Simulation techniques for equations of motion and application to the Solar System

Vlad Carare

March 9, 2018

Abstract

The report presents the theory, design and results of a program intended to simulate the Solar System. It shows the accuracy and precision of four different algorithms used to implement the equations of motion. These algorithms are: Euler, Euler-Cramer, Verlat and Euler-Richardson, in order of increasing complexity. For a 2-body problem, with a time step of 10^{-4} seconds, we get the following percentage errors -2.47%, $7.33\times(10^{-7})$%, 0.63% and $-7.76\times(10^{-10})$%, in the order stated above. All the algorithms correctly describe Earth's elliptical orbit around the Sun, as shown in graph 3.

1 Introduction

The aim of this report is to give the reader an insight into different techniques for implementing the equations of motions to describe the evolution of a system like a collection of charges or, in our case, the Solar System. It seeks to quantitatively illustrate how well each technique works. The algorithms tested and presented here are Euler, Euler-Cramer, Verlat and Euler Richardson, in the order of increasing complexity. Furthermore, the investigation makes use of conserved quantities, such as momentum, angular momentum and position of centre of mass, to quantify the accuracy of each algorithm. More about this in section 2.

In order to calculate the accuracy of the results and to make sure the simulation is physical, one had to get both real position, velocities and mass of the astronomical bodies to start with. If this was not the case, the system would have behaved in way which will make it not suitable for a good comparison with the Solar System. One reliable source of getting this kind of information presented in a convenient way, (Cartesian Coordinates) is the website Jet Propulsion Laboratory(JPL) HORIZONS Web-Interface [5]. One more important aspect with regard to a system of gravitationally orbiting bodies is that they tend to orbit around the common centre of mass. Therefore, the position and velocity vectors of the planets have to be taken wih respect to the common centre of mass of the Solar System. After introducing the initial position vectors, velocity vectors and masses, one can run the simulation for any amount of time and check with data on the JPL website for the corresponding calendar date. In our case, the starting data is taken for the date of 5th of December 1996. There are also other ways of testing the accuracy of our results, such as computing the conserved

quantities of the system before and after the simulation has taken place. Such conserved quanities include: total momentum, total angular momentum, total energy and the position of the centre of mass.

One could suggest that we could also simply use pen and paper to calculate the behaviour of the system and compare our data to that. But keep in mind that there is no analytic solution to the 3 body problem and even more so for the n-body problem. The only way to get close to the answer is to design a computer program which implements the equations of motion and Newton's law of universal gravity. Of course there are many errors in this type of approximation, but nevertheless it offers a good sense of direction to our problem.

Further in this report, section 2 offers the reader the necessary knowledge about the equations of motion and the law of gravity. It also uses these equations to justify choices and assumptions made.

Section 3 provides an overview into the functioning of the program. It contains a clear flowchart of the program, a class relationship diagram and 5 class diagrams.

Section 4 investigates and compares the accuracy of all the methods at different sizes of the time step.

Section 5 highlights the main findings and problems with the techniques.

2 Theory

Firstly, the most fundamental concepts that the reader needs to grasp before delving further into this paper, are Newton's law of universal gravitation (1), momentum (2) and angular momentum (3) of a particle and the position the centre of mass (4). Newton's law of universal gravitation states that the force between two point masses is directed along the line connecting the two points and is equal to:

$$F = G\frac{m_1 m_2}{r^2} \tag{1}$$

, where F is the force felt by each object, G is the universal gravitational constant, m_1 and m_2 are the two masses in question and r is the distance between them.

$$\vec{p} = m\vec{v} \tag{2}$$

, where $\vec{p}$ is the momentum of a particle, m is its mass and $\vec{v}$ is its velocity.

$$\vec{L} = \vec{r} \times \vec{p} \tag{3}$$

, where $\vec{L}$ is the angular momentum of a particle, $\vec{r}$ is its position vector with respect to the origin on the rotation axis and $\vec{p}$ is the particle's momentum.

$$\vec{R} = \sum_{i=1}^{N} \frac{m_i \vec{r_i}}{M} \tag{4}$$

, where $\vec{R}$ is the position of the centre of mass of a system, m_i is the mass of the i-th particle, $\vec{r_i}$ is the position vector of the particle and M is the total mass of all the particles in the system.

Secondly, in order to attempt to simulate the Solar System, there is the need of some physics methods which can be converted into computer algorithms. Some of these algorithm are directly derived from the equations of motion, such as the first two: Euler (equations (5)-(6)) and Euler-Cramer (equations (7)-(8)). The next two are still a consequence of the equations of motion but are a bit more complex: Verlat (equations (9)-(10)) and Euler-Richardson (equations (11)-(16)).

