Iterations and Fractals

Alina Enikeeva: 251160024
Erina Mitha: 251279433

February 2024

1 Abstract

Iterations and Fractals is the second part in a series of four reports that explores the implementation of Newton's Method to find the roots of functions and create fractals in the complex plane. Additionally, this laboratory examines the convergence of the solutions of the algorithm, comparing the expected and experimental errors.

2 Introduction

Fixed point iteration is a concept that uses a fixed point where g(x) = x, iterating through this formula to find the solution to an equation ("Fixed Point Iteration). The Babylonians used a fixed point iteration method for calculating the square root of a number, which converges for almost any number (Dominus). It works by making a guess for the root, improving the guess by plugging the guess into a formula, and then continuing to iterate through a formula until convergence (Wicklin). The Babylonian's method of fixed point iteration for square roots essentially finds the zeroes of the function $x^2 - n$, where n is the number we are trying to find the square root of (Dominus). This method of finding square roots is a special case of Newton's method.

Newton's Method: Newton's method finds the roots of the function by iterating through a formula. It uses an initial guess and continuously draws tangent lines that get progressively closer to the actual root of the function. The numerical method uses a formula with three inputs: the original function (f), its derivative (f'), and an initial guess for the roots (x_0). The algorithm works effectively, generally converging linearly or quadratically, depending on the type of function and multiplicity of its roots. This laboratory explores the implementation of the method and the convergence of its solutions.

Fractals and the Complex Plane: The laboratory also applies Newton's method to the complex plane to create fractals. Fractals, infinitely complex geometric patterns, emerge from the exploration of mathematical equations in the complex plane. Among these equations, $z^3 = 1$, $z^4 = 1$ and $z^5 = 1$ are particularly interesting to examine. They represent the roots of unity, solutions where raising a complex number to a certain power yields unity. These equations not only have fundamental significance in algebra but also give rise to fractal patterns when iteratively explored through numerical methods like Newton's method.

3 Problem 2.1

Newton's method is a fixed point iteration algorithm that attempts to find the root of a function, employing the following formula:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

where f is a real function and f' is the derivative of the function, and x_k is the most recent value of x calculated by the function. The first time the function is called, however, x_k takes the value of x_0, the initial guess of the root which is specified at the time of function call.

Essentially, Newton's method works by first "drawing" a tangent line for the function f at $f(x_0)$. The root of this tangent line is found using the iterative formula (shown above) and stored in x_{k+1}. The value of i increments in the "for loop" of the algorithm, and the root that was just calculated is then used for the value of x_k for the next iteration. The formula is iterated through for $i = 0$ until the maximum number of iterations (specified during the time of function call) or until $abs(x_{k+1} - x_k)$ is within the provided tolerance—whichever condition is met first. The code for this method can be seen in Figure 2.1.

4 Problem 2.2

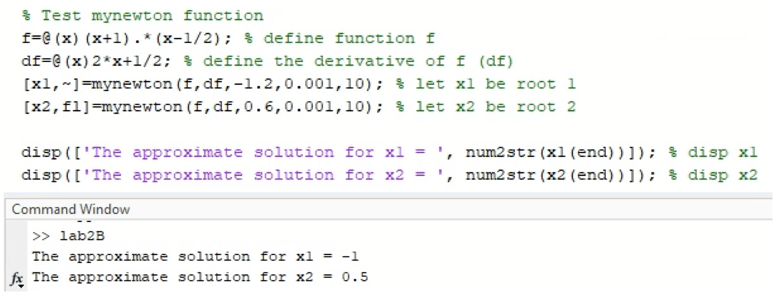
Suppose we take the function

$$f(x) = (x + 1)(x - \frac{1}{2})$$

whose derivative is

$$f'(x) = 2x + \frac{1}{2}$$

We can see in Figure 2.21 that the code tests the "mynewton" algorithm for the function f, producing the correct roots:



```
% Test mynewton function
f=@(x)(x+1).*(x-1/2); % define function f
df=@(x)2*x+1/2; % define the derivative of f (df)
[x1,~]=mynewton(f,df,-1.2,0.001,10); % let x1 be root 1
[x2,f1]=mynewton(f,df,0.6,0.001,10); % let x2 be root 2

disp(['The approximate solution for x1 = ', num2str(x1(end))]); % disp x1
disp(['The approximate solution for x2 = ', num2str(x2(end))]); % disp x2
```

Command Window

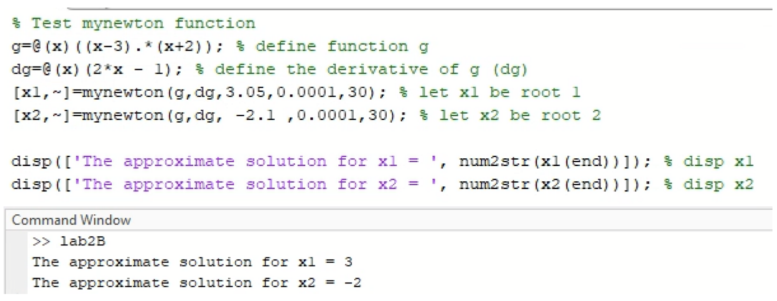
```
>> lab2B
The approximate solution for x1 = -1
The approximate solution for x2 = 0.5
```

