Project 1: Fractal Geometry
MAT128B Winter 2020

Caitlin Brown, Nikos Trembois, and Shuai Zhi

February 19, 2020

Contents

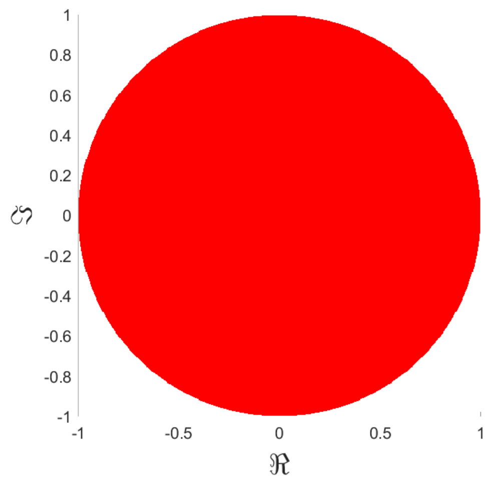


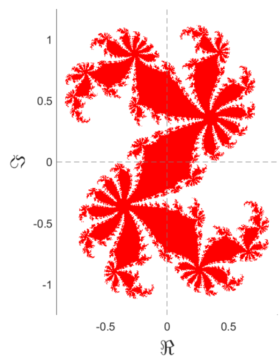
Figure 1: Orbit of $z = a + bi$ for $a = b = [-1, 1]$ under $\phi = z^2$

1 Introduction

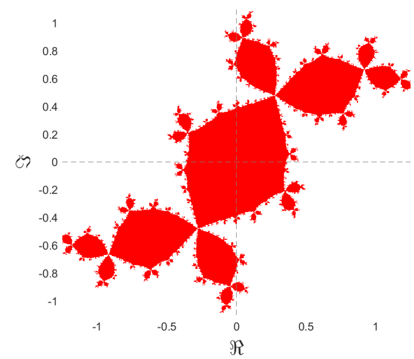
In this project, numerical analysis is used to understand and demonstrate fractals and their characteristics. The fractals will be generated from the orbits of complex functions. Orbits are the sequence of numbers that results from the process of applying a function to the output of the same function over and over, like a recursive function. 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. 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.

2 Unit Disk

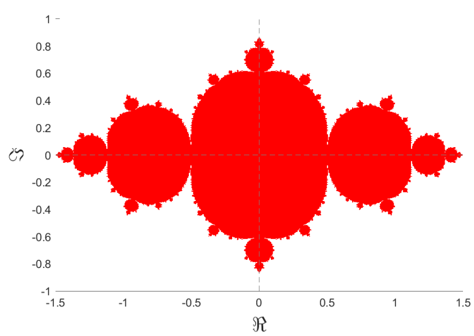
The orbit of complex values whose real and imaginary part were within [-1, 1] were calculated for the function $\phi(z) = z^2$. The filled Julia set is the map of the complex plane to its orbit under some function. The filled Julia set under $\phi(z) = z^2$ is the unit disk and is shown in figure 1. Under the same function the Julia set would create the unit circle, as it is the boundary of the filled Julia set.



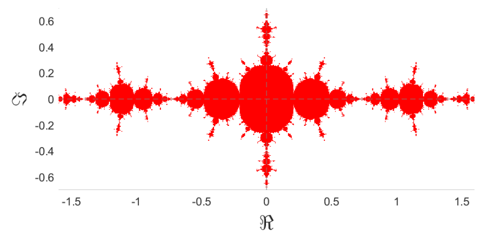
(a) Filled Julia Set of $z = 0.36 + 0.1i$