$$\vec{v}_{i+1} \approx \vec{v}_i + \vec{a}_i \Delta t \tag{5}$$
$$\vec{x}_{i+1} \approx \vec{x}_i + \vec{v}_i \Delta t \tag{6}$$

$$\vec{v}_{i+1} \approx \vec{v}_i + \vec{a}_i \Delta t \tag{7}$$
$$\vec{x}_{i+1} \approx \vec{x}_i + \vec{v}_{i+1} \Delta t \tag{8}$$

$$\vec{x}_{i+1} \approx \vec{x}_i + \vec{v}_i \Delta t + \frac{1}{2}\vec{a}_i (\Delta t)^2 \tag{9}$$
$$\vec{v}_{i+1} \approx \vec{v}_i + \frac{1}{2}(\vec{a}_{i+1} + \vec{a}_i) \Delta t \tag{10}$$

$$\vec{a}_i = \vec{F}(\vec{x}_i, \vec{v}_i, t_i)/m \tag{11}$$
$$\vec{v}_{mid} \approx \vec{v}_i + \vec{a}_i \Delta t \tag{12}$$
$$\vec{x}_{mid} \approx \vec{x}_i + \vec{v}_i \Delta t \tag{13}$$
$$\vec{a}_{mid} = \vec{F}(\vec{x}_{mid}, \vec{v}_{mid}, t + \frac{1}{2}\Delta t)/m \tag{14}$$
$$\vec{v}_{i+1} \approx \vec{v}_i + \vec{a}_{mid} \Delta t \tag{15}$$
$$\vec{x}_{i+1} \approx \vec{x}_i + \vec{v}_{mid} \Delta t \tag{16}$$

,where $\vec{v}_i$ is the velocity of a particle at a given time t, $\vec{v}_{i+1}$ is the velocity of a particle at a later time $t + \Delta t$, Δt is the **time step** chosen by the user, in seconds. $\vec{a}_i$ is the acceleration of a particle at a given time t, $\vec{x}_i$ and $\vec{x}_{i+1}$ are the positions of a particle at a time t and a time $t + \Delta t$ respectively. $\vec{F}(\vec{x}_i, \vec{v}_i, t_i)$ is the force on a particle at a given position, velocity and time, m is the mass of the particle. $\vec{F}(\vec{x}_{mid}, \vec{v}_{mid}, t + \frac{1}{2}\Delta t), \vec{v}_{mid}, \vec{x}_{mid}$ are the force on a particle, the position and the velocity of a particle at a given time $t + \frac{1}{2}\Delta t$.

Thirdly, in the endeavour to simulate the Solar System, there were good reasons to select the most massive bodies, as according to equation (1) [4], the bigger the mass, the stronger the force between the objects. In accordance to this equation, 5 bodies in the Solar System ought never miss: the Sun, which accounts to 99.86% of the mass of the system and the four gas giants: Jupiter, Saturn, Uranus and Neptune; which make up 99% of the rest of the mass [1]. Thus small asteroids and objects could be safely ignored. This can be demonstrated by calculating the force on Earth, firstly caused by a big planet and secondly, by one of its satellites. For example Uranus (mass 8.7 $\times 10^{25}$ kg) and its satellite, Titania (mass 3.5×10^{21} kg). The distance between them is of 4.36 $\times 10^8$ metres, while the distance from Uranus to Earth is 2.6 $\times 10^{12}$. So the distance from Titania to Earth can be approximated to be

the same as that from Uranus to Earth. Moreover, by using equation (17) twice, we find that the force exerted by Titania is of a order of 10^4 lower than that exerted by Uranus. Nevertheless, the program includes some of Jupiter's satellites, since their big mass, puts them on the same footing as some of the planets, for example Ganymede (mass 1.5×10^{23} kg) relative to Mercury (mass 3.3×10^{23} kg). The force they exert at equal distances away from them is of the same order of magnitude. Other objects included are satellites of Neptune (Triton, mass 2.1×10^{22} kg), Earth (Luna, mass 7.3×10^{22} kg), Saturn (Titan, mass 1.3×10^{23} kg) and a couple of dwarf planets: Eris (mass 1.5×10^{22} kg) and Pluto (mass 1.3×10^{22} kg). The selection stopped when objects got a mass lower than the order of 10^{22}.

Fourthly, an important test of this program is whether is holds up to a simple 2-body problem. 2 body problems, as opposed to bigger number of bodies, can easily be solved analitically, using Newton's Law of Motion in gravitational(17) and centripetal (18) form.

$$G = mg \tag{17}$$

$$F_c = m\omega^2 r \tag{18}$$

,where G is the gravitational force, F_c is the centripetal force, m is the mass of the object, g is the gravitational acceleratio, ω is the angular speed and r is the radius of the orbit.

To make things easier, we consider the case of a satellite orbiting a planet. The force that the satellite exerts on the planet is minuscule compared to what the planet exerts on the satellite and so the planet can be considered stationary while the satellite does a orbit around it. But how can we know what values for position and velocity to assign to the satellite in order to get a stable orbit? The derivation of the desired equation is shown in the appendix (20) and is quoted here:

$$v = \sqrt{\frac{GM}{r}} \tag{19}$$

Finally, plots and graphs are included in order to analyse the accuracy of each algorithm and to inspect the factors that contribute to the errors. One such plot may be done as followes: as we know, what we call a year is one revolution of Earth around the Sun, the Earth arriving at the same position as it started in. By computing its path for a year around the Sun, the program should be coming up with the initial position, just like in the previous example, but now with the Earth being the "satellite" of Sun and a year being its period of rotation.

