Important Note

You must write your code independently. Do not copy the code from any source. To receive full credit, your code must be well-documented with comments.

1 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 assignment, you'll approximate the rough circumference of a Mandelbrot Set fractal using function approximation. You'll proceed as follows:

- write a function computing the fractal
- use the bisection algorithm to approximate the boundary of the fractal
- use polynomial function approximation to find boundary as a function
- integrate the boundary curve to find its length

2 Project Requirements

Due Date

Sunday, 11:59 PM EST, October 5th, 2025

- 1-2 page report, includes:
  - work done
  - results
  - discussion of results
  - 500 words of text max (excluding figure captions)
- Link to your Github repository
- One PDF file with commented, executable code

3 Implementation Requirements

- function `it = fractal(c)` which takes complex `c` and returns the number of iterations till divergence
- function `m = bisection(fn_f, s, e)` which takes a function `fn_f`, bounds `s` and `e` on the initial guess and returns a point where the point sign of `f` changes
- find the boundary, find the polynomial approximating the boundary

- function `l = poly_len(p, s, e)` which takes the polynomial `p`, starting and ending points `s`, `e` and returns the curve length of the polynomial `l`

4 General Hints

Package your work into a script with each of the parts separate.

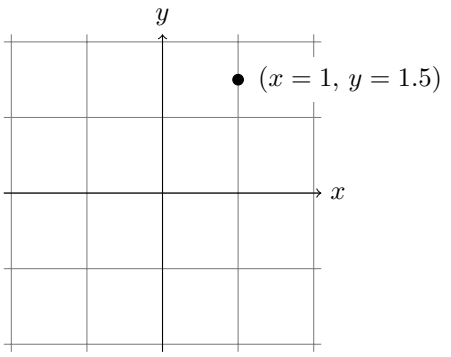
Plotting the Mandelbrot set can be extremely useful. If you form a matrix where each element corresponds to `it`, then you can visualize it using `imshow`.

5 Mandelbrot Fractal `it = fractal(c)`

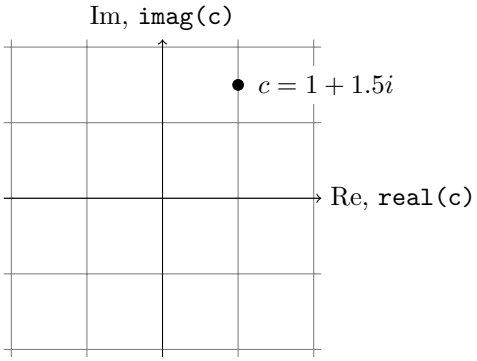
Useful read: `https://en.wikipedia.org/wiki/Mandelbrot_set`

5.1 Complex numbers as coordinates

We can represent a plane, $\mathbb{R}^2$ with coordinates x and y.



But it's also possible to use complex numbers where `real(c)` corresponds to the coordinate x and `imag(c)` corresponds to the coordinate y.



5.2 Mandelbrot Set

Mandelbrot Set is a set of points, some 2D points belong to it and some don't. Testing whether a point belongs, simply take

$$f_c(z) = z^2 + c \tag{5.1}$$

and, keeping c constant and starting with $z = 0$, keep reapplying the formula like so

$$z = f_c(f_c(f_c(f_c(...f_c(0)...)))) \tag{5.2}$$

if $|z|$ stays bounded, then the complex point c belongs to the Mandelbrot Set.

5.3 Numerical treatment of the Mandelbrot Set

For our applications, we won't try to determine whether the point c diverges, but rather whether z grows large quickly. We'll also want to determine how quickly.

Any point c for which $|z| > 2.0$ before 100 iterations is not in the set.

5.4 Implementation

Write a function that returns how many iterations it takes for a point to diverge ($|z| > 2.0$).

Name:

- `it = fractal(c)`

Inputs:

- `c` complex point c (representing coordinates)

Outputs:

- `it` the number of iterations after which $|z| > 2.0$

Use equation (5.1).

6 Bisection Method `m = bisection(fn_f, s, e)`

6.1 Binary Search

Imagine you have a sorted vector.

```
v = [2 3 5 7 11 13 17 19 23]
```

How do you find the position of a number greater than or equal to 17 quickly?

