ME20014- Venus Aerocapture

Determining the trajectory of a spacecraft that travels into Venus' orbit

Arya Mukherjee, Fahim Rahman & Hasan Rahman

01/12/2020

Abstract

An investigation was conducted to determine a spacecraft's trajectory as it approaches Venus's orbit and ends at the orbit's lowest point (periapsis). Using a second order ODE, containing the weight and drag forces, and the Runge Kutta method the spacecraft's trajectory was plotted, which gave an insertion altitude, apoapsis and periapsis of 1,329,000m, 1,202,800m and 95,471m, respectively. To then find the insertion altitude which exactly gave a 1,200,000m apoapsis, a shooting method was applied to the MATLAB code used to plot the trajectory, yielding an insertion altitude, apoapsis and periapsis of 1,329,005m, 1,200,000m and 95,471m. This accurate trajectory using the correct insertion altitude to give H=1200km was then made user-friendly by creating a GUI, which allowed the user to input any insertion altitude and find the corresponding apoapsis and periapsis.

Introduction

Modern aerospace technology has allowed the idea of a mission to Mars to be a question of when rather than if. However, to reach and interact with the furthest planets of our solar system, the theory of 'aerocapture' must be discussed. To save fuel on long space flights, aerocapture utilises the drag from the planet's atmosphere to slow the spacecraft enough to insert it into the planet's orbit as its trajectory is now changed from hyperbolic to elliptical. To determine the feasibility of 'aerocapture', a mission of inserting a satellite into Venus' orbit using an 'aerocapture' system was posed. This mission was solved by first plotting the trajectory of the spacecraft by deriving the corresponding ODE and then applying the Runge Kutta method. To then find the altitude which results in the correct apoapsis of 1200km, the shooting method was applied to the ODE which changed the problem from a BVP to an IVP. With the trajectory and correct altitude found, an accurate representation of the spacecraft's trajectory to reach the lowest point of Venus' orbit can be illustrated. The diagram below shows a sketch of the spacecraft's trajectory, with the initial conditions of altitude and displacement, and the required apoapsis of 1200km.

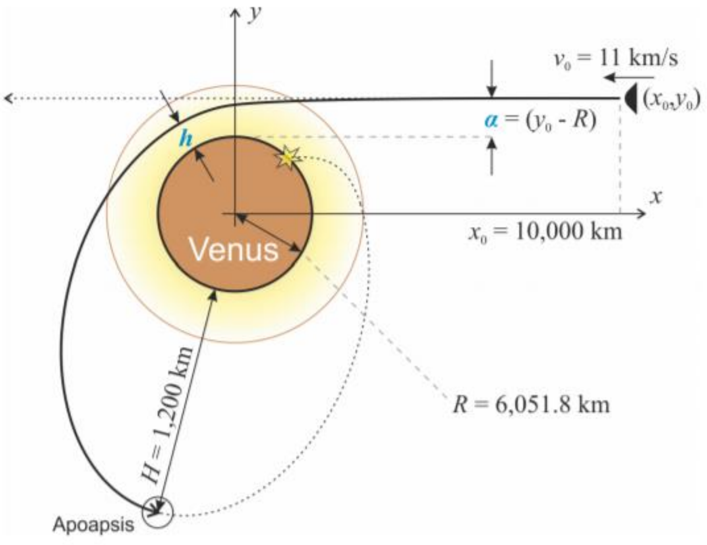
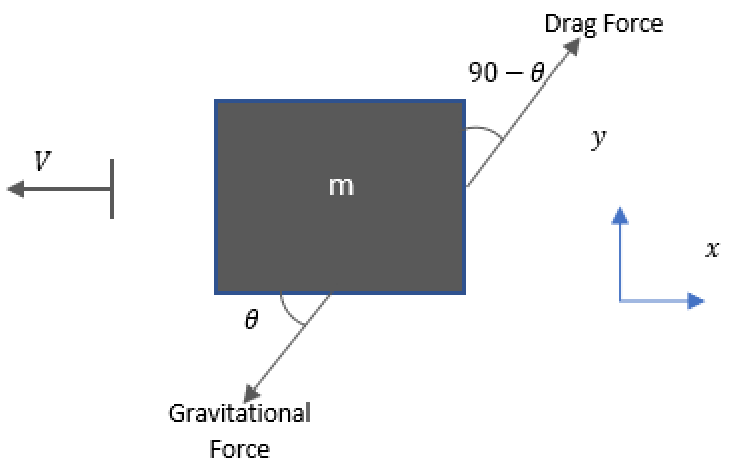


