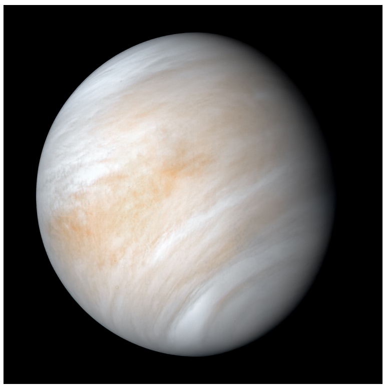
Numerical Modelling of Aerocapture for a Scientific Mission to Venus

Technical Report



[1]

1 Abstract

The task was to model a spacecraft entering the atmosphere of Venus using aerocapture, and find the insertion height needed to reach a 1200km apoapsis. To do this, MATLAB was used to modify pre-existing code and create new functions, such as a shooting method. The model created was able to return a value of 1329.074km, which met all the necessary requirements. Functionality was added to help understand the fuel burn needed at the apoapsis to circularise the orbit, and an interface was incorporated to increase the user interaction and allow editing if any conditions changed. Some understanding of the heating and drag effects was also needed to ensure the validity of the model and where limitations lay.

Nomenclature

ρ	Density	H	Apoapsis Altitude	m_i	Initial Mass
A	Area	h	Periapsis Altitude	R	Radius of Planet
a	Semi-major Axis	I_{sp}	Specific Impulse	r	Radius
C_D	Coefficient of Drag	M	Mass of planet	v_f	Final Velocity
g_0	Standard Gravity	m_f	Final Mass	v_i	Initial Velocity

2 Introduction

2.1 What is Aerobraking?

Aerobraking is the process of using the atmospheric drag to slow down and direct a spacecraft to its desired orbit, in turn using less fuel. The process is much longer than the conventional burn done on orbital insertion but it allows for greater payload to be carried in place of fuel.[2]

2.2 What is Aerocapture?

Aerocapture is the theoretical idea of achieving an orbit of a spacecraft around a planet by using atmospheric drag to slow it down to the optimal speed, instead of using an orbit brake-burn which can last 5-8 minutes. Aerocapture would significantly reduce fuel consumption thus increasing the maximum payload of the spacecraft.

What in particular makes aerobraking and subsequently aerocapture so difficult to achieve is the limitations on simulation and modeling the atmosphere of the planet. With so many variables to account for, there is inherent risk involved which makes it currently only suitable for unmanned space missions. Achieving aerocapture, despite the high risk associated with the maneuver, would be a breakthrough in deep-space exploration.

2.3 Outline of task

The task was to create a mathematical model in MATLAB to evaluate the possibility of using aerocapture to insert the company's scientific satellite into orbit around Venus. To achieve this, a series of objectives were compiled, which would be used to evaluate the progress and quality of work throughout the process.

1. Develop spacecraft guidance model in MATLAB:
    - Develop the basic mathematical model on which to base the software off.
    - Plan how the model will operate through the use of flow charts.
    - Implement plan and mathematical model into software in MATLAB.
2. Other considerations for the spacecraft guidance model:
    - Model burn process.
    - Consider heating and drag effects.
    - Develop a basic GUI for changing simulation values.

3 Mathematical Analysis

The first step of the analysis was to draw a free body diagram (FBD) of the system. Figure 1 (adapted from Figure 2 in the Assignment description [3]) shows that FBD.

