Lancaster University

An N-Body Gravitational System Simulation

Enrico Specogna
(Dated: 8/12/18)

The final project for the 281 Scientific Programming course described in this report is a simulation made with Python of a physical situation: a gravitational system where several bodies, affected by each other's gravitational attraction, move in orbits that cannot be predicted in analytic terms. That of course requires a program that uses a numerical approximation to find the properties of a body like its position or velocity and is able to do so iteratively. In nature an example of such a system can be obviously found in the Solar System: the program created is indeed an attempt to reproduce in a simple yet effective way the Solar System itself. However the picture that could be simulated of it has, as one would expect, important limits: these, along with other aspects of the simulation actually closer to reality, will be analyzed in the next paragraphs. The theory and the Physics involved in the simulation will be underlined, along with its usefulness in finding solutions to non-analytically solvable problems. The results of the simulation are then presented and subsequently analyzed too see what in the simulation is accurate and what is not, which will then allow the formulation of a conclusion that sums up all the important aspects of the project. Even if this program is very basic and to a certain extent rough, it will shown how it can still provide an interesting insight into one of the most famous and fascinating problems in the history of Physics: the N-body problem.

I. INTRODUCTION

In this Introduction (I) the main theory that has been used to create the simulation is outlined. Further on on this report it is possible to find the section Design (II) where the program structure is described along with the way the simulation itself flows. The results obtained are then analyzed in detail in the Result section (III). This section will then get to the last one, Conclusion (IV), which includes a final, overall comment on the project and its results.

A. The N-Body Problem [1] [2]

The motion of three bodies moving in space has been an historical problem in Physics since Newton first attempted to derive an analytic expression to predict the orbits of several massive bodies in the same system, through the gravitational law that he himself found. He soon found out that this could not be possible: the problem is that the planets' orbits are affected by the gravitational forces acting among them and these forces change as the position (in the orbit) of each planet changes, eventually leading, as we know today, to a chaotic behaviour. Chaotic does not mean that the orbits cannot be predicted or that they do not create a sensible physical pattern, but that it is not possible to find an analytic expression to describe their motion; in other words we cannot obtain anything like the well-known Suvat equations (1) used to find the trajectory of a particle in a uniform gravitational field, where acceleration can be considered as always constant. What we need is then an approximation of the Suvat equations which can be reiterated at multiple steps, to simulate the motion of the system of bodies: this is the topic of the following subsection (I B).

B. Theory [3] [4]

The main and most famous equation of motion are the Suvat equations (1) and (2) as defined below: they describe each point $\vec{x}$ of the trajectory followed by a particle that lies in a uniform gravitational field (like Earth if we consider the motion to take place near to its surface) and its velocity $\vec{v}$ at each step:

$$\vec{x} = \vec{x_0} + \vec{v_0}t + \frac{1}{2}\vec{a}t^2 \tag{1}$$

$$\vec{v} = \vec{v_0} + \vec{a}t \tag{2}$$

where $\vec{x_0}$ is the initial position of the particle, $\vec{v_0}$ its initial velocity, t the time and $\vec{a}$ the acceleration on the particle. The problem with equations (1) and (2) is that they consider $\vec{a}$ to be constant at any time t and neglects the effects that the particle (the motion of which we are considering) has on the other body: this is only an ideal approximation. In reality the force acting between the two bodies affects both and it is not constant. In fact, as the bodies' position changes so does the force acting between them and subsequently the acceleration $\vec{a}$ of both, simply because $\vec{a}$ of one body depends on the mass M of the other and the distance $\vec{r}$ between the two of them:

$$\vec{a} = \frac{GM}{|\vec{r}|^3}\vec{r} \tag{3}$$

where the gravitational constant G=6.67408 x 10^{-11}. The change of $\vec{a}$ will then act on their new position in their motion and so on, making it impossible to derive a general expression for the bodies' orbits; this is true not only for two-body systems but also for three-body and N-body systems, so in general the acceleration $\vec{a_i}$ acting on one of the bodies of the system that is taken into consideration will be the sum of all the accelerations created by each of the other objects [3]:

$$\vec{a_i} = \sum_{i,j=1}^{N} \frac{GM_i}{|\vec{r_i} - \vec{r_j}|^3}(\vec{r_i} - \vec{r_j}) \tag{4}$$

where M_i is the mass of the bodies one by one and $\vec{r_i} - \vec{r_j}$ the distance between the body considered and the others. Given that a general solution to this problem is not possible, to find the behaviour of the motion of an N-body system of planets we need to use numerical methods: the Python program which this report is all about is effectively an

implementation of a numerical approximation technique. A numerical method is simply an approximate calculation of the property of an object obtained through the repetition of a precise algorithm: in this case it is the reiterated calculation of the position, velocity and acceleration of the planets in the solar system for a defined number of steps. The properties, different at each step of the simulation, are calculated by the code at a set number 'N' of times. In the program here described two slightly different approximations (in the shape at least) are used:

One is called "Euler method" and the equations (5) and (6) below represent this approximation technique:

$$\vec{r}_{n+1} \approx \vec{r_n} + \vec{v_n}\Delta t \tag{5}$$

$$\vec{v}_{n+1} \approx \vec{v_n} + \vec{a_n}\Delta t \tag{6}$$

The other approximation is called the "Euler-Cromer" method:

$$\vec{v}_{n+1} \approx \vec{v_n} + \vec{a_n}\Delta t \tag{7}$$

$$\vec{r}_{n+1} \approx \vec{r_n} + \vec{v}_{n+1}\Delta t \tag{8}$$

To get the acceleration $\vec{a_n}$ we can simply use equation (4). Through both methods, assuming we have the position $\vec{r_n}$, the velocity $\vec{v_n}$ and the acceleration $\vec{a_n}$ of one body in the system at a step n, the program can calculate the position $\vec{r}$ at a step $n+1$, which is separated from n by the time Δt. Both methods are assuming $\vec{a}$ to be constant for the whole Δt time step, so the smaller Δt is, the more accurate will the result be. Once Δt is set, the program the can reiterate the calculation shown in "Euler" and "Euler-Cromer" methods for as many steps 'N' as we want. The difference between the two is very subtle, in fact while Euler uses $\vec{v_n}$ at the step n to calculate $\vec{r_{n+1}}$, Euler-Cromer uses the velocity at the step $n+1$: $\vec{v_{n+1}}$. This small difference in the shape actually brings to an important consequence in the simulation: in fact their accuracy varies consistently if we consider the simulation over a long period of time. The Euler method tends to be in fact way more unstable than the Euler-Cromer; while the error grows bigger and bigger for the first, it tends to be small and then negligible for the second. In general, once we have the system's information, we will also be able to calculate other quantities of it, like momentum $\vec{p_n}$ and kinetic energy $\vec{K_n}$ of any body of mass $\vec{m}$ at each time step n:

$$\vec{p_n} = M\vec{v_n} \tag{9}$$

$$\vec{K_n} = \frac{1}{2}M\vec{v_n} \tag{10}$$

or the total kinetic K_T,the potential energy U and the total linear momentum $\vec{p}$ of the system:

$$\vec{p_T} = \sum_{i=1}^{N} p_i \tag{11}$$

$$K_T = \sum_{i=1}^{N} K_i \tag{12}$$

$$U = \sum_{i,j=1}^{N} \frac{GM_iM_j}{|\vec{r_i} - \vec{r_j}|}(\vec{r_i} - \vec{r_j}) \tag{13}$$

U and K_T are then very useful to analyze the simulation itself, along with the Virial Theorem, which states that:

$$E = 2K_T - U \tag{14}$$

where E is an energy quantity that should be conserved in the system. Applying these properties to our system means understanding how well our simulation is behaving (see Results and Analysis section III).

II. DESIGN OF THE PROGRAM

The overall structure of the program created for this simulation can be outlined as follows: there are four different files or different bits of code. The first one contains a class called Particle: its aim is to define the main properties of each planet (or "particle") given to our Solar system. The properties are position, velocity, acceleration, name, mass, linear momentum and kinetic energy of the body. The second file is the core of the whole program: it contains the class called 'solarsystem'. This class encloses various methods, the main purpose of which is to calculate all the properties defined in the 'Particle' class for different time steps in the simulation. These methods make use of 'for loops' to take a body in the list of planets given to then apply to it the equations (7), (8) and (4) to calculate velocity, position and acceleration: it is important to specify that the order in which these two are written in the program makes the difference between using the 'Euler' method or the 'Euler-Cromer' method (see paragraph III B 1); in particular they are defined in this class instead of the 'Particle' class simply to stress the difference between these two classes, which is in fact to 'define' the single planet's properties for 'Particle' class and to 'calculate' these properties for 'solarsystem' class. Once we have obtained velocity and position, we can compute the other properties defined in the 'solarsystem' class, which are in this case not properties of the single planet but of the whole system; each of them is calculated in its respective method using the equations (12), (13) and (11) in the shape of coding. What we now need is a file that uses the classes described: the test file, which in this case is called 'solarsystemtest'. What this piece of code does is basically taking an input (each planet's position, velocity, ...) and passing this information (gathered in the variable called 'sol') to the methods defined in the two classes just described. A time step 'deltaT' (Δt) is set so that the properties of each planet can be recalculated for a precise number iterations 'N'. While the 'for loop' in the test file allows the initial input to be constantly **updated**, the line 'np.save...' **accumulates** all the data obtained at each time step and saves them into a file called 'solarsystemanalysis'. This file is then analyzed in the 'Analysis' file. The main feature of the this final bit of code is a loop that appends the data found in 'solarsystemanalysis' in defined arrays which are then plotted. The following is a more visual way to understand the flow of the whole process:

1. The initial conditions of the planets are provided to the test file in an numpy array as defined in the 'Particle' class.

2. These conditions are passed through all the methods in 'solarsystem' class, which recalculates them for the first time step.

3. The loop in the test file recalculates for the current for all the 'N' steps defined and for a Δt time step.

4. The initial conditions are updated N times to the initial conditions. N recalculations of the same properties are appended to the Data array and are then saved in a file called 'solarsystemanalysis'.

Now that all the information is saved, it can be uploaded analyzed in the Analysis file:

1. The data saved on the 'solarsystemanalysis' file is uploaded.

2. A loop takes specific data in each line of the saved information (there are of course N lines.)

3. The specific piece of data taken is appended to an empty array defined before the loop.

4. The arrays can be plotted in the form of a graph. e.g. x-position over y-position or total momentum over time.

The data initially obtained (and saved) in numerical shape with the test file, are at this point of the simulation available in a graphic form. Both kinds will be considered in the next section: 'Results and analysis'.

