Project 1: Fractal Geometry
MAT128B Winter 2020

Caitlin Brown, Nikos Trembois, and Shuai Zhi

February 19, 2020

Contents

Introduction

In this project, numerical analysis is used to understand and demonstrate fractals and their characteristics. The fractals will be generated from the orbits of complex functions. Orbits are the sequence of numbers that results picking a seed value as a function input, then the outputs are introduced as inputs and so on ad infinitum. So $orb(z_0) = z_0, z_1 = \phi(z_0), z_2\phi(\phi(z_0))...$ is the orbit of the initial point z_0 under the function ϕ. This clearly results in an iterative process, thus computers facilitate the generation of orbits. It is not hard to imagine that for certain initial values this process will diverge while other initial values will converge, or at least remain bounded by some value. The filled Julia set is all points whose orbit, using a polynomial function, remains bounded. The boundary of the filled set is called the Julia set.

The Mandelbrot set is defined as the set of complex numbers c for which the sequence c, c^2+c, $(c^2+c)^2+c$,... does not tend to infinity as the number of iterations tends to infinity. The Mandelbrot set appears to contain smaller copies of itself, however it is actually not self-similar. Each mini-Mandelbrot set has its own unique patterns.

In this project, fractals, specifically Julia and Mandelbrot sets, are explored. It is easy to get lost in their hypnotizing shapes, but one should not forget their utility in practice which is demonstrated with an application to Newton's method on the complex plane.

1 Unit Disk

The orbit of complex values whose real and imaginary part were within [-1, 1] were calculated for the function $\phi(z) = z^2$. The filled Julia set is the map of the complex plane to its orbit under some function. The filled Julia set under $\phi(z) = z^2$ is the unit disk and is shown in figure 1. Under the same function the Julia set would create the unit circle, as it is the boundary of the filled Julia set. It is not surprising that the orbit of z^2 should be the unit disk. The square of values less than 1 in magnitude is a value less than the original value (i.e. $x^2 < x$ if $x < 1$), so if the square is applied over and over to the result, it will approach zero and be bounded. Conversely, if a value is greater than 1 the square of the value is greater than the initial value, thus continually applying the square to the result will lead to values that grow unbounded. However, for other functions the outcome is not as intuitive. The algorithm for this problem is shown in Part 1 of the

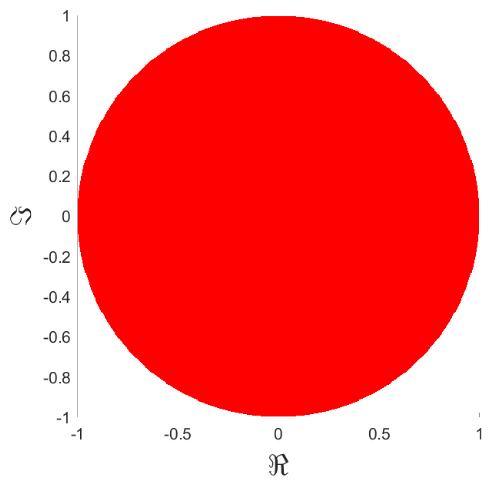


Figure 1: Orbit of $z = a + bi$ for $a = b = [-1, 1]$ under $\phi = z^2$

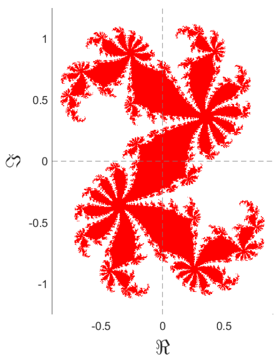
code and the function it calls can be found at the end of the code. Since most problems use a similar iteration in some form a function was created with a few options to do the calculations for multiple parts. The algorithm creates an array of points in the cartesian plane then inputs those points into a function. Then it determines if the value has grown too much to be considered bounded, which the user can set the maximum value for. If it has it either gives that point in the cartesian plane a discrete value or the value of the number of iterations it took to diverge.

