PHY407 Lab 01

Sang Bum Yi, 1004597714
Jianbang Lin, 1004970720

The workload was distributed as followings:

- Sang Bum Yi did question 2
- Jianbang Lin did questions 1 and 3.

Question 1

b) Equations used for the code are:

$$V_{x_{i+1}} = -\frac{GM_s x_i \Delta t}{r^3} + V_{x_i} \tag{1}$$

$$V_{y_{i+1}} = -\frac{GM_s y_i \Delta t}{r^3} + V_{y_i} \tag{2}$$

$$x_{i+1} = V_{x_{i+1}} \Delta t + x_i \tag{3}$$

$$y_{i+1} = V_{y_{i+1}} \Delta t + y_i \tag{4}$$

This program can take in different initial positions and velocity, and simulate the path it is going to take. In our program, we use the 'Eular-Cromer' method, which means when we update the position and velocity in a small time increment Δt, we update velocity first, and then use the newly updated velocity to update position.

c)

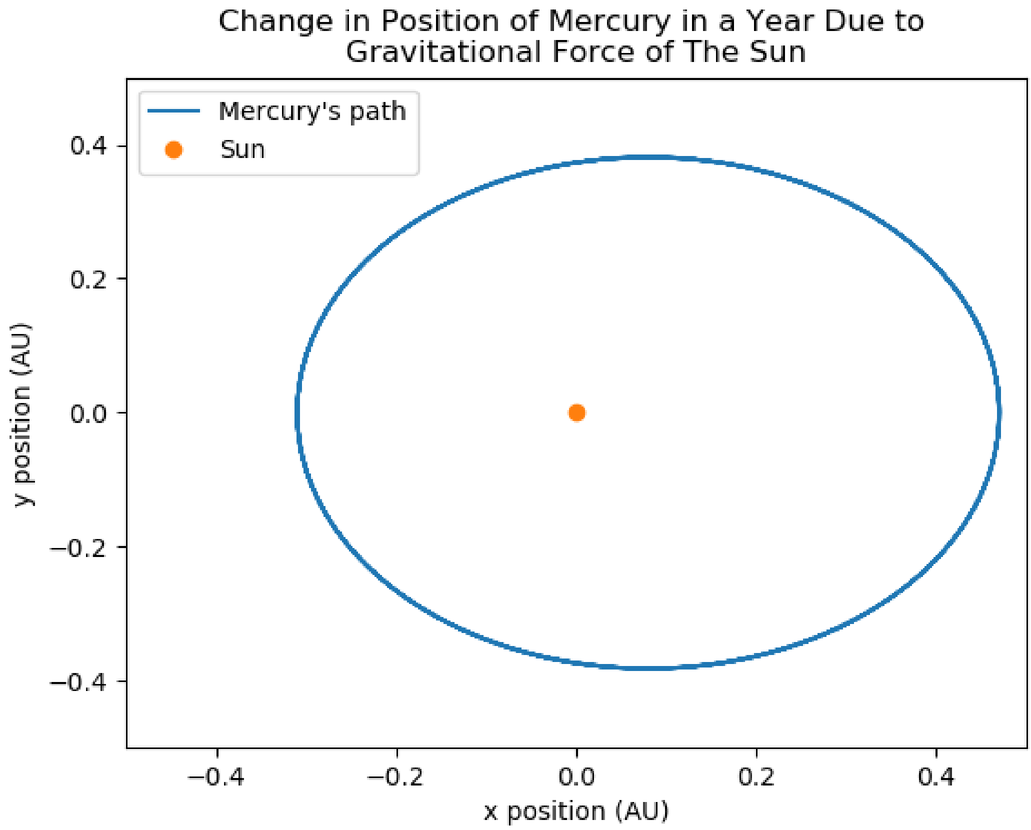

Figure 1: Mercury's path around the sun simulated using equations 1, 2, 3, 4. The sun's position is at (0,0).

This graph shows the path Mercury will take in a year and it is elliptical. In a year of Earth's time, Mercury will orbit around the sun several times, and all their paths are overlapped. The orbiting period of Mercury is 0.24 year. Also, the center of the ellipse is not (0,0), which means the sun is at the center of Mercury's orbit. The sun is slightly to the right of the ellipse center.