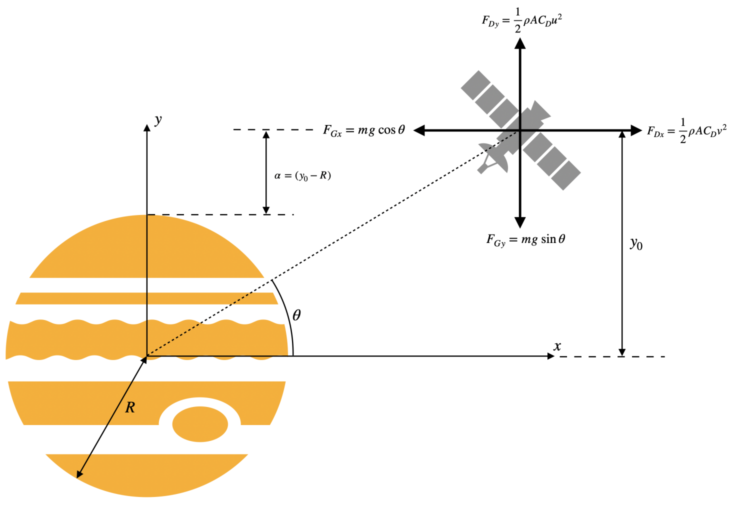


Figure 1: Diagram Showing FBD of the Spacecraft relative to planet

Initially, $mg\sin\theta$ and $mg\cos\theta$ were used as the gravitational force on the spacecraft but it was decided that a more appropriate course of action be to use Newton's law of gravitation, $F_G = G\frac{Mm}{r^2}$.

Newton's 2nd law was used to resolve the forces in the x and y direction:

$$\begin{pmatrix} \ddot{x} + \frac{1}{2m}\rho A C_D(\dot{x}^2+\dot{y}^2)^{\frac{1}{2}}\dot{x} - GM\frac{x}{(x^2+y^2)^{\frac{3}{2}}} = 0, \\ \ddot{y} + \frac{1}{2m}\rho A C_D(\dot{x}^2+\dot{y}^2)^{\frac{1}{2}}\dot{y} - GM\frac{y}{(x^2+y^2)^{\frac{3}{2}}} = 0 \end{pmatrix} \tag{1}$$

In order to evaluate the second order ODE's in Equation 1 numerically, the state variables had to be defined:

$$\begin{aligned} z_1 &= x & z_1' &= x' & z_3 &= y & z_3' &= y' \\ z_2 &= x' & z_2' &= x'' & z_4 &= y' & z_4' &= y'' \end{aligned} \tag{2}$$

Following this, the state variables for the x and y directions were given, through substituting the variables from 2:

For the x direction:

$$\begin{pmatrix} z_1' \\ z_2' \end{pmatrix} = \begin{pmatrix} z_1' = z_2 \\ z_2' = GM\frac{z_1}{(z_1^2+z_3^2)^{\frac{3}{2}}} - \frac{1}{2m}\rho A C_D z_2(z_2^2+z_4^2)^{\frac{1}{2}} \end{pmatrix} \tag{3}$$

For the y direction:

$$\begin{pmatrix} z_3' \\ z_4' \end{pmatrix} = \begin{pmatrix} z_3' = z_4 \\ z_4' = GM\frac{z_3}{(z_1^2+z_3^2)^{\frac{3}{2}}} - \frac{1}{2m}\rho A C_D z_4(z_2^2+z_4^2)^{\frac{1}{2}} \end{pmatrix} \tag{4}$$

An obvious observation is the state variable z_1 appears in the equations for the y direction and vice versa with z_3. This is a product of the angle relative to the spacecraft and planet changing in flight and needed to be reflected in the equation for the gravitational pull.

4 Planning Phase

In order to plan how to implement the software, a high level overview of the form of Figure 2 was developed. This allowed division of tasks and helped targets to be set. Pseudo code was used in parts to compliment Figure 2 so that when coding began, it was simple to understand the process.

