N-body Simulation of the Solar System

6408056

Department of Physics, University of Surrey, Guildford, GU2 7XH, UK

Abstract

In this paper the solar system's constituents are simulated using 3 different numerical techniques, from the simulation the orbits of the planets are plotted and their eccentricity, surface temperature, orbital period and their average distance from the sun is calculated. The average distance between the earth and the sun is found to be within 0.03% of the true value, its orbital period within 0.14% and its eccentricity is found to be within 6.23% of the true value, at 0.018.

Introduction

The aim of this project was to simulate the solar system's constituents' positions and velocities over a period of time, to do this the n-body problem must be solved. Solving the n-body problem requires taking the instantaneous positions and velocities of the bodies concerned and using this to predict their future positions and motions by considering the accelerations due to gravity on each body from every other body (Trenti & Hut 2008). To do this Newton's law of gravity and multiple methods of numerical integration have been implemented and compared. The methods implemented are Euler's method, the Leapfrog method and Yoshida's algorithm, a higher order derivative of the Leapfrog method. Such a simulation has multiple applications in astronomy, and similar simulations have provided evidence that lead to the theory of dark matter (Koupelis, 2014).

Theoretical background

To simulate the planet's positions the acceleration caused by all other bodies must be considered, from this the change in the velocity of the planet, and its new position can be found. This can be completed by considering the following series (Heggie, 2005),

$$\frac{\mathrm{d}\mathbf{v}_\mathrm{i}}{\mathrm{dt}} = -\mathrm{G}\sum_{\mathrm{j=1,j\neq i}}^{\mathrm{j=N}} \frac{\mathrm{m_j}(\mathbf{r}_\mathrm{i}-\mathrm{r_j})}{|\mathbf{r}_\mathrm{i}-\mathrm{r_j}|^3} \ (1)$$

The details on the numerical methods, Euler's method, the Leapfrog method and Yoshida's algorithm, used to solve this can be found in appendix a, b and c respectively. The errors and the computation times for these methods were compared and the fourth order Yoshida method was chosen, the details of this comparison can be found in appendix d. In the Yoshida method the position is calculated 4 times, and the velocity and acceleration is calculated 3 times for one time step. This method is therefore more computationally intensive than the Euler and Leapfrog methods which only require one calculation of acceleration per time step, this method is however more accurate for larger time steps, allowing their use, which leads to a reduction in overall computational time for the same accuracy.

Once the position of the planets are known we are able to calculate the average distance between the planets and the sun by summing up the distances at the end of all the time steps and then dividing the sum by the number of time steps. With the average position we can then go on to predict the temperature of the planet's surface using the following equation,

$$T = \left(\frac{L(1-\alpha)}{4\pi\sigma r^2}\right)^{\frac{1}{4}} \ (2)$$

Here L is the bolometric luminosity of the sun, α is the albedo of the planet and r is the average distance between the sun and that planet. The orbit time of the planet can also be calculated from the average distance between it and the sun using the following equation,

$$t_{orbit} = 2\pi\sqrt{\frac{r^3}{GM_{sun}}} \ (3)$$

Here G is the gravitational constant, r is again the average distance between the sun and the planet, and M_{sun} is the mass of the sun. This equation gives the time of the orbit in seconds but is converted into years in the program to allow for easier interpretation of the results. We can then use the following equation for the eccentricity of the planets' orbit,

$$e = \frac{r_{max}-r_{min}}{r_{max}+r_{min}} \ (4)$$

Where $r_{max}$and $r_{min}$are the maximum and minimum separation of the planet and the sun respectively.

These equations have been implemented using the Fortran 90 language inside Microsoft Visual Studio using Intel Parallel Studio XE 2018 Cluster's Visual Fortran compiler.

Numerical Details

The initial positions and velocity of the planets have been inputted into the code in Cartesian heliocentric coordinates and were taken from the NASA's Horizons system, at 01/01/2000 00:00 (Chamberlin & Park, 2019). To test the energy conservation of the program the initial and final kinetic energy of the system was recorded and an increase of 2.5% was found over 10 years and a 200 second step.

As the initial conditions of the program were taken from an accurate simulation we can compare the predictions of our simulation against the results given by the same situation, in this case the X position given by the simulation at 01/01/2001 00:00 is compared to the value given by the horizon system. The program was run for 13 different step times from 1 to 120 seconds and the computation time and percentage error was recorded. A computation time of 3278.88±0.05 seconds and a percentage error of 9.003% was found for a step of one second, using Yoshida's method, this value is not displayed in figure 1 so as not to obscure the relationship for lower computation times. This test indicated that the best trade-off between computation time and error was found at a step of 40 seconds, with a computation time of 83.81±0.05 seconds and an error of 9.0155%.

