Mandelbrot Set Fractal
David Hwang

Github link

Introduction

The mathematician Benoit Mandelbrot invented the field of fractal geometry after considering the coastline paradox. If you try to measure the coast of Britain, the finer the measurement you make, the longer the coast becomes. You end up trailing around every grain of sand. In this report, I will approximate the rough circumference of a Mandelbrot Set fractal using a function approximation. The report proceeds as follows:

• Introduce a function computing the fractal
• Use the bisection algorithm to approximate the boundary of the fractal
• Use a polynomial function approximation to find the boundary as a function
• Integrate the boundary curve to find its length

Fractal Function

The Mandelbrot Set Fractal is a set ot points; some 2D points belong to it and some don't. To test whether a point belongs, we put the point inside the fractal function, $f_c(z) = z^2 + c$ where c is a constant, and keep reapplying the formula as follows: $z = f_c(f_c(f_c(f_c(\dots f_c(0)\dots))))$ with starting $z = 0$. If $|z|$ stays bounded, then the complex point c belongs to the Mandelbrot Set. For this project, if $|z| < 2.0$ during the 100 iterations, I consider z is in the set. Considering that, I built a function, *it = fractal(c)*, that determines whether an imputed point, *c*, is in the set or not. As you can see in the code, this function returns 100 when the point is in the set. This is straightforward because it means that the inputted point stays under 2.0 after 100 iterations. Conversely, if the point is not in the set, the returned value means the number of iterations it takes to overcome 2.0. Of course, the value will be less than 100, and it depends on the speed at which it diverges.

```
%%%%%%%%%%%%%%%%%%%%%%%
%% Mandelbrot Fractal
%%%%%%%%%%%%%%%%%%%%%%%
% function fractal(), returns the number of iterations till divergence % c:
complex value
% Considering a point that does not diverge in 100 iterations is in the set
function it = fractal(c)
   z = 0; % initial z-value
   n = 0; % counter of number of iterations
   while abs(z)<=2 && n<100
       n = n+1;
       z = z^2 + c;
   end
   it = n;
end
```

Bisection algorithm to approximate the boundary of the fractal

We have built the function that enables us to determine the point's existence inside the Fractal Set. However, our purpose is to approximate the rough circumference of a Mandelbrot Set fractal. So finding the boundary point of the set for each x is important. Thus, in this part, we introduced two different functions that help to find the boundary point.

First, we introduce a function, *indicator_fn_at_x()*, that tells us whether the point is divergent or convergent at a given x. This is the extension of the last part. By receiving the number of iterations of the point from the *fractal()* function, we return 1 for the divergence and -1 for the no divergence. Furthermore, this function receives x(real value) as input and returns a function that depends on y(imaginary value), enabling us to scan a vertical line to a given x.

```
%%%%%%%%%%%%%%%%%%%%%%%
%% Bisection Method
%%%%%%%%%%%%%%%%%%%%%%%
% function indicator_fn_at_x(), returns a function that determines whether the
point diverges or not along a vertical line at a given x
function fn = indicator_fn_at_x(x)
   fn = @(y) (fractal(x + 1i*y) < 100)*2 - 1; % setting 1 for divergence and
-1 for no divergence
end
```

Second, we introduce a function that enables us to quickly find the boundary point at a given x. By taking the *indicator_fn_at_x()* function as an input, we scan from the given lowest y value to the greatest y value, and return the boundary y value of the Fractal Set at a given x value.

```
% function bisection(), returns a point where the point sign of f changes
% approximate the boundary of the fractal
function m = bisection(fn_f, s_y, e_y)
   y = linspace(s_y,e_y,10^3); % vertical vector of 100 points from s to e
   s_idx = 1; % lower bound point index, y
   e_idx = length(y); % upper bound point index, y
   while s_idx + 1 < e_idx
       m_idx = floor((s_idx+e_idx)/2); % get an integer mid value
       if fn_f(y(m_idx)) == 1
           e_idx = m_idx;
       else
           s_idx = m_idx;
       end
   end
   m = (y(s_idx)+y(e_idx))/2;
end
```