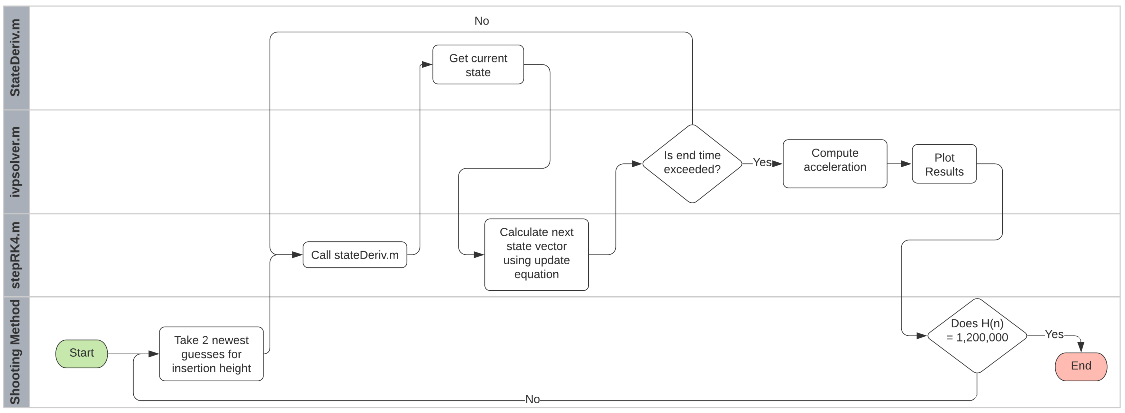


Figure 2: Swim-Lane flow chart showing what functions complete each stage of guidance (NOTE: Flowchart was an initial overview and did not fully reflect actual implementation)

5 Implementation Phase

5.1 Aerocapture Model

5.1.1 Code Modifications

Initially, there was some pre-existing code that had to be modified in order to gain functionality. The state derivative function had originally been meant for only two values of z, rather than the four that were produced by the state variables. Therefore, the dz output had to be modified by adding z_3 and z_4.

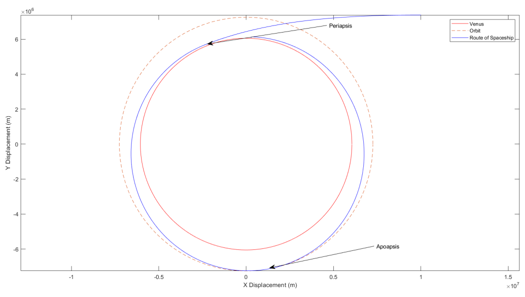
The IVP (Initial Value Problem) solver was also modified. The plot it created had to be re-written so that an outline of Venus and the path of the spacecraft could be seen. The IVP solver was, however, calling the a function which would use Euler's method to solver the equations, rather than the preferable Runge-Kutta (RK4). To solve this, the function for the RK4 method was written and then integrated into the IVP solver.

5.1.2 Shooting Method

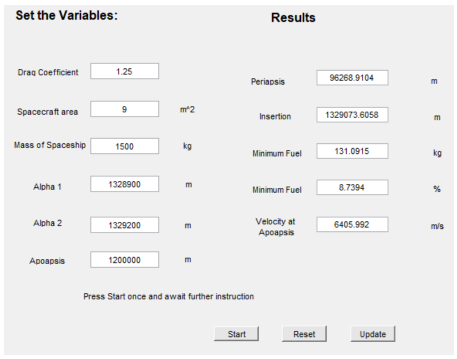
The shooting method could then be used to find the exact insertion height of the spacecraft when the boundary conditions are defined, in this case a maximum radius of 1200km from the surface.

The method specified inputting two values, which would together be able to inform the next guess. To do this, the IVP solver would have to be run for each of the guesses, and the relevant data be taken from it to calculate the next guess. This data was the maximum and minimum distances from the surface of Venus, which could be calculated using inequalities. Assuming that the centre of Venus was at the origin, the periapsis (minimum height) could be defined as the smallest altitude of the rocket once it had passed into negative x values. The apoapsis (maximum height) was the largest altitude of the rocket which had a negative y value. An error could then be found by subtracting 1200km from the apoapsis. These errors were then used to inform the next insertion height.

The code would iterate through this process until the error value was less than one. Once it had achieved this, it would create the path of the spacecraft as seen in Figure 3a. The relevant values were displayed; the insertion height, periapsis, and apoapsis with the speed at this point. The code would then use some of these values to run the *burn* function; used to understand how much fuel is needed during the burn to make the spacecraft orbit at the apoapsis. The value of the periapsis was 96.269km and the insertion height was given to be 1329.074km.