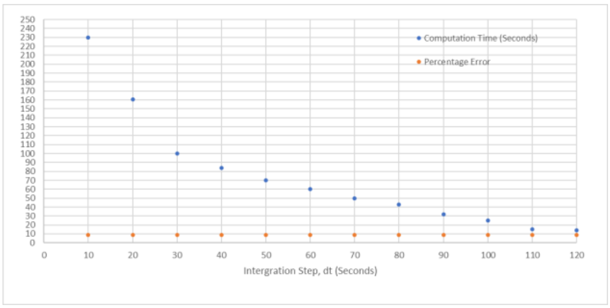


Figure 1: Graph of percentage error in Earth's X-position and computation time against time step for 1-year simulation time for the Yoshida Method (Chamberlin & Park, 2019)

Results

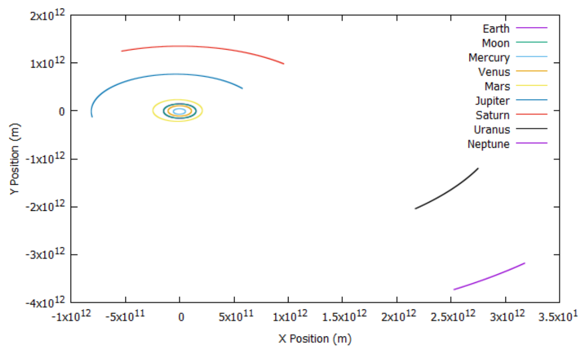


Figure 2: Plot of orbits for 5-year integration time, 500 second time step for Yoshida method

From figure 2 we can see the expected orbit shapes as well as the expected closed and open orbits given the five-year integration time.

Body	Expected Mean Distance (Au)	Mean Distance(AU)	Percentage Error
Mercury	0.387	0.395	1.95
Venus	0.723	0.723	0.03
Earth	1	1.000	0.03
Mars	1.524	1.529	0.34
Juipiter	5.203	5.225	0.43
Saturn	9.572	9.527	0.47
Uranus	20	19.501	2.49
Neptune	30.066	29.867	0.66

Figure 3 Mean distance to the sun calculated from a 200 year, 40 second step time of the Yoshida method against the literature value (Freedman & Kaufmann, 2005)

Body	Expected Eccentricity	Eccentricity	Percentage Error
Mercury	0.206	0.207	0.57
Venus	0.0068	0.007	7.97
Earth	0.017	0.018	6.23
Mars	0.093	0.095	1.87
Juipiter	0.048	0.050	3.90
Saturn	0.053	0.059	11.90
Uranus	0.0429	0.043	0.45
Neptune	0.01	0.013	29.75

Figure 4: Eccentricity of the orbits calculated from a 200 year, 40 second step time of the Yoshida method against the literature value (Freedman & Kaufmann, 2005)

Body	Expected Oribit Time (Years)	Orbit Time (Years)	Percentage Error
Mercury	0.24	0.248	3.17
Venus	0.615	0.615	0.06
Earth	1	1.001	0.14
Mars	1.88	1.893	0.71
Juipiter	11.86	11.899	0.33
Saturn	29.37	29.607	0.80
Uranus	84.099	83.924	0.21
Neptune	164.86	162.949	1.17

Figure 5: Orbital period of the planets to calculated from a 200 year, 40 second step time of the Yoshida method against the literature value (Freedman & Kaufmann, 2005)

From the above 3 figures we can see that the simulation tends to produce larger percentage errors for Mercury, Saturn, Uranus and Neptune, the errors are in general very small however, surprisingly so for a simulation that has only considered the gravitational force from a limited number of other bodies. What is clear from figure 4 is that something affecting Saturn's and Neptune's eccentricities has not been taken into account.

Summary & Conclusion

This report has described the applications of an n-body simulation and has successfully allowed for the prediction of the orbital periods, average distance from the sun, the eccentricity of the planets and the predicted surface temperature. To improve the program more numerical integration techniques, such as the Runge-Kutta and the velvet method would be considered, and their accuracies compared, more bodies within the solar system would be considered as well as other phenomenon such as radiation pressure and the rotation periods of the planets.

Appendix A

Euler method

The Euler method is a relatively simple method of numerical integration, it assumes a constant acceleration and velocity over the time step used. By adding up all the accelerations caused by other bodies, the target body's new position and velocity can be found using the following equations, (Heggie, 2005; Barker, 2017),

$$\mathbf{r}_{\mathrm{i+1}} = \mathbf{v}_{\mathrm{i}} \times \Delta \mathrm{t} + \mathbf{r}_{\mathrm{i}} \ (5)$$

$$\mathbf{v}_{\mathrm{i+1}} = \mathbf{a}(\mathbf{r}_{\mathrm{i}}) \times \Delta \mathrm{t} + \mathbf{v}_{\mathrm{i}} \ (6)$$

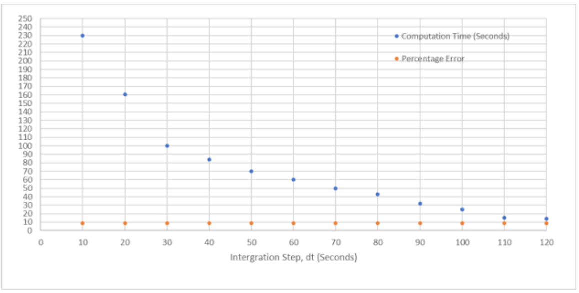


Figure 6: Graph of percentage error in Earth's X-position and computation time against time step for 1-year simulation time for the Euler method (Chamberlin & Park, 2019)

By completing these calculations over the required time, the positions of the bodies can be found, the resulting orbits from this method are shown below. The Euler method shown here has a local error that is proportional to the square of the step size.

