Matlab Experiment 12 Report: Iteration II: Fractal

Yikun Zhang*
SID: 14336275
Major: Mathematics and Applied Mathematics

Abstract

Fractal, a concept rooted in Mathematics for centuries, did not trigger mathematician research interests until the last century. In this experiment we are supposed to provide our understanding of fractal geometry. Our journey begins with some preliminary implementation of basic fractals via iterated programming. Knowing how Koch curve, Minkowski sausage, Hilbert curve, and Sierpinski triangle look like, we scrutinize their fractal dimensions. Additionally, the well-known continuous but nowhere differentiable Weierstrass function is visualized. In the consecutive part we explore other extending branches of fractals like complex function iterations and IFSs, where we obtain other famous fractals like Julia set and Mandelbrot set. Thanks to the convergency of IFSs, we figure out another method to visualize Sierpinski triangle and dragon curve apart from purely iterated programming.

Contents

*School of Mathematics, Sun Yat-sen University

1 Introduction and Purpose

"I find the ideas in the fractals, both as a body of knowledge and as a metaphor, an incredibly important way of looking at the world." Vice President of Yale University and Nobel Laureate Al Gore, *New York Times*, Wednesday, June 21, 2000, discussing some of the "big think" questions that intrigue him.[1] Indeed, the idea of fractals can be utilized to explain some counter-intuitive notions, as Benoit Mandelbrot did in his 1967 paper[2] about self-similarity, fractional dimensions, and a coastline's measured length.

In Mathematics a fractal is an abstract object used to describe and simulate naturally occurring objects. Artificially created fractals commonly exhibit similar patterns at increasingly small scales. It is also known as expanding symmetry or evolving symmetry. If the replication is exactly the same at every scale, it is called a self-similar pattern.[3] The simplest fractal would be the Koch snowflake, which begins with an equilateral triangle and then replaces the middle third of every line segment with a pair of line segments that form an equilateral bump.[3] Besides those self-similar figures, the well-known Weierstrass example of a everywhere continuous but nowhere differentiable function also falls into the category of fractals. Motivated by Weierstrass, Georg Cantor published a famous example known as Cantor set, which embraces the same cardinality as the real line, i.e., $\aleph$, but with Lebesgue measure 0. These examples not only enrich the variety of fractals but also serve as the impetus of the development of other branches of Mathematics.

In this experiment, we aim to make use of computer programming in Matlab to simulate some specific examples of fractals such as Koch snowflake, Sierpinski triangle, and so on. Furthermore, we will also discuss the fractal dimensions of these interesting figures and formulate their areas and circumferences. In addition, reproducing the Weierstrass function approximately is among the range of our experiments. After that, we are supposed to explore the iterations of complex functions, leading to attractors and repellors through the Julia and Mandelbrot set. Last but not least, the introduction of IFS (Iteration Function System) will be unfolding in a "chaos" but convergent iterations.

All our experiments are based on computer programming on the platform of Matlab 2016a, which may approximately give us the overview of fractals. Essentially, the main theme of our experiments will follow the Chapter 12 of the book "Mathematical Experiments" written by Shangzhi Li et.al[4].

2 Methods and Results

2.1 Generated Unit (Beginning of Fractals)

In this section we are going to visualize some famous fractals, where some iterative programming skills are heavily relied on.

First, in order to sketch the configurations of the Koch curve, we have to compute the coordinates of end points and turning points of each generated unit, given the coordinates of the end points of the original line. Let (x_1, y_1) and (x_2, y_2) be the coordinates of two end points of an original line. By the definition of the Koch curve, there will be five points of the resulting generating unit, whose coordinates are $P_1(x_1, y_1), P_2(\frac{2x_1+x_2}{3}, \frac{2y_1+y_2}{3}), P_3(\frac{x_1+x_2}{2} + \frac{\sqrt{3}(y_1-y_2)}{6}, \frac{y_1+y_2}{2} + \frac{\sqrt{3}(x_2-x_1)}{6}), P_4(\frac{x_1+2x_2}{3}, \frac{y_1+2y_2}{3}), P_5(x_2, y_2)$. Here when calculating the coordinate of P_3, we use the fact that the length of the equilateral triangle $\frac{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}{3}$ is equal to $\frac{2}{\sqrt{3}}$ times the distance between the mid point of the line and P_3. By setting the initial end points to be $(-3, 0), (3, 0)$, we obtain a Koch curve. See Figure 1 for details.

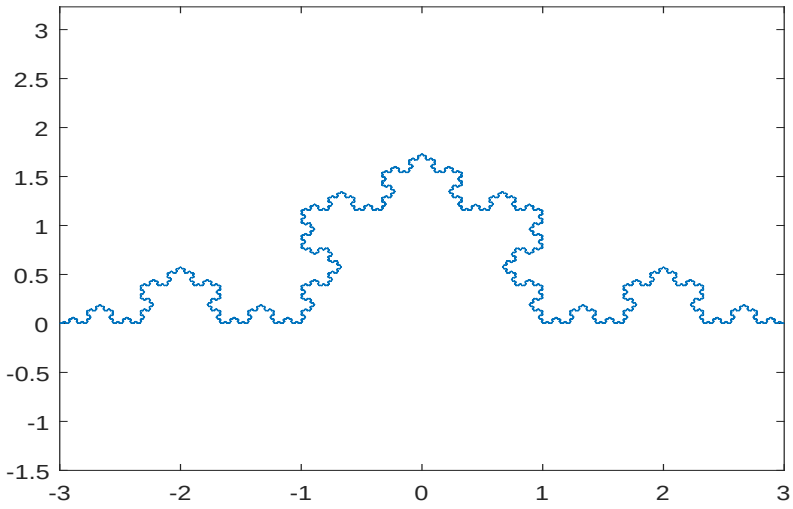

Figure 1: A Koch Curve (Iterative times: $N = 10$)

From Figure 1, it turns out that the length of the curve grows as the number of iterative times increases. In reality, we will mathematically prove that the length of a fractal curve tends to infinity while its area stabilizes to a constant number. Such a peculiar property is one of the interesting phenomena brought by fractals.