Figure 2.21: Finding the roots for $f(x)$ using Newton's method

Suppose we look at another example and let

$$g(x) = (x - 3)(x + 2) \Longrightarrow g'(x) = 2x - 1$$

As we can see, the "mynewton" algorithm correctly produces the roots of the function again:



```
% Test mynewton function
g=@(x)((x-3).*(x+2)); % define function g
dg=@(x)(2*x - 1); % define the derivative of g (dg)
[x1,~]=mynewton(g,dg,3.05,0.0001,30); % let x1 be root 1
[x2,~]=mynewton(g,dg, -2.1 ,0.0001,30); % let x2 be root 2

disp(['The approximate solution for x1 = ', num2str(x1(end))]); % disp x1
disp(['The approximate solution for x2 = ', num2str(x2(end))]); % disp x2
```

Command Window

```
>> lab2B
The approximate solution for x1 = 3
The approximate solution for x2 = -2
```

Figure 2.22: Finding the roots for $g(x)$ using Newton's method

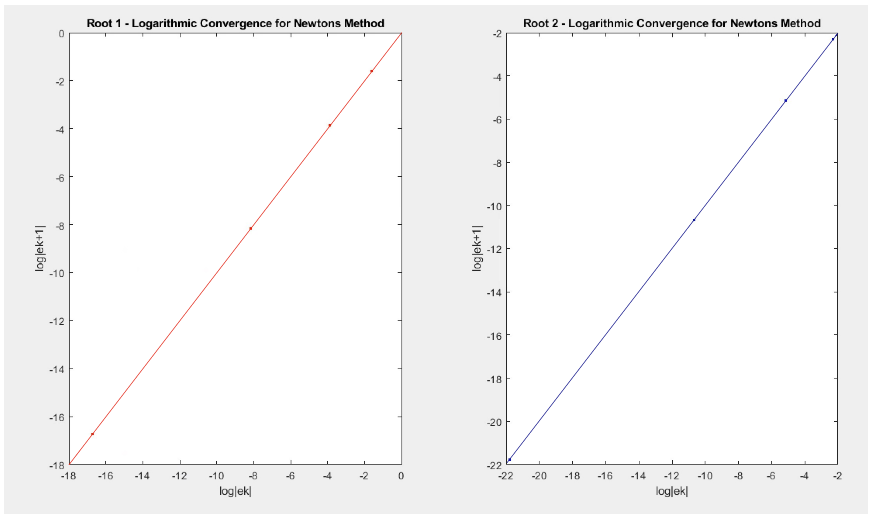
5 Problem 2.3

To determine the order of convergence in a log-plot, we can use the formula

$$log_e|e_{k+1}| \approx \alpha log_e|e_k| + log_e C$$

where we plot $log_e|e_k|$ on the x-axis and $log_e|e_{k+1}|$ on the y-axis.

Some of the code used to produce this linearization of the convergence is pictured in Figure 2.3. The following graphs were produced for $f(x)$, where the first subplot is for the root $x = -1$ and the second subplot is for the root $x = \frac{1}{2}$:



Plot 2.3: Plot for the Linearized Convergence of Newton's Method for $f(x)$.

The calculation for the slopes for each plot are as follows, where α_1 and α_2 are for the slopes of the convergence plots for the roots $x = -1$ and $x = \frac{1}{2}$, respectively:

$$\alpha_1 = \frac{y_1 - y_2}{x_1 - x_2} = \frac{0 - (-18)}{0 - (-18)} = \frac{18}{18} = 1$$

$$\alpha_2 = \frac{y_1 - y_2}{x_1 - x_2} = \frac{(-2) - (-22)}{(-2) - (-22)} = \frac{20}{20} = 1$$

Slope Analysis: The line of best first for both plots have a slope of $\alpha = 1$, which implies linear convergence. In Newton's method we expect quadratic convergence ($\alpha = 2$)if $f'(r) \neq 0$ and linear convergence ($\alpha = 1$) if $f'(r) = 0$ (Rate of convergence) for simple roots (Burton). The both roots of $f(x)$ are simple, or have a multiplicity of one, we should expect quadratic convergence for both roots if the derivative of the function at the roots is not zero (Weisstein). Looking at our specific function $f(x) = (x + 1)(x - \frac{1}{2})$,

$$f'(-1) = 2(-1) + \frac{1}{2} = -\frac{3}{2} \neq 0$$

$$f'(\frac{1}{2}) = 2(\frac{1}{2}) + \frac{1}{2} = 1 \neq 0$$

so we would expect quadratic convergence ($\alpha = 2$ for both roots of $f(x)$. Interestingly, we observe linear convergence ($\alpha = 1$) numerically. This is because taking the logarithms will inevitably give us a linear model. If we were to perform the same plot without the logarithmic scale, we would observe quadratic convergence.

6 Problem 2.4

Newton's method often requires very close guesses to the actual root of the function to work ("Content - Newton's Method"). In practice, however, this becomes tricky because the method assumes we already know the root. Similarly, determining the convergence (α), requires us to know the exact solution.

