AMS 595 Project 2: Plotting the Mandelbrot Set

Jane Condon

September 26, 2025

1 Introduction

In this project, we compute and plot the Mandelbrot set, which is a complex set of points for which the function:

$$f(z) = z^2 + c$$

remains bounded (instead of diverging). We use the following steps to accomplish this:

- Implement Mandelbrot Iteration Function
- Creating an Indicator Function for Boundary Detection
- Finding the Boundary Using Bisection Method
- Polynomial Approximation of the Boundary
- Plotting the Mandelbrot Set and Polynomial Fit
- Calculating the Boundary Length

2 Implementing Mandelbrot Iteration Function

In this section, we create a function called "fractal.m" to implement the Mandelbrot iteration function. This function takes complex number c as input and iterates $z \mapsto z^2 + c$ starting from $z = 0$. The function counts the number of iterations until $|z|$ exceeds 2 (stops at maximum 100 iterations). Note that points that do not diverge within the max number of iterations are considered to be part of the Mandelbrot set.

```
function it = fractal(c)
    % This function takes complex point c and
    % returns the number of iterations until divergence.

    max_it = 100; % Setting max number of iterations to 100
    z = 0; % Initializing iterated variable
    it = 0; % Initializing number of iterations variable

    while abs(z) <= 2 && it < max_it % Finding points belonging to the set
        z = z^2 + c; % Equation to generate Mandelbrot set
        it = it + 1; % Increasing iterations by 1
    end

    % If point is inside Mandelbrot set (no divergence),
    %  function returns 0
    if it == max_it % If we reach max iterations
        it = 0; % Number of iterations = 0
    end
end
```

Figure 1: Code used in the 'fractal.m' function.

3 Creating an Indicator Function for Boundary Detection

In this section, we create an indicator function along a vertical line at each x-value. The function returns 1 for points outside the Mandelbrot set (i.e., the point diverges), and -1 for points inside the Mandelbrot (i.e., the point does not diverge).

```
function fn = indicator_fn_at_x(x)
    % This function returns the indicator function at fixed x.
    fn = @(y) (fractal(x + 1i*y) > 0) * 2 - 1;
end
```

Figure 2: Code used to construct the indicator function.

4 Using Bisection Method to Find the Boundary

In this section, we implement the 'bisection' function (shown below), to find the y-coordinate where the indicator function changes sign.

```
function m = bisection(fn_f,s,e)
    % This function uses the bisection method to find the boundary point
    % where the indicator function fn_f changes sign. It takes a
    % function fn_f, and bounds s and e on the initial guess and returns
    %  a point where the point sign of f changes.

    tol = 1e-6; % Setting tolerance level to 1e-6
    max_iter = 100; % Setting max number of iterations to 100


    for k = 1:max_iter % Looping from 1-100 iterations
        m = (s + e)/2; % Calculating midpoint
        % Checking if the sign of fn_f changed between the
        %  [s, m] interval:
        if fn_f(s) * fn_f(m) < 0 % Boundary between s and m
            e = m; % Shrink the interval to [s,m]
        else % Boundary between m and e
            s = m; % Shrink interval to [m,e]
        end
        % If the interval becomes smaller than tolerance level,
        % function stops.
        if abs(e - s) < tol
            break;
        end
    end
end
```

Figure 3: Code used to construct the bisection function.

We apply this method along 1000 vertical lines which lie in the interval $x \in [-2, 1]$ to collect both upper and lower boundary points.

```
% Collecting the boundary points along the Mandelbrot set

% For x- values, we find the corresponding y-value where
% the boundary of the Mandelbrot set occurs, using the
% bisection method on vertical lines.

x_values = linspace(-2, 1, 1000); % Generating 1000 x-values in [-2, 1] in
y_values_upper = zeros(size(x_values)); % Initializing array for upper bou
y_values_lower = zeros(size(x_values)); % Initializing array for lower bou

for i = 1:length(x_values)
    x = x_values(i); % Picking one x-value

    % Creating an indicator function for this x-value. The function
    % should take a y-value and return:
    %  1 if point is outside the Mandelbrot set
    % -1 if point is inside the Mandelbrot set
    fn = indicator_fn_at_x(x);

    % Applying the bisection method to find the boundary along this
    % vertical line.

    y_values_upper(i) = bisection(fn, -1.5, 1.5); % Upper boundary
    y_values_lower(i) = bisection(fn,1.5,-1.5); % Lower boundary
end
```

Figure 4: Code used to construct the bisection function.

