PHY407: Lab 1

Date: September 17th. 2021

Lab Partners: Brendan Halliday and Summer Eldrige

Contributions:

- Q3. and the pseudocode for Q1b, Q2a completed by Brendan Halliday
- Q1, Q2 code, plots and explanatory notes completed by Summer Eldridge

Q1b.
Pseudocode:
import important libraries
#store useful constants and initial values (i.e. G and initial positions and velocities)
#initialize an empty lists for x and y positions and velocities
#initialize an empty time list as well

#repeat this next section of code for as many increments of t as necessary

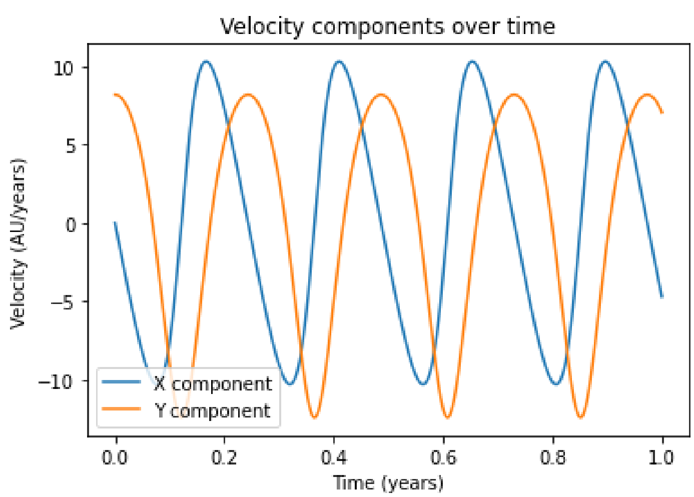
    #append initial time, x and y positions and velocities to their corresponding list
    # update time, x and y positions and velocities using the equations derived in part Q1 part a
    # i.e. use r=(x**2+y**2)**(1/2), vx=vx-dt*G*x*r**(-3), vy=vy-dt*G*y*r**(-3), x=x+dt*vx, and y=y+dt*vy
    # append updated values to their corresponding list

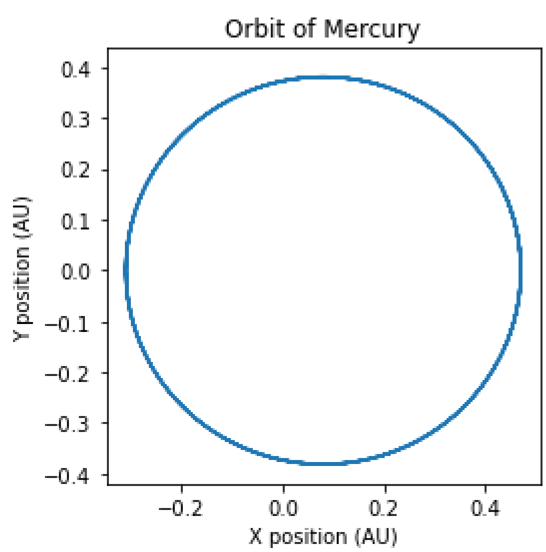
#plot orbits using list values previously calculated
#plot velocity components as a function of time

Q1c.
Code:

Code for this program is included as Lab01_Q1_c.py

Plots:

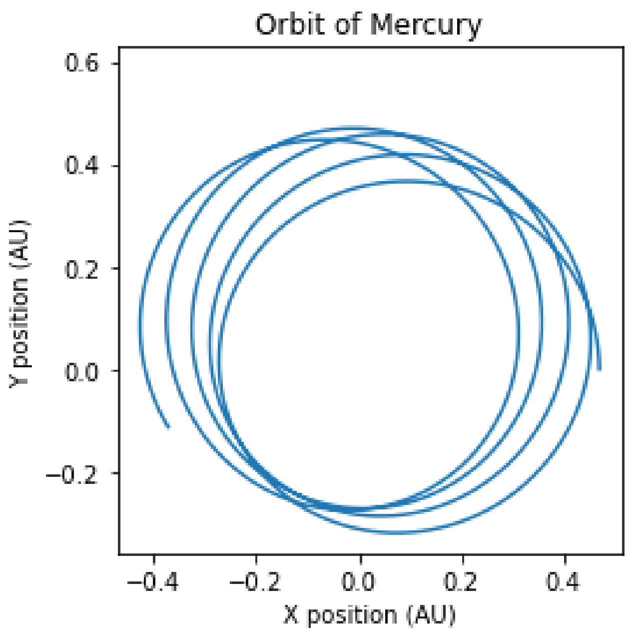




Q1d:
Code:

