Analysis of Simulating the Solar System Using Kinematic Techniques

Lewis Butcher

12/12/17

Abstract

In this report we compare two kinematic techniques known as the Euler method and the Euler-Cromer method using a code designed to simulate a N-body system, in this case the Solar System, to see how each body within the system interacts. The simulation and methods stated are used to predict the positions and velocities of planets after a given amount of days. The simulation is found to produce the most accurate results when using the Euler-Cromer method for a large time-step of approximately 3600s. We also look at the total momentum and energy within the system in which the momentum is found to fluctuate around $5.545 \times 10^{31} kgm^{-1}$ and the energy is found to be inaccurate and increases at a steady rate over time.

1 Introduction

One of the major problem in Physics is the N-body problem, this involves calculating and predicting the motion of celestial bodies who's gravitational forces influence each others motion. The complexity of solving this problem increases with the amount of bodies within a system, therefore solving it by hand is difficult due to the amount of calculations involved increasing the chances of error. Therefore producing a code to solve this problem is much more efficient and accurate.
In this particular case the code I produced is used to calculate the velocities, positions, momentum and energies for a N bodied system. The simulation carried out within this report is a simulation of the Solar-system, in which there has been inputted eleven 'massive' particles which include, The sun, Mercury, Venus, Earth, The Moon, Mars, Jupiter, Saturn, Uranus, Neptune and Pluto. There are two different methods in solving the problem implemented in the code, the first method is known as the Euler method (Equations 4 and 5) and the second method is known as the Euler-Cromer method (Equations 6 and 7).
Section 2 of the report discusses what equations are in the code and the theory behind them. Section 3 discusses the structure and design of the code. Section 4 discusses the findings and results produced. Finally in section 5 one will discuss the key findings from the data.

2 Theory

The N-body gravitational system is a challenging problem in Physics. Simulating the affect of gravity between N 'massive' particles requires using the equation 1. This equation calculates the net acceleration of particle which is influenced under N 'massive' particles gravitational force, it is given by

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

Where $\vec{g}_i$ is the net acceleration for the i^{th} particle, G is the gravitational constant, m_j is the mass of the particle j^{th}, r_{ij} is the distance between the i^{th} particle and the j^{th} particle, $\hat{i}_{ij}$ is the unit vector of the distance between the i^{th} particle and the j^{th} particle [1].
The equation is a result from equating Newtons law of Gravitation 2 and Newtons second law of motion 3.

$$\vec{F}_{ij} = \sum_{i \neq j}^{N} \frac{-Gm_i m_j}{r_{ij}^2} \hat{r}_{ij} \tag{2}$$

$$\vec{F} = m\vec{a} \tag{3}$$

Thus arises another challenge to approximate the velocity and position of the particle. Simulating dynamic can be done with several different methods, two in particular are the Euler method, equations 4 and 5, and the Euler-Cromer method, equations 6 and 7. These two methods are the ones implicated in the code used to find the results displayed in this report. The Euler method approximates the position and velocity of the particle at a time $t + \Delta t$.

Given that we know the time t of the particle earlier. We also assume that the acceleration, a, is constant throughout these time-steps.

$$\vec{v}_{n+1} \approx \vec{v}_n + \vec{a}_n \Delta t \tag{4}$$

$$\vec{x}_{n+1} \approx \vec{x}_n + \vec{v}_n \Delta t \tag{5}$$

Where $\vec{v}_{n+1}$ is the velocity at the end of the time-step, $\vec{v}_n$ is the velocity at the start of the time-step, $\vec{a}_n$ at the start of the time-step and Δt is the time-step [1].
The Euler-Cromer method is known for being more accurate than the Euler method for a longer time-iteration range tho they are similar to each other. This method uses the velocity at the end of the step rather than at the beginning like the Euler method does, hence giving more stable results.

