Fractal Friday!

Adam Fattal (adam@adamfattal.com)

March 6, 2020

A fractal is a pattern that never ends and is self-similar on every scale. They are generated by something called a positive feedback loop, which is essentially when a function that's reiterated yeilds the same result. Fractal patterns could be found everywhere in nature, from seashells to trees to galactic structures. Although the nature of fractals are complicated, the algorithms and processes that are used to produce them are fairly simple. And that's what *Fractal Friday* is all about, using basic algorithms to produce interesting fractal patterns. By default, all algorithms will be written in MATLAB. However, if you need a Python, Java, or any other alternitive, send us an email and we will be glad to find an alternitive!
-Adam (@phy.sics) and Yogesh (@yogesh_wagh_1729)

1 Week 1: The Sierpinski Triangle!

For the first pattern, we're going discussing is what is perhaps the most iconic Fractal Pattern, the Sierpinski triangle. The algorithm that is used to produce this pattern is known as a Chaos Game. The game goes as following [1]:

1. Pick 3 points which could be connected to a triangle (but don't connect them). For clarity's sake, we'll label them Points **A, B, and C**.

2. Pick an arbitraty point that exists within the region enclosed by the "imaginary triangle".

3. Get a dice and roll it. If the dice rolls to 1 or 2, mark a new point where its coordinates is half the distance between the initial point and **A**. If the dice rolls 3 or 4, do the same but for point **B**, and if it rolls 5 or 6, do the same for point **C**.

4. Using the new point as the new starting point, repeat the process a large number of time (our script reiterated 15,000 times).

[1]Note: You can actually try this out on pen and paper, however it will take a while for you to notice a clear pattern

The output of this Chaos game should be something that resembles the following:

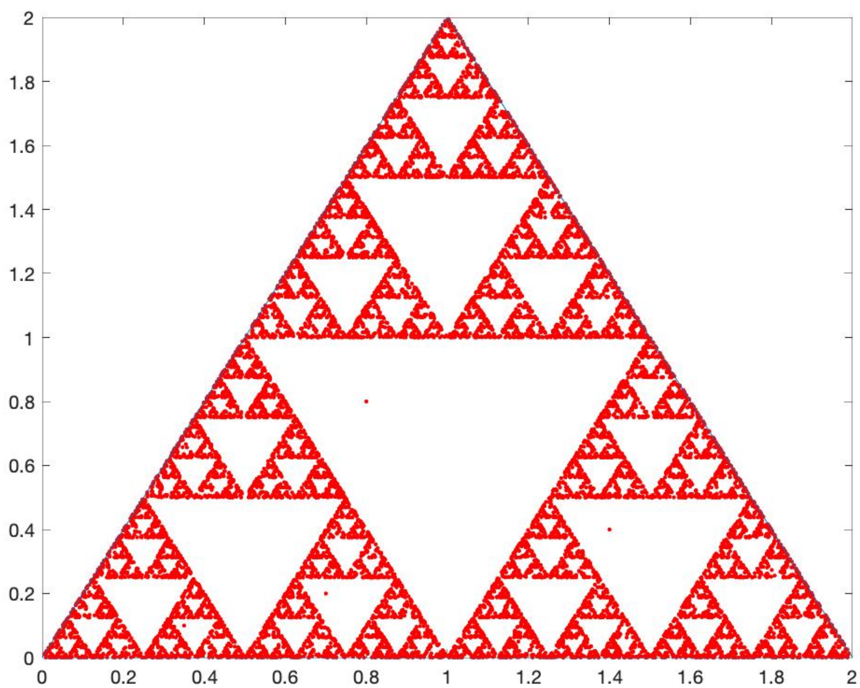

Fig. 1: A Sierpinski Triangle Formed by the MATLAB Script on Page 3.