(a) Path of spacecraft around Venus with an apoapsis of 1200km



Set the Variables:

Drag Coefficient	1.25	
Spacecraft area	9	m^2
Mass of Spaceship	1500	kg
Alpha 1	1328900	m
Alpha 2	1329200	m
Apoapsis	1200000	m

Results

Periapsis	96268.9104	m
Insertion	1329073.6058	m
Minimum Fuel	131.0915	kg
Minimum Fuel	8.7394	%
Velocity at Apoapsis	6405.992	m/s

Press Start once and await further instruction

(b) GUI interface

Figure 3: Coded Deliverables

5.2 Graphical User Interface (GUI)

A GUI was added to make the shooting method code more interactive using the in-built function *guide*. As in Figure 3b, the interface was saved as a graphic and the underlying code could be edited to give functionality. It was integrated such that the user could edit the values of variables and see the effects, rather than changing them in the code manually each time. The results were presented to the user when the update button was pressed, meaning all the data was contained in one place.

The GUI was also responsible for creating a dialog when the graph could not be plotted correctly, due to the input values not converging as expected. This was done by checking whether certain variables had a value assigned to them. If there was no value where there should be one, it meant that an input value in the shooting method was not within a range where the problem could be solved.

5.3 Apoapsis Burn Procedure (Circularization Burn)

5.3.1 Mathematical Modelling

The aerocapture procedure predicted the spacecraft to continue on an elliptical orbit, eventually crashing into the planet. Therefore, a move to a circular orbit was needed. To do this, a *circularization burn* (or 'Apogeee kick') is used at the point of the apoapsis (altitude $= H$).

To calculate the target speed of the circular orbit, the *vis-viva equation* (5) was used.

$$v^2 = \mu\left(\frac{2}{r} - \frac{1}{a}\right) \tag{5}$$

Substituting the value resulted in Equation 6:

$$v_f = \sqrt{GM\left(\frac{2}{H+R} - \frac{1}{H+R}\right)} = \sqrt{\frac{GM}{H+R}} \tag{6}$$

A full equation detailing the speed change needed to circularize the orbit (7).

$$\Delta v = v_f - v_i = \sqrt{GM}\left(\sqrt{\frac{1}{H+R}} - \sqrt{\frac{2}{H+R} - \frac{1}{\frac{H+h}{2}+R}}\right) \tag{7}$$

To calculate the fuel required to complete this burn the Tsiolkovsky rocket equation (8) was used:

$$\Delta v = I_{\text{sp}} g_0 \ln\frac{m_i}{m_f} \tag{8}$$

Rearranging for the minimum mass of propellant required:

$$m_{\text{propel.}} = m_i - m_f = m_i - \frac{m_i}{e^{\left(\frac{\Delta v}{I_{\text{sp}} \cdot g_0}\right)}} \tag{9}$$

5.3.2 Numerical Modelling

The known values were then substituted from the problem into Equation 7 to get a theoretical maximum value for ΔV. This was found to be:

$$\Delta v_{\text{theoretical}} = v_f - v_i = 6.693 - 6.411 = 0.283 \text{ km s}^{-1} \text{ (3sf)} \tag{10}$$

However, due to other external factors (such as drag) the MATLAB simulation gave the velocity at the apoapsis as 6.4060 km s^{-1}. Therefore the actual change in velocity is:

$$\Delta v = v_f - v_i = 6.693 - 6.406 = 0.287 \text{ km s}^{-1} \text{ (3sf)} \tag{11}$$

When substituted into Equation 9, it was found that the minimum mass of fuel required, assuming that the apogee kick motor had a specific impulse, I_{sp}, of 320 seconds [1], was as follows:

$$m_{\text{propel.}} = m_i - m_f = 131 \text{ kg or } \approx 9\% \; m_i \tag{12}$$

These calculations were then incorporated into the MATLAB model to output the minimum fuel required to complete the burn and the final orbital velocity. This made the code more extensible.

5.4 Heating on entry into the atmosphere

As the spacecraft enters Venus' atmosphere, much of its kinetic and potential energy is converted to thermal energy. Due its high entry speed, a shock wave is formed at the front of the spacecraft. In the wake of the shock wave, a temperature increase occurs which then heats the surface of the spacecraft due to convection. In these conditions, the material of the spacecraft must be able to withstand high temperatures, with tiles made of special ceramic material designed to burn away as it encounters high temperature plasma flow from the shock wave [5]. The inside of the spacecraft is protected by the change of phase and the convection of the flow away from its surface.