Different time steps account for different magnitude of errors.

3 Design

The program uses object-oriented coding. It has been separated in 6 java files containing 7 classes and an input file containing the starting data for the: name, position, velocities and mass; of the 17 most massive bodies in the Solar System. The class relationship diagram (figure 1) illustrates this.

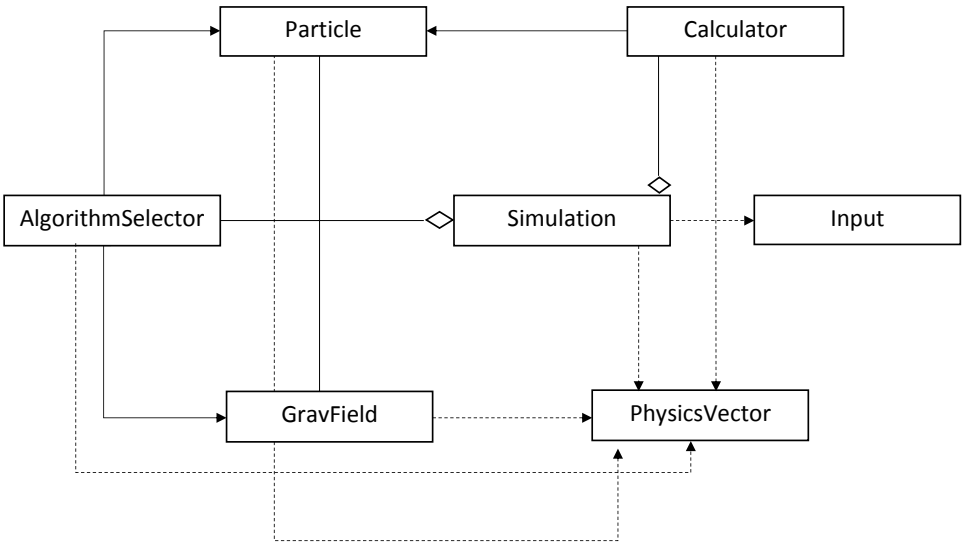

Figure 1: Diagram represents the relationships between all the classes of the program.

The information is taken in Cartesian Coordinates from JPL Horizon [5] and the first calendar date is the 5th of December 1996. Once compiled, it prompts the user to introduce paramateres such as: time step, interval of measurements and also give the user the option to choose whether he wants to receive intermediate results or not. If yes, prompts the user to introduce a value for the intermediate time interval, at which he wants intermediate results.

Class Input, represented in table 1, takes care of creating a scanner to run through the file. The main method uses the information from the file to generate instances of the Particle class (table 2) and GravField class (table 3), i.e. astronomical objects and gravitational fields. Each object is influenced by the gravitational field of all the other objects. The contributions sum up to the gravitational acceleration at a given point in space, where a given celestial body is at a given moment. This is what GravField class is made to represent. Evidently, from Newton's 3rd Law, every action has a reaction, the body we are measuring at is not going to feel any influence from itself.

Input
+read(solarSystem: Particle[], gravField: Particle.GravField[]): void

Table 1: Class diagram for 'Input' class.

Particle
-position: PhysicsVector -velocity: PhysicsVector -mass: double -name: String -G: final double
<<constructor>> Particle() <<constructor>> Particle(position: PhysicsVector, velocity: PhysicsVector, mass: double) <<constructor>> Particle(name: String, Rx: double, Ry: double, Rz: double, Vx: double, Vy: double, Vz: double, mass: double) +returnName(): String +returnPosition(): PhysicsVector +returnVelocity(): PhysicsVector +returnMass(): double +returnMomentum(): PhysicsVector +returnAngularMomentum(): PhysicsVector +returnEnergy(): double +applyEuler(timeStep: double, acceleration: double): void +applyEulerCramer(timeStep: double, acceleration: double): void +applyVerlat(timeStep: double, acceleration: double, objectNumber: int, solarSystem: Particle[], gravField: Particle.GravField[]): void +applyEulerRichardson(timeStep: double, acceleration: double, objectNumber: int, solarSystem: Particle[], gravField: Particle.GravField[]): void

Table 2: Class diagram for 'Particle' class.

GravField
-gravAcceleration: PhysicsVector
<<constructor>> gravField() +returnAcceleration(): PhysicsVector +resetAcceleration(): void +gravAcceleration(x: Particle): void

Table 3: Class diagram for 'GravField' class.

We consider each astronomical object as point-like object with no radius to simplify our program, since this has no bearing on how the gravitational field is distributed, as long as the planets are round [3], which, to a good approximation, they are. Another reason for this assumption is that, as you can see from the law of gravity(1), we are interested in the distance to the centre of mass of a body, not the distance to its surface. Thus we can safely assume the objects are points interacting in space.

The names of all the objects is included in the input file. The name helps the reader interpret the answers and also acts like a fingerprint for every object and may be used in some methods to differentiate between the bodies.