$$\vec{v}_{n+1} \approx \vec{v}_n + \vec{a}_n \Delta t \tag{6}$$

$$\vec{x}_{n+1} \approx \vec{x}_n + \vec{v}_{n+1} \Delta t \tag{7}$$

Where $\vec{v}_{n+1}$ is the velocity at the end of the time-step, $\vec{v}_n$ is the velocity at the start of the time-step, $\vec{a}_n$ at the start of the time-step and Δt is the time-step [1].
Calculating the total momentum of a N body system of 'massive' particles is also another challenging problem. Total momentum is always conserved within a system. We calculate the momentum by using:

$$\vec{p_i} = \sum_{i \neq j}^{N} m_i \vec{v_i} \tag{8}$$

Where $\vec{p_i}$ is the total momentum for the i^{th} particle, m_i is the mass of the i^{th} particle and $\vec{v_i}$ is the velocity of the i^{th} particle.
The energy in a system is given by the Virial theorem it displays the relation between the Kinetic energy and the Potential energy of a system. It states that for n points of particles bound together into a system the time average of the KE of th particles, plus one half of the time average of the GPE of the particles is equal to zero [3].

$$KE + \frac{1}{2}U = 0 \tag{9}$$

Where KE is the Kinetic Energy and is calculated by

$$\sum KE = \sum_{i}^{N} \frac{1}{2} m_i \vec{v_i^2} \tag{10}$$

Where m_i is the mass of the i^{th} particle and v_i is the velocity of the i^{th} particle. and U is the Gravitational Potential Energy and is calculated by

$$\sum U = \sum_{i}^{N} \frac{-G m_i m_j}{r_{ij}} \tag{11}$$

Where m_i is the mass of the i^{th} particle, m_j is the mass of the j^{th} particle, r_{ij} is the distance between the two prticles i and j and G is the gravitational constant.

3 Design

The Java code is made up from 4 files, SolarSystem.java, Particle.java, GravField.java and PhysicsVector.java. Below are class diagrams for GravField (1) and Particle (1) as well as a flow chart for the main method SolarSystem, Figure.1.

3.1 GravField

GravField
$\underline{G}$ final Newtons Gravitational Constant (6.67408×10^{-20}) $\text{Nkm}^2 kg^{-2}$
+getg(X: Particle [N]): PhysicsVector [N]

Table 1: GravField Class Diagram

GravField is a class designed to calculate the total acceleration on each particle in the system due to the effect of gravity caused by the other particles.

Methods:
+getg(X: Particle [N]): PhysicsVector [N]: Caclulates the total acceleration for each particle and stores the answers in an array of length N.
Calculate by hand g for a two body system and sum the total g's acting on one body.

3.2 Particle

Particle
- mass: double - position: PhysicsVector - velocity: PhysicsVector - Name: String - $\underline{G}$ final Newtons Gravitational Constant (6.67408×10^{-20}) Nkm$^2kg^{-2}$
<< constructor >> Particle(pi: double, pj: double, pk: double, vi: double, vj: double, vk: double, m: double, n: String) +Distance(a: PhysicsVector, b: PhysicsVector): double +getMomentum(Z: Particle[N]): double +getKinetic(Z: Particle[N]): double +getPotential(Z: Particle[N]): double +upParticle(g: PhysicsVector, t: double, M: int): void +GetMass(): double +GetVelocity(): PhysicsVector +GetPosition(): PhysicsVector +GetName(): String

Table 2: Particle Class Diagram

Particle is a class designed to solve several calculations involving the 3D vectors for position and velocity, in the form $xi + yj + zk$. The class contains methods for solving the distance between particles, total momentum, Kinetic energy, Potential energy as well as a method to update the positions and velocites of each particle per time-step.

Constructors:

Particle(pi: double, pj: double, pk: double, vi: double, vj: double, vk: double, m: double, n: String): is the constructor which creates two PhysicsVectors for position and velocity as well as a double for the mass and a string for the name.
Create a particle (1,2,3,4,5,6,10,"Name") and print out to see if it works.

