PHYS281 Final Report

Simulating an N-Body Gravitational System Using Python

Finlo Heath

(Dated: December 14, 2018)

Abstract

Python is an object-oriented programming language with wide applications in data production and scientific analysis. The aim of this project was to develop an N-body gravitational system, specifically our solar system, using Python. This was achieved by developing two classes which, upon receiving the initial parameters for a body, update the acceleration, velocity and position of said body. A fixed time-step is used to dictate how often these parameters are updated and a data file is created containing the information of each body at each iteration of the system. Finally, an analysis file is used to present figures and data which allow the user to inspect the results of the simulation. the program produces spatial data with good accuracy in general and conserves fundamental quantities such as angular momentum and total energy to a reasonable degree. The project serves as an example of how computer programming can be utilised to produce large amounts of data which cannot be easily calculated analytically.

I. INTRODUCTION

In Phys281, we have developed an understanding of the foundations of Python and learned how it can be used in our degree and the wider world; particularly in running complex algorithms that cannot be performed analytically. The final project tasked students with simulating an N-body gravitational system. As advised, this project attempted to develop a simulation of our solar system including the Sun, all the planets, Earth's moon and Pluto.

The program approximates celestial motion using Newtonian dynamics. Relativistic effects are not accounted for and this is a potential source of error in the simulation, particularly in the orbit of Mercury[1]. Acceleration was calculated using Newton's equation for the acceleration of N massive particles moving due to the gravity of one another[2]:

$$\vec{g}_i = \sum_{j\neq i}^{N} \frac{-Gm_j}{r_{ij}^2}\hat{r}_{ij} \tag{1}$$

Where $\vec{g}_i$ is the acceleration vector of body i, $G = 6.67 \times 10^{-11} Nm^2Kg^{-2}$[3] is Newton's Gravitational Constant, m_j is the mass of body j, $\hat{r}_{ij}$ is the unit vector of the separation of bodies i and j where r_{ij} is the magnitude of that separation.

Equation 1 enables the calculation of the acceleration of each body in the system, including the Sun and is applicable to all bodies in the system by assuming that all bodies in the system have spherically symmetric mass distributions, such that we may consider them as point masses [3]. This acceleration vector is then used in conjunction with velocity and position vectors in order to be able to track a particle's motion over a time period. Two methods of obtaining the velocity and position were implemented in my program, the Euler-Cromer algorithm[2]:

$$\begin{aligned}\vec{v}_{n+1} &\approx \vec{v}_n + \vec{a}_n\Delta t\\ \vec{x}_{n+1} &\approx \vec{x}_n + \vec{v}_{n+1}\Delta t\end{aligned} \tag{2}$$

and the Euler forward method[2]:

$$\begin{aligned}\vec{v}_{n+1} &\approx \vec{v}_n + \vec{a}_n\Delta t\\ \vec{x}_{n+1} &\approx \vec{x}_n + \vec{v}_n\Delta t\end{aligned} \tag{3}$$

Where for both sets of equations, $\vec{v}$ is the velocity vector, $\vec{x}$ is the position vector, $\vec{a}$ is the acceleration vector and n and $n+1$ are the n^{th} and $(n+1)^{th}$ iterations of their respective vectors. Δt is the time-step between iterations, that is, the amount of time between updating each velocity and position vector. Equation 1 is run the over the same number of iterations so that the three equations work in unison.

The more appropriate of these two methods for this system is the Euler-Cromer algorithm, given that it is a modification of the Euler Forward method which yields more stable solutions for oscillatory systems[4]; although the user may choose which they would prefer to use when running the simulation.
Equations 1, 2 and 3 form the mathematical basis for our simulation and allow the bodies to orbit with elliptical motion in accordance with Kepler's first law[3]. The Simulation runs for a total time, updating the values using the above equations every time-step and recording them at each time-step in order to track change in the values over the simulation.

In reality time is continuous, not discrete, but modelling with time-steps allows us to apply the simulation to scenarios where acceleration is not constant by implementing the ability to update the acceleration at regular intervals. This operates under the approximation that the acceleration is constant during the time between each step and is therefore more accurate when the time-step used is smaller[5]. This is discussed in further detail in Section III.

The remaining body of the report contains three main sections. Section II details how the design of the program evolved. Section III provides analysis on the data produced by the simulation and how our approximations have affected its accuracy. Section IV is a discussion of the overall performance of the simulation and potential improvements which could be made.

II. DESIGN

The program went through distinct stages in its development. Beginning with basic projectile motion under constant acceleration, It was then extended and adapted the program to the stage at which it can simulate an N-body gravitational system. This section is split into the main way-points in developing the code.

A. Implementing Acceleration as a Variable

The projectile motion work consisted of a contained class which utilised the Euler forward method (Equation 3) to update position and velocity. Development began by changing this to the more stable Euler-Cromer algorithm.

Next, A second file was made containing a Class with methods to implement Newtons gravitational acceleration, Equation 1. By importing the projectile motion class, the acceleration class ran a method to update the acceleration at each time-step. In the test file, the acceleration update method is run before the Euler-Cromer update method to ensure that the velocity and position update correctly at each stage.

Once the acceleration method was implemented, the program was able to simulate x-y motion of the Earth and its moon, a two-body gravitational system, for a desired time period. This is shown in FIG. 1.