Let us assume we have functions that we know converge quadratically, where the functions are well behaved and have no multiplicities in their roots. For this we can use functions f and g from Problem 2.2. The following program attempts to detect quadratic convergence in the function f, with the code pictured in Figure 2.4 and the corresponding plot show in Plot 2.4:

7 Problem 3.1

The purpose of this section is to explore the roots of unity in the complex plane and apply Newton's method to find these roots numerically. The cube roots of unity, denoted as $z^3 = 1$, have three unique solutions in the complex plane. The goal is to investigate how different initial values of z_0 in the complex plane converge to these roots.

Theory: The cube roots of unity can be expressed as $z = \exp\left(\frac{i2\pi n}{3}\right)$, where n is an integer. Using Euler's formula, $\exp(i\theta) = \cos\theta + i\sin\theta$, we can compute the roots as follows:

$$r_1 = 1, \quad r_2 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i, \quad r_3 = -\frac{1}{2} - \frac{\sqrt{3}}{2}i$$

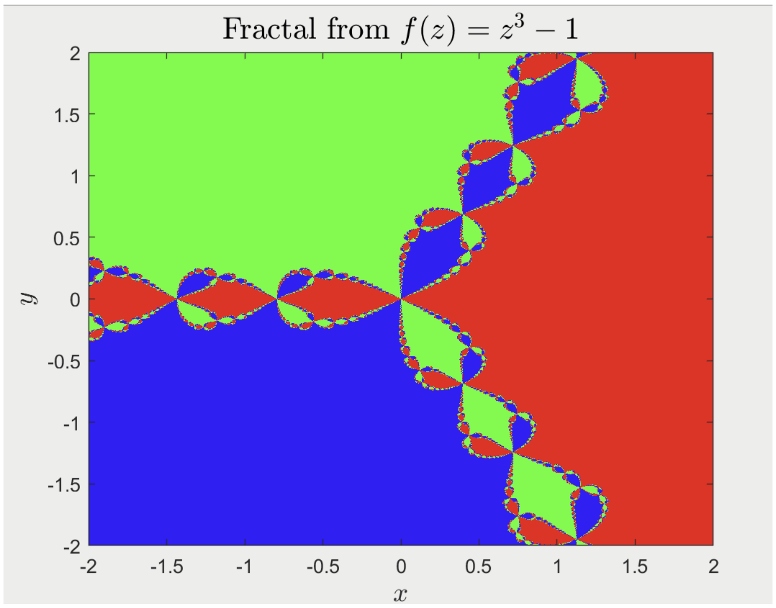
Newton's method is employed to find the roots numerically. The iteration formula is:

$$z_{n+1} = z_n - \frac{f(z)}{f'(z)}$$

where $f(z) = z^3 - 1$ and $f'(z) = 3z^2$.

By iterating this formula, we can determine which initial values z_0 in the complex plane converge to each of the three cube roots of unity. The fractal generated by the convergence patterns highlights the intricate nature of the roots in the complex plane and provides visual insights into the behavior of Newton's method.

The following Matlab program grids up a rectangle in the complex plane and determines the convergence of each grid point z_0, producing Plot 3.1:



Plot 3.1: Fractal form $f(z) = z^3 - 1$.

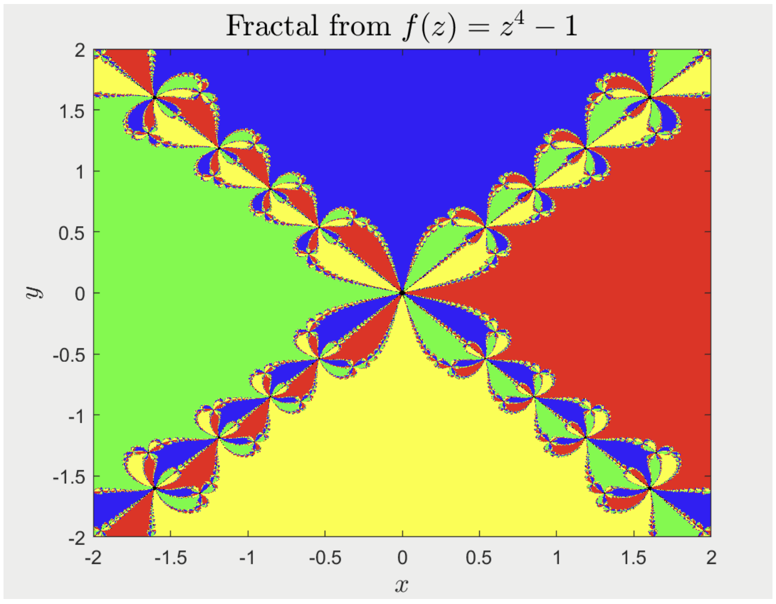
8 Problem 3.2

This problem, while similar to the previous one, uses a relation that states that for each root of a polynomial f, there exists a small disk around the root such that all points within the disk have the same color. To determine a suitable radius ε for this disk, one can visually inspect the picture. However, there are also explicit formulas to calculate ε. Joel Friedman, in his paper "On the Convergence of Newton's Method" (Theorem 2.2), provides an explicit formula for the radius of the disk:

$$r = \frac{\eta}{2d}$$

where η is the minimum distance between two roots of f, and d is the degree of f.