5.5 The aerodynamic model

The aerodynamic model used fundamentally relies on the fact that drag only varies with atmospheric density, which is too simplistic when applied to space. The actual drag coefficient equation for the hypersonic/supersonic transition regime, the regime for space travel at hypersonic speeds, can be seen in equation 13. This equation uses the Sigmoid bridging function, which allows for a mathematical average of the drag coefficients to be found with a continuous and differentiable function, $P(t) = \frac{1}{1+e^{-t}}$. The parameter t is computed through a linear interpolation.

$$t = 20(\text{Ma} - \text{Ma}^{\text{Sup}})/(\text{Ma}^{\text{hyp}} - \text{Ma}^{\text{Sup}}) - 10 \tag{13}$$

Where $\text{Ma}^{\text{Sup}} = 2$ and $\text{Ma}^{\text{hyp}} = 10$ are the selected boundaries of a fully established supersonic and hypersonic regime.

$$C_D = C_D^{\text{Sup}} + (C_D^{\text{hyp}} - C_D^{\text{Sup}})P(t) \tag{14}$$

C_D^{Sup} and C_D^{hyp} are the supersonic and hypersonic drag coefficients computed at the local mach number as if the flow field was fully developed [6]. During atmospheric entry the transonic and subsonic regimes are inconsequential as they only take effect at altitudes below 20km. The spacecraft will rotate around its own centre of mass, causing different regions of the spacecraft to act as the leading edge. Each scenario will

[1]Approximate value is from Leros™ 1b-1c Apogee upper-stage thrusters [4].

have different drag coefficients as the spacecraft tumbles through the atmosphere. These can be found by averaging the drag coefficients with weightings on the numbers of faces,edges and corners.

6 Conclusions

The success of the project can be assessed by referring back to the initial objectives and the project brief. Code was created which was able to successfully solve the ODE's that it was presented with. The model could extract data from the ODE's, such as the periapsis, apoapsis and insertion height, allowing a complete path of the spacecraft to be plotted. Through the addition of information pertaining to the burn process, heating effects and drag differences, and an easier to use program, the model was analysed and improvements were suggested and implemented where possible. One such example of this is the improvement of the GUI; allowing results to be shown within the GUI rather than in the command window. From the model, to reach an apoapsis of 1200 km and insertion height $\alpha = 1329.074$km was needed.

With the successful completion of all the initial objectives, it is an encouraging sign for the future of aerocapture, showing it can be effectively modelled and analysed.

References

[1] NASA. *Contrast-enhanced false color view of Venus from Mariner 10 (Public Domain)*. URL: https://en.wikipedia.org/wiki/Venus#/media/File:PIA23791-Venus-NewlyProcessedView-20200608.jpg. (accessed: 19/11/2020).

[2] Jill Prince, Richard Powell, and Dan Murri. "Autonomous aerobraking: A design, development, and feasibility study". In: *Advances in the Astronautical Sciences* 142 (Jan. 2012).

[3] A. Hunter. "Assignment 2: Numerical Modelling of Aerocapture for a Scientific Mission to Venus". In: *Modelling Techniques 1 (ME20014)* (2020).

[4] MOOG ISP. *Upper Stage Engines*. URL: https://satsearch.s3.eu-central-1.amazonaws.com/datasheets/satsearch_2ko2dr_moog_inc_leros_1b.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256%5C&X-Amz-Credential=AKIAJLB7IRZ54RAMS36Q%5C%2F20201129%5C%2Feu-central-1%5C%2Fs3%5C%2Faws4_request&X-Amz-Date=20201129T143510Z&X-Amz-Expires=600&X-Amz-Signature=4dcc4cbb909ae3efeb914139b5858273722078597f23081671b00c3cf6e605e4%5C&X-Amz-SignedHeaders=host. (accessed: 29/11/2020).

[5] NASA. *Speed Regimes*. URL: https://www.grc.nasa.gov/www/BGH/hihyper.html. (accessed: 30/11/2020).