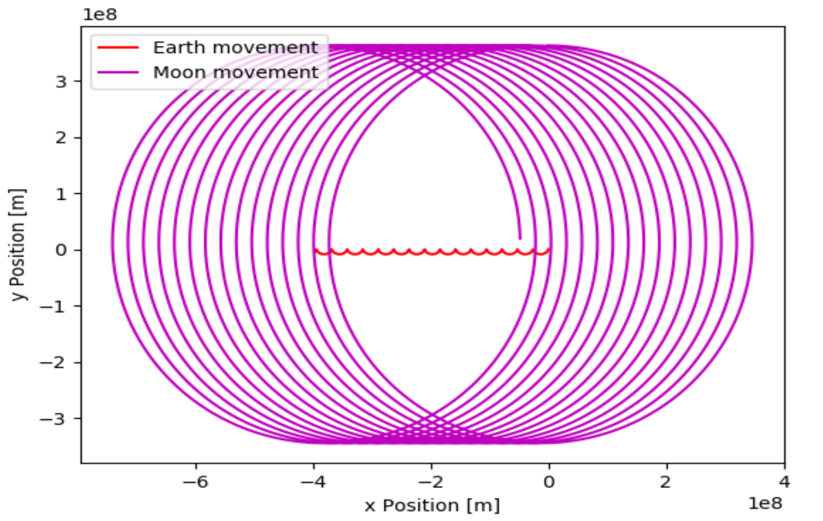


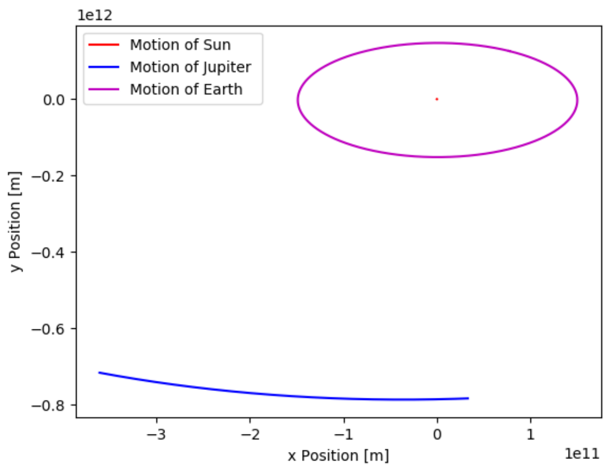
FIG. 1. A graph displaying the positions of the Earth and its Moon in the x-y plane over a period of one year, where the Earth started at the origin.

B. Extending to Three and then N Bodies

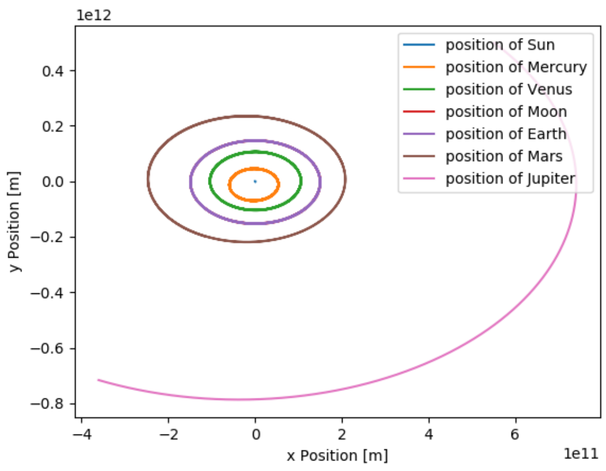
The Challenge of creating a three-body system is the necessity of loops to ensure that the acceleration update method runs such that it obeys Equation 1 for all i and j. This required correctly indexing the list of bodies in the system to access the parameters of the bodies in the list such as mass or velocity components. For the three-body system, a trial of the Sun, Earth and Jupiter was selected. This is shown in FIG. 2.

The jump from a three-body to an N-body system consisted mainly of automating the analysis process in the test file, rather than editing the two classes. This involved automating the plotting of each body's motion in the x-y plane onto the same figure through indexing the list of bodies rather than creating new lists for the x and y positions of each body. A simulation of seven bodies was used, shown in FIG. 2.

Clearly, this result of the planets orbiting the Sun and the Moon orbiting with Earth, visible only upon zooming in on the purple Earth orbit, is superficially true to the result we expect. Whether or not they are quantitatively accurate will be discussed in Section III.



(a) Three-Body System



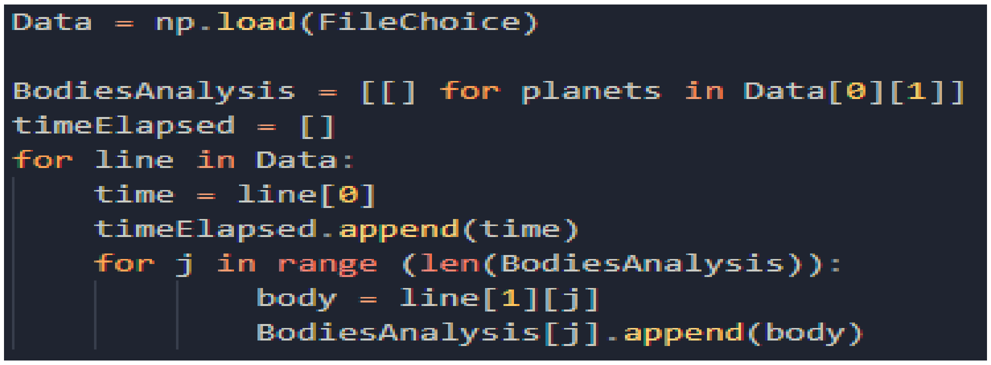
(b) Seven-Body System Using N-Body Program

FIG. 2. A graph displaying the positions of celestial Bodies in the x-y plane where the Sun started at the origin. Left is a three-body system over a period of one year. Right is a seven-body system over a period of 5 years.

C. Writing to a Separate file & Analysis

Following the creation of a working N-body system, the major remaining task was to restructure the program such that it created a data file and then performed analysis on this data file separately. This consisted of gathering all the iterations of the bodies with each of their parameters at each iteration into a single data file, along with a list of equal length giving the time at each iteration.

the analysis file then retrieves the list of Bodies and times at each iteration by using index notation. An example of this is given below in FIG. 3. After this step, loops to produce lists of the position of each body and the angular momentum and kinetic energy of the system were developed; then corresponding functions to produces figures of each were added.



```
Data = np.load(FileChoice)

BodiesAnalysis = [[] for planets in Data[0][1]]
timeElapsed = []
for line in Data:
    time = line[0]
    timeElapsed.append(time)
    for j in range (len(BodiesAnalysis)):
        body = line[1][j]
        BodiesAnalysis[j].append(body)
```

FIG. 3. A figure displaying code using index notation to create a list of lists of the iterations of each body from the data file (FileChoice) and a list of the time elapsed at each iteration. Data is a list of lists of the time (index 0 in Data) and the parameters of each body(in distinct lists for each body at index 1 in Data), at each iteration. The code is taken from *Analysis1.2.py* which forms part of the final simulation.