Methods:

+Distance(a: PhysicsVector, b: PhysicsVector): double: returns the magnitude of the vector dispalcement between two PhysicsVectors (a_1, a_2, a_3) and (b_1, b_2, b_3) by $\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+(a_3-b_3)^2}$.
subtract (1,2,3) from (3,5,7) and check that we obtain (2,3,4), then take the magnitude of that vector and check it equals $\sqrt{29}$.

+getMomentum(Z: Particle[N]): double: returns total momentum of the system by

summing the individual momenta of each particle using $\vec{p} = m\vec{v}$.
Create an array for a particle with (1,0,0) for the velocity and a mass of 10kg, Use the equation to check that we obtain the momentum as $10kgms^{-1}$.

+**getKinetic(Z: Particle[N]): double**: returns the total Kinetic energy of the system by summing the individual kinetic energy for each particle using $KE = \frac{1}{2}mv^2$.
Create a an array for a particle with the velocity (1,0,0) for a particle of mass $m = 10kg$. *Using the equation for KE make sure we obtain* $5J$.

+**getPotential(Z: Particle[N]): double**: returns total Potential energy of the system by summing the individual energies for each particle using $U = \frac{-GMm}{r}$.
Consider two masses for $10kg$ *and* $100kg$ *with a distance of* $100m$ *between them, use the equation and check we obtain* 6.67408×10^{-11}.

+**upParticle(g: PhysicsVector, t: double, M: int): void**: returns the updated position and velocity of a particle by using the Euler method (4 and 5) or the Euler-Cromer method (6 and 7).
Work out a simple two body system by hand and use the same values for the method and check you obtain the same values.

+**GetMass(): double**: returns the mass component of the particle array.
Create a Particle and check it returns a double for the Mass.

+**GetVelocity(): PhysicsVector**: returns the PhysicsVector for the particles velocity.
Create a particle and check it returns the PhysicsVector for the particles velocity.

+**GetPosition(): PhysicsVector**: returns the PhysicsVector for the particles position.
Create a particle and check it returns the PhysicsVector for the particles position.

+**GetName(): String**: returns the Name of a particle.
Create a particle and check it returns the String for the particles Name.

3.3 SolarSystem

This is the main method for the code (Figure.1). It is designed to allow the user to input the method, time-step and the amount of days to run the simulation. Each of the 'massive' particles used in the simulation are denoted here, with the velocity, position, mass and name all stated. The result are then all printed to 3 files (PositionsVelocities.txt, Momentum.txt and Energy.txt).

