Escaping the Solar System

MATLAB Design Lab

Caroline Krzyszkowski and Tayyab Jafar
Queen’s University

PHYS104/6, Tuesday @ 2:30pm

1/04/14

Abstract

Using MATLAB, a self-sustaining human colony was launched out of orbit, and with the use of Jupiter's gravitational field for a gravity assist, the colony was able to leave the solar system with enough kinetic energy and went off to the nearest star, Proxima Centauri. With MATLAB we were able to make a model of the colony leaving the solar system and do it as efficiently as possible. The initial velocity the colony was launched out of orbit was $40,100 m/s$ (this high initial velocity was needed to escape the gravitational pull of the sun) and the final velocity, given time for the function to run, was $8080 m/s$. The initial velocity the colony was launched at was still less than escape velocity for Earth, which is $42,128\ m/s$. In order to leave the solar system, our colony required the gravity assist from Jupiter.

Introduction

Newton's Law of Gravitation is what keeps planets in their orbit. There is a force between every particle in the universe that acts along the line joining the two particles. This force can be expressed as $F = \frac{-GM_1M_2}{r^2}$, where $G = 6.67 \times 10^{11}\ N\frac{m^2}{kg^2}$ and this is the gravitational constant; M_1 and M_2 represent the masses of the two particles, and r is the distance between the two particles.

The colony is essentially acting like a satellite[1] around the Sun, as Jupiter is a satellite around the Sun as well. Satellites remain in orbit due to the gravitational force experienced by the larger object. We can express this force as, $F = ma$; the satellite is orbiting in a circle therefore, undergoing centripetal acceleration, $F = \frac{mv^2}{r}$. The force between the satellite and the body it is orbiting is the gravitational force, $F = \frac{-GM_1M_2}{r^2}$, so $\frac{-GM_1M_2}{r^2} = \frac{M_1v^2}{r}$ which simplifies to $\frac{GM_2}{r} = v^2$. This equation relates the orbiting speed to the distance the satellite is from the orbiting body. To escape the gravitational pull and increase the distance from the body the satellite is orbiting, the speed of the orbiting satellite must be increased. If the speed is increased enough the satellite will reach escape velocity, which is the minimum velocity required to escape the gravitational field of the body it is orbiting.

Using the ideas of potential energy, escape velocity can be determined. The Law of Conservation of Energy can be stated as the total energy in any process remains unchanged, the energy can be transformed from one form or another, transferred between objects, as long as the total amount will remain constant. This can also be shown as $\Delta K + \Delta U + [\textit{change in all other forms of energy}] = 0$. The potential energy for the gravitational force can be derived from the ideas of work and becomes $U = \frac{-GmM}{r}$. In terms of escape velocity, the body that is being orbited is said to have zero kinetic and potential energies therefore, the equation becomes $\frac{1}{2}mv^2 - \frac{GmM}{r} = 0$, rearranging for velocity it is $v = \sqrt{\frac{2GM}{r}}$. Since the human colony is originally starting at Earth's orbit and orbiting around the Sun the radius of the orbit from the Earth to the Sun is $r = 1.5 \times 10^{11}\ m$, the mass of the sun is $1.9891 \times 10^{30}\ kg$, and the gravitational field constant is

[1] throughout this report we will refer to the space colony as either a colony, satellite or spacecraft.

$G = 6.67 \times 10^{11} N\frac{m^2}{kg^2}$. Plugging these values in, the escape velocity is found the be approximately $42,100 m/s$. Therefore, minimum velocity required for the space colony to escape the gravitational pull of the Earth is $42,100 m/s$.

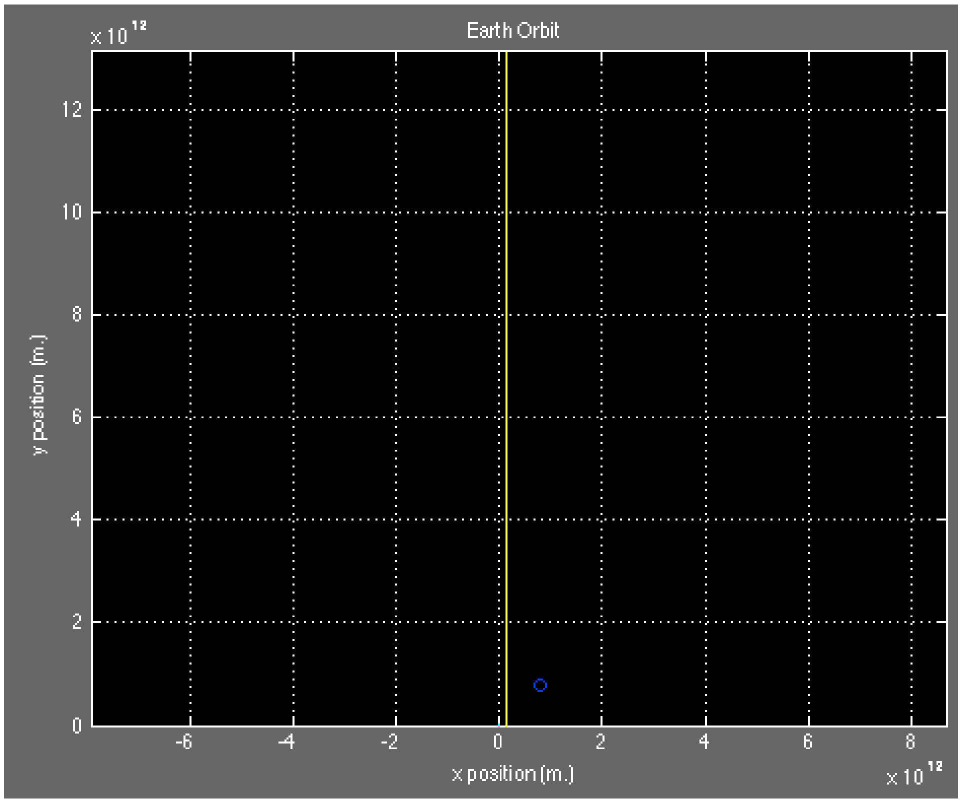
Gravity assist is useful in accelerating the space colony by using the force of gravity from another planet. In this scenario, Jupiter's gravitational field will be used by the space colony to acquire enough speed to leave the solar system and head to Proxima Centauri. The colony will start with enough velocity to leave the orbit of the Earth, but not enough to escape the gravitational pull of the Sun on its own. Therefore, the colony will need that gravitational assist from Jupiter to break free from the pull of the Sun and head to Proxima Centauri. The ideas of the conservation of energy can be used to explain gravitational assist. The colony will decelerate as it heads to Jupiter, and then it will join the orbit of Jupiter for a bit to accelerate by 'stealing' its orbital energy. However, since Jupiter's mass is extremely massive it will have a negligible effect on Jupiter, and then it will decelerate as it leaves the gravitational field of Jupiter. The energy gained from accelerating towards Jupiter by stealing its orbital energy is greater than the kinetic energy lost from exiting Jupiter's gravitational field, leading to an increase in kinetic energy and a greater speed. There is a limit to the use of gravitational assist, and that is that the planets are rarely aligned correctly so that the satellite can make use of the gravitational field. In this model, Jupiter will be started wherever needed on its orbit in order to be aligned correctly with the colony that will be coming towards it.

These ideas are currently relevant and have been put in use to accelerate spacecrafts for specific missions. Rocket fuel is expensive and limited, therefore, the ideas of gravitational assists are really powerful in aiding missions. Some examples of gravitational assist being used already are, Mariner 10 which was slingshot around Venus in order to get to Mercury and Voyager 1 which also was slingshot around both Jupiter and Saturn to escape the Sun's gravitational field completely to get to interstellar space. All of these ideas are powerful and useful to make feasible and economic missions.

