Gravitational Particle Simulation

Tehmoor Hussain

10/12/17

Abstract

A mathematical model of particles producing gravitational fields and this effect on the motion of other particles was implemented in Java. The error for low time intervals was was found for different bodies and algorithms and found to be a constant $2ms^{-1}$. Which meant that the error was $\alpha\ (\Delta t)^2$ as predicted by theory. The Euler-Cramer algorithm was found to conserve energy as the radius of orbit remained constant, the Euler algorithm added energy into the system for higher time intervals and time. This corresponded to a non constant radius ie. $r\ \alpha\ ln(t)sin(\omega t)$. An increase in time period each orbit of 1.31 times was found. Which is the difference from the analytic solution of constant semi major axis for solar system or radius for two body system, from the simulated solution.

Contents

1 Introduction

Newton's law of universal gravitation and Newton's laws of motion were implemented in the Java programming language and are the subject of this report. This was in order to see the effect of the gravitational field produced by a set of particles due to their masses, on other particles. In particular, on their motion which was modelled by the Euler method and the Euler-Cramer method.

In order to model a n-body system of particles, the Java programming language was chosen due to its widespread use in mathematical modelling. As of now there are no realistic analytic solutions to the n-body problem of gravitational interaction; however, numerical techniques can be used to approximate the solutions. In a mathematical model, the accuracy of the solution obtained is dependent on the techniques used. This is why multiple techniques were used, in order to compare their relative effectiveness.

By modelling a part of the solar system we can use our simulation to obtain the position, acceleration and velocities of all the the planets implemented. This is otherwise difficult unless by other mathematical models or by observation, which provides a justification to the model.

After Newton's laws of motion and gravitation were implemented using numerical modelling, a simple model was constructed consisting of two bodies orbiting their shared centre of mass. This was in order to test the accuracy of the model against known analytic solutions of the two body problem. Also, conserved quantities were measured in the simulation like energy and momentum to see if the simulation adhered to reality.

Section 2 explains the theory used in the simulation, section 3 explains how the code used in the model works. While Section 4 provides a description of how the testing procedure of the model was carried out. Section 5 and 6 respectively are where the findings in the report are detailed and any viable conclusions based on data collected are recorded.

2 Theory

Newton's law of gravitation in summation form describes the field produced by a point mass in terms of its mass and the radial distance from its centre to another point mass placed in the field[1].

$$\vec{a_i} = \sum_{i \neq j}^{N} \frac{-Gm_j\vec{r}_{ij}}{r_{ij}^3} \tag{1}$$

Where $\vec{a}_i[ms^{-2}]$ is the total acceleration felt by a point mass placed in the total field produced by multiple mass's $m_j[kg]$. $\vec{r}[m]$ is the radial vector from $m_i[kg]$ to $m_j[kg]$ and $r[m]$ is its magnitude. This equation is is used so each particle in the simulation produces its own field which is felt by every other particle, this in turn causes them to accelerate and change position. $i \neq j$ in the sum so the radial vector isn't calculated from a mass to itself.

The following equation represents the gravitational potential energy of a mass m_j placed in the field produced by mass's m_i. Where $U_j[J]$ is the total gravitational potential energy of m_j and r is the magnitude of the distance between m_i and m_j.

$$U_j = \sum_{i \neq j}^{N} \frac{G m_i m_j}{r_{ij}} \tag{2}$$

It was used to calculate energy conservation in the system along with the following equation to see if the simulation was an accurate model.

The kinetic energy of a particle is given by the following expression and was used in conjunction with the potential energy to check total energy conservation for a particle. $E_j[J]$ is the kinetic energy of a particle with mass m_j moving at velocity $v_j[ms^{-1}]$.

$$E_j = \frac{1}{2} m_j v_j^2 \tag{3}$$

The Virial theorem describes the potential energy to be twice that of the kinetic energy hence it can be written as follows[3].

$$2E_j - U_j = 0 \tag{4}$$

In order to simulate the position and velocity of the particles the Euler method was used after obtaining the acceleration of a particle. $\vec{v}_{n+1}[ms^{-1}]$ is the next velocity in the step, $\vec{v}_n$ is the current velocity, $\vec{a}_n$ is the current acceleration, $\vec{x}_{n+1}[m]$ is the next position, $\vec{x}_n$ is the current position, $\Delta t[s]$ is the time interval selected and $\mathcal{O}(\Delta t)^2$ is high order terms in the expansion.

$$\begin{aligned} \vec{x}_{n+1} &= \vec{x}_n + \vec{v}_n \Delta t + \mathcal{O}(\Delta t)^2 \\ \vec{v}_{n+1} &= \vec{v}_n + \vec{a}_n \Delta t + \mathcal{O}(\Delta t)^2 \end{aligned} \tag{5}$$

The Euler-Cramer method is essentially the same aside from the velocity is updated first which provides a more stable algorithm. Where the error in the algorithm from the expected value is given by the contribution from $\mathcal{O}(\Delta t)^2$.