Matlab Code

```

%% triangle co-ordinates
m=[0 1 2];
n=[0 2 0];


%starting point
ax=zeros(1,15001);
ay=zeros(1,15001);
ax(1)=0.8;
ay(1)=0.8;

%% dice
r=randi(6,1,15000); %15000 random integers frpm 1 to 6
for i=1:15000
    if r(1,i)==1
        x=0;
        y=0;
        elseif r(1,i)==2
        x=0;
        y=0;
        elseif r(1,i)==3
        x=1;
        y=2;
    elseif r(1,i)==4
        x=1;
        y=2;
    else
        x=2;
        y=0;
    end
    ax(1,i+1)=(ax(1,i)+x)/2;
    ay(1,i+1)=(ay(1,i)+y)/2;
    plot(ax(1,i),ay(1,i),'r.');
    hold on;
end

line([m(1),m(2)],[n(1),n(2)]);
line([m(2),m(3)],[n(2),n(3)]);
line([m(1),m(3)],[n(1),n(3)]);
```

2 Week 2: Barnsley Fern!

The next pattern is a pattern that seems to shock people as soon as they see it. Not because it is too abstract to be fathomed by the average mind, but for the exact opposite reason. It is shocking because it is something that is natural and something we see in our everyday lives, especially because it displays how a very familliar structure can be generated by a simple mathematical algorithm.

There are 4 2-dimensional transformations necessary to produce Barnsley's Fern, each one being an iterated function. That is what we call an ISF, or Iterated Function System. The system goes as following:

- $f_1 = \begin{bmatrix} 0.00 & 0.00 \\ 0.00 & 0.16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

- $f_2 = \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 1.60 \end{bmatrix}$

- $f_3 = \begin{bmatrix} 0.20 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 0.26 \end{bmatrix}$

- $f_4 = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 0.44 \end{bmatrix}$

The output of these four functions are:

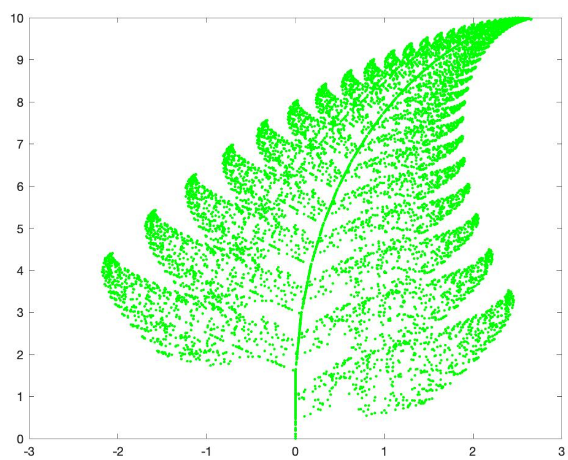

Fig. 2: A Barnsley Fern Formed by the MATLAB Script on Page 4.

f_1 produces the stem of the Fern, f_2 produces the smaller leaflets, f_3 produces the largest left-hand side leaflet, and f_4 the largest right-hand side leaflet.

Fig. 2 was produced with 10,000 iterations!

Matlab Code

```
r=rand(1,10000);
x(1,10000)=zeros;
y(1,10000)=zeros;
x(1,1)=0;
y(1,1)=0;
for i=1:9999
    if r(i)<0.01
        x(i+1)=0;
        y(i+1)=0.16*y(i);
    elseif r(i)<0.85
        x(i+1)=0.85*x(i)+0.04*y(i);
        y(i+1)=-0.04*x(i)+0.85*y(i)+1.6;
    elseif r(i)<0.93
        x(i+1)=0.2*x(i)-0.26*y(i);
        y(i+1)=0.23*x(i)+0.22*y(i)+1.6;
    else x(i+1)=-0.15*x(i)+0.28*y(i);
        y(i+1)=0.26*x(i)+0.24*y(i)+0.44;
    end
    plot(x(i),y(i),'g.');
    hold on;
end
```

3 Week 3: Mandelbrot Set!

This week, the pattern we're going to be discussing is what is known as the *Mandelbrot Set*. The Mandelbrot Set can be obtained by contour plotting the set of all complex numbers, z, where the following condition is applied: $f(z) = z^2 + c$ converges when z is reiterated from $z = 0$. (i.e $f(0), f(f(0)), f(f(f(0))), ...$)

Plotting $Im[c]$ vs. $Re[c]$, the following image is produced:

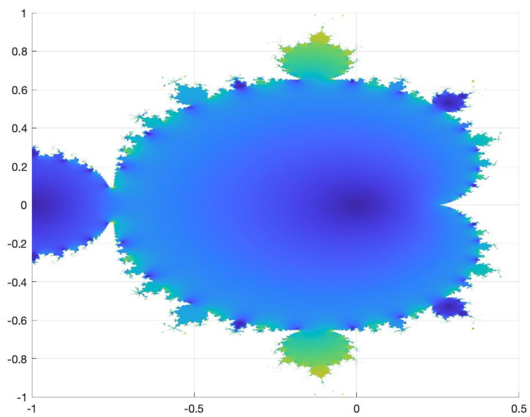

Fig. 3: The Mandelbrot Set Plotted by the MATLAB Script below.