The program iteratively applies Newton's method to find the roots of the function $f(z) = z^4 - 1$, assigning a color to each grid point based on which root it converges to. Pictured in Plot 3.2 is the fractal form of the function $f(z) = z^4 - 1$ produced by this program:



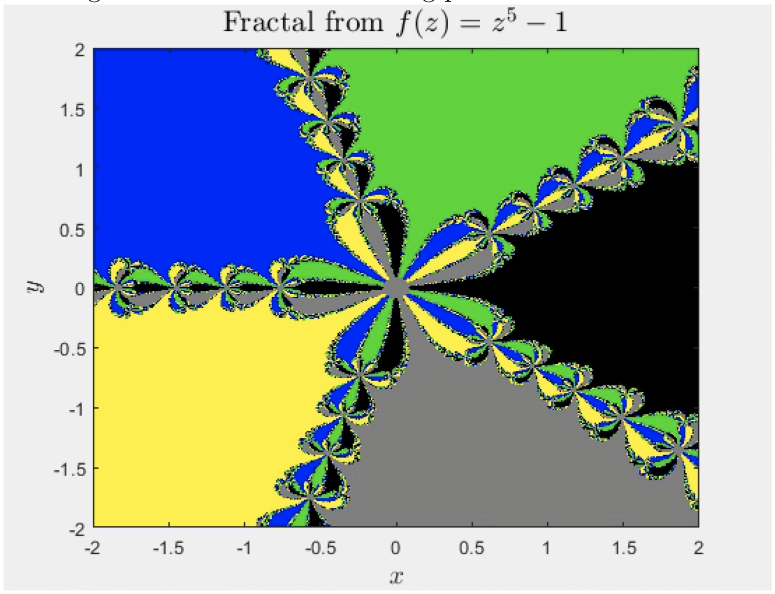
Plot 3.2: Fractal form $f(z) = z^4 - 1$.

9 Problem 3.3

Similarly, the solutions to $z^5 = 1$ can be expressed using the polar form of complex numbers:

$$z = \exp\left(\frac{2\pi n i}{5}\right)$$

where n is an integer ranging from 0 to 4. These solutions represent the five distinct fifth roots of unity. Each root is evenly distributed around the unit circle, with equal angular separation of $\frac{2\pi}{5}$ radians. Some of the code used to create the fractal form of the function $f(z) = z^5 - 1$ can be seen in Figure 3.3. Here is the resulting plot:



Plot 3.3: Fractal form $f(z) = z^5 - 1$.

10 Summary and Conclusions

This laboratory can essentially be summarized in two parts: the implementation of Newton's method and the application of Newton's method for fractals in the complex plane.

The implementation of Newton's method: Problem 2 investigated how one can find the roots of the function through a numerical process instead of an algebraic one. It interestingly shows the power of an iterative formula, allowing us to find approximate solutions using an initial guess. The section also analyzes the convergence of these solutions, showing us how for many functions Newton's method can effectively converge quadratically.

However, there are a handful of cases where this method will not work. For example, the function needs to be differentiable. Furthermore, the derivative of the function at the root cannot be zero since the derivative is found in the denominator of the iterative formula. Another case is if the derivative is very small (but not quite zero), and the approximate solutions given by the method may diverge to infinity ("Content - Newton's Method").

While the algorithm may only converge linearly for some functions, or not at all for others, the method presents a small window into the power of numerical analysis.

The application of Newton's method for fractals in the complex plane: Problem 3 investigates a more intricate, visual application of Newton's method. Fractal patterns provide a visual representation of the behavior of complex roots under iteration, offering insights into the dynamics of complex functions and numerical methods for root-finding.

In summary, the applications of Newton's method are extensive, going beyond just finding the roots of the function or creating fractals; the algorithm can be implemented to help find maximums, minimums, inflection points (Jenn).

11 Teamwork Statement

The work was split evenly between both team members for both the problems and the written analysis. The Abstract, Introduction, and the Conclusions were worked on by both partners. For the numerical findings, Erina focused mainly on problems 2.1-2.4. Alina worked through problems 3.1-3.3. Although the work was divided for the laboratory problems, both partners contributed in some way to each problem.

12 References:

Burton, Aaron. “Newton’s Method and Fractals.” Whitman.

“Content - Newton’s Method.” AMSI, Australian Mathematical Sciences Institute, 2015, amsi.org.au/ESA_Senior_Years/SeniorTopic3/3j/3j_2content_2.html:~:text=Continuing%20in%20this%20way%2C%20we,an%20equation%20-%20when%20it%20works.

Dominus, Mark. “More about Fixed Points and Attractors.” The Universe of Discourse, 19 July 2007, blog.plover.com/math/attractors-2.html:~:text=The%20Babylonian%20method%20for%20calculating,%20x2%20%2D%20n.).