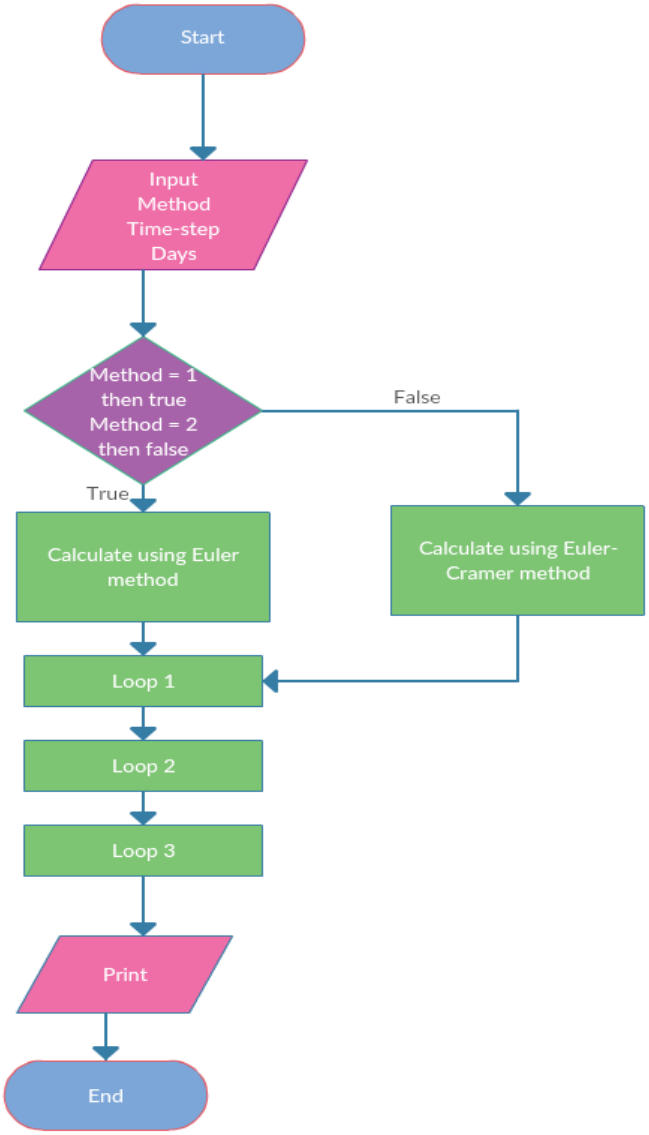
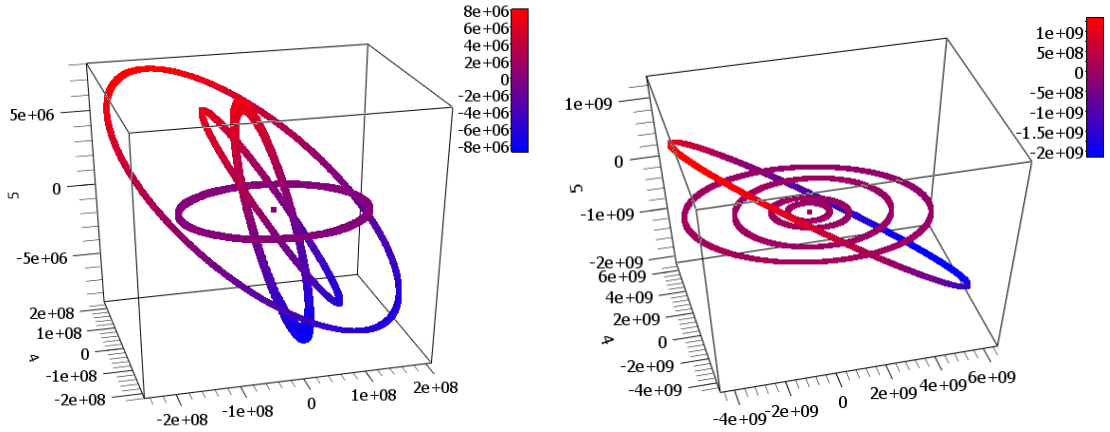


Figure 1: Flow Chart for the SolarSystem.java class.

Loop 1: Uses a for loop to implement each planets acceleration using the Euler or Euler-Cromer method to update the velocity and position.
Loop 2: Uses an if loop to print the velocity and position for each full day.
Loop 3: Uses an if loop to print the Kinetic and Potential energy for each full day.

4 Results

The main aim for this java code was to produce a functioning model of the Solar-system involving object-orientated techniques. The results collected are from two different methods in calculating the positions and velocities of the planets. Below are two 3D plots of the trajectories of planets for the inner Solar system for 50 years, Figure 2a and the outer Solar-system for 300 years, Figure 2b .



(a) Graph of the trajectories of planets from the inner solar system for 50 years.

(b) Graph of the trajectories of planets from the outer solar system for 300 years.

The difference in distances between the outer and inner is so large that plotting them on a single graph wouldn't be presentable due to the differences in the radius orbit.
Each simulation carried out began on the date $6th$ December 2017 at midnight, these initial values were taken from the NASA Horizons website [2]. The results that have been analysed have also been compared against the data from the NASA Horizons website [2].

4.1 Euler method

The Euler method was used in calculating the positions of Saturn over 365 days with 3 different time-steps, 1s, 60s, and 3600s. The resultant position for Saturn for each day was collected into a file and displayed on graph 3.