Figure 1: Sketch of the spacecraft's trajectory around Venus and its features

Derivation of the ODEs

Free body diagram of the spacecraft:



Gravitational Force, $F = G\frac{Mm}{(r-R)^2}$ (**1**) (lumen, n.d.)

Where

- G is the gravitational constant.
- M and m are the masses of Venus and the spacecraft respectively.
- R and r are the radius of Venus and distance between the spacecraft and Venus' surface respectively.

Drag Force, $D = \frac{C_d A\rho V^2}{2}$ (**2**) (NASA, 2015)

Where

- C_d is the drag coefficient.
- A is the cross-sectional area.
- ρ is the density of Venus' atmosphere
- V is the velocity of the spacecraft.

Using Newton's second law for the x and y directions respectively:

$$m\ddot{x} = -\frac{GMm\cos\theta}{(\sqrt{x^2+y^2}-R)^2} + \frac{C_d A\rho\cos(90-\theta)}{2}(\dot{x}^2+\dot{y}^2)$$

$$m\ddot{y} = -\frac{GMm\sin\theta}{(\sqrt{x^2+y^2}-R)^2} + \frac{C_d A\rho\sin(90-\theta)}{2}(\dot{x}^2+\dot{y}^2)$$

Where

r = $\sqrt{x^2+y^2}$ and $V^2 = \dot{x}^2+\dot{y}^2$.

$$\text{Hence, } \ddot{x} = -\frac{GM\cos\theta}{(\sqrt{x^2+y^2}-R)^2} + \frac{C_d A\rho\cos(90-\theta)}{2m}(\dot{x}^2+\dot{y}^2)$$

$$\ddot{y} = -\frac{GM\sin\theta}{(\sqrt{x^2+y^2}-R)^2} + \frac{C_d A\rho\sin(90-\theta)}{2m}(\dot{x}^2+\dot{y}^2)$$

With the state vector variable being:

$$Z = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} y \\ \dot{y} \\ x \\ \dot{x} \end{pmatrix}$$

Hence, the derivatives of the state variables are:

$$z'_1 = \dot{y} = z_2$$

$$z'_3 = \dot{x} = z_4$$

$$\theta = \tan^{-1}\frac{z_1}{z_3}$$

$$z'_2 = \ddot{y} = -\frac{G \cdot M \cdot sin\theta}{(\sqrt{{z_3}^2 + {z_1}^2} - R)^2} + \frac{C_d \cdot A \cdot \rho \cdot \sin(90-\theta)}{2m}({z_4}^2 + {z_2}^2)$$

$$z'_4 = \ddot{x} = -\frac{G \cdot M \cdot cos\theta}{(\sqrt{{z_3}^2 + {z_1}^2} - R)^2} + \frac{C_d \cdot A \cdot \rho \cdot \cos(90-\theta)}{2m}({z_4}^2 + {z_2}^2)$$

These derivatives of the state variables will be used in the MATLAB code (see appendix) in order to calculate the displacements and velocities of the spacecraft in the x and y direction.

Assumptions

When deriving the coupled first order ordinary differential equations (ODEs), the only forces taking in consideration where the gravitational and drag forces. However, the effect of lift was assumed to be 0. As an improvement, the lift force can be implemented by using **Equation 3**:

Lift, $L = \frac{C_l A \rho V^2}{2}$ (NASA, 2015)

Where C_L is the lift coefficient and can be found through experiments.

As the problem became a boundary value problem (BVP), the shooting method was used (see appendix) in order to get the actual value of α for the desired orbit (where the apoapsis was 1200km).

Another assumption that was made was that the spacecraft was only analysed in two dimensions. Whereas in real life, the spacecraft will experience forces in three dimensions. As an initial improvement, the spacecraft's free body diagram can implement the forces in the z axis as well as the x and y axis.