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

This equation is the momentum $\vec{p}_j[kgms^{-1}]$ of a particle with mass m_j and moving at velocity $\vec{v}_j$.

$$\vec{p}_j = m_j \vec{v}_j \tag{7}$$

The Solution to the two body problem for velocity is given as follows if they are equal masses orbiting a fixed distance away from their centre of of mass. Where m is the mass of one of the bodies and $\vec{r}$ is the vector pointing from one mass to the other.

$$\vec{v} = \sqrt{\frac{Gm}{2\vec{r}}} \tag{8}$$

Where $T[s]$ is the time period of an orbiting mass, m at radius r.

$$T^2 = \frac{4\pi^2 r^{\frac{3}{2}}}{Gm} \tag{9}$$

3 Design

Particle
-Position: PhysicsVector -Velocity: PhysicsVector -AccelerationSum: PhysicsVector -Momentum: PhysicsVector -VirialConstant: double -PotentialEnergySum: double -KineticEnergy: double -ParticleMass: double -Time: double
<< *constructor* >> Particle(InputTime: double, GenericMass: double, InitialPosition: PhysicsVector, InitialVelocity: PhysicsVector) +euler(): void +eulerCramer(): void +calculateAccelerationSum(PlanetField: GravField, BodyA: Particle, BodyB: Particle): PhysicsVector +resetAccelerationSum(): PhysicsVector +getPosition(): PhysicsVector +getMomentum(): PhysicsVector +calculatePotentialEnergySum(GenericField: GravField): double +calculateEnergyConservation(): double +getVirialConstant(): double +getVelocity(): PhysicsVector +resetPotentialEnergySum(): double

Table 1: Class diagram for the Particle class

Particle is a class representing particles, with a given initial velocity and position are then moved according to the acceleration calculated in the calculateAccelerationSum() method. With either the euler() or eulerCramer method represented by equations 5 and 6 respectively.

Constructors:
Particle(InputTime: double, GenericMass: double, InitialPosition: PhysicsVector, InitialVelocity: PhysicsVector) is the constructor where Velocity = InitialVelocity, Position = InitialPosition, ParticleMass = InputMass, Time = InputTime for each particle.
Methods:
+euler(): void: implements equation 5 in Java to move the particle.
+eulerCramer(): void: implements equation 6 in Java to move the particle.
+calculateAccelerationSum(PlanetField: GravField, BodyA: Particle, BodyB: Particle): PhysicsVector: adds the current value of acceleration for that body to the total acceleration AccelerationSum, then returns AccelerationSum.

+resetAccelerationSum(): PhysicsVector: resets the value of AccelerationSum used after AccelerationSum has been calculated for a particle and used to move it.
+getPosition(): PhysicsVector: returns the current value of position for a particle.
+getMomentum(): PhysicsVector: calculates the momentum of a particle then returns its momentum using equation 7.
+calculatePotentialEnergySum(GenericField: GravField): double: adds the current potential energy from the GravField class due to the fields of all the other particles to the total potential energy of that particle, PotentialEnergySum.
+calculateEnergyConservation(): double: calculates the kinetic energy of a particle using equation 3, then uses the total potential energy as well to implement the Virial theorem, equation 4. Lastly it returns the energy left over after the arithmetic operation in 4 has been completed, the VirialConstant.
+getVelocity(): PhysicsVector: returns the velocity of the particle.
+resetPotentialEnergySum(): double: resets the total potential energy to zero. Used after the total potential energy has been calculated for a particle then also used.

GravField
-PlanetMass: double -RadialMagnitude: double -PotentialEnergy: double -RadialVector: PhysicsVector -RadialUnit: PhysicsVector -Acceleration: PhysicsVector -GravConstant final: double
<< *constructor* >> GravField(InputMass: double) +calculateAcceleration(BodyA: Particle, BodyB: Particle): PhysicsVector +getPotentialEnergy(MassA: double): double

Table 2: Class diagram for the GravField class

Gravfield is a class representing a gravitational field produced by a point particle, with a corresponding method representing the acceleration experienced by another particle placed in the field. Also has a method to represent the potential energy of a particle placed in the field.