Second, to construct a Sierpinski triangle, we again resort to iterations. Here we sketch the lines of the triangle following a small trick: we begin the line on a vertex of the original triangle and extend it to the mid point of an edge; then outline the edges of the smaller triangle whose vertexes are three mid points of the original triangle; when returning to the mid point of the original edge, we head to the next vertex of the triangle. By iterating this algorithm, the Sierpinski triangle is unfolding. See Figure 2 for details.

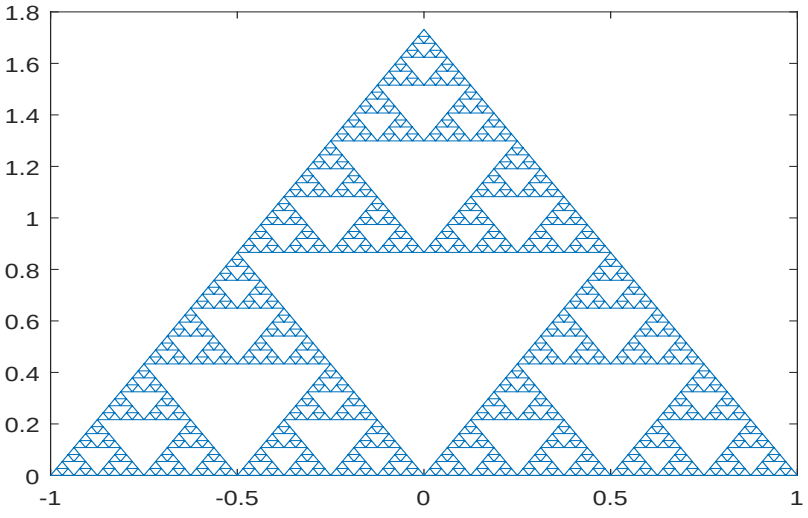

Figure 2: A Sierpinski triangle (Iterative times: $N = 6$)

When zooming in the preceding fractals we are exposed to no new patterns. The reason lies in the fact that such fractals are self-similar. Thanks to self-similarity, Benoit often said that a fractal is understood just as much through what has been removed as through what remains.[5]

Given an equilateral triangle, we apply the generated algorithm to its edges and obtain a Koch snowflake as the number of iterative times tends to infinity. See Figure 3 for details.

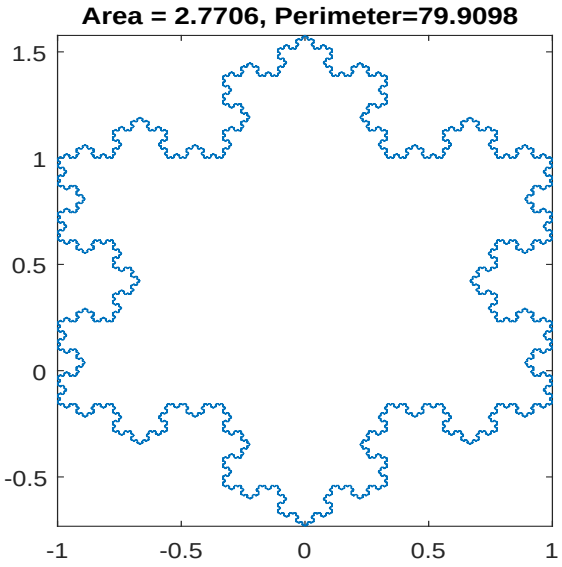

Figure 3: A Koch Snowflake (Iterative times: $N = 10$)

Assume that the perimeter of the original triangle is c and the area is s. Then by direct computations, after iterating n times, the perimeter of the Koch snowflake becomes

$$c[1 + \frac{1}{3} + \frac{1}{3} \cdot \frac{4}{3} + \cdots + \frac{1}{3}(\frac{4}{3})^{n-2}] = c(\frac{4}{3})^{n-1} \to \infty, \text{ as } n \to \infty,$$

while its area is

$$s[1 + 3 \cdot \frac{1}{9} + \frac{1}{3} \cdot \frac{4}{9} + \cdots + \frac{1}{3} \cdot (\frac{4}{9})^{n-1}] = s[\frac{8}{5} - \frac{3}{5}(\frac{4}{9})^{n}] \to \frac{8}{5}s, \text{ as } n \to \infty.$$

Obviously, every point on the Koch snowflake is not smooth and thus it is nowhere differentiable. Many other fractals embrace the same property as the Koch snowflake. Based on these results, it seems that we cannot simply define the dimension of the Koch snowflake as the usual definition in the Euclidean space. However, when it comes to the dimension of a given space, we have various ways to define it. One may calculate the dimension by figuring out the linearly independent vectors that span this space. On the other hand, we can decompose the figure into some self-similar sub-figures and magnify those sub-figures by a factor of $\mathbf{N}$ to yield the original figure. As an example, the square may be broken into $\mathbf{N}^2$ self-similar copies of itself, each of which must be magnified by a factor of $\mathbf{N}$ to yield the original figure. Now we see an alternative way to specify the dimension of a self-similar object: The dimension is simply the exponent of the number of self-similar pieces with magnification factor $\mathbf{N}$ into which the figure may be broken.[6] Therefore, the dimension of a self-similar fractal can be defined by

$$\text{fractal dimension} = \frac{log(\text{number of self-similar pieces})}{log(\text{magnification factor})}.$$

The fractal dimension is a ratio providing a statistical index of complexity comparing how detail in a fractal pattern changes with the scale at which it is measured. It has also been characterized as

a measure of the space-filling capacity of a pattern that tells how a fractal scales differently from the space it is embedded in.[7] Dy this definition, the dimension of the Koch snowflake is $d_1 = \frac{\log 4}{\log 3}$ and the dimension of the Sierpinski triangle is $d_2 = \frac{\log 3}{\log 2}$, where we exclude the middle triangles. Since the dimension of a line is 1 and the dimension of a plane is 2, the dimension of the Sierpinski triangle sits between these two sets. The fact may in part explain why the area of a Sierpinski triangle is zero in Lebesgue measure.