Angular momentum is given by equation:

$$\vec{L} = \vec{r} \times \vec{p} = m \cdot \vec{r} \times \vec{v} \tag{5}$$

To check the angular momentum conserved throughout orbiting, it is not enough to know the position of the Mercury, we also need to know its velocity. The velocity of Mercury in x,y directions is shown as following:

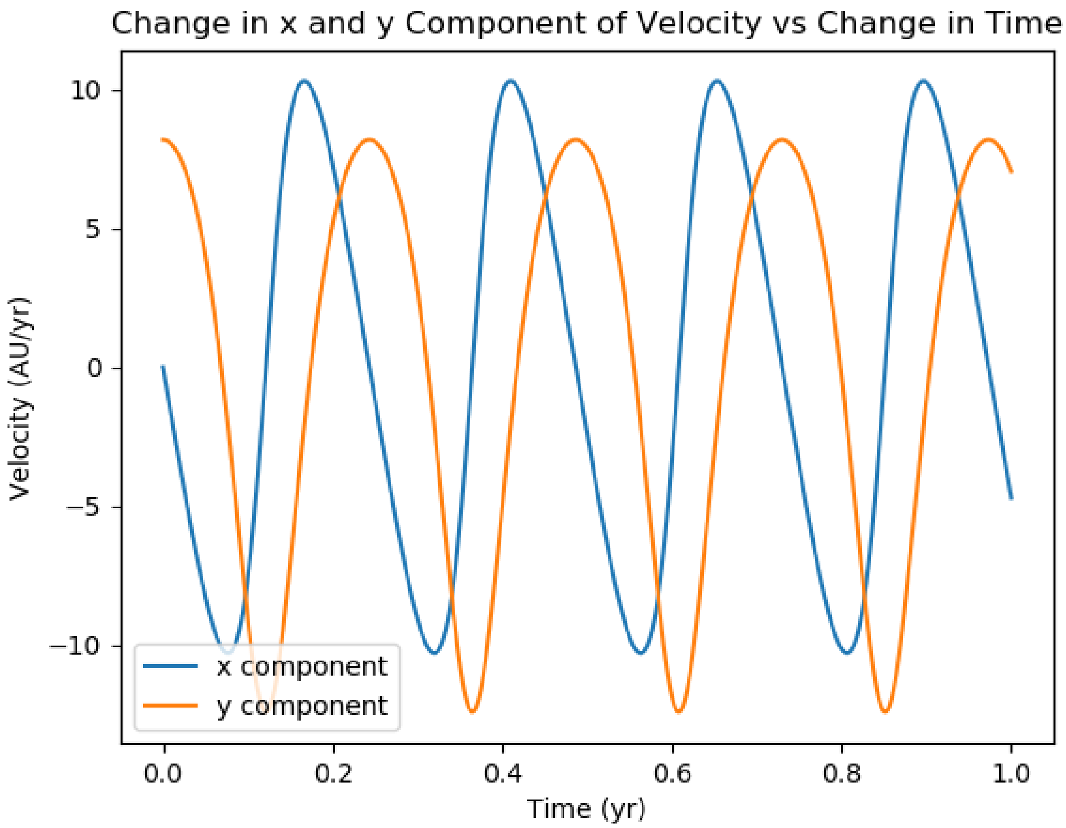

Figure 2: x, y components of velocity of Mercury during its one year orbit.

The velocity in x,y directions are periodic as well, it is because as Mercury approaches the sun, its gravitational potential energy will be converted into kinetic energy, thus increasing its velocity. Therefore when the velocity is the largest, it means the Mercury is at the point closest to the sun, and when the velocity is the smallest, it means the Mercury is at the point furthest away. Position and velocity both have the same period 0.24 year, they have the same magnitude in each oscillation, this implies angular momentum is conserved.

Angular momentum conservation is checked directly in the code by calculating the difference between final angular momentum and initial momentum. The angular momentum difference is 0, which matches our result above.

d)

When the mass of objects becomes too large, Newtonian mechanics becomes less accurate, and general relativity is much more accurate when dealing with massive objects. When we use general relativity in our gravitational force equation, velocity becomes following:

$$V_{x_{i+1}} = -\frac{GM_s x_i \Delta t}{r^3}\left(1 + \frac{\alpha}{r^2}\right) + V_{x_i} \tag{6}$$

$$V_{y_{i+1}} = -\frac{GM_s y_i \Delta t}{r^3}\left(1 + \frac{\alpha}{r^2}\right) + V_{y_i} \tag{7}$$

For a more obvious visualization, the value of α is increased from $1.1 \cdot 10^{-8}$ to 0.01. Position equations will stay the same as equations (3), (4). The position graph is shown as following:

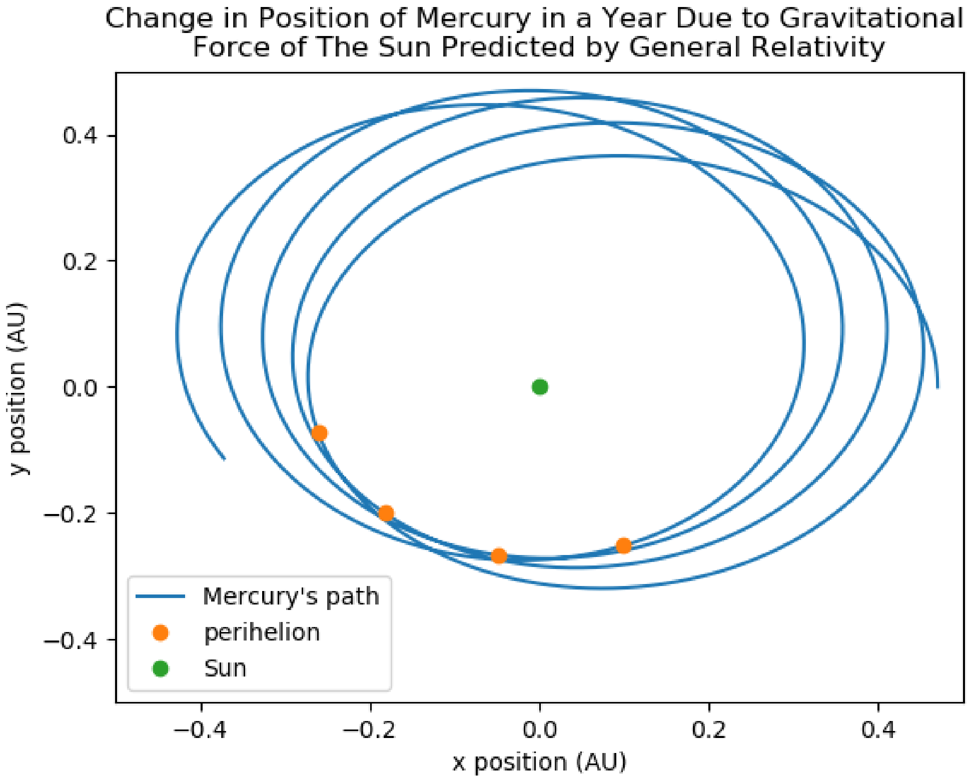

Figure 3: Mercury's path simulation using general relativity, the sun's position is at (0,0).

This graph shows Mercury's orbital precession, Mercury still moves in an elliptical path, but the elliptical path also rotates around the sun. When precession occurs, perihelion will be different for each orbit. As shown in this graph, the closest point to the sun differs slightly for each orbit.

Question 2

a)

The following is the pseudocode to compute and plot the logistic map, which is given as:

$$x_{p+1} = r(1 - x_p)x_p \quad (8)$$

where 'p' refers to a year and 'r' refers to the maximum reproduction rate.

```
# Pseudocode
# As code comments
# From keyboard, read the maximum population, r
# From keyboard, read the number of years, p
# From keyboard, read the initial value of x, x0
# Create an array with length of p
# Compute x_p using equation 12 and assign it to the array
# Plot the array
```

c)

The figure 4 below shows the evolution of populations at a variety of maximum reproduction rates between 2 and 4. The initial population and the maximum year were set to 0.1 and 50, respectively.

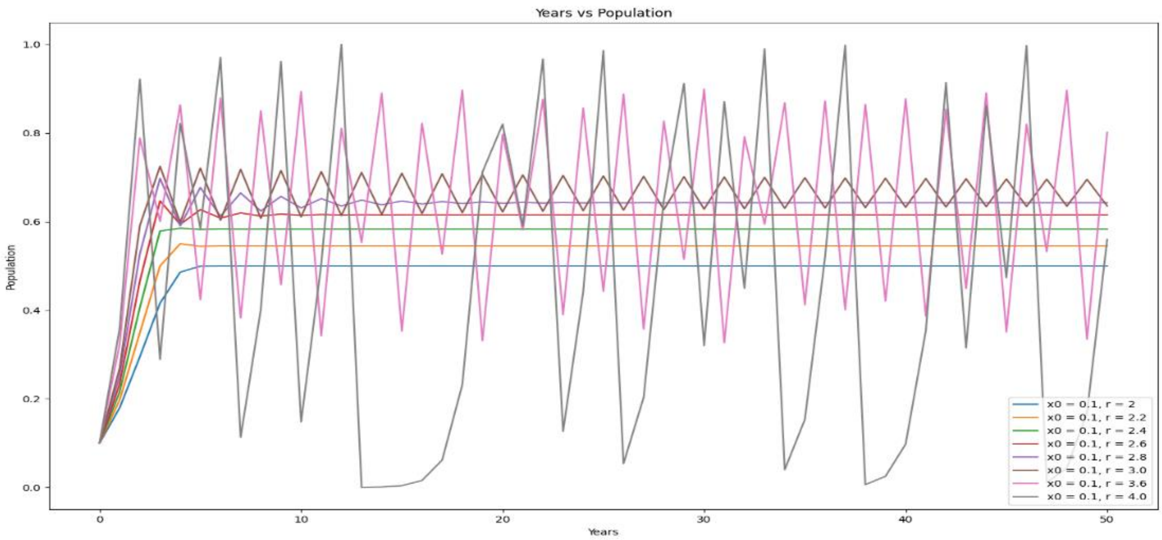

Figure 4: The evolution of populations at different maximum reproduction rates, which were taken in the range between 2 and 4.