D. Final Program

The final amendments to the program involved correcting and further automating the methods. This included adding the ability for the user to input key variables during the running process, the details of this are outlined in

FIG. 4. Another small but important change was to save data ever 100 time-steps when running the simulation rather than every time-step. This decreases the time the program takes to run a given simulation without sacrificing the accuracy acquired via a small time-step.

Referring back to the SolarSystem Class, which contains the acceleration update method, a gravitational potential energy method was implemented using the same logic and replacing the acceleration equation with Newton's equation for the gravitational potential energy of N massive particles due to the gravity of one another[6]:

$$U_i = \sum_{j \neq i}^{N} \frac{-G m_i m_j}{r_{ij}} \tag{4}$$

Where U_i is the gravitational potential energy of body i and m_i its mass and the remaining variables are defined as in equation 1.

The gravitational potential energy method allowed for the calculation of the total energy of the system at each iteration and thus a corresponding graph. The simple equation for total energy incorporates Virial theorem[7], which states that for an N-body system, the following equation holds true:

$$Constant = 2 \times KineticEnergy + PotentialEnergy \tag{5}$$

Where potential energy is negative, as defined in equation 4. This means that the quantity $2 \times KineticEnergy + PotentialEnergy$ should be conserved in the simulation.

The angular momentum calculation was corrected by creating a list of the position vectors of the centre of mass of the whole system. Prior to this, the assumption had been made, incorrectly, that the centre of mass is the origin given that this is the starting position of the Sun. Not only is this incorrect for the initial centre of mass, the system as a whole moves over time due to the gravitational pull of the other bodies, hence this change was necessary.

The flow chart, FIG. 4 shows the overall design of the final program. The final Analysis output of the program is graphs displaying the positions of the bodies in the x-y plane, followed by graphs of the angular momentum and kinetic, potential and total energies of the whole system, over the total time that the simulation runs for. The program also prints the average, standard deviation, minimum and maximum values when the given graph is displayed.

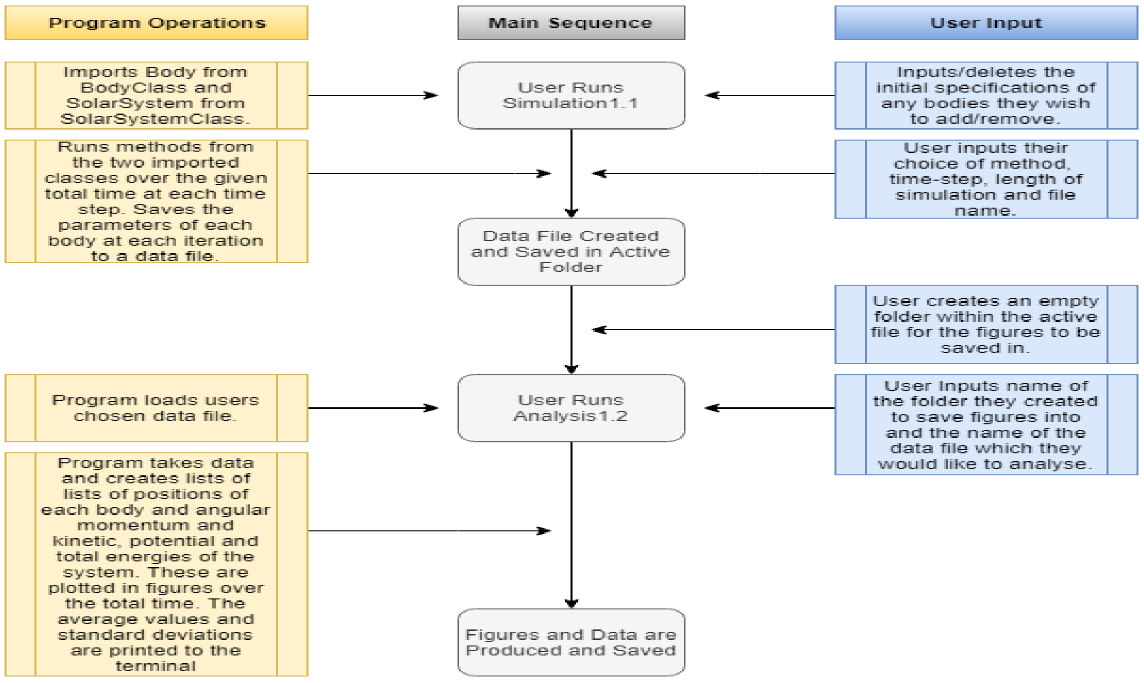

Program Operations
Main Sequence
User Input
Imports Body from BodyClass and SolarSystem from SolarSystemClass.
User Runs Simulation1.1
Inputs/deletes the initial specifications of any bodies they wish to add/remove.
Runs methods from the two imported classes over the given total time at each time step. Saves the parameters of each body at each iteration to a data file.
User inputs their choice of method, time-step, length of simulation and file name.
Data File Created and Saved in Active Folder
User creates an empty folder within the active file for the figures to be saved in.
Program loads users chosen data file.
User Runs Analysis1.2
User Inputs name of the folder they created to save figures into and the name of the data file which they would like to analyse.
Program takes data and creates lists of lists of positions of each body and angular momentum and kinetic, potential and total energies of the system. These are plotted in figures over the total time. The average values and standard deviations are printed to the terminal
Figures and Data are Produced and Saved

FIG. 4. A flow chart which outlines the design of the final program, showing how the program runs an N-body gravitational simulation and the required user input.