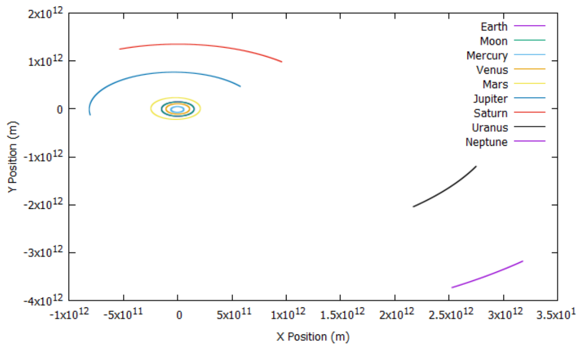


Figure 7: Plot of orbits for 5-year integration time, 500 second time step for Euler method.

The Euler method produces closed orbits, but the error quickly increases over time and mercury quickly deviates from its orbit.

Appendix B

Leapfrog Method

The leapfrog method is similar to the velocity Verlet method, it gets its name from the way it updates positions and velocities at interleaved time points following a similar pattern to two people leap frogging. The leapfrog method is second order and hence produces a smaller local error for the same time step when compared to the first order Euler method. This method can be implemented using the following equations,

$$\mathbf{r}_{\mathrm{i+1}} = \mathbf{r}_{\mathrm{i}} + \Delta \mathrm{t}(\boldsymbol{v}_i) + \frac{1}{2}\Delta t^2 \boldsymbol{a}(\mathbf{r}_{\mathrm{i}}) \ (7)$$

$$\mathbf{v}_{\mathrm{i+1}} = \boldsymbol{v}_i + \frac{1}{2}\Delta t(\boldsymbol{a}(\mathbf{r}_{\mathrm{i}}) + \boldsymbol{a}(\mathbf{r}_{\mathrm{i+1}})) \ (8)$$

Here we can see that this method is calculating the average acceleration of a time step before calculating the velocity and taking into account the acceleration when considering the change in position of a body. This is a large step up from the Euler method and one that is simple to implement.

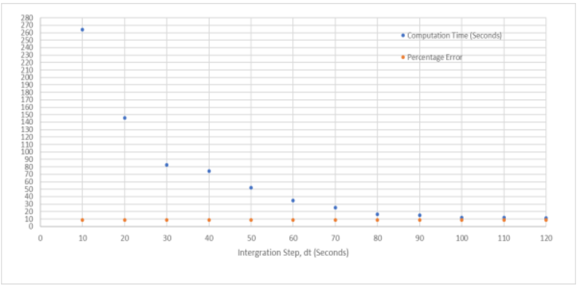


Figure 8: Graph of percentage error in Earth's X-position and computation time against time step for 1-year simulation time for the Leapfrog method (Chamberlin & Park, 2019)

We can also see from figure 6 and figure 8 that the Leapfrog method does not lead to a large increase in the computation time, despite having to calculate the acceleration twice.

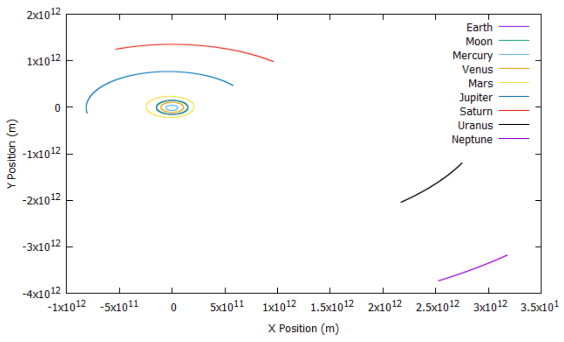


Figure 9: Plot of orbits for 5-year integration time, 500 second time step for Leapfrog method.

The leapfrog method, like the Euler method produces closed orbits but less deviation in Mercury's path.

Appendix C

Yoshida Method

The chosen method of numerical integration in the paper was the 4th order Yoshida integrator, in this method the position is calculated 4 times, and the velocity and acceleration is calculated 3 times for one time step. This method is therefore more computationally intensive as the Euler and leap frog methods only require one calculation of acceleration per time step, this method is however more accurate for larger time steps which leads to a reduction in overall computational time for the same accuracy. The Yoshida method benefits by being a higher order integrator than the previous methods mentioned, it is also, like the leapfrog method symplectic in nature meaning it is good for long time integrations on Hamiltonian systems, like the n body simulation presented here (Yoshida, 1990).