To sketch out the so-called Minkowski sausage, we again record the turning points of a sub-pattern iteratively. Given the complex representations of two end points of a line, z_1, z_2, we directly compute the complex coordinates of turning points, z_1, $\frac{3z_1+z_2}{4}$, $\frac{3z_1+z_2}{4} + i(\frac{z_2-z_1}{4})$, $\frac{z_1+z_2}{2} + i(\frac{z_2-z_1}{4})$, $\frac{z_1+z_2}{2}$, $\frac{z_1+z_2}{2} - i(\frac{z_2-z_1}{4})$, $\frac{z_1+3z_2}{4} - i(\frac{z_2-z_1}{4})$, $\frac{z_1+3z_2}{4}$, z_2. With these iterative formulas, we obtain a Minkowski sausage in Figure 4.

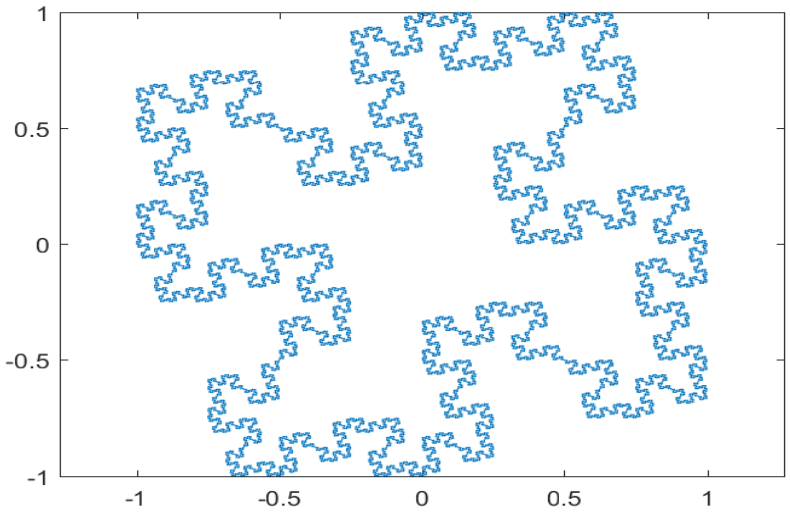

Figure 4: A Minkowski Sausage (Iterative times: $N = 6$)

By definition, the dimension of the Minkowski sausage is $d_3 = \frac{\log 8}{\log 4} = \frac{3}{2}$.

In a similar fashion the Hilbert curve is obtained by deforming the original line into some small segments which are perpendicular to each other. As the number of iterations increases, the Hilbert curve will spread out the entire square. Because of this space-filling property, the Hausdorff dimension of the Hilbert curve is 2. More precisely, its graph is a compact set homeomorphic to the closed unit interval, with Hausdorff dimension 2. The Euclidean length of n^{th} approximation of the Hilbert curve is $2^n - \frac{1}{2^n}$.[8] See Figure 5 for details.

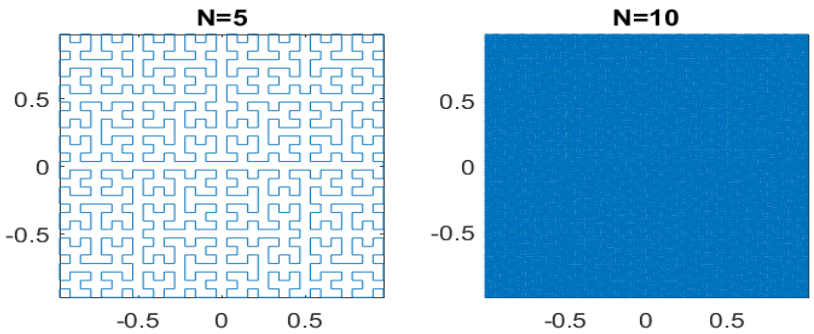

Figure 5: Hilbert curves (Iterative times: $N = 5$ and $N = 10$)

Nevertheless, one must keep in mind that the relationship of an increasing fractal dimension with space-filling might be taken to mean fractal dimensions measure density, but that is not so; the two are not strictly correlated. Instead, a fractal dimension measures complexity, a concept related to certain key features of fractals: self-similarity and detail or irregularity.[7]

Apart from the previous examples, fractals have a long history in Mathematical Analysis, dating from the well-known Weierstrass function,

$$W(x) = \sum_{k=1}^{\infty} \lambda^{(s-2)k} sin(\lambda^k x), \text{ where } \lambda > 1, 1 < s < 2.$$

To visualize the graph of the Weierstrass function via Matlab, we set $\lambda = 3$ and $s = 1.25, 1.5, 1.75$ respectively. Meanwhile, the summation is up to $N = 50$ because of time complexity in dealing with symbolic operations. See Figure 6 for details.

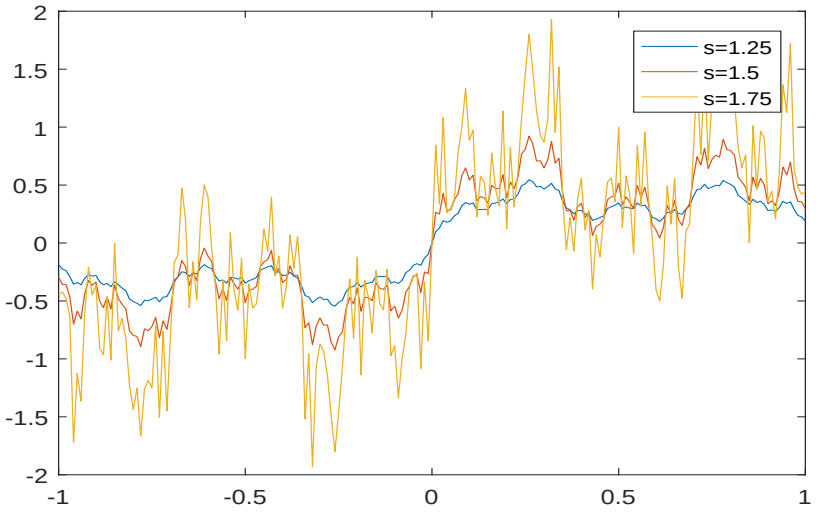

Figure 6: Weierstrass Functions

From Figure 6, the fluctuation of the Weierstrass function will be more violent and the smoothness of the graph of the function deteriorates as the value of s increases.

2.2 Complex Dynamics

Iterations with complex functions have long-lasting connections with fractals. The most attractive examples that bridge the gap between analytic mappings and fractals are the Mandelbrot set and Julia set, which are very popular complex dynamical system given by the family of complex quadratic polynomials.