2 Introduction to Fractals

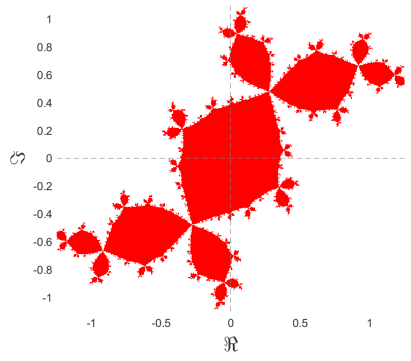
The unit disk is a great illustration of the orbit of different initial points under $\phi = z^2$. However, adding a constant to the function (i.e. $\phi(z) = z^2 + c$) creates more interesting maps, which turn out to be fractals.

3 Julia Set

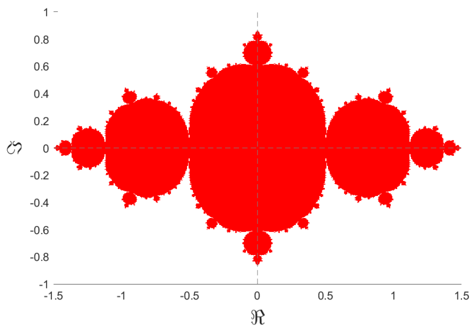
The Julia set is the boundary of the filled Julia set. So, it marks the edge of the points whose orbits converge under a given function. All points inside the boundary converge, including the line, while those outside of it diverge. The Julia set is found using the inverse iteration method, which is an attractor. As the name suggests, initial points will be attracted to certain values as



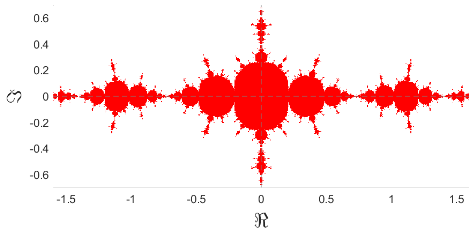
(a) Filled Julia Set of $z = 0.36 + 0.1i$



(b) Filled Julia Set of $z = -0.123{+}0.745i$

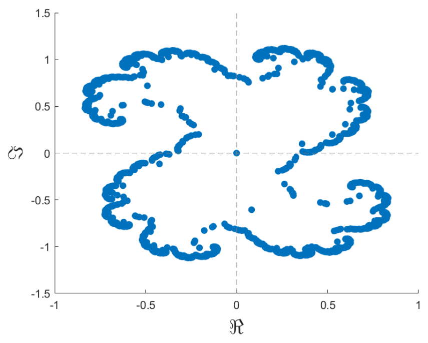


(c) Filled Julia Set of $z = -0.749$

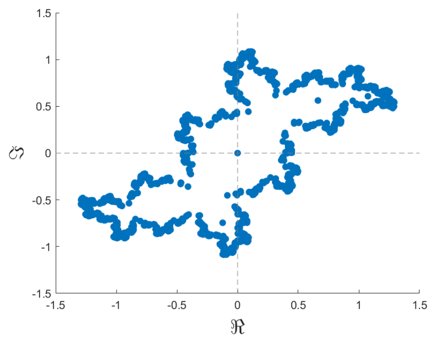


(d) Filled Julia Set of $z = -1.25$

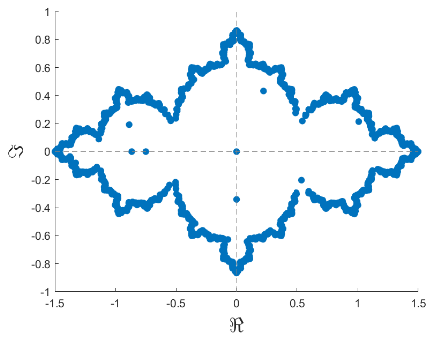
Figure 2: Filled Julia sets with 500 points in real and imaginary axis