Methods and Materials

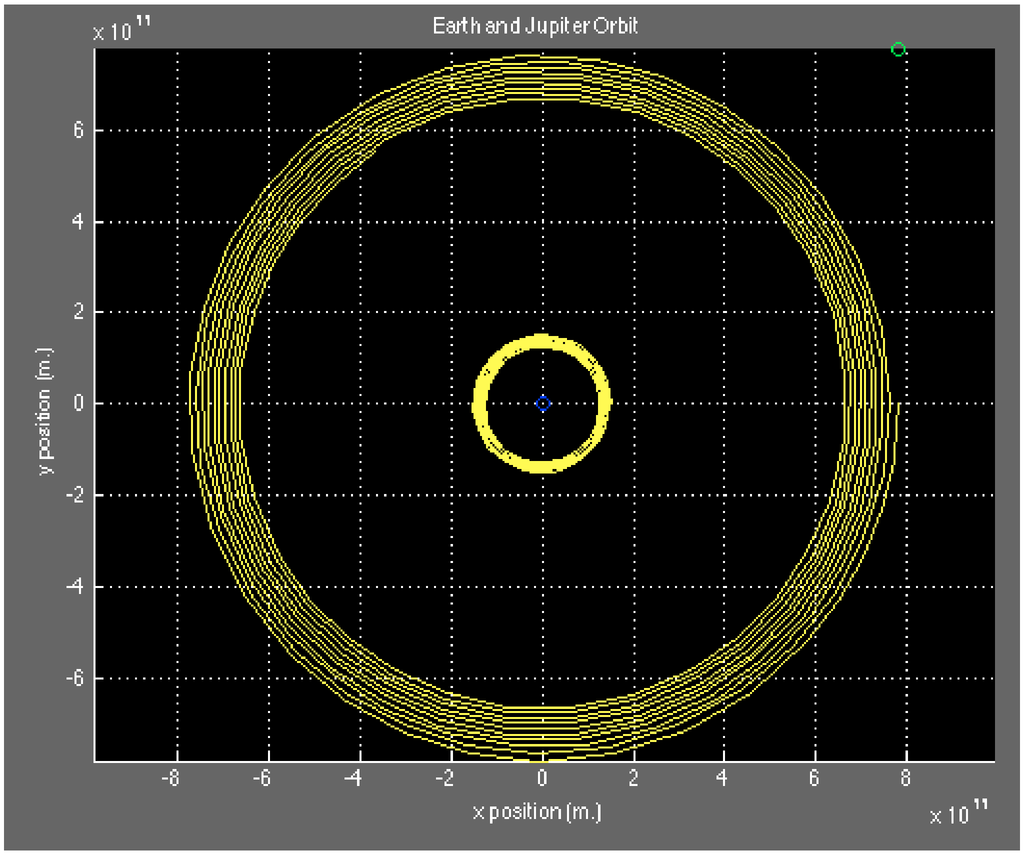
Navigating MATLAB was very tricky in the beginning and as expected we made a lot of scripts that did not work, or worked but really did not help us in any way of achieving our desired result. In the initial scripts, we were just trying to plot the orbit of Earth and Jupiter, including just writing out circles for their orbits. It took a long time for us to figure out what the ODE45 did and how it worked. Most of this came from finding scripts online and seeing what they were modelling and how they were modelling it.

Once we understood ODE45, we decided to break down the problem into smaller steps and try to complete each step. This started with modelling the orbits of the planets. We wrote a script that modelled the orbits of both Jupiter and Earth around the Sun, and as our result:



[Fig. 1]

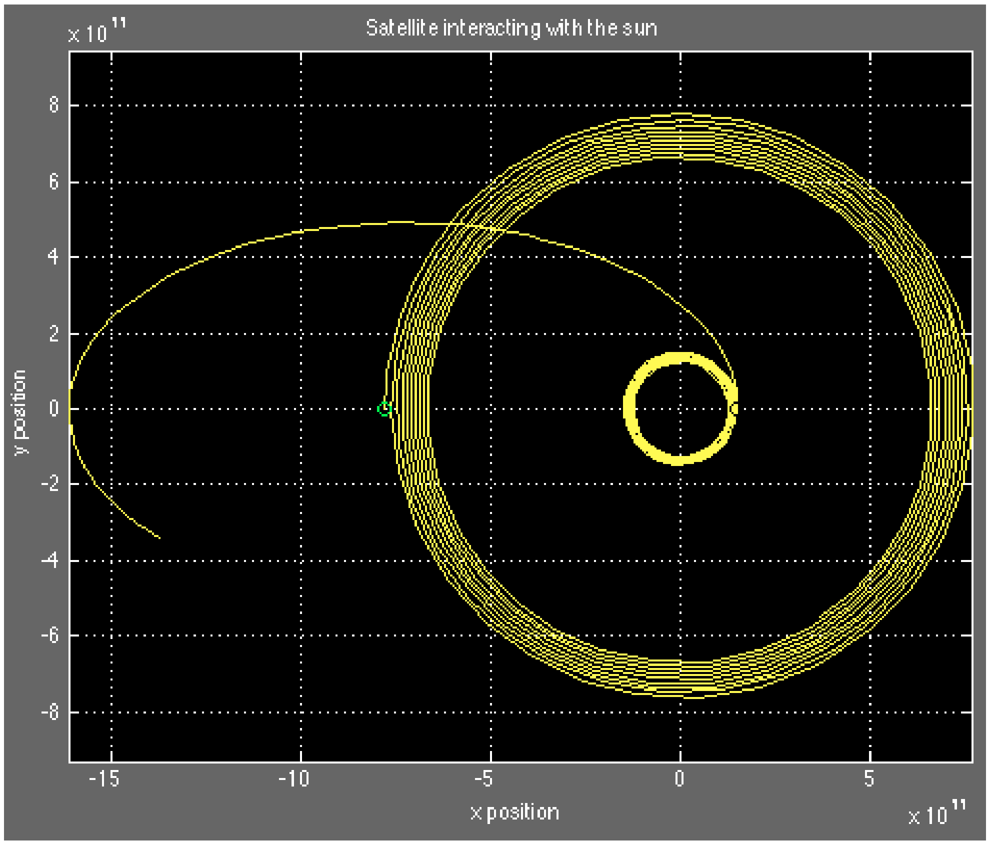
Unfortunately, the script was not quite modelling the orbits how we wanted it to and it had a lot of issues and errors, as you can see from the figure. It was then decided to break up the script into two components, one for Earth and another for Jupiter, hoping that this would fix the issue. So we went ahead and made two functions to put inside the ODE45 to get Earth's and Jupiter's orbits, and then the script to model and graph their orbits. The following is the resulting graph from this script:



[Fig 2]

We finally thought we had a good idea of MATLAB and were so close. We were ready to put in our colony and launch it out of the solar system. Going off of what we had previously, we decided to write a function to model the satellite, and then use that function in an ODE45 to have a graph of the model.