“Fixed Point Iteration.” BYJUS, BYJU’S, 13 Apr. 2022, byjus.com/maths/fixed-point-iteration/:~:text=The%20fixed%20point%20iteration%20method%20uses%20the%20concept%20of%20a, g(x)%20%3D%20x.

Friedman, J. (1989). On the convergence of Newton’s method. Journal of Complexity, 5(1), 12–33. https://doi.org/10.1016/0885-064x(89)90010-1

Jenn. “Newton’s Method.” Calc Workshop, 22 Feb. 2021, calcworkshopi.com/derivatives/newtons-method/:~:text=Newton%27s%20Method%2C%20also%20known%20as,us%20to%20solve%20by%20hand.

“Newton’s Method.” UT Calculus, University of Texas, web.ma.utexas.edu/users/m408n/CurrentWeb/LM4-8 2.php: :text=In%20typical%20situations%2C%20Newton%27s%20method,answer%20good%20to%2030%2B%20digits. Accessed 9 Feb. 2024.

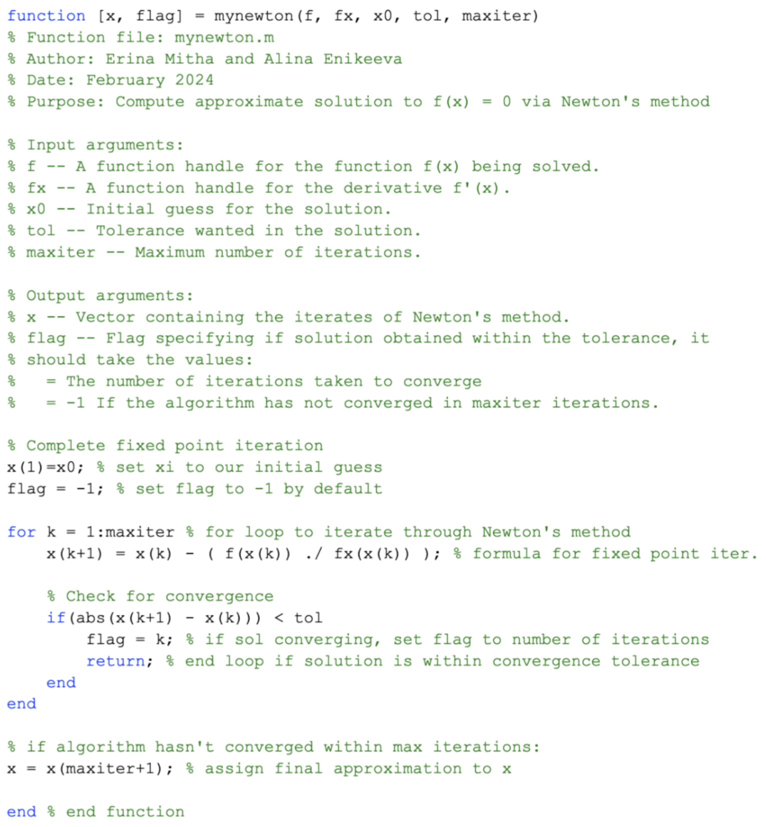
“Notes: Rates of Convergence.” Whitman.

Sauer, Tim. Numerical Analysis. Pearson, 2018.

Weisstein, Eric W. “Simple Root.” Simple Root, Mathworld—a Wolfram Web Resource, mathworld.wolfram.com/SimpleRoot.html. Accessed 9 Feb. 2024.

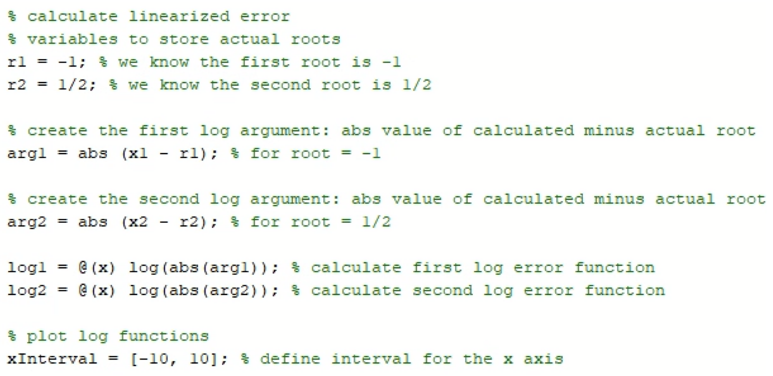
Wicklin, Rick. “The Babylonian Method for Finding Square Roots by Hand.” The DO Loop, SAS, 16 May 2016, blogs.sas.com/content/iml/2016/05/16/babylonian-square-roots.html.