E. Potential Design Improvements

Further features which would have been desirable in the final program include adding a third method by which the user can update the position and velocity and implementing classes into all files. to achieve the latter, a complete rewrite of the simulation and analysis files would be required. This was not feasible in the time frame given that one large rewrite had already occurred in moving from a single test file to the model where a new file is created.

With regard to including a third update method, attempts were made to utilise the Verlet algorithm, equation 6, which involves using the acceleration a time-step ahead in order to transition more smoothly in large acceleration changes [2]. Conceptually, this requires calculating the acceleration of a given step n using either the Euler-Cromer or Euler Forward methods and then using that to calculate the velocity and position of step $(n-1)$. In practice, this was not accomplished.

$$\begin{aligned} \vec{x}_{n+1} &\approx \vec{x}_n + \vec{v}_n \Delta t + \frac{1}{2}\vec{a}_n(\Delta t)^2 \\ \vec{v}_{n+1} &\approx \vec{v}_n + \frac{1}{2}(\vec{a}_{n+1} + \vec{a}_n)\Delta t \end{aligned} \tag{6}$$

The Verlet algorithm. Where the variables are defined as in equation 2.

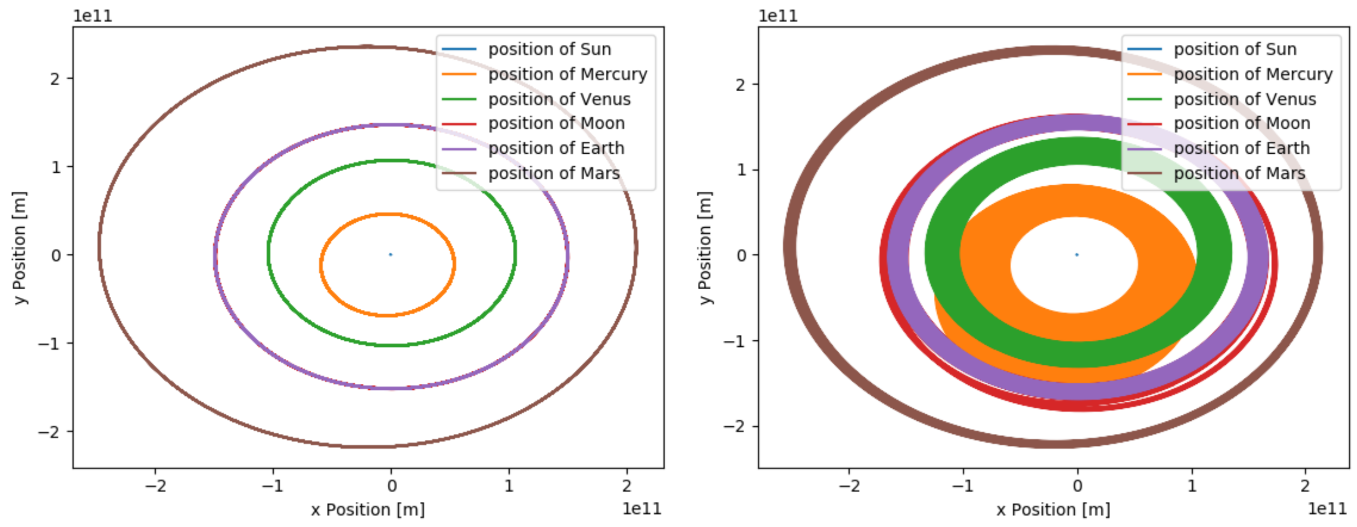
Finally, fits on the figures produced by the simulation were not implemented. Attempts to do this appropriately through the use of *numpy.poly1d* were made but overall a curve which fitted correctly was not obtained. Clearly the use of a more complex fitting method is required to accomplish this and would have been implemented given more time.

III. RESULTS ANALYSIS

This section details the results produced by the final program using the analysis file to assess different data files produced by the simulation.

A. Effects of Method and Time-Step

As mentioned in the section prior, the program contains two methods by which the position and velocity vectors can be updated; the Euler-Cromer method being a more stable adaption of the Euler Forward method[4]. This is reflected in the results produced by the simulation, shown in FIG. 5



(a) Euler-Cromer (b) Euler Forward

FIG. 5. Graphs displaying the positions of celestial bodies in the x-y plane where the Sun started at the origin, using a time-step of 1000 seconds, over a total of 50 years.

Clearly the deviation in the orbits is much larger using the Euler Forward method with sections where the Earth and Moon orbits do not overlap. The table below quantitatively displays the differences in the two graphs.

