Dynamical Systems Modeling in Python

Leyton Taylor

February 17, 2020

1 Abstract

The goal of this paper was to compute a number of orbits of three different functions and record and analyze the results. I chose to create a python program in order to do this. I utilized a python package called matplotlib in order to visualize the orbits of the three different functions and further analyze whether the orbits were fixed, periodic, eventually periodic or no pattern over 100 iterations of each of the given functions.

2 Intro

The three functions to analyze are as follows

$$F(x) = x^2 - 2 \tag{1}$$

$$G(x) = x^2 + c \text{ for some } c < -2 \tag{2}$$

$$D(x) = \begin{cases} 2x & \text{if } 0 <= x < 1/2 \\ 1 & \text{if } 1/2 <= x < 1 \end{cases} \tag{3}$$

The seeds were chosen as follows. Random floating point on the interval $[0, 1)$ for the doubling function D, Random floating point on the open interval $(-2, 2)$ for functions F and G

3 Generating the Orbits

The labs instructions were to choose ten random seeds for each of the functions on the predefined intervals given to us. Instead of choosing the seeds myself I had python choose each of the ten seeds for me. The code below uses a comprehension to generate a list of ten randomly chosen initial seeds for each of the functions.

```
randomFseeds=[random.uniform(-1.99,1.99) for k in range(10)]
randomGseeds=[random.uniform(-1.99,1.99) for k in range(10)]
randomDseeds=[random.random() for k in range(10)]
```

I then defined each of the functions in three separate methods. I applied each of the functions to the seed passed to it 100 times. Keeping track of the number of iterations each time applied the given function. Each iteration a tuple of the form $(i, F^i(x))$ appended to a list resulting in a list of tuples holding all the necessary information of each iteration.

$$F(x) = x^2 - 2 \tag{4}$$

```
def F(x):
    outList=[(0,x)]
    temp=x
    for i in range(1,100):
        lastOut=temp**2-2
        temp=lastOut
        outList.append((i,temp))
    return outList
```

$$G(x) = x^2 + c \text{ for some } c < -2 \tag{5}$$

```
def G(x):
    outList=[(0,x)]
    temp =x
    for i in range(1,100):
        lastOut=temp**2-2.5
        temp=lastOut
        outList.append((i,temp))
    return outList
```

$$D(x) = \begin{cases} 2x & \text{if } 0 <= x < 1/2 \\ 1 & \text{if } 1/2 <= x < 1 \end{cases} \tag{6}$$

```
def D(x):
    outList=[(0,x)]
    temp=x
    for i in range(1,100):
        if(temp<.5 and temp>=0):
            lastOut=temp*2
            temp=lastOut
            outList.append((i,temp))
        else:
            lastOut=temp*2-1
            temp=lastOut
            outList.append((i,temp))

    return outList
```

Lastly I generated a dictionary for each of the functions. The keys represent the initial seeds passed to the functions and the values represent the corresponding list of tuples representing each iteration of the function from $1 - 100$

A method defined as *genOrbits*() creates and returns these three dictionaries and prints the randomly chosen seeds for the user.

```
def genOrbits():
    Fout=dict()
    Gout=dict()
    Dout=dict()

    randomFseeds=[random.uniform(-1.99,1.99) for k in range(10)]
    randomGseeds=[random.uniform(-1.99,1.99) for k in range(10)]
    randomDseeds=[random.random() for k in range(10)]
    Fout={x:F(x) for x in randomFseeds}
    Gout={x:G(x) for x in randomGseeds}
    Dout={x:D(x) for x in randomDseeds}

    print("F seeds: "+str(randomFseeds)+'\n'
          +"G seeds: "+str(randomGseeds)+'\n'
          +"D seeds: "+str(randomDseeds))

    return Fout,Gout,Dout
```

