Rocket Flight Simulation in MATLAB/SIMULINK Environment

Technical Report · January 2021

DOI: 10.13140/RG.2.2.11008.35846

1 author:

Emre Sayin
Istanbul Technical University
12 PUBLICATIONS **3** CITATIONS

UUM517E - Spacecraft Dynamics

2020-21 Fall - Project

Emre SAYIN

Lecturer: Assit. Prof. Dr. Cuma YARIM

ROCKET FLIGHT SIMULATION IN MATLAB/SIMULINK ENVIRONMENT

Contents

1 Introduction

Aim of this project is to find the position and velocity variation, Kepler elements, ground track, orbit in 3-D space, range, time of flight, maximum dynamic pressure, maximum velocity, landing coordinates of a rocket flight in which the initial parameters of the rocket are given that launched from a known coordinate with given velocity, azimuth and elevation angle. There is also a ground station tracking the rocket during its flight.

Assumptions of the project are:

- Earth is perfectly spherical.
- There is no nutation on spinning of the Earth.
- Angular velocity of the Earth is constant.
- The atmosphere rotates with the same angular velocity of Earth.
- Thrust force acts on velocity direction.
- There is no lift force produced by rocket.
- Drag coefficient of the rocket is constant.
- Reference area of the rocket is constant.
- Rocket motor produces a constant and instantaneous exhaust velocity.
- Exhaust velocity of the rocket becomes zero instantaneously when the fuel ran out.

While executing the solution, curvature and rotation of the Earth, variation of gravitational acceleration, atmospheric drag and thrust force of the rocket are taken into account.
In order to solve the problem, MATLAB and Simulink were used together. Constants and parameters are defined in MATLAB script, mass variation of the rocket, variation of atmospheric density, forces acting on rocket and equations of motion are calculated in Simulink environment. Final results are found in MATLAB script.

2 Coordinate Transformations

The initial position parameters of the rocket are given in Geographic Spherical Coordinate System (LLA) as latitude, longitude and height. Furthermore, the initial velocity parameters are given in Topocentric Horizon (TH) coordinate system. However, in this project, the equations of motion are written for an inertial coordinate system. Thus, given initial parameters need to be converted into the Earth-Centered Inertial (ECI) coordinate system in order to work on an inertial system.

Firstly, initial LLA positions need to be converted into ECEF coordinate system and then, it needs to be converted into ECI coordinate system. For the initial velocity parameters, firstly, values

need to be converted into ECEF coordinate system from TH coordinate system and then, it can be converted into ECI coordinate system.

For this project, conversion from LLA to ECEF coordinates can be basically done by converting the spherical coordinates into cartesian coordinate system since the LLA parameters are basically parameters of the spherical coordinate system given in Figure 1.

In order to convert LLA positions into ECEF positions, a MATLAB function was written called "LLAtoECEF", given in Appendix A.1.

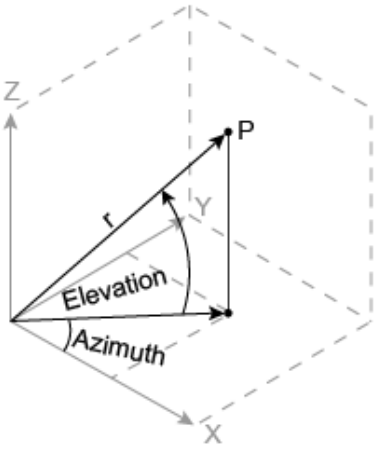


Figure 1: Used notations for spherical coordinate system[2]

Secondly, initial velocity parameters given in TH coordinate system need to converted into ECEF coordinates. The visualization of TH coordinate system is given in Figure 2. The TH coordinate system consists of a vector with azimuth (A) and elevation (a) angles where the azimuth angle is positive clockwise angle from the north between 0 and 360 deg. Elevation angle is the angle measured from horizontal line of sight between 0 to 180 deg. If the length of a vector in TH coordinate system is known as V_0, vector in TH can be constructed as:

$$\vec{V_0} = V_0 \cos a \sin A\hat{\boldsymbol{i}} + V_0 \cos a \cos A\hat{\boldsymbol{j}} + V_0 \sin a\hat{\boldsymbol{k}} \tag{1}$$

Conversion from TH to ECEF can be executed by a sequence of rotations, which can be presented as a uniqe transformation matrix [4]:

$$Q = \begin{bmatrix} \text{-sin}\theta & \text{-sin}\phi\text{cos}\theta & \text{cos}\phi\text{cos}\theta \\ \text{cos}\theta & \text{-sin}\phi\text{sin}\theta & \text{cos}\phi\text{sin}\theta \\ 0 & \text{cos}\phi & \text{sin}\phi \end{bmatrix} \tag{2}$$

Where θ is longitude and ϕ is the latitude angle. In order to convert TH to ECEF coordinate system, a MATLAB function named "topo2geo" is written given in Appendix A.2. The function constructs the initial velocity matrix in TH by using Equation 1 and converts it into ECEF by using the transformation matrix given in Equation 2.