Further in the program, the main method calls the 'select' method inside the 'SelectAlgorithm' class, presented in table 4. This class contains all the information about the objects and bridges the communication between them. It keeps track of the number of steps taken and at each step it invokes the chosen algorithm from the 'Particle' class.

As a test for the accuracy of our program several quantities which are conserved by the laws of physics have been calculated, namely: the total momentum of the solar system, the total angular momentum of the Solar System. The 'Calculator' class, shown in table 5, keeps track of the total momentary: momentum and angular momentum and also calculates the momentary position of centre of mass.

AlgorithmSelector
+select(choice: int, timeStep: double, interval: double, solarSystem: Particle[], gravField: Particle.GravField[]): void

Table 4: Class diagram for 'AlgorithmSelector' class.

Calculator
+calculateMomentum(solarSystem: Particle[]): PhysicsVector
+calculateAngularMomentum(solarSystem: Particle[]): PhysicsVector
+calculateEnergy(solarSystem: Particle[]): double
+calculateCM(solarSystem: Particle[]): PhysicsVector

Table 5: Class diagram for 'Calculator' class.

The overviews of the process is depicted in figure 2.

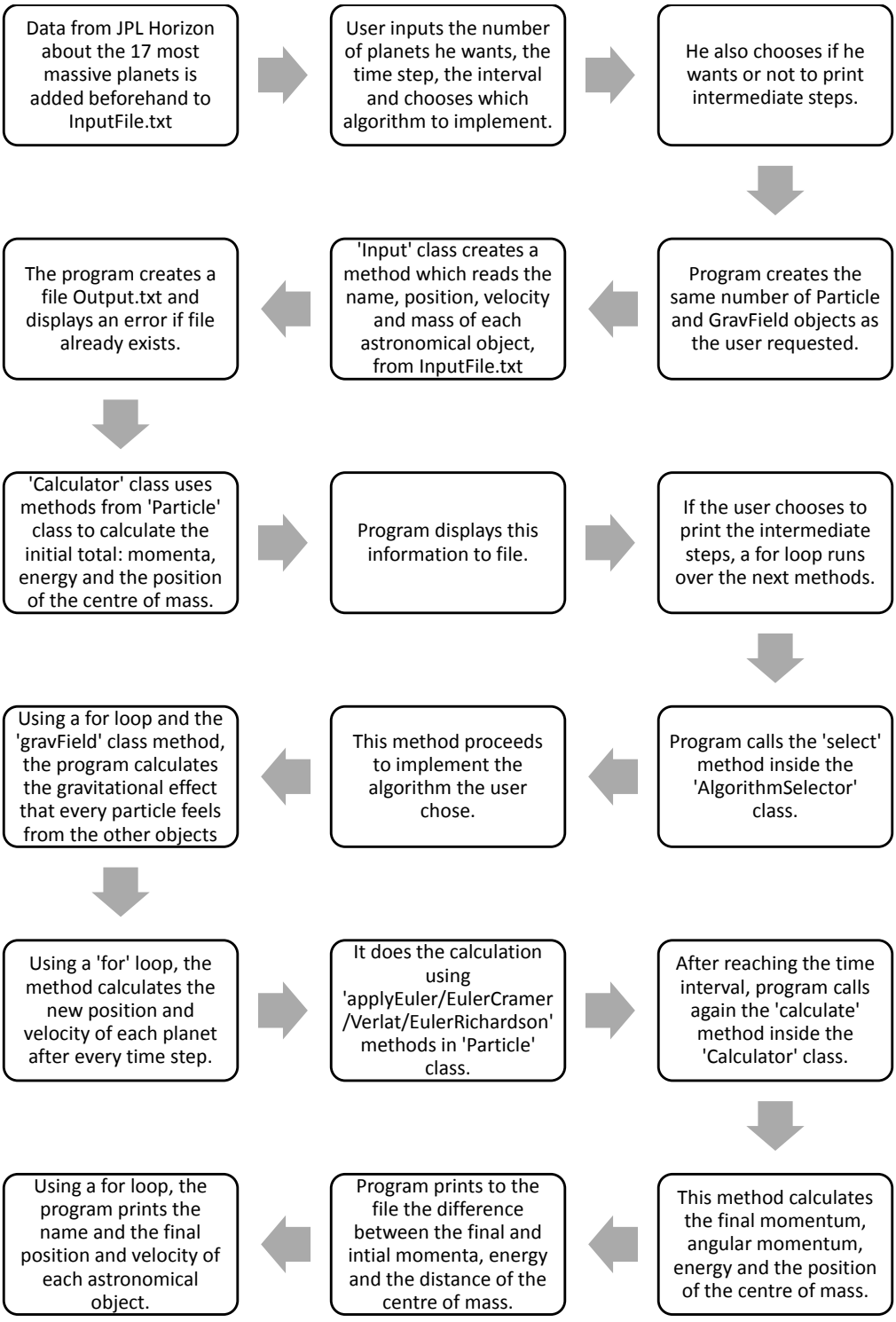

Figure 2: Overview of the process of simulating the Solar System

4 Results

The effects of choosing differently sized time steps for the four algorithms have been quantified by applying the 2-body satellite problem discussed in section 2. In this case, for simplicity, the gravitational constant G has been set to a value of 1, the position of the planet has been set to (0,0,0)metres, the position of the satellite to (0,10,0)metres. Using equation (19), for a stable orbit, we find we need a velocity of (10,0,0)metres/second.