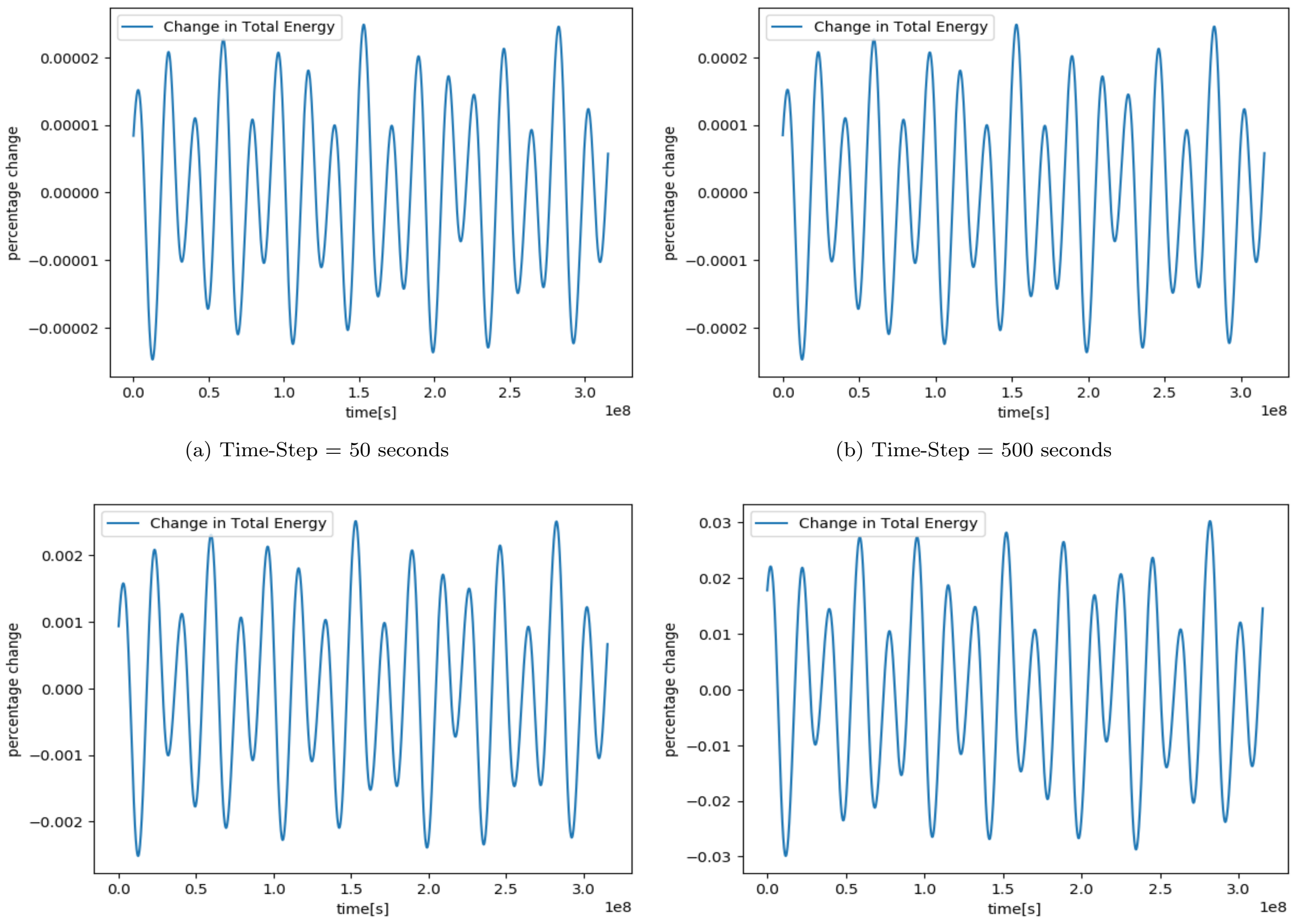
Body	$\langle r_c \rangle$	σ_{r_c}	$\langle r_f \rangle$	σ_{r_f}
Sun	0.00	0.00	0.00	0.00
Mercury	0.395	0.0558	0.675	0.190
Venus	0.703	0.00929	0.808	0.0570
Moon	1.00	0.0120	1.10	0.0549
Earth	1.00	0.0119	1.06	0.0350
Mars	1.53	0.100	1.56	0.104

Values given in astronomical units [au]

TABLE I. A table displaying the average radial distance of each body from the sun over a 50 year period using the Euler-Cromer (FIG.5a) and Euler Forward (FIG.5b) methods.
$\langle r_c \rangle$ is the average radial distance from the sun using the Euler-Cromer algorithm, $\langle r_f \rangle$ is the average radial distance from the sun using the Euler Forward method.
The standard deviation in the radial distances using the Euler-Cromer and Euler Forward methods are given by σ_{r_c} and σ_{r_f} respectively.

We can clearly see that the Euler-Cromer method is performing accurately given that Earth's average radial distance from the sun is 1.00au, which is the correct definition of one astronomical unit[9]. Note that the Moon orbits primarily around Earth, the consequence of this being its secondary orbit about the sun. It is this secondary orbit presented in TABLES I & II.

The size of the time-step dictates the resolution at which the simulation will be valid, the smaller the time-step, the more accurate the simulation, especially when looking at smaller segments of a body's orbit[5].The figure below displays the difference between four different time-steps all running the Euler-Cromer method.



(a) Time-Step = 50 seconds

(b) Time-Step = 500 seconds

(c) Time-Step = 5000 seconds

(d) Time-Step = 50000 seconds

FIG. 6. Graphs displaying the percentage change in the total energy of a 5 body system, over a total of 10 years, where the time-step used increases by a factor of 10 with each graph going from (a) to (d).

It is clear from FIG. 6 that the time-step is linearly, inversely proportional to the percentage change in total energy of the system. That is to say that reducing the time-step by a factor of β (where β is an arbitrary value) will increase the accuracy of the change in total energy of the system by β.

This clearly demonstrates how a smaller time-step produces more accurate results; which is sensible given that it is simply increasing the frequency with which the simulation is updated. However, the number of calculations incurred by reducing the time-step by β increases by the same same factor β and consequently so to does the run time. In summary:

$$\Delta T \propto \frac{1}{\Delta E_{tot}} \propto \frac{1}{N_{calc}} \tag{7}$$

Where ΔT is the time-step, ΔE_{tot} is the change in total energy of the system over time, N_{calc} is the total number of calculations which must be performed in running the simulation.

This is evidence that the accuracy of the Euler-Cromer algorithm itself scales inversely with the size of the time-step used as expected[5].

A simulation which tests the limits of both the choice of method and time-step is computing a large number of bodies over a long time period. This meant executing an eleven body simulation of a 250 year period, displayed below.