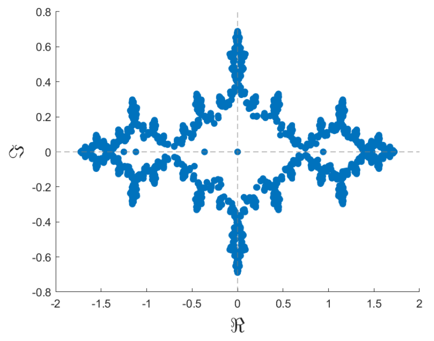
(a) Julia Set of $z = 0.36 + 0.1i$



(b) Julia Set of $z = -0.123 + 0.745i$



(c) Julia Set of $z = -0.749$



(d) Julia Set of $z = -1.25$

Figure 3: Julia sets with 10,000 iterations of the inverse function

it is iterated. In fact, the plots below are examples of a specific type of attractor called strange attractors. This is because the attractor maps to a fractal shape. Since the Julia set uses inverse function, $\phi = z^2 + c$ becomes $\psi = \sqrt{z - c}$. Due to the square root, any given iteration is actually attracted to two different values, one being the negative root and the other the positive, which are chosen at random.

4 Fractal Dimensions

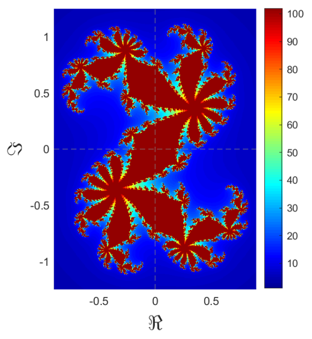
Fractals are shapes that don't have integer dimensions. A line, square and cube have one, two and three dimensions respectively, while a fractal may have dimension 1.585 (Sierpinski Triangle) or 2.3 ($\approx$ a choppy sea surface during a storm). Mathematicians measure dimension by observing the effects of scaling the length of a shape on the shape's "mass". For example, scaling the length of a line by $\frac{1}{2}$ creates a line $\frac{1}{2}$ the size of the original, and because $\frac{1}{2}^1 = \frac{1}{2}$, a line is one dimensional. Similarly, scaling the length of a square by $\frac{1}{2}$ produces a square $\frac{1}{4}$ the size of the original, and $\frac{1}{2}^2 = \frac{1}{4}$. The same calculation can be done on a Sierpinski Triangle, although it has infinite length and zero area. Scaling a side by $\frac{1}{2}$ gives a new triangle $\frac{1}{3}$ the size of the original. $\frac{1}{2}^x = \frac{1}{3}$ invites the utilization of logarithms and gives rise to equation (1) from Bisoi and Mishra 2001. In nature, and in Julia sets for that matter, scaling factors are not so apparent and box counting coupled with least squares linear approximations through points on log-log plots must be used.

5 Julia Set Connectivity

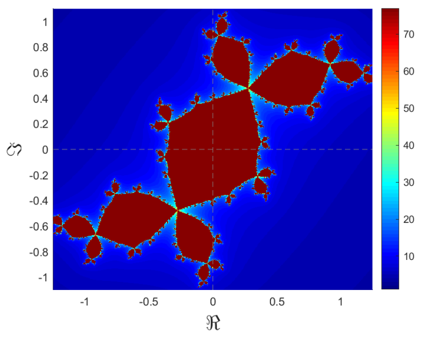
A Julia set is considered connected if a point in the set can travel to another point in the set without having to leave the set. In other words, there is only one boundary that encloses all points. Julia discovered a simple criterion to determine if Julia sets are connected. He found that if the orbit of the initial point 0 is bounded, then the set is connected, otherwise it is not. Therefore, the connectivity of set under the function $\phi(z)$ can be easily found, but careful consideration must be required for the number of iterations to use. Too few iterations may not be enough to capture the diverging effect of the orbit, while too many may require an unnecessary number of computations. The algorithm in Part 5 of the code below shows how the connectivity is determined. Notice that the output never definitively states a Julia set is connected because it is not generally known if a value is truly converged when there are a finite number of iterations. It is possible a point in a set that stays bounded at the $10,000^{th}$ iteration will diverge at the $20,000^{th}$, however unlikely that may seem.