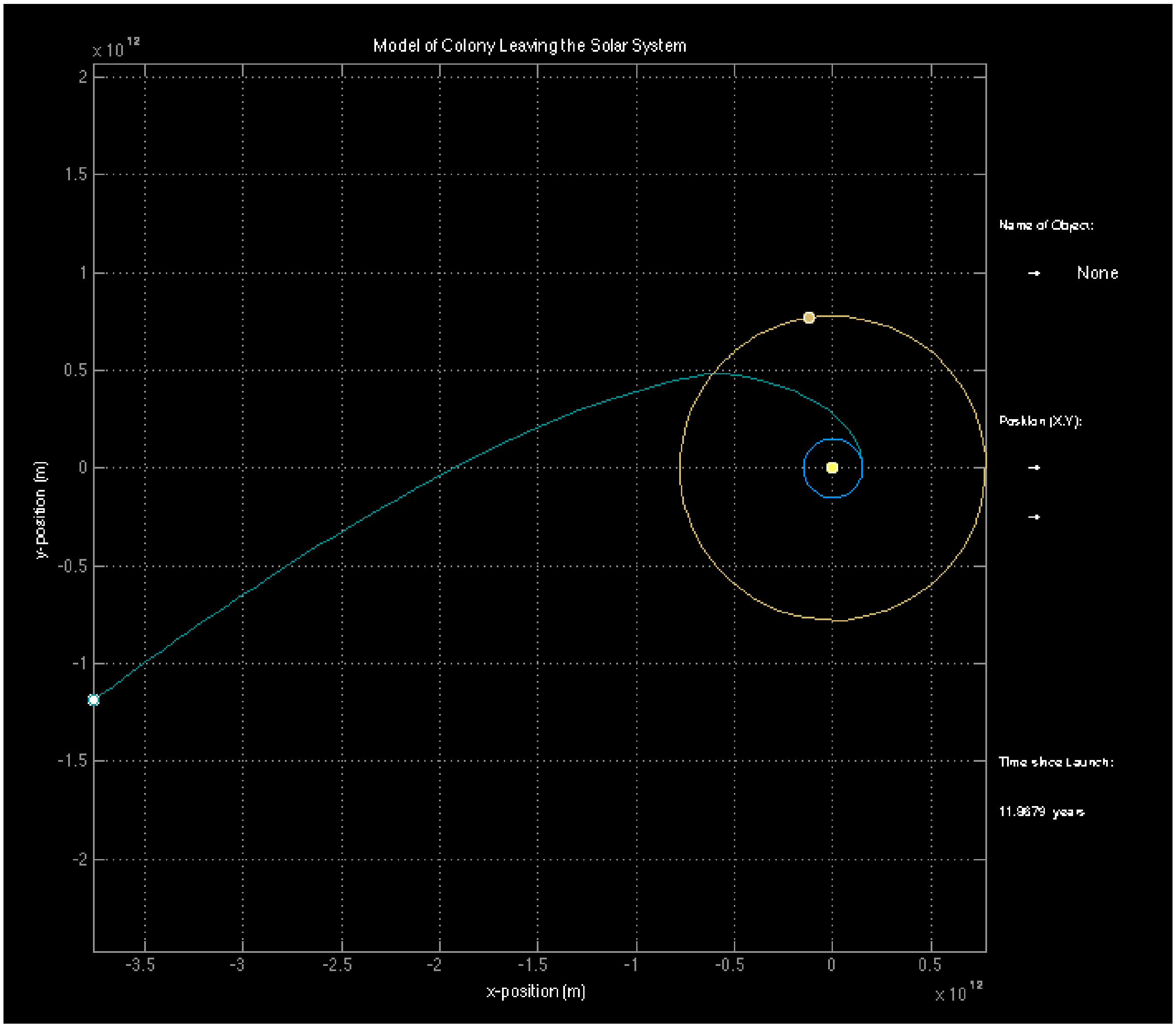
We wrote a function for the model similar to the functions we had written previously, however there was a major flaw with the equations we had derived; we did not account that the distance between the satellites and the other planets would be different in this model, which made sense as to why this function was not working. It was only taking the gravitational pull of the Sun into consideration. Our next script puts the functions we had written previously into ODE45, and then graphs it. The graph demonstrates that only the gravitational pull from the Sun is being taken into consideration.



[Fig. 3]

This left us pretty confused, not to mention that we had almost run out of time without completing our task. We went to Eric Toombs during his lab day to ask questions and to try complete the lab. We learned a lot from him, including that we had been doing a lot of unnecessary work. For example, we did not need to model Earth's orbit; just start our colony from the same orbit as Earth. As well, Jupiter's orbit did not have to be modelled with an ODE45, but instead could just be modeled as a circle. We needed to write a script for the satellite that took into consideration the gravitational pull from the Sun as well as Jupiter. Once the colony had moved close to Jupiter's orbit, we needed to position Jupiter to be in line with the colony so that the timing was correct and it could use Jupiter's gravitational assist to leave the solar system. After playing around with the script, we found the minimum velocity to launch the colony out of orbit that was also slow enough that without Jupiter's gravitational assist it would not be able to leave the solar system. Once we found where the colony would pass by Jupiter, we changed the starting angle of Jupiter and placed it in line with the colony.

As a result, the colony slingshotted around Jupiter. The following figure is our final script as it models the colony leaving the solar system:



[Fig. 4]

As observed from the final graph, we had finally achieved our result. However, attached at the end of this report is another version of the final script[2]. It was modified to be more visually

[2] After a long period of obsession with MATLAB and striving for perfection i.e. making it possible to achieve a gravitational assist with any starting value possible, it came to me that everything I was doing was just a little bit over-the-top, along with wasting time doing unnecessary things. However please feel free to check out the final *final* script.

pleasing, with some interactivity, and most importantly it allows the user to input their own initial values to model the orbit given a different set of instructions.

Results

Our final script illustrates that the space colony managed to escape the solar system using a gravity assist from Jupiter. The escape velocity for Earth is approximately $42,128\ m/s$, however, the colony was launched at a velocity of $40,100\ m/s$ which is not enough to escape the Sun's gravitational pull without any assistance from an outside source. However, by aligning Jupiter correctly the colony was able to use Jupiter's gravitational pull to escape the solar system.

The final velocity of the colony was approximately $8,080\ m/s$, this is the speed much later after going around Jupiter. All of the speed it initially started with would have been slowing down due to the g force felt from the Sun, but as it closed towards Jupiter, it got more of a boost due to the colony "falling" towards Jupiter. It begins to slow down once it has left the gravity of Jupiter far behind it, but just enough for it to escape the gravity of the Sun.

We were given an initial set of assumptions made in regards to our model. For example, we excluded Earth's gravity, we set Jupiter and the space colony at a predetermined path and that the orbits were circular. However we did change one thing, we did not start the colony at Earth, but at about $0.01 \times 10^9\ m$ above the Earth ($\approx\ 10,000\ km$). There are two reasons for this, one being that the orbit was plotted as a point. Each point would represent the very centre of the Earth, but the Earth isn't a point. It has a radius of about $6,378\ km$. This means to be accurate, the start position must be the orbit of the Earth + the radius of the Earth, at a minimum. That value would be around $(1.5 \times 10^{11}) + (6.4 \times 10^6)\ metres$. It's not major difference, but this will be important later on.

While our model demonstrates that we managed to get our satellite to escape the solar system, it isn't a perfect model and has its own flaws. For instance, take a look at our velocities. Our initial velocity is not the speed of the spacecraft alone. It is both the velocity of the Earth and colony.

$$v_t = 40.1 km/s$$

Total velocity

$$v_E = 29.8 km/s$$

Earth's velocity

$$40.1km/s - 29.8km/s = 10.3km/s = v_s$$

Starting velocity of the space colony

Now there are multiple problems with this. Despite the model showing that we managed to escape the Solar System, this would not happen in reality. For example, we neglected the gravity exerted by the Earth. At $10.3\,\frac{km}{s}$, we would surely fall back down to the Earth, as the required escape velocity is at least $11.2\,\frac{km}{s}$, and when I mean at least, this also depends on another factor, the atmosphere. Due to the atmosphere, it is not possible to bring an object's speed up to $11.2\,\frac{km}{s}$. Atmospheric drag would destroy the object or it would burn up long before it escapes Earth. In reality, most objects are put into low Earth orbit, with speeds of about $8\,\frac{km}{s}$, which is then accelerated to the escape velocity of that altitude, which is around $10.9\,\frac{km}{s}$. That is much more efficient and practical as the change in speed is much less than the difference from starting at rest on the Earth to $11.2\,\frac{km}{s}$.

So what does this mean? It means to be efficient, our colony should start somewhere way up above the Earth where the atmosphere isn't an issue. To make things much more clean and simple, remember the minimum required position was $(1.5 \times 10^{11}) + (6.4\ \times 10^{6})\ metres$? Well we rounded way up and assumed that our colony is starting at $(1.51 \times 10^{11})\ m \approx 1,000,000\ km$ above the center of the Earth. That is quite an over exaggeration for our chosen value but in the end, it still results in a success for our colony.

Discussion

We managed to complete our goal, that is we launched a self-sustaining human colony to Proxima Centauri. We expected that we could leave Earth's orbit will less velocity than escape velocity, but still be able to escape the solar system by using a gravity assist from Jupiter. Our final graph illustrated that result.

Despite the fact that we achieved our goal, there is quite a lot that could have been done differently. We wasted a lot of time trying to get unnecessary objects to work, such as modelling Earth's orbit. Had we asked for clarification earlier, we wouldn't have wasted a few weeks trying to model three objects at the same time. However, this did mean that we were able to go beyond our goal by gaining a better understanding of MATLAB. Once we had written our final script we went even further by changing it to make it interactive so that anyone could try modelling a colony leaving the solar system, but could also see the colony leaving the solar system with the default setting we had modelled the original colony.

In the end, we gained valuable MATLAB skills while accomplishing our goal and ending up with the results that were expected.