To use the Yoshida method the following equations are integrated into the program (Yoshida, 1990),

$$\mathbf{r_i^1} = \mathbf{r_i} + c_1\mathbf{v_i}\Delta t \ (9)$$

$$\mathbf{v_i^1} = \mathbf{v_i} + d_1\mathbf{a}(\mathbf{r_i^1})\Delta t \ (10)$$

$$\mathbf{r_i^2} = \mathbf{r_i^1} + c_2\mathbf{v_i^1}\Delta t \ (11)$$

$$\mathbf{v_i^2} = \mathbf{v_i^1} + d_2\mathbf{a}(\mathbf{r_i^2})\Delta t \ (12)$$

$$\mathbf{r_i^3} = \mathbf{r_i^2} + c_2\mathbf{v_i^2}\Delta t \ (13)$$

$$\mathbf{v_i^3} = \mathbf{v_i^2} + d_2\mathbf{a}(\mathbf{r_i^3})\Delta t \ (14)$$

$$\mathbf{r_{i+1}} = \mathbf{r_i^3} + c_4\mathbf{v_i^3}\Delta t \ (15)$$

$$\mathbf{v_{i+1}} = \mathbf{v_i^3} \ (16)$$

Where,

$$\omega_0 = -\frac{2^{\frac{1}{3}}}{2-2^{\frac{1}{3}}} \ (17)$$

$$\omega_1 = -\frac{1}{2-2^{\frac{1}{3}}} \ (18)$$

$$\Sigma_{i=1}^{4} c_i = 1 \ (19)$$

$$\Sigma_{i=1}^{3} d_i = 1 \ (20)$$

$$c_1 = c_4 = \frac{\omega_1}{2} \ (21)$$

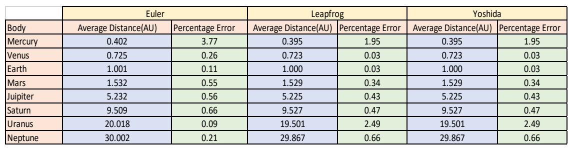
$$c_2 = c_3 = \frac{\omega_0+\omega_1}{2} \ (23)$$

$$d_1 = d_3 = \omega_1 \ (24)$$

$$d_2 = \omega_0 \ (25)$$

Appendix D

A comparison



	Euler		Leapfrog		Yoshida	
Body	Average Distance(AU)	Percentage Error	Average Distance(AU)	Percentage Error	Average Distance(AU)	Percentage Error
Mercury	0.402	3.77	0.395	1.95	0.395	1.95
Venus	0.725	0.26	0.723	0.03	0.723	0.03
Earth	1.001	0.11	1.000	0.03	1.000	0.03
Mars	1.532	0.55	1.529	0.34	1.529	0.34
Juipiter	5.232	0.56	5.225	0.43	5.225	0.43
Saturn	9.509	0.66	9.527	0.47	9.527	0.47
Uranus	20.018	0.09	19.501	2.49	19.501	2.49
Neptune	30.002	0.21	29.867	0.66	29.867	0.66

Figure 10: Comparison of the different integration techniques used and their respective errors

Figure 10 gives us an idea of the errors produced by the different integration techniques, we can see that at this number of decimal places there isn't much of a difference between the Leapfrog method and the Yoshida method, but that is because the number of significant figures present is obscuring that the error in the Yoshida method is ever so slightly smaller. This indicates that over short periods of time and with smaller time steps the Leapfrog method is suitable for all but the most accurate procedures.

Appendix E

Gnuplot command to plot a 2d plot of all the planets is as follows,

```
p 'Earth.dat' u 1:2 title "Earth" w l, \
'Moon.dat' u 1:2 title "Moon" w l, \
'Mercury.dat' u 1:2 title "Mercury" w l, \
'Venus.dat' u 1:2 title "Venus" w l, \
'Mars.dat' u 1:2 title "Mars" w l, \
'Jupiter.dat' u 1:2 title "Jupiter" w l, \
'Saturn.dat' u 1:2 title "Saturn" w l, \
'Uranus.dat' u 1:2 title "Uranus" w l, \
'Neptune.dat' u 1:2 title "Neptune" w l
```

References

Trenti, M. & Hut, P. 2008. N-body simulations (gravitational). [Online]. [17 March 2019]. Available from: http://www.scholarpedia.org/article/N-body_simulations_(gravitational)

Koupelis, T (2014). In Quest of The Universe. (7th ed.). Burlington, MA: Jones & Bartlett Learning.

Chamberlin, A.B. & Park, R.S. (2019). Horizons Web Interface. [Online]. [1st March 2019]. Available from: https://ssd.jpl.nasa.gov/?horizons

Heggie, D.C (2005). The Classical Gravitational N-Body Problem. Edinburgh: University of Edinburgh.

