

School of Electrical and Electronic Engineering

**EEEN 30150**

**Modelling and Simulation**

Major Project 2

|  |  |
| --- | --- |
| Student Name: | Paul Doherty |
| Student Number: | 10387129 |
| Date: | 13/05/2013 |
| Project Title: | 2-Body Problem |

# Introduction

The following report is an account of how my teammates and I solved the two – body problem. The problem required us to set up the equations of motion for the planet earth taking into account the gravitational effects of the sun and the planet Jupiter. These equations then had to be solved by a numerical method presented to us in the module. The solution of the problem then had to be visualised by creating an animation.

The work load was split evenly, Andy Connell and me focussed on the development and the presentation of the analysis and on the numerical solution of the differential equations while Scott Condron focussed on the visualisation of our solution. Although we had assigned each other with these roles there was constant interaction between the three of us during each stage of the assignment.

# Modelling the System

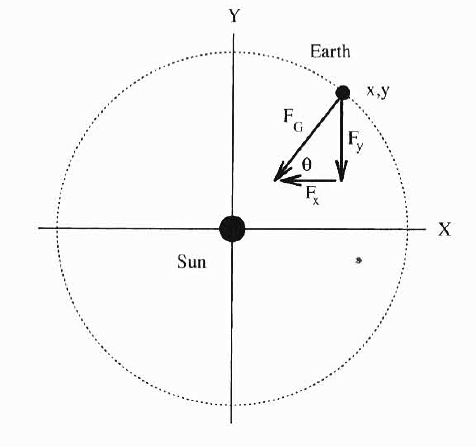
Analysing earth and Jupiter’s orbit was done by using Newton’s Law of Gravitation. Which states that every point mass in the universe attracts every point mass with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them:

Where is the gravitational constant.

This equation allows us to separately analyse the positions of earth and Jupiter with respect to the sun. The assumption was made that the suns mass is sufficiently large so that its motion can be neglected. We then considered Newton’s second law of motion which states that the acceleration of a body is directly proportional, and in the same direction as, the net force acting on the body, and inversely proportional to its mass:

Using this with respect to the problem at hand we get:

Taking that the sun will be centred on the origin, the below diagram shows the system for the earth and the sun, the same analysis is used for Jupiter and the sun.



Using analysis from the diagram in conjunction with Newton’s gravitational and second law we get:

and :

and represent the distance between the sun and the respected planet (earth or Jupiter in this case) for their respected coordinates.

The force equations for the coordinates of Jupiter will be the same as above except with the mass of the sun replaced with the mass of Jupiter . The minus signs indicate that the force is pointing towards the sun.

For the above system of second-order differential equations to be solved in MATLAB using a numerical method it must be split into two first-order differential equations:

## Units

Because of the scale of this problem we decided that SI units would not match and that it would make more sense to analyse the problem in astronomical units (AU). 1 AU is defined as the average distance between the Sun and Earth. Time will be measured in years. This choice of units requires a corresponding change in the units of mass. This can be derived from the Earth’s orbit which; is approximately circular. We look at the corresponding force of circular motion:

Using the fact that the velocity of earth is (AU/year) since AU we get:

Substituting, we can now model the 2-body problem as follows:

These equations can now be solved for the cases of the earth and Jupiter to show their orbit around the sun.

# Numerical Solution of ODE’s

We decided to solve the ODE’s using the Euler method:

## Initial Conditions

Before we could start to implement the Euler method the initial conditions for the positions of the sun to the Earth/Jupiter and the initial velocities of Earth/Jupiter needed to be set. These initial conditions were researched online and the values we chose are:

Initial position of Earth

Initial position of Earth

Initial position of Jupiter

Initial position of Jupiter

Initial velocity of Earth

Recall: AU/yr

Initial velocity of Earth

Initial velocity of Jupiter

Initial velocity of Jupiter

## Time Step and Iterations

## 

The problem asks us to run our solution for at least three Jovian years. One Jovian year is defined as the length of time it takes the Jupiter to complete one full orbit of the sun and is approximately equal to earth years. Choosing the time step to be years this means that the Euler Method needs to be iterated for each Jovian year.

## Coding the Solution

Now that the formulas to model the system have been derived, the initial conditions set and the time-step realised we can finally use the Euler-Method to solve the problem. It was decided that the solution of the problem would take the form of a function M-file, where the functions’ inputs would decide how many Jovian years the program would run for and how fast the animation would display the solutions. The MATLAB code is for the numerical solution is as follows, simulation and function not included:

npoints=k\*118500;

dt = 0.0001; % time step in years.