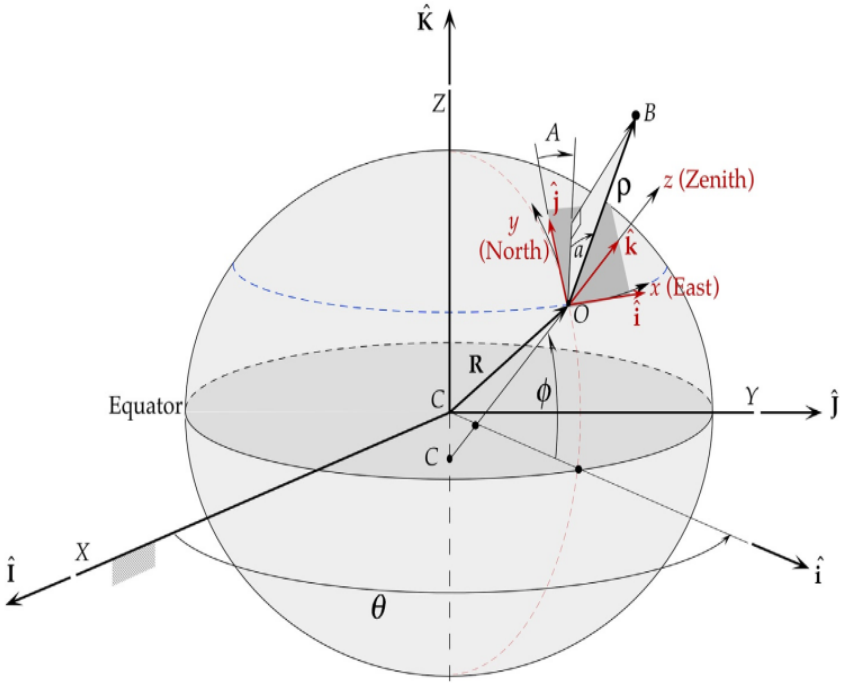


Figure 2: Visualization of Topocentric Horizon Coordinate System with ECEF Coordinate System[4]

The conversion from ECEF into TH coordinate system can be executed by multiplying the transpose of matrix Q given in Equation 2. Resultant matrix in TH coordinate system will be like the one in Equation 1. By using Equation 1, one can find the azimuth and elevation angles for TH coordinate system. In order to execute the conversion from ECEF into TH coordinate system, a MATLAB function named "ECEF2topo" is written and given in Appendix A.3. This function will be used particularly for finding the elevation and azimuth angles between the tracking antenna of ground station and the rocket.

After converting LLA and TH coordinate systems into ECEF for the rocket, now both position and velocity vectors are in ECEF coordinates. In order to convert ECEF into ECI coordinates, the MATLAB function "ECEFtoECI" is taken from [3] and modified. The modified function is given in Appendix A.4. ECEFtoECI function requires the ECEF position-velocity vectors and Julian Day as input. In order to find the Julian day of given time, two MATLAB functions named "J0" and "getJday" are taken from [4] and implemented into the main code. The function "getJday" computes the Julian day of given input time. The Julian day value is used inside of the "ECEFtoECI" function in order to find the rotation angle that is needed for conversion of ECEF into ECI.

In order to convert the ECI into ECEF, a function taken from [5] is modified and constructed as a MATLAB Function Block. The view of ECItoECEF block is given in Figure 4; content of the block is given in Appendix A.5 Moreover, another MATLAB Function block is written in order to convert ECI into LLA directly. View of the ECIconvLLA function is given in Figure 3; the content of ECIconvLLA function is given in Appendix A.6. The function ECIconvLLA is a modified combination of ECI into ECEF and ECEF to LLA conversions. The modified ECI to

ECEF function is taken from [5] and ECEF to LLA conversion is basically done by converting cartesian coordinates into spherical coordinates by "cart2sph" command of MATLAB.