For the values of 'r' (maximum reproduction rate) less than 3, the populations show an exponential growth and hits the maximum, as shown in the flatness of the lines. However, once the value of 'r' exceeds 3, such maximum of the population no longer exists. Instead, the population shows a chaotic behavior with oscillations, whose intensity gets greater as the value of 'r' gets closer to 4.

d)

The figure 5 below shows a bifurcation diagram of the evolution of population at various maximum reproduction rates, with fixed initial population. The maximum reproduction rate was incremented by 0.1 between 2 and 4. On the contrary, the figure 6 shows the same bifurcation diagram with the increment of 0.005 for the maximum reproduction rate.

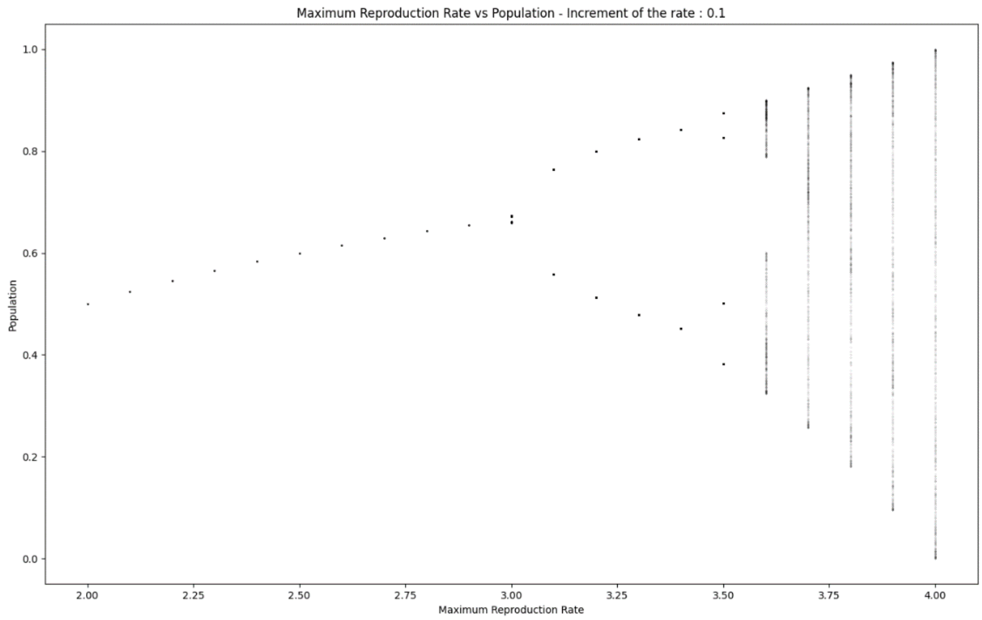

Figure 5: A bifurcation diagram of the population at the values of r (maximum reproduction rates) between 2 and 4 in increments of 0.1, with the initial population of 0.1 and the maximum year of 2000. For r < 3, only the last 100 values of population were plotted, whereas the last 1000 values of population were plotted for r > 3.

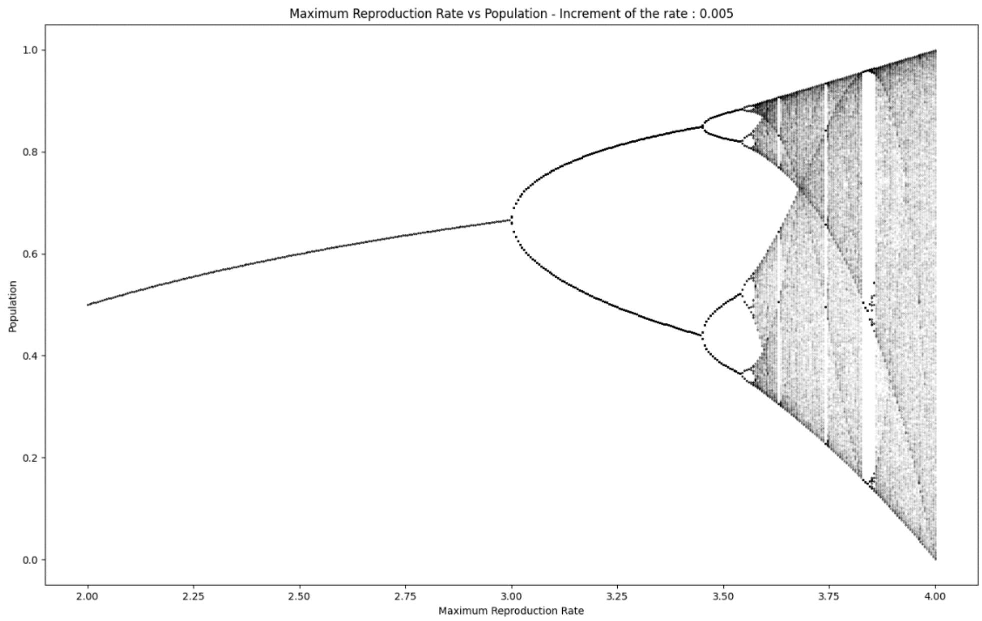

Figure 6: A bifurcation diagram of the population at the values of r (maximum reproduction rates) between 2 and 4 in increments of 0.005, with the initial population of 0.1 and the maximum year of 2000. For r < 3, only the last 100 values of population were plotted, whereas the last 1000 values of population were plotted for r > 3.