The shooting method works by solving a BVP using initial value problem (IVP) methods, such as the Runge Kutta method. The shooting method considers the boundary conditions as a multivariate function of initial conditions at some point, reducing the boundary value problem to finding the initial conditions that give a root, where the resultant error is equal to 0.

There were assumptions that were made by using the shooting method. These included:

- The relationship of the boundaries was linear.
- The relationship between the two errors was linear.

These assumptions were slightly bypassed by iterating the error code multiple times in order to get an error of 0.

Spacecraft trajectory

Using the ivpSolver function the trajectory of the spacecraft was plotted, shown in **Figure 2**. This function works by stepping using the Runge-Kutta method, which calculates four different slopes and uses them as weighted averages. The green line highlights the periapsis, the smallest distance between the spacecraft and the red dashed line represents the apoapsis (semi-major axis in the elliptical orbit). The insertion altitude (α) was set to 1,329,000m, which resulted in an apoapsis of 1,202,800m and a minimum altitude of 95,471m.

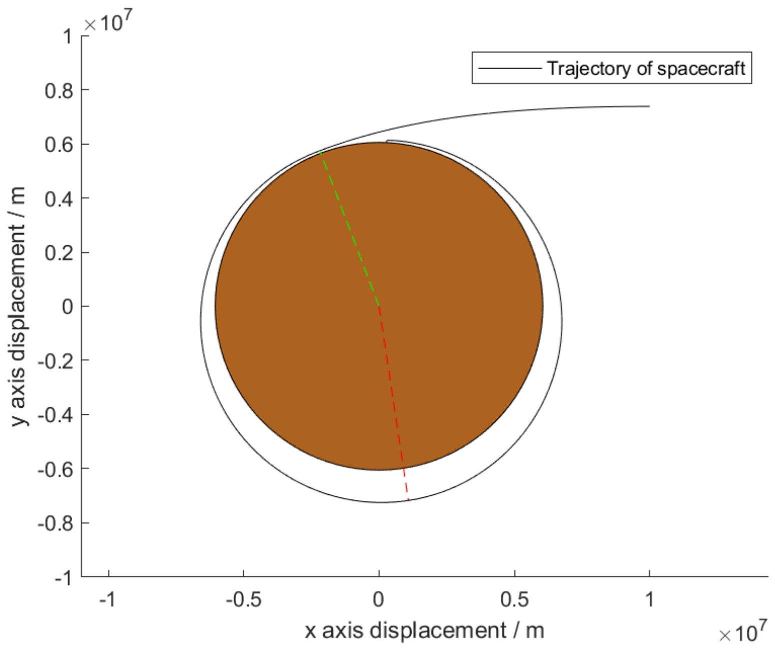


Figure 2: The spacecraft's trajectory found by using the ODE and Runge Kutta method

After finding the spacecraft’s trajectory, the shooting method was applied to the BVP to find the accurate insertion altitude of the spacecraft for the correct apoapsis to be yielded. This was achieved by inputting two random altitudes which set the altitude boundaries. A new altitude was then found using the shooting method, where this altitude was computationally iterated multiple times until the insertion altitude yielded an apoapsis that was the same as the required apoapsis, within 1%. The MATLAB scripts (see Appendix) all assumed the assumptions of linearity between the errors and altitude boundaries described in ‘Assumptions’. The insertion altitude found was 1,329,004.71m, the apoapsis was 1,200,000.000006315m and the periapsis was 95,470.6917m. The diagram below shows the iterations that the MATLAB script computed to find the final insertion altitude.

