I have seen the Mandelbrot set in the and knew it dealt with math in its creation, but never gave it much thought. I am glad we have a chance to play with it before the end of the semester. I will break down the code and try to explain how it works in coming up with what it does:

```
def mandelbrot(c, max_iter):
    z = 0
    n = 0
    while abs(z) <= 2 and n < max_iter:
        z = z*z + c
        n += 1
    return n
```

The mandelbrot routine is run for the given number of iterations or unless the complex number ever becomes greater than or equal to 2, which means it diverges.

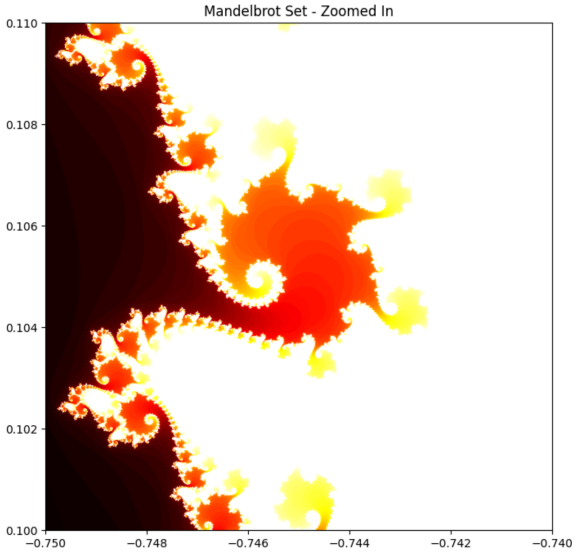
```
for i in range(width):
    for j in range(height):
        n3[i, j] = mandelbrot(r1[i] + 1j * r2[j], max_iter)
```

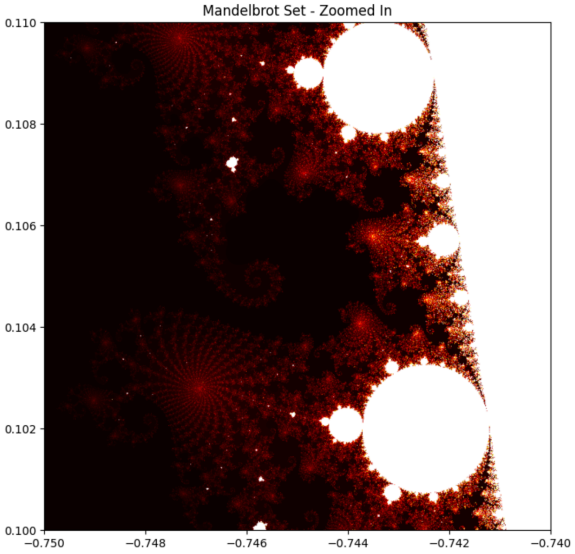
Here we define a point on the complex plane i, j with r1[i] serving as the coefficient of our complex number and r2[j] serving as the real number.

```
# Parameters for the plot
xmin, xmax, ymin, ymax = -2.0, 1.0, -1.5, 1.5
width, height = 800, 800
max_iter = 100
```

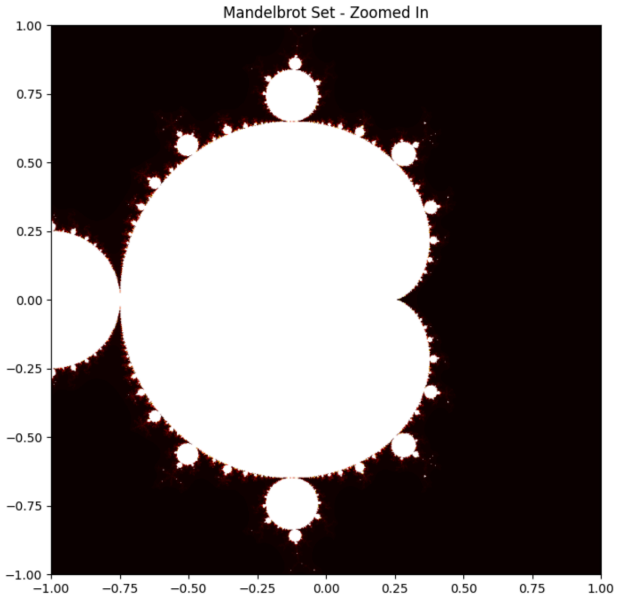
```
# Generate the Mandelbrot Set
n3 = mandelbrot_set(xmin, xmax, ymin, ymax, width, height, max_iter)
# Parameters for the initial zoom level
xmin, xmax, ymin, ymax = -0.75, -0.74, 0.1, 0.11  # Zoom into this specific area
width, height = 800, 800  # Resolution of the image
max_iter = 1000  # Number of iterations
```