6 Divergent Orbits

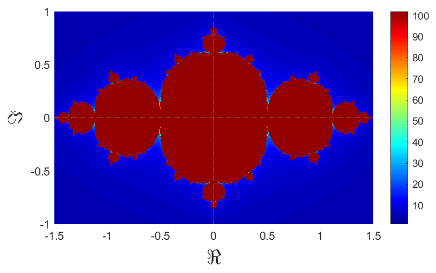
The number of iterations it takes for a value to diverge creates interesting plots. Coloring of the orbits is according to the number of iterations that the orbits take to diverge. We used the filled Julia set functions and set the threshold of divergence to $|z| = 100$, and then used the spectrum chart to represent the number iterations by color. Points on the complex plan that took many iterations to diverge, or didn't converge at all, have a red color; while points that diverged quickly have blue color. The orbits with similar



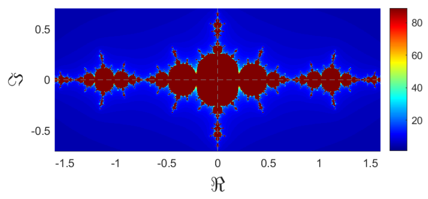
(a) Julia Set of $z = 0.36 + 0.1i$



(b) Julia Set of $z = -0.123 + 0.745i$



(c) Julia Set of $z = -0.749$



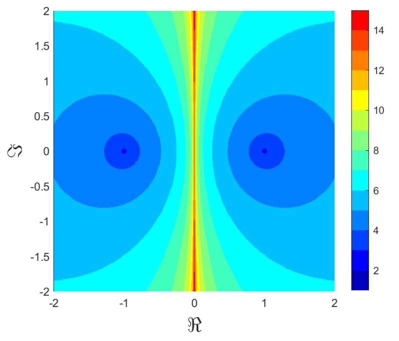
(d) Julia Set of $z = -1.25$

Figure 4: Filled Julia sets colored based on number of iterations to diverge

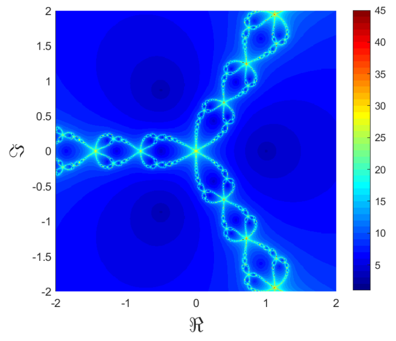
iteration number would have the similar colors. Additionally, the orbits of similar colors have similar shape and iteration numbers. Accordingly, self similarity can be found in orbits with the same color.

7 Newton's Method in Complex Plane

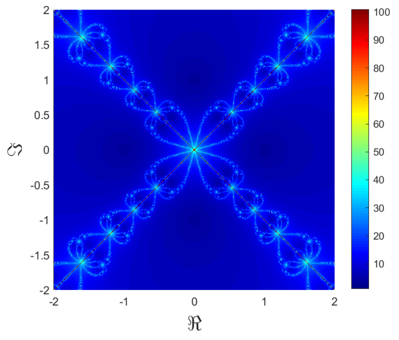
Root finding can be a surprisingly difficult task. The linear case is trivial and roots of second order polynomials are solved with the quadratic equation. However, as the order increases the analytic equations to solve for the roots become more intricate and no known analytic equation exists for polynomials of orders higher than 6. Not to mention, this is just for real functions! Finding the roots of complex functions is even more difficult. Fortunately iterative methods, with the help of plots, simplify the process; although,