Constructors: GravField(InputMass: double): this constructor is passed the mass of the particle it is calculating the field of for use in the class.
Methods: +calculateAcceleration(): PhysicsVector: This method is an implementation of equation 1, the summation part is implemented in the main method, however. It is passed two particles and calculates the radial vector from one to the other and then uses equation 1.
+getPotentialEnergy(): double: This method is an implementation of equation 2, the summation part is implemented in the main method, however. It is passed the mass of a body placed in the field to be used to calculate the gravitational potential energy.

FileIO
+readFromFile(MassArray: double[], NameArray: double[], GenericPositions: PhysicsVector[], GenericVelocities: PhysicsVector[], GenericComponents: double[]): void +writeToFile(GenericNames: String, GenericBody: Particle, filewriter: PrintWriter): void

Table 3: Class diagram for the FileIO class

FileIO This is a class used mainly to write and read useful data from and to file. It was mainly created so the main method was less cluttered and so some code could be reused.

Methods:
+readFromFile(): void: this method is used to read the initial conditions of the simulation to various arrays representing those initial conditions and properties.
+writeToFile(): void: This method is used to write all useful data to file to be looked at or plotted as a graph.

Planet	Simulated Semi Major Axis(m)	Expected Semi Major Axis(m)
Venus	1.093E11	1.082E11
Earth	1.514E11	1.496E11
Mars	2.477E11	2.279E11

Table 4: Table showing difference between expected semi major axis and calculated semi major axis[2]

The following flowchart represents the main method in the Java code. It is however, a simplification and doesn't include the part where new objects are created with their corresponding initial conditions. This is done by initialising the elements of the PhysicsVector arrays which represent the particles used in the simulation and the fields that those particles possess using a for loop.