Algorithm / Time step (seconds)	Euler	Euler-Cramer	Verlat	Euler-Richardson
10^{-4}	-2.47%	$7.33\times(10^{-7})$%	0.63%	$-7.76\times(10^{-10})$%
10^{-3}	$-2.03\times(10^{1})$%	$4.06\times(10^{-4})$%	6.74%	$-7.87\times(10^{-7})$%
10^{-2}	$-9.80\times(10)$%	0.03%	$-8.52\times(10^{4})$%	$-7.94\times(10^{-4})$%
10^{-1}	$-5.11\times(10^{2})$%	-0.60%	$-2.83\times(10^{5})$%	-0.71%
1	$-2.52\times(10^{4})$%	$-2.50\times(10^{2})$%	$-8.58\times(10^{3})$%	$-2.70\times(10^{3})$%

Table 6: Percentage difference, with respect to the initial distance, of the final distance of the satellite after 10 full orbits around the planet. Accuracy of the 4 algorithms as the time step increases from 10^{-4} seconds to 1 second.

As presented in the table 6 Euler-Richardson algorithm gives a accuracy of up to 10^{10} orders of magnitude better at small time steps, of 10^{-4} seconds, than Euler and Verlat. As the time step size increases, the Euler-Cramer algorithm approaches the accuracy of the much more complex Euler-Richardson. That happens at a time step of 10^{-1} seconds. Further increasing the time step of 1 second leads the Euler-Cramer rule to a better accuracy than the Euler Richardson to a order of 10. This suggests that the Euler Cramer algorithm is the best at dealing with big time steps. On the other hand, the Euler and Verlat algorithms have similar order of percentage difference for a time step of 10^{-4} seconds. Whereas the percetange difference for Euler algorithm monotonically increases from am order of 10^{0} to an order of 10^{4}, the Verlat algorithm is more unpredictable and it oscillates between a percentage difference of 10^{4} at a time step of 10^{-2} seconds, to a percentage difference of 10^{3} at a time step of 1 seconds.

If we consider the breakdown of an algorithm at the point where its percentage difference goes above 10%, then we conclude that the Euler Algorithm breaks down first, at 10^{-4} seconds and Euler-Cramer & Euler-Richardson break down at around the same time step size, between 10^{-1} and 1 seconds. However, since, by analysing figure 3, we see that we got a nice orbit for a time step of 1 second, we conclude that this breakdown error is observable only at small distances.

Figure 3 has been obtained by applying the four methods to a real life example. G has been restated to its real value and starting data for the 17 most massive planets has been taken from JPL Horizon. A time step of 1 second, an intermediate interval (for which we get a point of data) is 1 month and the total interval of compiling is 13 months. Thus it was expected that the Earth would do a full revolution around the centre of mass and then come back to its position. We also plotted the position of the Sun with respect to the centre of mass. The findings are displayed below.