4 Results

Utilizing pythons package matplotlib I created two visualizations of the orbits. First a method $plotLine(orbitDict, seed)$ plots each output of iteration as a line graph with the y axis representing the range of the outputs and the x axis representing the number of iterations. Second a method $plotPeriod(orbitDict, seed)$ which plots the recurrences of the seed or any arbitrary value throughout the 100 iterations. I hoped to be able to use $plotPeriod(orbitDict, seed)$ to analyze if the functions were periodic, eventually periodic, or fixed. The code for each is shown below.

```
def plotLine(orbitDict,seed):
    plt.ylabel='Output of Iteration'

    xAxis=[orbitDict[seed][n][0] for n in range(100)]
    yAxis=[orbitDict[seed][n][1] for n in range(100)]


    plt.plot(xAxis,yAxis)
    plt.show()



def plotPeriod(orbitDict,seed):
    xAxis=[]
    yAxis=[]

    for k in orbitDict[seed]:
        if(k[1]==seed):
            yAxis.append(seed)
            xAxis.append(k[0])

    plt.plot(xAxis,yAxis, 'ro')
    plt.axis([0,100,-2,2])
    plt.show()
```

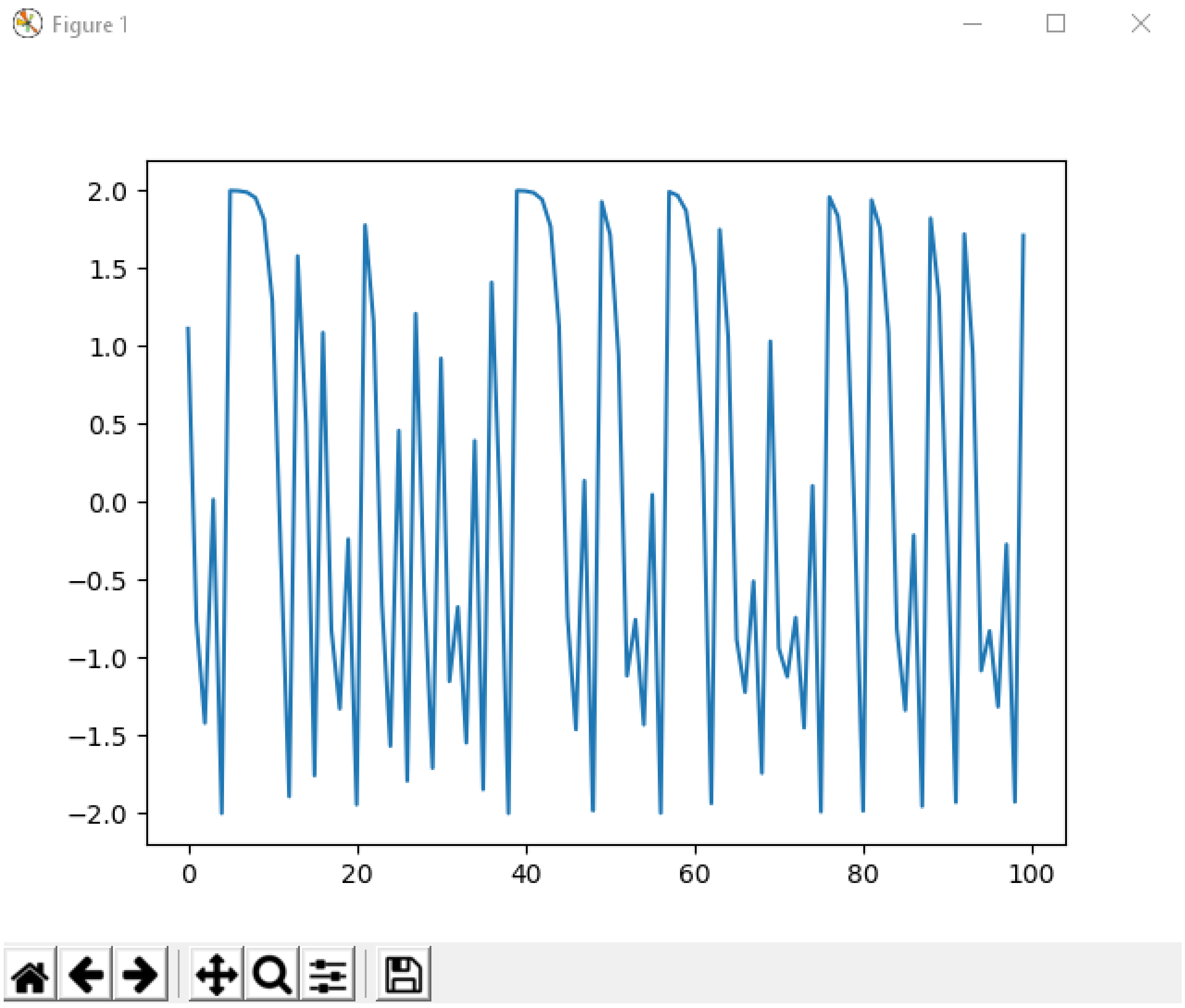
4.1 Analysis

Applying the program yeilds the following results.

