PHY407 Lab 1

(Yonatan Eyob Q2,Q3), (Kivanc Aykac Q1)
Student Numbers: 1004253309, 1004326222

September 18, 2020

Question 1

1.a

Nothing to submit.

1.b

Pseudo code is as follows:

```
– Set the initial conditions for position and velocity
– Set some constants
– Create a time list with time step, dt
– Calculate radius and determine velocity
– Update position then velocity
– Store position and velocity in order
– Plot velocity vs time
– Plot $x$ vs $y$
```

The key thing to remember here is that the current position and velocity is going to be used to update the velocity value and this new velocity is going to be used to update the position value, as the Euler-Cromer Method is being utilized here.

1.c

In this section, goal is to determine the Newtonian orbit of Mercury over several Mercurian years that take place in one Earth year using the Euler-Cromer

Method, and plotting velocity as a function of time and the orbit (x vs. y) in space. Additionally, the

$$\begin{aligned} x = 0.47\text{AU}, \quad & y = 0.0\text{AU} \\ v_x = 0.0\text{AU/yr}, \quad & v_y = 8.17\text{AU/yr} \end{aligned} \tag{1}$$

Where Sun's position is at $x = 0.0$, $y = 0.0$

The essential functions used in getting the plots below are as follows:
For Euler-Cromer Method calculation for determining the position and velocity:

```
def euler_cromer_posn_vel_rad(p, v):
    r = np.sqrt(np.sum(np.square(p)))
    v_new = v - dt*G*m_sun*p/(r**3)
    p_new = p + v_new*dt
    return p_new, v_new
```

For angular momentum calculation:

```
def ang_mom(position, velocity, mass):
    angmom = position[:,0]*mass*velocity[
        :,1] - position[:,1]*mass*velocity[:,0]
    return angmom
```

where first and second columns of both the "pos" (position) and the "vel" (velocity) are associated with $\hat{x}$ and $\hat{y}$, respectively.

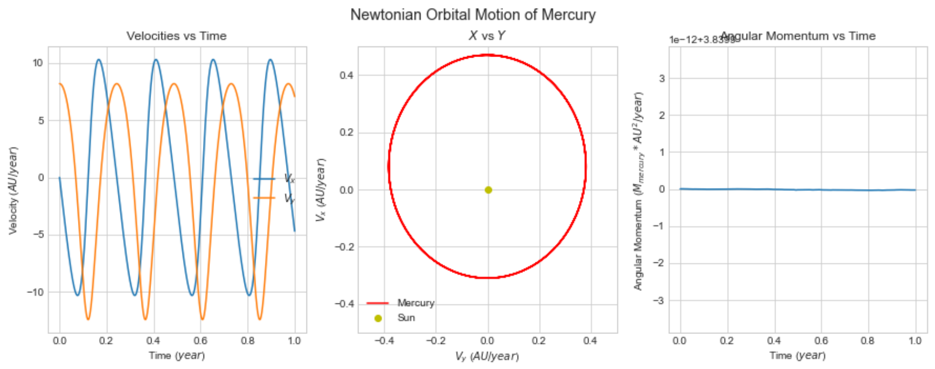


Figure 1: Newtonian Orbit of Mercury

As can be seen on Figure 1-"Velocity vs Time" graph, both of the velocities oscillate for a little more than four periods; implying that in 1 Earth year (time range used in generating the data), a little more than four Mercurian years pass. Also the asymmetrical nature of the two velocities suggest that the orbit is an ellipse, not a circle.
On Figure 1-"$xvsy$" graph, orbit of the planet is shown. Only an ellipse is visible, which actually contains a little more than four ellipses that are not visible

because of overlapping. The Sun does not move in the model therefore is indicated by a yellow dot at the focal point, at $(x = 0, y = 0)$.
On Figure 1-"Angular Momentum vs Time" graph, it's clearly visible that angular momentum is not changing since Python did not change the window margins for the plot to capture any difference in values. It is constant throughout $(\sim 3.8399 M_{mercury} * AU^2/year)$, which makes sense since a non-constant angular momentum would imply an external torque that is **not** present in this system.