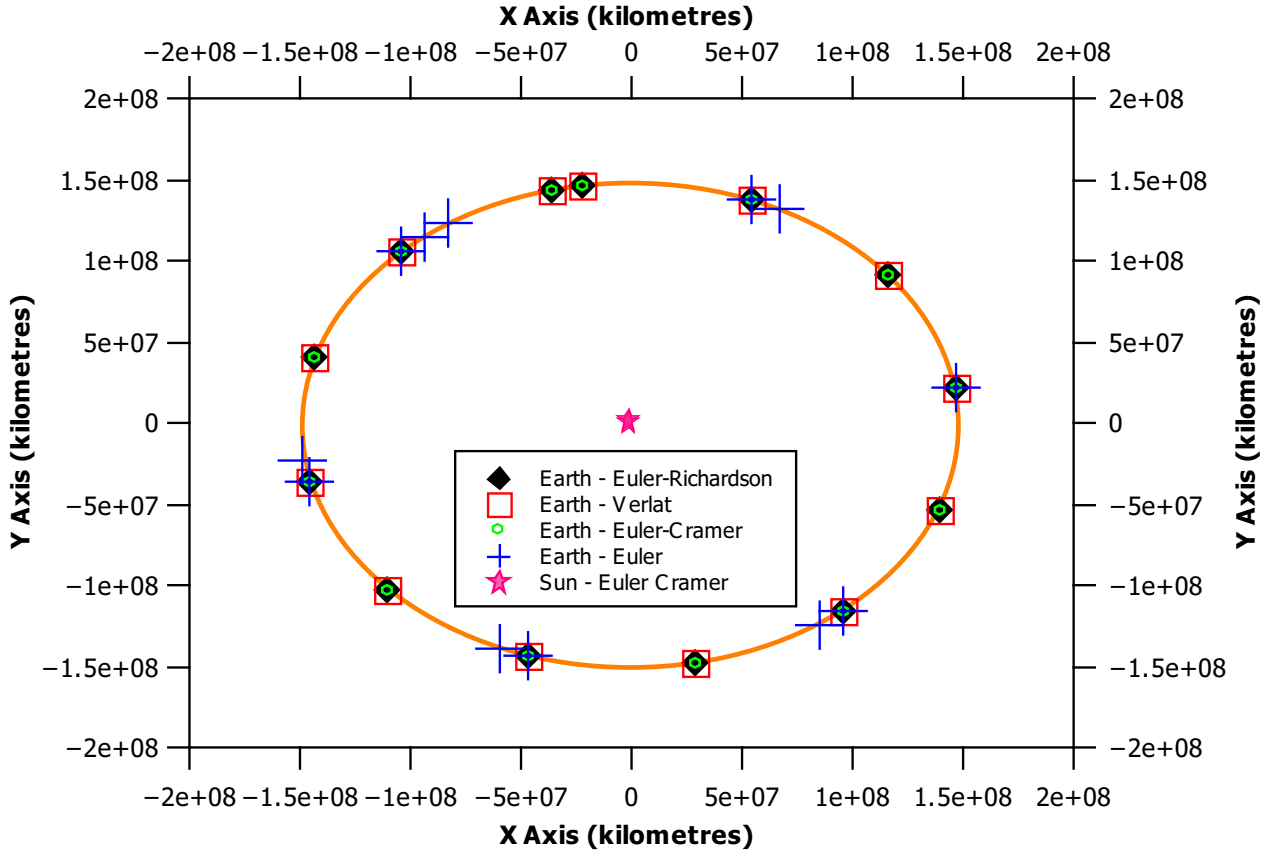

Figure 3: The figure captures the effect of applying the four algorithms: Euler, Euler-Cramer, Verlat and Euler-Richardson to find out the position of Earth once every 1 month, for a total interval of 13 months. The legend describes the algorithm used for each data point and an ellipse has been appended to illustrate the path that the Earth takes around the Sun in one year.

It is striking how well the algorithms match the expected behaviour. All but Euler's algorithm give approximately the same data points, as seen in the figure, while Euler's data follows a different pattern but contours the same ellipse. Since we know from real life that Earth orbits the Sun but that in Our program Earth orbits the centre of mass of the Solar System, we can conclude from the graph that the Sun and the centre of mass are at least within 10^6 kilometres of eachother.

For a further comparison, the final position and velocity of Earth given by different algorithms have been put into a tables 7 and 8,alongside with the data from JPL Horizons for the date of 30th of November 1997. Why the 30th of November and not the 5th of December, which would be exactly one year later? That is because in our program, we defined a month to be 30 days, but as we know, in a year we have 365 days. So in our setting of the intermediate interval to be of 30 days, we lose 5 days in a year, hence why our data matches the date of 30th of November and not the 5th of December.

Position(KM): / Source	X	Y	Z
JPL Horizon	$5.44652 \times(10^7)$	$1.37218\times(10^8)$	$2.95945\times(10^4)$
Euler	$6.67647\times(10^7)$	$1.31183\times(10^8)$	$3.32293\times(10^4)$
Euler-Cramer	$5.44025\times(10^7)$	$1.37243\times(10^8)$	$2.95944\times(10^4)$
Verlat	$5.44034\times(10^7)$	$1.37243\times(10^8)$	$2.96568\times(10^4)$
Euler-Richardson	$5.44025\times(10^7)$	$1.37243\times(10^8)$	$2.95940\times(10^4)$

Table 7: The position of the Earth after a year(365 days), given by different algorithms and compared to the data from JPL Horizon

Velocity(KM/S): / Source	X	Y	Z
JPL Horizon	$-2.80920\times(10)$	$1.11122\times(10)$	$1.13963 \times(10^{-4})$
Euler	$-2.69337\times(10)$	$1.35859\times(10)$	$1.09471\times(10^{-3})$
Euler-Cramer	$-2.80971\times(10)$	$1.10998\times(10)$	$1.10359\times(10^{-4})$
Verlat	$-2.80971\times(10)$	$1.10998\times(10)$	$1.23320\times(10^{-4})$
Euler-Richardson	$-2.80972\times(10)$	$1.10997\times(10)$	$1.20179\times(10^{-4})$

Table 8: The velocity of the Earth after a year(365 days), given by different algorithms and compared to the data from JPL Horizon

Running the program with a time step of 1 second for an interval of one year takes approximately 35 min for Verlat and Euler-Richardson, as they are more complex. On the other hand, Euler-Cramer and Euler finish at around 20 minutes, reflecting the lower level of complexity.

As a further test for the most accurate algorithm yet, Euler Richardson(as demonstrated in table 6), the values at each month of the total: momentum and angular momentum of the Solar System have been plotted on two graphs, 4 and 5, as a function of time. Keep in mind that in a real life situation, total momentum and angular momentum of a system are conserved. The result shows that during the the 13 months period, the total momentum is conserved within a precision of 3 decimal places, while the total angular momentum is conserved within a precision of 7 decimal places.