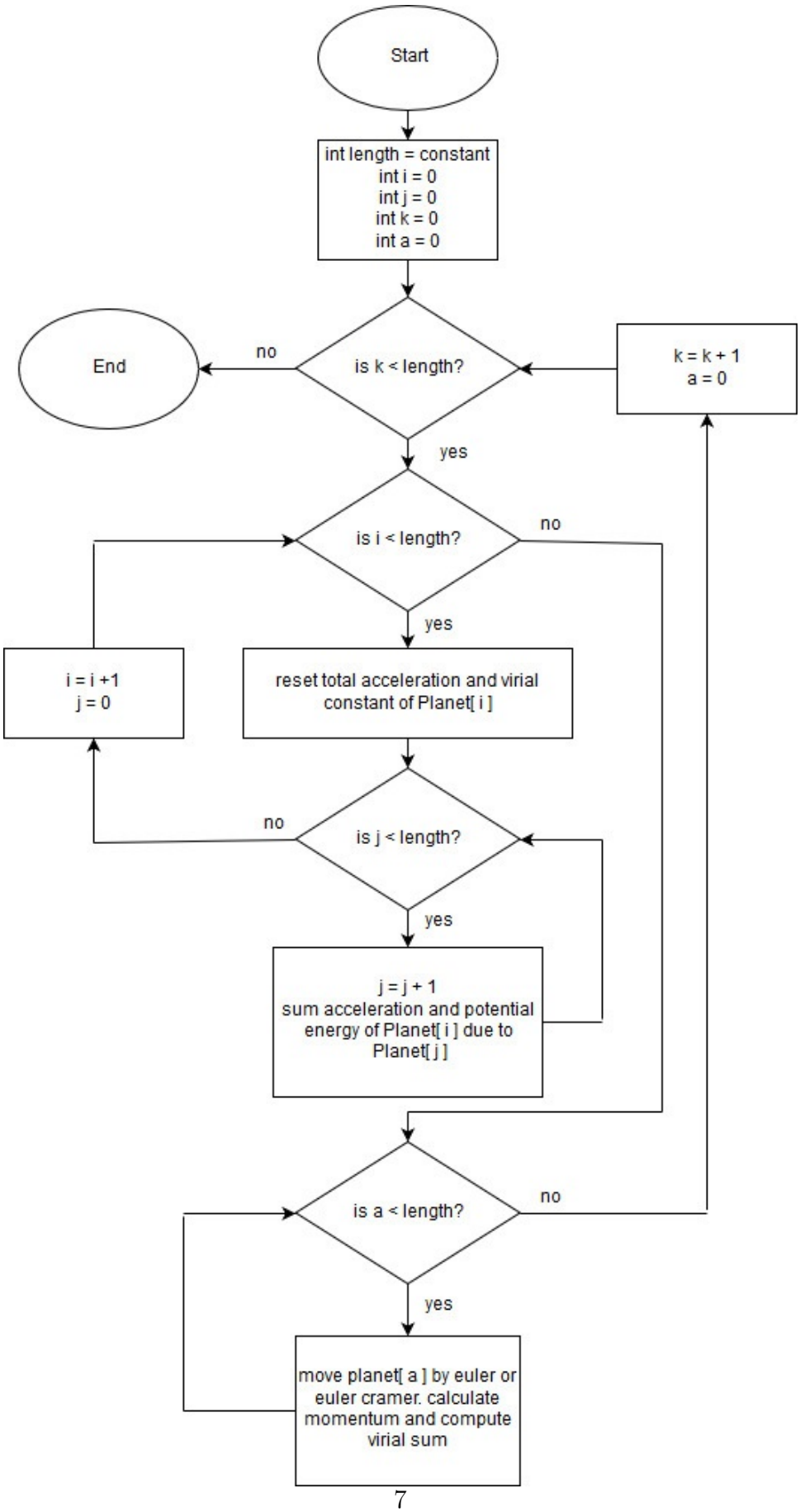

Figure 1: Flowchart of main method

4 Testing

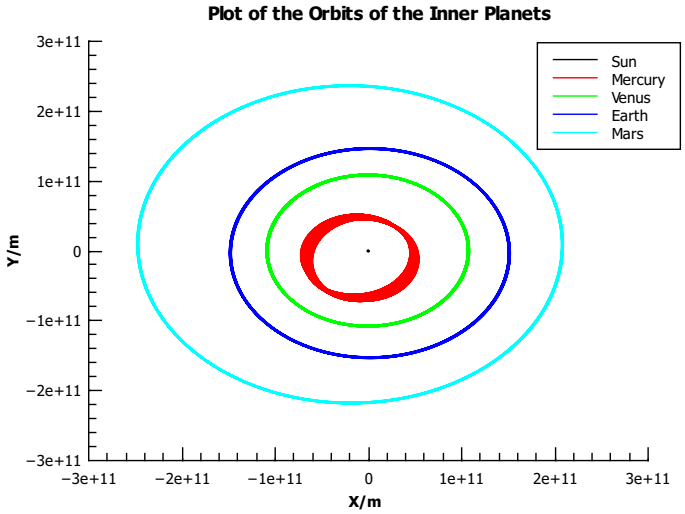

Figure 2: Elliptical orbits around Sun

From figure 2 and table 4 and we can see there is a good agreement between expected and simulated semi major axis. However for the further planets out from the sun the error seems to increase, this could be due to implementation of only 7 planets hence less force pulling the planets out.

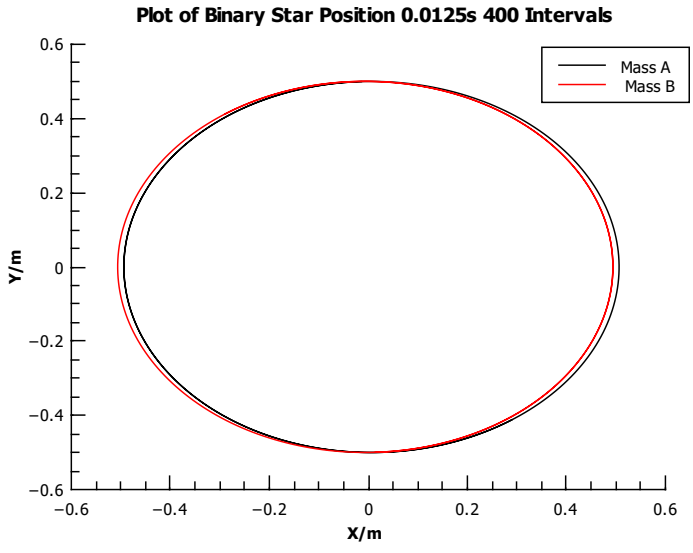

Figure 3: Graph showing orbit of two mass system

Looking at figure 3 we can see the orbits of the two masses in the two body problem. It is as expected, they orbit a shared centre of mass in a circular orbit. Both masses were set to

$1kg$, the distance between them 1m, the gravitational constant to $2Nm^2kg^{-2}$, and velocity to $1ms^{-}1$ in order for them to orbit as required from equation 8. The Euler-Cramer algorithm was used, the reasoning behind this is explained in the analysis.