13 Code Appendix



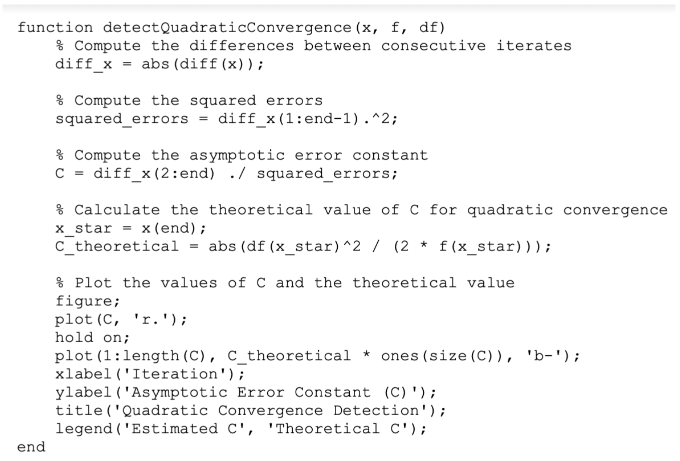
```
function [x, flag] = mynewton(f, fx, x0, tol, maxiter)
% Function file: mynewton.m
% Author: Erina Mitha and Alina Enikeeva
% Date: February 2024
% Purpose: Compute approximate solution to f(x) = 0 via Newton's method

% Input arguments:
% f -- A function handle for the function f(x) being solved.
% fx -- A function handle for the derivative f'(x).
% x0 -- Initial guess for the solution.
% tol -- Tolerance wanted in the solution.
% maxiter -- Maximum number of iterations.

% Output arguments:
% x -- Vector containing the iterates of Newton's method.
% flag -- Flag specifying if solution obtained within the tolerance, it
% should take the values:
%   = The number of iterations taken to converge
%   = -1 If the algorithm has not converged in maxiter iterations.

% Complete fixed point iteration
x(1)=x0; % set xi to our initial guess
flag = -1; % set flag to -1 by default

for k = 1:maxiter % for loop to iterate through Newton's method
    x(k+1) = x(k) - ( f(x(k)) ./ fx(x(k)) ); % formula for fixed point iter.

    % Check for convergence
    if(abs(x(k+1) - x(k))) < tol
        flag = k; % if sol converging, set flag to number of iterations
        return; % end loop if solution is within convergence tolerance
    end
end

% if algorithm hasn't converged within max iterations:
x = x(maxiter+1); % assign final approximation to x

end % end function
```

Figure 2.1: Code for Newton's Method.



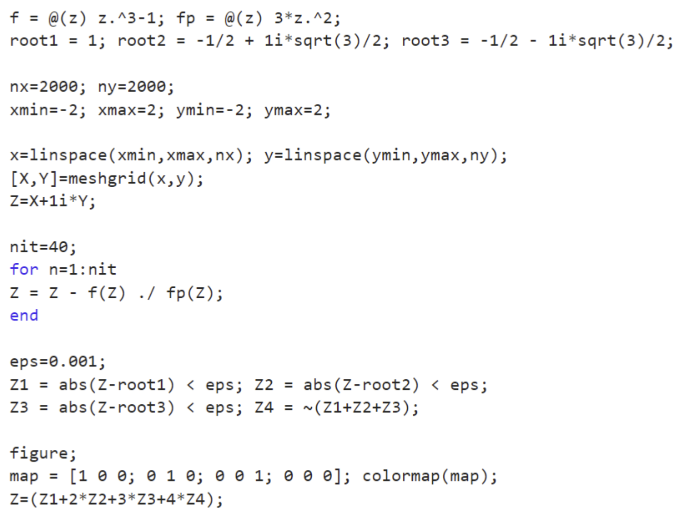
```
% calculate linearized error
% variables to store actual roots
r1 = -1; % we know the first root is -1
r2 = 1/2; % we know the second root is 1/2

% create the first log argument: abs value of calculated minus actual root
arg1 = abs (x1 - r1); % for root = -1

% create the second log argument: abs value of calculated minus actual root
arg2 = abs (x2 - r2); % for root = 1/2

log1 = @(x) log(abs(arg1)); % calculate first log error function
log2 = @(x) log(abs(arg2)); % calculate second log error function

% plot log functions
xInterval = [-10, 10]; % define interval for the x axis
```

Figure 2.3: Partial Code for the Linearized Convergence of Newton's Method for $f(x)$.



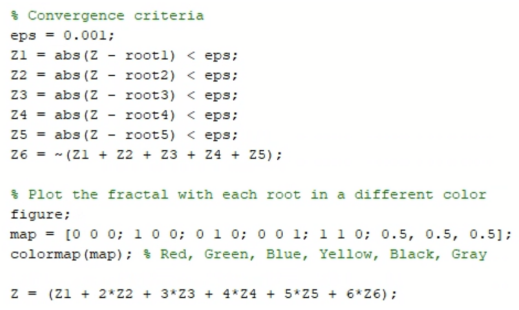
```
function detectQuadraticConvergence(x, f, df)
    % Compute the differences between consecutive iterates
    diff_x = abs(diff(x));

    % Compute the squared errors
    squared_errors = diff_x(1:end-1).^2;

    % Compute the asymptotic error constant
    C = diff_x(2:end) ./ squared_errors;

    % Calculate the theoretical value of C for quadratic convergence
    x_star = x(end);
    C_theoretical = abs(df(x_star)^2 / (2 * f(x_star)));

    % Plot the values of C and the theoretical value
    figure;
    plot(C, 'r.');
    hold on;
    plot(1:length(C), C_theoretical * ones(size(C)), 'b-');
    xlabel('Iteration');
    ylabel('Asymptotic Error Constant (C)');
    title('Quadratic Convergence Detection');
    legend('Estimated C', 'Theoretical C');
end
```