The figure 6 reveals that the population converges to a single value for the value of r (maximum reproduction rate) less than 3. However, for r > 3, the population oscillates between two final values. This phenomenon of having twice more final values at a certain point is called "period doubling". Approximately at r = 3.4 and 3.6, the period doubling occurs again, with the population oscillating among four and eight final values, respectively. Then, the evolution of the population shows a pattern of a vertical spray, which means a chaotic behavior with undetermined final values. The populations at r = 3.75 or 4.0 are obvious examples of such chaotic behavior. According to figure 5 and 6, the chaotic behavior seems to occur beyond r = 3.570.

e)

The same bifurcation diagram was reproduced with the narrower range and smaller increments of 'r', which were [3.738, 3.745] and 10^{-5} respectively. The diagram can be found in figure 7 below.

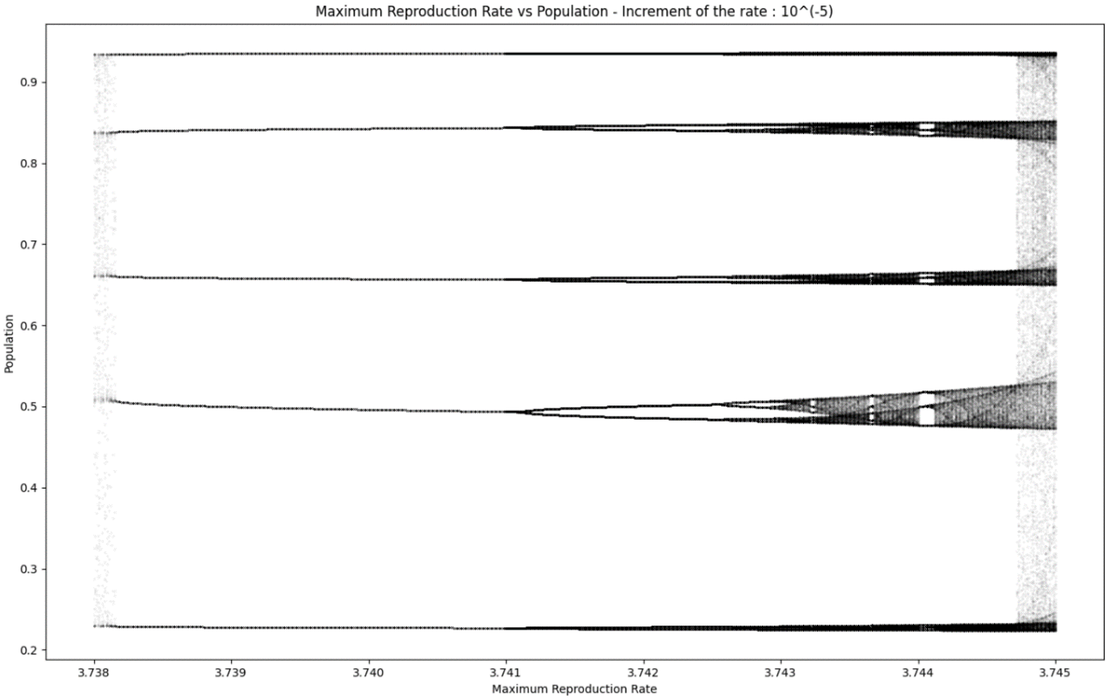

Figure 7: A bifurcation diagram of the population at the values of r (maximum reproduction rates) between 3.738 and 3.745 in increments of 10^{-5}, with the initial population of 0.1 and the maximum year of 2000. For r < 3.741, only the last 100 values of population were plotted, whereas the last 1000 values of population were plotted for r > 3.

The figure 7 provides a microscopic view into figure 6 at the range of ‘r’ between 3.738 and 3.745. As can be seen by comparing figure 6 and 7, the starting points have converged to a few values. From these starting points, the populations start to undergo the same period doubling and become chaotic again.

f)

With two initial populations that differ from each other only slightly, two population evolutions were plotted in figure 8 below, with r = 4.0. The value of ‘r’ was chosen such that the population evolutions would show obvious chaotic behaviors with the given initial populations. In this case, the initial population was set to 0.1, which is identical to the value used in part c, and the value of ‘r’ was chosen to be 4.0 because the amplitude of the oscillation was the greatest in figure 4.