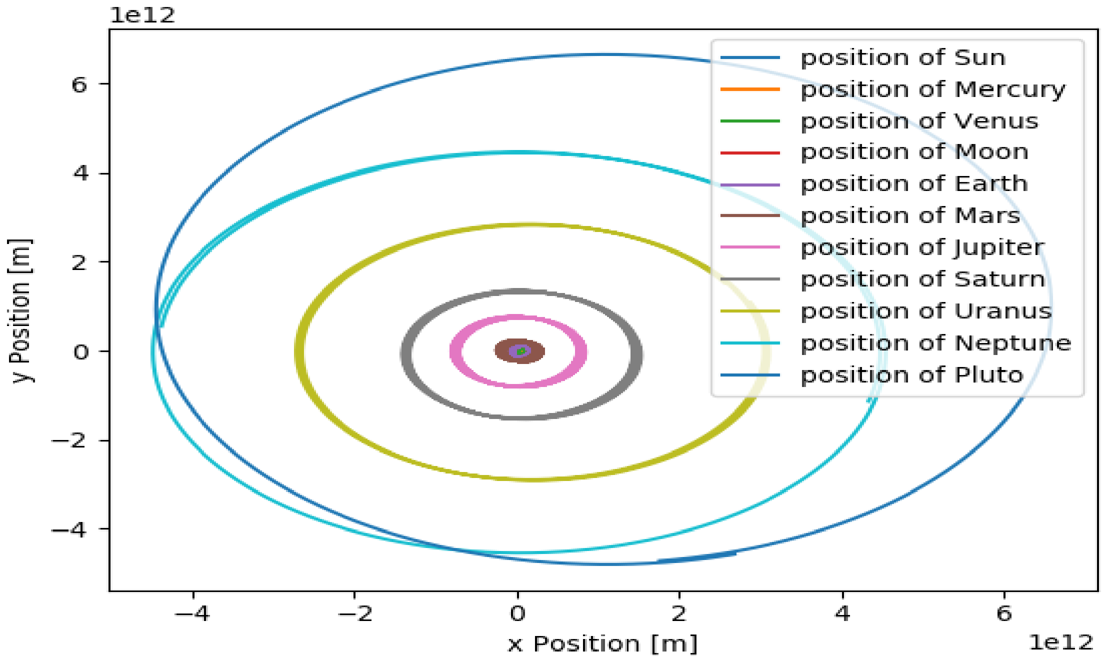


FIG. 7. A Graphs displaying the positions of celestial bodies in the x-y plane where the Sun started at the origin, using a time-step of 1000 seconds, over a total of 250 years.

This plot displays clearly that the system as a whole undergoes translational motion as a whole, producing thicker lines as each orbit laps past its former. Under the given time-step of 1000 seconds, it is clear that the simulation did not deteriorate despite the run-time of 250 years. Allowing for the overall translational motion of the system, it is reasonable to argue that Pluto completed just over one orbit in the 250 years, a sensible result given Pluto's actual orbit is 248 years[10].

Despite the accuracy of the Euler-Cromer method, it still incurs a very long run-time as the computational time is proportional to the square of the number of bodies in the system[5]; the data file which produced FIG. 7 required just under three hours to create. A method which could improve this is the Barnes-Hut algorithm which greatly reduces the number of body interactions calculated[5]. This is done by approximating all particles in a given far away region as a single particle at the centre of mass of that region and involves defining a 'far away' and each region in the volume of the system[8].

B. Orbital Period

Using the data from TABLE I, we can calculate the average orbital period of each body (appendix A). These are compared to data from NASA[10] in the table below. Note that the Euler-Cromer algorithm is used as we are only concerned with the accuracy of our best method.

Orbital Period about the Sun (Years)

	Euler-Cromer Data		**NASA Data**
Body	$\langle T_c \rangle$	σ_{T_c}	T_{NASA}
Sun	0.00	0.00	0.00
Mercury	0.248	0.0132	0.241
Venus	0.589	0.000895	0.615
Moon	1.00	0.00131	1.00
Earth	1.00	0.00130	1.00
Mars	1.89	0.0316	1.88

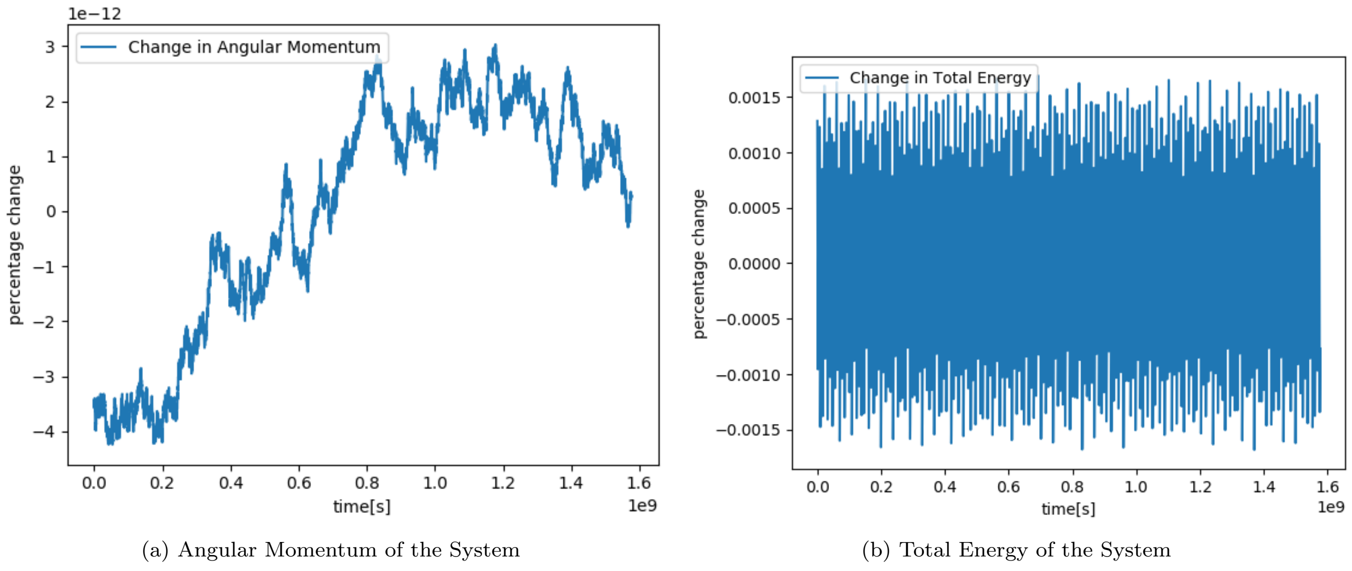
Values given as decimals of Earth Years

TABLE II. A table displaying the average orbital period of each body about the Sun over a 50 year period using the Euler-Cromer method (FIG.5a) and a time-step of 1000 seconds.
$\langle T_c \rangle$ is the average orbital period about the Sun using the Euler-Cromer algorithm, the standard deviation in the orbital period is given by σ_{T_c}. T_{NASA} is the average orbital period given by NASA[10]