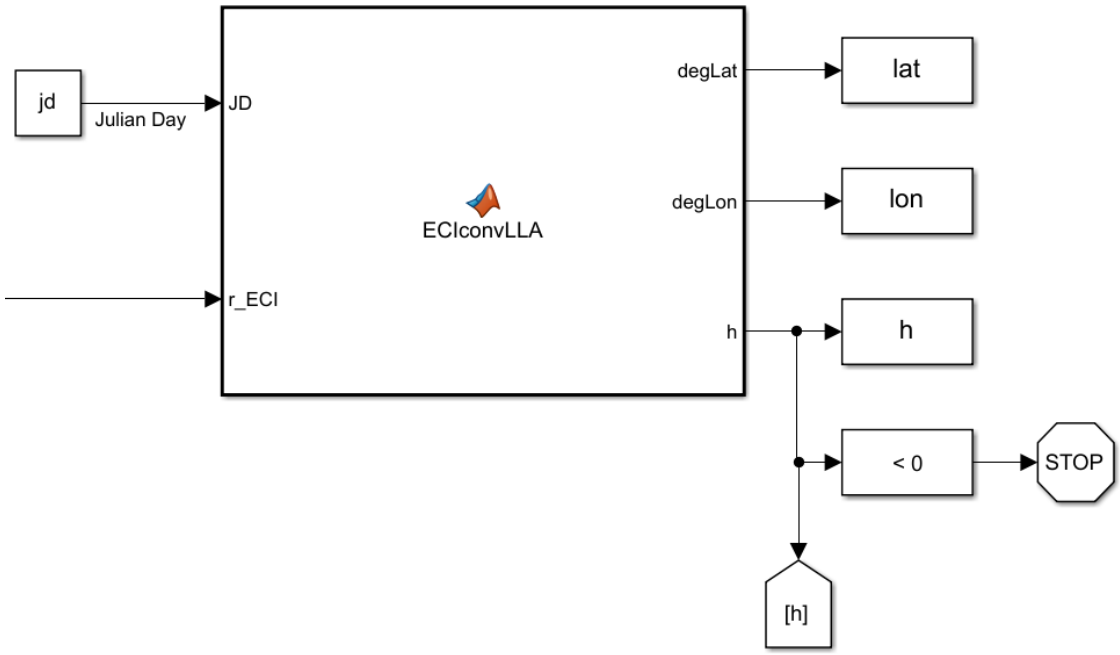


Figure 3: View of the ECIconvLLA function block

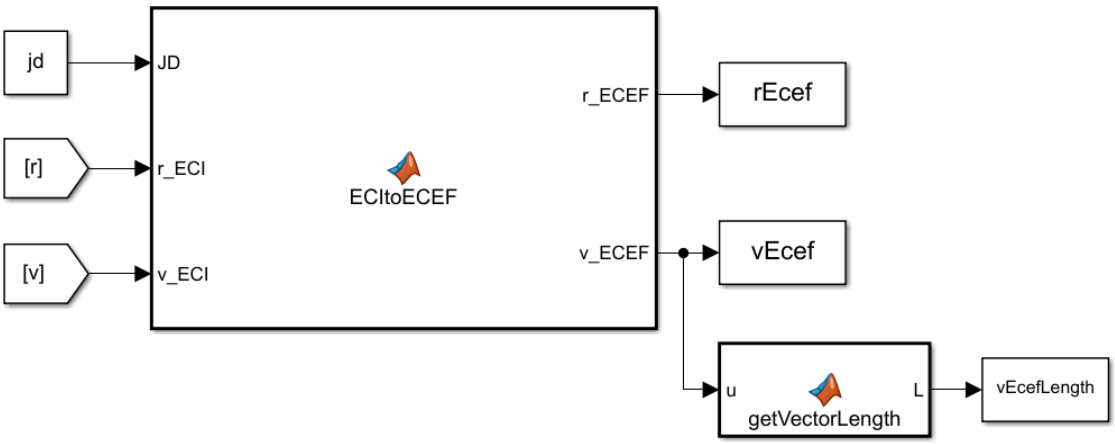


Figure 4: View of the ECItoECEF function block

3 Variation of Atmospheric Density

It is known that the atmospheric density changes with increasing altitude. However, it is also known that this variation is not linear. Thus, there is not an exact equation that calculates the atmospheric density in any altitude. In order to find the atmospheric density, the atmospheric model is taken from [1]. According to [1], atmospheric model of the Earth can be constructed as three different equations in which the first equation is valid up to 11000 km, the second equation is valid between 11000 and 25000 meters, the third equation is valid for the altitudes above 25000 meters. The euqations are:

For $h > 25000$ (Upper Stratosphere)

$$T = -131.21 + 0.00299h$$

$$p = 2.488 \times \left[\frac{T + 273.1}{216.6}\right]^{-11.388} \tag{3}$$

$$\rho = \frac{p}{0.2869(T + 273.1)}$$

For $11000 < h < 25000$ (Lower Stratosphere)

$$T = -56.46$$

$$p = 22.65e^{(1.73 - 0.000157h)} \tag{4}$$

$$\rho = \frac{p}{0.2869(T + 273.1)}$$

For $h < 11000$ (Troposphere)