Consider a complex quadratic map

$$f_c(z) = z^2 + c,$$

where c is a complex parameter. The Julia set are the initial points such that the iteration of f_c are convergent, i.e.,

$$J(f_c) = \{z \in \mathbb{C} : \forall n \in \mathbb{N}, |f_c^n(z)| \leq 2\}.$$

Alternatively, given an initial value of z_0, says 0, the Mandelbrot set are consisted of the possible values of complex parameter c such that the absolute values of $f_c^n(z)$ is bounded as n tends to infinity, i.e.,

$$M = \{c \in \mathbb{C} : \exists s \in \mathbb{R}, \forall n \in \mathbb{N}, |f_c^n(0)| \leq s\}.$$

In practice it is sufficient to simply take $s = 2$.

In order to implement complex dynamics via computer programming, we let $f_c^k(z) = x_k + iy_k, c = p + iq$ and rewrite the iterative formula into

$$x_{k+1} = x_k^2 - y_k^2 + p, y_{k+1} = 2x_k y_k + q, k = 0, 1, 2, ...$$

Denoting $x_k^2 + y_k^2$ by r_k, the Julia set becomes those initial points (x_0, y_0) such that the sequence $\{r_k\}_{k=0}^{\infty}$ is bounded, while the Mandelbrot set is the parameters (p, q) that make the same condition happened.

With an initial bounded value $M \leq 2$, we divide the rectangular region $R = \{(p, q)| - M \leq p, q \leq M\}$ into $a \times b$ grids and use the grid points as the initial values. If we denote the number of iterative times that makes the preceding iteration with the (i, j) grid point exceed the bounded region by k_{ij}, then we set the pixel of this point to be $log(k_{i,j})$ and obtain Julia sets given different parameters. See code in the Appendix and Figure 7 for details. In a similar fashion, we draw the Mandelbrot set in Figure 8.

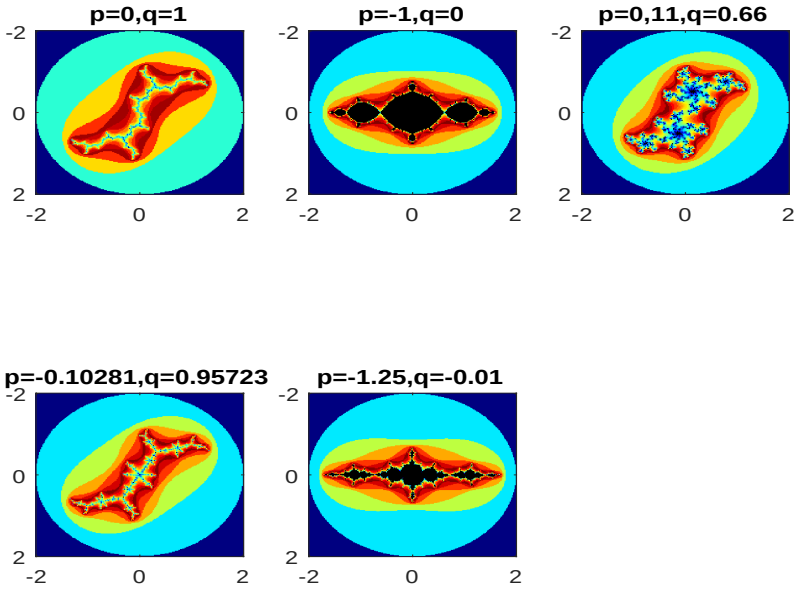

Figure 7: Julia sets with different parameters

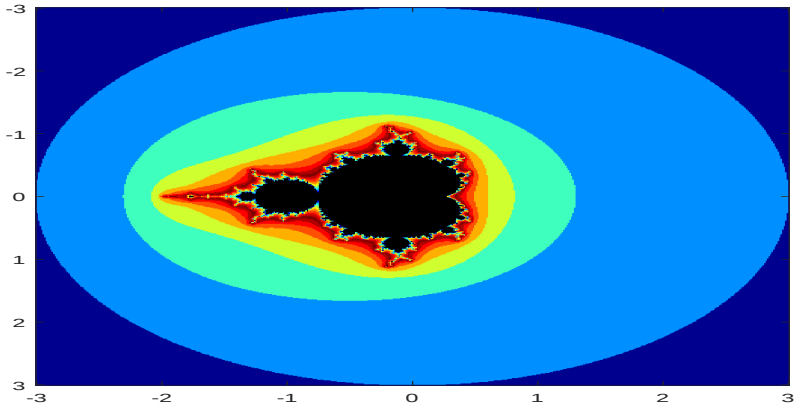

Figure 8: Mandelbrot set

Since the Julia set and Mandelbrot set embrace some intrinsic connections, we are eager to inspect how Julia sets look like when their parameters are selected from different parts of the Mandelbrot set. We utilize the *ginput()* function in Matlab to manually select points from Figure 8. See Figure 9 for details.