1.d

In this section, goal is to plot the precession of the Mercury's perihelion (nearest point) by also considering the General Relativistic correction term for the gravitational force. The same initial conditions as the above section is used (eq.: 1). By taking α to be $0.01AU^2$, the effect is exaggerated for pedagogical aims.

$$\begin{gathered} v_x^{i+1} = v_x^i - dt\tfrac{GM_s}{r^3}x_i\left(1+\tfrac{\alpha}{r^2}\right) \\ x^{i+1} = x^i + v_x^{i+1}dt \end{gathered} \tag{2}$$

where $\alpha = 0.01AU^2$ for this model.
Accounting for the correction in 2, the updated Euler-Cromer Method function used is below:

```
def GR_euler_cromer(p, v, a):
    r = np.sqrt(np.sum(np.square(p)))
    v_new = v - dt*(G*m_sun*p/(r**3))*(1+a/(r**2))
    p_new = p + v_new*dt
    return p_new, v_new
```

As done similarly above, the plots are as follows:

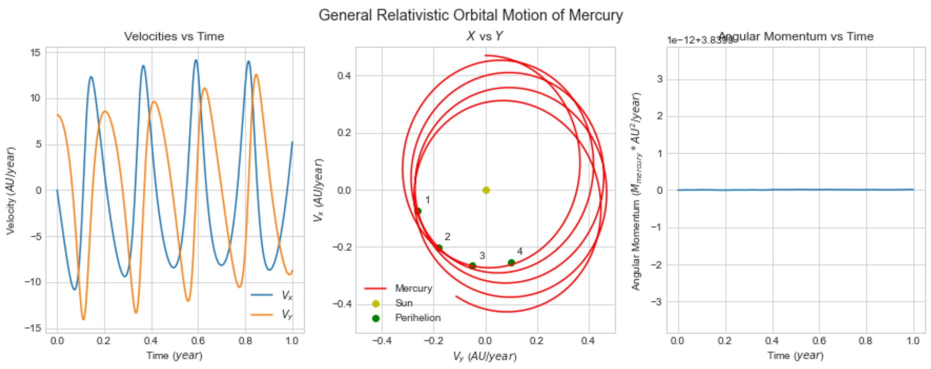


Figure 2: Newtonian Orbit of Mercury

As can be seen on Figure 2, the effect of the general relativistic correction term is significant when compared to the graphs above(Figure 1). First difference is the upwards shift on the "Velocity vs Time" graph as time evolves. This can be explained by observing the middle graph ($xvsy$): as the orbit precession evolves, the perihelion is not reached at the same position as one revolution ago, therefore the gravitational acceleration at a certain x is different from the one that affects Mercury one revolution after at the same x. This constant precession causes the velocities to change over time. This claim can be backed up by observing the different locations of the four points on the middle graph on 2 that represent four perihelion points that were reached in the four complete revolutions throughout the sampling. The associated number of the revolution to the perihelion is numbered next to the points.
Additionally, as can be seen on Figure 2-"Angular Momentum vs Time" graph, there is no angular momentum change as there is no external force just like in the above Newtonian case. And similarly, the constant value for it is: $\sim 3.8399 M_{mercury} * AU^2/year$.

Question 2

2.a

Pseudo-code to determine the evolution of the population of a species as a function of years is as follows:

- Define a function that requires 3 scalar values as inputs (initial normalized population (xp), growth rate (r), number of years considered (pmax))

- i equals zero, z equals an array of zeros with length equal to the number of years considered

- Make the first value of the z array equal to the initial normalized population (x0)

- make a loop that turns the elements in z into xp for year p from 0 to pmax

  1.Create loop from i=0 to i= pmax-1

  2.First step of the loop makes an element in z equal to r multiplied by (1 - previous element of z) multiplied by previous element of z, this starts from the second element of z

  3.Second (and final) step of the loop increases i by 1 (in (2b) ill include a script that demonstrates what i'm trying to pseudocode)

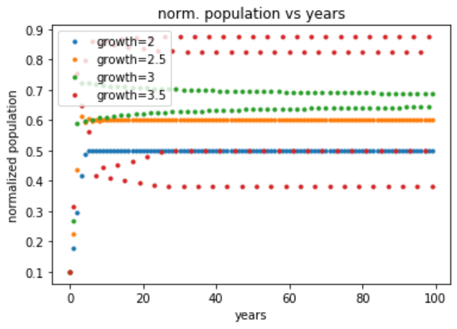
– Create an array that has a length equal to pmax and let the elements start at 0 and end at the pmax. make this array the x values and make the outputs of the function your y values.