III. RESULTS AND ANALYSIS,

The results obtained can be easily summarized plotting the orbits of the system that was simulated for the inner bodies of the Solar System (the outer ones like Jupiter or Saturn can be found in Appendix A):

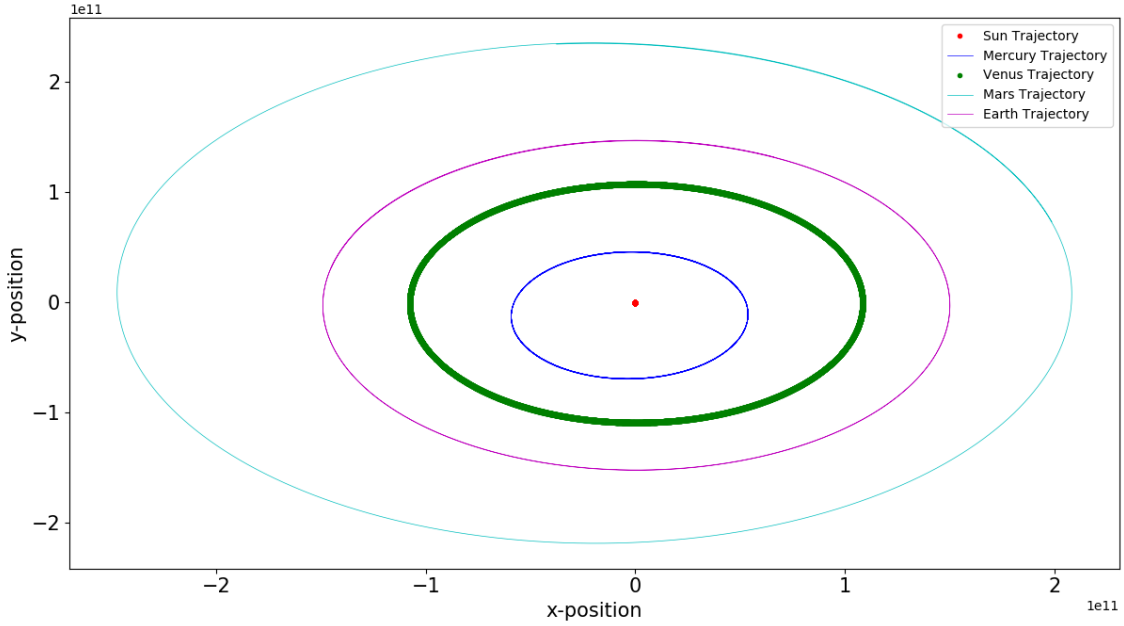


FIG. 1. Orbit of the inner bodies of the Solar System obtained with a time step of 3600s (1 hour) and through 10000 steps.

This is not however sufficient: we need to present the data that the simulation obtained and the presentation of the latter cannot be made separately from an adequate analysis. It is important to stress that, in describing how the program is behaving from its most basic predictions to a simulation at a large scale, we cannot compare the results we get from our simulation to the predictions of the ones obtained through professional or academic programs, as they are way too advanced compared to the one here presented. Then the only option left for a direct comparison is checking the difference between the program's predictions and the same data calculated analytically: this is what is done in the very next paragraph and it is a procedure that can only be carried out considering very simple systems of body, like a two-body system. To see how everything works for more bodies, we can only use indirect methods of analysis, which imply using plotting extensively to examine properties of the system rather than properties of the single body; this is the matter treated in the last part N-Body Systems III C of this section.

A. Correctness of the Simulation's Functions

The first thing that needs to be checked is whether the program is working properly or not, which means to see if the equations described in the Introduction (I) were exactly put into code and if the program itself uses them correctly. Once this is verified, it is possible to create actual data with the simulation and to examine it. A simple two-body system is taken into consideration: planet Earth is set at the centre of the simulation's coordinates system, with a Satellite of mass m=100kg rotating around it. It is sufficient to run the simulation for 2 iterations 'N' and a $\Delta T = 600$s seconds. The data obtained are then compared to the analytic solutions of equations (1), (2), (3), (9) and (10). In the steps considered, both for the analytic and the approximated equations, the acceleration is maintained constant to allow the calculation.

	x-Position (m)	y-Velocity (m/s)	x-Acceleration (m/s^2)	y-Linear Momentum (Kgm/s)	Kinetic Energy (J)
Initial	1.000x10^8	2.000x10^3	-4.000x10^{-2}	2.000x10^3	2.000x10^5
Analytic; Final	0.988x10^8	1.976x10^3	-3.983x10^{-2}	1.976x10^3	1.952x10^5
Simulated; Final	1.000x10^8	1.999x10^3	-3.982x10^{-2}	1.996x10^3	1.999x10^5

TABLE I. Table comparing some properties of the Satellite obtained both through the analytic equations and the simulation.

As we see from Table I, the main properties predicted by the simulation are compatible with the same ones obtained analytically, meaning the program itself is written correctly. The properties defined in the other methods of the 'solarsystem' class are also correctly written, as they depend on the ones shown in Table I.

B. Two Body Systems

As it has already been mentioned, simple situations like two-body systems allows us to make a direct comparison between Data from the simulation and Data obtained by analytic calculation, as shown in the next two paragraphs.