Matlab Code

```
r=rand(1,10000);
x(1,10000)=zeros;
y(1,10000)=zeros;
x(1,1)=0;
y(1,1)=0;
for i=1:9999
    if r(i)<0.01
        x(i+1)=0;
        y(i+1)=0.16*y(i);
    elseif r(i)<0.85
        x(i+1)=0.85*x(i)+0.04*y(i);
        y(i+1)=-0.04*x(i)+0.85*y(i)+1.6;
    elseif r(i)<0.93
        x(i+1)=0.2*x(i)-0.26*y(i);
        y(i+1)=0.23*x(i)+0.22*y(i)+1.6;
    else x(i+1)=-0.15*x(i)+0.28*y(i);
        y(i+1)=0.26*x(i)+0.24*y(i)+0.44;
    end
    plot(x(i),y(i),'g.');
    hold on;
end
```

4 Week 4: Fractal Tree!

Recursion is the process of a function referring to itself. If you have followed this series, you'd be very familiar with this process, because all fractals display recusion. In computer code, a recursion would happen when you have a function that has an operation which leads the function to call on itself. Suppose you have a tree that symmetrically branches out into 2 braches (symetrically meaning that the length and angle of the left branch is equal to the right brach), and then those 2 branches symmetrically branch out into 4, then 8, etc. That is a recursion process that leads to a fractal pattern. That fractal pattern is called a Fractal Tree.

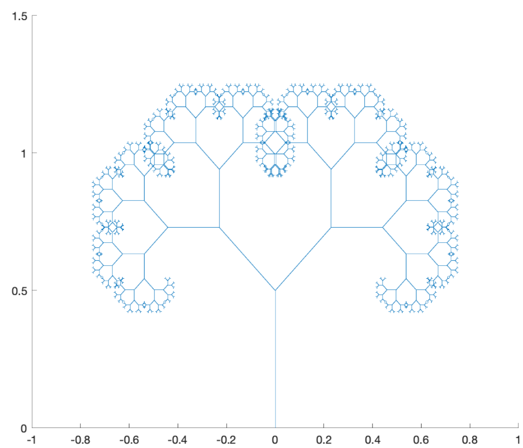

Fig. 4: A Fractal Tree Plotted by the MATLAB Script below.

Matlab Code

```
function def_xyz=tree(xold,yold,theta,length)

% xold,yold = value of x and y co-ordinate for every
    iteration
% theta= angle of rotation for every new branch
%length= length of branch for every new iteration


ratio=0.65; % multiplication factor for length of new
    branch that'll reduce the length

%new co-ordinates
```

```
xnew=xold+length*cos(theta);
ynew=yold+length*sin(theta);

if length > 0.005

    line([xold,xnew],[yold,ynew]);

    % brach on right side of every iteration
    tree(xnew,ynew,theta+pi/4,length*ratio);

    % brach on left side of every iteration
    tree(xnew,ynew,theta-pi/4,length*ratio);
    axis([-1 1 0 1.5]);
    pause(0.001);

end
end
```

5 Week 5: Random Fractal Tree

Last week, we covered the classical Fractal Tree, which is totally based on a systematic set of branching of lines with a constant length and angle ratio. This week's pattern is a little different. We will assign a *randi*() function that governs the ratio of branch$_n$ to branch$_{n+1}$. It assigns a random value between 0.2 and 0.9, which changes in every iteration, resulting in this asymetrical figure. In fact, each time you run the program, you will get a totally different figure. Another thing we did is we added a random color coder to the code which results in an even more exotic figure.

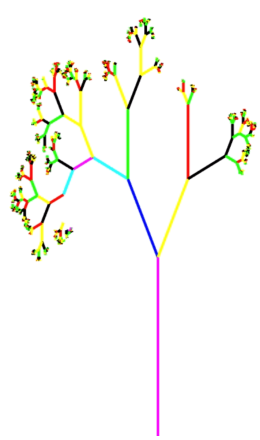

Fig. 5: An Asymmetrical Random Fractal Tree Plotted by the MATLAB Script below.

