Chase Allen

219303124

Phys 163

Dr. Block

5/13/2020

Deterministic Chaos: A recreation of famous results

GOALS

The underlying purpose behind this project is to recreate some famous results from the logistic map's function; $x_{n+1}=4\lambda x_n(1-x_n)$. Normally the function would exclude the 4, however by including it we can limit λ's domain to between 0 and 1 rather than 0 and 4. In order to arrive at what are known as **fixed points**, a specific λ must be chosen and is plugged into the logistic function with some initial x_0 and as you get the result x_{n+1} you plug it right back into the equation for some specified number of iterations. As the number of iterations is increased, the resulting set of points will approach and become some fixed set of values making further iterations redundant. These are the current λ's fixed points. For the purpose of this project, 10000 iterations will be used to reach the fixed points of a given lambda. The full logistic map can be generated by starting at λ = 0.0 and iterating over the function and then incrementing by some small amount and repeating until λ = 1.0. A smaller step size will give a more accurate logistic map with more data to work with. The specific result we were looking to replicate is the approximation of the *feigenbaum constant:* **δ**, which is a representation of the ratio of distance at which the next onset of bifurcation occurs from the last. The way this is done is through the formula

$\delta = (\lambda_{chaos} - \lambda_n)/(\lambda_{chaos} - \lambda_{n+1})$ in which the feigenbaum constant becomes more accurate as λ_n becomes a larger value of 2^n. However, as λ increases, the space between the next period doubling point and the last becomes smaller it becomes an issue of detecting with precision what the exact λ is for the onset of the next bifurcation. While we could use iterate and increment method to find all period doubling points up to a specific precision to approximate the feigenbaum constant, however that would be incredibly inefficient to achieve the purpose of this project as 10000 iterations with each step of lambda, which to detect a higher number of bifurcations requires step sizes as small as 0.0001, would result a larger load for the program and would take up more time than is necessary. This is where one can implement an algorithm similar to that of a binary search to find locations of the onset of the 2-cycle λ, 4-cycle λ , 8-cycle λ , and so on.

METHODOLOGY

The program is split into two main files; Driver.cpp and LogiMapProcessor.py. The cpp files purpose is to do all the number crunching for data points of interest, write them to a couple different files and the python script can take these data points and visualize them in a manner that allows us to see specific points of data for the user to get an idea of the inner workings of this map. To simply start, three methods were constructed to iterate through the logistic function a specified amount of times, increment lambda by the incredibly small step size of 0.0001, and write the data points to a .dat file. As was previously mentioned, this would not be for the purpose of finding the period-doubling points due to its inefficiency but was for the sole purpose of being able to visualize the map in the python script. Now came the task of finding the period-doubling points in a manner that wouldn't take ages to compute but still maintained some

degree of accuracy so the approximation of the feigenbaum constant could be done with relatively low error. A hybrid bracketing method was implemented to address this problem, and the implementation consisted of 3 methods; findCycle, cleanVector, sweepCycles. The findCycle method was the brain of this whole process. It takes in parameters of the cycle trying to be found, the left brackets lambda, the right brackets lambda, and the number of iterations to run this bracketing method to get as close to these cycle onsets as possible. It does this by being given a left bracket $\lambda = 0.0$ and right bracket $\lambda = 0.89248$, which is the value for the onset of chaos; a place where the number of occurring bifurcations is infinite, and since we are only interested in finding period-doubling points of 64-cycles and lower, of which all first occurrences happen prior to this chaos point, it makes sense to use this value as our upper bound. With the specified period-doubling point to be found being passed as a parameter in mind, the function calculates a middle bracket, passes all three brackets to a cleaning vector to remove possible duplicates from float precision loss, and then checks if the middle bracket has the less than or greater-than/equal-to the number of bifurcations that is being looked for. If it is equal to or greater than the number of bifurcations being sought after, the point must exist somewhere to the left of the current middle bracket and the left bracket is kept while the right bracket is set equal to the current middle bracket and a new middle bracket is calculated. If it is less then the point must exist somewhere to the right of the current middle bracket, so the right bracket is kept and the left bracket is set equal to the current middle bracket and a new middle bracket is calculated. Iterating through this process hundreds of times allows us to get incredibly close to the period-doubling point sought after. After all of the iterations complete, to ensure we are as close to the correct point as possible, it checks the final middle bracket calculated and checks how