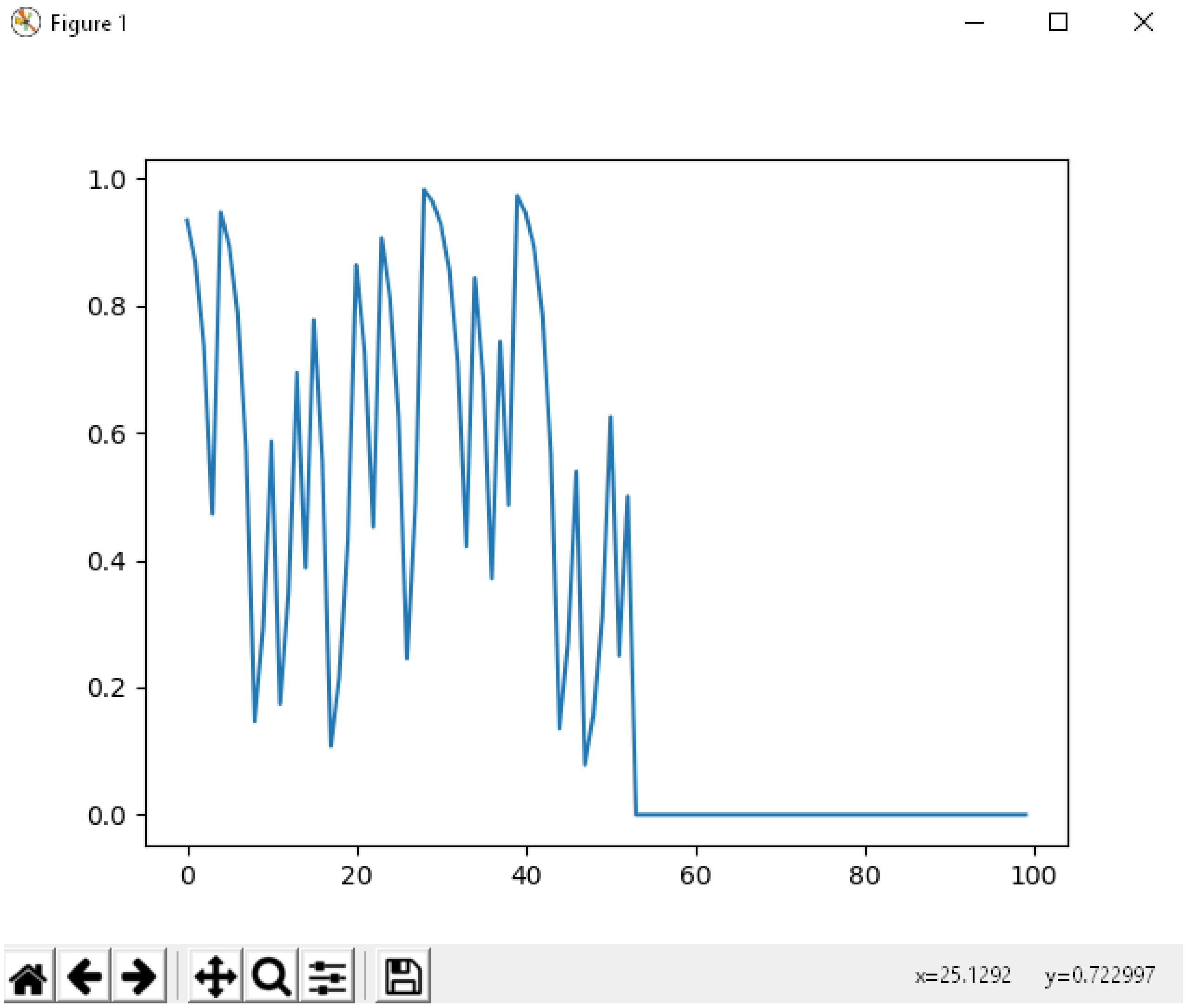
```
>>> f,g,d=genOrbits()
F seeds: [1.1125496682798748, -1.3883301844690137, 1.9640721811820903, -0.1285232265122065,
-0.8298511083452103, 0.582705472913269, -1.3383659561225012, -1.6466955491732436, -0.2660480786474808,
1.1241119916096702]
G seeds: [0.8066017273152768, -1.0559498841183155, -0.4810014568068619, -0.9879492186313561,
1.4824948033623804, 0.41816258540138773,
1.0899696340835823, -0.1104950664233535,
0.11794385229858739, 1.9481936286266304]
D seeds: [0.9341667778116479, 0.9410383243765882, 0.5189400625274854, 0.7094719948934287,
0.93245732037569, 0.16559116325416778,
0.22409062917417377, 0.16305149717259237,
0.5572726197080015, 0.14088525508195082]

>>> f[1.1125496682798748][0:50]
[(0, 1.1125496682798748), (1, -0.7622332356103405), (2, -1.419000494530991),
(3, 0.01356240347919746),  (4, -1.9998160612118674), (5, 1.9992642786809474),
(6, 1.9970576560096487), (7, 1.9882392814267522), (8, 1.9530954402083678),
(9, 1.8145817985627177), (10, 1.2927071036751077), (11, -0.32890834410791436),
(12, -1.8918193011761897), (13, 1.5789802683027667), (14, 0.49317868768947726),
(15, -1.756774782008885), (16, 1.0862576347023656), (17, -0.8200443510508222),
(18, -1.327527262309636), (19, -0.23767136782468312), (20, -1.9435123209163443),
(21, 1.7772401415536354), (22, 1.1585825207495861), (23, -0.6576865426135348),
(24, -1.567448411665055), (25, 0.4568945232313042), (26, -1.7912473946412393),
(27, 1.2085672288090277), (28, -0.5393652534488673), (29, -1.7090851233720392),
(30, 0.9209719589316183), (31, -1.1518106508616577), (32, -0.6733322245616444),
(33, -1.5466237153668672), (34, 0.39204491693521204), (35, -1.8463007831052627),
(36, 1.4088265816951062), (37, -0.01520766270928231), (38, -1.9997687269949207),
(39, 1.9990749614668855), (40, 1.9963007015638294), (41, 1.9852164910642376),
(42, 1.941084516393404), (43, 1.7678090997822156), (44, 1.1251490132728073),
(45, -0.7340396979312283), (46, -1.4611857218610311), (47, 0.13506371377054283),
(48, -1.9817577932225088), (49, 1.927363950998148)]
>>>
```