[6] S.F. Rafano Carná and R. Bevilacqua. "High fidelity model for the atmospheric re-entry of CubeSats equipped with the Drag De-Orbit Device". In: *Acta Astronautica* 156 (2019), pp. 134–156. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2018.05.049. URL: http://www.sciencedirect.com/science/article/pii/S009457651731336X.

Appendices

A State Derivative

```
function dz = stateDeriv(t,z)
% Calculate the state derivative for a spaceship
%
%     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.
M = 4.8675*10^24; % Venus Mass
G = 6.67*10^-11; %Gravitational Constant
m = evalin('base','m'); %Spacecraft Mass called from intialvariables.m
R = 6051.8*10^3; %Radius of Venus
A = evalin('base','A'); % Area of Spacecraft called from intialvariables.m
drag = evalin('base','drag');% Coefficient of Drag called from intialvariables.m
%x = z(1);
%y = z(3);
rp= sqrt((z(1).^2)+(z(3).^2))-(R); %height of the spacecraft
rho = profileVenus(rp);
% State Derivatives
dz1 = z(2);
dz2 = (-G*M*z(1))/(((z(1)^2)+(z(3)^2))^(3/2))
-((drag*sqrt((z(2)^2)+(z(4)^2))*z(2)*A*rho)/(2*m));
dz3 = z(4);
dz4 = (-G*M*z(3))/(((z(1)^2)+(z(3)^2))^(3/2))-
((drag*sqrt((z(2)^2)+(z(4)^2))*z(4)*A*rho)/(2*m));
dz = [dz1; dz2; dz3; dz4];
```

B IVP Solver

```
function [t,z] = ivpsolver(t0,z0,dt,tend)
% 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) = t0;
z(:,1) = z0;

% Continue stepping until the end time is exceeded
n=1;
while t(n) ≤ tend
    % Increment the time vector by one time step
    t(n+1) = t(n) + dt;

    % Apply Runge-Kutta method for one time step
    z(:,n+1) = stepRungeKutta(t(n), z(:,n), dt);

    n = n+1;
end
% Compute acceleration from velocity (not needed for solving the
% problem, but nice to have for completeness)
ddz = diff(z(2,:)) / dt;

rp=6051.8*10^3 ; % Plot of the planet Venus
th=0:0.01:2*pi;
xp = rp*cos(th);
yp =rp*sin(th);

rp1=6051.8*10^3 +1200000; %1200km orbit trail
th=0:0.01:2*pi;
xp1 = rp1*cos(th);
yp1 =rp1*sin(th);

figure(1)
plot(xp,yp,'r');
hold on
plot(xp1,yp1, '--')
plot(z(1,:),z(3,:),'b');
hold off
axis('equal')
```

```
ylabel('Y Displacement (m)')
xlabel('X Displacement (m)')
legend('Venus','Orbit','Route of Spaceship')
```

C Runge-Kutta Method

```
function znext = stepRungeKutta(t,z,dt)
% stepRungeKutta    Compute one step using the Runge-Kutta 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(t,z);

% Calculate the next state vector from the previous
A = dt*stateDeriv(t,z);
B = dt*stateDeriv((t+(dt/2)),(z+(A/2)));
C = dt*stateDeriv((t+(dt/2)),(z+(B/2)));
D = dt*stateDeriv((t+dt),(z+C));

znext = z + ((1/6)*(A+(2*B)+(2*C)+D));
```