Extension Tasks

MATLAB comes bundled with a very extensive and intuitive GUI designer called App Designer which made the GUI design quite easy even though object-oriented programming is notoriously complicated. The UI was split into 2 sections, the input section and the graphing section. The input grid section has 7 text inputs for the 3 time values and 4 position values, it also has 2 radio buttons that control whether the path of the spacecraft is animated or not and a slider that controls said animation speed (by adjusting the amount of data points added on each iteration). When the “Simulate” button is pressed, all the input data is fetched and validated, if any fields fail to meet the validation criteria a popup is displayed with the relevant error message. Once all the data has been properly validated, the program calls the ivpSolver function to get the position and velocity vector of the simulation. This data is then passed to the custom class function GraphDraw which plots the 4 graphs in the tabbed graphing section. If the animation button is set to on, it will animate the plotting of the spacecraft path at a speed set by the slider value. After the main path had been plotted, it would also plot other graphs of displacement, velocity and acceleration all against time in their own tab. MATLAB doesn't have a very robust way of dealing with concurrent call-backs from button presses, they're simply added to a queue, but this behaviour didn't work well with the animated graphing and it frequently caused it to crash. So, to work around that, a try catch statement was set up around the plotting which would halt all plotting and clear the figure if too many call-backs were made.

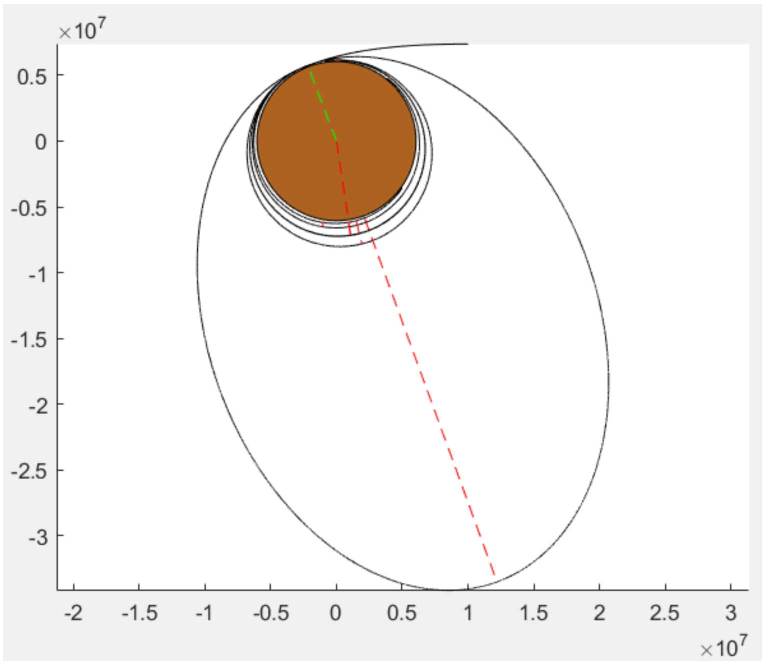
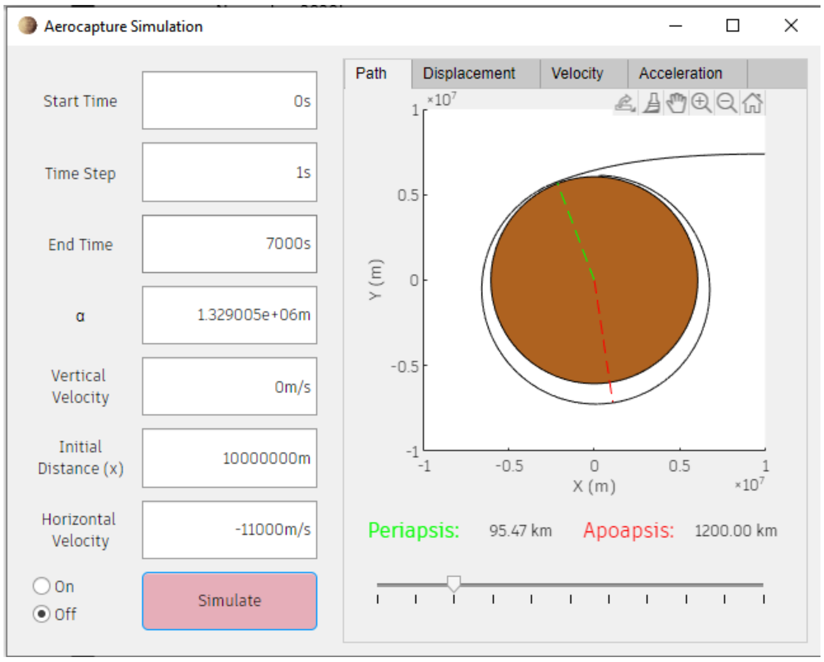