Code for this program is included as Lab01_Q1_d.py

Plots:



Q2a
Pseudocode:

store useful constants such as G, Mass of Jupyter(Mj)
and initial positions and velocities for both Earth and Jupyter
where (ex, evx, jx, jvx) and (ey, evy, jy, jvy) are the positions and velocities of the x component and y component
of Earth (e) and Jupyter (j) respectively.
initialize T=10 and time increment dt=0.0001
#initialize empty lists for the x and y positions and velocities for both planets and time

#repeat next section for T/dt time increments

#append updated positions and velocities of Jupyter and Earth to their corresponding lists

#update the position and velocity of Jupyter using
jr = (jx ** 2 + jy ** 2) ** (1 / 2)
jvx = jvx - dt * G * jx * jr ** (-3)
jvy = jvy - dt * G * jy * jr ** (-3)
jx = jx + dt * jvx
jy = jy + dt * jvy

update the position and velocity of earth using:
the distance of Earth from the sun er = (ex ** 2 + ey ** 2) ** (1 / 2)
evx = evx - dt * (G * ex * er ** (-3) + G * Mj * (ex - jx) * ejr ** (-3))
evy = evy - dt * (G * ey * er ** (-3) + G * Mj * (ey - jy) * ejr ** (-3))
ex = ex + dt * evx
ey = ey + dt * evy
where we take into account the gravitational effect of Jupyter by using

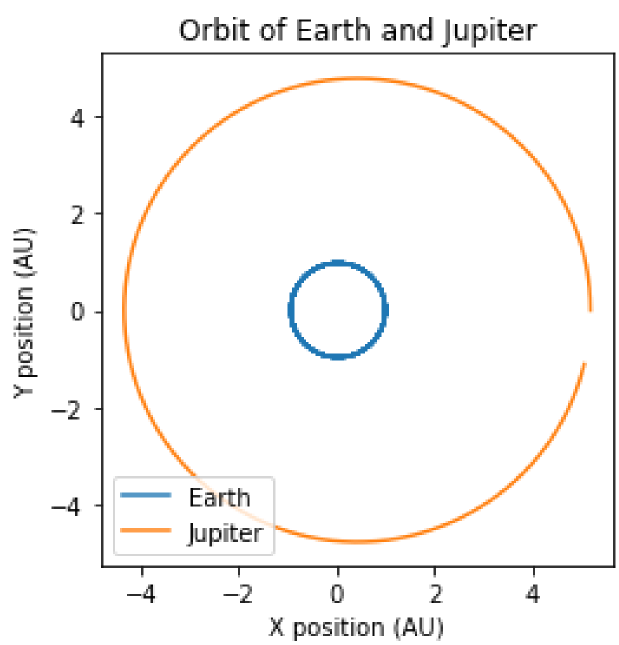
the distance between the two planets ejr = ((ex - jx) ** 2 + (ey - jy) ** 2) ** (1 / 2)

#plot the orbits for both Earth and Jupyter

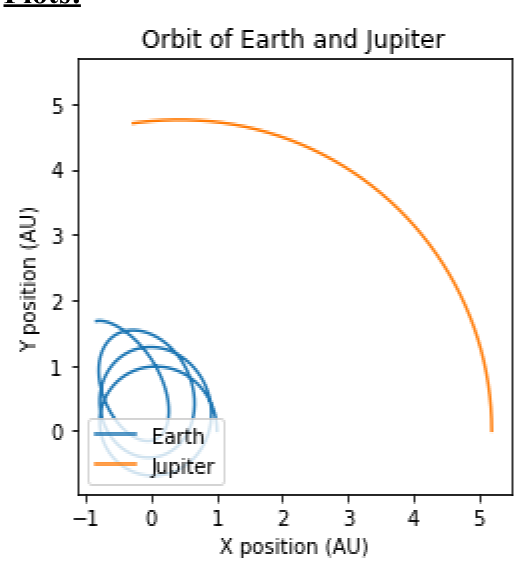
Code:

Code for this program is included as Lab01_Q2_a.py

Plots:

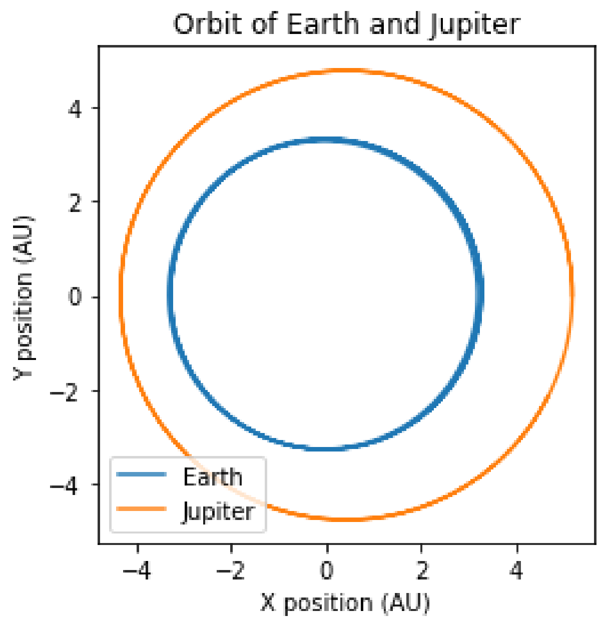


Q2b.
Plots:



If you go farther, earth swings too close to the sun and gets flung out of the solar system

Q2c.
Plots:



Q3a.
Pseudocode:

import libraries and helper module Lab1_Q3
store useful constants such as values for M and N

use my_bins() (see appendix) function to generate 1000 bins between -5 to 5

initialize empty list my_TIME for storing the time values
for my_hist() (see appendix)

This next block should be iterated for each value of N

    # generate R which is an array of N Gaussian distributed numbers
    # time my_hist() which is my histogram program from Lab1_Q3_helper_functions.py
    # append runtime to array my_TIME

#Appendix(These functions are defined at the beginning of Lab01_Q3_b.py)

#define function my_hist():
#This function takes in an array R of values and returns the histogram of those values
#initialize empty list
repeat the code below for each bin
    # initialize accumulator to integer value zero for each bin
    # repeat the code below for each value in R
        # for each j in R, check if value is within
        # the bin under question
        # if so, add one to the accumulator
        # if not, do not change accumulator value
    # once each j has been checked, append accumulator value to empty list

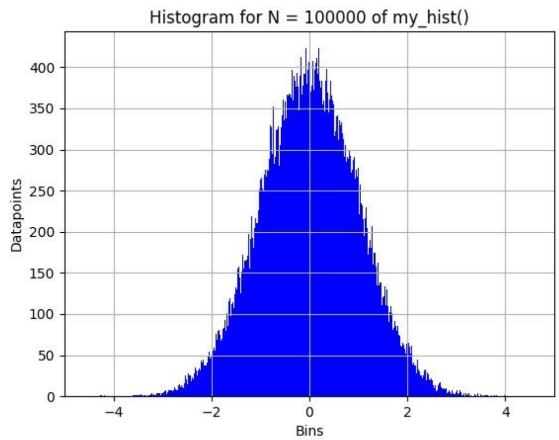
#define my_bins()
this function takes in an interval and number of bins and returns an array of equally spaced bins on a given interval
np.linspace is used