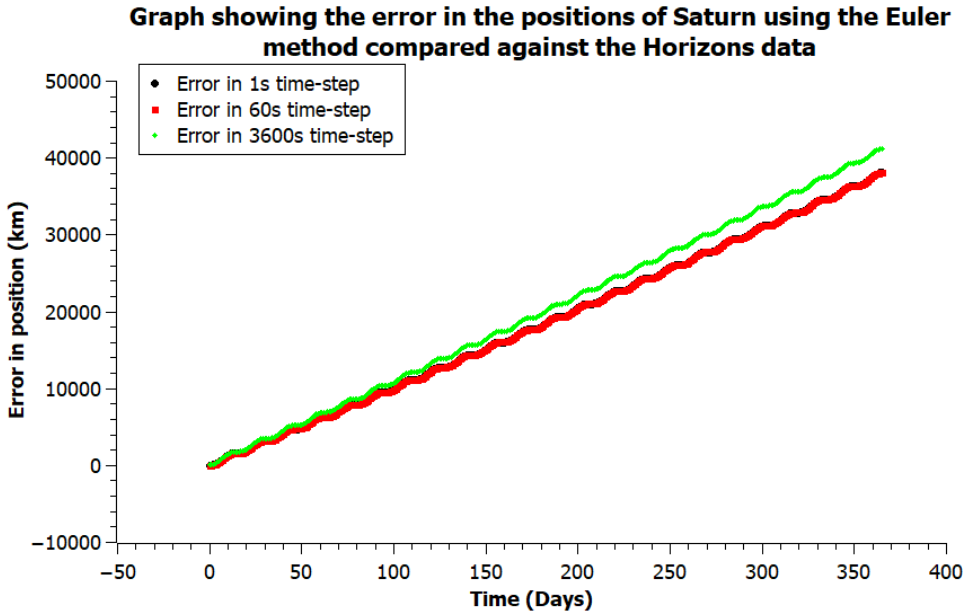


Figure 3: Graph of the error in the position of Saturn for different time-steps using the Euler method.

Each data point is found by taking the Horizon Databases [2] values and finding the absolute value between the magnitudes of both Horizons data and the data found using the Euler method.
The table below show the error found in the results from the Euler method for every 60 days for each time-step.

Time-step (s)	Days	Error (km)
1	0	0.0306286811
1	60	6271.21825
1	120	11993.4405
1	180	17868.7630
1	240	24352.2829
1	300	31058.1850
1	360	37360.7798
60	0	1.83181786
60	60	6202.73417
60	120	12011.1458
60	180	17894.5531
60	240	24386.2435
60	300	31100.4314
60	360	37411.4575
3600	0	110.237613
3600	60	6783.55407
3600	120	13073.8238
3600	180	19442.3562
3600	240	26424.2661
3600	300	33635.5871
3600	360	40452.4835

Table 3: Table showing the error in position for every 60 days using the Euler method for three different time-steps over 365 days.

4.2 Euler-Cromer method

For the results using the Euler-Cromer method the same procedure previously carried out with the Euler method has been performed for 365 Days for 3 different time-steps, 1s, 60s and 3600s. The error calculated using the Euler-Cromer method has been displayed below on graph 4