Figure 4: Graph showing the multiple spacecraft trajectories yielded by the shooting method and the iterations to find the final insertion altitude

As well as a GUI, the code was made as efficiently as possible, especially whilst writing the bvpSolver, utilising features such as lambda functions to improve the runtime of the code and reduce memory usage. Also, where possible, inbuilt functions were preferred over loops in favour of efficiency and keeping the big O of the code as close to n as possible. To keep the number of files low, all programs called the same version of ivpSolver. To achieve this, ivpSolver was adjusted to take varargin as the final argument so that it could differentiate whether it was called by the GUI or by bvpSolver. This was necessary as when it's called by ivpSolver, it needs to plot a graph whilst the GUI plots its own graph.


Aerocapture Simulation

Start Time	0s
Time Step	1s
End Time	7000s
α	1.329005e+06m
Vertical Velocity	0m/s
Initial Distance (x)	10000000m
Horizontal Velocity	-11000m/s

On [✓ Off]

Periapsis: 95.47 km Apoapsis: 1200.00 km

Figure 5: *Picture showing the calculated Apoapsis and Periapsis using the GUI.*

Conclusion

Overall, this investigation was successful, as it showed how a spacecraft can be landed onto a planet such as Venus using only its atmosphere and gravitational forces. The investigation displayed how the insertion altitude effects the spacecraft's orbit around Venus. The investigation also showed how programming can solve IVP and BVP using the Runge Kutta/Euler and the shooting methods respectively. The problem was successfully developed further using a GUI in order to make the program more user-friendly.

References

2020. Assignment 2: Numerical Modelling Of Aerocapture For A Scientific Mission To Venus. [ebook] Bath: Univesity of Bath. Available from: https://moodle.bath.ac.uk/pluginfile.php/1648631/mod_resource/content/0/ME20014_Assignment2_Aerocapture_Venus.pdf [Accessed 20 November 2020].

lumen, n.d. Newton's Law of Universal Gravitation | Boundless Physics [Online]. Available from: https://courses.lumenlearning.com/boundless-physics/chapter/newtons-law-of-universal-gravitation/#:~:text=The%20mathematical%20formula%20for%20gravitational [Accessed 23 November 2020].

mathworks, 2020. Data space to figure units conversion [Online]. Available from: https://uk.mathworks.com/matlabcentral/fileexchange/10656-data-space-to-figure-units-conversion [Accessed 22 November 2020].

NASA, 2015. The Drag Equation [Online]. Available from: https://www.grc.nasa.gov/www/k-12/airplane/drageq.html#:~:text=The%20drag%20equation%20states%20that [Accessed 23 November 2020].

NASA, 2015. The Lift Equation [Online]. Available from: https://www.grc.nasa.gov/www/k-12/airplane/lifteq.html [Accessed 27 November 2020].

Appendix

MATLAB code defining and implementing the ODEs of the spacecraft into ivpSolver.m (stateDeriv.m)

```
function dz = stateDeriv(z)
% Calculate the state derivative for a mass-spring-damper system
%
%     DZ = stateDeriv(T,Z) computes the derivative DZ = [V; A] of the
%     state vector Z = [X; V], where X is displacement, V is velocity,
%     and A is acceleration.
% z =[y, dy,x, dx]

M = 4.8675e24; % Mass of Venus(kg)
m = 1500; % Mass of satellite (kg)
C = 1.25; % Drag coefficient
A = 9;
G = 6.67408e-11;
r = hypot(z(1,end),z(3,end)); %Current distance from origin
R = 6051800; %Radius of Venus
theta = atan2d(z(1,end),z(3,end)); %Angle between spacecraft position and
origin
phi = atan2d(z(2,end),z(4,end)); %Angle between spacecraft velocity and
origin
z_sign = sign(z(:,end)); %Signs of current positions and velocities
rho = profileVenus(r-R);
grav =(-G*M)/(r^2);
drag = (((-C*A*rho)/(2*m)) * (z(2,end)^2 +z(4,end)^2));

%Imeplementation of ODEs using previously defined sign values as gravity
%acts opposite to the position and drag acts opposite to the velocity
dz1 = z(2,end);
dz2 = (z_sign(1)* grav *abs(sind(theta))) + (z_sign(2)* drag
*abs(sind(phi)));
dz3 = z(4,end);
dz4 = (z_sign(3)* grav *abs(cosd(theta)))  + (z_sign(4)* drag
*abs(cosd(phi)));

dz = [dz1; dz2; dz3; dz4];
end
```