1. *Comparison Between Euler and Euler-Cromer Numerical Approximation,*

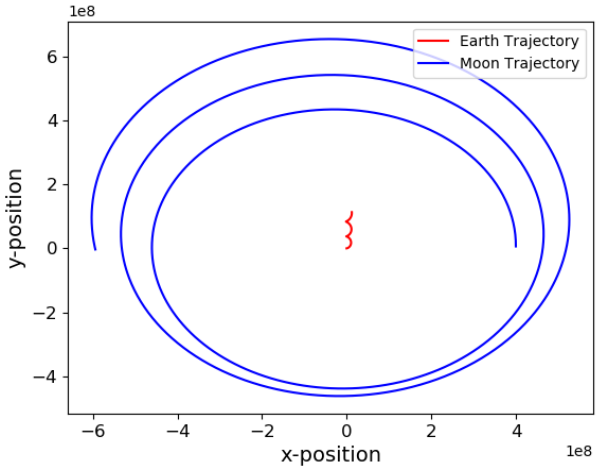
Knowing that the data can be calculated correctly (which means coherently to how they were defined), it is possible to start comparing the two numerical approximations used in this simulation: Euler and Euler-Cromer. We can for example obtain both the acceleration and the velocity of the Satellite on both the x and y directions and compared them to the ones calculated by hand. In this case the Satellite is the Moon: it has a mass of 7.349x10^{22} kg and an initial position $\vec{d} = (4.000\text{x}10^8\text{m}, 0.000\text{m}, 0.000\text{m})$:

	x-Velocity (m/s)	x-Acceleration (m/s^2)	y-Velocity (m/s)	y-Acceleration (m/s^2)
Initial	0.000x10^{-1}	-2.489x10^{-3}	1.023x10^{-2}	0.000x10^3
Analytic; Final	-1.494x10^{-1}	-2.489x10^{-3}	1.023x10^3	-7.644x10^{-7}
Euler-Cromer; Final	-1.494x10^{-1}	-2.491x10^{-3}	1.023x10^3	-7.644x10^{-7}
Euler; Final	-1.495x10^{-1}	-2.491x10^{-3}	1.022x10^3	-7.646x10^{-7}

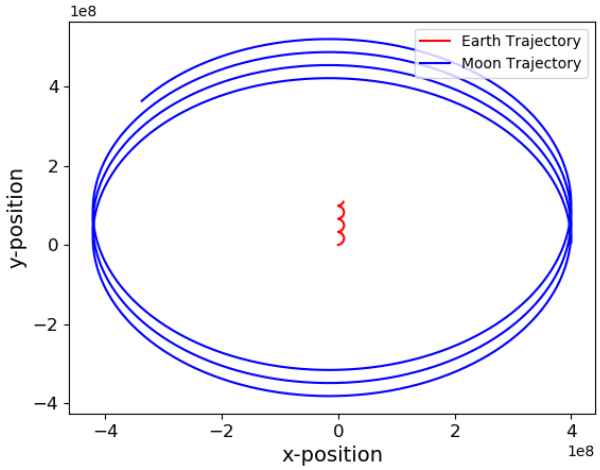
TABLE II. Table comparing Euler and Euler-Cromer results when calculating the Final Velocity and the Final Acceleration in x and y direction.

As we see, for 2 'N' steps with $\Delta t = 60$s, the Moon varies its position but the way this is calculated by the two methods is different: Table II proves that Euler-Cromer is more accurate as a method for the approximation of the orbits of the planets in our system. In general, the Euler method tends to give results the get more inaccurate as we increase the time step, making it 'unstable'.

This instability is more evident when we consider a larger number of iterations, as shown in the following pictures:



(a) Euler Method's Orbit.



(b) Euler-Cromer Method's Orbit.

FIG. 2. A comparison of the orbits created with the two approximation methods using 500 iterations 'N' over time period Δt = 6000s. Earth's orbit is not fixed at a single point because the mass of the Moon compared to Earth's influences its orbit visibly.

As shown in the Figure 2 above, while the Euler method's orbit gets larger with time, the Euler-Cromer method is more 'stable' in the long term, as the orbit of the Moon around Earth oscillates between the same two x-positions $4.000\times\ 10^8$ m. Another visual way to demonstrate how Euler's methods is more inaccurate and unstable than Euler-Cromer's can be found in subsection III C, where the two methods are examined by the point of view of Momentum and Energy conservation.

2. *Simulation at Different Time Steps* Δt

The two-body system used in the two previous paragraphs can be analyzed running the simulation at different ΔT steps; a longer step in physical terms simply means that the acceleration is considered to be constant for a bigger time interval; the effect on the calculation of the final x-Position through the Euler-Cromer method and 2 'N' reiterations is shown in the following table.

	Initial Position (m)	Final Position Calculated (m)	Final Position Predicted (m)
$\Delta T = 60s$	4.000x10^8	4.000x10^8	3.999x10^8
$\Delta T = 600s$	4.000x10^8	4.000x10^8	3.999x10^8
$\Delta T = 6000s$	4.000x10^8	3.999x10^8	3.990x10^8
$\Delta T = 60000s$	4.000x10^8	3.955x10^8	3.913x10^8
$\Delta T = 600000s$	4.000x10^8	-4.815	-9.144x10^{-5}

TABLE III. Table comparing the final x-Position at different time intervals Δt.