Finally, the last conserved quantity which was calculates is the position of the centre of mass. In a real life system, the position of the centre of mass of a system should stay the same. Graph 6 sums up our findings for a time step of 1 second, an intermediate interval of 1 day and a total interval of 1 month.

It is clear from the graph that the centre of moves. What is interesting is that it moves in a similar direction for every algorithm, even if it does only for a very small distance - 5000M for Euler's algorithm and 2500M for all the others. That suggests that it is not the program which has a fault but rather that in our ignorance of all the other bodies in the Solar System, we dismissed some that contribute to mantaining the position of the centre of mass. Another important fact is that for Euler algorithm, the position of the centre of mass

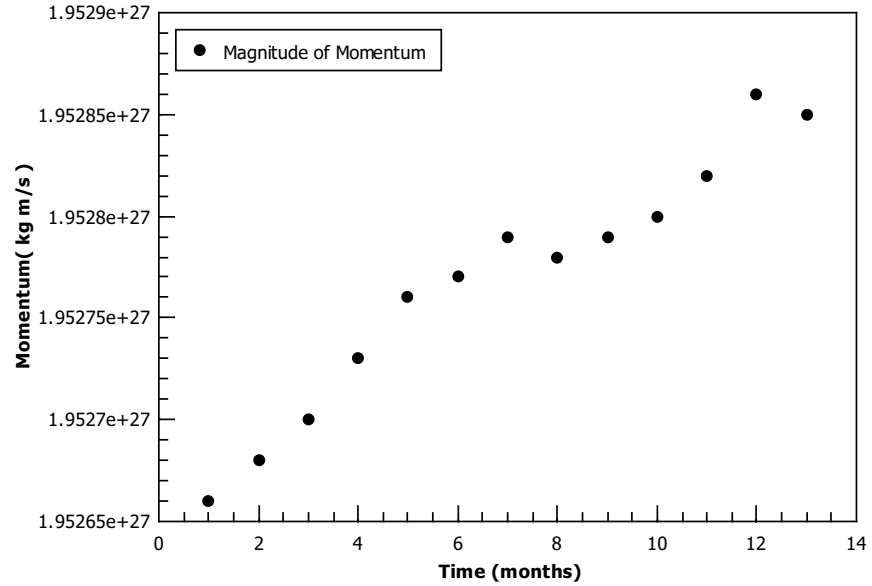

Figure 4: The evolution of the total momentum of the Solar System in a year, as given by Euler Richardson algorithm.

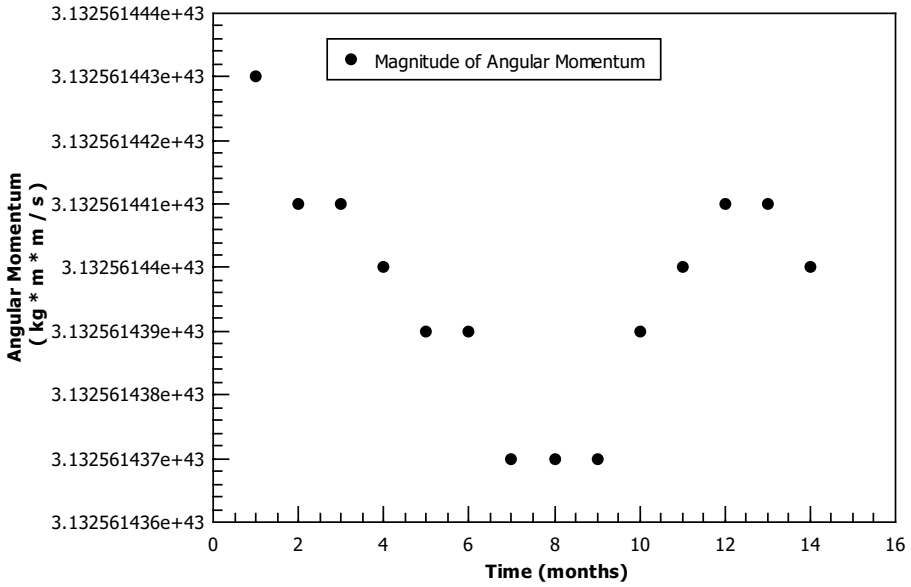

Figure 5: The evolution of the total angular momentum of the Solar System in a year, as given by Euler Richardson algorithm. The time step used is 1 second with an intermediate interval of 1 month.