MATLAB code that solves ODE using Runge Kutta 4 (stepRK4.m)

```
function znext = stepRK4(z,dt)
% stepRungeKutta    Compute one step using the RungeKutta method
%     ZNEXT = stepRungeKutta(T,Z,DT) computes the state vector ZNEXT at the
next
%     time step T+DT
% Calculate the state derivative from the current state
dz = stateDeriv(z);

% Calculate A, B,C and D for dz
A = dt * dz;

B = dt * stateDeriv(z + A/2);
C = dt * stateDeriv(z + B/2);
```

```
D = dt * stateDeriv(z + C);
% Calculate the next state vector from the previous one using Euler's
% update equation

znext = z +(A+2*B+2*C+D)/6;
end
```

Initial Value Problem MATLAB code for the spacecraft (ivpSolver.m)

```
function [t,z] = ivpSolver(inputs,varargin)
% ivpSolver    Solve an initial value problem (IVP) and plot the result
%
%     [T,Z] = ivpSolver(T0,Z0,DT,TE) computes the IVP solution using a step
%     size DT, beginning at time T0 and initial state Z0 and ending at time
%     TEND. The solution is output as a time vector T and a matrix of state
%     vectors Z.

% Set initial conditions
t(1) = inputs(1);
t0= inputs(1);
dt = inputs(2);
tend = inputs(3);
z(:,1) = inputs(4:end);

% Continue stepping until the end time is exceeded
n=1;
while t(n) <= tend
    % Apply RungeKutta's method for one time step
    new_z = stepRK4(z(:,n), dt);

    % Breaks the loop once stateDeriv returns a non valid results (due to
    % the spacecraft hitting the surface of Venus
    if any(isnan(new_z)|isinf(new_z))
        break
    else
        % Increment the time vector by one time step
        t(n+1) = t(n) + dt;
        z(:,n+1) = new_z;
        n=n+1;
    end
end

% Plots spacecraft path if additional argument is passed to the function
% when it's called
if isempty(varargin)
    %Caclucates an array of total displacements
    H = hypot(z(1,:),z(3,:));

    %Defines initial conditions of apoapsis and periapsis
    max_H=[0;0];
    min_H=[1e9;0];
    main_ax=gca;

    % Draws Venus
    rectangle(main_ax,'Position',[-6051800 -6051800 12103600
12103600],'Curvature',[1,1],'FaceColor', '#ae6220');

    % Assigns variable for animated spacecraft path plotting
    displacement = animatedline(main_ax);
    axis(main_ax,[-1e7,1e7,-1e7,1e7],'auto','equal');

    % Disables panning and zooming of plot
    disableDefaultInteractivity(main_ax)

    % Specifies the time at which the spacecraft crosses the y-axis for the
    % first time (necessary to properly calculate periapsis and apoapsis
and
    % ignore intial displacement)
    start_t = 840/dt;

    % Defines lines for periapsis and apoapsis
    max_line=line(main_ax,0,0,'Color','red','LineStyle','--');
    min_line=line(main_ax,0,0,'Color','green','LineStyle','--');

    % Variable that controls the animation speed
    update_step=50;

    for i= t0:update_step:size(t,2)
        if i+update_step<size(z,2)
            % Adds path data points in chunks

addpoints(displacement,z(3,i+1:i+update_step),z(1,i+1:i+update_step));
        else
            addpoints(displacement,z(3,i:end),z(1,i:end));
        end
        if i>=start_t
            % Calculates max and min displacements at current point in path
            [h_max,h1_pos]=max(H(start_t:i));
            [h_min,h2_pos]=min(H(start_t:h1_pos));
            if h_max >max_H(1)
                max_H=[h_max;h1_pos];
                 % Updates apoapsis line if new apoapsis is found
                set(max_line,'XData',[0 z(3,max_H(2)-1+start_t)],'YData',[0
z(1,max_H(2)-1+start_t)]);
            else
            end
            if h_min <min_H(1)
                min_H=[h_min;h2_pos];
                % Updates periapsis line if new periapsis is found
                set(min_line,'XData',[0 z(3,min_H(2)-1+start_t)],'YData',[0
z(1,min_H(2)-1+start_t)]);
            else
            end
        end
        drawnow;
    end
end
```