As Table III above demonstrates, while Δt increases, the final position as predicted by the program becomes more and more different from the one calculated analytically, resulting in two completely different results for $\Delta t = 600000s$. Considering the acceleration to be constant is a very rough estimate for the properties of the Satellite, when calculated, and for its orbit when we plot it. To explain this we can refer to equations (7) and (8): it is clear that the velocity at the end of each step will get bigger when Δt is increased, also because the acceleration will act on the Satellite throughout the whole step, which is longer. As a consequence, the final position will have bigger magnitude; in other words it is going to be very far from the initial . The difference between the two positions limiting the step will then be evident in the plot of the orbit in the form of many segments joined together that represent each step of the

simulation. As the program runs on through other steps the Satellite will get each time further from the Earth, as the following graph shows:

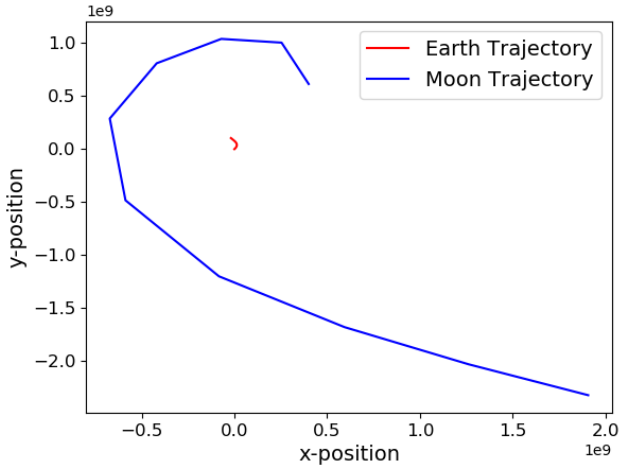


FIG. 3. The effect on the Moon-Earth system when we consider a very long time step (Δt = 600000 s) through 10 'N' repetitions.

A big time step can then potentially 'tear apart' the system of bodies. It is important to specify that the Δt just used would still be acceptable if applied to an N-body system (see Appendix A, where the orbit is obtained with a big time step.).

C. N-Body Systems

In this section is presented the final and most complete form of the project described so far in this report: the simulation of the Solar System. The same program used so far in the Report and Analysis section for two-body system can be extended to an N-body system, more precisely to a system with the main planets orbiting around the Sun: Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus and Neptune. As already mentioned in the first paragraph of this section of the report, it is not possible to calculate analytically data relevant to the bodies of the system and compare it with the simulation; we can still check the validity of the simulation for the N-body case making use of an indirect check, which can be found in the conservation laws of all the physical systems in nature. There are two conservation laws that will be used to examine the behaviour of the simulation for the bodies just quoted: the conservation of the total linear momentum of the system and the so called 'Virial Theorem', which is an energy conservation law. Both of these laws are defined in the 'solarsystem' class of the code; the important difference from the properties used in the previous part of the analysis is that these conservation are defined for the system as a whole, not for each singular planet in it.

1. The Total Momentum Conservation

An interesting way to test how well the simulation recreates the Solar System is to see how the total linear momentum of the system varies. Equation (11) is the definition of this property of the overall system: the useful aspect of this formula is that it describes a quantity that should not vary with time, it should be constant. We can then see how big is the variation of the Total Momentum in our system. To do this, we can see graphically how the Momentum Variation over time (see Appendix B) and then make a plot which analyze the 'Fractional change' of the Total Momentum at each ΔT step. The latter quantity is defined as follows:

$$\Delta p_T = \frac{p_0 - p_T}{p_0} \tag{15}$$

where Δp_T is the fractional change of the total momentum p_T at one time step and p_0 the total momentum at the first Δt step.

Once we apply equation (15) to each step of the simulation, we will get the following plot:

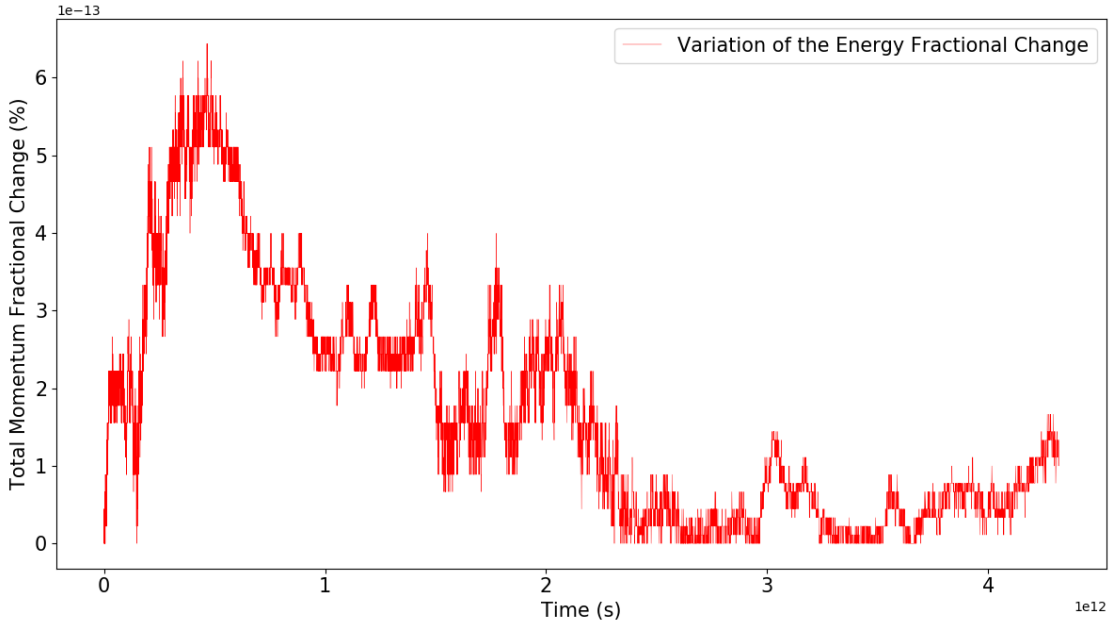


FIG. 4. The fractional change of the total momentum of the system, for a time step $\Delta t = 86400$ s (one day) through 10000 'N' repetitions.