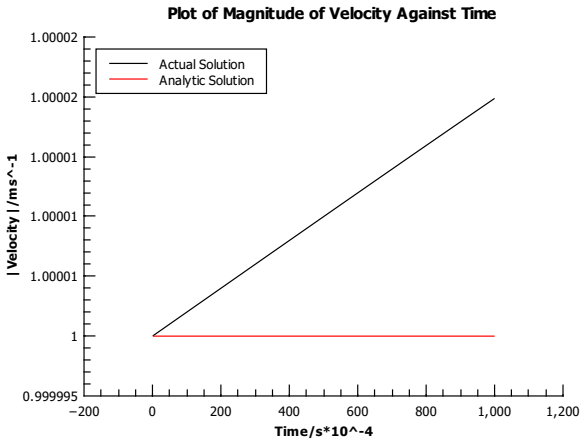

Figure 4: Plot of the magnitude of the velocity of one mass against intervals

From figure 4 the magnitude of the velocity stays roughly constant as expected for the velocity of a circular orbit. The gradient of the actual solution produced by the model was found to be 1.988e-08 ms^{-2}. This is small enough to be roughly the same as the gradient produced by the analytic solution for velocity which is zero. So it was found that the magnitude of the velocity is as expected from the equations of motion.

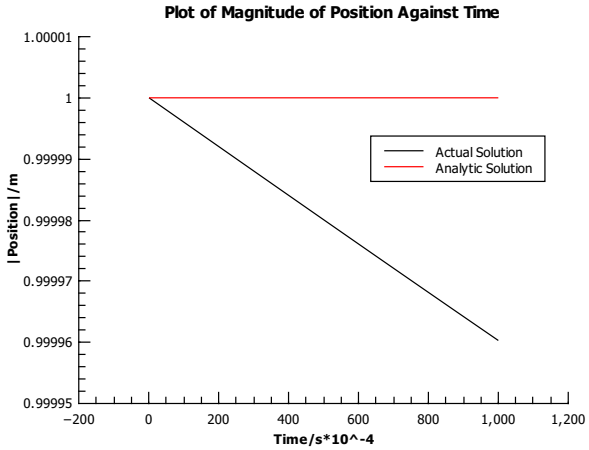

Figure 5: Plot of the magnitude of the position of one mass against intervals

From figure 5 the magnitude of the position is constant for a circular orbit as was intended when the two mass system was set up. The gradient of the actual position calculated by the simulation is -3.976e-08ms^{-1}. Which is small enough to negligible and close enough to 0, the gradient of the analytic solution. Hence it can be said the gradient is as expected.

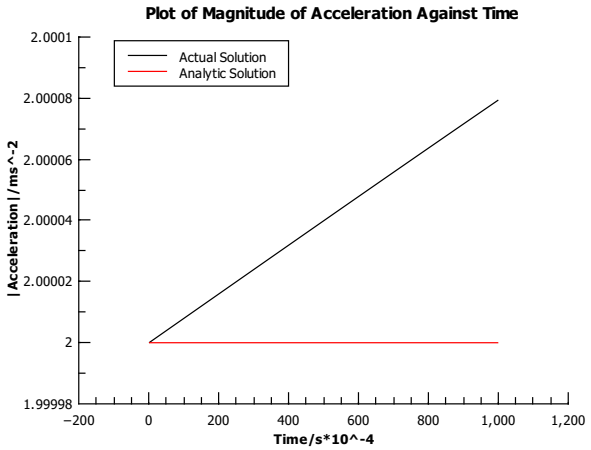

Figure 6: Plot of the magnitude of the acceleration of one mass against intervals

From figure 6 the magnitude of the acceleration is constant for a circular orbit as was intended when the two mass system was set up. The gradient of the actual acceleration calculated by the simulation is 7.952e-08ms^{-3}, which is small enough to negligible and close enough to 0, the gradient of the analytic solution. Hence we can say the gradient is as expected.