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



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



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

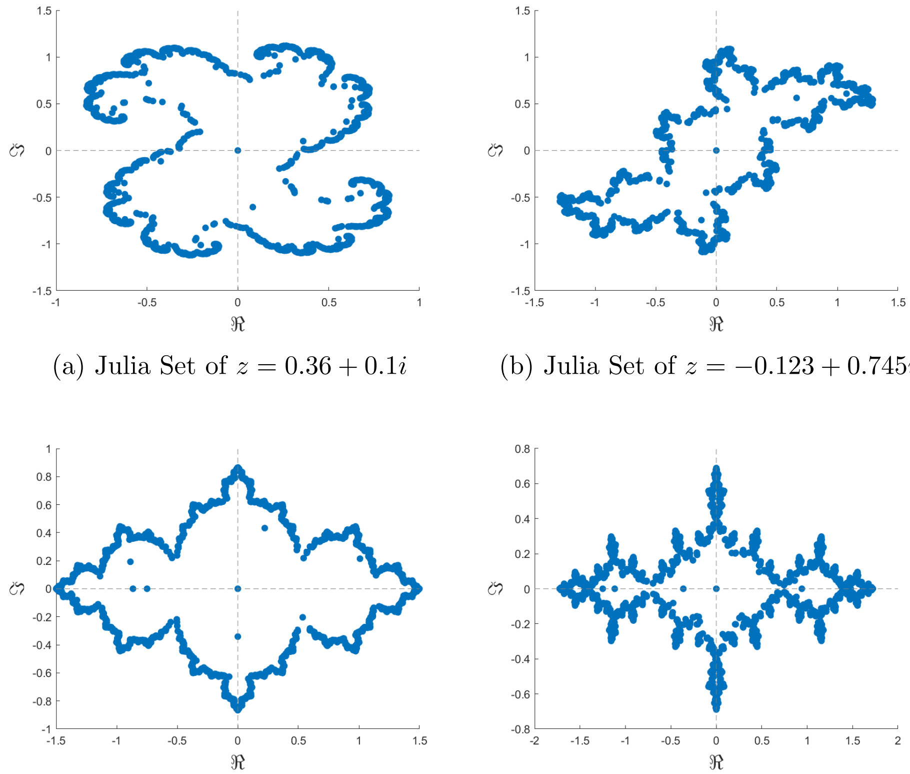
Figure 2: Filled Julia sets with 500 points in real and imaginary axis

3 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.

4 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) 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 $z = -1.25$

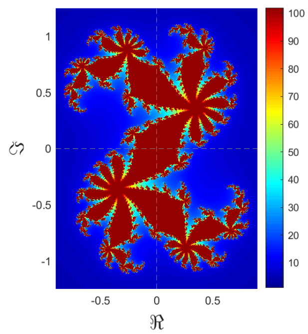
Figure 3: Julia sets with 10,000 iterations of the inverse function

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 begin the negative root and the other the positive, which are chosen at random.

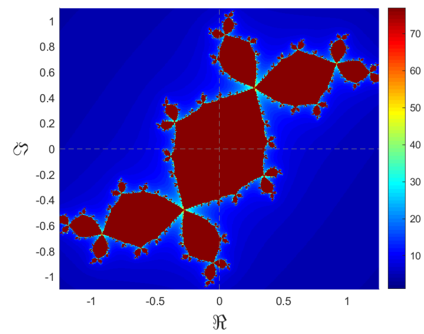
5 Fractal Dimensions

6 Julia Set Connectivity

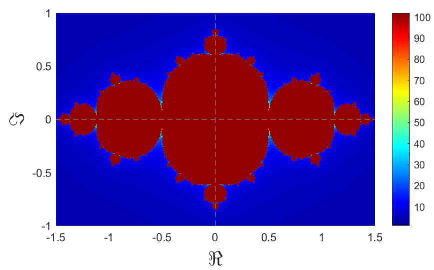
A Julia set is considered connected if a point in the set can travel to another point in the set without having to leave the set. In other words, there is only



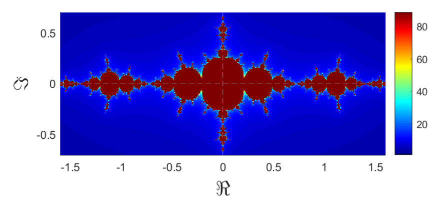
(a) Julia Set of $z = 0.36 + 0.1i$



(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 converge

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 number of iterations to use. Too few iterations may not be enough to capture the diverging effect of the orbit, while to many may require an unnecessary number of computations.