$$T = 15.04 - 0.00649h$$

$$p = 101.29 \times \left[\frac{T + 273.1}{288.08}\right]^{5.256} \tag{5}$$

$$\rho = \frac{p}{0.2869(T + 273.1)}$$

These equations are modeled in Simulink environment. Model is given in Figure 5

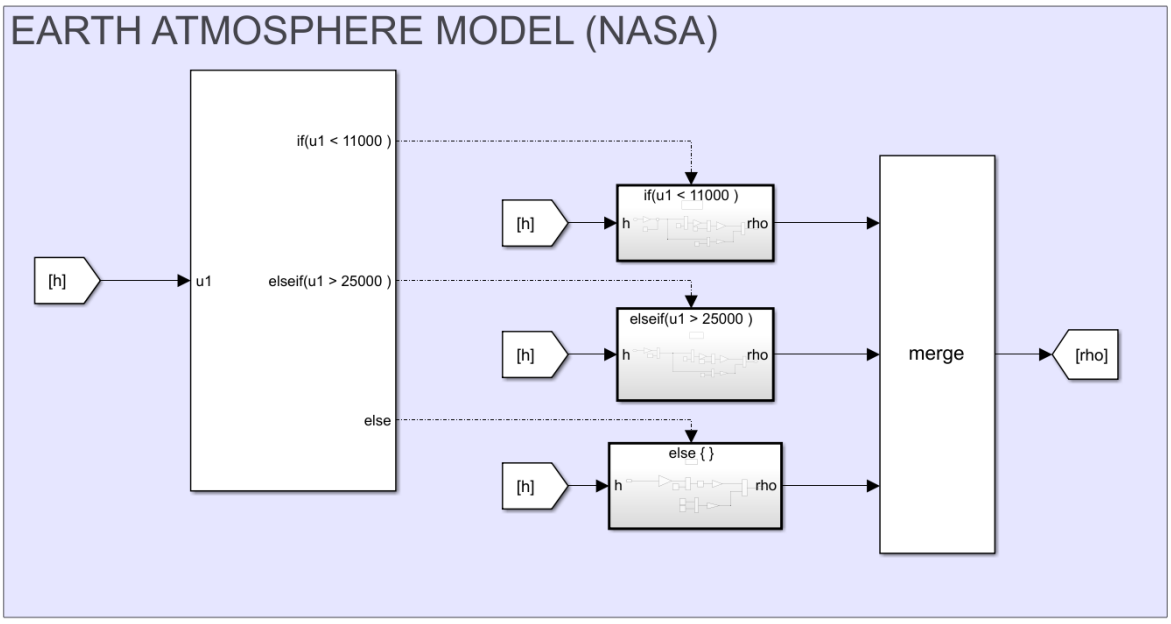


Figure 5: Atmospheric model in Simulink

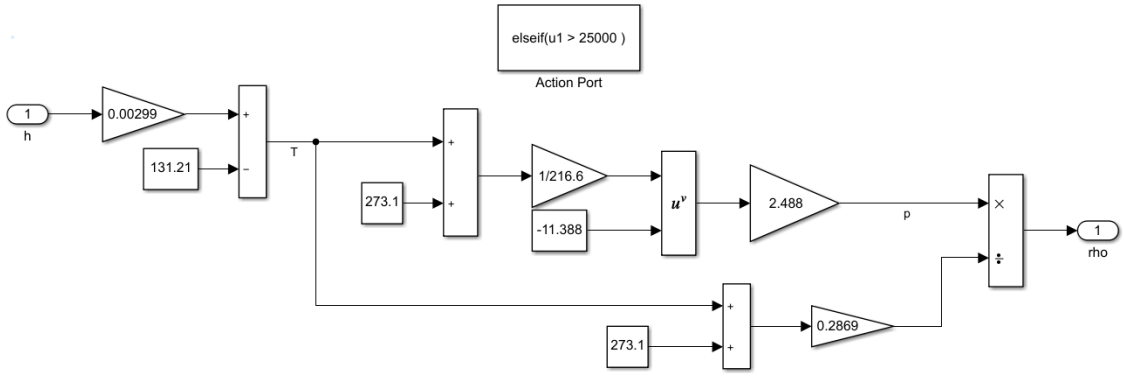


Figure 6: Modeling of Equation 3

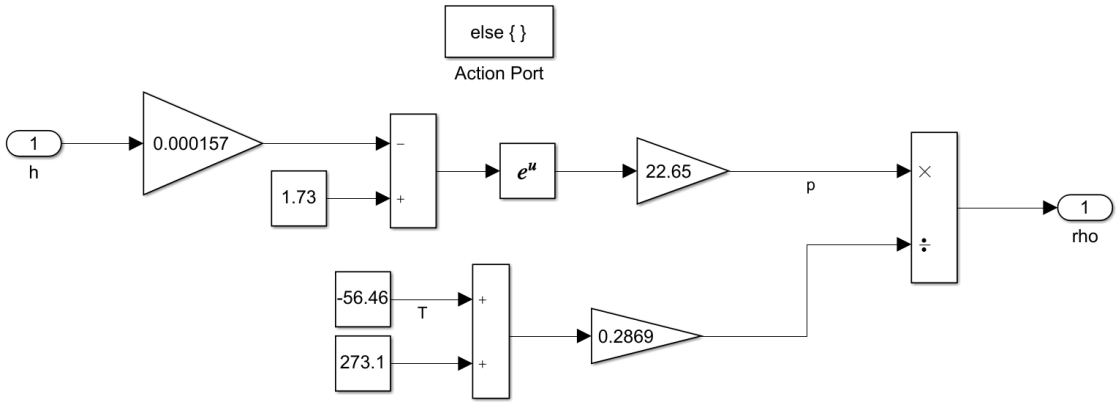


Figure 7: Modeling of Equation 4

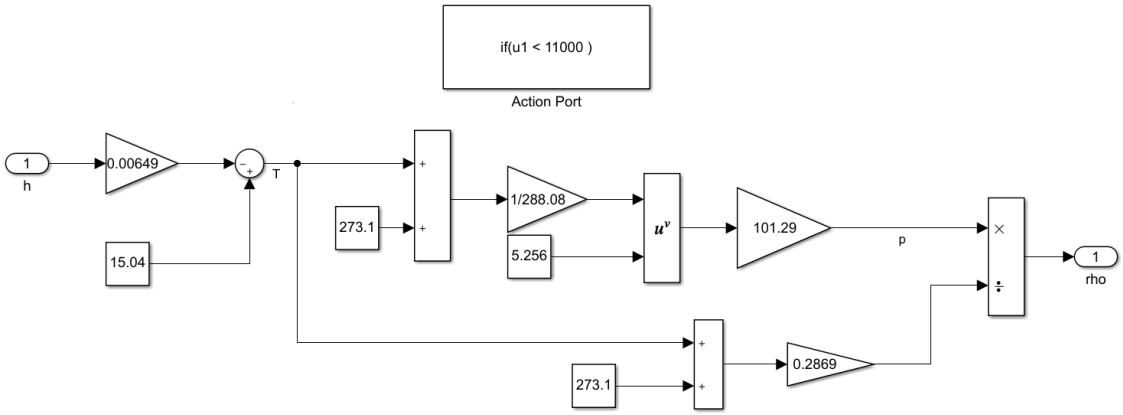


Figure 8: Modeling of Equation 5

4 Mass Change of the Rocket

The mass of the rocket will not remain constant since the motor throws out the propellant for a while. Mass change of the rocket can be calculated from:

$$m = m_0 + \dot{m}t \tag{6}$$

Where m_0 is initial mass of the rocket, $\dot{m}$ is mass flow rate, and t is the time elapsed. Initial mass of the rocket is:

$$m_0 = m_{fuel} + m_f \tag{7}$$

Where m_{fuel} is mass of the fuel and m_f is the final mass of the rocket.

Mass of the rocket will decrease until the fuel runs out. After that, mass will remain constant and there will be no thrust force as a result of this. Mass change model of the rocket is given in Figure 9. If-action subsystem block in Figure 9 gives the same input as output.

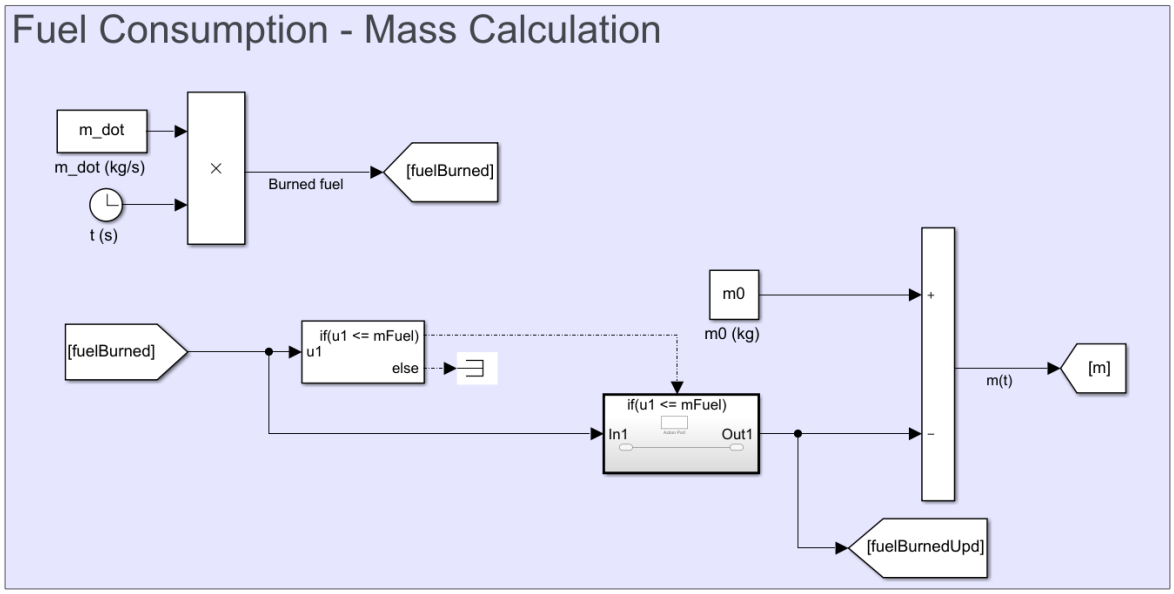


Figure 9: Mass change modeling of the rocket

5 Forces Acting on Rocket

5.1 Gravitational Force

During the flight, a gravitational force acts on the rocket. This force can be calculated as:

$$\vec{F}_g = -\frac{m\mu}{r^3}\vec{r} \tag{8}$$

Where μ is gravitational constant and $\vec{r}$ is the position vector. Simulink model of Equation 8 is given in Figure 10.