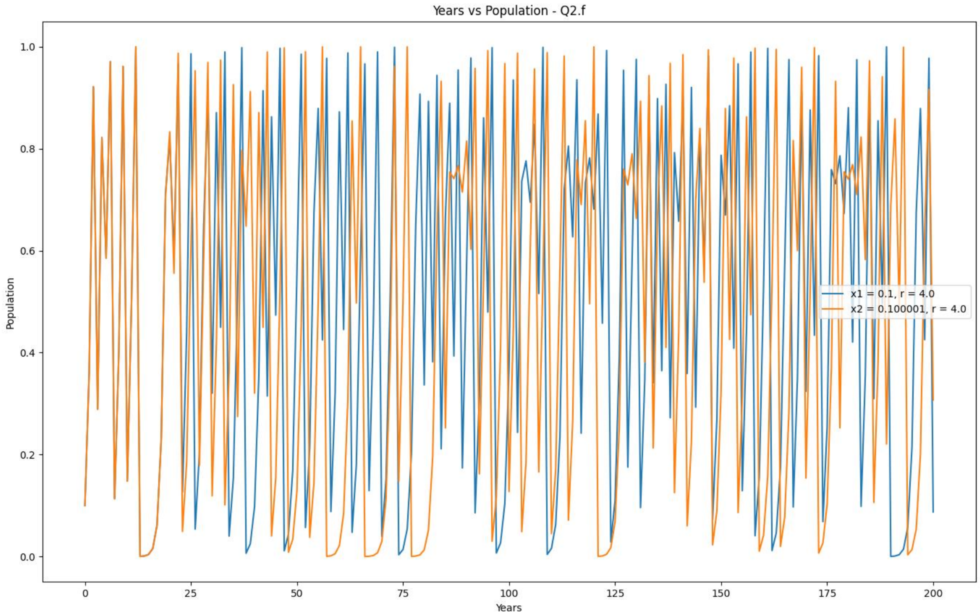

Figure 8: An evolution of two populations with initial populations of 0.1 and 0.100001 at r = 4.0.

The number of iterations was later modified after plotting figure 9 in part g, in a way that shows both exponential growth and saturation of the population difference. Since the exponential growth occurred at the year of 25, the range of years was chosen to be [0, 200].

g)

The absolute value of the difference between two populations from part f was taken and plotted against years in a logarithmic scale, as seen in figure 9. Then, the line of best fit was plotted for years between 0 and 25, where a rapid growth was observed.

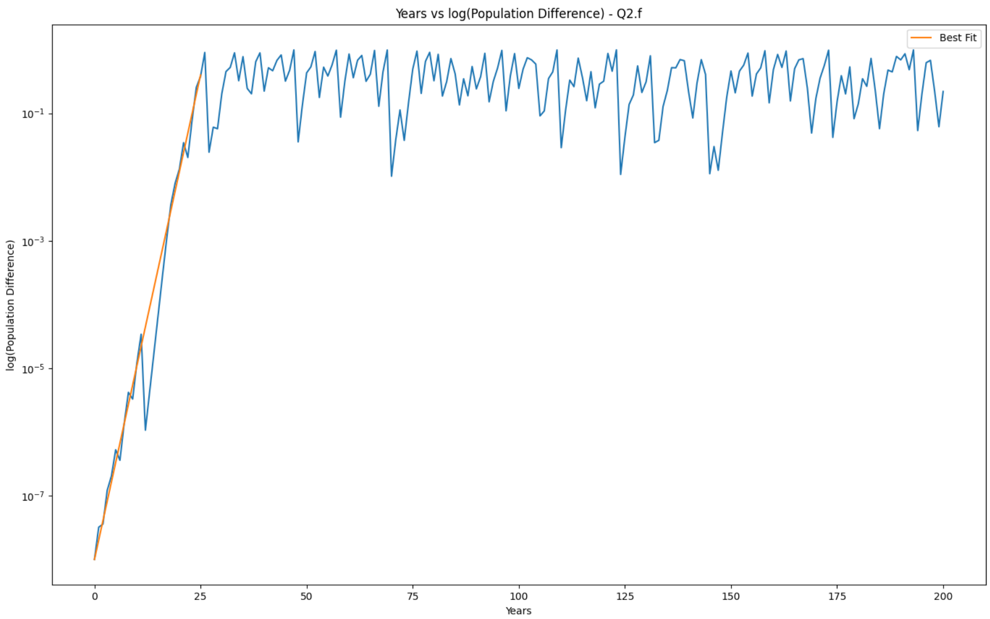

Figure 9: The difference between two populations in a logarithmic scale. The blue line represents the difference, whereas the orange line represents the line of best fit.