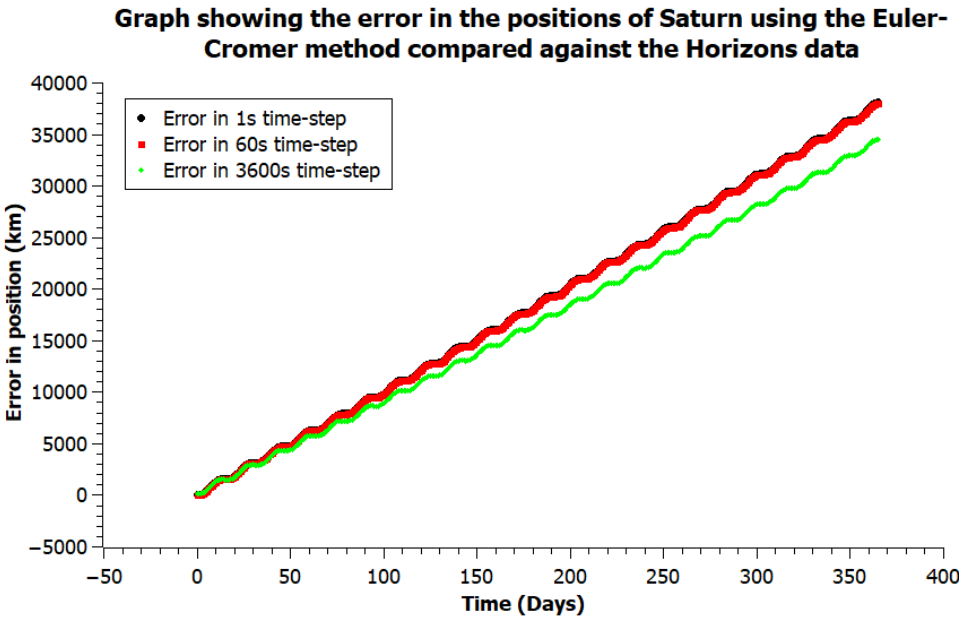


Figure 4: Graph of the error in the position of Saturn for different time-steps using the Euler-Cromer method.

The values plotted are found by performing the same calculations for the error in the Euler Methods results.
The table below show the error found in the results from the Euler-Cromer method for every 60 days for each time-step.

Time-step (s)	Days	Error (km)
1	0	0.0306286811
1	60	6192.76054
1	120	11992.8375
1	180	17867.8542
1	240	24351.0682
1	300	31056.6626
1	360	37358.9487
60	0	1.83081793
60	60	6184.80204
60	120	11974.9582
60	180	17840.0685
60	240	24313.3981
60	300	31009.1377
60	360	37301.6062
3600	0	109.477468
3600	60	5706.87125
3600	120	10901.82800
3600	180	16172.5422
3600	240	22052.7738
3600	300	28157.2111
3600	360	33860.6638

Table 4: Table showing the error in position for every 60 days using the Euler-Cromer method for three different time-steps over 365 days.

4.3 Euler Euler-Cromer Comparison

Each error data point from the Euler method graph 3 and Euler-Cromer metho graph 4 for Saturn have been plotted onto a single graph for comparison.