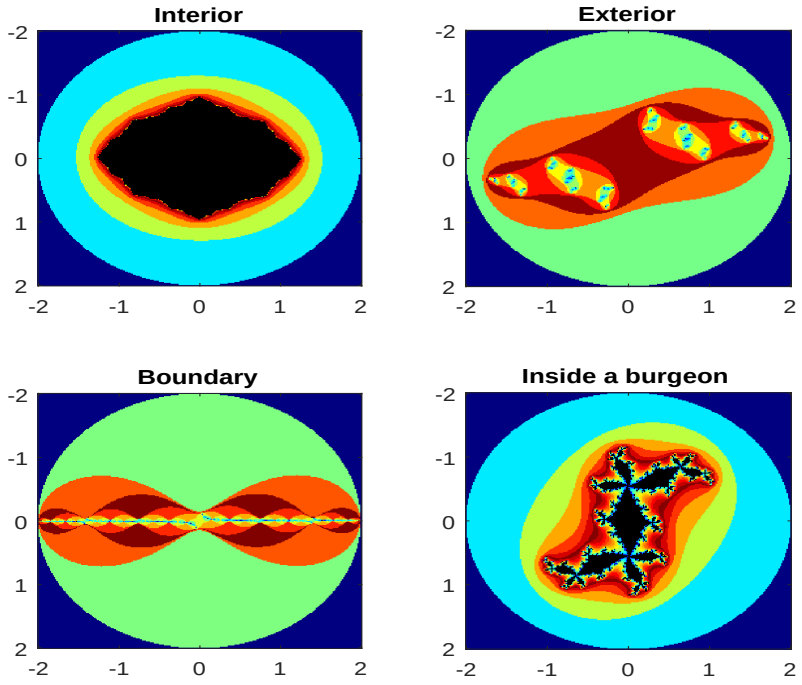

Figure 9: Julia sets with different parameters chosen from Mandelbrot sets

Julia sets and Mandelbrot sets can also be generalized to high-dimensional cases, where $z_{k+1} = z_k^n + c, k = 0, 1, ...$ Therefore, we utilize the complex numbers to represent the points in this case and visualize the third, fourth, and fifth order of Mandelbrot sets. See Figure 10 for details.

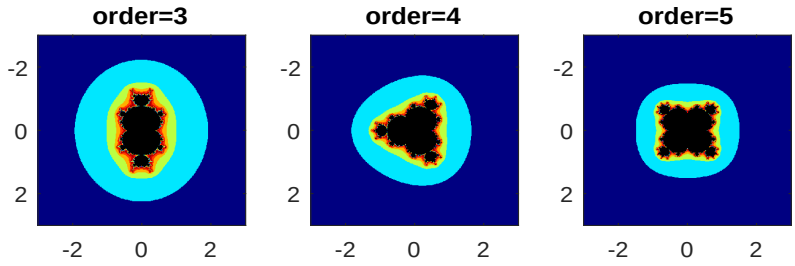

Figure 10: The third, fourth, and fifth order of Mandelbrot sets

Then we again manually select some initial points on the previous Mandelbrot sets and plot their corresponding Julia sets. See Figure 11 for details.

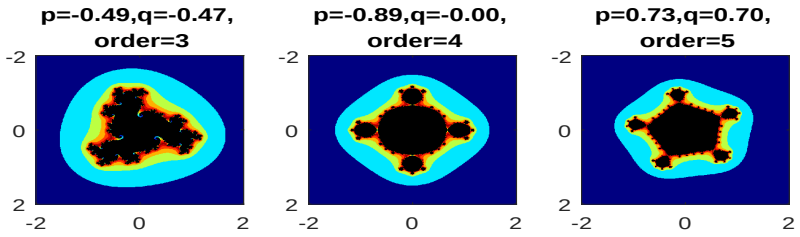

Figure 11: The third, fourth, and fifth order of Julia sets

2.3 Iterated Function Systems

In mathematics, iterated function systems (IFSs) are a method of constructing fractals; the resulting fractals are often self-similar.[9] We launch our discussion of this compelling method by **chaos game**. Starting with an initial random point, the sequence is a given fraction of the distance between the previous point and one of the vertices of a predefined polygon; the vertex is chosen at random in each iteration. Repeating this iterative process a large number of times, selecting the vertex at random on each iteration, and throwing out the first few points in the sequence, will often (but not always) produce a fractal shape.[10]

Now we present an elementary chaos game. Given three non-collinear points A, B, C, three preassigned probabilities p_A, p_B, p_C, and an initial point Z_0, Z_{n+1} has p_A probability to be $\frac{Z_n+A}{2}$, p_B probability to be $\frac{Z_n+B}{2}$, and p_C probability to be $\frac{Z_n+C}{2}$. Then as the number of iterative times tends to infinity, it turns out that the resulting points will stabilize to a limit figure called attractors. Now, besides visualizing the ultimate limit figure of this chaos game, we investigation whether the choices of probabilities p_A, p_B, p_C have some effects on attractors. See Figure 12 for details.

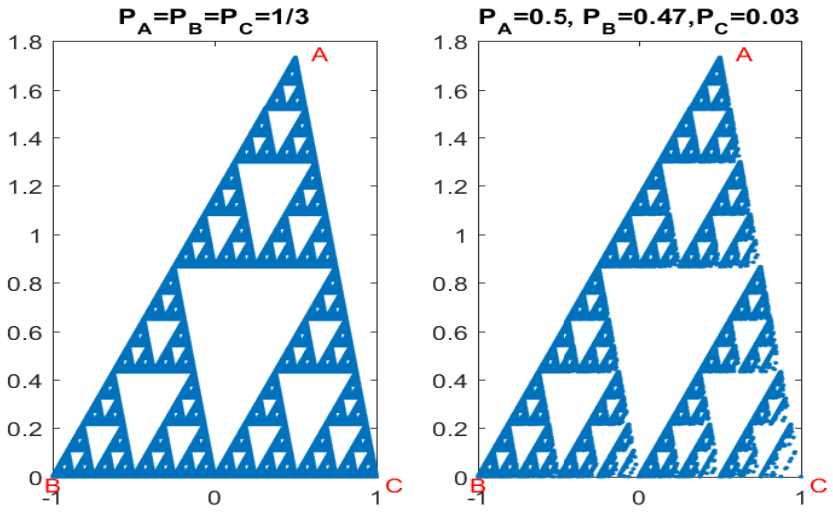

Figure 12: Attractors of the chaos game given different probabilities

From this comparative graphes, one can easily notice that the surrounding configuration of the vertex with smaller probability is more obscure given the identical iterative times. Hence the higher the probability is, the more attractive the attractor will be in an IFS.

Consider the affine transformation $\omega_1(Z) = sZ + 1, \omega_2(Z) = sZ - 1$ with the identical probability $p_1 = p_2 = \frac{1}{2}$, where s, Z are all complex numbers. By setting $s = 0.5 + 0.5i$, we visualize the graph of attractors of this IFS. See Figure 13 for details.