5 Analysis

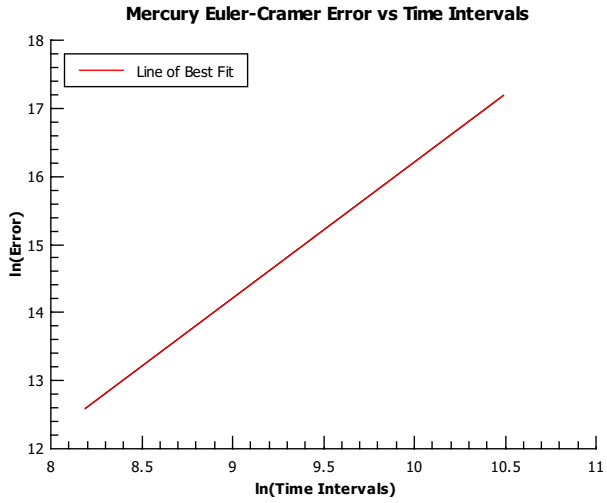

Figure 7: Graph showing difference between Mercury's expected and simulated position as a function of time intervals

Figure 7 was plotted by finding the difference between the expected analytic x - position of Mercury from the NASA horizons database[2] and comparing it with the value from the simulation. This was tried for ten time intervals and a graph plotted of the natural log of both, the gradient was found. This is suggestive that Error α $(\Delta t)^2$ from table 5.

Planet	Euler(ms^{-1})	Euler-Cramer(ms^{-1})
Mercury	2.002	1.997
Jupiter	1.978	2.029

Table 5: Table showing error for different planets using Euler and Euler-Cramer

A graph was plotted for the Euler method and the same gradient was found, suggesting that it too follows the Error α $(\Delta t)^2$ relationship. The calculation was repeated for the other extreme to a high velocity low mass body, a low velocity high mass body: Jupiter. For both the Euler and Euler-Cramer methods the same gradient was found see table 5.

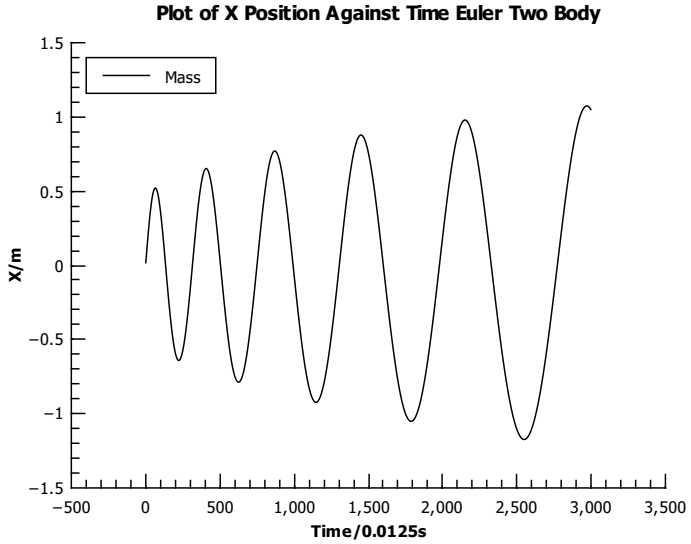

Figure 8: x-position for Euler algorithm for two mass system

The expected x-position should form a sine curve however from figure 8 the relationship is $x\ \alpha\ Aln(t)sin(\omega t)$. This indicates as there is an increase in amplitude due to energy entering the system. The symbols are defined as follows, $w[\theta/s]$ is the angular frequency, t is the time and A is a constant. This relationship will hold for radius and the positions of bodies in the solar system simulation as well. But it only happens at higher time intervals.

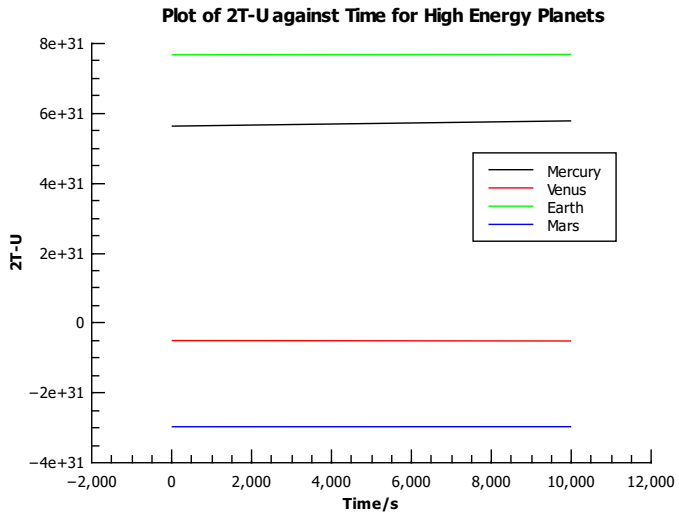

Figure 9: Low time interval energy conservation for Euler-Cramer

From equation 4 the plot of Viral theorem for a body the value should be roughly constant. Looking at figure 7 for the Euler-Cramer method the energy is roughly constant hence conserved, this means that over long periods the method will be fairly stable as no new energy is added, for small time intervals. The same result was found for the Euler method, as the simulation can give more accurate calculations of velocity and position at lower values of Δt.