2.b

Nothing to submit.

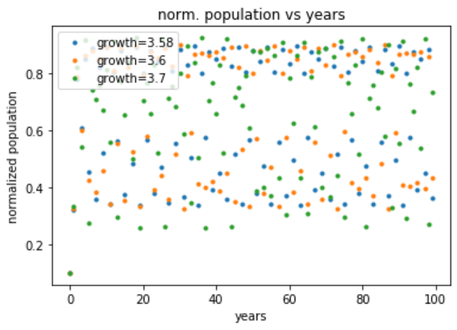
2.c

The evolution of the population over $p_{max} = 50$ for $r = \{2.0, 2.5, 3.0, 3.5\}$, with $x_0 = 0.1$ is plotted below.



growth-rate=2,2.5,3,3.5

When the growth rate is below 3 the xp vs p graph appears to approach some horizontal-asymptote (a single xp value) as years (p) increase. The horizontal-asymptote seems to get lower as the growth rate (r) gets lower. The plot trend gets more scattered as r increases above 3. At r=3 the xp vs p graph seems to alternate between 2 xp values when p is large. At r=3.5 it looks like the xp vs p graph alternates between 4 points when p is large.

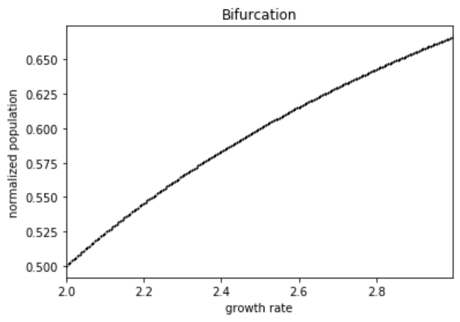


very chaotic when r≥3.58

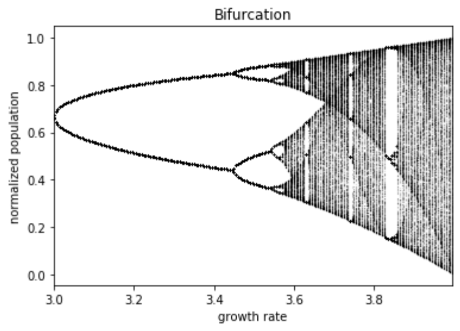
The graph gets very messy when r is greater than some point between 3.5 and 3.6 as shown in the graph above (it is hard to pinpoint exactly which r value).

2.d

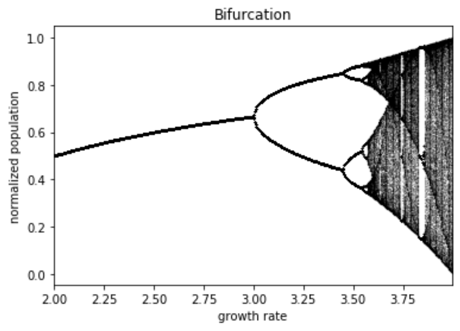
Below there are 3 xp vs r graphs plotted.The first graph only considers 2<r<3 and only plots the last 100 xp's from 2000 years, the second graph only considers 3<r<4 and only plots the last 1000 xp's from 2000 years, the third graph only considers 2<r<4 and only plots the last 1000 xp's from 2000 years.