Figure 2.4: Partial code for the algorithm that detects quadratic convergence.



```
f = @(z) z.^3-1; fp = @(z) 3*z.^2;
root1 = 1; root2 = -1/2 + 1i*sqrt(3)/2; root3 = -1/2 - 1i*sqrt(3)/2;

nx=2000; ny=2000;
xmin=-2; xmax=2; ymin=-2; ymax=2;

x=linspace(xmin,xmax,nx); y=linspace(ymin,ymax,ny);
[X,Y]=meshgrid(x,y);
Z=X+1i*Y;

nit=40;
for n=1:nit
Z = Z - f(Z) ./ fp(Z);
end

eps=0.001;
Z1 = abs(Z-root1) < eps; Z2 = abs(Z-root2) < eps;
Z3 = abs(Z-root3) < eps; Z4 = ~(Z1+Z2+Z3);

figure;
map = [1 0 0; 0 1 0; 0 0 1; 0 0 0]; colormap(map);
Z=(Z1+2*Z2+3*Z3+4*Z4);
```

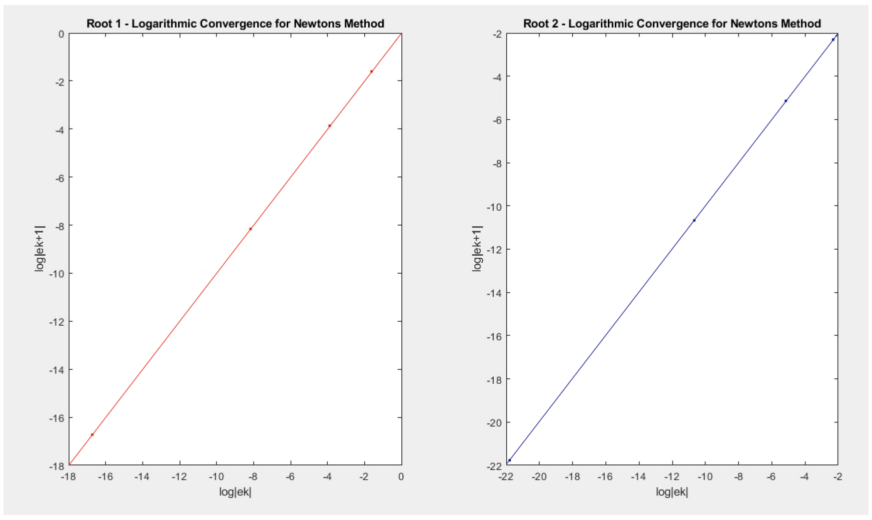
Figure 3.1: Partial code for the fractal form of $f(z) = z^3 - 1$.



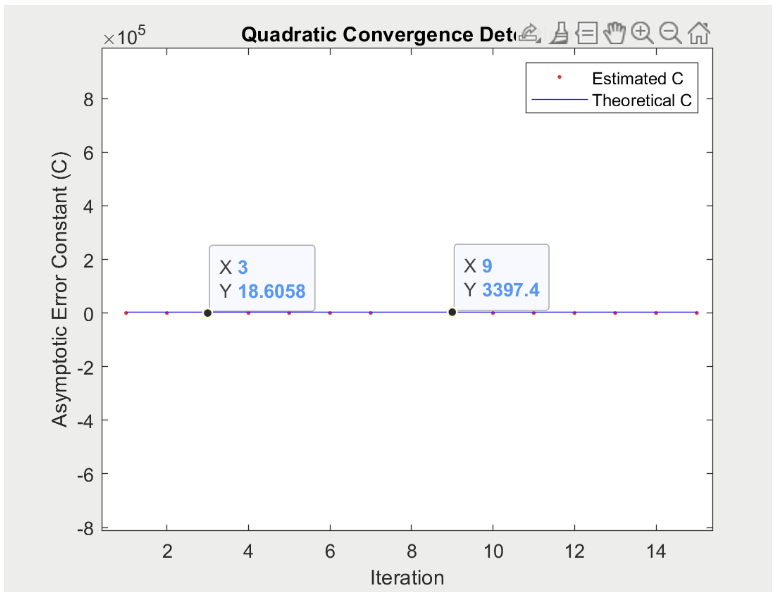
```
% Convergence criteria
eps = 0.001;
Z1 = abs(Z - root1) < eps;
Z2 = abs(Z - root2) < eps;
Z3 = abs(Z - root3) < eps;
Z4 = abs(Z - root4) < eps;
Z5 = abs(Z - root5) < eps;
Z6 = ~(Z1 + Z2 + Z3 + Z4 + Z5);

% Plot the fractal with each root in a different color
figure;
map = [0 0 0; 1 0 0; 0 1 0; 0 0 1; 1 1 0; 0.5, 0.5, 0.5];
colormap(map); % Red, Green, Blue, Yellow, Black, Gray

Z = (Z1 + 2*Z2 + 3*Z3 + 4*Z4 + 5*Z5 + 6*Z6);
```

Figure 3.3: Partial code for the fractal form of $f(z) = z^5 - 1$.