Freedman, R.A & Kaufmann, W.J (2005). Universe. (7th ed.). New York: W H Freeman and Company.

Barker, C.A. 2017. Euler's Method Theoretical Introduction. [Online]. [17 March 2019]. Available from: Numerical Methods for solving Differential Equations Euler’s Method

Skeel, R.D. 1993. Variable Step Size Destabilizes the Stömer/Leapfrog/Verlet Method. BIT Numerical Mathematics. 33(1), pp. 172–175

Yoshida, H. 1990. Construction of higher order symplectic integrators. Physics Letters. 150 (5, 6, 7).

Code

```
PROGRAM CelestialMech

IMPLICIT NONE


INTEGER:: i, k, t

DOUBLE PRECISION:: time, G, dt, L, o, pi, Au, d1, d2, d3, c1, c2, c3, c4

DOUBLE PRECISION, DIMENSION(3):: Vector, dvdt, dvdt2

DOUBLE PRECISION, DIMENSION(10):: Mass, Albedo, temp, OrbitTime, X1, Y1, Z1, X2, Y2, Z2, Vx, Vy, Vz, r,
avgr, rmin, rmax, e

CHARACTER(len=5), DIMENSION(10)::Body=(/'Sun', 'Earth', 'Moon', 'Mer', 'Venus', 'Mars', 'Jup', 'Sat', 'Ura', 'Nep'/)

CHARACTER(LEN=10) :: the_time

CHARACTER(LEN=1):: choice

G = 6.67408E-11; L=3.828E26; o=5.6705E-8; pi=4.0*atan(1.0); Au = 149598000E3

d1=1.351207192; d3=d1; d2=-1.702414384; c1=0.675603596; c4=c1; c2=-0.175603596; c3=c2; rmax=0.0;
rmin=9E9999

!Defining constants for Yoshida integration, to calculate acceleration and temperature


OPEN(12, file='ExtraData.dat'); OPEN(13, file='IntialConditions.dat')

OPEN(14, file='Earth.dat');OPEN(15, file='Moon.dat');OPEN(16, file='Mercury.dat')

OPEN(17, file='Venus.dat');OPEN(18, file='Mars.dat');OPEN(19, file='Jupiter.dat')

OPEN(20, file='Saturn.dat'); OPEN(21, file='Uranus.dat');OPEN(22, file='Neptune.dat')

!Gives every planet their own file, making plotting, finding the data and formatting much easier

READ(13,*) Albedo, Mass, X1, Y1, Z1, Vx, Vy, Vz
```

```
!Reads initial conditions
CLOSE(13)

WRITE(6,*) 'This program is designed to calculate the positions of the planets'
WRITE(6,*) 'solar systems planets using newtons law of gravity'
WRITE(6,*) 'and various numerical integration techniques, times, and time steps'
WRITE(6,*) 'Would you like to choose a time step, integration time,'
WRITE(6,*) 'and numerical technique or use the default settings?'
WRITE(6,*) 'The default settings are 1 year, 40 seconds and the Yoshida Method'
WRITE(6,*) 'Press y to enter your own settings, n for default.'
READ(5,*) Choice



IF (choice =='y' .or. choice =='Y') THEN

!Gives user a way to change settings without changing the code directly
!Which can be easier in some cases

  WRITE(6,*) 'Please enter the numerical integration technique you would like to use'
  WRITE(6,*) 'Y for Yoshida, L for leapfrog, E for Euler'
  READ(5,*) Choice
  WRITE(6,*) 'Please enter the integration time.'
  READ(5,*) time
  WRITE(6,*) 'Please enter the time step'
  READ(5,*) dt

ELSE IF (choice =='n'.or. choice =='N') THEN
!This choice still allows the option to change the code directly
 dt=40; time=31536000; choice='y'

END IF

WRITE(6,*) 'Coordinates are printed to files labeled with the planets name, such'
WRITE(6,*) 'as "Earth.dat", the coordinates are printed, X, Y, Z for 3d plotting'
WRITE(6,*) 'For other data such as orbit time and temperature please see the file'
WRITE(6,*) '"ExtraData.dat"'
```

```
!Tells user where to find the data generated


CALL DATE_AND_TIME(TIME=the_time)
WRITE(12,*) 'Start time: ', the_time
!Records the time the calculation started for later use


X2=X1; Y2=Y1; Z2=Z1




IF (choice =='y'.or. choice =='Y') THEN