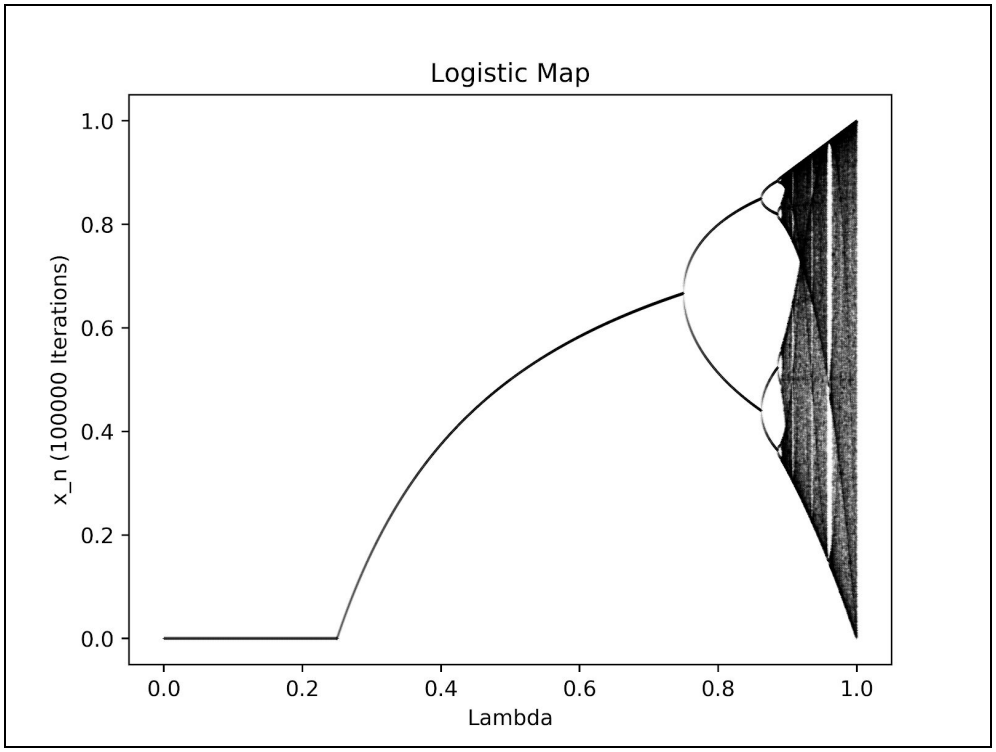
many bifurcations are occurring at that point, and if it is less than the number we want, it steps λ forward until it reaches the correct number of points. This is hybrid method is much more efficient in calculation of these period-doubling points as the bracket method gets incredibly close to the point with say only 1000 iterations and allowing the steps required to reach the point to be drastically reduced, and due to such, the number of calculations required to find these points in total is far less than generating and sweeping through an entire logistic map to find such points. The fixed points for the final λ are recorded to a file and the λ is returned. Since we are interested in finding the period-doubling points of 2-64, the sweep cycles method calls the findCycle method for points 2, 4, 8, 16, 32, and 64, and prints them to a file called cycleLocations.dat. Finally, there are 2 more methods implemented to help us receive our fabled feigenbaum constant; feigenbaum and approximateFeigenbaum. The feigenbaum method is simply the formula for calculating the feigenbaum constant however approximateFeigenbaum takes a specified upper bound and takes the previously calculated period-doubling locations for the specified upperbound and the previous onset and approximates the value and returns it. In the main method there is a loop to iterate through the cycles utilizing the approximateFeigenbaum method and writes the received value to a file called feigenbaum.dat. Once the program is finished, it lets user know that if they wish to visualize the data, they are to run the LogiMapProcessor.py script, which includes the logistic map with the locations of onsets clearly shown, a graph of the last 64 points of each cycle, and a graph that shows a horizontal line of an incredibly accurate feigenbaum approximation and all cycle calculations overlayed for comparison.