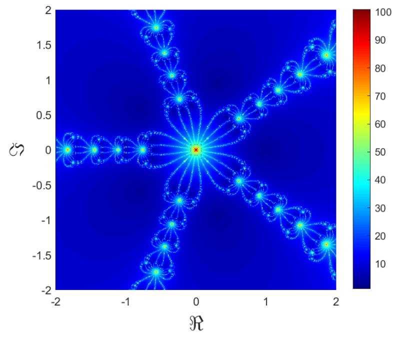
(a) Iterations to roots of $z^2 - 1$



(b) Iterations to roots of $z^3 - 1$



(c) Iterations to roots of $z^4 - 1$



(d) Iterations to roots of $z^5 - 1$

Figure 5: Roots of complex functions of form $z^n - 1$

some accuracy will be lost. The plots in figure 5 are not just interesting, but also insightful. When plotting the Newton fractals and coloring the points with the number of iterations required to converge, the location of the roots become evident. For Newton's method, and any iterative method I can think of, the closer an initial guess is to a root, the less iterations are required. So the roots are found where the plots indicate the least amount of iterations are, in this case dark blue. Looking at figure 5b it appears the roots around $(1, 0i), (-0.5, 0.86i), (0.5, -0.86i)$. This agrees with the known values of $(1, 0i), (-0.5, \frac{\sqrt{3}}{2}i), (-0.5, -\frac{\sqrt{3}}{2}i)$. The process extends to all subfigures in figure 5 and to any complex function whose roots are desired.

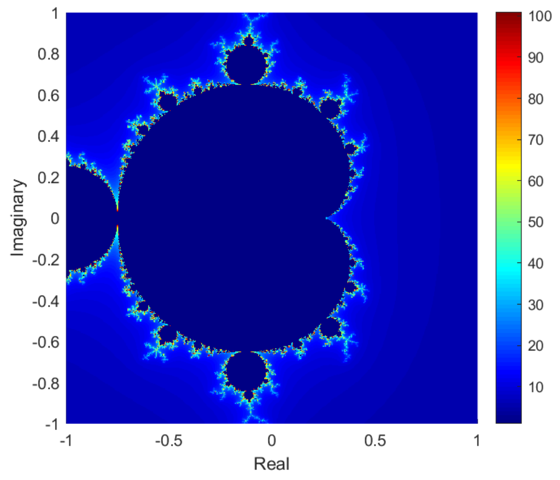


Figure 6: Mandelbrot set showing which constants for a connected Julia Set under $\psi = z^2 + c$

8 The Mandelbrot Set

In the Julia sets for $z_{n+1} = z^2 + c$, we fix c and cycle through z_0 values to create one image. Different Julia set images can be created with different values of c. In the Mandelbrot set, however, we will fix $z_0 = 0$ and cycle through all c values in the complex plane. The coloring of the image indicates the number of iterations required for the particular value to diverge. By starting with an initial point at the origin and searching through different constants in the complex plane the Mandelbrot set must be all values of c such that the Julia sets are connected, since bounded orbits of 0 form connected Julia sets.

9 Conclusion

Fractals are interesting shapes that come about when finding the orbits of complex functions. Due to the iterative nature of orbits, visually representing them was a lengthy process. However, with computers, fractals can be easily visualized. One can switch the function under which the orbit is to be calculated by changing a few values. Previously, this would have required manually recalculating the orbit of each point. Fractals, which gets its name from fractional dimensions, are more than just interesting shapes. Fractals can represent important values for problem solving. Take Newton's iterations for example, the roots of an equation can be found by observing which values in the domain converge to a root the fastest.

