$ denote the affine geometric transformations that map domain blocks D_j to R_i. There are 8 such constraction maps: 4 rotations, 2 diagonal flips, vertical and horizontal flips, so the maps should really be indexed as w_{ij}^k, $1 \le k \le 8$. In many cases, only the zero rotation map is employed so we can ignore the k index, which we shall do from here on for simplicity.

  Since we are now working in the discrete domain, i.e., pixels, as opposed to continuous spatial variables (x, y), some kind of "decimation" is required in order to map the larger $2n \times 2n$-pixel domain blocks to the smaller $n \times n$-pixel range blocks. This is usually accomplished by a "decimation procedure" in which nonoverlapping 2×2 square pixel blocks of a domain block D_j are replaced with one pixel. This definition of the w_{ij} maps is a formal one in order to identify the spatial contractions that are involved in the fractal coding operation.

  The decimation of the domain block D_j is accompanied by a decimation of the image block $u(D_j)$ which is supported on it, i.e., the $2n \times 2n$ greyscale values that are associated with the pixels in D_j. This is usually done as follows: The greyscale value assigned to the pixel replacing four pixels in a 2×2 square is the average of the four greyscale values over the square that has been decimated. The result is an $n \times n$-pixel image, to be denoted as $\tilde{u}(D_j)$, which is the "decimated" version of $u(D_j)$,

- For each range block R_i, $1 \le i \le N_{\mathcal{R}^{(n)}}$, compute the errors associated with the approximations,

$$u(R_i) \approx \phi_{ij}(u(w_{ij}^{-1}(R_i)) = \phi_{ij}\tilde{u}(D_j)\,, \quad \text{for all} \;\; 1 \le j \le N_{\mathcal{D}^{(m)}}\,, \tag{33}$$

where, for simplicity, we use affine greyscale transformations,

$$\phi(t) = \alpha t + \beta\,. \tag{34}$$

The approximation is illustrated in Figure 11.

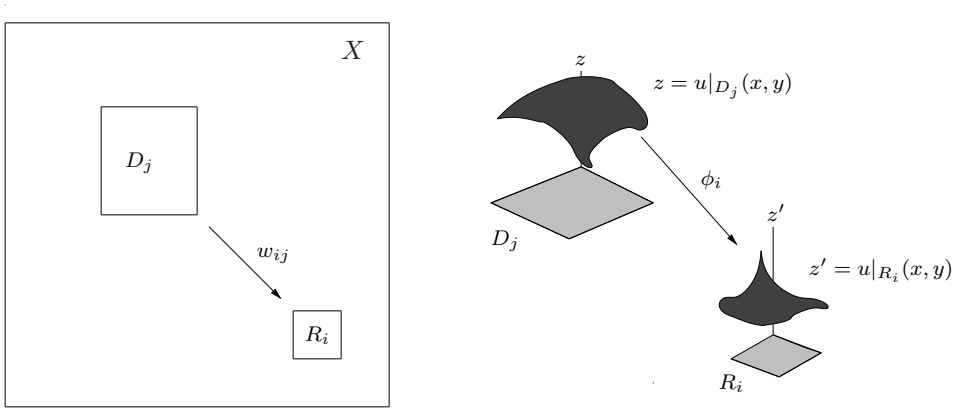


Figure 11. Left: Range block R_i and associated domain block D_j. **Right:** Greyscale mapping ϕ from $u(D_j)$ to $u(R_i)$.

In each such case, one is essentially determining the best straight line fit through n^2 data points $(x_k, y_k) \in \mathbb{R}^2$, where the x_k are the greyscale values in image block $\tilde{u}(D_j)$ and the y_k are the corresponding greyscale values in image block $u(R_i)$. (Remember that you may have to take account of rotations or inversions involved in the mapping w_{ij} of D_j to R_j.) This can be done by the method of least squares, i.e., finding α and β which minimize the total squared error,

$$\Delta^2(\alpha, \beta) = \sum_{k=1}^{n} (y_i - \alpha x_i + \beta)^2\,. \tag{35}$$

As is well known, minimization of Δ^2 yields a system of linear equations in the unknowns α and β.

Now let Δ_{ij}, $1 \le j \le \mathcal{D}^{(m)}$ denote the approximation error Δ associated with the approximations to $u(R_i)$ in Eq. (33). Choose the domain block $j(i)$ that yields the lowest approximation error.