The differences between two population evolutions show a linear growth in a logarithmic scale and saturates after year 25. The linear growth between year 0 and year 25 in the logarithmic scale is equivalent to an exponential growth in numeric values. The line of best fit was drawn with the model function of $ae^{p\lambda}$, where p refers to years, λ refers to the Lyapunov exponent, and 'a' refers to a constant. As a result of fitting, the Lyapunov exponent was determined to be 0.7

Question 3

One of the reasons people prefer to use numpy is because its calculation speed is very fast, faster than the traditional calculation method. One example is matrix multiplication, one commonly used method is to use row by column multiplication, which is implemented using for-loop. Then the run-time for this method is compared with run-time using numpy.dot:

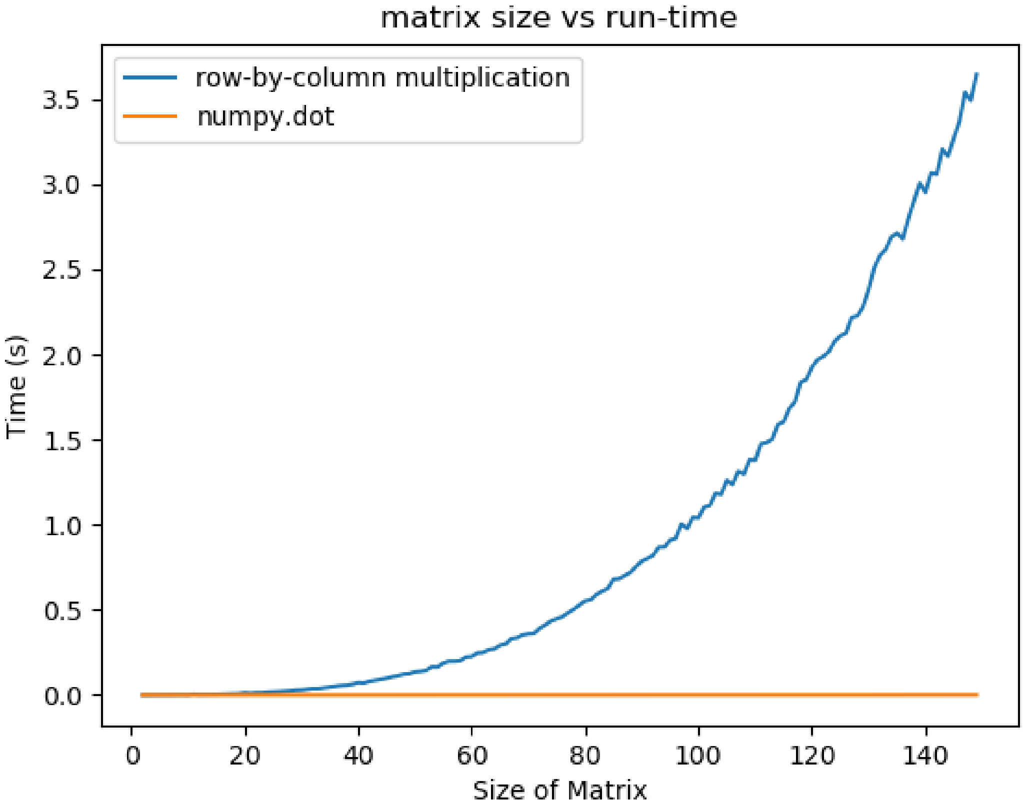

Figure 10: Increase in run time with respect to increase in matrix size when doing matrix multiplication using row-by-column multiplication vs using numpy.dot.

The run-time for both methods is around 0s when the matrix size is small. As matrix size increases, the run-time for nump.dot method stays constant at 0s. On the other hand, the run-time for row-by-column multiplication method increases cubically. It means numpy.dot is a much more efficient method of doing matrix multiplication, it takes constant time, while commonly used method row-by-column multiplication method takes cubic time.

Appendix A – Run Time

run time using row multiplication

matrix size	run time
2	0.0
3	0.0009989738464355469
4	0.0009958744049072266
5	0.000997304916381836
6	0.0019943714141845703
7	0.0006515979766845703
8	0.0009968280792236328
9	0.0
10	0.0070972442626953125
11	0.004006385803222656
12	0.0
13	0.0
14	0.009405136108398438
15	0.0009968280792236328
16	0.0051996707916259766
17	0.008210420608520508
18	0.0062291622161865234
19	0.0020949840545654297
20	0.01451563835144043
21	0.008142948150634766
22	0.008919000625610352
23	0.017636775970458984
24	0.017780542373657227
25	0.017417430877685547
26	0.014615774154663086
27	0.02506542205810547
28	0.023775339126586914
29	0.02583932876586914
30	0.0250852108001709
31	0.0344090461730957
32	0.0312504768371582
33	0.033315181732177734
34	0.04372358322143555
35	0.0400547981262207
36	0.05117464065551758
37	0.057303428649902344
38	0.057083845138549805
39	0.06835556030273438
40	0.07856249809265137
41	0.06497645378112793
42	0.07689952850341797
43	0.09042501449584961
44	0.09616756439208984