x\_e\_initial=1; % Initial position of Earth in AU

y\_e\_initial=0;

v\_e\_x\_initial=0; % Initial velocity of Earth in AU/yr

v\_e\_y\_initial=2\*pi;

x\_j\_initial=5.2; % Initial position of Jupiter in AU, assume at opposition initially

y\_j\_initial=0;

v\_j\_x\_initial=0; % Initial velocity of Jupiter in AU/yr

v\_j\_y\_initial= 2.7549; % This is 2\*pi\*5.2 AU/11.85 years = 2.75 AU/year

% Create arrays to store position and velocity of Earth

x\_e=zeros(npoints,1);

y\_e=zeros(npoints,1);

v\_e\_x=zeros(npoints,1);

v\_e\_y=zeros(npoints,1);

% Create arrays to store position and velocity of Jupiter

x\_j=zeros(npoints,1);

y\_j=zeros(npoints,1);

v\_j\_x=zeros(npoints,1);

v\_j\_y=zeros(npoints,1);

r\_e=zeros(npoints,1);

r\_j=zeros(npoints,1);

% Initialise positions and velocities of Earth and Jupiter

x\_e(1)=x\_e\_initial;

y\_e(1)=y\_e\_initial;

v\_e\_x(1)=v\_e\_x\_initial;

v\_e\_y(1)=v\_e\_y\_initial;

x\_j(1)=x\_j\_initial;

y\_j(1)=y\_j\_initial;

v\_j\_x(1)=v\_j\_x\_initial;

v\_j\_y(1)=v\_j\_y\_initial;

for i = 1:npoints-1; % loop over the timesteps

% Calculate distances to Earth from Sun, Jupiter from Sun and Jupiter

% to Earth for current value of i

r\_e(i)=sqrt(x\_e(i)^2+y\_e(i)^2);

r\_j(i)=sqrt(x\_j(i)^2+y\_j(i)^2);

% Compute x and y components for new velocity of Earth

v\_e\_x(i+1)=v\_e\_x(i)-4\*pi^2\*x\_e(i)\*dt/r\_e(i)^3;

v\_e\_y(i+1)=v\_e\_y(i)-4\*pi^2\*y\_e(i)\*dt/r\_e(i)^3;

% Compute x and y components for new velocity of Jupiter

v\_j\_x(i+1)=v\_j\_x(i)-4\*pi^2\*x\_j(i)\*dt/r\_j(i)^3;

v\_j\_y(i+1)=v\_j\_y(i)-4\*pi^2\*y\_j(i)\*dt/r\_j(i)^3;

%

% Use Euler Cromer technique to calculate the new positions of Earth and

% Jupiter. Note the use of the NEW vlaue of velocity in both equations

x\_e(i+1)=x\_e(i)+v\_e\_x(i)\*dt;

y\_e(i+1)=y\_e(i)+v\_e\_y(i)\*dt;

x\_j(i+1)=x\_j(i)+v\_j\_x(i)\*dt;

y\_j(i+1)=y\_j(i)+v\_j\_y(i)\*dt;

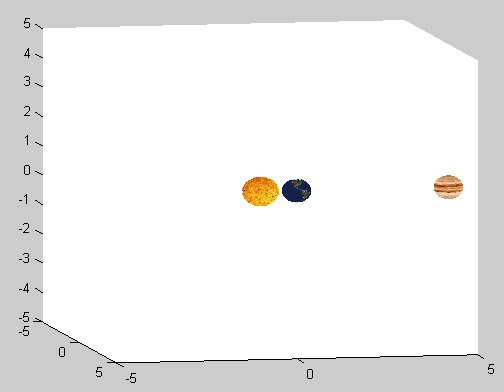
end;

# Simulation

The results of the solution were then visualised as a MATLAB movie. Below is an example frame from the movie:



Below is a graph that displays the planets in a static position with pictures of the planets:



Unfortunately this could not be implemented into the movie because it was compiling to slow.

Below is the code in its entirety:

function[x t] = earth\_jupiter\_orbit(k,j)

%EARTH\_JUPITER\_ORBIT(K,J) displays an animation of the Earth and Jupiter's

%orbit around the sun

%earth\_jupiter\_orbit(k,j) takes two arguments. argument 1 is the amount

%of Jovian years that the animation runs for, this value needs to be at

%least 3. Argument 2 is how fast the user wishes to see the animation this

%needs to be at a reasonible speed

if k<3,error('argument 1 needs to be at least 3');end

if j<50,error('Animation speed too slow, argument 2 needs to be at least 50');end

npoints=k\*118500;

dt = 0.0001; % time step in years.

x\_e\_initial=1; % Initial position of Earth in AU

y\_e\_initial=0;

v\_e\_x\_initial=0; % Initial velocity of Earth in AU/yr

v\_e\_y\_initial=2\*pi;

x\_j\_initial=5.2; % Initial position of Jupiter in AU, assume at opposition initially

y\_j\_initial=0;

v\_j\_x\_initial=0; % Initial velocity of Jupiter in AU/yr

v\_j\_y\_initial= 2.7549; % This is 2\*pi\*5.2 AU/11.85 years = 2.75 AU/year

% Create arrays to store position and velocity of Earth

x\_e=zeros(npoints,1);

y\_e=zeros(npoints,1);

v\_e\_x=zeros(npoints,1);

v\_e\_y=zeros(npoints,1);

