Dynamical Systems Modeling with Python

Leyton Taylor

February 2020

1 Abstract

The goal of this paper was to study how orbits behave specifically how the speed of convergence is related to the derivative of a function at a given fixed points. Twelve functions were given to us an initial seed was chosen to be the same for each of the functions. I chose to evaluate the behavior by creating a python program that iterates through each of the functions with the given seed and returns the rate of convergence.

2 Intro

We were given a value of .2 as the initial seed for each function. The given functions are as follows:

$F(x) = x^2 + .25$
$F(x) = x^2$
$F(x) = x^2 - .24$
$F(x) = x^2 - .75$
$F(x) = .4x(1-x)$
$F(x) = x(1-x)$
$F(x) = 1.6x(1-x)$
$F(x) = 2x(1-x)$
$F(x) = 2.4x(1-x)$
$F(x) = 3x(1-x)$
$F(x) = .4\sin(x)$
$F(x) = \sin(x)$

For each of the functions I calculated and then output
a.) The fixed point p
b.)$|F'(p)|$
c.)Attracting or Neutral
d.)The number of iterations for the orbit .2 to get within a certain distance to p

3 Method

In order to calculate the orbits for each of the given functions I created a method to which I could pass each of the functions to that would calculate the amount of iterations needed for .2 to converge to the given fixed point. A list of lamda functions and their corresponding derivatives named $functionList$ was created so I could iterate through each one avoiding manual calculations.

```
functionList=[[lambda x: x**2+.25, lambda x:2*x],
[lambda x: x**2, lambda x: 2*x],
[lambda x: x**2-.24, lambda x:2*x],
[lambda x: x**2-.75, lambda x: 2*x],
[lambda x: .4*x*(1-x), lambda x:.4 -.8*x],
[lambda x: x*(1-x), lambda x: -2*x+1],
[lambda x: 1.6*x(1-x), lambda x: 1.6-3.2*x],
[lambda x: 2*x(1-x),lambda x: 2-4*x],
[lambda x:2.4*x(1-x),lambda x: 2.4-4.8*x],
[lambda x: 3*x(1-x), lambda x: 3-6*x],
[lambda x: .4*sin(x), lambda x: .4*cos(x)],
[lambda x: sin(x), lambda x: cos(x)]]
```

Each of the functions and their corresponding derivatives was passed to a $CalcOrbits()$ method which output the the amount of iterations needed for the given seed to get within a certain distance $Epsilon$ to the fixed point of the function calculated by a $calcFixedPoint(F)$ method called within $CalCOrbits()$ Whats pretty cool is other than inputting the functions into the list there are no other manual computations done. The program does all of the computations. This means that you could pass any arbitrary function and seed and the analysis would still be done so long as the fixed point can be computed by the $calcFixedPoint(F)$ method.

Below is the $CalcOrbits()$ method:

```
def CalcOrbits(F,Df,o=.2, Epsilon=10**-4):
    fixedPoints=calcFixedPoint(F)
    temp=o
    iterationCount=0
    loop=True
    while(loop):
        lastOut=F(temp)
        temp=lastOut
        iterationCount+=1
        for p in fixedPoints:

            if(abs(temp-p)<Epsilon):
                loop=False
```

```
    if(Df(fixedPoints[k])!=1 for k in range(len(fixedPoints))):
        prop='Attracting'
    else:
        prop='Neutral'

    return('FixedPoints: '+ str(fixedPoints)+'\n'
          + '|Df(p)|: '+str(abs(Df(fixedPoints[k]) for k in range(len(fixedPoints))))+'\n'
          + prop + '\n'+
          +str(iterationCount))
```

And the corresponding $calcFixedPoint(F)$ method:

```
def calcFixedPoint(F):
    x=Symbol('x', real=True)
    f=implemented_function(Function('f'), F)
    lam_f=lambdify(x,f(x))
    return solve(lam_f(x),x)
```

