Nicholas Manfredi
Project 2: Report
MAT331

For the algorithms implemented for parts (2), (4), (6), and (7), they all had used a small program that was designed to find the correlating point p, for each iteration of the function. For some examples, we can give create algorithms that, when executed, determine the Julia set through inverse iteration. Inverse iteration is capable of solving and plotting julia sets through examining nearby neighbouring pixels. Some characteristics of the inverse iteration method(IIM) can include that for any C, the equation $p_c(z) = z$ has two finite solutions.Those being $u_0 \neq \infty \neq v_0$ - Additionally, if c ≠ ¼, there must be at least one repelling a fixed point. We can then say that one has $J_c = \overline{\{z \in \mathbb{C} : p_c^k(z) = u_0 \text{ for some } k \in \mathbb{Z}\}}$. We must account, that in general $p_c^k(z) = u_0$ which only has 2^k total solutions. These solutions can be obtained through recursively solving for the total number of preimages of u_0 that were iterated using the equation $Z^2 + c = u_0$. These preimages can be shown through $n(k) = 2^{k+1} - 1,\ k = 0,\ 1,\ ...$ These preimages were then plotted onto a x,y plane and was stored and graphed using list_plot. Similarly, the algorithm used for part (4), used the same process but instead was saved into a list, L, and was then plotted onto a matrix using matrix_plot. This caused some difficulty and confusion due to having trouble determining that the coordinate used consisted of a real number, x, and an imaginary number, y. The modified IIM algorithm for part (6) used a different approach. This time one of the roots were randomly chosen during each stage of each iteration of preimages. These iterated preimages can only give an approximate plot of the Julia sets (J_c). For the modified algorithm used in part (7), is similar yet faster than choosing a random preimage. This algorithm uses the same idea as choosing a random preimage, but instead assigns both of the complex numbers used to 0 and 1 respectively while returning only one if given a 0 or 1. This I believe is quicker and more efficient due to it being two steps ahead when plotting. This also is an approximation when plotting Julia sets. In addition to these algorithms, previously in lecture we have used the Boundary Scanning Method (BSM). The BSM in comparison to the IIM is more elementary. This algorithm goes through and creates an approximation for J_c . This is obtained through coloring all the completely labelled pixels. This then produces the desired Julia set.

The efficiency for each algorithm is fairly similar, all of which are extremely efficient. The algorithm used in (2) and (4) both use the same preimage function therefore having similar runtimes. The modification made for the algorithm used in (6) made it slightly more efficient. This allowed the program to be graphed using random preimages versus both as does the previous algorithm. This allows it to be one step ahead. This increases efficiency while not sacrificing much quality. The algorithm used in (7) had the quickest runtime while having the worst quality image. This algorithm enables plotting of the complex numbers two steps ahead instead of one.

I had a decent amount of trouble comprehending the formulas and methods that were needed to be implemented. This project took up significantly more time than I had expected. Luckily, I had my teaching assistant, Mu Zhao, available to help guide me through this project. If he had not been able to meet, I would most certainly have been unable to meet the deadline. He was able to help me with comprehending the theorem as well as cleaning up my syntax. He is a wonderful source of information and very willing to help.