Matlab Code

```
function def_xyz=tree(xold,yold,theta,length)

% xold,yold = value of x and y co-ordinate for every
    iteration
% theta= angle of rotation for every new branch
%length= length of branch for every new iteration


% ratio=0.65; % multiplication factor for length of new
    branch that'll reduce the length
ratio=randi([20,90])/100;
%new co-ordinates
xnew=xold+length*cos(theta);
ynew=yold+length*sin(theta);

if length > 0.005

    line([xold,xnew],[yold,ynew],'linewidth',2,'Color',[
        randi([0,1]) randi([0,1]) 0]);

    % brach on right side of every iteration
    tree(xnew,ynew,theta+pi/5,length*ratio);

    % brach on left side of every iteration
    tree(xnew,ynew,theta-pi/5,length*ratio);
    axis([-3 3 0 2.5]);
    whitebg([1 1 1]);
    pause(0.0001);

end
end
```

To run the code, save the code above as *tree.m* and type in the following command:

```
tree(0,0,pi/2,1);
```

6 Week 6: Exponential Set

In *Week 3*, we covered the Mandelbrot Set, which is essentially plotting the set of complex numbers for which $f(z) = z^2 + c$ converges when continously reiterated. That produces **Fig. 3**, which is perhaps the most iconic fractal pattern on earth. The same process can be done to any arbitrary function, each function producing a different pattern. This week we will be setting $f(z) = e^x + c$.

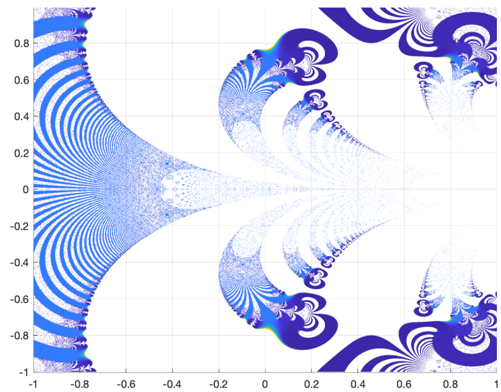

Fig. 6: An Asymmetrical Random Fractal Tree Plotted by the MATLAB Script below.

Matlab Code

```
[x,y]=meshgrid(-1:0.001:1,-1:0.001:1); %% background

 z=exp(x+i*y); %% cosine function initialization
z1=exp(z);
z2=exp(z1);
z3=exp(z2);
z4=exp(z3);
z5=exp(z4);
z6=exp(z5);
z7=exp(z6);
z8=abs(exp(z7));

surf(x,y,z8,'EdgeColor','none');

zlim([0,60]);
caxis([0,60]);
view(2)
```

7 Week 7: Sin(z) Set

Let's say we have a complex number $z = x + iy$. A nice way to yield a fractal pattern from z would be be reiterating a function on z (See *Week 3 and 6*). That is exactly what we're going to be doing this week, with $sin(z)$. The code on page 12 performs a reiteration of $sin(z)$ 12 times, so the final output would be $sin(sin(sin(sin(sin(sin(sin(sin(sin(sin(sin(sin(x + iy))))))))))))$. The following pictures are cool patterns produced at specific x and y values:

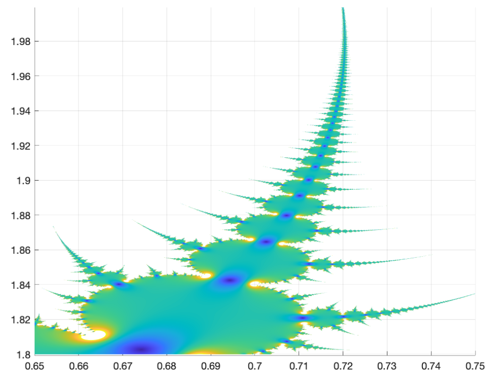

Fig. 7: A $sin(x + iy)$ reiteration, where $x = 0.65 \rightarrow 0.75$ and $y = 1.8 \rightarrow 2$.
"The Scorpion Tail"

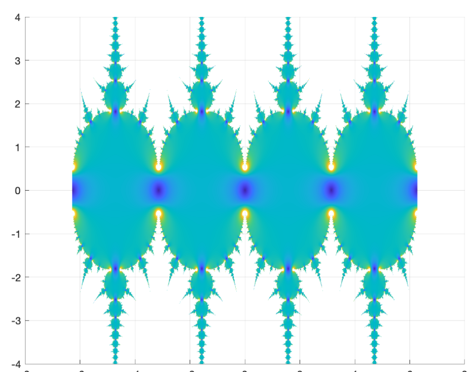

Fig. 8: A $sin(x + iy)$ reiteration, where $x = -2\pi \rightarrow 2\pi$ and $y = -4 \rightarrow 4$.