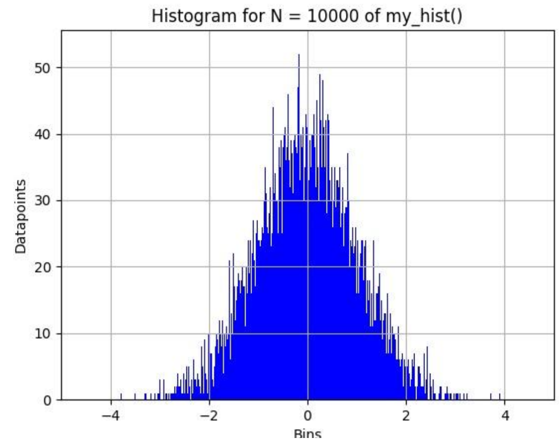
<u>Timing Values for my_hist():</u>

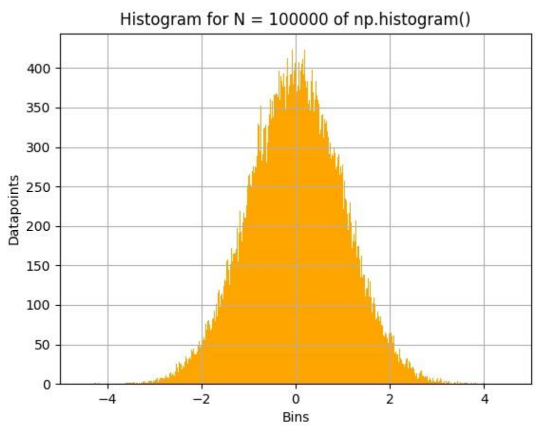
N=10, Time = 0.007s
N=100, Time = 0.07s
N=1000, Time= 0.6s
N=10000, Time= 6.7s
N=100000, Time= 63.1s
N=1000000, Time= 618.7s

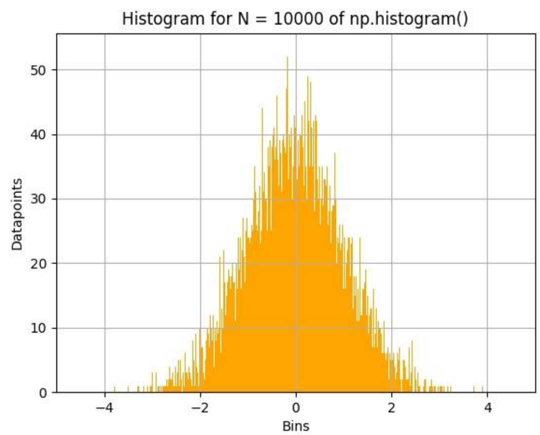
<u>Q3b.</u>

The following comparison shows that the two functions(my_hist() and np.histogram())produce identical results.



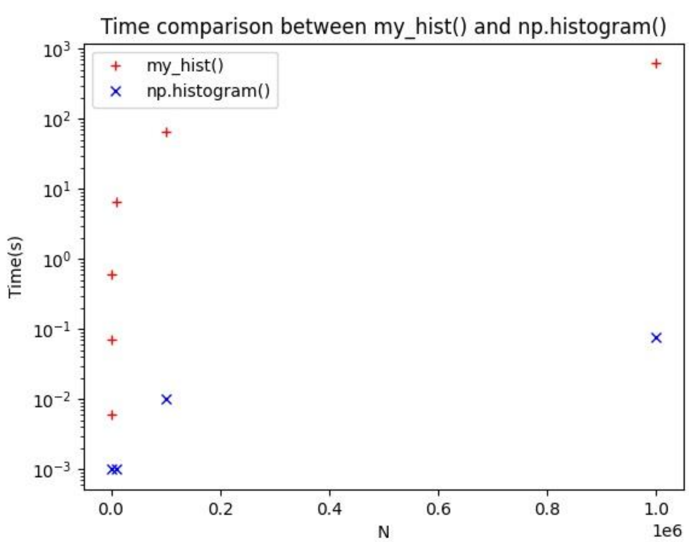






Timing graph:

The time graph below compares the speeds of the two functions where np.histogram is shown to be faster. This discrepancy is due to the fact that np.histogram uses a more efficient algorithm.



Code:

The code for this program is included in Lab01_Q3_b.py