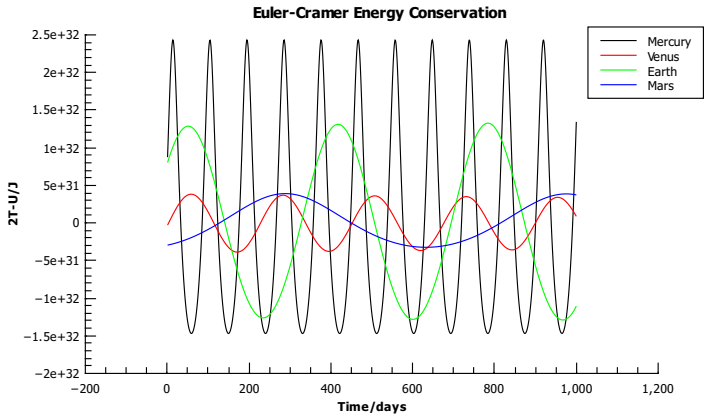

Figure 10: Graph showing energy oscillation for Euler-Cramer

For the inner planets the Virial theorem output is compromised of regular periodic functions from figure 10, for a large time interval of one day. This is due to the calculation of kinetic energy and potential energy no longer being smooth due to larger differences in position and velocity between calculations. The graph is effectively showing the interchange between kinetic and potential energy.

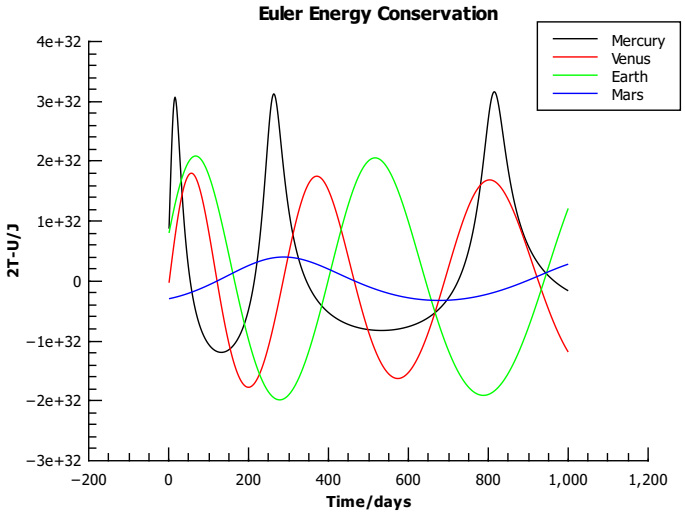

Figure 11: Virial theorem energy conservation for Euler method

Looking at figure 11 the period of interchange between kinetic energy and potential energy increases with time. This is in agreement with figure 8, as the positions of the same point at every orbit is getting further away from its true position. This is probably due the Euler algorithm adding energy to the system from its order of calculating position then velocity, which is effectively increasing the time period each orbit.

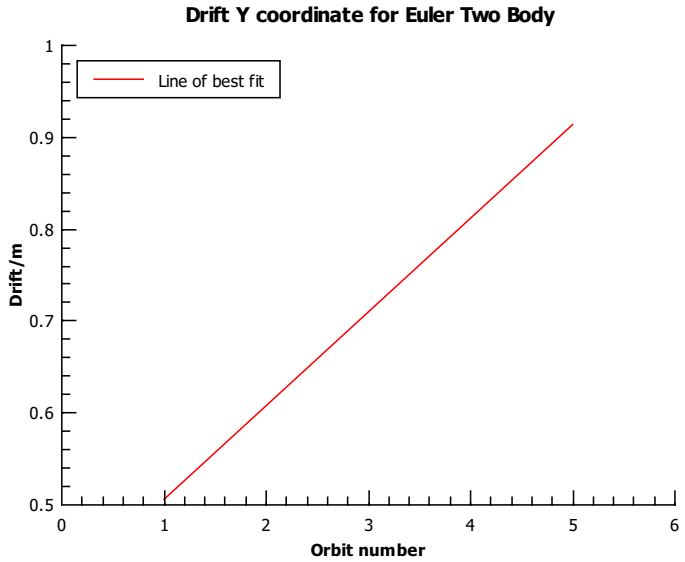

Figure 12: How far the Y coordinate of the orbit moves by each orbit

A graph was plotted to see at large time intervals (0.0125s in this case) how much the orbit moves by upon completion. The gradient was found to be 1.07E-01 $m/orbit$ which is how much the Euler method is off the Euler-Cramer by. The the Euler-Cramer is effectively

stable at 0.5m radius for a two body system. The bodies in the solar system are found to drift by 1.21E-01 $m/orbit$ from figure 13. Hence from equation 9 the time period would increase by approximately 1.31 times each orbit for bodies in the solar system.