5 Polynomial Approximation of the Bound

In this section, we implement the following steps:

- Select a subset of valid x-values to focus on the main fractal region.
- Fit a 15th degree polynomial to the boundary points using polyfit.
- Evaluate the polynomial at dense x-values to obtain a curve representing the fractal boundary.

```
valid_idx = (x_values > -1.5 & x_values < 0.5); % Setting range of x-values

x_fit = x_values(valid_idx); % Real parts of c to fit
y_fit_upper = y_values_upper(valid_idx); % Imaginary parts of c to fit (upp
y_fit_lower = y_values_lower(valid_idx) ; % Imaginary parts of c to fit (lo

% Fitting a 15-th order polynomial to the boundary points

p_upper = polyfit(x_fit, y_fit_upper, 15); % Upper boundary
p_lower = polyfit(x_fit, y_fit_lower, 15); % Lower boundary

% Evaluating polynomial fit at a dense set of x-values
% Gives a smooth curve so that we can compare with raw values

x_x = linspace(min(x_fit), max(x_fit), 500); % 500 points in same range
y_y_upper = polyval(p_upper, x_x); % Evaluating polynomial at 'x_x'
y_y_lower = polyval(p_lower,x_x); % Evaluating polynomial at 'x_x'
```

Figure 5: Code used to implement the steps above.

6 Calculating the Boundary Length

In this section, we implement the function poly_len(p,s,e) to calculate the curve length using the following formula:

$$l = \int_a^b \sqrt{1 + \left(f'(x)\right)^2}\, dx$$

```
function l = poly_len(p, s, e)
    % This function computes the approximate length of a polynomial curve
    % y = f(x) over the interval [s,e].

    % Function takes the following inputs:
    % p: polynomial coefficients from polyfit function
    % s: starting x value (left bound)
    % e: ending x-value (right bound)

    % Function returns the following output:
    % L: total arc length of the curve y = f(x) on [s,e]

    % Uses the following formula:
    % Length = ∫ sqrt(1 + (dy/dx)^2) dx   from x = s to e


   % First, get derivative of the polynomial f(x)

    dp = polyder(p); % Derivative coefficients

    ds = @(x) sqrt(1 + (polyval(dp, x)).^2);
    l = integral(ds, s, e);
end
```

Figure 6: Code used to construct the poly_len function.

We obtain a curve length of approximately 5.9211.

```
Approximate boundary length: 5.9211
>>
```

Figure 7: Output returned by the main script.

7 Visual Representation

In this section, we plot the raw boundary points and the fitted polynomial curves (for the upper and lower bounds). We overlay this over a plot of the Mandelbrot set to visualize the accuracy of the methods used in this project.

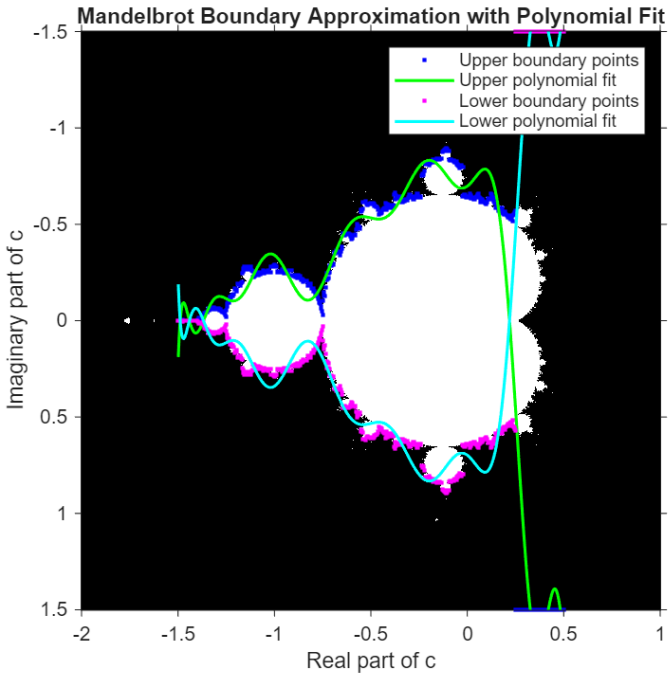

Figure 8: Plot of the Mandelbrot set with calculated boundary points and boundary curve.

8 Final Thoughts

Based on the plot above, our estimate of the curve length is not as accurate as we'd like it to be. We've underestimated the length a little bit. One possible reason for this is overfitting. High-degree polynomials, such as the 15th-degree polynomial used here, tend to under or overestimate the true curve, leading to overfitting. Another possible reason is the boundary points chosen: it appears in the plot that some extreme outliers have been chosen, which ties into the overfitting problem. To get a more accurate estimate, we may need to generate more boundary points or adjust the polynomial (such as using several lower-degree polynomials, also known as splines, instead of a high-degree polynomial). This makes sense in the context of the coastline paradox, which states that if you measure the length of coastline using smaller units, you can capture the complexity of the coastline, resulting in a larger measurement.