Boundary Value Problem MATLAB code for the spacecraft (bvpSolver.m)

```
function [A] = bvpSolver(H,a1,a2)
% bvpSolver  Solve a boundary value problem (BVP) and plot the result
%
% Y0 = BVPSOLVER(HDESIRED,A1,A2) finds the necessary final altitude A
% required for an apoapsis of 1200km from the input apoapsis HDESIRED
% and initial altitude inputs of A1 and A2.
% The two altitude guesses A1 and A2 set the altitude boundaries.
% HDESIRED is the necessary apoapsis of 1200km
```

```matlab
% Set initial conditions in seconds
t0 = 0;
dt = 1;
tend = 40000;
a = [a1,a2];
errors=[];
R=6051800;
Hdesired=H+R;

% Sets up input in correct order for ivpSolver
inputs =[t0,dt,tend,0,0,10000000,-11000];

% Inputs the initial conditions and altitude A into IVPSOLVER for A1
% and then A2 to find an apoapsis of H1 and then H2.

% Sets up an anonymous function to return the apoapsis of any z data set
h_max = @(z) max(hypot(z(1,840/dt:end),z(3,840/dt:end)));

inputs(4)=a1;
[~,z] = ivpSolver(inputs);

% ERROR1 computs the error of the output apoapsis compared to
% HDESIRED.
error1 = (h_max(z) - Hdesired)/Hdesired;
errors=[errors,error1];

inputs(4)=a2;
[~,z] = ivpSolver(inputs);

% ERROR2 computs the error of the output apoapsis compared to
% HDESIRED.
error2 = (h_max(z) - Hdesired)/Hdesired;
errors=[errors,error2];

% Tolerances for HFINAL to equal HDESIRED
Upperbound = 1.00000001*Hdesired;
Lowerbound = 0.99999999*Hdesired;

% While loop reduces ERORROR3 in H3 by using STEPRK4 and IVPSOLVER
% until final apoapsis HFINAL is within the tolerances for HDESIRED
m=3;
H=h_max(z);
% Iterates until the H value is between the desired limits
while H > Upperbound || H < Lowerbound
    % Extrapolates a new altitude value from previous values
    a(m) = a(m-1) - ((a(m-1)-a(m-2))/(errors(m-1)-errors(m-2)))*errors(m-
1);
    inputs(4) = a(m);
    [~,z] = ivpSolver(inputs);

    % Calculates apoapsis and error for the altitude value
    H = h_max(z);
    error = ((H-Hdesired)/Hdesired);
    errors = [errors,error];
    m = m + 1;
end
% Returns final altitude value
A = (a(end)-R);
```