Acknowledgements

Toombs, Eric. (2014). *Escaping the Solar System and Slowing Down Jupiter a Teeny Weeny Bit*. Retrieved February 10, 2014, from https://ilt.seas.harvard.edu/images/material/596/318/proxima_centauri.pdf

(2011). *A MATLAB Script for Interplanetary Gravity-Assist Trajectory Design and Optimization*. Web. Retrieved March 30, 2014, from http://www.cdeagle.com/interplanetary/flyby_matlab.pdf

(2012). "Draw Ellipse." *Documentation Center - MathWorks*. Web.Retrieved March 30, 2014, from http://www.mathworks.com/help/pde/ug/pdeellip.html

(2012). "Draw Circle." *Documentation Center*. Web. Retrieved March 30, 2014, from http://www.mathworks.com/help/pde/ug/pdecirc.html

(2011). plotting circles - MATLAB Answers - MATLAB Central - MathWorks. Retrieved April 1, 2014, from http://www.mathworks.com/matlabcentral/answers/3058-plotting-circles.

(2009). "Calculating Satellite Orbits from Newton's Law of Gravity." Web. Retrieved March 30, 2014, from http://www.civerson.com/E215/pages/56.html

(2014). "Escape Velocity." *Wikipedia*. Wikimedia Foundation. Web. Retrieved March 30, 2014, from http://en.wikipedia.org/wiki/Escape_velocity

Giancoli, Douglas C. (2008). *Physics for Scientists & Engineers*. 4th ed. Vol. 2. Harlow: Pearson Education Limited. 1328 pages.

(2014). "Gravity Assist." *Wikipedia*. Wikimedia Foundation. Web. Retrieved March 30, 2014, from http://en.wikipedia.org/wiki/Gravity_assist

(2014). "Jupiter." *Wikipedia*. Wikimedia Foundation. Web. Retrieved March 30, 2014, from http://en.wikipedia.org/wiki/Jupiter

(2014). "Sun." *Wikipedia*. Wikimedia Foundation. Web. Retrieved March 30, 2014, from http://en.wikipedia.org/wiki/Sun

(2002). “MATLAB Central.” *MathWorks*. Retrieved March 30, 2014, from http://www.mathworks.com/matlabcentral/

We would also like to thank Josh Bonnatt and Eric Toombs for all their help they've given us while working on this lab. [3]

Appendix

[Script 1: Earth Orbit]

→{Download Here}←

```
function orbittrial_1()
% This script combines the function we will put in the ODE45, and then the
% ODE45 after it to keep it all in one script.

function [yp] = Orbit_model( t,z )
% Function to model the Earth's orbit as well as Jupiter's and their
% gravitational pull between them

%%What we know:
% Do not worry about Earth's gravity, assume circular
% orbits, assume once we have left the solar system we are heading to
% Proxima Centauri, assume that once Earth reaches its escape velocity it
% will interact directly with jupiter and so little time will pass that it
% will head on a path that is perpendicular to both orbits

% What is it doing?
% Leaving Earth's gravitational field and then using
% Jupiter's gravitational field for a gravity assist to increase the
% velocity to further propel it out of the solar system

% How is it doing it?
% Through a model that will change in velocity
% initially and then interact with Earth's gravitational fields

% Next we need to figure out the forces on Earth and Jupiter that will allow
% us to determine the velocity it is travelling at
% Since we are using ODE45 deriving the acceleration from the force will
% allow us to solve for velocity, which will then allow us to solve for
% position

%% Givens:
% Distance of Earth to Sun: 1.5ell m.
% Distance of Earth to Jupiter: 6.2e11 m
% Mass of the sun (M1): 1.9891e30 kg
% Mass of the Earth: 5.97e24 kg
% Mass of Jupiter (M2): 1.8986e27 kg
% gravitational field constant: G= 6.67e-11 m^3/kg*s^2
```

[3] along with Professor Fraser for allowing us to do the design lab on MATLAB!

```
%% Deriving the equations:
% F=ma
% F=-G*M*m/r^2   gravitational force, x and y components need to be found
% r=(x+y)^1/2    this can be obtained by plotting Earth at a coordinate and
% the sun at the origin, we want to simplify the variables so they would
% only be in terms of x and y, and then find the x and y components of the
% force
% cos(theta)=x/r     sin(theta)=y/r   these are the x,y components
% Therefore, Fx=-G*M*m*x/(x+y)^3/2    Fy=-G*M*m*y/(x+y)^3/2
% Therefore, ma=-G*M*m*x/(x+y)^3/2 so ax=-G*M*x/(x+y)^3/2 and ay=-G*M*y/(x+y)^3/2
% Where m is the mass of either the sun when modelling Earths orbit, or
% Jupiter when modelling Jupiter's orbit because we are looking for the
% orbit of Earth
% With these steps done we can now look at using matlab

%% Constants
G=6.67e-11 %m^3/kg*s^2 gravitational field constant
M= 1.9891e30  %mass of the sun

%% Variables
%Variables for Earth Calculations:
x=z(1);   %earth's position in x
y=z(2);   %earth's position in y
vx=z(3);  %earth's velocity in x
vy=z(4);  %earth's velocity in y
%Variables for Jupiter Calculations:
x=z(5);   %jupiters position in x
y=z(6);   %jupiters position in y
vx=z(7);  %jupiters velocity in x
vy=z(8);  %jupiters velocity in y

%% Functions
%these four model the gravitational pull of the Sun-Earth
%yp(1)=z(3); %these two functions model the velocity of the mass, (taken from the integration of yp3,4)
%p(2)=z(4);
%yp(3)=-G*M*x/(x^2+y^2)^(3/2); %these two model the acceleration in the x,y directions which gives us the velocity
%yp(4)=-G*M*y/(x^2+y^2)^(3/2);
%these four model the gravitational pull of Jupiter
%p(5)=z(7);
%yp(6)=z(8);
%yp(7)=-G*M*x/(x^2+y^2)^(3/2);
%yp(8)=-G*M*x/(x^2+y^2)^(3/2);
%yp=[yp(1);yp(2);yp(3);yp(4);yp(5);yp(6);yp(7);yp(8)]
yp=[z(3);z(4);-G*M*x/(x^2+y^2)^(3/2);-G*M*y/(x^2+y^2)^(3/2);z(7);z(8);-G*M*x/(x^2+y^2)^(3/2);-G*M*y/(x^2+y^2)^(3/2)]

%% Conclusion about this function
%It was not working to make a function of 8 variable therefore, this ended
%up not being used. We however, will use the ideas at the beginning of this
%function to continue with our next functions where we break it up:
%earth_model and jupiter_model.
end

%The following is the ODE45 that will model the above function
%[t,y]=ode45(@orbit_model,[tmin,tmax],[x(tmin),y(tmin),vx(tmin),vy(tmin)]);
[t,z]=ode45(@Orbit_model,[0,3.2e8],[1.5e11,0,0,41000,7.7e30,0,0,420]);
```

```
%The following is the plot for the graph
plot(z(:,1),z(:,2),0,0,z(:,3),z(:,4),7.785e11,7.785e11,'bo'),axis equal, grid on
hold on
%These are the graph's labels
title('Earth Orbit'),xlabel('x position'),ylabel('y position')
hold on
end
```

[Functions for]:

Earth's orbit: →**{Download Here}**←	**Jupiter's orbit:** →**{Download Here}**←	**Plotting Circles[4]:** →**{Download Here}**←

function [ep]=earth_model(t,e)
 	
%% Constants
G=6.67e-11 %m^3/kg*s^2 gravitational field constant
M= 1.9891e30 %mass of the sun
 	
%% Variables
%Variables for Earth Calculations:
x=e(1);   %earth's position in x
y=e(2);   %earth's position in y
vx=e(3); %earth's velocity in x
vy=e(4); %earth's velocity in y
 	
%% functions
%these four model the gravitational pull of the Sun-Earth
%e(1)=e(3); %these two functions model the velocity of the mass, (taken from the integration of yp3,4)
%e(2)=e(4);
%e(3)=-G*M*x/(x^2+y^2)^3/2; %these two model the acceleration in the x,y directions which gives us the velocity
%e(4)=-G*M*y/(x^2+y^2)^3/2;
ep=[vx;vy;-G*M*x/(x^2+y^2)^(3/2);-G*M*y/(x^2+y^2)^(3/2)]
 	
%% Conclusions about this function
%This function is working and is modelling Earth's velocity/position
%therefore, this is the one we will be using for Earth.
end
	function [jp]=jupiter_model(t,j)
	
	%% Constants
	G=6.67e-11 %m^3/kg*s^2 gravitational field constant
	M= 1.9891e30 %mass of the sun
	
	%Variables for Jupiter Calculations:
	x=j(1); %jupiters position in x
	y=j(2); %jupiters position in y
	vx=j(3); %jupiters velocity in x
	vy=j(4); %jupiters velocity in y
	
	%these four model the gravitational pull of Jupiter
	%jp(5)=j(7);
	%jp(6)=j(8);
	%jp(7)=-G*M*x/(x^2+y^2)^(3/2);
	%jp(8)=-G*M*x/(x^2+y^2)^(3/2);
	jp=[vx;vy;-G*M*x/(x^2+y^2)^(3/2);-G*M*y/(x^2+y^2)^(3/2)]
	
	%% Conclusions about this function
	%This function is working and is modelling Jupiter's velocity/position
	%therefore, this is the one we will be using for Jupiter.
	
	end
		function circle(x,y,r)
		
		% this function plots a circle
		% x and y are the coordinates of the center of the circle
		% r is the radius of the circle
		% 0.01 is the angle step, bigger values will draw the circle faster but
		% you might notice imperfections (not very smooth)
		% This function will plot the orbit of Jupiter since we do not need to use
		% the ODE45 as we had been previously.
		ang=0:0.01:2*pi;
		xp=r*cos(ang);
		yp=r*sin(ang);
		plot(x+xp,y+yp);
		end

[4] (2011). plotting circles - MATLAB Answers - MATLAB Central - MathWorks.

