Miles Honsberger

Project 2 Report

MAT 331

11/10/22

Comparison of the Algorithms

f2(c,N) - generating the points of the Julia set

The function in algorithm 2 was used for listing the points in the Julia set, all of which collectively equal the non-attractive fixed point, z_o. In using this function, we refer to the function f1 to generate a non-attractive fixed point using an input of some complex number. The list generated in algorithm 2 shows us the points in the Julia set that ultimately lead to the point z_o.

```
f2(complex(2,1),5)

[(0.8643929452353858-1.372145115699254j),
 (1.1667171316871796+1.0290309416730703j),
 (0.9803252676417578-1.071181354590485j),
 (1.03204002388382+1.006673658469344j),
 (0.9963398046186616-1.0140972465857063j)]
```

Figure 1: Algorithm f2, with a complex number 2+i and N = 5. This shows the set of points in the Julia set $J_5(f) = f^{\circ 5}(c) = z_o$. The five points in the Julia set leading to z_o.

A graphical representation of this process looks slightly different, as it shows direction of the points in the Julia set ultimately leading to the point z_o.

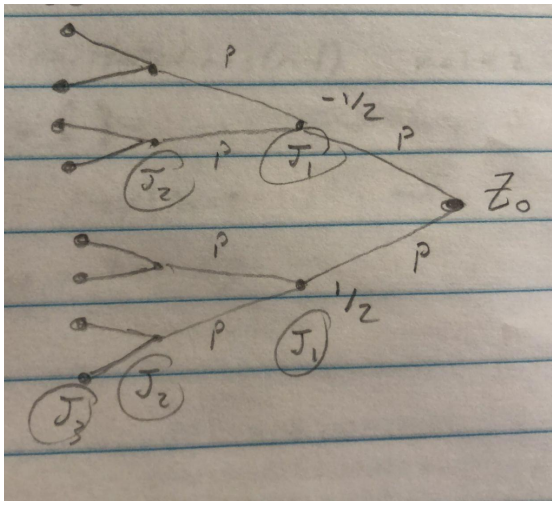

Figure 2: Graphical representation of algorithm f2, each subsequent N^{th} iteration of the Julia set is represented as a branch from the original non-escaping fixed point defined in the first algorithm

Plt.imshow - generating an image of the non-escaping fixed point created in f1
By creating an image with a color gradient and a graphical representation over the entire image, we can better see the nature of the complex numbers. If we take the graphical representation of a scatter plot for the same point we get an image of such:

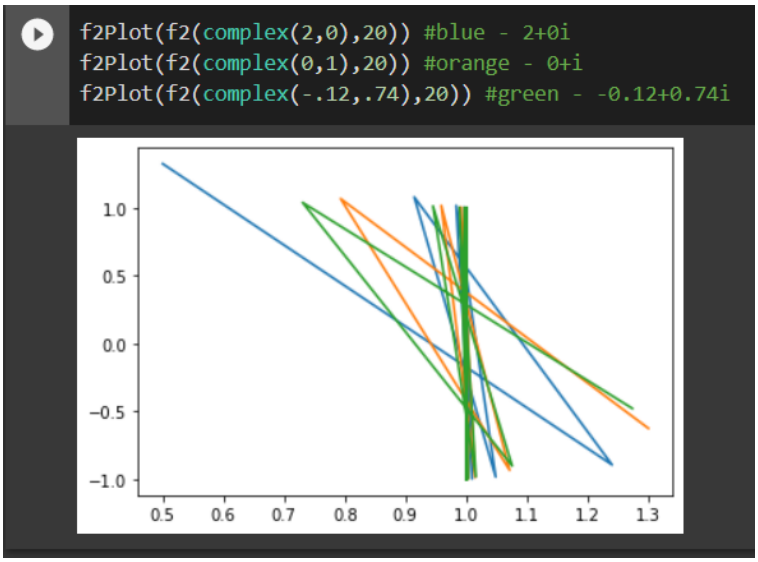

Figure 3: Scatterplot representation of the solution to $p(z) = z^2 + c$ where c is the input complex numbers.

This placement of the points was consistent even when rerunning the code with different values of N, as the N values did little to change the overall behavior of the plot.

