Algorithms for Plotting Julia Sets

As a reminder, the Julia set for a given function (or set of functions, in the case of an iterated function system), is the set of points that are invariant to function iteration.

For example,

- The unit circle is invariant to the function $f(z) = z^2$
- The Sierpinski triangle is invariant to the union of the images of the three maps that make up the IFS.

We will look at two algorithms for plotting the Julia set. One method is used when the Julia set is an attracting set (for example, forward iteration of the IFS for the Sierpinksi triangle), and another method is used when the set is a repelling set (for example, the unit circle when iterating z^2.

When the Julia set is attracting, we can use the Random Iteration Algorithm to construct the set. The algorithm works because of the existence of the special orbit that comes close to every point of the set. The following is Matlab code for the Sierpinski triangle- You can either type it in directly to the command window, or create a script file with these commands (save as something like, `RandomIteration.m`.

The Random Iteration Algorithm
Initialize by choosing a random point in the plane, $\mathbf{x}$.

1. Choose a function $f_i(x)$ at random.
2. Compute $f_i(\mathbf{x})$ and plot it.
3. Reset $\mathbf{x} \leftarrow f_i(\mathbf{x})$.
4. Repeat from step 1.

In Matlab, this might look like the following:

```
% Random Iteration Algorithm
%  Set up for the Sierpinski triangle.


% Create the table for the IFS (number of functions x 6 numbers)
A=[0.5 0 0 0.5 0 0
   0.5 0 0 0.5 0.5 0
   0.5 0 0 0.5 0 0.5];

[NumFunc,N]=size(A);
if N~=6
```

```
  error('Number of entries in each row of matrix should be 6')
end

%Randomly initialize a point:
x=randn(2,1);

%Number of iterations (number of points to plot):
NumIts=5000;

% The matrix Y will hold the points to plot at the end:
Y=zeros(2,NumIts);
Y(:,1)=x(:);

for j=1:500
  % Discard the first 100 points
  i=randi(NumFunc);
  x(1)=A(i,1)*Y(1,1)+A(i,2)*Y(2,1)+A(i,5);
  x(2)=A(i,3)*Y(1,1)+A(i,4)*Y(2,1)+A(i,6);
  Y(:,1)=x;
end


for j=1:NumIts-1
  % Select the function at random:
  i=randi(NumFunc);
  Y(1,j+1)=A(i,1)*Y(1,j)+A(i,2)*Y(2,j)+A(i,5);
  Y(2,j+1)=A(i,3)*Y(1,j)+A(i,4)*Y(2,j)+A(i,6);
end
```

Matlab and Complex Numbers

Before going into the complex plane, let's look at some Matlab commands for complex numbers:

Matlab	Explanation
`abs`	Complex magnitude
`angle`	Phase angle (argument in $[-\pi, \pi]$).
`conj`	Complex conjugate
`i` or `j`	Imaginary unit, $\sqrt{-1}$
`real`	Real part of the complex number
`imag`	Imaginary part of the complex number
`complex`	Makes a complex number. `complex(3,4)` is $3 + 4i$.

Using the Matlab commands, the polar form for complex number z is:

```
z = abs(z) * exp(i * angle(z) )
```

We'll be studying $Q_c(z) = z^2 + c$- Matlab can perform these operations using complex numbers, as well as take the square root- but the root will be the "positive square root". Let's look at an example:

$$z = 1 + i = \sqrt{2}\mathrm{e}^{i\frac{\pi}{4}} \quad \Rightarrow \quad \sqrt{z} = 2^{1/4}\mathrm{e}^{i\frac{\pi}{8}}, \text{ and } 2^{1/4}\mathrm{e}^{i\frac{9\pi}{8}}$$

To check this in Matlab, type:

```
z=1+i;
sqrt(z)
2^(1/4)*exp(i*pi/8)
2^(1/4)*exp(i*9*pi/8)
```

Random Iteration for $Q_c(z)$

While forward iteration of $Q_c(z)$ generates a Julia set that is repelling, iterating the **inverse** will make the set **attracting**. Therefore, we can use the random iteration algorithm to plot our Julia sets as well- Think of it as having two functions in something like an IFS; one function is the positive root, the other is the "negative root".

Rewriting the code from before,

```
% Random Iteration Algorithm - Q_c(z)

c=0; %Comes from Q_c(z)=z^2+c
%c=-0.84+0.17*i; %Try this one!

% Initialize a point in the complex plane:
z=complex(rand(), rand());

NumIts=1000;

Y=zeros(1,NumIts);
Y(1)=z;

for k=1:NumIts-1

   m=randi(2);
   if m==1
      Y(k+1)=sqrt(Y(k)-c);
   elseif m==2
```

```
      Y(k+1)=-sqrt(Y(k)-c);
   end

end
plot(Y,'.');
axis equal;
```

Escape Time Algorithms

In this case, the points on the Julia set are repelling, so that points that are "outside" the set initially will move away from the set. Similarly, if the Julia set is more like a Cantor set, then all the other points move away from the set.