Matlab Code

```
%% background
[x,y]=meshgrid(0.65:0.0001:0.75,1.8:0.0001:2);

%% sine function initialization
z=sin(x+i*y);
z1=sin(z);
z2=sin(z1);
z3=sin(z2);
z4=sin(z3);
z5=sin(z4);
z6=sin(z5);
z7=sin(z6);
z8=sin(z7);
z9=sin(z8);
z10=sin(z9);


z11=abs(sin(z10));


surf(x,y,z11,'EdgeColor','none');

zlim([0,1]);
caxis([0,1]);
view(2)
```

8 Week 8: Cos(z) Set + Bonus Python Code

Last week we covered the reiteration of $sin(z)$ on the complex plane. We will do the exact same thing this week but on $cos(z)$. No more is needed to be said, so without further ado, look at those amazing patterns

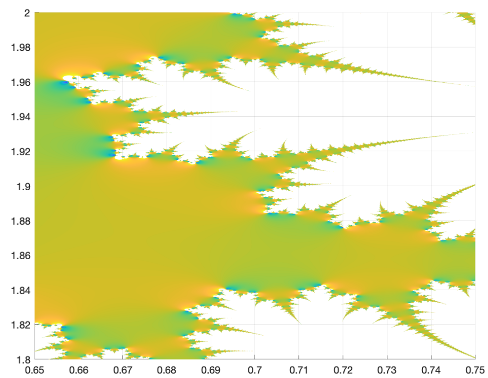

Fig. 9: A $cos(x + iy)$ reiteration, where $x = 0.65 \rightarrow 0.75$ and $y = 1.8 \rightarrow 2$.

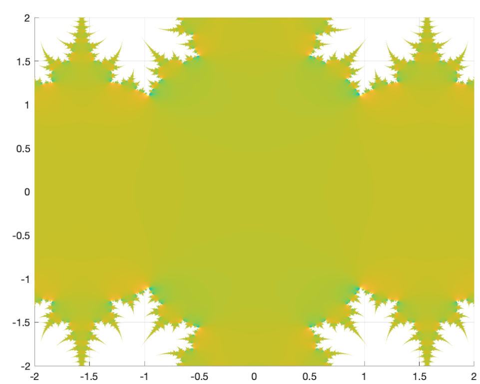

Fig. 10: A $cos(x + iy)$ reiteration, where $x = -2 \rightarrow 2$ and $y = -2 \rightarrow 2$.

As you can see, from -2 to 2 for both x and y, an amazing symmetrical shape is produced. Try out different intervals for yourslef!

Matlab Code

```
%% background
[x,y]=meshgrid(-2:0.001:2,-2:0.001:2);

%% cose function initialization
z=cos(x+i*y);
z1=cos(z);
z2=cos(z1);
z3=cos(z2);
z4=cos(z3);
z5=cos(z4);
z6=cos(z5);
z7=cos(z6);
z8=cos(z7);
z9=cos(z8);
z10=cos(z9);


z11=abs(cos(z10));


surf(x,y,z11,'EdgeColor','none');

zlim([0,1]);
caxis([0,1]);
view(2)
```