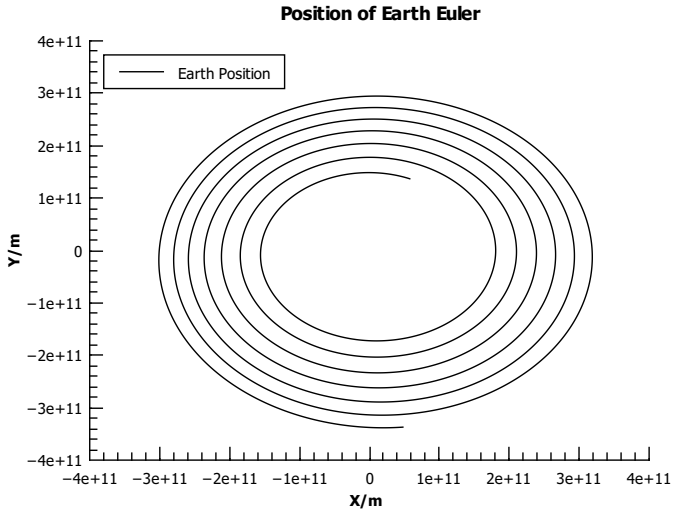

Figure 13: Effect of the Euler algorithm on the Earth's orbit for time interval of 1 day

6 Summary

A program in Java was written which modelled a system of particles generating their own gravitational fields and therefore moving according fields generated by other particles. It consisted of a Particle and and GravField class, representing a particle and gravitational field respectively. The Particle class implemented the Euler and Euler-Cramer algorithms to move the particles. The GravField class implemented Newton's law of gravitation and vector addition.

The simulation which was produced was shown to be working with a level of accuracy. This was done by comparing the simulated solutions to a two body problem with the analytic solutions which were calculated separately. The system consisted of two equal mass's orbiting equal distances away from a shared centre of mass.

The natural log of difference between simulated solution and assumed analytic solution against different time intervals was plotted and the gradient obtained. This was repeated for different algorithms and planets to see if it was valid for different conditions. Indeed, this was the case, all of the graph's had a gradient of $2ms^{-1}$. Which meant that the error was $\alpha\ (\Delta t)^2$ as predicted by theory in equation 5.

When the energy is plotted for the system at low time intervals for both Euler and Euler-Cramer conserve energy but as soon as the time interval increases the energy is not conserved

for the Euler algorithm. In fact the increase in radius at each orbit was found to follow the relationship $r \; \alpha \; Aln(t)sin(\omega t)$, which corresponds to roughly an increase in orbital period of 1.31 times each orbit. This corresponds to an input of energy into the system by the Euler algorithm. So the Euler-Cramer is a better solution to the n - body problem than the Euler algorithm as energy is roughly conserved.

The system simplified the solar system to just the first seven bodies so this may also account for errors in the simulation, as the bodies in the simulation do not feel the full gravitational field expected. Also the bodies in the system were simulated as point particles which is not true in reality, however served as a good approximation as the bodies were so far away from each other.

Future work could include implementation of the other main bodies in the solar system and other more accurate algorithms such as the Verlat algorithm. The effect of increased velocity on the accuracy of the simulation could be investigated ie. for small fast bodies like Mercury. Bodies could be given radii so to model collisions and combining of bodies to form larger bodies, if the bodies got too close to each other. Also perhaps finding the constant A for different oscillatory systems.

References

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

[2] Giorgini, J.D., Yeomans, D.K., Chamberlin, A.B., Chodas, P.W., Jacobson, R.A., Keesey, M.S., Lieske, J.H., Ostro, S.J., Standish, E.M., Wimberly, R.N., "JPL's On-Line Solar System Data Service", Bulletin of the American Astronomical Society, Vol 28, No. 3, p. 1158, 1996

[3] Swinburne University of Technology