The result of the above procedure: You have fractally encoded the image u. The following set of parameters for all range blocks R_i, $1 \le i \le N_{\mathcal{R}^{(n)}}$,

$$\begin{aligned} &j(i), && \text{index of best domain block,} \\ &\alpha_i, \beta_i, && \text{affine greyscale map parameters,} \end{aligned} \tag{36}$$

comprises the **fractal code** of the image function u. The fractal code defines a fractal transform T. The fixed point $\bar{u}$ of T is an approximation the image u, i.e.,

$$u \approx \bar{u} = T\bar{u}\,. \tag{37}$$

This is happening for the same reason as for our IFSM function approximation methods outlined in the previous section. Minimization of the approximation errors in Eq. (33) is actually minimizing the "tiling error"

$$\|u - Tu\|\,, \tag{38}$$

originally presented in Eq. (25). We have found a fractal transform operator that maps the image u – in "pieces," i.e., in blocks – close to itself.

Moral of the story: You store the fractal code of u and generate its approximation $\bar{u}$ by iterating T, as shown in the next example.

In Figure 12, are shown the results of the above block-based IFSM procedure as applied to the 512×512 *Boat* image. 8×8-pixel blocks were used for the range blocks R_i and 16×16-pixel blocks for the domain blocks D_j. As such, there are 4096 range blocks and 1024 domain blocks.

The bottom left image of Figure 12 is the fixed point attractor $\bar{u}$ of the fractal transform defined by the fractal code obtained in this procedure.

You may still be asking the question, "How to we iterate the fractal transform T to obtain its fixed point attractor?" Very briefly, we start with a "seed image," u_0, which could be the zero image, i.e., an image for which the greyscale value at all pixels is zero. You then apply the fractal operator T to u_0 to obtain a new image u_1, and then continue with the iteration procedure,

$$u_{n+1} = Tu_n\,,\ n \geq 0\,. \tag{39}$$

After a sufficient number of iterations (around 10-15) for 8×8 range blocks, the above iteration procedure will have converged.

But perhaps we haven't answered the question completely. At each stage of the iteration procedure, i.e, at step n, when you wish to obtain u_{n+1} from u_n, you must work with each of its range