  !Yoshida Integration, if loop allows for choice and allows quick testing of all
  !integration methods without having to change the code


  DO t = 0, NINT(time/dt)
    !Do loop over all time for the required number of time steps
    WRITE(14, '(e13.5,e13.5,e13.5)') X2(2), Y2(2), Z2(2);  WRITE(15, '(e13.5,e13.5,e13.5)') X2(3), Y2(3), Z2(3)
    WRITE(16, '(e13.5,e13.5,e13.5)') X2(4), Y2(4), Z2(4);  WRITE(17, '(e13.5,e13.5,e13.5)') X2(5), Y2(5), Z2(5)
    WRITE(18, '(e13.5,e13.5,e13.5)') X2(6), Y2(6), Z2(6);  WRITE(19, '(e13.5,e13.5,e13.5)') X2(7), Y2(7), Z2(7)
    WRITE(20, '(e13.5,e13.5,e13.5)') X2(8), Y2(8), Z2(8);  WRITE(21, '(e13.5,e13.5,e13.5)') X2(9), Y2(9), Z2(9)
    WRITE(22, '(e13.5,e13.5,e13.5)') X2(10), Y2(10), Z2(10)
    !Prints X & Y coordinates to files


    X1=X2; Y1=Y2; Z1=Z2
    !Defines new positions X1 as the final positions of last time step, X2


    DO k=2,10 !k= planet force acting on, i= planet causing force.


      dvdt=0.0 !Resets acceleration for next planet
      X2(k)=X1(k)+c1*Vx(K)*dt; Y2(k)=Y1(k)+c1*Vy(k)*dt; z2(k)=z1(k)+c1*Vz(k)*dt
      !X2 is the variable point, and X1 is the initial position of the planets at the
      !start of the time step


      DO i=1,10  !calculating acceleration, dvdt, at new point
        !Do loop sums over different planets to add their contributions to the acceleration
        IF (i /= k)Then !Avoids calculating acceleration of a planet on itself
          r(k) = sqrt((X2(k)-X1(i))**2+(Y2(k)-Y1(i))**2+(Z2(k)-Z1(i))**2)
          !calculating 'r' from planets new position and other planets initial positions
          !This means that the acceleration for the last planet will still be considering
          !The other planets positions as the start of the time step
          Vector=(/(X2(k)-X1(i)), (Y2(k)-Y1(i)), (Z2(k)-Z1(i))/)/r(k)  !unit Vector for direction of force
          dVdt=-(G*Mass(i)/r(k)**2)*vector+dVdt !Newtons Law  to calculate acceleration due to gravity
        END IF
      END DO

      Vx(k)=Vx(k)+d1*dvdt(1)*dt; Vy(k)=Vy(k)+d1*dvdt(2)*dt; Vz(k)=Vz(k)+d1*dvdt(3)*dt
      X2(k)=X2(k)+c2*Vx(k)*dt;  Y2(k)=Y2(k)+c2*Vy(k)*dt; Z2(k)=Z2(k)+c2*Vz(k)*dt;
      dvdt=0.0 !Acceleration reset in between as we are now considering it at a new point

      DO i=1,10   !calculating acceleration, dvdt, at new point
        IF (i /= k)Then
          r(k) = sqrt((X2(k)-X1(i))**2+(Y2(k)-Y1(i))**2+(Z2(k)-Z1(i))**2)
          Vector=(/(X2(k)-X1(i)), (Y2(k)-Y1(i)), (Z2(k)-Z1(i))/)/r(k)
          dVdt=-(G*Mass(i)/r(k)**2)*vector+dVdt
        END IF
      END DO

      !Yoshida integration step
      Vx(k)=Vx(k)+d2*dvdt(1)*dt; Vy(k)=Vy(k)+d2*dvdt(2)*dt; Vz(k)=Vz(k)+d2*dvdt(3)*dt
      X2(k)=X2(k)+c3*Vx(k)*dt;  y2(k)=y2(k)+c3*Vy(k)*dt;  z2(k)=z2(k)+c3*Vz(k)*dt
      dvdt=0.0  !Acceleration reset in between as we are now considering it at a new point

      DO i=1,10   !calculating acceleration, dvdt,  at new point
        IF (i /= k)Then
          r(k) = sqrt((X2(k)-X1(i))**2+(Y2(k)-Y1(i))**2+(Z2(k)-Z1(i))**2)
          Vector=(/(X2(k)-X1(i)), (Y2(k)-Y1(i)), (Z2(k)-Z1(i))/)/r(k)
          dVdt=-(G*Mass(i)/r(k)**2)*vector+dVdt
        END IF
      END DO

      !Yoshida integration step
      Vx(k)=Vx(k)+d3*dvdt(1)*dt; Vy(k)=Vy(k)+d3*dvdt(2)*dt; Vz(k)=Vz(k)+d3*dvdt(3)*dt
      x2(k)=x2(k)+c4*vx(k)*dt; y2(k)=y2(k)+c4*vy(k)*dt; z2(k)=z2(k)+c4*vz(k)*dt
```

```
      dvdt=0.0 !Acceleration reset in between as we are now considering it at a new point