pmax=2000 only considering the last 100 years



pmax=2000 only considering the last 1000 years



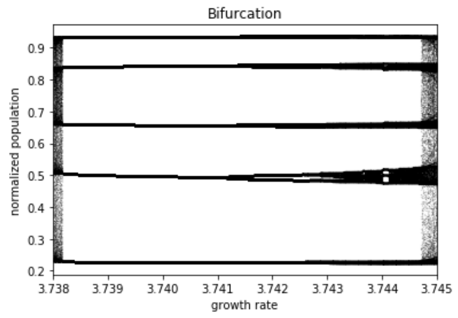
pmax=2000 only considering the

last 1000 years

The bifurcation diagram is a plot with r being the independent variable and xp being the dependent variable. In the previous plot, as p got bigger, xp alternated between a different number of values depending on r. The reason we were only concerned with the last 100 or last 1000 p when plotting the bifurcation diagram is because the diagram is interested in the points xp alternated between for large p for different r values. The beautiful pattern of the bifurcation diagram is the consequence of the alternating xp values for large p values in the previous plot. The bifurcation plots these xp values in relation to r values. The paths of the plots in the bifurcation diagram periodically doubles as r increases and it shows that the logistic map for r>3.6 can look very chaotic but it isn't truly random.

2.e

Similar to 2.d, but now for $3.738 \leq r \leq 3.745$ in increments of 10^{-5}.

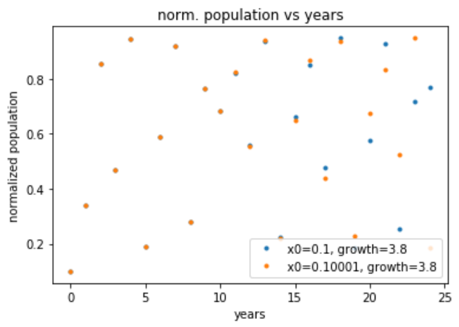


$3.738 \leq r \leq 3.745$

When we look at the bifurcation diagram in question 2d, the region $3.738 \leq r \leq 3.745$ looks messy but as we can see from the zoomed in graph above, it isn't as messy as it seemed. There are 5 paths that are doubling to 10 (around $r = 3.7415$).

2.f

Letting r=3.8, we will plot two different xp vs p evolution histories with slightly different initial conditions onto one graph and compare the outcomes.

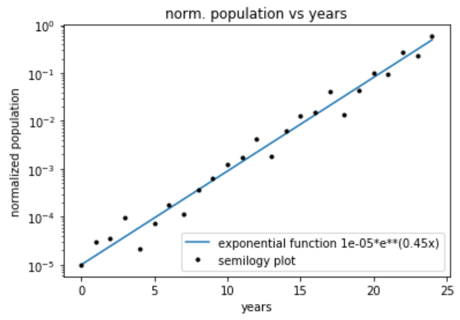


very small changes to the initial condition creates a very different outcome

I chose $x0^{(1)} = 0.1$ and $x0^{(2)} = 0.1 + 0.1/10^4$ and r=3.8. I picked r=3.8 because the bifurcation diagram looks very chaotic at r=3.8. I chose 25 iterations (pmax=25) because the graph in question (2g) stops growing exponentially at p=26. I changed the original condition by $1/100000^{th}$ and the change in the graph seems drastic. This is chaotic indeed.

2.g

Now we will be plotting s= $\|$xp$^{(2)} - xp^{(1)}\|$ vs p (s is the dependant variable, p is independent) on a semilogy graph and finding the Lyapunov exponent of the system.

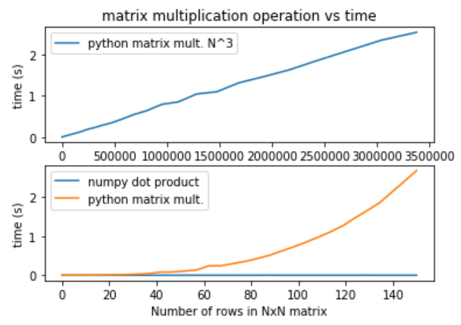


Lyapunov exponent=0.45 in this best-fit line

The agreed best-fit line is f(x)=$10^{-5}e^{0.45x}$ so Lyapunov exponent $\approx$ 0.45 in this system.

Question 3

In this section we will compare the operation speed of the np.dot function with the operation speed of the of a python script that does matrix multiplication manually.



according to the graph, the numpy dot function makes matrix multiplication a lot faster than
The bottom graph indicates that the np.dot() function does matrix multiplication MUCH faster than a script that does matrix multiplication manually.
The top graph became linear when the x-axis was cubed. This means that the time spent on a python script computing matrix multiplication manually with two NxN matrices is proportional to N^3.