[Script 2: Earth and Jupiter Orbits]:

→{Download Here}←

```
%This is the second trial of modelling Earth and Jupiter's orbits

%This following circle will model Jupiter's orbit as a circle
%circle(0,0,7.785e11);
hold on

%Model Earth's orbit
%[t,y]=ode45(@earth_model,,[tmin,tmax],[x(tmin),y(tmin),vx(tmin),vy(tmin)]);
[t,e]=ode45(@earth_model,[0,3.2e8],[1.5e11,0,0,29900]);

%Model Jupiter's Orbit
%[t,y]=ode45(@jupiter_model,,[tmin,tmax],[x(tmin),y(tmin),vx(tmin),vy(tmin)]);
[t,j]=ode45(@jupiter_model,[0,3.2e9],[7.785e11,0,0,-13070]);

%plot of both ODE45 functions and the circle
plot(e(:,1),e(:,2),0,0,'bo'),axis equal,grid on
plot(j(:,1),j(:,2),7.785e11,7.785e11,'go'),axis equal,grid on
hold on
title('Earth and Jupiter Orbit'),xlabel('x position'),ylabel('y position')
hold on
```

[Function - Satellite]:

→{Download Here}←

```
function [sp]=satellite_model(t,s)
%% Deriving the equations for the satellite

%F=ma
% F=-G*M*m/r^2   gravitational force, x and y components need to be found
% r=(x+y)^1/2    this can be obtained by plotting Earth at a coordinate and
% the sun at the origin, we want to simplify the variables so they would
% only be in terms of x and y, and then find the x and y components of the
% force
% cos(theta)=x/r     sin(theta)=y/r   these are the x,y components
% Therefore, Fx=-G*M*m*x/(x+y)^3/2    Fy=-G*M*m*y/(x+y)^3/2
% Therefore, ma=-G*M*m*x/(x+y)^3/2 so ax=-G*M*x/(x+y)^3/2 and ay=-G*M*y/(x+y)^3/2
% Where m is the mass of either the sun when modelling Earths orbit, or
% Jupiter when modelling Jupiter's orbit because we are looking for the
% orbit of Earth
% With these steps done we can now look at using matlab

%% Constants
G=6.67e-11 %m^3/kg*s^2 gravitational field constant
M1= 1.9891e30  %mass of the sun
M2= 1.9e27 %kg  mass of the jupiter
M3= 5.97e24  %kg  mass of the earth

%Variables for satellites Calculations:
x=s(1);   %satellites position in x
y=s(2);   %satellites position in y
vx=s(3);  %satellites velocity in x
vy=s(4);  %satellites velocity in y
```

```
%these four model the gravitational pull on the satellite from the three
%bodies around it (sun, earth, jupiter)
%sp(5)=s(3);
%sp(6)=s(4);
%sp(7)=(-G*x(M1+M2+M3))/(x^2+y^2)^(3/2);
%sp(8)=(-G*y(M1+M2+M3))/(x^2+y^2)^(3/2);
sp=[vx;vy;(-G*x*(M1+M2+M3))/(x^2+y^2)^(3/2);(-G*y*(M1+M2+M3))/(x^2+y^2)^(3/2)]

%% Conclusions about this function
%This function is not working and we need to come up with another solution.

end
```

[Satellite Interacting with the Sun]:

→{Download Here}←

```
%The following three functions represent the ODE45 that will map the three
%bodies.
[t,e]=ode45(@earth_model,[0,3.2e8],[1.5e11,0,0,29800]);
[t,j]=ode45(@jupiter_model,[0,3.2e9],[-7.785e11,0,0,13070]);
[t,s]=ode45(@satellite_model,[0,3.2e8],[1.51e11,0,0,40100]);

%These map the three bodies onto the graph
plot(e(:,1),e(:,2),1.5e11,0,'bo'),axis equal, grid on
hold on
plot(j(:,1),j(:,2),-7.785e11,0,'go'),axis equal, grid on
hold on
plot(s(:,1),s(:,2),1.51e11,0,'ko'),axis equal, grid on

%Adding titles to the graph
title('Satellite interacting with the other bodies'),xlabel('x position'),ylabel('y position')
hold on
```

[Final Script - Model of Colony Leaving the Solar System]:

→{Download Here}←

```
function model()
% this function is the model which contains another function responsible for calculating the ode45 function
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                              CALCULATIONS AND DERIVATIONS
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////
% rs represents the displacement between the colony and the sun.
% rj-rs represent the displacement between the colony and Jupiter
% F=ma
% F=-G*M*m/r^2(r-vector for direction) gravitational force, x and y components will be part of an r vector so matlab will be able to
calculate them from us
% Therefore, F due to sun=-G*MS*m*rs/(rs)^3    F due to Jupiter=-G*MJ*m*(rj-rs)/(rj-rs)^3
% Therefore, ma(sun)=-G*MS*m*rs/(rs)^3 so as=-G*MS*rs/(rs)^3 and a(jupiter)=-G*MJ*(rj-rj)/(rj-rs)^3
% Where G is the gravitational field constant, m is the mass of the colony, MS is the mass of the sun, and MJ is the mass of Jupiter
% With these steps done we can now look at using matlab
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                                          CONSTANTS
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////
% the below are defined constants required in the function
% OMEGA in rad/sec is the angular frequency of Jupiter
% orbj in m, is the orbit radius between jupiter and the sun
% G is the gravitational field constant, in m^3/kg*s^2
% MS is the mass of the sun in kg
OMEGA = 1.68E-8;
orbj = 7.785E11;
G = 6.67E-11;
MS = 1.9891E30;
MJ = 1.9E27;

% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                               SPACE COLONY FUNCTION
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% function responsible for setting up values required for the ode45 function
function [sp]=satmod(t,s)
% THETA in this function is used to calculate jupiters orbit
% theta in radians, is a calculated adjustment value for placing jupiter in the same location as space colony while it travels by
% rs is the position of the colony in m, broken into x and y from the relationship x^2 +y^2 = r^2
% it is then placed into the first and second column vectors in s (ie. x and y)
% vs is the velocity of the colony in m/s, broken into vx and vy
% just as rs, it is placed into the third and fourth column vectors in s (ie. vx and vy)
theta = .548632*pi;
THETA = (OMEGA*t)+theta;
rs =  s(1:2);
vs =  s(3:4);

% rj in m, is the position of jupiter in x and y
rj =  [cos(THETA) sin(THETA)]'*orbj;
% aj is the accel. of colony due to the force from jupiter in m/s^2
% asun is the accel. of colony due to the force from sun in m/s^2
% as is the accel. of spacecraft from the addition of the two forces
% from the sun and jupiter
% sp is the output of the function
aj = (-G*MJ/(norm(rj-rs)).^3*(rj-rs));
asun = (-G*MS/(norm(rs).^3))*(rs);
as = asun + aj;
sp = [vs;as]; %output of the function
end
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                                           INITIAL CONDITIONS
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% IPx and IPy are the starting/initial positions of the colony in the x and y direction, in m
% IVx and IVy are the initial velocities of the colony in the x and y direction, in m/s
% tmin and tmax are used for the time elapsed from the beginning of the function to the end in seconds
IPx = 1.51E11;
IPy = 0;
IVx  = 0;
IVy = 40100;
tmin = 0;
tmax = 3.74265E8;
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                                           ODE45 FUNCTION
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% [t,s]=ode45(@satmod,[tmin,tmax],[x(tmin),y(tmin),vx(tmin),vy(tmin)]);
% this returns the value of a system of 4 first-order ODEs.
[t,s]=ode45(@satmod,[tmin,tmax],[IPx,IPy,IVx,IVy]);
% where [tmin,tmax] represents the time elapsed (from zero seconds to one full jovian year in seconds: 3.74265E8).
% sidenote: the time for the space colony and jupiter to intersect = 4.396E7 seconds in this scenario.
% IPx = 1.51E11 which is the starting position of the satellite from the sun, in the y direction in meters. This can
% be changed, but we decided to start 0.01E9 meters above the Earth's orbit
% in order to illustrate the escape, also since this would plot the center
% of the earth and not the surface. If this does change, the initial velocity
% would have to increase if the starting position decreased (as it is closer to the sun), and vice-versa
% if it increases its starting position.
% IPy and IVx are the y position (m) and velocity (m/s) in the x position; these are starting at 0 in our scenario.
% IVy = 40100 which is the initial velocity in the y direction (m/s).

% below are the same expressions from the function above in order to call it when run outside of the function
THETA = (OMEGA*t)+theta;
rj =  [cos(THETA) sin(THETA)]*orbj;
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                                           PLOTTING THE ORBITS
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% the script below plots the orbit of the space colony with axis having the
% same aspect ratio in each direction.
% s(:,1) and s(:,2) call upon all values
% from the first and second column (x and y) of the space colony.
% IPx,IPy, 'ko' define the x and y coordinate to plot a key (black) circle
% to indicate the starting position of the space colony.
plot(s(:,1),s(:,2),IPx,IPy,'ko'),axis equal, grid on
hold on
% the script below plots the orbit of Jupiter and its location at the end of tmax
% where rj(end,1),rj(end,2),'go' represent the final values at the end of its calculated
% vector in order to define an x and y coordinate, plotting a green circle
% to indicate jupiter at the end of tmax
plot(orbj*cos(THETA),orbj*sin(THETA),rj(end,1),rj(end,2),'go');
%These are the graph's labels
title('Model of Colony Leaving the Solar System'),xlabel('x-position (m)'),ylabel('y-position (m)')
hold on
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                              AESTHETICALLY PLEASING CIRCLE
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% the script below plots a filled in circle at the origin to indicate the sun
% given the radius of the sun in meters
% linspace generates 100 linearly spaced vectors from zero to 2*pi
% in order to plot a full circle, defined with a red line, filled with a
% red circle, at the origin.
radius = 6.955E8;
THETA = linspace(0,2*pi);
SX = radius * cos(THETA);
SY = radius * sin(THETA);
plot(SX,SY,'-r');
fill(SX,SY,'r');
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%                                           FINAL VELOCITY
% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% this calculates the final velocity in the x and y direction, in m/s
% and throws out the number in the command window. can be hidden
% by placing a semicolon at the end if not required.
velocityx = s(end,3);
velocityy = s(end,4);
v = sqrt(velocityx^2 + velocityy^2)
end
```

FINAL FINAL SCRIPT[5]

→ {Download Here} ←

```
% ======================================================================================================
%                    WELCOME TO THE GRAVITY ASSIST MODEL-er
% ======================================================================================================
function gravityassist()
% calls upon the function


% ------------------------------------------------------------------------
% used to begin global variables required to direct the script given a
% certain input in the menu
op=0; clc
% ends global variables
% ------------------------------------------------------------------------


% User Menu Begins
while op~=4
disp ('Welcome to our Interactive Gravity Assist Model')
disp ('by Caroline Krzyszkowski and Tayyab Jafar')
disp (' ')
disp ('As the name suggests, this script will model a space colony, Colony Pierniczka (Polish: Gingerbread Cookie),')
disp ('escaping (or orbiting) the solar system to head towards Proxima Centauri, via a gravity assist from Jupiter.')
disp ('It plots the orbits of Jupiter and the trajectory of the colony.')
disp ('The orbit of the Earth (in blue) is also displayed as a reference, along with the Sun.')
disp (' ')
disp ('Please be aware that the script will not run given small or very large inputs.')
disp ('If the default model is not interactive, re-run the script.')
disp ('Furthermore, the custom model has a predefined theta value which places Jupiter at a certain position.')
disp ('To change Jupiters position at a time t, change theta inside the script.')
disp (' ')
disp ('If you would like to see the default model, press 1.')
disp ('If you would like to create your own model, press 2.')
disp ('If you do not like science, press 3 and please re-evaulate your life.')
disp (' ')
disp('1)Default')
disp('2)Custom')
disp('3)Exit')
op = input ('Please choose an option: ');
disp (' ')
% User Menu Ends


% ------------------------------------------------------------------------
switch op
        case 1 %Default
        DefaultModel
break
        case 2 %Custom Orbit
        CustomModel
        break
        case 3 %break
        break
        otherwise
        disp('THAT WAS NOT AN OPTION')
```