MATLAB code for GUI extension (MainGUI.m)

```
function GraphDraw (app,t,z,data)
            % Clears plots on main figure
            cla(app.PathFigure);
            % Sets time variables
            t0=data(1);
            dt = data(2);
            %Caclucates an array of total displacements
            H=hypot(z(1,:),z(3,:));
            %Defines initial conditions of apoapsis and periapsis
            max_H=[0;0];
            min_H=[1e9;0];
            % Gets axis reference of main figure
            main_ax=app.PathFigure;
            % Draws Venus
            rectangle(main_ax,'Position',[-6051800 -6051800 12103600
12103600],'Curvature',[1,1],'FaceColor', '#ae6220');
            % Disables panning and zooming of plot
            disableDefaultInteractivity(main_ax)
            % Defines lines for periapsis and apoapsis
            max_line=line(main_ax,0,0,'Color','red','LineStyle','--');
            min_line=line(main_ax,0,0,'Color','green','LineStyle','--');
            % Specifies the time at which the spacecraft crosses the y-axis for the
            % first time (necessary to properly calculate periapsis and apoapsis
and
            % ignore intial displacement)
            start_t = 840/dt;

            % Since the default method of handling concurrent callbacks in
            % MATLAB is to queue them, this try catch statement works
            % around that and discards multiple button callbacks in case
            % the user calls for too many simulations at once
            try
                % Checks if the user has selected animation to be on or off
                if app.ButtonOn.Value
                    %Animated path plot
                    displacement = animatedline(main_ax);
                    axis(main_ax,[-1e7,1e7,-1e7,1e7],'equal');
                    % Locks axis to square shape
                    pbaspect(main_ax,[1 1 1]);
                    % Gets animation speed from slider value
                    update_step=round(app.Slider.Value+1);
                    for i= t0:update_step:size(t,2)
                        % Adds path data points in chunks
                        if i+update_step<size(z,2)

addpoints(displacement,z(3,i+1:i+update_step),z(1,i+1:i+update_step));
                        else
                            addpoints(displacement,z(3,i:end),z(1,i:end));
                        end
                        if i>=start_t
                            [h_max,h1_pos]=max(H(start_t:i-update_step));
                            [h_min,h2_pos]=min(H(start_t:h1_pos));
                            if h_max >max_H(1)
                                max_H=[h_max;h1_pos];
                                % Updates apoapsis line if new apoapsis is found
                                set(max_line,'XData',[0 z(3,max_H(2)-
1+start_t)],'YData',[0 z(1,max_H(2)-1+start_t)]);
                                % Scales axis to match size of plot whilst
                                % keeping Venus centered and axis square
                                if h_max > (1e7*sind(45))
                                    axis(main_ax,"auto","equal");
                                    x_lim=max(abs(main_ax.XLim));
                                    y_lim=max(abs(main_ax.XLim));
                                    main_ax.XLim=[-x_lim,x_lim];
                                    main_ax.YLim=[-y_lim,y_lim];
                                end
                            else
                            end
                            if h_min <min_H(1)
                                min_H=[h_min;h2_pos];
                                % Updates periapsis line if new periapsis is found
                                set(min_line,'XData',[0 z(3,min_H(2)-
1+start_t)],'YData',[0 z(1,min_H(2)-1+start_t)]);
                            else
                            end
                        end
                        drawnow;
                    end
                else
                    % Holds axis to allow for multiple plots
                    hold(main_ax)
                    % Calculates and draws periapsis and apoapsis lines
                    [h_max,h1_pos]=max(H(start_t:end));
                    [h_min,h2_pos]=min(H(start_t:h1_pos));
                    max_H=[h_max;h1_pos];
                    set(max_line,'XData',[0 z(3,max_H(2)-1+start_t)],'YData',[0
z(1,max_H(2)-1+start_t)]);
                    min_H=[h_min;h2_pos];
                    set(min_line,'XData',[0 z(3,min_H(2)-1+start_t)],'YData',[0
z(1,min_H(2)-1+start_t)]);
                    % Plots path of spacecraft
                    plot(main_ax,z(3,:), z(1,:),'black')
                    % Scales axis to fit path
                    axis(main_ax,"auto","equal");
                    x_lim=max(abs(main_ax.XLim));
                    y_lim=max(abs(main_ax.XLim));
                    main_ax.XLim=[-x_lim,x_lim];
                    main_ax.YLim=[-y_lim,y_lim];
                    hold(main_ax,'off')
                end
            catch
                % Resets the figure if too many callbacks are enqueued
                cla(app.PathFigure)
            end

            % Calculates acceleration of the spacecraft
            ddz = diff(hypot(z(2,:),z(4,:))) / dt;

            % Plots displacement, velocity and acceleration against time on
            % other tabs of program
            plot(app.DisplacementFigure,t,((z(1,:).^2+z(3,:).^2).^0.5),'b')
            plot(app.VelocityFigure,t,((z(2,:).^2+z(4,:).^2).^0.5),'g')
            plot(app.AccelerationFigure,t(2:end),ddz,'r')

            % Updates text boxes with current apoapsis and periapsis values
            app.Apoapsis.Text=(sprintf('%0.2f',(max_H(1)-6051800)/1000)+" km");
            app.Periapsis.Text=(sprintf('%0.2f',(min_H(1)-6051800)/1000)+" km");
        end
    end
```