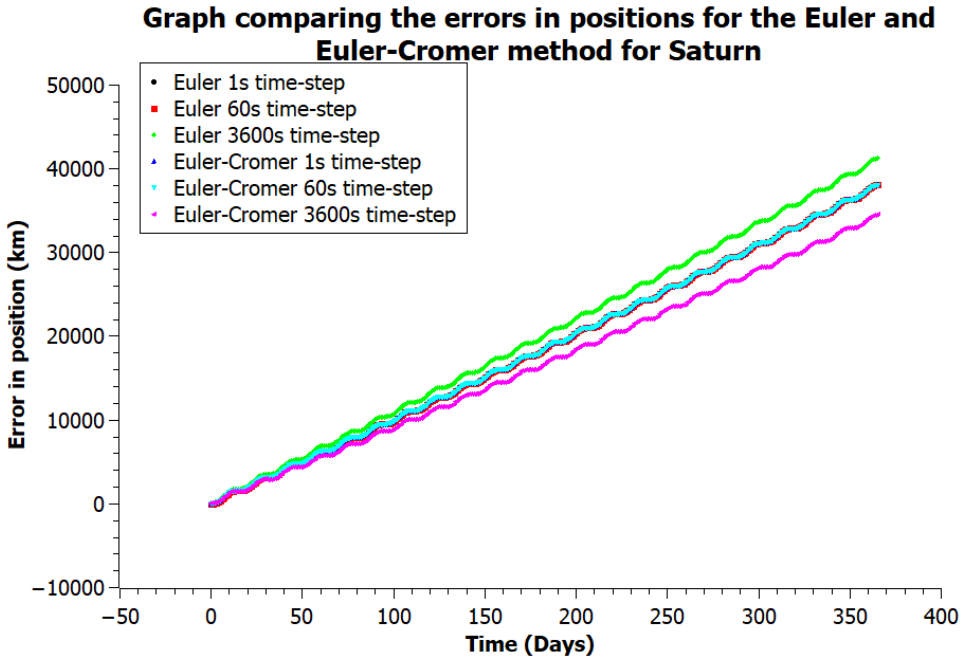


Figure 5: Graph of the error in the position of Saturn for different time-steps using the Euler and Euler-Cromer method.

4.4 Momentum

Using Figure 6 we see that the total momentum of the system isn't constant. The momentum varies from approximately $5.746 \times 10^{31} kgms^{-1}$ to $6.471 \times 10^{31} kgms^{-1}$. This inconsistency over 100 years is mainly due to the high order of values used in the calculations. We can also see that the total momentum fluctuates around $5.545 \times 10^{31} kgms^{-1}$

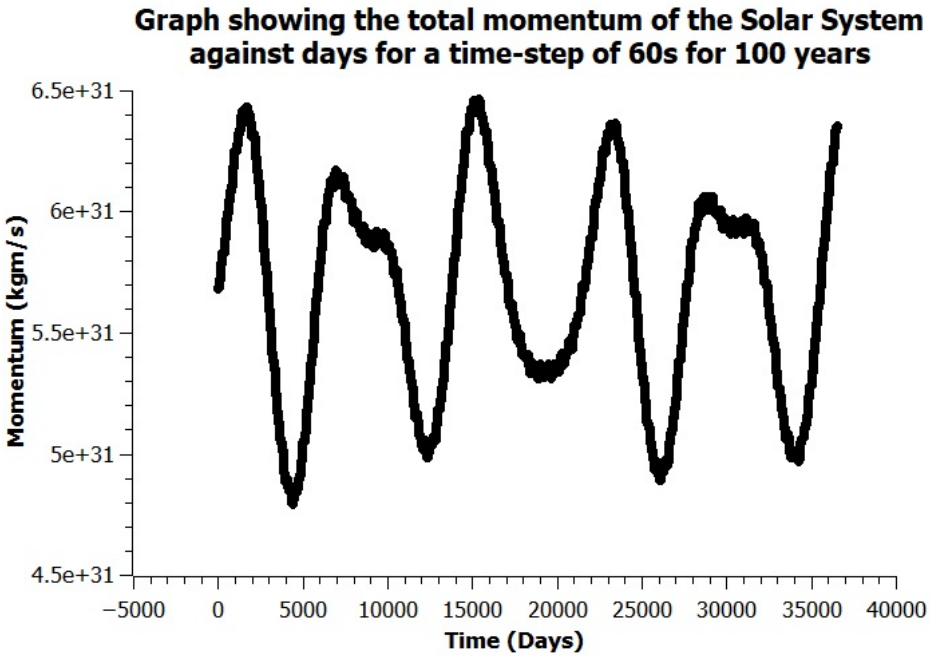


Figure 6: Graph of the total momentum for 100 years.