Result of Calling *plotLine*() on F, D with seeds $1.1125496682798748, 0.8066017273152768, 0.9341667778116479$

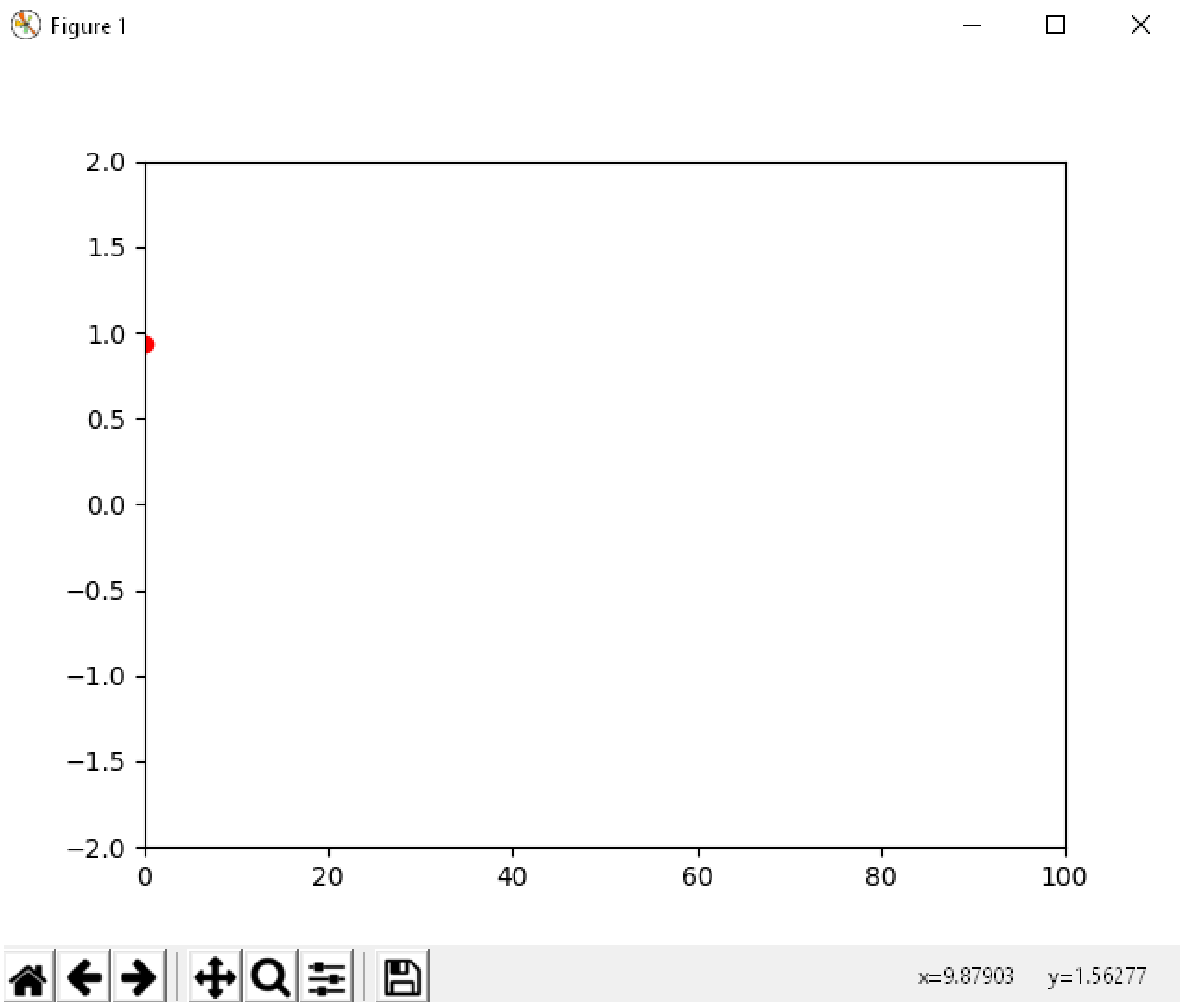
```
>>> plotLine(f, 1.1125496682798748)
```



```
>>> plotLine(d,0.9341667778116479)
```



Because none of the three seeds ever reoccurred throughout the orbit the orbit makes the output of the *plotPeriod*() method pretty boring. The graph created for each function resembles the graph for the function D below.



5 Discussion

After plotting a few sample seeds and looking at the orbits over time it is apparent that the doubling function D seems to always be eventually fixed no matter what seed I have put into the function the output at around 60 iterations seems to go to 0.

The function F seems to have consistently chaotic behavior for all of the seeds I tried, but I did not see a pattern in any of the sample graphs.

When trying to analyze and plot the function G I ran into several problems. From what the instructions said I used the open interval (-2,2) to generate the random seeds put into G, but was never able to find a c value to put into the function that when iterated 100 times didn't end up in an overflow error. I think the function is so chaotic that for almost any seed the output of 100 iterations will result in a value too large to be stored on the computer.

Another problem with iterating functions in python is the issue of rounding. Each iteration python rounds the output. This especially bad when the starting seeds are long floating points. Each iteration loses accuracy. Outputs can be plain wrong because of this. A possible solution could be to not apply operations on the values, but to store the operations in memory. Preserving the exact values of the outputs and eliminating the rounding problem. Although this would become an unviable solution fairly quickly as the orbits being calculated get very large.