RESULTS

Displayed in the diagram to the right is a logistic map generated with utilizing the logistic function;

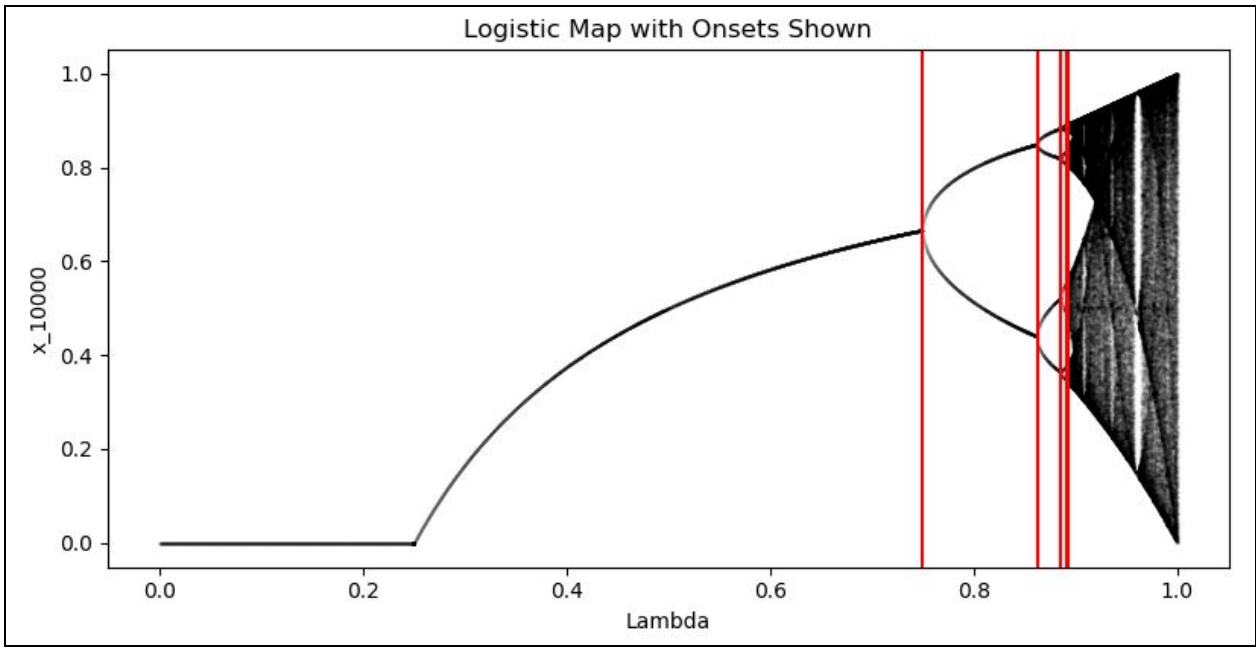
$$x_{n+1}=4\lambda x_n(1-x_n)$$

with initial condition $x_0 = 0$ generated by the 3 methods mentioned in the methodology portion of this paper resulting in a total point count of 136,222. You can see where the first three bifurcations occur between λ values of .7 and .9 but being able to point out where the splits happen becomes a lot harder as the space between the doubling points becomes smaller as lambda approaches the value of chaos. As the bracket method was employed the following locations were found for the period-doubling points up to the 64-cycle;