45	0.09331750869750977
46	0.0960230827331543
47	0.11319470405578613
48	0.1558229923248291
49	0.16783905029296875
50	0.1589951515197754
51	0.17324447631835938
52	0.1680600643157959
53	0.20272469520568848
54	0.20807623863220215
55	0.2151355743408203
56	0.24132704734802246
57	0.27898550033569336
58	0.2633700370788574
59	0.3262319564819336
60	0.29512476921081543
61	0.31851863861083984
62	0.34853267669677734
63	0.3376476764678955
64	0.3343238830566406
65	0.34436869621276855
66	0.3782222270965576
67	0.3823208808898926
68	0.31755757331848145
69	0.3349733352661133
70	0.3774111270904541
71	0.3850233554840088
72	0.36347246170043945
73	0.5584931373596191
74	0.4054434299468994
75	0.49242329597473145
76	0.9409997463226318
77	0.9604735374450684
78	1.3234670162200928
79	1.4685320854187012
80	1.511120319366455
81	1.5060789585113525
82	1.722282886505127
83	1.6116664409637451
84	1.6546950340270996
85	1.641934871673584
86	1.3852894306182861
87	1.546724557876587
88	1.5990245342254639
89	1.333282709121704
90	1.3710923194885254
91	1.7289247512817383
92	1.527104139328003

93	1.7871065139770508
94	1.878084659576416
95	2.269184112548828
96	1.954625129699707
97	1.8780486583709717
98	1.5990221500396729
99	1.9502696990966797
100	1.9893338680267334
101	2.13036847114563
102	2.359166145324707
103	2.466498613357544
104	2.1337292194366455
105	2.317445755004883
106	2.399123430252075
107	2.4741029739379883
108	2.840160369873047
109	2.6035025119781494
110	2.830099582672119
111	2.6101508140563965
112	2.6345176696777344
113	2.9195640087127686
114	2.863537073135376
115	3.842611074447632
116	3.3479349613189697
117	3.7353899478912354
118	3.687554359436035
119	3.8671813011169434
120	3.476806163787842
121	3.8401756286621094
122	4.105631589889526
123	4.549307823181152
124	2.309020757675171
125	2.1381125450134277
126	2.566091775894165
127	4.455413818359375
128	4.563885927200317
129	4.831898212432861
130	4.924997806549072
131	4.761199712753296
132	5.236056566238403
133	5.088540077209473
134	4.9468841552734375
135	5.903693675994873
136	5.622583866119385
137	6.00100040435791
138	6.019036531448364
139	6.3389503955841064
140	6.388739585876465

141	6.579891681671143
142	6.787429094314575
143	6.832928895950317
144	7.020257472991943
145	7.105443239212036
146	7.4393086433410645
147	7.540153741836548
148	7.557368755340576
149	7.810436964035034

run time using numpy.dot

matrix size	run time
2	0.0
3	0.0
4	0.0
5	0.0
6	0.0
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0
17	0.000997781753540039
18	0.0
19	0.0
20	0.0
21	0.0
22	0.0
23	0.0
24	0.0
25	0.0
26	0.0
27	0.0
28	0.0
29	0.0
30	0.0
31	0.0
32	0.0
33	0.0
34	0.0
35	0.0
36	0.0

37	0.0
38	0.0
39	0.0
40	0.0
41	0.0
42	0.0
43	0.0
44	0.0
45	0.0
46	0.0
47	0.00015115737915039062
48	0.0
49	0.0
50	0.0
51	0.0
52	0.0
53	0.0
54	0.0
55	0.0
56	0.0
57	0.0
58	0.0
59	0.0
60	0.0
61	0.0
62	0.0
63	0.0
64	0.0
65	0.003906726837158203
66	0.0
67	0.0
68	0.0
69	0.004518747329711914
70	0.0
71	0.0
72	0.0
73	0.0007619857788085938
74	0.0
75	0.0
76	0.0
77	0.0
78	0.0010004043579101562
79	0.0
80	0.0
81	0.0
82	0.0
83	0.0
84	0.0

85	0.0
86	0.0
87	0.0
88	0.0
89	0.0
90	0.0
91	0.0
92	0.0
93	0.0
94	0.0009827613830566406
95	0.0
96	0.0
97	0.0
98	0.0
99	0.0
100	0.0
101	0.0009963512420654297
102	0.0
103	0.0
104	0.0
105	0.0
106	0.0
107	0.0
108	0.0
109	0.0
110	0.0
111	0.0
112	0.0
113	0.0010819435119628906
114	0.0
115	0.0
116	0.0009970664978027344
117	0.007955312728881836
118	0.0
119	0.0
120	0.0
121	0.0
122	0.0009965896606445312
123	0.0008084774017333984
124	0.0
125	0.0
126	0.0
127	0.0
128	0.0
129	0.0
130	0.0
131	0.0
132	0.0

133	0.0010294914245605469
134	0.0
135	0.0
136	0.008011817932128906
137	0.0
138	0.0
139	0.0
140	0.0
141	0.0
142	0.0
143	0.0
144	0.0
145	0.0
146	0.0
147	0.0
148	0.0
149	0.0