4.5 Energy

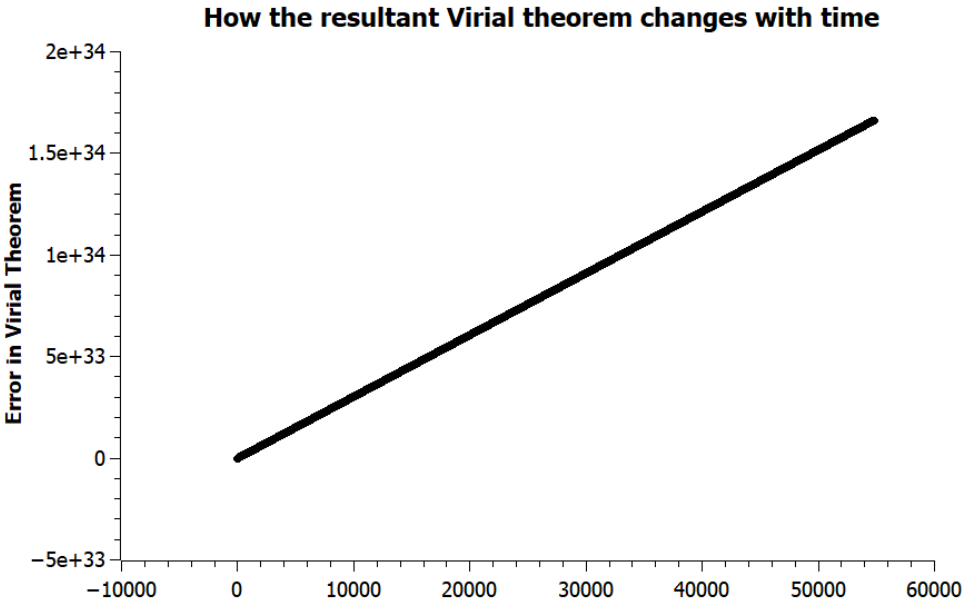


Figure 7: Graph of the values for the Virial theorem obtained when the simulation ran for 150 years with a time-step of 60s.

5 Summary

The results show that the simulation is able to map out the trajectories of planets, Graphs 2a and 2b, however the accuracy of the positions of the planets varies depending on the method used within the code. Firstly, when comparing the the methods used in this code, Euler and Euler-Cromer, we see different variations in accuracy due to the different time-steps used in the calculations. Looking at figure 5 we see that that for the Euler method the accuracy of the results for a small time-step of 1s is poor, the total difference from the initial error (Day 0) to the final error (Day 365) is $38086.5km$ (6 s.f). When the time-step is 60s we find that the difference between the initial error and final error is $38136.8km$ (6 s.f). When the time-step is 3600s we see that as the simulation approaches 365 days the error increases from $110.238km$ to $41230.3km$ with a difference of $41120.1km$ (6 s.f). Furthermore, using figure 5 once more, we see that for the Euler-Cromer method the accuracy of the results for a small time-step of 1s is yet again poor, the difference between the initial and final error here is $38084.7km$ (6 s.f). When the time-step is 60s the difference between the initial and final error is $38024.6km$ (6 s.f). However when we use a large time-step of 3600s we see that the positions accuracy increases with a difference between the initial and final of $34417.3km$ (6 s.f). Therefore this suggests that the best method for predicting the position of planets is the Euler-Cromer method for a large time-step of approximately 3600s.
Secondly, the momentum of the system has been found to fluctuate throughout the simulation around $5.545 \times 10^{31} kgms^{-1}$, we also find that the maximum momentum for the simulation is $6.471 \times 10^{31} kgms^{-1}$ and the minimum momentum is $5.746 \times 10^{31} kgms^{-1}$. The conservation of momentum for the simulation does not hold, this could be due to error within the code itself and also since there are errors on the positions and velocities of each planets, the error in the momentum is going to increase significantly since it is calculated using these values.
Finally, the energy within the system is found not to follow the Virial Theorem, the program was run for 150 years, in which the energy in the system increased from 0J to $1.72348 \times 10^{34} J$. therefore the energy and error within the system increases with time. This anomaly is most probably due to incorrect coding since the pattern is similar to the pattern found in the error of both methods.
The investigation carried out in this course was very limited. Having only 2 weeks to produce an accurate code would be near impossible due to the complexity of the coding.
Overall the results found were acceptable considering the limitations and the time spent on the code. To improve this one would spend more time on correcting mistakes within the code and using a better method for updating the velocities and positions of the planets e.g The Verlet algorithms.

References

[1] I. Bailey, J. Nowak, PHYS281 course notes, Michaelmas 2015

[2] NASA Horizons

[3] Virial Theorem