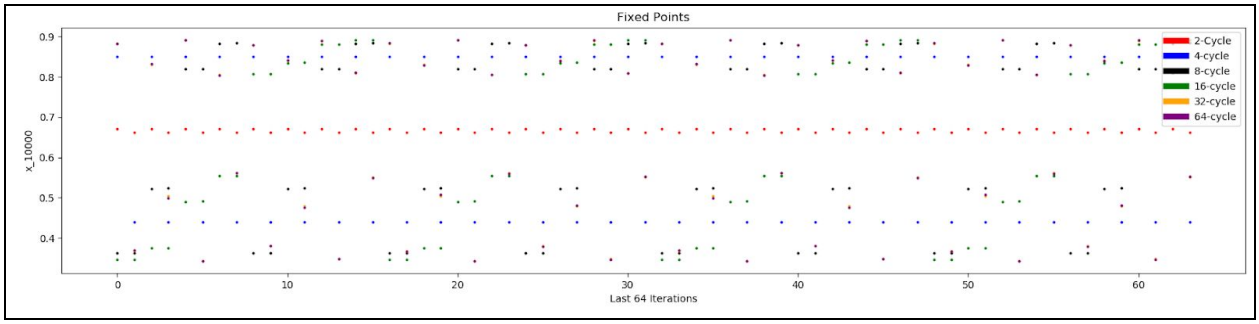


Cycle-2	0.750029
Cycle-4	0.862326
Cycle-8	0.886018
Cycle-16	0.891092
Cycle-32	0.892177
Cycle-64	0.892406

All of these points, although not 100% accurate, bring us incredibly close to these period-doubling points, and I attribute the inaccuracy to my detection of the points through possible overstepping during the iteration phase of the bracketing algorithm and vector cleaning eliminating the values too close to one another via the vector cleaning process. The logistic map with the bifurcation appears as;



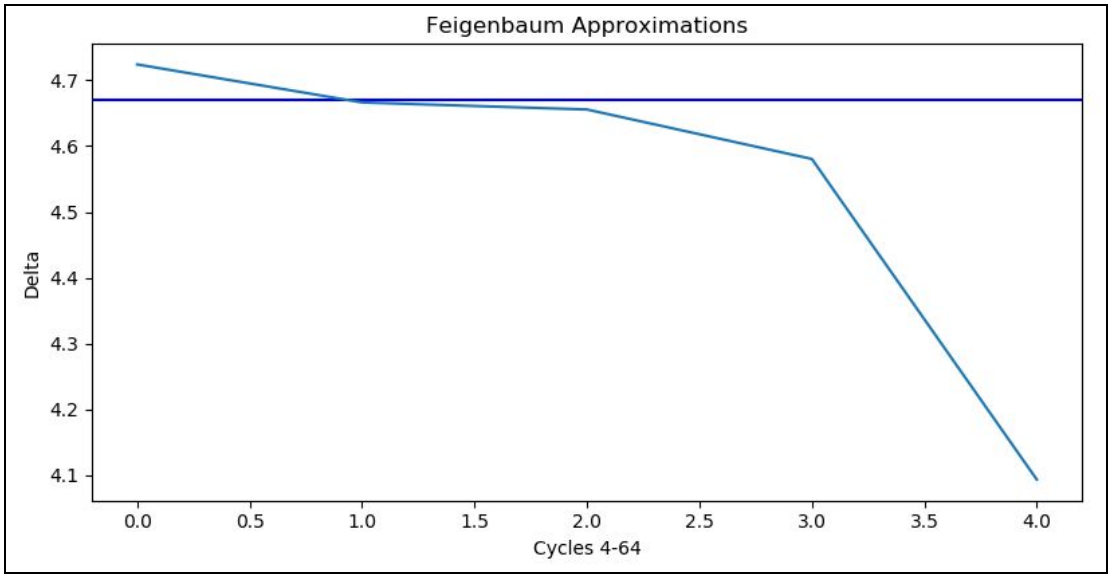
,and until expanded to an incredibly small precision, it appears to be fairly accurate. The fixed points for each of these period-doubling locations was also saved to a file and plotted utilizing matplotlib in python and is shown below;