Figure 4 shows how small is the variation of the total momentum at each point of the simulation analyzed from the initial p_0. The biggest of the oscillations shown in the graph has measure:

$$\Delta p_T = 6.439 \times 10^{-13}\%$$

This value is very small,indicating that the simulation can replicate the conservation of momentum really well; if the total momentum is conserved than we can also say that properties like the velocity of the planets is approximated with fairly good accuracy. After the peak we see between 0s and 10^{12}s, the oscillation of the total momentum decreases, which probably indicates that the system is becoming more stable as the simulation runs on; in other words, the planets start from random initial positions (generated by HORIZONS Web-Interface) and as time passes and they start acting on each other through a mutual gravitational attraction, the system acquires its 'natural' shape and motion. Having said this, we can then also assume that the acceleration has been well defined in the program, otherwise we would not see the simulation becoming more and more stable after a properly long time (depending on the choice of both the time step and the number of iterations); the accelerations in fact eventually lead to a stable situation. This effect is only achieved using the Euler-Cromer approximation though. If the Euler approximation were used, the behaviour of the simulation would be the opposite of what we see in Figure 4: the system would in fact tend to instability as the peak of the momentum fractional change is well beyond the first part (time) of the simulation, as one can see in appendix B.

2. Total Energy Conservation: the Virial Theorem

Another way to assess how well the simulation is making predictions is to analyze the conservation of energy: for this purpose the Virial Theorem can be used. This theorem simply states that in an N-body system, the quantity defined by Equation (14) is always conserved. Similarly to what was done in paragraph III C 1, we can use the energy conservation to find information about the accuracy of the program; applying the theorem to the simulation we are going to get again an oscillation, with a peak at the beginning that becomes more stable as the simulation goes on. We can also confirm again how Euler-Cromer method is the most robust of the two used to simulate the system of planets. Once have the Variation of the total energy of the system (Appendix C), we can get its fractional change.

Applying the Virial Theorem to the system, we get the following result: In this case we still get a Fractional Change,

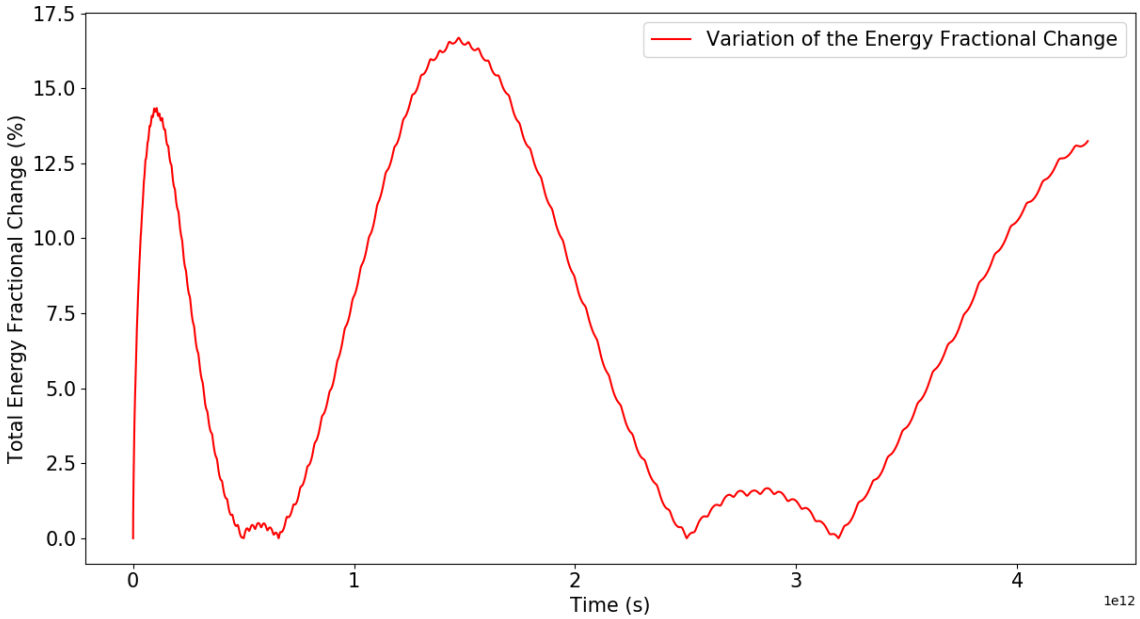


FIG. 5. The fractional change of the Total Energy of the system, for a time step $\Delta T = 86400$ s (one day) through 10000 'N' repetitions.