[5] Note: this script uses the command whitebg([0 0 0]) which makes the figure black. For whatever reason, this effect is permanent, so if you run this script and run anything else afterwards, you will end up with inverted and dark colours. To undo the effect, just type in the command window: whitebg([1 1 1])

```
            pause(2);
            disp('TRY AGAIN.')
            pause(3);
            gravityassist
% if case 1 is inputted, it will run the default model function,
% and custom model for case 2. If any other options are chosen, it will end
% the function via the break command.
% -------------------------------------------------------------------------

end
end


% //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


function DefaultModel()
% this function is the default model which contains nultiple functions
% responsible for calculating the ode45 and plotting the results.

% =========================================================================================================
%           CALCULATIONS AND DERIVATIONS
% =========================================================================================================
% rs represents the displacement between the colony and the sun.
% rj-rs represent the displacement between the colony and Jupiter
% F=ma
% F=-G*M*m/r^2(r-vector for direction) gravitational force, x and y components will be part of an r vector so matlab will be able to calculate them from us
% Therefore, F due to sun=-G*MS*m*rs/(rs)^3          F due to Jupiter=-G*MJ*m*(rj-rs)/(rj-rs)^3
% Therefore, ma(sun)=-G*MS*m*rs/(rs)^3 so as=-G*MS*rs/(rs)^3 and a(jupiter)=-G*MJ*(rj-rj)/(rj-rs)^3
% Where G is the gravitational field constant, m is the mass of the colony, MS is the mass of the sun, and MJ is the mass of Jupiter
% With these steps done we can now look at using matlab


% =========================================================================================================
%           CONSTANTS
% =========================================================================================================
OMEGA = 1.68E-8;
orbj = 7.785E11;
G = 6.67E-11;
MS = 1.9891E30;
MJ = 1.9E27;
ER = 1.5E11;
% defined constants required in the function
% OMEGA in rad/sec is the angular frequency of Jupiter
% orbj in m, is the orbit radius between jupiter and the sun
% G is the gravitational field constant, in m^3/kg*s^2
% MS is the mass of the sun in kg


% =========================================================================================================
%           SPACE COLONY FUNCTION
% =========================================================================================================
function [sp]=satmod(t,s) % function responsible for setting up values required for the ode45 function
theta = .548632*pi;
THETA = (OMEGA*t)+theta;
rs =  s(1:2);
vs =  s(3:4);
% THETA in this function is used to calculate jupiters orbit
% theta in radians, is a calculated adjustment value for placing jupiter in the same location as space colony while it travels by
% rs is the position of the colony in m, broken into x and y from the relationship x^2 +y^2 = r^2
% it is then placed into the first and second column vectors in s (ie. x and y)
% vs is the velocity of the colony in m/s, broken into vx and vy
% just as rs, it is placed into the third and fourth column vectors in s (ie. vx and vy)


% -------------------------------------------------------------------------
rj =  [cos(THETA) sin(THETA)]'*orbj;
% rj in m, is the position of jupiter in x and y
% -------------------------------------------------------------------------


aj = (-G*MJ/(norm(rj-rs)).^3*(rj-rs));
asun = (-G*MS/(norm(rs).^3))*(rs);
as = asun + aj;
sp = [vs;as];
% aj is the accel. of colony due to the force from jupiter in m/s^2
% asun is the accel. of colony due to the force from sun in m/s^2
% as is the accel. of spacecraft from the addition of the two forces
% from the sun and jupiter
% sp is the output of the function
end


% ===================================================================================================================
%                     INITIAL CONDITIONS
% ===================================================================================================================
% IPx and IPy are the starting/initial positions of the colony in the x and y direction, in m
% IVx and IVy are the initial velocities of the colony in the x and y direction, in m/s
% tmin and tmax are used for the time elapsed from the beginning of the function to the end in seconds
IPx = 1.51E11;
IPy = 0;
IVx  = 0;
IVy = 40100;
tmin = 0;
tmax = 3.74265E8;


% ===================================================================================================================
%                     ODE45 FUNCTION
% ===================================================================================================================
% [t,s]=ode45(@satmod,[tmin,tmax],[x(tmin),y(tmin),vx(tmin),vy(tmin)]);
% this returns the value of a system of 4 first-order ODEs.
[t,s]=ode45(@satmod,[tmin,tmax],[IPx,IPy,IVx,IVy]);
% where [tmin,tmax] represents the time elapsed (from zero seconds to one full jovian year in seconds: 3.74265E8).
% sidenote: the time for the space colony and jupiter to intersect = 4.396E7 seconds in this scenario.
% IPx = 1.51E11 which is the starting position of the satellite from the sun, in the y direction in meters. This can
% be changed, but we decided to start 0.01E9 meters above the Earth’s orbit
% in order to illustrate the escape, also since this would plot the center
% of the earth and not the surface. If this does change, the initial velocity
% would have to increase if the starting position decreased (as it is closer to the sun), and vice-versa
% if it increases its starting position.
% IPy and IVx are the y position (m) and velocity (m/s) in the x position; these are starting at 0 in our scenario.
% IVy = 40100 which is the initial velocity in the y direction (m/s).


% --------------------------------------------------------------------------
THETA = (OMEGA*t)+theta;
rj =  [cos(THETA) sin(THETA)]*orbj;
% these are the same expressions from the function above in order to call it when run outside of the function


% ===================================================================================================================
%                     PLOTTING THE ORBITS
% ===================================================================================================================
function icantbelievethisworked()
% the function is only used for the 'default model'. since we have
% predetermined values from the model, we can make things look a bit nicer.
% it sets up a figure with a callback that executes on mouse motion,
% essentially 'records' the mouse position and when it enters a
% certain range in the axes (X,Y), it will place information off to the
% side. in this case, when hovering over bodies, it will place the name,
% location and so forth on the side. this one is more about appearance than
% functionality.
% --------------------------------------------------------------------------


% these are just handles used in the function to display labels, positions, etc.
figHdl = figure('WindowButtonMotionFcn', @hoverCallback);
axesHdl = axes;
lineHdl = plot(0,0,'wo','MarkerFaceColor',[1 1 0],'Parent',axesHdl);
textHdl = text('Color', 'white','Position',[1.25E12 1E12]);
posXHdl = text('Color', 'white','Position',[1.25E12 0E12]);
posYHdl = text('Color', 'white','Position',[1.25E12 -0.25E12]);
% google 'function handles matlab' for more info...


% --------------------------------------------------------------------------
text(0.85E12,-1.5E12,'Time since Launch:','FontSize',8);
% displays the 'text' at a given position. featuring fontsize 8!
% --------------------------------------------------------------------------


tim = tmax./(60*60*24*365);
text(0.85E12,-1.75E12,[num2str(tim) '  years'],'FontSize',8);
% same as before, but in case someone wants to play around with the script
% and combine the custommodel with this, it saves a line by displaying the
% time inputted and displays it on the figure.


% ------------------------------------------------------------------------
text(0.85E12,1.25E12,'Name of Object:','FontSize',8);
text(1E12,1E12,'\rightarrow','FontSize',8);
text(0.85E12,0.25E12,'Position (X,Y):','FontSize',8);
text(1E12,0E12,'\rightarrow','FontSize',8);
text(1E12,-0.25E12,'\rightarrow','FontSize',8);
% I hate commenting.
% ------------------------------------------------------------------------


function hoverCallback(~, ~)
        % grab the x & y axes coordinate where the mouse is
        mousePoint = get(axesHdl, 'CurrentPoint');
        mouseX = mousePoint(1,1);
        mouseY = mousePoint(1,2);
        % if the distance is between a given threshold, set the text
        % object's string to show the data at that point.
        if mousePoint(1,1) < 1E11 && mousePoint(1,1) > -1E11 && mousePoint(1,2) < 1E11 && mousePoint(1,2) > -1E11
                set(textHdl,'String', {'Sun'});
                set(posXHdl,'String','0  m');
                set(posYHdl,'String','0  m');
        elseif mousePoint(1,1) < -0.5E11 && mousePoint(1,1) > -2E11 && mousePoint(1,2) < 9E11 && mousePoint(1,2) > 7E11
                set(textHdl, 'String', {'Jupiter'});
                precision = 3;
                XPOS = num2str(rj(end,1),precision);
                YPOS = num2str(rj(end,2),precision);
                set(posXHdl,'String',[XPOS ' m']);
                set(posYHdl,'String',[YPOS ' m']);
        elseif  mousePoint(1,1) < -3.5E12 && mousePoint(1,1) > -4E12 && mousePoint(1,2) < -1E12 && mousePoint(1,2) > -1.25E12
                set(textHdl, 'String', {'Pierniczka'});
                precision = 3;
                XPOS = num2str(s(end,1),precision);
                YPOS = num2str(s(end,2),precision);
                set(posXHdl,'String',[XPOS ' m']);
                set(posYHdl,'String',[YPOS ' m']);
        else
                set(textHdl, 'String', 'None')
                set(posXHdl, 'String', '')
                set(posYHdl, 'String', '')
        end
end
end
icantbelievethisworked; %calls the function to run
% it took me hours of figuring out an error before this thing stopped crashing.


% ------------------------------------------------------------------------
set(gcf,'units','normalized','outerposition',[0 0 1 1])
% this makes the figure fullscreen when it opens. I did this because when
% the figure is small, it tends to squish labels together. I don't know any
% other methods to fix this, so fullscreen it is!
% ------------------------------------------------------------------------


whitebg([0 0 0]),axis vis3d, grid on;
% this sets up the figure to be black, to sort of simulate the "blackness"
% of space. the figure is not 3D but it restricts text from being stretched.


% ------------------------------------------------------------------------
set(gcf,'Color',[0 0 0]);
set(gca, 'XColor', [0.5 0.5 0.5], 'YColor', [0.5 0.5 0.5], 'ZColor', [0.5 0.5 0.5])
% these sets the color of the background and axes of the plot, to black and grey in
% this case.
% ------------------------------------------------------------------------