And then, we can simulate the functions that we have built to see their performance as below. Note that since the fractal is symmetric about the real axis (the x-axis), we set the range of y to start from 0 to 1.2. This means the boundary points that we get are also symmetric to the x-axis in the plane. Thus, we do not need to find the boundary points below the x-axis, but we should remember that when we calculate the length of the boundary. As one example of the result of the MATLAB code below, we get the boundary point `(-1.774775,0.004204)` for a given x = -1.774775. From the simulation below, we can find the fractal boundary for $x \in [-2, 1]$ for at least 10^3 points.

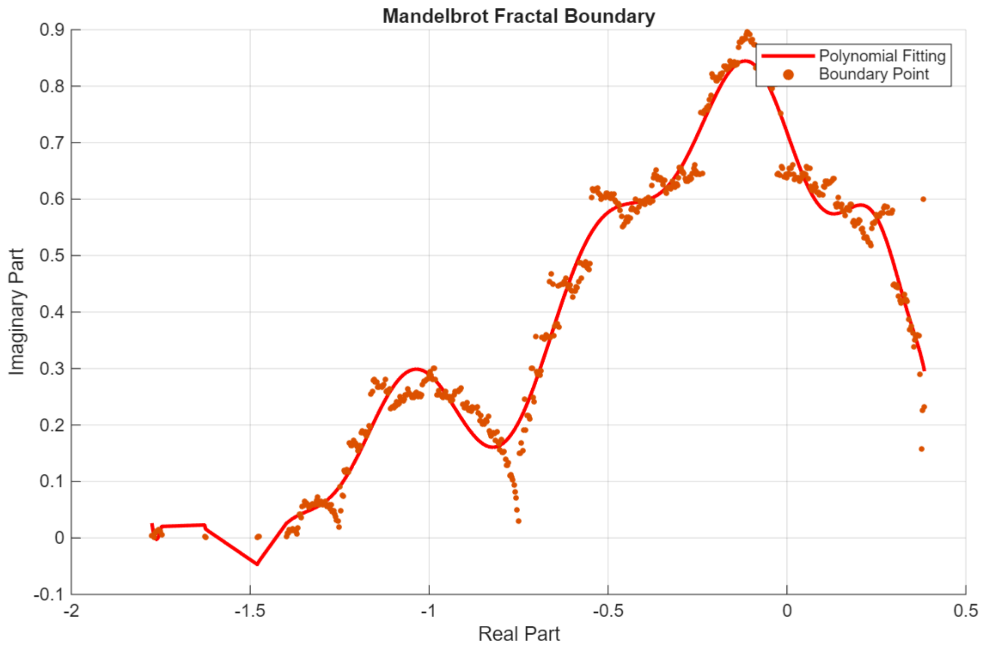
```
s_x = -2; e_x = 1; % x belongs to [-2,1]
s_y = 0; e_y =1.2; % can be changed; arbitrary assigned value;
% The Mandelbrot set is symmetric about the real axis (the x-axis). When
searching for the upper boundary, you only need to search on one side (e.g.,
y≥0).

x = linspace(s_x,e_x,10^3); % horizontal vector of 10^3 points for [-2,1]
y = zeros(1,length(x));
for i=1:length(x)
   fn_f = indicator_fn_at_x(x(i));
   m = bisection(fn_f,s_y,e_y);
   fprintf('(%f,%f)\n',x(i),m); % print the boundary point
   y(i) = m; % put boundary imagine value into y vector
end
```