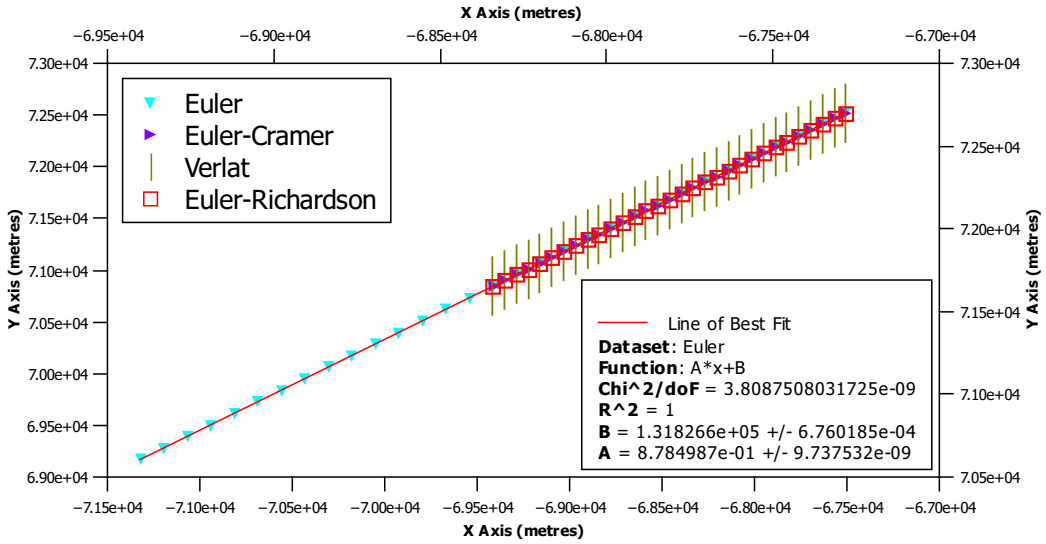

Figure 6: The position of the centre of mass as given by all four algorithms. Data point of the same kind represent different times. The initial data points are on the top right. The time step used is 1 second with an intermediate interval of 1 month.

advances times as further as the other ones.

5 Summary

First, the report proves that for the satellite problem with a time step of 10^{-4} seconds, all the methods give an error of less than 3%, thus they all accomplish their function. While Euler and Verlat algorithms are giving errors between 0% and 3%, the Euler-Cramer and Euler-Richardson give a staggeringly low error of the order of 10^{-7}% and 10^{-10}% respectively. While the errors increase when changing from a time step of 10^{-4} to 1 seconds, their order approaches the same value: between 10^{4}% for Euler and 10^{2}% for Euler-Cramer.

Secondly, plot 3 shows that all four algorithms match the expected behaviour. They calculate the position of the Earth with respect to the centre of mass of the Solar System, at every month of the year(with a time step of 1 second). The plot depicts an elliptical orbit corresponding to all the algorithms. By also plotting the position of the sun it is clear that the centre of mass is within a close distance (at least 10^{6} kilometres) of the Sun, reflecting its mass dominance over the other bodies in the system.

Thirdly, tables 7 and 8 show that the results after 1 year (with a time step of 1 second), for all the algorithms, but for Euler's, match the position and velocity given by JPL Horizons. All algorithms give similiar values within 3 decimal places, except for Euler's. What is worth noting is that because our definition of a month to be 30 days, we lose 5 days each year, as the year is 365 days. Thus the program printed data from the 30th of November 1997 instead of the 5th of December 1997.

Fourthly, the report tested the accuracy of the Euler Richardson method, as this has stood out as the most precise in table 6. The program calculates the value of conserved

quantities like total momentum and total angular momentum of the Solar System and prints the results for every month(with a time step of 1 second). Ideally these would not change, but it is our duty to quantify the errors. As shown in the tables 4 and 5, for a period of 1 year, the momentum stays the same within a precision of 3 decimal places and the angular momentum stays the same within a precision of 7 decimal places.

Lastly, another conserved quantity has been plotted every day for a month(with a time step of 1 second). That is the position of the centre of mass of the Solar System. From plot 6 it is clear that for every algorithm, the centre of mass moves. It moves 5000 metres for Euler's Algorithm and 2500 metres for all the others. That is not a big difference, and a possible cause it the fact that we didn't take into account all the objects in the Solar System.

6 Appendix

Derivation of the orbital speed of a satellite:

$$G = mg \tag{20}$$
$$F_c = m\omega^2 r \tag{21}$$
$$mg = m(\frac{v}{r})^2 r \tag{22}$$
$$g = \frac{v^2}{r} \tag{23}$$
$$v = \sqrt{gr} \tag{24}$$
$$v = \sqrt{\frac{GM}{r^2} r} \tag{25}$$
$$v = \sqrt{\frac{GM}{r}} \tag{26}$$

, where G is the gravitational force, F_c is the centripetal force, m is the mass of the object, g is the gravitational acceleratio, ω is the angular speed, r is the radius of the orbit, v is the tangential speed of the object and M is the mass of the planet being orbited.

References

[1] Wikipedia, Solar System

[2] I. Bertram, J. Nowak, PHYS281 course notes, Michaelmas 2016

[3] D.E. Rutherford, Classical mechanics, Edinburgh : Oliver and Boyd ; New York : Interscience Pub., 1957.

[4] Hugh D. Young, Roger A. Freedman; Sears and Zemansky's university physics : with modern physics, 14th edition, Global edition. 2016

[5] Ryan S. Park, JPL Horizon, 2016