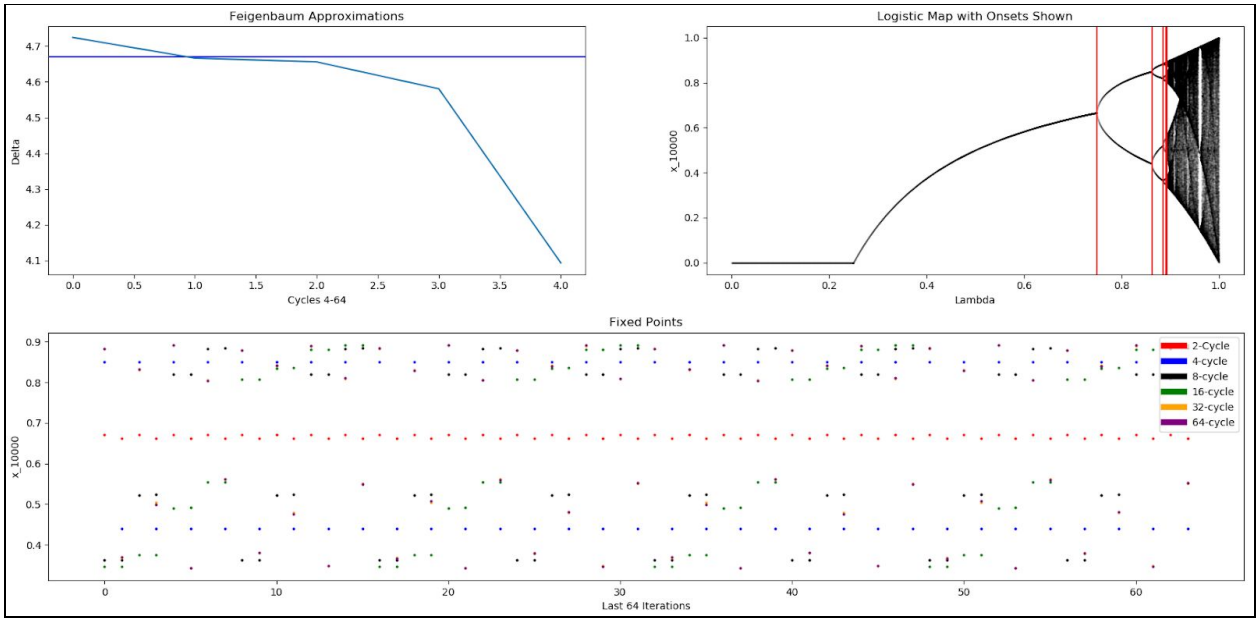
and upon inspection, if you follow a specific cycle by color you will actually see that the points will be bouncing between n-many points for that n-cycle. It was noticed during the calculation of the feigenbaum values that rather than the approximation becoming more and more accurate as the number of 2^N-cycles found increased, due to errors in my algorithms ability to definitively find the period-doubling point for higher order onsets, the feigenbaum approximation became less accurate. The found feigenbaum values are;

$(\lambda_{chaos} - \lambda_2)/(\lambda_{chaos} - \lambda_4)$	4.72412
$(\lambda_{chaos} - \lambda_4)/(\lambda_{chaos} - \lambda_8)$	4.66633
$(\lambda_{chaos} - \lambda_8)/(\lambda_{chaos} - \lambda_{16})$	4.6556
$(\lambda_{chaos} - \lambda_{16})/(\lambda_{chaos} - \lambda_{32})$	4.58045
$(\lambda_{chaos} - \lambda_{32})/(\lambda_{chaos} - \lambda_{64})$	4.0934

and yields this graph from the python script;



with the deep blue line being what the values should be approaching. The first four approximations appear to be in the ballpark of the theoretical feigenbaum approximation $\boldsymbol{\delta} = 4.6692$, however as previously mentioned we can see these values deviate as the precision of higher cycles becomes much more difficult to calculate. In all though, the program accomplished what it was originally intended to do to some degree; Collect enough data to visualize the logistic map clearly, find the period-doubling points using a bracketing method and write these points as well as the fixed points for that specific λ to files, calculate the feigenbaum values utilizing these newly found period-doubling locations and write them to a file and finally utilize a python script to visualize all of this newly collected data to create the figure you see below.



Works Cited

1. Block, M. "Deterministic Chaos: The Period Doubling Path to Chaos", email

Accessed 4/20/2020

2. Wikipedia. "Feigenbaum constants", https://en.wikipedia.org/wiki/Feigenbaum_constants

Accessed 5/13/2020