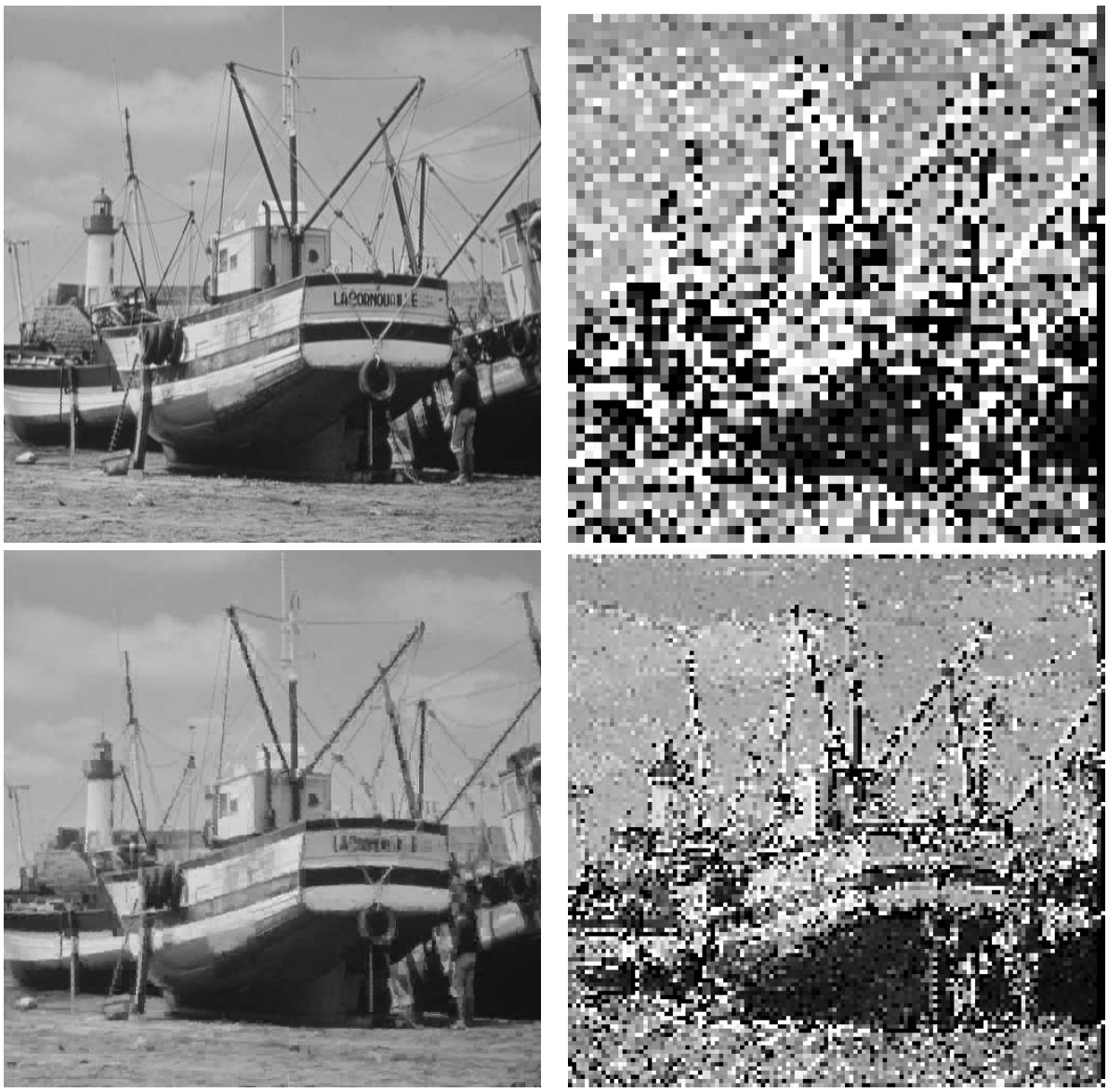


Figure 12. Clockwise, starting from top left: Original *Boat* image. The iterates u_1 and u_2 and fixed point approximation $\bar{u}$ obtained by iteration of fractal transform operator. ($u_0 = \mathbf{0}$.) 8×8-pixel range blocks. 16×16-pixel domain blocks.

blocks R_i separately. One replaces image block $u_n(R_i)$ supported on R_i with a suitably modified version of the image $u_n(D_{j(i)})$ on the domain block $D_{j(i)}$ as dictated by the fractal code. The image block $u_n(D_{j(i)})$ will first have to be decimated. (This can be done at the start of each iteration step, so you don't have to be decimating each time.) It is also important to make a copy of u_n so you don't modify the original while you are constructing u_{n+1}! Remember that the fractal code is determined

by approximating u_n with parts of itself!)

There are still a number of other questions and points that could be discussed. For example, better approximations to an image can be obtained by using smaller range blocks, R_i, say 4×4-pixel blocks. But that means small domain blocks D_j, i.e., 8×8 blocks, which means greater searching to find an optimal domain block for each range block. The searching of the "domain pool" for optimal blocks is already a disadvantage of the fractal coding method.

That being said, various methods have been investigated and developed to speed up the coding time by reducing the size of the "domain pool." This will generally produces less-than-optimal approximations but in many cases, the loss in fidelity is almost non-noticeable.

Some references (Yes, these are old!)

Original research papers

- J. Hutchinson, Fractals and self-similarity, Indiana Univ. J. Math. **30**, 713-747 (1981).
- M.F. Barnsley and S. Demko, Iterated function systems and the global construction of fractals, Proc. Roy. Soc. London **A399**, 243-275 (1985).
- A. Jacquin, Image coding based on a fractal theory of iterated contractive image transformations, IEEE Trans. Image Proc. **1** 18-30 (1992).

Books

- M.F. Barnsley, *Fractals Everywhere*, Academic Press, New York (1988).
- M.F. Barnsley and L.P. Hurd, *Fractal Image Compression*, A.K. Peters, Wellesley, Mass. (1993).
- Y. Fisher, *Fractal Image Compression, Theory and Application*, Springer-Verlag (1995).
- N. Lu, *Fractal Imaging*, Academic Press (1997).

Expository papers

- M.F. Barnsley and A. Sloan, A better way to compress images, *BYTE* Magazine, January issue, pp. 215-223 (1988).
- Y. Fisher, A discussion of fractal image compression, in *Chaos and Fractals, New Frontiers of Science*, H.-O. Peitgen, H. Jürgens and D. Saupe, Springer-Verlag (1994).