      avgr(k)=sqrt((X2(k)-X1(1))**2+(Y2(k)-Y1(1))**2+(Z2(k)-Z1(1))**2)+avgr(k)
      !Summing up total distance between the planet and the sun to calculate the mean late

      r(k) = sqrt((X2(k)-X1(1))**2+(Y2(k)-Y1(1))**2+(Z2(k)-Z1(1))**2)
      !Considers new distance (r) between planet and sun to calculate eccentricity

      IF (r(k) < rmin(k)) THEN
        !If r is smaller than the smallest r so far it is stored and used to calculate eccentricity
        rmin(k)=r(k)
      ELSE IF (r(k) > rmax(k)) THEN
        rmax(k)=r(k)
        !If r is larger than the largest so far it is stored and used to calculate eccentricity
      END IF

    END DO

  END DO

ELSE IF (choice == 'l'.or. choice =='L') THEN !Leapfrog method

  DO t = 0, NINT(time/dt)

    !Do loop over all time for the required number of time steps
    WRITE(14, '(e13.5,e13.5,e13.5)') X2(2), Y2(2), Z2(2);  WRITE(15, '(e13.5,e13.5,e13.5)') X2(3), Y2(3), Z2(3)
    WRITE(16, '(e13.5,e13.5,e13.5)') X2(4), Y2(4), Z2(4);  WRITE(17, '(e13.5,e13.5,e13.5)') X2(5), Y2(5), Z2(5)
    WRITE(18, '(e13.5,e13.5,e13.5)') X2(6), Y2(6), Z2(6);  WRITE(19, '(e13.5,e13.5,e13.5)') X2(7), Y2(7), Z2(7)
    WRITE(20, '(e13.5,e13.5,e13.5)') X2(8), Y2(8), Z2(8);  WRITE(21, '(e13.5,e13.5,e13.5)') X2(9), Y2(9), Z2(9)
    WRITE(22, '(e13.5,e13.5,e13.5)') X2(10), Y2(10), Z2(10)
    !Prints X & Y coordinates to files

    X1=X2; Y1=Y2; Z1=Z2
    !Defines new positions X1 as the final positions of last time step, X2

    DO k=2,10 !k= planet force acting on, i= planet causing force.
```

```
dvdt=0.0; dvdt2=0.0 !Resets acceleration for when a new planet is considered

DO i=1,10  !calculating acceleration of intial point

  IF (i /= k) THEN !if command avoids calculating acceleration of a planet on itself
    r(k) = sqrt((X2(k)-X1(i))**2+(Y2(k)-Y1(i))**2+(Z2(k)-Z1(i))**2)
    !calculating 'r' from intial positions of time step
    Vector=(/(X2(k)-X1(i)), (Y2(k)-Y1(i)), (Z2(k)-Z1(i))/)/r(k)  !unit Vector for direction of force
    dVdt=-(G*Mass(i)/r(k)**2)*vector+dVdt !Sums up all contributions to accelerations
  END IF

END DO

X2(k)=X2(K)+dt*Vx(k)+0.5*(dt**2)*dvdt(1); Y2(k)=Y2(K)+dt*Vy(k)+0.5*(dt**2)*dvdt(2)
Z2(k)=Z2(K)+dt*Vz(k)+0.5*(dt**2)*dvdt(3)
!Calculating  new positions

DO i=1,10  !calculating acceleration from new positions, dvdt2

  IF (i /= k) THEN
    r(k) = sqrt((X2(k)-X1(i))**2+(Y2(k)-Y1(i))**2+(Z2(k)-Z1(i))**2)
     !calculating 'r' from intial positions of time step
    Vector=(/(X2(k)-X1(i)), (Y2(k)-Y1(i)), (Z2(k)-Z1(i))/)/r(k)  !unit Vector for direction of force
    dVdt2=-(G*Mass(i)/r(k)**2)*vector+dVdt2
  END IF

END DO

Vx(k)=Vx(k)+0.5*dt*(dvdt(1)+dvdt2(1)); Vy(k)=Vy(k)+0.5*dt*(dvdt(2)+dvdt2(2))
Vz(k)=Vz(k)+0.5*dt*(dvdt(3)+dvdt2(3))
!New velocity calculated using the average acceleration between intial and final positions

avgr(k)=sqrt((X2(k)-X1(1))**2+(Y2(k)-Y1(1))**2+(Z2(k)-Z1(1))**2)+avgr(k)
!Summing up total distance between the planet and the sun to calculate the mean late

r(k) = sqrt((X2(k)-X1(1))**2+(Y2(k)-Y1(1))**2+(Z2(k)-Z1(1))**2)
```

```
      !Considers new distance (r) between planet and sun to calculate eccentricity

      IF (r(k) < rmin(k)) THEN
        !If r is smaller than the smallest r so far it is stored and used to calculate eccentricity
        rmin(k)=r(k)
      ELSE IF (r(k) > rmax(k)) THEN
        rmax(k)=r(k)
        !If r is larger than the largest so far it is stored and used to calculate eccentricity