D Shooting Method

```
function [alpha, Apoapsis]= shootingmethod(alpha1,alpha2);

H=evalin('base', 'H'); % calling in the defined apoapsis to complete the BVP
%Guess 1
R=6051.8*10^3; %radius of planet
[t,z] = ivpsolver(0,[10000000,-11000,alpha1+R,0],1,8500);
x = z(1,:);
y = z(3,:);
altitude = -(R)+(sqrt((x).^2+(y).^2));
MaxHeight1 = max(altitude(y<0));
MinHeight1 = min(altitude(x<0));
Error1= MaxHeight1-H ;

TF1=isempty(MaxHeight1); % isempty checks whether a variable is an empty array
                          %and if so will show an error message, otherwise the
                          % function will continue
 if TF1==1;
 d = dialog('Position',[300 300 250 150],'Name','Error');
    txt = uicontrol('Parent',d,...
           'Style','text',...
           'Position',[20 80 210 40],...
           'String','Your value for Alpha1 is out of range, please choose a ...
               different height.');

btn = uicontrol('Parent',d,...
           'Position',[89 20 70 25],...
           'String','Okay',...
           'Callback','delete(gcf)');

 else


     %Guess 2
R=6051*10^3;
[t,z] = ivpsolver(0,[10000000,-11000,alpha2+R,0],1,8500);
x = z(1,:);
y = z(3,:);
altitude = -(R)+(sqrt((x).^2+(y).^2));
MaxHeight2 = max(altitude(y<0));
MinHeight2 = min(altitude(x<0));
Error2 = MaxHeight2-H;

TF2=isempty(MaxHeight2);

```

```
if TF2==1;
 d = dialog('Position',[300 300 250 150],'Name','Error');
    txt = uicontrol('Parent',d,...
            'Style','text',...
            'Position',[20 80 210 40],...
            'String','Your value for Alpha2 is out of range, please choose a ...
                different height.');

btn = uicontrol('Parent',d,...
            'Position',[89 20 70 25],...
            'String','Okay',...
            'Callback','delete(gcf)');

 else
%3rd Guess
Error_a = Error1;
Error_b = Error2;
alpha_a = alpha1;
alpha_b = alpha2;
  while abs(Error_b) > 1
      alpha_c = alpha_b-Error_b*((alpha_b-alpha_a)/(Error_b-Error_a));
      alpha_a = alpha_b;
      alpha_b = alpha_c;
      [t,z] = ivpsolver(0,[10000000,-11000,alpha_b+R,0],1,8500);
      x = z(1,:);
      y = z(3,:);
      altitude = -R+(sqrt((x).^2+(y).^2));
      MinHeight = min(altitude(x<0));
      MaxHeight = max(altitude(y<0));
      Error_c =  MaxHeight-H;
      Error_a =  Error_b;
      Error_b =  Error_c;
      dx=z(2,:);
      dy=z(4,:);
      velocity= sqrt(dx.^2+dy.^2);
      MinVel=min(velocity(y<0));


  end
  Apoapsis= MaxHeight;
  TF3=isempty(Apoapsis);

if TF3==1;
 d = dialog('Position',[300 300 250 150],'Name','Error');
    txt = uicontrol('Parent',d,...
            'Style','text',...
            'Position',[20 80 210 40],...
            'String','At least one of your inital guesses is out of range, please ...
                choose a different height ');

 btn = uicontrol('Parent',d,...
            'Position',[89 20 70 25],...
            'String','Okay',...
            'Callback','delete(gcf)');


else
  burn(MinVel./1000,MaxHeight./1000,MinHeight./1000)  ; % runs burn function
  % The variables are assigned to the base workspace, so they can be picked
  % up by the intialvariables.m workspace
  assignin('base','periapsis',MinHeight)
  assignin('base','alpha',alpha_b)
  assignin('base','speed',MinVel)
  assignin('base','apoapsis',MaxHeight)
  %The push button updates values so the relevant time this is needed to be
  %done is specified with the instructions below.
  d = dialog('Position',[300 300 250 150],'Name','Instruction');
    txt = uicontrol('Parent',d,...
            'Style','text',...
            'Position',[20 80 210 40],...
            'String','Please press Update on the main controller to see results');

 btn = uicontrol('Parent',d,...
            'Position',[89 20 70 25],...
            'String','Okay',...
            'Callback','delete(gcf)');

end


 end
end
```