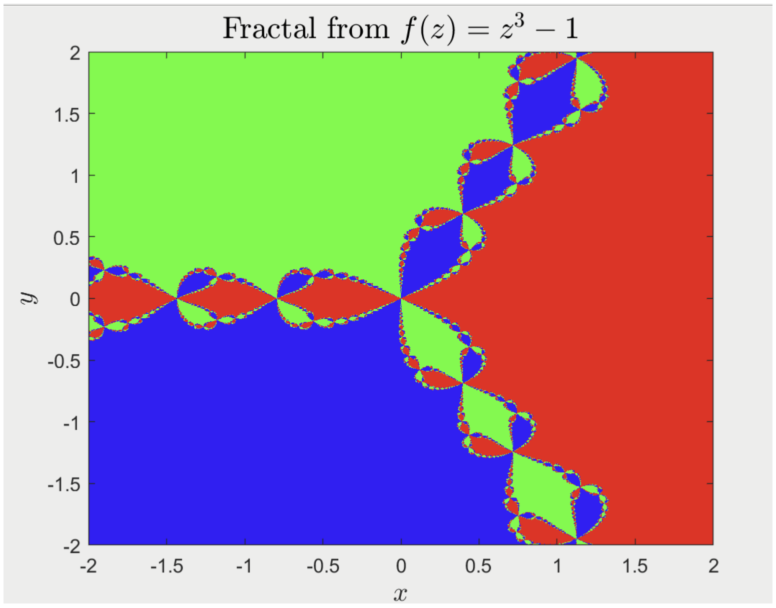
14 Plot Appendix



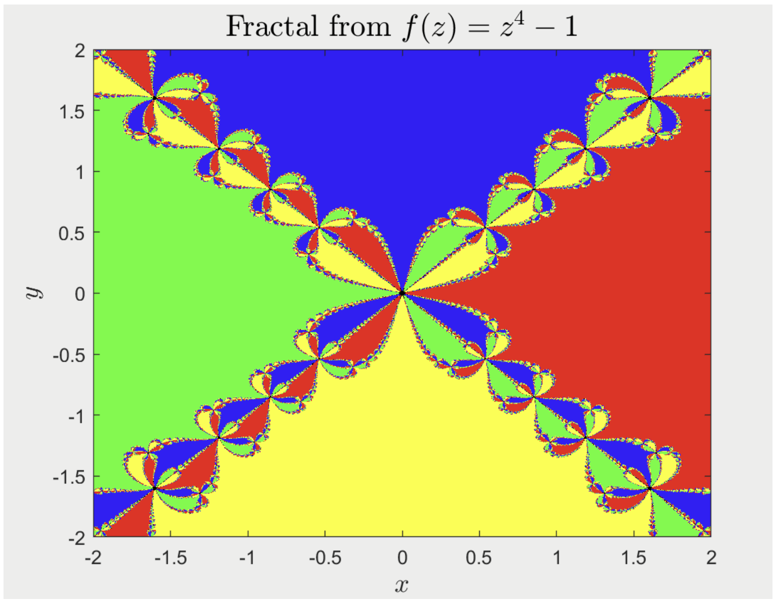
Plot 2.3: Plot for the Linearized Convergence of Newton's Method for $f(x)$.



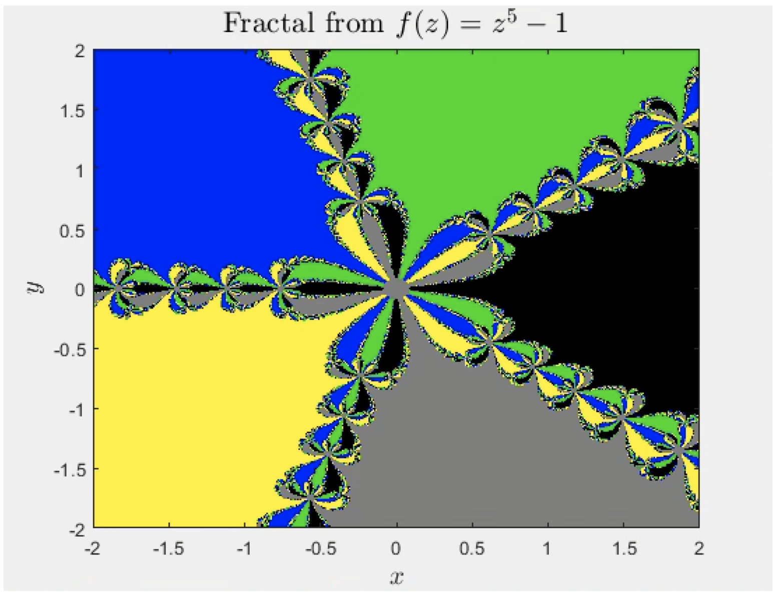
Plot 2.4: Quadratic convergence of Newton's method.



Plot 3.1: Fractal form $f(z) = z^3 - 1$.



Plot 3.2: Fractal form $f(z) = z^4 - 1$.



Plot 3.3: Fractal form $f(z) = z^5 - 1$.