Algorithm 1 Binary Search Algorithm

function BINARY_SEARCH(el, V)

$V \leftarrow$ vector of elements

$el \leftarrow$ element to find

$s\_idx \leftarrow$ index of first element of V

$e\_idx \leftarrow$ index of last element of V

while e_idx more than 1 away from s_idx **do**

$m\_idx \leftarrow$ average of s_idx and e_idx

if V at m_idx is greater than el **then**

$e\_idx \leftarrow m\_idx$

else

$s\_idx \leftarrow m\_idx$

return average of s_idx and e_idx

6.2 Implementation

Use the idea from the binary search algorithm to write a binary section algorithm where you find the point at which an indicator function switches sign.

You'll need to write an indicator function which gives positive values when the point is outside of the set and negative values when it is in the set.

You cannot scan the entire 2D space at once. You'll need to focus on a line. The easiest way to implement an indicator function is using an anonymous function generator looking something like this.

```
function fn = indicator_fn_at_x(x)
  % returns an indicator function along a vertical line at a given x
```

```
  % it uses equivalence of logical variables (true - 1, false - 0)
  % to produce a value of 1 for divergence and -1 for no divergence

  fn = @(y) (fractal(x + 1i * y) > 0) * 2 - 1;
end
```

Implement a bisection function that finds the point on the boundary of the fractal.

Name:

- `m = bisection(fn_f, s, e)`

Inputs:

- `fn_f` indicator function for a particular x
- `s` lower bound point, y
- `e` upper bound point, y

Outputs:

- `m` the boundary point of a fractal

The upper bound has to be chosen above the fractal. The lower bound has to be chosen within the fractal, $y = 0$ would do fine.

Find the fractal boundary for $x \in [-2, 1]$ for at least 10^3 points.

7 Polynomial function fitting

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 $(a_n, a_{n-1}, \ldots, a_1, a_0)$ which best fits the data (taking the sum of squares of the distance between the approximation and each data point as the measure).

Turns out, choosing those a_i isn't that difficult and MATLAB has a function for it.

```
p = polyfit(x, y, order);
```

where `p` is the set of coefficients a_i, `x`, `y` are the data points and `order` is the order of the polynomial (e.g. 1 for linear and 2 for quadratic).

Once we have the coefficients, we can evaluate the polynomial at new x values using

```
y = polyval(p, x);
```

where the parameters are as before and `x` can be a vector, making `y` a vector.

7.1 Implementation

Implement a script (not a function) that takes your fractal boundary points found in the previous part and fits a polynomial of order 15 to them.

Fit a polynomial of order 15 to your fractal boundary data points.

Remember to select only the points along the actual fractal. The boundary goes flat to the left and right of the fractal. Discard those points (by hand tuning the range of `x`). Plotting is very useful here.

8 Boundary Length `l = poly_len(p, s, e)`

Now that you have the approximate fractal boundary, you'll have to find its length.

8.1 Curve Length

The curve length of a function is given by the formula for s, the path length

$$l = \int_a^b \sqrt{1 + \left(\frac{df}{dx}\right)^2}\, dx$$

Polynomials of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x_1 + a_0$$

have a very easy derivative $\frac{dy}{dx}$ equal to

$$\frac{df}{dx} = a_n n x^{n-1} + a_{n-1}(n-1) x^{n-2} + \cdots + a_1$$

so, the curve length derivative is

$$ds = \sqrt{1 + \left(\frac{df}{dx}\right)^2}$$

which can be directly used in

$$l = \int_a^b ds$$

8.2 Numerical Integration

Numerical integration is a well established branch of mathematics and it's typically a bad idea to try and reimplement well established algorithms.

8.3 Implementation

Write a function that uses MATLAB's `integral` or Octave's `quad` to compute the curve length of a polynomial. Apply it to the fractal boundary. You'll need to write an anonymous function `ds` with only `x` as its arguments.

Name:

- `l = poly_len(p, s, e)`

Inputs:

- `p` fitted polynomial coefficients
- `s` left bound on x
- `e` right bound on x

Outputs:

- `l` the curve length of the polynomial

Use your function to find the approximate length of the fractal. Be very careful about bounds, fitted polynomials have a tendency to misbehave outside of the fitted range.