Bonus

In *Week 3*, many of you asked for a Mandelbrot Set alternative for Python, so our friend, Lukas Kretschmann made an efficient alternative for Python which you can find on https://github.com/LukasKretschmann/Mandelbrotmenge. The code goes as following:

```python
      import numpy as np
import matplotlib.pyplot as plt
from numpy import newaxis
import time

plt.rcParams["figure.figsize"] = 64, 64

#Iteration count
def Iteration_count(c,threshold):
    z = complex(0, 0)
    for iteration in range(threshold):
        z = z**2 + c
        if abs(z) > 4:
            break
    return iteration

#plot
def mandelbrot(threshold, img_size=1000):
    r_Axis = np.linspace(-2,1, img_size)
    i_Axis = np.linspace(-1.5,1.5, img_size)
    plot = np.empty((r_Axis.size, i_Axis.size))

    #color for plot
    for index_x,ix in enumerate(r_Axis):
        for index_y,iy in enumerate(i_Axis):
            plot[index_x, index_y] = Iteration_count(complex(ix, iy
                                      ), threshold)
    return plot

def save_img(arr):
    plt.imshow(arr.T, interpolation="nearest")
    plt.savefig("mandel.jpg", dpi=300)

t0 = time.time()
out = mandelbrot(50, 1000)
t1 = time.time()
t1-t0

plt.imshow(out.T)
plt.show()

save_img(out)
```

And the output is:

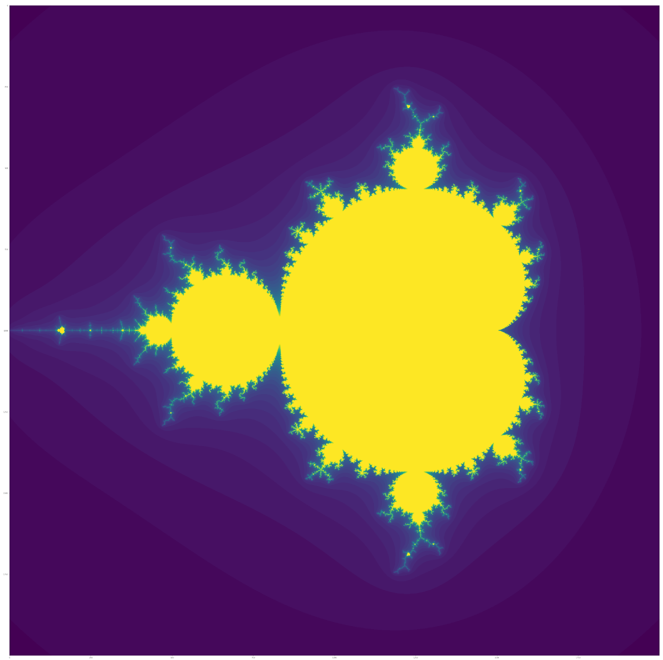

Fig. 11: Mandelbrot set produced by python algorithm

9 Week 9: Wagh's Accidental Fractal

This week, we were trying to generate a Koch Snowflake pattern. However, when Yogesh was writing the script, he accidentally made a mistake and the algorithm generated a fractal pattern which we've never heard of. That's why we're calling it the Wagh Pattern (Yogesh's Last name is Wagh). The following is the output:

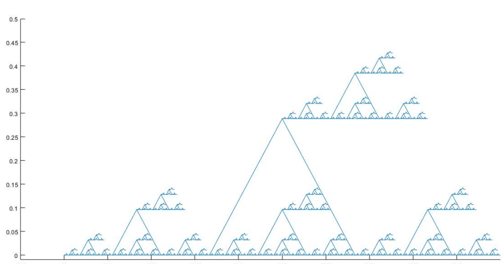

Fig. 12: Yogesh's Accidental Fractal Pattern

Matlab Code

```
function def_abc=Koch(x,y,xe,ye,len)

lenth=sqrt((x-xe)*(x-xe)+(y-ye)*(y-ye));
theta=pi/3;
x1=x+len/3;
y1=y;

x2=x1+(len*cos(theta))/3;
y2=y1+(len*sin(theta))/3;

x3=x2+(len*cos(theta))/3;
y3=y2-(len*sin(theta))/3;

x4=x3+len/3;
y4=y3;

if lenth > 0.005

    line([x,x1],[y,y1]);
    line([x1,x2],[y1,y2]);
    line([x2,x3],[y2,y3]);
    line([x3,x4],[y3,y4]);

    Koch(x,y,x1,y2,len/3);
    Koch(x1,y1,x2,y2,len/3);
    Koch(x2,y2,x3,y3,len/3);
    Koch(x3,y3,x4,y4,len/3);

    axis([-0.1 1 -0.01 0.5]);
    whitebg([1 1 1]);
    pause(0.001);

end
end
```

To run it, type the following conditions in the MATLAB command line:

```
Koch(0,0,1/3,0,1);
```

Fractal Joke:

What does letter B from Benoit B. Mandelbrot stands for?

Benoit B. Mandelbrot