Now, we take a "window" of the plane, like $-2 \leq x \leq 2$ and $-2 \leq y \leq 2$, and then subdivide that further into an array of small boxes. We will iterate the point at the center of each box, and if the point "escapes" (the magnitude is greater than some amount), then we will store the iteration value and plot the original starting point with a color corresponding to the iteration on which the point escapes. If the point never escapes, we might leave that point black.

The Escape Time Algorithm
Initialize by subdividing the plane into an array of points. For each (x, y) in the array:

1. Iterate the function using the point (x, y).
2. If the length of the point exceeds a certain number, stop and store the iteration number. Paint the original coordinate (x, y) a color corresponding to the iteration number.

   It's a good idea to have a default value (like 0) if the point never escapes.
3. Repeat, using the next point in the array.

In the case of the Julia set for $Q_c(z) = z^2+c$, the escape time algorithm is straightforward. Here is some code- You might notice that it doesn't do exactly what we described above. What is it doing instead?

EscapeTime01.m

```
c = -0.84+0.17*i;

NumPtsX=500; % Number of points in the mesh for one axis
Len=1.5;
MaxIts=30;  % Maximum number of iterations
x=linspace(-Len,Len,NumPtsX);
y=linspace(-Len,Len,NumPtsX);
[xtrans,ytrans]=meshgrid(x,y);
```

```
ztrans=xtrans+i*ytrans;

for k=1:MaxIts;
    ztrans=ztrans.^2+c;
    t=exp(-abs(ztrans));
end
colormap jet
pcolor(t);
shading flat;
```

Escape Time for IFS

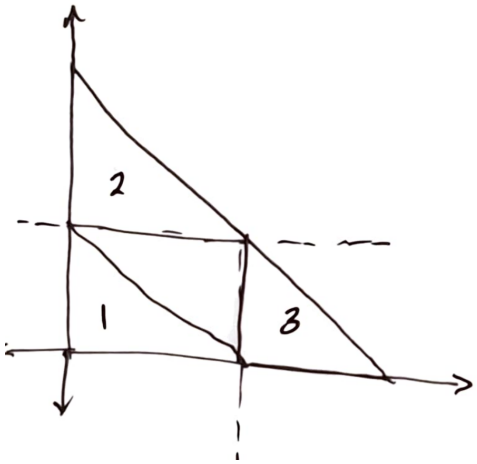
The inverse for an IFS is a little tricky. Since the image of each (forward) function is one piece of the whole, we subdivide the plane using those pieces. On the fractal itself it is straightforward, but extending those inverses to points on the plane is not unique. Below we have subdivided the plane into three pieces for the Sierpinksi triangle. When we get a point (x, y), we first have to locate where in the plane it lies, and then we will apply the appropriate inverse function to it.

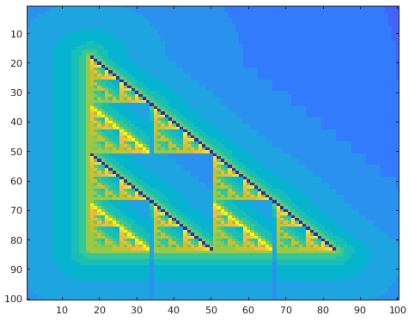
There are three regions in the plane below. The region for "function 1" is the set of all (x, y) such that both x and y are less than or equal to $1/2$. The region for "function 2" is the set of all (x, y) such that $y > 1/2$, and the region for "function 3" is the region where $x > 1/2$ and $y \leq 1/2$.

Remember that our functions are each of the form below, which gives the inverse function shown:

$$f_i(\mathbf{x}) = A\mathbf{x} + \mathbf{b} \quad \Rightarrow \quad f_i^{-1}(\mathbf{x}) = A^{-1}(\mathbf{x} - \mathbf{b})$$

So again, once the region corresponding to a given point is located, we then apply the appropriate inverse function to it. Below we show the result (using a 100×100 grid of points).



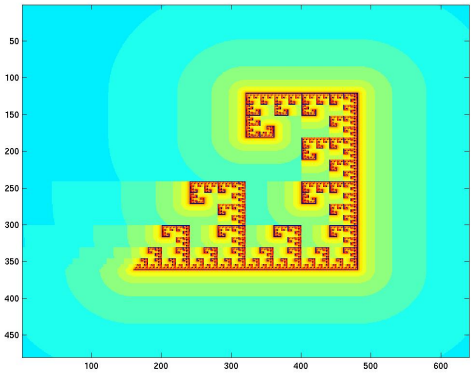


Homework for the Lab

On the class website, there are several scripts- some for the escape time algorithm and some for the random iteration algorithm. Try each of them out before continuing.

The homework is to produce a plot using both the random iteration algorithm and the escape time algorithm for each of the following Julia sets:

1. For $Q_c(z) = z^2 + c$, if $c = -1$
2. For $Q_c(z) = z^2 + c$, if $c = 0.3 + 0.6i$
3. The IFS for the "rug" shown below (you don't need to copy the coloring that I have below):



4. For the "Barnsley Fern" below, geometrically determine a way of subdividing the fern for the inverse functions. You only need to do this geometrically, and you can look up the Barnsley Fern on Wikipedia if you're not sure of the IFS.