The standard settings used to define the grid. It can be changed to focus on a smaller or larger field of the fractal. The max_iter sets how many times the calculation is completed and plotted. You can see the difference between setting the value from 100 (left) to 5000 (right):





Zooming out the fractal with minimum and maximum values from -1 to 1, you see the whole shape:

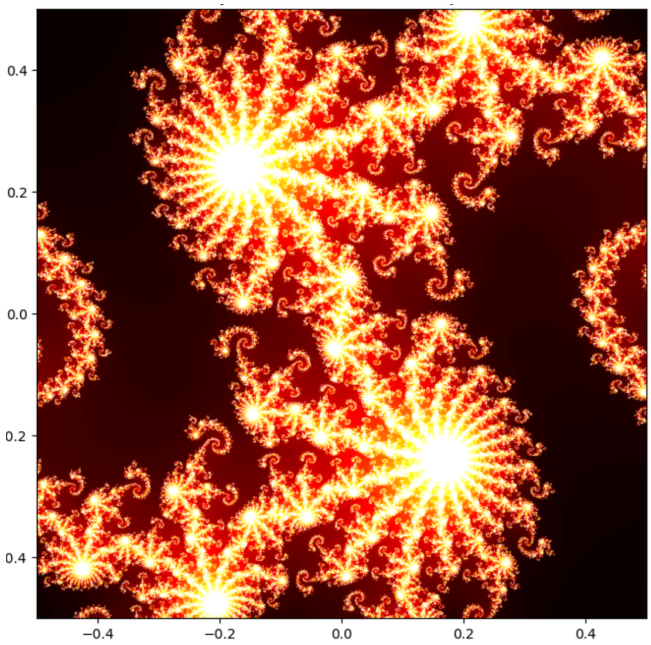


I should mention that the color coding represents the number of times values are repeated in the product of the complex numbers when the series does not diverge. The lighter the color, the more frequent those values appear.

Looking at the Julia set, the function works similarly, with the biggest difference being that the c value, rather than being derived each time, is instead set to a defined constant.

```
# Choose a value for the complex constant c
c = complex(-0.8, 0.156)
```

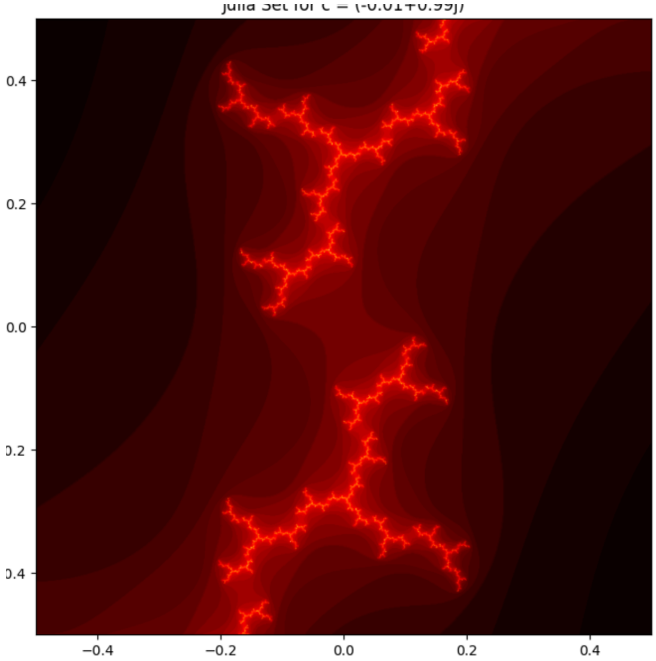
It is still important that the values do not diverge, so numbers between -1 and 1 are necessary. The first thing I did was zoom in on the fractal to see it in more detail:



Minimizing the image here does not do it justice, but you can somewhat appreciate the repeated branching patterns created. Lastly, I want to play with the constant to see how if affects the image.