E Burn

```
function [v_final_km,min_fuel_kg,deviation,min_fuel_percent] = burn(v_i,H,h)
% burn Calculate burn fuel mass needed
%
%     Uses the vis-viva equation and rocket equation to calculate the
%     mass needed for a apogee burn (circulization burn).

% constants
    % Orbital
    G = 6.67408E-20;  % Gravitaional constant in km^3 kg^-1 s^-2
    M = 4.8675E24; % Mass of planet in kg
    R = 6051.8; % Radius of planet in km

    % Fuel / Rocket equation
    SpImp = 320; % Specific Impulse in seconds
    m_i = evalin('base', 'm'); % Spacecraft initial mass in kg
    g = 9.81E-3; % Standard gravity in km s^-1

% Calculate Δ v (theoretical):
Δ_v_theoretical=sqrt(G*M)*(sqrt(1/(H+R))-sqrt(2/(H+R)-1/(((H+h)/2)+R)));

% Calculate actual Δ_v using actual speed v_i from simulation
Δ_v_actual=sqrt((G*M)/(H+R)) - v_i;

    % calculates deviation in speed from theoretical to actual
    deviation=Δ_v_theoretical-Δ_v_actual;

    % Remembers final orbit velocity
    v_final_km = sqrt((G*M)/(H+R));

% Calculate min mass of propelant required
min_fuel_kg=m_i - m_i/(exp(Δ_v_actual/(g*SpImp)));

min_fuel_percent=min_fuel_kg/m_i * 100;

assignin('base','minfuel',min_fuel_kg)
assignin('base','minfuelp',min_fuel_percent)
end
```

F Initial Variables

```
function varargout = intialvariables(varargin)
% INTIALVARIABLES MATLAB code for intialvariables.fig
%      INTIALVARIABLES, by itself, creates a new INTIALVARIABLES or raises the ...
    existing
%      singleton*.
%
%      H = INTIALVARIABLES returns the handle to a new INTIALVARIABLES or the ...
    handle to
%      the existing singleton*.
%
%      INTIALVARIABLES('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTIALVARIABLES.M with the given input arguments.
%
%      INTIALVARIABLES('Property','Value',...) creates a new INTIALVARIABLES or ...
    raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before intialvariables_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to intialvariables_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help intialvariables

% Last Modified by GUIDE v2.5 01-Dec-2020 16:08:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @intialvariables_OpeningFcn, ...
                   'gui_OutputFcn',  @intialvariables_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before intialvariables is made visible.
function intialvariables_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to intialvariables (see VARARGIN)

% Choose default command line output for intialvariables
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes intialvariables wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = intialvariables_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double


% --- Executes during object creation, after setting all properties.
```

```matlab
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
drag =str2double(get(handles.edit1, 'String'));
assignin('base','drag',drag)
A =str2double(get(handles.edit2, 'String'));
assignin('base','A',A)
m =str2double(get(handles.edit3, 'String'));
```

```matlab
assignin('base','m',m)
H =str2double(get(handles.edit6, 'String'));
assignin('base','H',H)
alpha1=str2double(get(handles.edit4, 'String'));
alpha2=str2double(get(handles.edit5, 'String'));
shootingmethod(alpha1, alpha2)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.edit1, 'string', '1.25')
set(handles.edit2, 'string', '9')
set(handles.edit3, 'string', '1500')
set(handles.edit4, 'string', '1328900')
set(handles.edit5, 'string', '1329200')
set(handles.edit6, 'string', '1200000')
set(handles.edit7, 'string', '0')
set(handles.edit8, 'string', '0')
set(handles.edit9, 'string', '0')
set(handles.edit10, 'string', '0')
set(handles.edit11, 'string', '0')





function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double




% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a double




% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a double




% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
```

```matlab
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a double





% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a double



% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
periapsis=evalin('base', 'periapsis');
string1 = periapsis; set(handles.edit7, 'String', num2str(string1));
alpha=evalin('base', 'alpha');
string2 = alpha; set(handles.edit8, 'String', num2str(string2));
minfuel=evalin('base', 'minfuel');
string3 = minfuel; set(handles.edit9, 'String', num2str(string3));
minfuelp=evalin('base', 'minfuelp');
string4 = minfuelp; set(handles.edit10, 'String', num2str(string4));
speed =evalin('base', 'speed');
string5 = speed; set(handles.edit11, 'String', num2str(string5));
```