Looking at TABLE II, we observe that the The Moon, Earth and Mars all align with the NASA data within the standard deviation. The Closer bodies, Mercury and Venus, do not. An obvious possible cause of the disparity in the Mercury data is relativistic effects[1] which, as mentioned in the introduction, are significant and unaccounted for by the program. Such effects are not significant enough to be causing this lack of precision in the Venus orbit however. This means it is likely an error in the data used in the code causing the lack of agreement or possibly the use of too large a time-step, reducing the precision of the data values in the bodies with smaller orbits.

C. Angular Momentum and Energy Conservation

Angular momentum and the total energy are fundamental quantities of the N-body system and as such, they should be conserved[11]. Thus a sensible parameter for the success of the system is the change in these quantities over time. As conserved quantities, we expect that an ideal N-boy simulation would produce zero percentage change in angular momentum and total Energy of the system over a given time. Since discrete rather than continuous methods were used one cannot expect a perfect result, however, the smaller the percentage change, the more accurate the simulation. FIG. 8 displays the results of this analysis using the simulation.



(a) Angular Momentum of the System

(b) Total Energy of the System

FIG. 8. Graphs displaying the change in two fundamental quantities of the system. This is a 6 body system over a period of 50 years, using a time-step of 1000 seconds through the Euler-Cromer algorithm.

First, discussing FIG. 8a. It is clear that, despite the erratic appearance of the data, the percentage change in angular momentum of the system is negligible. In essence, the system conserves angular momentum to an order of $10^{-12}\%$. This is evidence that the system has been successful in conserving this quantity. This success comes as a consequence of adding the function to accurately calculate the centre of mass at each iteration, discussed in Section II.D.

This cannot be said for FIG. 8b to the same degree. In Section II it was stated that the Virial theorem should hold for our system[7]. The percentage change in total energy of the system shown in FIG. 8b is derived from the Virial theorem. The system has an overall error of the order $10^{-3}\%$ in its conservation of total energy. This is a good result although too large to consider negligible and so insinuates an error in either the code producing the data or that which analyses it. One sensible physical explanation for this amount of error could be including mercury in the system since we did not consider the implications on energy conservation of the relativistic effects of its orbit[1].

Looking back at FIG. 6, in particular FIG. 6a, we see that for a 5 body system using a time-step of 50 seconds over a period of 10 years, the percentage change in total energy of that system is smaller by a factor of roughly 60 times. The difference between the time-step used in these two simulations is only a factor of 20, so a factor of 3 decrease in accuracy remains unaccounted for by the change in time-step. This is most likely due to the introduction the sixth body, Jupiter in FIG. 8. Other than the Sun, Jupiter's mass is the largest of the bodies in the solar system so it has a large effect on the orbits of the other bodies when introduced.

FIG.8b oscillates fairly consistently over the 50 year period so we can ascertain that the increase in length of the simulation is not the cause of the inaccuracy. This oscillation is too frequent to be distinguished in FIG. 8b, but becomes clear when using the program as one is able to magnify the figures.

IV. SUMMARY

Considering the results outlined in the previous section, we can conclude that overall, the N-body system is simulated with reasonable accuracy. The program simulates any given number of bodies well with the main caveat being the increase in the length of time it takes to run.

While there is the option to use either the Euler-Cromer algorithm or Euler Forward method, the inaccuracy of the latter is much greater given the use of an oscillatory system. Therefore all reasonable results given by the program are using the Euler-Cromer algorithm.

Of all the bodies that have been incorporated in trials of the simulation, only three present notable issues. The data for Mercury's orbital period, and thus its average orbital radius using equation A1, is not correct within the standard error given by our data (Appendix B):

$$\begin{aligned} \langle T_{Mercury} \rangle &= 0.2485 \pm 0.0001 \quad Earth - Years \\ \langle T_{NASA} \rangle &= 0.241 \quad Earth - Years \end{aligned} \tag{8}$$

To be in reasonable agreement, $\langle T_{NASA} \rangle$ would have to be in range of two standard errors of $\langle T_{Mercury} \rangle$[12] which it is not. Again, it is likely due to the unaccounted for relativistic effects which are significant given the speed at which Mercury orbits the sun[1]. This point has been reiterated many times but it is prudent to do so given that this effect is arguably the most significant physical phenomenon unaccounted for by the program.

Pluto did not pose any issue when it came to the accuracy of the data but due to the length of its orbital period, a simulation must run for a very long time in order to gather meaningful data on Pluto; demonstrated by FIG. 7. One solution to this would be to implement a separate time-step for the more distant bodies such as Pluto and Neptune. This would allow for the program to perform fewer calculations and so the simulation would take less time to run. As these bodies have a much lower orbital velocity than those with smaller average radial distances[13], there is a smaller percentage error created by updating their velocity and position values less frequently. This is a feature which would be investigated further if given more time.

The final body which requires discussion is Venus. like Mercury, the mean orbital period of Venus calculated by the simulation was not within two standard errors of the true value[10] and so was not within reasonable agreement[12]:

$$\begin{aligned} \langle T_{Venus} \rangle &= 0.589112 \pm 0.000007 \quad Earth - Years \\ \langle T_{NASA} \rangle &= 0.615 \quad Earth - Years \end{aligned} \tag{9}$$

Not only is there disagreement between the two values, the standard error of $\langle T_{Venus} \rangle$ is the smallest of any of the bodies in the six body simulation from which the data was taken. This heavily implies an error in the code logic rather than a lack of precision and is therefore a fundamental flaw in the operation of the program as there is no physical explanation for the program producing data which does not agree with the true value. Resolving this issue would be the primary goal in developing a more accurate simulation.