defined as ΔE_T for the energy rather than Δp_T as used for the Total Momentum. The peak for the Virial Theorem is found between 0s and $2{\times}10^{12}$s as:

$$\Delta E_T = 0.1668\%$$

This values is certainly higher than the Δp_T found in the last paragraph, but there is a reason that can explain that: the total Kinetic Energy differently from the total momentum has a factor of v^2 rather than just v, so that the error in the calculation of the velocity is squared, influencing negatively on the evaluation of the Total Kinetic energy variation through time and therefore giving the peak just mentioned. In general, the behaviour is still similar to the momentum's fractional change: we first an initial, narrow peak and we have the tallest peak, which got wider than the previous. After this, more peaks follow but they get wider and shorter, meaning that the system has a found a balance in its motion, oscillating between the same values of total energy within the same amount of time; this cannot be shown here as we would need a more powerful calculator.

D. The Flaws of This Program

If we do not consider that the program described here lacks important considerations like relativistic corrections or the implementation of bodies that have a significant influence in the motion of the Solar system like the Oort Cloud, we can identify the main problem of this program in the approximations used: the Euler and the Euler-Cromer method. Both in fact are Taylor expansions of the first order of the usual equations of motion like equation (1) and (2): this means that all of the terms in the infinite sum of order N ≥ 2 are ignored; then, every time we iterate the calculation for the properties of a planet we will have an error of Δt^2. Choosing a high Δt time step will make the simulation more and more rough while it runs, as it was demonstrated in Section III B 2 (this true in the two-body case. For N-bodies, as Appendix A demonstrates, we can still use a big time step, if chosen properly). An analogy to this situation to understand how a big time step affects the quality of our result is thinking of how the way in which we define the variation of any continuous mathematical function depends on how big Δx is (derivative in the two-dimensional case). In the program's simulation we obviously cannot have Δt tending to 0, as this would mean that we can actually find an analytic solution that describes a physical system that is in reality chaotic. Still this does not mean that the algorithms used are not *valid* as an approximation of the Solar System.

IV. CONCLUSION

The Python program here described principally was aimed at simulating an approximation of the Solar System and this goal was achieved. The program created is in fact capable of simulating the orbits of the main bodies in the solar system as (Figure 8) shows. Its power however does not rely on the fact that many different plots of the planets' orbits can be obtained (which are the graphical part of the results) but rather that we have a tool that can tell us different properties like position or velocity of a body in a system. What makes this tool unique is the mere fact that it allows us to carry out a calculation that would not be possible to make in analytic terms, as already mentioned. This simulation in fact uses the algorithms described in Equations (5),(6),(8) and (7), which are repeated throughout a number of 'N' iterations, each with a time step Δt. It was in particular found out that the Euler-Cromer approximation gave back better results and in general a better simulation. The graphs produced in particular showed that not only the latter method is better but also helped to make a prediction: the system simulated will reach a point of 'natural' balance of motion, so that it total energy and momentum will have a constant oscillation, a little bit like a resonant frequency dictated by the multiple forces acting in the system: after all the Solar system can be seen as a group of bodies that oscillates with a (almost) definite period around the Sun. The results obtained could still surely be improved though: a future project could in fact involve the implementation of better and more complex techniques, starting from the simpler ones like the Euler-Richardson approximation (which differently from Euler-Cromer's method makes use of the velocity in the middle of the time step v_{mid} instead of the velocity at the end of the step v_{n+1}) to reach methods as the Runge-Kutta algorithms or Verlet's approximation.

[1] https://en.wikipedia.org/wiki/N-bodyproblem
[2] https://www.britannica.com/science/n-body-problem
[3] I. Bertram, R. Long, PHYS281 course notes, Michaelmas 2018
https://modules.lancaster.ac.uk/pluginfile.php/1781020/modresource/content/5/CourseNotes20181102.pdf
[4] Python and Matplotlib Essentials for Scientists and Engineers, Matt A Wood, Morgan Claypool Publishers, 2015
http://iopscience.iop.org/book/978-1-6270-5620-5