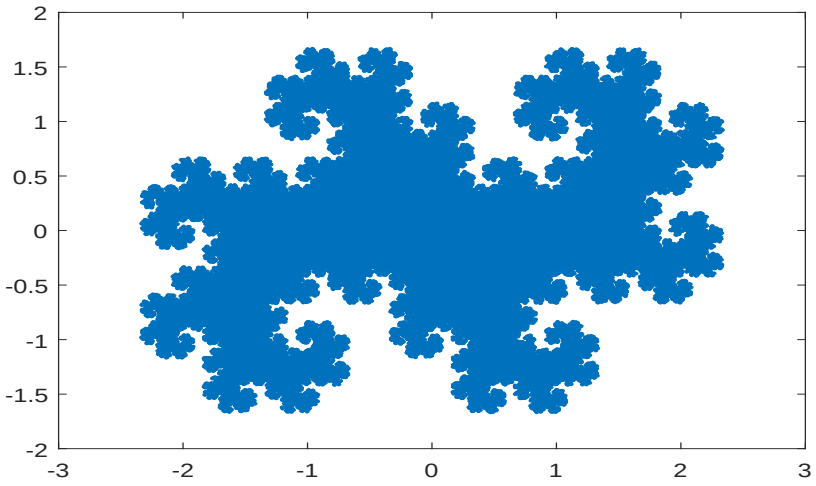

Figure 13: Attractors' configuration of the affine transformation $\omega_1(Z) = sZ + 1, \omega_2(Z) = sZ - 1$

What amazes us is that the resulting figure is essentially identical to the so-call dragon curve, a fractal constructing by self-similar patterns. This example vividly indicate the intrinsic connections between fractals and IFSs.

In addition, we consider a more general transformation $\omega_1(Z) = \sqrt{Z - C}, \omega_2(Z) = \sqrt{Z + C}$. In this case, the graphes of attractors are sparser. See Figure 14 for details.

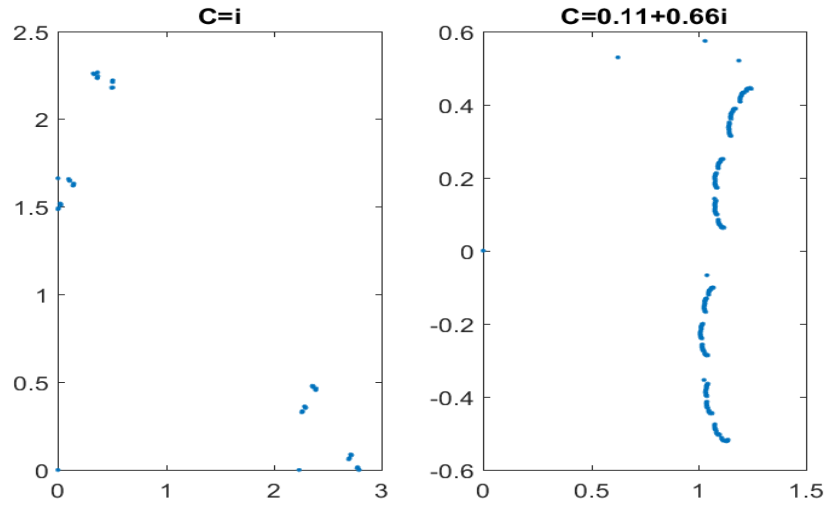

Figure 14: Attractors' configuration of the affine transformation $\omega_1(Z) = \sqrt{Z - C}, \omega_2(Z) = \sqrt{Z + C}$

3 Conclusion and Discussion

In this experiment we explore one of the most mysterious realm of Mathematics, fractal geometry. Our journey begins with some preliminary implementation of basic fractals via iterated programming. Knowing how Koch curve, Minkowski sausage, Hilbert curve, and Sierpinski triangle look like, we scrutinize their fractal dimensions. Additionally, the well-known continuous but nowhere differentiable Weierstrass function is visualized. In the consecutive part we explore other extending branches of fractals like complex function iterations and IFSs, where we obtain other famous fractals like Julia set and Mandelbrot set. Thanks to the convergency of IFSs, we figure out another method to visualize Sierpinski triangle and dragon curve apart from purely iterated programming. However, the magic of fractals is far more than what we have presented here. All frantic mathematicians can dive more deeply into such an attractive field.

4 Reference

[1] Michael Frame, Benoit Mandelbrot, and Nial Neger ***Fractal Geometry.*** URL: `http://users.math.yale.edu/public_html/People/frame/Fractals/` Retrieved 30 December, 2017.

[2] Benoit Mandelbrot (1967) ***How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension.*** Science. 156 (3775): 636-638.

[3] Wikipedia ***Fractal.*** URL: `https://en.wikipedia.org/wiki/Fractal` Retrieved 30 December, 2017.

[4] Shangzhi Li, Falai Chen, Yaohua Wu, and Yunhua Zhang (1999) ***Mathematical Experiments.*** Textbook Seris for 21st Century, Higher Education Press.

[5] Michael Frame ***Benoit Mandelbrot.*** URL: `http://users.math.yale.edu/public_html/People/frame/Fractals/` Retrieved 30 December, 2017.

[6] Robert L. Devaney ***Fractal Dimension.*** Department of Mathematics, Boston University URL: `http://math.bu.edu/DYSYS/chaos-game/node6.html` Retrieved 30 December, 2017.

[7] Wikipedia ***Fractal dimension.*** URL: `https://en.wikipedia.org/wiki/Fractal_dimension` Retrieved 30 December, 2017.

[8] Wikipedia ***Hilbert curve.*** URL: `https://en.wikipedia.org/wiki/Hilbert_curve` Retrieved 30 December, 2017.

[9] Wikipedia ***Iterated function system.*** URL: `https://en.wikipedia.org/wiki/Iterated_function_system` Retrieved 30 December, 2017.

[10] Wikipedia ***Chaos game.*** URL: `https://en.wikipedia.org/wiki/Chaos_game` Retrieved 30 December, 2017.