To conclude, in general the final iteration of the program can simulate an N-body gravitational system to a reasonable accuracy for bodies which do not experience significant relativistic affects. The program adapts, without requiring restructuring, to new data although due to the use of the Euler-Cromer algorithm alone, it becomes exponentially slower with each body added to the simulation.

If given more time to further improve the program, the focal point would be on identifying and correcting the code which is causing disagreements between the simulation data and true values. Following this, the conservation of the total energy of the system could be investigated.

Work to further the functionality of the program could include implementing a third, quicker or more accurate method as well as adding appropriate fits to each of the figures in order to quantify the behaviour of each of the parameters using mathematical functions.

V. BIBLIOGRAPHY

[1] Conover E. ScienceNews: Einsteins general relativity reveals new quirk of Mercurys orbit (2018). [online] available at: https://www.sciencenews.org/article/einstein-general-relativity-mercury-orbit [Accessed 5^{th} December 2018]

[2] Bertrum I, Long R. PHYS281 Programming and Modelling Project:An introduction to scientific programming using Python (2018). [online] available at: https://modules.lancaster.ac.uk/pluginfile.php/1781020/mod_resource/content/5/Course_Notes_2018_11_02.pdf [Accessed 8^{th} December 2018]

[3] Young HD, Freedman RA. University Physics with Modern Physics. 14th Edition. Ford AL, contributing author. Edinburgh Gate, Harlow, Essex, England: Pearson Education; 2016.[Chapter 13].

[4] Nikolic BK. University of Delaware: Computational Methods of Physics $^4_6$60: Euler-Cromer Method (2018). [online] available at: http://www.physics.udel.edu/~bnikolic/teaching/phys660/numerical_ode/node2.html [Accessed 8^{th} December 2018]

[5] Pretorius F. University of Princeton: N-body Simulations. [online] available at: http://physics.princeton.edu/~fpretori/Nbody/intro.htm [Accessed 9^{th} December 2018]

[6] Nave R. HyperPhysics Concepts: Gravitational Potential Energy. [online] available at: http://hyperphysics.phy-astr.gsu.edu/hbase/gpot.html [Accessed 10^{th} December 2018]

[7] Clausius RJE. Philosophical Magazine: "On a Mechanical Theorem Applicable to Heat". 1870. Series 4. 40: 122127.

[8] Barnes J, Hut P. Nature. 324 (4): 446449: A hierarchical O(N log N) force-calculation algorithm. 1986. [online] available at: https://www.nature.com/articles/324446a0[Accessed 11^{th} December 2018]

[9] Chodas P. Nasa (Centre for Near Earth Object Studies) : Glossary: Astronomical Unit (2018). [online] available at: https://cneos.jpl.nasa.gov/glossary/au.html [Accessed 12^{th} December 2018]

[10] Williams DR. Nasa:Planetary Fact Sheet - Metric. [online] available at: https://nssdc.gsfc.nasa.gov/planetary/factsheet/ [Accessed 12^{th} December 2018]

[11] Nave R. HyperPhysics Concepts: Conservation Laws. [online] available at: http://hyperphysics.phy-astr.gsu.edu/hbase/conser.html [Accessed 13^{th} December 2018]

[12] Tseplin V. PHYS132 - Basic Physics Skills: Lecture 3: Errors in physical sciences . [online] available at: file:///C:/Users/heathf/Downloads/PHYS132%20Lecture%2003%20-%20Errors%20-%20ver03.pdf [Accessed 13^{th} December 2018]

[13] Nave R. HyperPhysics Concepts: Kepler's Laws. [online] available at: http://hyperphysics.phy-astr.gsu.edu/hbase/kepler.html [Accessed 12^{th} December 2018]

[14] Tseplin V. PHYS132 - Basic Physics Skills: Lecture 4: Random Errors . [online] available at: file:///C:/Users/heathf/Downloads/PHYS132%20Lecture%2004%20-%20Random%20Errors%20-%20ver01%20(1).pdf [Accessed 13^{th} December 2018]

Appendix A: Derivation of Orbital Period from Average Orbital Distance

We will obtain the orbital period of a given body using Kepler's third law[13]:

$$T^2 = a^3 \tag{A1}$$

Where T is the orbital period in years and a is the average orbital distance of the body from the Sun, or more properly its semi-major axis, given in astronomical units. Hence since the Earth has an orbital radius of $1.00au$, its orbital period is:

$$T^2 = 1.00^3$$

$$T = +\sqrt{1.00^3}$$

$$T = 1 \ Year$$

This is of course the orbital period we expect for the Earth. Note, the period is given by the positive square-root as a duration of time cannot be negative.

Appendix B: Derivation of Standard Error in Orbital Periods of Mercury and Venus

We will obtain the standard error of a given body using the following equation[14]:

$$\alpha = \frac{\sigma_c}{\sqrt{N}} \tag{B1}$$

Where α is the standard error in the mean value, σ_c is the standard deviation (as defined in Section III) and N is the number of iterations which the simulation goes through.

For the simulation used to produce TABLE II, $N = 1.58 \times 10^4$ based on dividing the number of seconds in 50 years by the time-step of 1000 seconds and then by a factor of 100 as the program only saves the data every 100 time-steps. A value should be given as the mean with its standard error. Taking the values from TABLE II and using equation B1, we find:

$$\begin{aligned} \langle T_{Mercury} \rangle &= 0.2485 \pm 0.0001 \quad Earth - Years \\ \langle T_{Venus} \rangle &= 0.589112 \pm 0.000007 \quad Earth - Years \end{aligned} \tag{B2}$$