10 Appendix

10.1 Code

```
% MAT 128B: Project 1
% UC Davis Winter 2020
% Nikos Trembois, Caitlin Brown, and Shuai Zhi

clc; close all; clearvars

global c xRange yRange pts bsave
% Constants that will be used for z^2 - c plots
c = [0.36 + 0.1i, -.123 - .745i,-.749, -1.25];
xRange = [0.9, 1.25, 1.5, 1.6]; % Range of x values for plotting
    window
yRange = [1.25, 1.1, 1, 0.7]; % Range of y values for plotting
    window
pts = 500; % Number of points in x and y directions of plot
bsave = 0; % boolean value (0,1) to save plots to file or not

%% Part 1: Fractals
phi = @(z,c) z^2; % Defining the function
% Iterating the Julia Set with function defined at bottom of
    script
M = FilledJuliaSet(phi,1,1,pts,0,'nocolor',2);

figure(); hold on
xlabel('\Re','Fontsize',18); ylabel('\Im','Fontsize',18)
colormap([1 0 0; 1 1 1]);
image( [-1 1], [-1 1], M')
axis xy; axis equal; axis([ -1 1 -1 1])
hold off
if bsave == 1
    saveas(gcf,'../Figures/UnitDisk.png')
end

%% Part 2:  Fractals
clearvars -except c xRange yRange pts bsave
phi = @(z,c) z^2 + c;
M = cell(length(c),1);
for i = 1:length(c)
    M{i} = FilledJuliaSet(phi,xRange(i),yRange(i),pts,c(i),'
        nocolor',2);
end

% Plotting the maps of orbits
for i = 1:length(c)
```

```
    figure(); hold on
    colormap([1 0 0; 1 1 1]); % Define colors for map
    image( [-xRange(i) xRange(i)], [-yRange(i) yRange(i)], M{i}')
    axis xy; axis equal; ax = gca; % Formatting below
    ax.XLim = [-xRange(i) xRange(i)]; ax.YLim = [-yRange(i) yRange
        (i)];
    plot(ax.XLim,[0,0],'LineStyle','--','Color',[.5,.5,.5])
    plot([0,0],ax.YLim,'LineStyle','--','Color',[.5,.5,.5])
    xlabel('\Re','Fontsize',18); ylabel('\Im','Fontsize',18)
    outerpos = ax.OuterPosition;
    ti = ax.TightInset;
    left = outerpos(1) + ti(1);
    bottom = outerpos(2) + ti(2);
    ax_width = outerpos(3) - ti(1) - ti(3);
    ax_height = outerpos(4) - ti(2) - ti(4);
    ax.Position = [left bottom ax_width ax_height];
    hold off
    if bsave == 1 % Save plot to file if bsave = 1
        ssave = strcat('../Figures/FilledJulia',num2str(i),'.png')
            ;
        saveas(gcf,ssave)
    end
end

%% Part 3: Julia Sets
clearvars -except c xRange yRange pts bsave
psi = @(z,c) sqrt(z - c); % Define plot
x = zeros(pts,length(c)); % initialize a x vector for each
    constant
y = zeros(pts,length(c)); % initialize a y vector for each
    constant
for k = 1:length(c)
    clear z;
    z = c(k);
    for j = 1:10000 % Use 10,000 iterations
        x(j,k) = real(z(j));
        y(j,k) = imag(z(j));
        % Randomly choose positive or negative root with equal
            weighting
        if randi([0 1],1,1) == 1
            z(j+1) = psi(z(j),c(k));
        else
            z(j+1) = -psi(z(j),c(k));
        end
    end
end

% Plot the Julia sets for different constant values
for i = 1:length(c)
```

```
    figure(); hold on
    scatter(x(:,i),y(:,i),'filled') % Plot the Julia Set
    ax = gca; % Plot formatting below
    plot(ax.XLim,[0,0],'LineStyle','--','Color',[.5,.5,.5])
    plot([0,0],ax.YLim,'LineStyle','--','Color',[.5,.5,.5])
    xlabel('\Re','Fontsize',18); ylabel('\Im','Fontsize',18)
    hold off
    if bsave == 1 % Save plot to file if bsave = 1
        ssave = strcat('../Figures/Julia',num2str(i),'.png');
        saveas(gcf,ssave)
    end
end

%% Fractal Dimension
% Using square domain to capture fractal dimension more easily
clearvars -except c xRange yRange pts bsave
phi = @(z,c) z^2 + c;
range = 2; % define a range used for both x and y, for square
    spacing
M = cell(length(c),1);
for i = 1:length(c)
    M{i} = FilledJuliaSet(phi,range,range,pts,c(i),'nocolor',2);