7 Divergent Orbits

The number of iterations it takes for a value to diverge creates interesting plots where it is

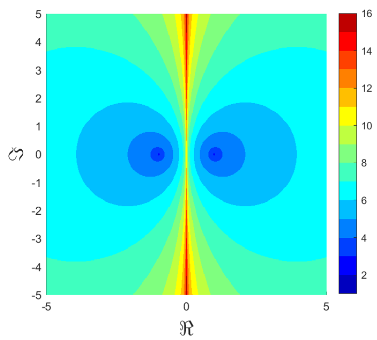
8 Newton's Method in Complex Plane

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, 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.

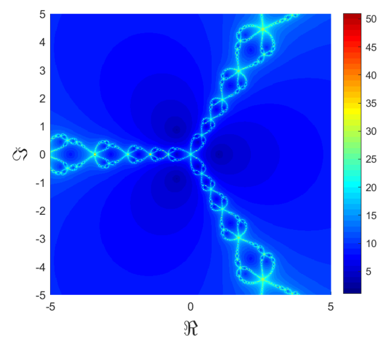
9 The Mandelbrot Set

10 Conclusion

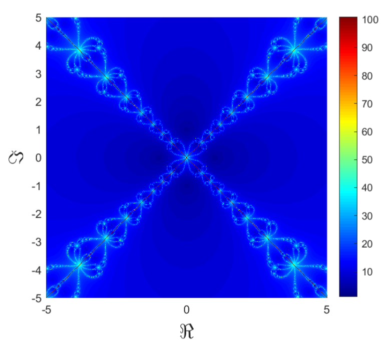
Fractals are interesting shapes that come about when finding the orbits of complex functions. Due to the iterative nature of orbits, visually representing was a lengthy process. However, with computers, fractals can be easily visualized. One can switch the function under which the orbit is be calculated for with just a few keystrokes. Previously, this would have required manually recalculating the orbit of each of point. Fractals, which gets its name from fractional dimensions, are more than just interesting shapes. These 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.



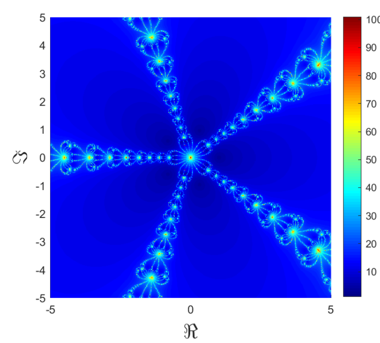
(a) Iterations to roots of $z^2 - 1$



(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 $z^n - 1$

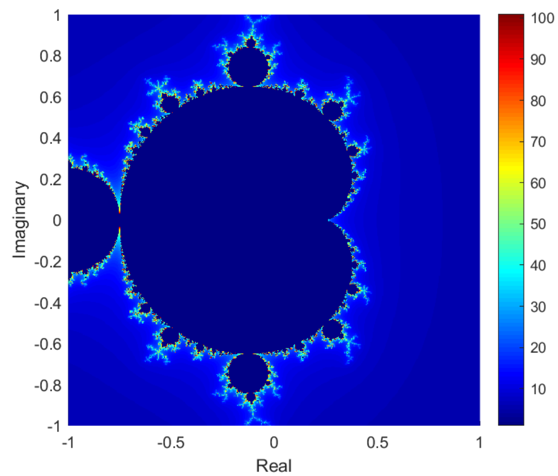


Figure 6: Mandelbrot set ...

11 Appendix

11.1 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
psi = @(z) z^2 + 3; % Calculate inverse iteration undder some
    function psi
z = 0; % Initialize value
max_iter = 1000; % Use 1000 iterations to determine if orbit
    diverges
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

%% 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(-5,5,500); b = linspace(-5,5,500); % 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
```