5 Appendix

```
% Exercise 1 (Koch)
N=10;Koch=cell(N,1);
Koch{1,1}=[-3  0;3  0];
for i=1:N-1
    points=Koch{i,1};
    n=size(points);
    new_p=zeros(4*n(1)-3,2);
    new_p(1,:)=points(1,:);
    for j=1:n(1)-1
        new_p(4*j-2,:)=(2*points(j,:)+points(j+1,:))/3;
        new_p(4*j,:)=(points(j,:)+2*points(j+1,:))/3;
        new_p(4*j-1,1)=(points(j,1)+points(j+1,1))/2+(points(j,2)-points(j+1,2))*sqr
        new_p(4*j-1,2)=(points(j,2)+points(j+1,2))/2+(points(j+1,1)-points(j,1))*sqr
```

```
        new_p(4*j+1,:)=points(j+1,:);
    end
    Koch{i+1,1}=new_p;
end
re_p=Koch{end,1};
h=figure;
plot(re_p(:,1),re_p(:,2))
axis equal
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex11','-dpdf','-fillpage')

% Exercise 1 (Sierpinski Triangle)
function z = triangle(initial_p, n)

% Constants
a = initial_p(2);

% Generate point sequence
z = [initial_p(1); initial_p(3)];
for k = 1:n
    z = [z+initial_p(1); z+a; z+initial_p(3)]/2;
end

% Close triangle
z = [z; a; initial_p(1)];

N=6;vertex=[-1;1;sqrt(-3)];
h=figure;
plot(triangle(vertex,N))
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex12','-dpdf','-fillpage')

% Exercise 2
N=10;Koch=cell(N,1);
Koch{1,1}=[-1 1;1 1;0 1-sqrt(3);-1 1];
for i=1:N-1
    points=Koch{i,1};
    n=size(points);
    new_p=zeros(4*n(1)-3,2);
    new_p(1,:)=points(1,:);
    for j=1:n(1)-1
        new_p(4*j-2,:)=(2*points(j,:)+points(j+1,:))/3;
        new_p(4*j,:)=(points(j,:)+2*points(j+1,:))/3;
        new_p(4*j-1,1)=(points(j,1)+points(j+1,1))/2+(points(j,2)-points(j+1,2))*sqr
        new_p(4*j-1,2)=(points(j,2)+points(j+1,2))/2+(points(j+1,1)-points(j,1))*sqr
        new_p(4*j+1,:)=points(j+1,:);
    end
    Koch{i+1,1}=new_p;
end
re_p=Koch{end,1};
A=polyarea(re_p(:,1),re_p(:,2));
peri=0;
for i=1:length(re_p)-1
    peri=peri+sqrt(sum((re_p(i,:)-re_p(i+1,:)).^2));
end
```

```
h=figure;
plot(re_p(:,1),re_p(:,2))
title(['Area = ' num2str(A),', Perimeter=' num2str(peri)])
axis image
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex21','-dpdf','-fillpage')

% Minkowski
N=6;Mink=cell(N,1);
Mink{1,1}=[-1;-i;1;i;-1];
for i=1:N-1
    z=Mink{i,1};
    n=length(z);
    new_point=zeros(8*(n-1)+1,1);
    new_point(1)=z(1);
    for j=1:n-1
        theta=atan(imag(z(j+1)-z(j))/real(z(j+1)-z(j)));
        new_point(8*j-6)=3/4*z(j)+1/4*z(j+1);
        new_point(8*j-5)=3/4*z(j)+1/4*z(j+1)+1i*(z(j+1)-z(j))/4;
        new_point(8*j-4)=1/2*z(j)+1/2*z(j+1)+1i*(z(j+1)-z(j))/4;
        new_point(8*j-3)=1/2*z(j)+1/2*z(j+1);
        new_point(8*j-2)=1/2*z(j)+1/2*z(j+1)-1i*(z(j+1)-z(j))/4;
        new_point(8*j-1)=(z(j)+3*z(j+1))/4-1i*(z(j+1)-z(j))/4;
        new_point(8*j)=(z(j)+3*z(j+1))/4;
        new_point(8*j+1)=z(j+1);
    end
    Mink{i+1,1}=new_point;
end
re_mink=Mink{end,1};
h=figure;
plot(re_mink)
axis equal
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex31','-dpdf','-fillpage')

% Exercise 3
function z = hilbert(n)

% Constants
a = 1 + 1i;
b = 1 - 1i;

% Generate point sequence
z = 0;
for k = 1:n
    w = 1i*conj(z);
    z = [w-a; z-b; z+a; b-w]/2;
end

h=figure;
subplot(1,2,1),plot(hilbert(5))
title('N=5')
axis equal
axis image
subplot(1,2,2),plot(hilbert(10))
```

```
title('N=10')
axis equal
axis image
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex32','-dpdf','-fillpage')

% Exercise 4
h=figure;
s1=[1.25;1.5;1.75];N=50;lambda=3;
syms s x k
W_x = symsum((lambda.^((s-2).*k)).*sin(lambda.^k*x),k,1,N);
t=-1:0.01:1;
f=subs(subs(W_x,s,s1),x,t);
plot(t,f)
legend('s=1.25','s=1.5','s=1.75')
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex4','-dpdf','-fillpage')

% Exercise 5
function julia(arg1,arg2,p,q)

N=arg1;
M=max(2,sqrt(p^2+q^2));
x1=linspace(-M,M,arg2);
y1=linspace(-M,M,arg2);
[xtrans,ytrans]=meshgrid(x1,y1);
t=ones(size(xtrans));
n=size(xtrans);
for i=1:n(1)
    for j=1:n(2)
        x=xtrans(i,j);
        y=ytrans(i,j);
        for k=1:N
            if x^2+y^2>M^2
                break;
            end
            temp=x;
            x=x^2-y^2+p;
            y=2*temp*y+q;
        end
        t(i,j)=k;
    end
end
t=log(t);

fig = gcf;
fig.Position = [90 90 600 600];
imagesc(x1,y1,t);
colormap( [jet();flipud( jet() );0 0 0] );
axis('equal','image');

h=figure;
subplot(2,3,1),julia(50,200,0,1)
title('p=0,q=1')
subplot(2,3,2),julia(50,200,-1,0)
```

```
title('p=-1,q=0')
subplot(2,3,3),julia(50,200,0.11,0.66)
title('p=0,11,q=0.66')
subplot(2,3,4),julia(50,200,-0.10281,0.95723)
title('p=-0.10281,q=0.95723')
subplot(2,3,5),julia(50,250,-1.25,-0.01)
title('p=-1.25,q=-0.01')