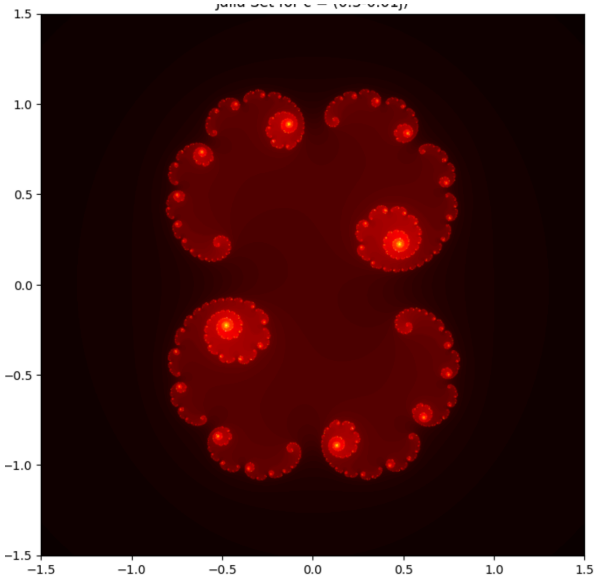
Playing with extreme values is not particularly noteworthy:

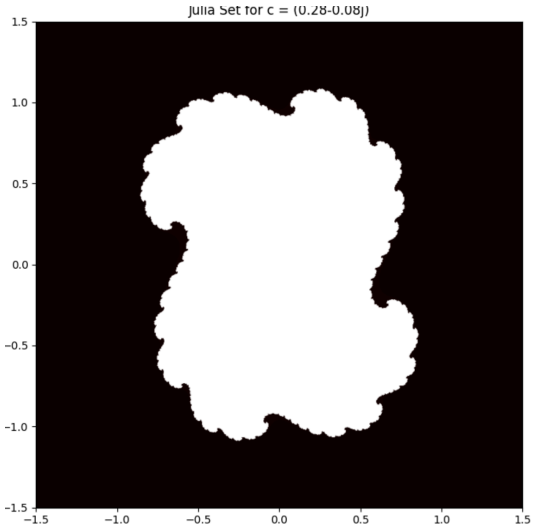
```
# Parameters for the plot
#xmin, xmax, ymin, ymax = -1.5, 1.5, -1.5, 1.5
xmin, xmax, ymin, ymax = -.5, .5, -.5, .5
width, height = 800, 800  # Image resolution
max_iter = 256  # Number of iterations for accuracy

# Choose a value for the complex constant c
#c = complex(-0.8, 0.156)
c = complex(-0.01, 0.99)
```



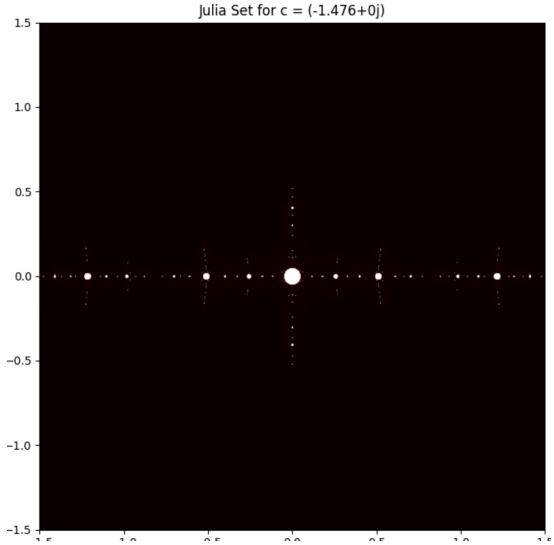
It seems finding the best patterns either involves researching them or trial and errors:

```
# Parameters for the plot
#xmin, xmax, ymin, ymax = -1.5, 1.5, -1.5, 1.5
xmin, xmax, ymin, ymax = -1.5, 1.5, -1.5, 1.5
width, height = 800, 800  # Image resolution
max_iter = 256  # Number of iterations for accuracy

# Choose a value for the complex constant c
#c = complex(-0.8, 0.156)
c = complex(0.3, -0.01)
```






```
# Choose a value for the complex constant c
#c = complex(-0.8, 0.156)
c = complex(0.28, -0.08)
```




```
# Choose a value for the complex constant c
#c = complex(-0.8, 0.156)
c = complex(-1.476, 0)
```