end
% Calculating the fractal dimension
for i = 1:length(c)
    r = 2*range/pts; % calculate resolution
    fprintf('For c = (%4.2f, %4.2fi)',real(c(i)),imag(c(i)))
    FractalDimension(M{i},r)
end

%% Part 5: Connectivity of the Julia Set
clearvars -except c xRange yRange pts bsave
max_iter = 1000; % Use 1000 iterations to determine if orbit
    diverges
for k = 1:length(c)
    psi = @(z) z^2 + c(k);
    fprintf('For c = (%4.2f, %4.2f)',real(c(k)),imag(c(k)))
    z = 0; % Initialize value
    for i = 1:max_iter
        z = psi(z); % Calculate the orbit
        if abs(z) > 100 % Considered divergent when magnitude is
            greater than 100
            fprintf('The orbit diverged after %i iterations, the
                set is not connected\n',i)
            break
        end
    end
    if abs(z) < 100
```

```
        fprintf('The set did not diverge after %i iterations\n',
            max_iter)
        fprintf('It is reasonable to assume the Julia set is
            connected\n')
    end
end

%% Part 6: Coloring Divergent Orbits
clearvars -except c xRange yRange pts bsave
phi = @(z,c) z^2+ c;
for i = 1:length(c)
    % Once again using the FilledJuliaSet function is used with
        options
    % 'colored' to color plots based on iterations to diverge
    % past an absolute value of 100
    M{i} = FilledJuliaSet(phi, xRange(i), yRange(i), pts, c(i), '
        colored', 100);
end

% Plotting the Diverging orbits
for i = 1:length(c)
    figure(); hold on
    % plot map of diverging orbits
    image( [-xRange(i) xRange(i)], [-yRange(i) yRange(i)], M{i}')
    axis xy; axis equal; ax = gca; % Plot formatting below
    ax.XLim = [-xRange(i) xRange(i)]; ax.YLim = [-yRange(i) yRange
        (i)];
    plot(ax.XLim,[0,0],'LineStyle','--','Color',[.5,.5,.5])
    plot([0,0],ax.YLim,'LineStyle','--','Color',[.5,.5,.5])
    xlabel('\Re','Fontsize',18); ylabel('\Im','Fontsize',18)
    colormap(jet(max(max(M{i})))); colorbar; hold off
    if bsave == 1 % Save plot to file if bsave = 1
        ssave = strcat('../Figures/ColoredJulia',num2str(i),'.png'
            );
        saveas(gcf,ssave)
    end
end

%% Part 7: Newton's Method in the Complex Plane
clearvars -except c xRange yRange pts bsave
% Defining function that represents f/f' for a general f = z^n - 1
g = @(z,n) (z^n - 1)/(n*z^(n-1));
a = linspace(-2,2,500); b = linspace(-2,2,500); % set plottig
    window
nmax = 5; % The maximum polynomial order to plot to
M = cell(nmax-1,1);
for n = 2:nmax % start with order 2 as order 1 is not interesting
    M{n-1} = 100*ones(length(a),length(b)); % initialize values to
    gn = @(z) (z^n - 1)/(n*z^(n-1)); % Redefine g for specific
        value of n
    for r = 1:length(a)
        for i = 1:length(b)
            z = a(r) + 1i*b(i);
            for j = 1:100
                if abs(z^n-1) > 0.001 % Testing for convergence
                    z = z - gn(z); % Newton's Iteration
                else
                    if j < 3 % Printing values of possible roots
                        fprintf('For n = %i, The root is near %2
                            .4f, %2.4f\n',n,a(r),b(i));
                    end
                    M{n-1}(r,i) = j; % Set value to number of
                        iterations
                    break;
                end
            end
        end
    end
end

% Plotting the Newton iterations
for i = 1:nmax-1
    figure(); hold on
    % Plotting map of iterations of convergence
    image( [min(a) max(a)], [min(b) max(b)], M{i}')
    axis xy; axis equal; ax = gca; % Plot formatting below
    ax.XLim = [min(a) max(a)]; ax.YLim = [min(b) max(b)];
    xlabel('\Re','Fontsize',18); ylabel('\Im','Fontsize',18)