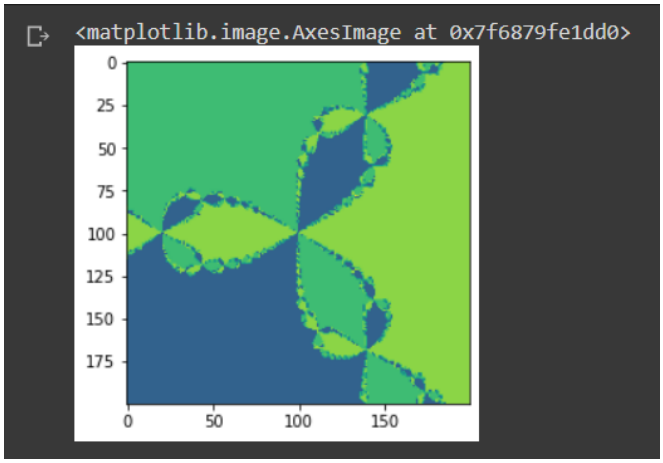

Figure 4: Image representation of of the same points of c as a solution to $p(z) = z^2 + c$ using plt.imshow

Removing any of the three values of "c" from the graph yields an identical shape but a different color gradient, showing that the points have a similar graphical nature but differ in where their points lie on the graph itself.

Part 5(c) - modify f1 to take only 1 solution based on probability, and compare time elapsed for the function to run and resulting image quality

Regarding the elapsed time, no clear difference was discernible between the two functions. Even when calculating the average time elapsed between the two, the most common averages were shared between the two

```
avg_time(timed_f1(complex(2,1)),50) #original function with time measurement implemented
```

```
9.775161743164062e-06
```

```
[115] avg_time(timed_f1Mod(complex(2,1)),50) #the function modified to use probability
```

```
9.298324584960938e-06
```

Figure 5: Average time elapsed between the original function f1 (top) and the modified version in which probability determines the chosen solution (bottom)

Surprisingly, the image quality was consistent for the modified function despite probability being used to determine the solution. The difference between the original and modified images is apparent, although the shape of the graph itself has been preserved save for an overall shift in the graph and change in the color gradient.

```
[75] #modified version of the image
     plt.imshow(mpic)
     plt.imshow(mpic2)
     plt.imshow(mpic3)
```

```
<matplotlib.image.AxesImage at 0x7f687a770c10>
```

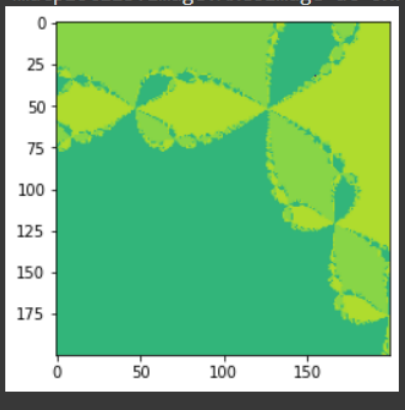

Figure 6: Modified image when using probability to determine the solution. The shape is noticeably identical minus the shift and color change

Modifications made in part 5

The modification in question requires that instead of one of the two solutions being chosen based on which is greater than or equal to 1, that the solution is chosen based on probability in which both solutions have equal chances to be chosen. In the code, this is done by replacing the parameter to check which solution is greater than or equal 1 by defining a variable to represent the probability using the random() function. This variable is called “probability”, for convenience, and rather than check which solution is greater than or equal to 1, we instead check whether probability is greater or less than 0.5. Either parameter works, but in my code I chose for the code to check whether it is greater than 0.5, in that case the first solution is chosen. If that check is not met, then an else check selects the second solution.

Programming and Math Challenges Encountered

Recursion - finding a way to create a recursive function of f(...(f(f(x))) for an N number of times

As this semester is my first experience with the Python programming language, I was not aware of how to write code such that a function can call itself multiple times.

Through trial and error I found that, using input c, you could create a loop that iterates for N times and set c = f(c)

time.time() - replacing the output of other functions with a time value even in the absence of time being used

This was a short-lived issue where running all previous functions that did not use the time() function would display a time value.

This was ultimately resolved by removing the time import, which fixed the functions in question, and then importing time() once again. No error was seen in the functions requiring time()

plt.imshow() - this function does not directly take in complex numbers

Image data of complex type cannot be converted to float, when directly inputting the function f2 into the function. Instead, the Newtonian image functions from hw5 worked better to create an image by putting the desired value of "c" into the "center" field of the function.