% Create arrays to store position and velocity of Jupiter

x\_j=zeros(npoints,1);

y\_j=zeros(npoints,1);

v\_j\_x=zeros(npoints,1);

v\_j\_y=zeros(npoints,1);

r\_e=zeros(npoints,1);

r\_j=zeros(npoints,1);

% Initialise positions and velocities of Earth and Jupiter

x\_e(1)=x\_e\_initial;

y\_e(1)=y\_e\_initial;

v\_e\_x(1)=v\_e\_x\_initial;

v\_e\_y(1)=v\_e\_y\_initial;

x\_j(1)=x\_j\_initial;

y\_j(1)=y\_j\_initial;

v\_j\_x(1)=v\_j\_x\_initial;

v\_j\_y(1)=v\_j\_y\_initial;

for i = 1:npoints-1; % loop over the timesteps

% Calculate distances to Earth from Sun, Jupiter from Sun and Jupiter

% to Earth for current value of i

r\_e(i)=sqrt(x\_e(i)^2+y\_e(i)^2);

r\_j(i)=sqrt(x\_j(i)^2+y\_j(i)^2);

% Compute x and y components for new velocity of Earth

v\_e\_x(i+1)=v\_e\_x(i)-4\*pi^2\*x\_e(i)\*dt/r\_e(i)^3;

v\_e\_y(i+1)=v\_e\_y(i)-4\*pi^2\*y\_e(i)\*dt/r\_e(i)^3;

% Compute x and y components for new velocity of Jupiter

v\_j\_x(i+1)=v\_j\_x(i)-4\*pi^2\*x\_j(i)\*dt/r\_j(i)^3;

v\_j\_y(i+1)=v\_j\_y(i)-4\*pi^2\*y\_j(i)\*dt/r\_j(i)^3;

%

% Use Euler Cromer technique to calculate the new positions of Earth and

% Jupiter. Note the use of the NEW vlaue of velocity in both equations

x\_e(i+1)=x\_e(i)+v\_e\_x(i)\*dt;

y\_e(i+1)=y\_e(i)+v\_e\_y(i)\*dt;

x\_j(i+1)=x\_j(i)+v\_j\_x(i)\*dt;

y\_j(i+1)=y\_j(i)+v\_j\_y(i)\*dt;

end;

z\_k=zeros(npoints,1);

%plot(x\_e,y\_e);

%hold on;

%plot(x\_j,y\_j, 'r');

%plot3(x\_j,y\_j,z\_k, 'r');

rs=0.3;

re=0.3;

rj=0.3;

%Radius of Jupiter = 69'911km = 0.00046732617 AU

%Radius of Sun = 1'392'000km = 0.00930494527 AU

%Radius of Earth = 12'756km = 8.52685933x10^-5AU

%%3D plot

%making one image to show we tried to put image on it

%the compiling time was too long for the movie

axis([-5 5 -5 5 -5 5])

for t = 1:length(x\_e)/j

[x,y,z] = sphere(30);

pic = figure();

n = (t)\*j;

surf(rs\*x,rs\*y,rs\*z)

hold on

axis([-5 5 -5 5 -5 5])

surf(re\*x+x\_e(n),re\*y+y\_e(n),re\*z+z\_k(n))

surf(rj\*x+x\_j(n),rj\*y+y\_j(n),rj\*z+z\_k(n))

title('Earth and Jupiters Orbit')

xlabel('x (AU)')

ylabel('y (AU)')

zlabel('z (AU)')

hold off

M(t)=getframe(pic);

close(pic);

end;

movie(M)

%%

figure(2)

earth=imread('earth.jpg');

sun = imread('sun.jpg');

jupiter = imread('jupiter.jpg');

rs=0.5;

re=0.4;

rj=0.4;

%making one image to show we tried to put image on it

%the compiling time was too long for the movie

warp(rs\*x,rs\*y,rs\*z, sun)

axis([-5 5 -5 5 -5 5])

hold on

warp(re\*x+x\_e(1),re\*y+y\_e(1),re\*z+z\_k(1), earth)

warp(rj\*x+x\_j(1),rj\*y+y\_j(1),rj\*z+z\_k(1), jupiter)

%%

%%

%2D plot

ah=axes;

for n=1:500:length(x\_e)

plot(x\_j(n),y\_j(n),'o')

hold on

plot(x\_e(n),y\_e(n),'ro')

hold off

set(ah,'XLim',[min(x\_j) max(x\_j)],'YLim',[min(y\_j) max(y\_j)]);

M(n)=getframe;

end

xlabel('x(AU)');

ylabel('y(AU)');

zlabel(' ');

title('3 body simulation - Jupiter Earth');

end