Appendix A: Orbit for the Outer Planets

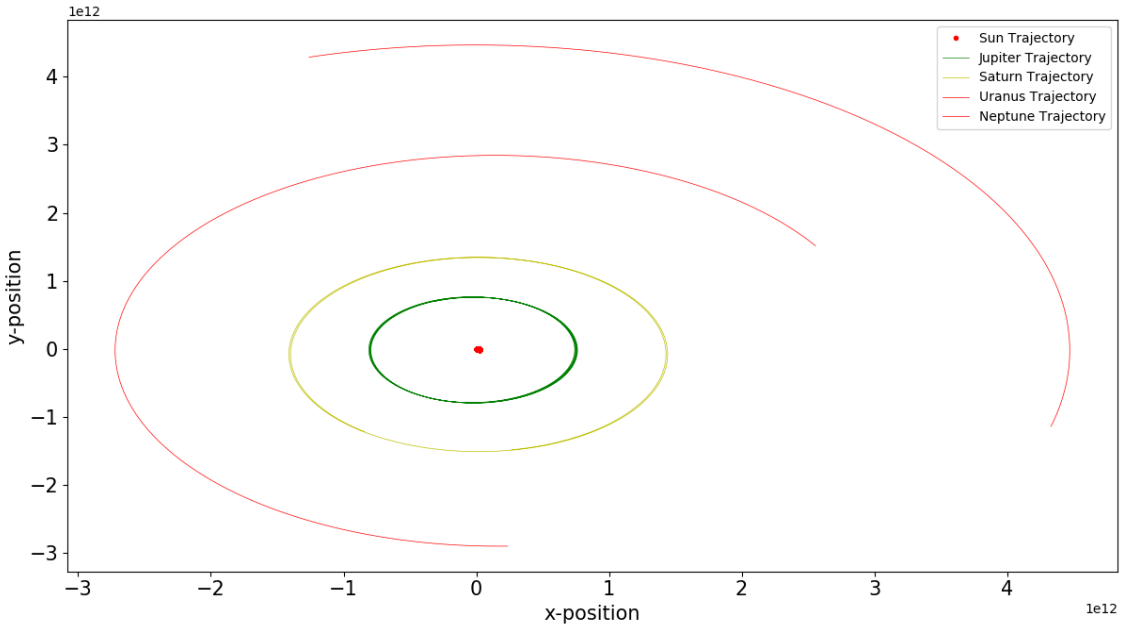


FIG. 6. Orbit of the outer bodies of the Solar System obtained with a time step of 86400s (1 day) and through 20000 steps.

Appendix B: Total Momentum Variation

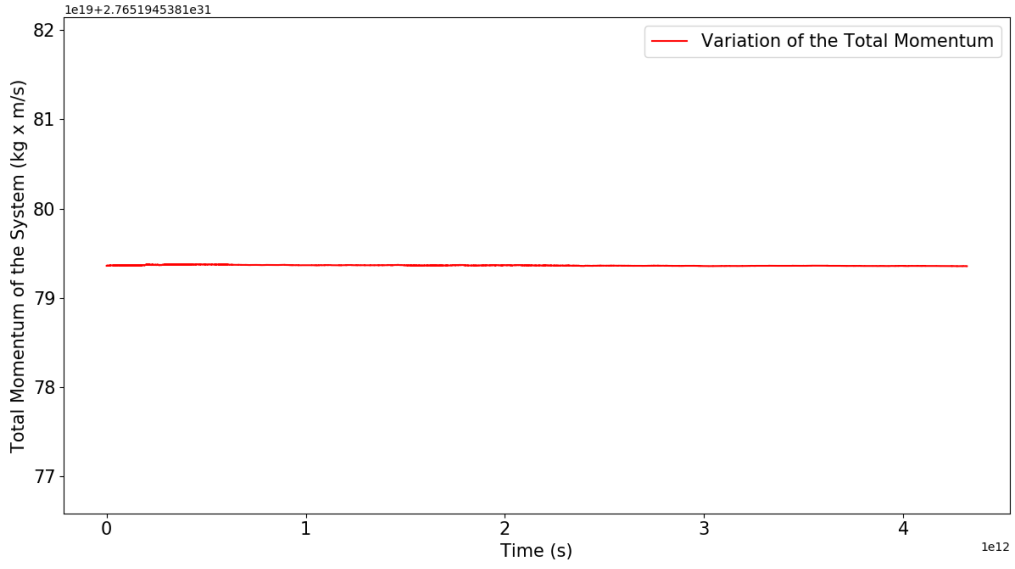


FIG. 7. Variation of the total momentum obtained with a time step of 86400s (1 day) and through 10000 steps.

Appendix C: Total Energy Variation

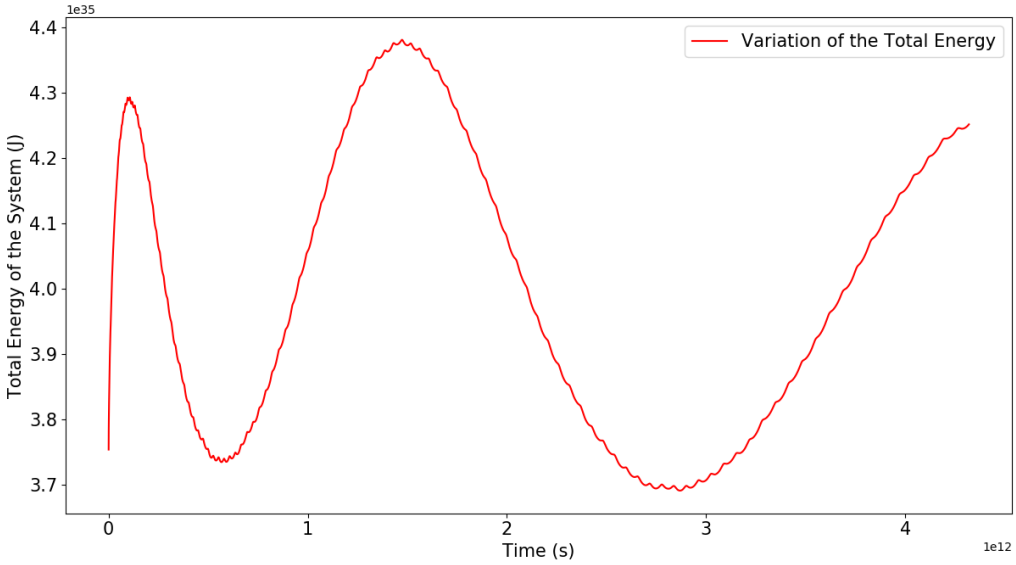


FIG. 8. Variation of the total energy obtained with a time step of 86400s (1 day) and through 10000 steps.