4 Results

Applying the program yielded the following results:

```
Microsoft Windows [Version 10.0.17134.1246]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Leyton\Documents\UNI\DynamicalSystems\Lab2>python CalcOrbits.py
Function [lambda x: x**2+.25, lambda x:2*x],

FixedPoints: [0.500000000000000]
|Df(p)|: 1.00000000000000
Neutral
Iteration Count: 991
0.49900023302850594
-----

Function [lambda x: x**2, lambda x: 2*x],

FixedPoints: [0, 1]
|Df(p)|: 0
Attracting
Iteration Count: 3
2.560000000000002e-06
-----

Function [lambda x: x**2-.24, lambda x:2*x],

FixedPoints: [-0.200000000000000, 1.20000000000000]
|Df(p)|: 0.400000000000000
Attracting
Iteration Count: 1
-0.19999999999999998
-----

Function [lambda x: x**2-.75, lambda x: 2*x],

FixedPoints: [-0.500000000000000, 1.50000000000000]
|Df(p)|: 1.00000000000000
```

```
Neutral
Iteration Count: 499479
-0.5009999980733626
-----

Function [lambda x: .4*x-.4*x**2, lambda x:.4 -.8*x],

FixedPoints: [-1.50000000000000, 0.0]
|Df(p)|: 0.400000000000000
Attracting
Iteration Count: 6
0.0005900458704323005
-----

Function [lambda x: x*1-x**2, lambda x: -2*x+1],

FixedPoints: [0]
|Df(p)|: 1
Neutral
Iteration Count: 990
0.0009995846736983224
-----

Function [lambda x: 1.6*x-1.6*x**2, lambda x: 1.6-3.2*x],

FixedPoints: [0.0, 0.375000000000000]
|Df(p)|: 0.400000000000000
Attracting
Iteration Count: 8
0.37453031908956835
-----

Function [lambda x: 2*x-2*x**2,lambda x: 2-4*x],

FixedPoints: [0, 1/2]
|Df(p)|: 0
Attracting
Iteration Count: 4
0.4998589445046272
-----

Function [lambda x:2.4*x-2.4*x**2,lambda x: 2.4-4.8*x],

FixedPoints: [0.0, 0.583333333333333]
|Df(p)|: 0.400000000000000
Attracting
Iteration Count: 5
0.5842573583035319
-----

Function [lambda x: 3*x-3*x**2, lambda x: 3-6*x]]

FixedPoints: [0, 2/3]
|Df(p)|: 1
Attracting
Iteration Count: 55374
0.6676666505122555
-----

Function [lambda x: .4*sin(x), lambda x: .4*cos(x)]

FixedPoints: [0]
|Df(p)|: 0.400000000000000
Attracting
Iteration Count: 6
0.000812730862500752
-----
```

```
Function [lambda x: sin(x), lambda x: cos(x)]

FixedPoints: [0]
|Df(p)|: 1
Neutral
Iteration Count: 29922
0.00999990034973537
-----
```

Each function did end up working when put into the function, although some of the orbits took a significant amount of time to calculate.

5 Discussion

After a brief analysis of the results it is not hard to see that the functions with Neutral fixed points took significantly more iterations to pass the threshold *Epsilon*. A conjecture one could make about the output is that when $|F'(p)|$ is smaller the point p is more "attractive" and I think this is true.

Further if $|F'(p)| = 1$ then the fixed point is neutral, so the orbits will ever so slowly approach the fixed point going around and around and around and around... but eventually the orbit will come close enough to get within the threshold *Epsilon* we set.

One thing to note is that I had to use an $Epsilon = 10^-4$ because when I ran the program with $Epsilon = 10^-5$ the function $lambdax : x * *2 - .75$ just really did not seem to want to finish.

6 Comments and Thoughts

Seeing how slow my program was calculating even these tiny orbitals is a bit unnerving. $50,0000$ iterations is really a fairly small amount of iterations to compute, so I am not quite sure why it took so long. I don't think python is a very good programming language for fast versatile computations. It is a pretty slow language compared to lots of other high level languages with similar packages, but other than that I think one way to speed up the computations of the orbits would be using optimized tail recursion instead of an iterative approach which I used for simplicity. I'm not sure, but I think it would not only be faster, but would be more well suited for very large computations especially.

The...end