    colormap(jet(max(max(M{i})))); colorbar
    hold off
    if bsave == 1 % Save plot to file if bsave = 1
        ssave = strcat('../Figures/Newton',num2str(i),'.png');
        saveas(gcf,ssave)
    end
end


%% Part 8: Mandelbrot Set
clearvars -except c xRange yRange pts bsave
phi = @(z,c) z^2 + c; % function to plot
a = linspace(-1,1,pts); % Plotting Window
b = linspace(-1,1,pts);
M = ones(length(a),length(b));

for r = 1:length(a)
    for i = 1:length(b)
        z = 0; % start with z = 0
        cm = a(r) + b(i)*1i;
        for j = 1:100
            z = phi(z,cm);
            if abs(z) > 100
                M(r,i) = j;
                break;
            end
        end
    end
end

% Plot Mandelbrot set
figure(); hold on
xlabel('Real'); ylabel('Imaginary')
colormap(jet(100)); colorbar
image( [-1 1], [-1 1], M')
axis xy; axis equal; axis([ -1 1 -1 1])
if bsave == 1 % Save plot to file if bsave = 1
    saveas(gcf,'../Figures/Mandelbrot.png')
end

%% Functions
function FractalDimension(M,r)
    n = length(M);
    N = 0; % Initialize value of N for box-counting
    for i = 1:n
        for j = 1:n
            if M(i,j) == 1
                N = N+1;
            end
        end
    end
    d = log(N)/log(1/r);
    fprintf(' the fractal dimension is %5.4f\n', d);
end

% This function determines if a point is in a Julia Set
function [ M ] = FilledJuliaSet(phi, xrange, yrange, pts, c,
    colored, maxvalue)
    a = linspace(-xrange, xrange, pts);
    b = linspace(-yrange, yrange, pts);
    if (strcmp(colored,'colored'))
        M = zeros(length(a), length(b));
    else
        M = ones(length(a), length(b));
    end
    for r = 1:length(a)
        for i = 1:length(b)
            clear z;
            z = a(r) + 1i*b(i);
            for j = 1:100
                z = phi( z, c );
                if abs( z ) > maxvalue
                    if (strcmp(colored,'colored'))
                        % If the colored option is 'colored', then
                        % color the julia set by the number of
                        % iterations to converge
                        M(r,i) = j;
                    else % otherwise give discrete value
                        M(r,i) = 2;
                    end
                    break
                end
            end
        end
    end
    if (strcmp(colored,'colored'))
        M(M==0) = max(max(M)) + 1;
    end
end
```

10.2 Group Work

The group met on several occasions to discuss division of work and how to collaborate. We initially used a divide and conquer method, but began collaborating as the problems were challenging and we needed the skills and knowledge of the whole group. Nikos and Caitlin collaborated on the first two problems, while Shuai and Nikos worked on the coloring of Julia sets. Shuai and Nikos also worked on using polar coordinates for the Julia set boundary, but we resulted in using cartesian coordinates for its simpler implementation which still gave good results. Caitlin did the background reading and code for calculating the fractal dimension. For the first part, generating the unit disk, Nikos created a function for calculating the Julia set, which in turn could be used for many of the other parts of the project. So the same function was repurposed for part ii, iv, and vi. As the Mandelbrot set and calculating the connectivity of Julia set were similar to the other Julia set questions, Nikos created the code for that as well. Nikos did background reading and the algorithm implementation for part vii, Newton's Method in the complex plane. As for the report, everyone contributed to the sections they worked on.