Polynomial function approximation to find the boundary as a function

One of the most fundamental examples of function approximation (known today as machine learning) is least squares polynomial fitting. The idea is to take a polynomial $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$ and choose such a set of values for $(a_n, a_{n-1}, \cdots, a_1, a_0)$ which best fits the data. To find the best fit coefficients, we take the sum of squares of the distance between the approximation and each data point as the measure. Since MATLAB has its own function for this method, finding the best-fit coefficients is not difficult. We use a polynomial of order 15, and note that we only need to choose the points along the actual fractal. In the previous simulation, we set the range of $x \in [-2, 1]$, and found that there are some x values not included in the fractal set. Thus, we need to sort out those points by hand-tuning the range of x. Below, we first filter the range of x to find the best fit coefficients, and plot the polynomial function with the boundary point to see the performance of the polynomial function.

```
%%%%%%%%%%%%%%%%%%%%%%%
%% Polynomial function fitting
%%%%%%%%%%%%%%%%%%%%%%%
% Note: x, y are the data points represeting the fractal boundary
% Discard the left and right points of the fractal
% hand tuning
indices = find(y>0.001);
% Remove the indices corresponding to the discarded points
x_filtered = x(indices);
y_filtered = y(indices);
scatter_y = y_filtered; % original data point of y before putting into the
polynomial
order = 15; % set the order of polynomial to 15
p = polyfit(x_filtered,y_filtered,order); % p: set of coefficients;
y_filtered = polyval(p,x_filtered); % evalueate the polynomial using new x
value
figure(1); clf;
hold on;
plot(x_filtered, y_filtered, 'r-', 'LineWidth', 2); % Plot the polynomial fit
scatter(x_filtered, scatter_y, 10, 'filled')
xlabel('Real Part');
ylabel('Imaginary Part');
legend('Polynomial Fitting','Boundary Point')
title('Mandelbrot Fractal Boundary');
grid on;
```

As a result of the code, we get the figure of the upper x-axis part of the Fractal Boundary. In the plane, the x-axis is the real value, and the y-axis is the imaginary part of the point. Also, the red line stands for the polynomial function that we created, and the orange dots stand for the boundary points. From the figure, we can confirm that the polynomial function fits quite well with the boundary points. Although, for some part, like when x=-0.75, the polynomial line does not perfectly match with the boundary dot point, we know there has to be a limitation due to the use of a polynomial model with order 15. There would be two reasons for this. The first is that the polynomial function may not have the property to perfectly describe the fractal. The possible second reason is that the order of 15 may not be enough or too much to describe the fractal.



Integrate the boundary curve to find its length

Lastly, we need to calculate the length of the polynomial function that we created to approximate the rough circumference of a Mandelbrot Set fractal. From the last part, we got the polynomial function that describes the upper x-axis part of the fractal. Therefore, to get the total length of the fractal, first, we need to calculate the length of the polynomial function and multiply it by 2 to consider the lower x-axis part of the fractal. We create a function, *poly_len()*, to integrate the polynomial function, and finally multiply the result by 2. As a result, we get the rough total length of the fractal(including the upper and lower parts of the x-axis): **6.4810**

```
%%%%%%%%%%%%%%%%%%%%%%%
%% Boundary Length
%%%%%%%%%%%%%%%%%%%%%%%
% function poly_len(), returns the curve length of the polynomial l
% find boundary as a function
function l = poly_len(p, s_x, e_x)
   p_prime = polyder(p); % derivative of the polynomial; coefficients of df/dx
   ds = @(x) sqrt(1 + (polyval(p_prime,x)).^2);
   l = integral(ds,s_x,e_x); % return integration value
end
% Adjust the range of x to consider the filtered part
s_integrate = min(x_filtered);
e_integrate = max(x_filtered);
l = poly_len(p, s_integrate, e_integrate);
% since fractal is symmetry to that x-axis, to get the total boundary length
we mulitpy 2 to the l
final_length = l*2;
```

Reference

You, Chenyu. MATLAB Project 2. 2025.

“Mandelbrot Set.” Wikipedia, Wikimedia Foundation, 2025, https://en.wikipedia.org/wiki/Mandelbrot_set. Accessed 5 Oct. 2025.