title('Model of Colony Leaving the Solar System','Color','w');
xlabel('x-position (m)','Color','w');
ylabel('y-position (m)','Color','w');
box on;
hold on
% these lines just set up the figure with the proper labels. box displays
% the boundary of the graph; all just visual features.
% --------------------------------------------------------------------------
plot(s(:,1),s(:,2),s(end,1),s(end,2),'wo','MarkerFaceColor',[1 1 1],'Color','w',[0 0.5 0.5]);
% plots the orbit of the space colony
% s(:,1) and s(:,2) call upon all values
% from the first and second column (x and y) of the space colony.
% s(end,') returns the final X and Y Position of the colony, and plots a
% circle representing it.
% --------------------------------------------------------------------------


plot(orbj*cos(THETA),orbj*sin(THETA),'Color',[0.8 0.7 0.3]);
plot(rj(end,1),rj(end,2),'wo','MarkerFaceColor',[0.8 0.7 0.3]);
plot(0,0,'wo','MarkerFaceColor',[1 1 0]);
plot(ER*cos(THETA),ER*sin(THETA),'Color',[0 0.5 1]);
% plots the orbit of Jupiter and its location at the end of tmax
% where rj(end,1),rj(end,2),’go’ represent the final values at the end of its calculated
% vector in order to define an x and y coordinate, plotting a green circle
% to indicate jupiter at the end of tmax


% ========================================================================================================
%                    FINAL VELOCITY
% ========================================================================================================
velocityx = s(end,3);
velocityy = s(end,4);
v = sqrt(velocityx^2 + velocityy^2);
disp(['Final velocity = ' num2str(v) 'm/s'])
% this calculates the final velocity in the x and y direction, in m/s
% and throws out the number in the command window. can be hidden
% by placing a semicolon at the end if not required.
end


% //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
% //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


function CustomModel()
% this function is the custom model which contains functions required to
% calculate the ode45 command and plot its results, given initial values
% inputted by the user.


% ========================================================================================================
%                    CALCULATIONS AND DERIVATIONS
% ========================================================================================================
% rs represents the displacement between the colony and the sun.
% rj-rs represent the displacement between the colony and Jupiter
% F=ma
% F=-G*M*m/r^2(r-vector for direction) gravitational force, x and y components will be part of an r vector so matlab will be able to calculate them from us
% Therefore, F due to sun=-G*MS*m*rs/(rs)^3                    F due to Jupiter=-G*MJ*m*(rj-rs)/(rj-rs)^3
% Therefore, ma(sun)=-G*MS*m*rs/(rs)^3 so as=-G*MS*rs/(rs)^3 and a(jupiter)=-G*MJ*(rj-rj)/(rj-rs)^3
% Where G is the gravitational field constant, m is the mass of the colony, MS is the mass of the sun, and MJ is the mass of Jupiter
% With these steps done we can now look at using matlab


% ========================================================================================================
%                    CONSTANTS
% ========================================================================================================
OMEGA = 1.68E-8;
orbj = 7.785E11;
G = 6.67E-11;
MS = 1.9891E30;
MJ = 1.9E27;
ME = 5.9736E24;
ER = 1.5E11;
% defined constants required in the function
% OMEGA in rad/sec is the angular frequency of Jupiter
% orbj in m, is the orbit radius between jupiter and the sun
% G is the gravitational field constant, in m^3/kg*s^2
% MS is the mass of the sun in kg


% ========================================================================================================
%                    SPACE COLONY FUNCTION
% ========================================================================================================
function [sp]=satmod(t,s) % function responsible for setting up values required for the ode45 function
theta = .548632*pi;
THETA = (OMEGA*t)+theta;
rs =  s(1:2);
vs =  s(3:4);
% THETA in this function is used to calculate jupiters orbit
% theta in radians, is a calculated adjustment value for placing jupiter in the same location as space colony while it travels by
% rs is the position of the colony in m, broken into x and y from the relationship x^2 +y^2 = r^2
% it is then placed into the first and second column vectors in s (ie. x and y)
% vs is the velocity of the colony in m/s, broken into vx and vy
% just as rs, it is placed into the third and fourth column vectors in s (ie. vx and vy)


% ------------------------------------------------------------------------
rj =  [cos(THETA) sin(THETA)]'*orbj;
% rj in m, is the position of jupiter in x and y
% ------------------------------------------------------------------------


aj = (-G*MJ/(norm(rj-rs)).^3*(rj-rs));
asun = (-G*MS/(norm(rs).^3))*(rs);
as = asun + aj;
sp = [vs;as];
% aj is the accel. of colony due to the force from jupiter in m/s^2
% asun is the accel. of colony due to the force from sun in m/s^2
% as is the accel. of spacecraft from the addition of the two forces
% from the sun and jupiter
% sp is the output of the function
end


% ==========================================================================================================
%                    INITIAL CONDITIONS
% ==========================================================================================================
% IPx and IPy are the starting/initial positions of the colony in the x and y direction, in m
% IVx and IVy are the initial velocities of the colony in the x and y direction, in m/s
% tmin and tmax are used for the time elapsed from the beginning of the function to the end in seconds


%these will allow the user to input their own initial values
disp('Please choose the initial starting variables for the model');
disp (' ')
IPx  = input('type your starting x position in m (recommended 1.51E11): ');
IPy = input('type your starting y position in m (recommended 0): ');
IVx = input('type your initial x velocity in m/s (recommended 0): ');
IVy = input('type your initial y velocity in m/s (recommended 40100): ');
tmin = input('type your initial time in s (recommended 0): ');
tmax = input('type your final time in s (recommended 3.74E8): ');


% ==========================================================================================================
%                    ODE45 FUNCTION
% ==========================================================================================================
% [t,s]=ode45(@satmod,[tmin,tmax],[x(tmin),y(tmin),vx(tmin),vy(tmin)]);
% this returns the value of a system of 4 first-order ODEs.
[t,s]=ode45(@satmod,[tmin,tmax],[IPx,IPy,IVx,IVy]);
% where [tmin,tmax] represents the time elapsed (from zero seconds to one full jovian year in seconds: 3.74265E8).
% sidenote: the time for the space colony and jupiter to intersect = 4.396E7 seconds in this scenario.
% IPx = 1.51E11 which is the starting position of the satellite from the sun, in the y direction in meters. This can
% be changed, but we decided to start 0.01E9 meters above the Earth’s orbit
% in order to illustrate the escape, also since this would plot the center
% of the earth and not the surface. If this does change, the initial velocity
% would have to increase if the starting position decreased (as it is closer to the sun), and vice-versa
% if it increases its starting position.
% IPy and IVx are the y position (m) and velocity (m/s) in the x position; these are starting at 0 in our scenario.
% IVy = 40100 which is the initial velocity in the y direction (m/s).



% ------------------------------------------------------------------------
THETA = (OMEGA*t)+theta;
rj =  [cos(THETA) sin(THETA)]*orbj;
% these are the same expressions from the function above in order to call it when run outside of the function


% ==========================================================================================================
%                    PLOTTING THE ORBITS
% ==========================================================================================================
whitebg([0 0 0]),axis vis3d, grid on;
%this sets up the figure to be black, to sort of simulate the "blackness"
%of space. the figure is not 3D but it restricts text from being stretched.


% ------------------------------------------------------------------------
set(gcf,'Color',[0 0 0]);
set(gca, 'XColor', [0.5 0.5 0.5], 'YColor', [0.5 0.5 0.5], 'ZColor', [0.5 0.5 0.5])
%these sets the color of the background and axes of the plot, to grey in
%this case.
% ------------------------------------------------------------------------


title('Model of Colony Leaving the Solar System','Color','w');
xlabel('x-position (m)','Color','w');
ylabel('y-position (m)','Color','w');
box on;
% these lines just set up the figure with the proper labels. box displays
% the boundary of the graph; all just visual features.


% ------------------------------------------------------------------------
hold on %TO YOUR HORSES
% ------------------------------------------------------------------------


plot(s(:,1),s(:,2),s(end,1),s(end,2),'wo','MarkerFaceColor',[1 1 1],'Color',[0 0.5 0.5]),axis equal, grid on;
% plots the orbit of the space colony with axis having the
% same aspect ratio in each direction.
% s(:,1) and s(:,2) call upon all values
% from the first and second column (x and y) of the space colony.
% IPx,IPy, ‘ko’ define the x and y coordinate to plot a key (black) circle
% to indicate the starting position of the space colony.


% ------------------------------------------------------------------------
plot(orbj*cos(THETA),orbj*sin(THETA),'Color',[0.8 0.7 0.3]);
plot(rj(end,1),rj(end,2),'wo','MarkerFaceColor',[0.8 0.7 0.3]);
plot(ER*cos(THETA),ER*sin(THETA),'Color',[0 0.5 1]);
plot(0,0,'wo','MarkerFaceColor',[1 1 0]);
% plots the orbit of Jupiter and its location at the end of tmax
% where rj(end,1),rj(end,2),’go’ represent the final values at the end of its calculated
% vector in order to define an x and y coordinate, plotting a circle
% to indicate jupiter at the end of tmax
% along with a circle to indicate the sun
% ------------------------------------------------------------------------


text(rj(end,1),rj(end,2),' \leftarrowJupiter','FontSize',8);
text(s(end,1),s(end,2),' \leftarrowPierniczka','FontSize',8);
text(0,0,' \leftarrowSun','FontSize',8);
%the lines here labels the objects/plots above, just for reference.


% ==========================================================================================================
%                    FINAL VELOCITY
% ==========================================================================================================
velocityx = s(end,3);
velocityy = s(end,4);
v = sqrt(velocityx^2 + velocityy^2);
disp(['Final velocity = ' num2str(v) 'm/s'])
% this calculates the final velocity in the x and y direction, in m/s
% and throws out the number in the command window. can be hidden
% by placing a semicolon at the end if not required.

% Captain Obvious: this ends the function
end
end
```