      END IF

    END DO

  END DO

ELSE IF (choice == 'e'.or. choice =='E') THEN !Euler Method

  DO t = 0, NINT(time/dt)

    !Do loop over all time for the required number of time steps
    WRITE(14, '(e13.5,e13.5,e13.5)') X2(2), Y2(2), Z2(2);  WRITE(15, '(e13.5,e13.5,e13.5)') X2(3), Y2(3), Z2(3)
    WRITE(16, '(e13.5,e13.5,e13.5)') X2(4), Y2(4), Z2(4);  WRITE(17, '(e13.5,e13.5,e13.5)') X2(5), Y2(5), Z2(5)
    WRITE(18, '(e13.5,e13.5,e13.5)') X2(6), Y2(6), Z2(6);  WRITE(19, '(e13.5,e13.5,e13.5)') X2(7), Y2(7), Z2(7)
    WRITE(20, '(e13.5,e13.5,e13.5)') X2(8), Y2(8), Z2(8);  WRITE(21, '(e13.5,e13.5,e13.5)') X2(9), Y2(9), Z2(9)
    WRITE(22, '(e13.5,e13.5,e13.5)') X2(10), Y2(10), Z2(10)
    !Prints X & Y coordinates to files

    X1=X2; Y1=Y2; Z1=Z2

    DO k=2,10 !k= planet force acting on, i= planet causing force.
      dvdt=0

      DO i=1,10  !calculating acceleration

        IF (i /= k)Then !Avoids calculating acceleration of a planet on itself
          r(k) = sqrt((X2(k)-X1(i))**2+(Y2(k)-Y1(i))**2+(Z2(k)-Z1(i))**2)
          !calculating 'r' from intial positions of time step
          Vector=(/(X2(k)-X1(i)), (Y2(k)-Y1(i)), (Z2(k)-Z1(i))/)/r(k)  !unit Vector for direction of force
          dVdt=-(G*Mass(i)/r(k)**2)*vector+dVdt !Newtons law of gravitation
        END IF

      END DO

      X2(k) = Vx(k)*dt + X2(k); Y2(k) = Vy(k)*dt + Y2(k); Z2(k) = Vz(k)*dt + Z2(k)
      !Assume velocity is constant over time step and use this to calculate new positions
      Vx(K) = dVdt(1)*dt + Vx(k); Vy(k) = dVdt(2)*dt + Vy(k); Vz(k) = dVdt(3)*dt + Vz(k)
      !Assume acceleration is constant over time step and use this to calculate new velocities

      avgr(k)=sqrt((X2(k)-X1(1))**2+(Y2(k)-Y1(1))**2+(Z2(k)-Z1(1))**2)+avgr(k)
      !Summing up total distance between the planet and the sun to calculate the mean late

      r(k) = sqrt((X2(k)-X1(1))**2+(Y2(k)-Y1(1))**2+(Z2(k)-Z1(1))**2)
      !Considers new distance (r) between planet and sun to calculate eccentricity

      IF (r(k) < rmin(k)) THEN
        !If r is smaller than the smallest r so far it is stored and used to calculate eccentricity
        rmin(k)=r(k)
      ELSE IF (r(k) > rmax(k)) THEN
        rmax(k)=r(k)
        !If r is larger than the largest so far it is stored and used to calculate eccentricity
      END IF

    END DO

  END DO

END IF

avgr = avgr/NINT(time/dt)

Temp = ((L*(1-albedo))/(4.0*pi*o*(avgr)**2))**0.25 -273.15
!Calculating surface temp. from flux, albedo, and average distance of planet to sun
OrbitTime = (2*pi*((avgr**3)/(G*Mass(1)))**0.5)/31536000
```

```
!Calculating orbit time from average distance of planet to sun and the mass of the sun, in years
e=(rmax-rmin)/(rmax+rmin)
!Calculating eccentricity of orbit

WRITE(12,*) 'Body, Avg. dist. to sun, Orbital Period, Temp, eccentricity'
!Printing titles to make data easier to interpret.

DO k = 2, 10

  WRITE(12,'(a5, f6.3, f8.3, f9.3, f8.4)') Body(k), avgr(k)/au, orbitTime(k), TEMP(K), e(k)
  !Printing data to file

END DO

WRITE(12,'(a10, f6.1)') 'Step time:', dt
WRITE(12,'(a17, f13.2)') 'Integration time:', time
WRITE(12,*) 'Number of steps:', NINT(time/dt)
WRITE(12,'(a11, a2)') 'Method used', Choice
CALL DATE_AND_TIME(TIME=the_time)
WRITE(12,*) 'Final time: ', the_time
!Recording all the final data in a file

CLOSE(12); CLOSE(14); CLOSE(15); CLOSE(16); CLOSE(17); CLOSE(18); CLOSE(19); CLOSE(20)
CLOSE(21); CLOSE(22)

END PROGRAM
```