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex5','-dpdf','-fillpage')

% Exercise 6
function mandelbrot(times,inter_leng)

M=3;
pin=linspace(-M,M,inter_leng);
qin=linspace(-M,M,inter_leng);
[ptrans,qtrans]=meshgrid(pin,qin);
count=ones(size(ptrans));
n=size(ptrans);
for i=1:n(1)
    for j=1:n(1)
        p=ptrans(i,j);q=qtrans(i,j);
        x=0;y=0;
        for k=1:times
            if x^2+y^2>M^2
                break;
            end
            temp=x;
            x=x^2-y^2+p;
            y=2*temp*y+q;
        end
        count(i,j)=k;
    end
end
count=log(count);

fig = gcf;
fig.Position = [90 90 600 600];
imagesc(pin, qin, count);
colormap( [jet();flipud( jet() );0 0 0] );
axis('square','equal','image');

mandelbrot(50,500)
miu=ginput(4);
h=figure;
subplot(2,2,1),julia(50,250,miu(1,1),miu(1,2))
title('Interior')
subplot(2,2,2),julia(50,250,miu(2,1),miu(2,2))
title('Exterior')
subplot(2,2,3),julia(50,250,miu(3,1),miu(3,2))
title('Boundary')
subplot(2,2,4),julia(50,250,miu(4,1),miu(4,2))
title('Inside a burgeon')
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex62','-dpdf','-fillpage')
```

```
% Exercise 7
function julia_N(times,inter_leng,p,q,N)

M=max(2,sqrt(p^2+q^2));
x1=linspace(-M,M,inter_leng);
y1=linspace(-M,M,inter_leng);
[xtrans,ytrans]=meshgrid(x1,y1);
count=ones(size(xtrans));
z=xtrans+1i*ytrans;

for n=1:times
    z=z.^N+p+1i*q;
    inside=abs(z)<=M^2;
    count=count+inside;
end
count=log(count);

fig = gcf;
fig.Position = [90 90 600 600];
imagesc(x1,y1,count);
colormap( [jet();flipud( jet() );0 0 0] );
axis('equal','image');

function mandelbrot_N(times,inter_leng,N)

M=3;
pin=linspace(-M,M,inter_leng);
qin=linspace(-M,M,inter_leng);
[ptrans,qtrans]=meshgrid(pin,qin);
count=ones(size(ptrans));
z0=ptrans+1i*qtrans;

z=z0;
for n=1:times
    z=z.^N+z0;
    inside=abs(z)<=M^2;
    count=count+inside;
end
count=log(count);

fig = gcf;
fig.Position = [90 90 600 600];
imagesc(pin, qin, count);
colormap( [jet();flipud( jet() );0 0 0] );
axis('square','equal','image');


h=figure;
subplot(1,3,1),mandelbrot_N(50,500,3)
title('order=3')
pq1=ginput(1);
subplot(1,3,2),mandelbrot_N(50,500,4)
title('order=4')
pq2=ginput(1);
subplot(1,3,3),mandelbrot_N(50,500,5)
title('order=5')
pq3=ginput(1);
set(h,'Units','Inches');
```

```
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex71','-dpdf','-fillpage')

h=figure;
subplot(1,3,1),julia_N(50,500,pq1(1),pq1(2),3)
title(sprintf('p=%1.2f,q=%1.2f,\n order=%1.0f',pq1(1),pq1(2),3))
subplot(1,3,2),julia_N(50,500,pq2(1),pq2(2),4)
title(sprintf('p=%1.2f,q=%1.2f,\n order=%1.0f',pq2(1),pq2(2),4))
subplot(1,3,3),julia_N(50,500,pq3(1),pq3(2),5)
title(sprintf('p=%1.2f,q=%1.2f,\n order=%1.0f',pq3(1),pq3(2),5))
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex72','-dpdf','-fillpage')

% Exercise 8
N=10000000;z=zeros(N,1);
A=1/2+1i*sqrt(3);B=-1;C=1;z(1)=C;
prob=rand(N-1,1);
for k=1:N-1
    if prob(k)<1/3
        z(k+1)=(z(k)+A)/2;
    elseif prob(k)>=1/3 && prob(k)<2/3
        z(k+1)=(z(k)+B)/2;
    else
        z(k+1)=(z(k)+C)/2;
    end
end
h=figure;
subplot(1,2,1),plot(z,'.')
text(-1.05,-0.03,'B','Color','r')
text(1.05,-0.03,'C','Color','r')
text(0.6,1.75,'A','Color','r')
title('P_A=P_B=P_C=1/3')
prob=rand(N-1,1);
for k=1:N-1
    if prob(k)<0.5
        z(k+1)=(z(k)+A)/2;
    elseif prob(k)>=0.5 && prob(k)<0.97
        z(k+1)=(z(k)+B)/2;
    else
        z(k+1)=(z(k)+C)/2;
    end
end
subplot(1,2,2),plot(z,'.')
text(-1.05,-0.03,'B','Color','r')
text(1.05,-0.03,'C','Color','r')
text(0.6,1.75,'A','Color','r')
title('P_A=0.5, P_B=0.47,P_C=0.03')
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex8','-dpdf','-fillpage')

% Exercise 9
h=figure;
s=1/2+1i*1/2;p1=1/2;
N=100000;w=zeros(N,1);prob=rand(N-1,1);
```

```
for k=1:N-1
    if prob(k)<p1
        w(k+1)=s*w(k)+1;
    else
        w(k+1)=s*w(k)-1;
    end
end
plot(w,'.')
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex9','-dpdf','-fillpage')

% Exercise 10
h=figure;
C=i;p1=1/2;
N=10000000;w=zeros(N,1);prob=rand(N-1,1);
for k=1:N-1
    if prob(k)<p1
        w(k+1)=sqrt(w(k)-C);
    else
        w(k+1)=sqrt(w(k)+C);
    end
end
subplot(1,2,1),plot(w,'.','MarkerSize',6)
title('C=i')
C=0.11+1i*0.66;p1=1/2;
N=10000000;w=zeros(N,1);prob=rand(N-1,1);
for k=1:N-1
    if prob(k)<p1
        w(k+1)=sqrt(w(k)-C);
    else
        w(k+1)=sqrt(w(k)+C);
    end
end
subplot(1,2,2),plot(w,'.','MarkerSize',6)
title('C=0.11+0.66i')
set(h,'Units','Inches');
pos = get(h,'Position');
set(h,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3), pos(4)])
print('Ex10','-dpdf','-fillpage')
```