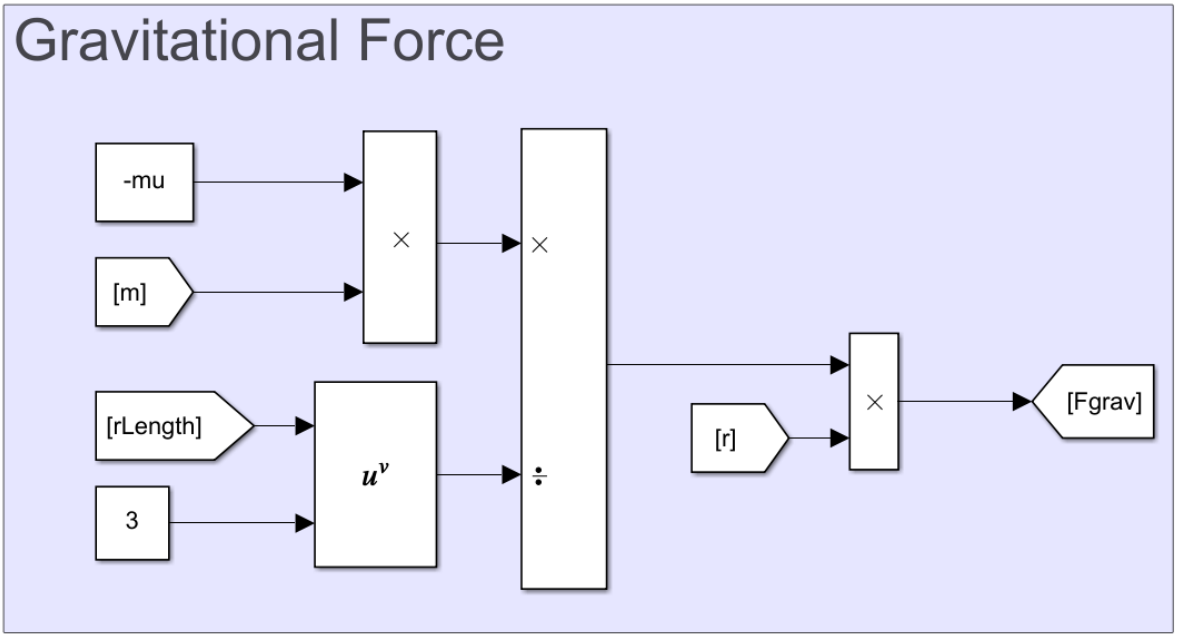


Figure 10: Modeling of gravitational force

5.2 Thrust Force

Motor of the rocket will produce a thrust force. If one assumes that the thrust force acts on the direction of rocket relative velocity vector, $\vec{v}_{rel}$, where.

$$\vec{v}_{rel} = \vec{v} - \vec{v}_{atm} = \vec{v} - \vec{\omega}_e \times \vec{r} \tag{9}$$

Where $\vec{v}$ and $\vec{r}$ are the velocity and position vectors in inertial frame, $\vec{\omega}_e$ is the angular velocity vector of the Earth. Construction of the relative velocity vector in Simulink is given in Figure 11. The content of "getVectorLength" function is given in Appendix A.7. "Normalize" block gives the unit vector on the direction of input vector as the output.

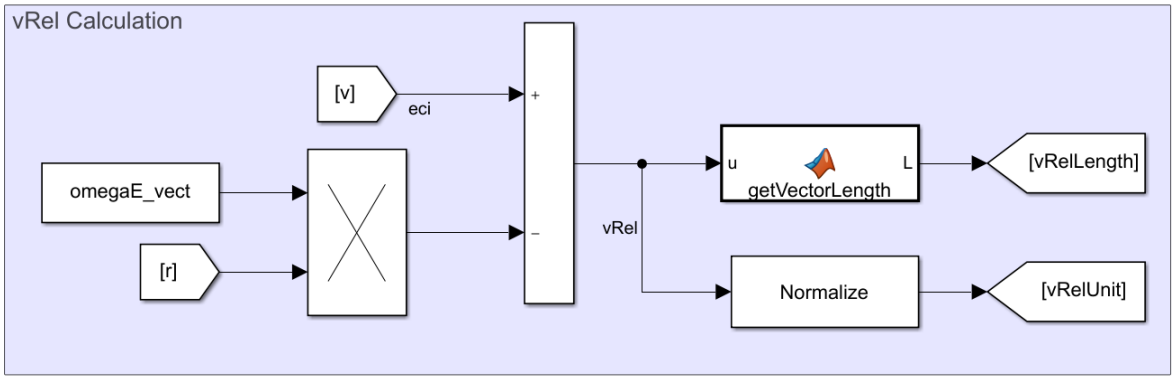


Figure 11: Calculation of relative velocity vector in Simulink

Magnitude of the thrust force can be expressed as:

$$F_t = \dot{m}c \tag{10}$$

Where $\dot{m}$ is mass flow rate of the rocket and c is the exhaust velocity. Both of them are assumed to be constant for this project. Since the direction of the thrust vector is on the positive relative velocity vector, Equation 10 should be multiplied by unit vector on v_{rel} direction. Thus, the thrust vector can be found as:

$$\vec{F_t} = \dot{m}c\frac{\vec{v}_{rel}}{|v_{rel}|} \tag{11}$$

The rocket will produce thrust force until its fuel runs out. After that, magnitude of thrust force will be zero. The Simulink model of the thrust force change is given in Figure 12. According to model, the thrust force becomes zero when the burned fuel reaches the total fuel mass, which is executed by using the "switch" block.The parameter "vRelUnit" is the unit vector in relative velocity direction.

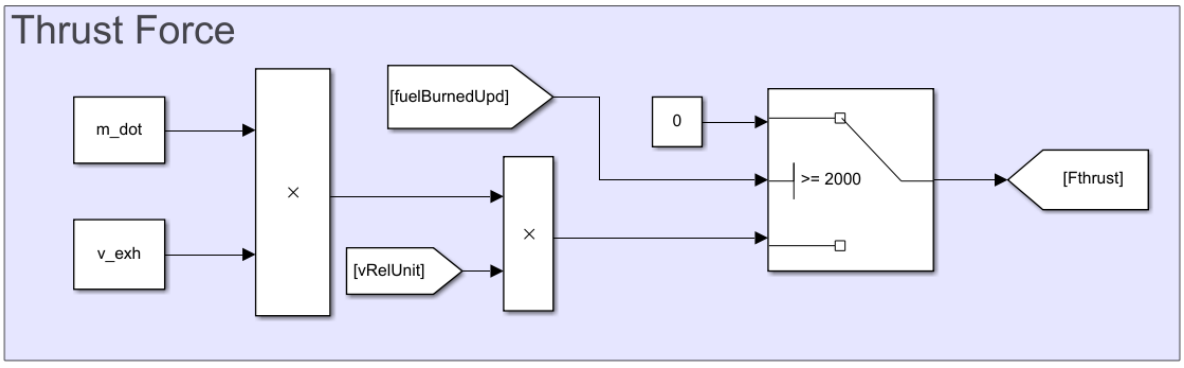


Figure 12: Modeling of thrust force

5.3 Drag Force

Since the rocket moves in air, there will be a resistance force due to this movement. Force produced by the resistance of air is called "drag force". The direction of drag force vector will be on the reverse direction of relative velocity vector. Drag force vector can be calculated as:

$$\vec{F}_d = -\frac{1}{2}\rho v_{rel}^2 c_d A \frac{\vec{v}_{rel}}{|v_{rel}|} \tag{12}$$

Where ρ is the air density (taken from atmospheric model), c_d is drag coefficient of the rocket and A is the cross sectional area of the rocket that faces with air. During the flight, c_d and A are assumed to be constant. Simulink model of Equation 12 is given in Figure 13. Furthermore, dynamic pressure, which affects the loads acting on the rocket during flight, can be defined as:

$$p_{dyn} \equiv \frac{1}{2}\rho v_{rel}^2 \tag{13}$$

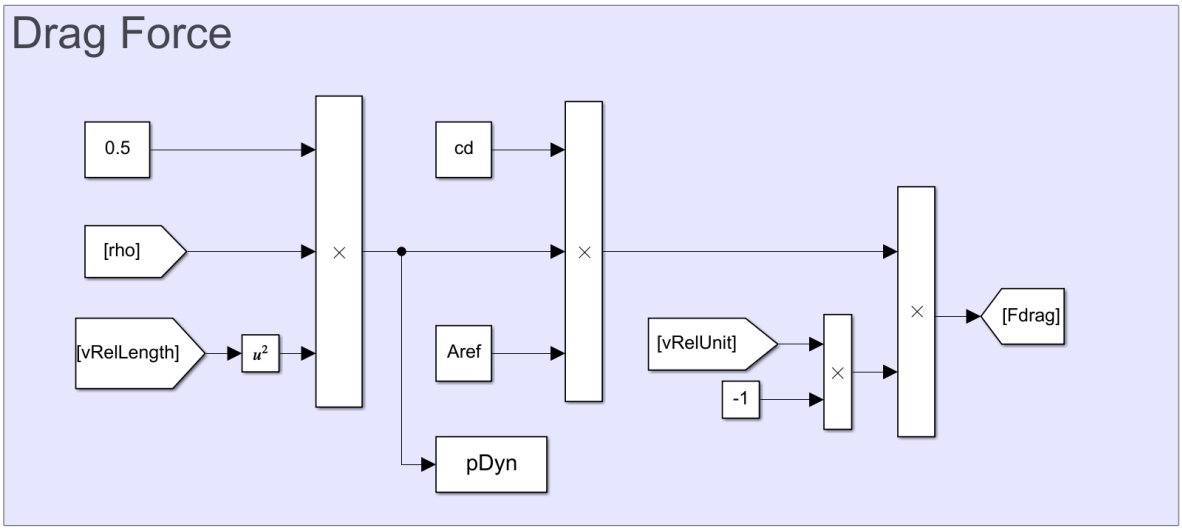


Figure 13: Modeling of drag force

6 Equations of Motion

According to Newton's 2^{nd} law, for an inertial frame, the net force acting on the rocket will be equal to mass of the rocket multiplied by the second derivative of position with respect to time, which can be formulated as:

$$\vec{F}_{net} = m\frac{d^2\vec{r}}{dt^2} \tag{14}$$

For the rocket in this project, Equation 14 can be modified by inserting gravitational, thrust and drag forces:

$$\vec{F}_g + \vec{F}_t + \vec{F}_d = m\frac{d^2\vec{r}}{dt^2} \tag{15}$$

Thus:

$$\frac{d^2\vec{r}}{dt^2} = \frac{\vec{F}_g + \vec{F}_t + \vec{F}_d}{m} \tag{16}$$

One can obtain the velocity and position vector of the rocket during flight, by integrating Equation 16 two times with given initial parameters of position and velocity. Integrating one time gives the velocity vector $\vec{v}$, and two times gives the position vector $\vec{r}$. The Simulink model of equations of motion is given in Figure 14.

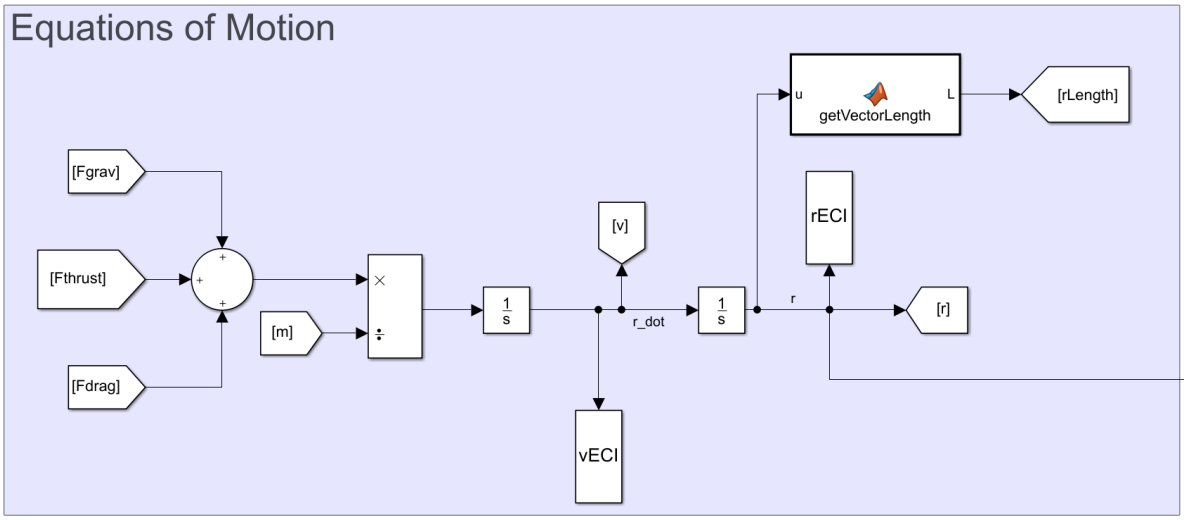


Figure 14: Modeling of equations of motion

Finally, obtained initial position and velocity vectors in ECI coordinates are sent into Simulink file. After the calculations in Simulink, obtained position and velocity vectors during flight are sent into MATLAB workspace by using "To Workspace" blocks as "rECI" and "vECI" that shown in Figure 14. As the last operation in Simulink, the position vector obtained in ECI coordinates are converted into LLA coordinates by using the MATLAB Function Block "ECIconvLLA" given in Figure 3. Finally, a "Compare to Constant" block were inserted after height value which says that "stop the simulation if height starts to become negative".

7 Output Calculations

Finally, by using the values sent into MATLAB workspace, the desired output calculations are executed. Astronomical constants are taken from [6].

- Total flight time of the rocket is found by taking the total simulation time since the simulation stops when the rocket landed.
- Time vs position and velocity vector components in ECI coordinate system and altitude variation are plotted.
- Time vs Kepler elements during the flight are plotted. The modified function that calculates Kepler elements for a given state vector named "coe_from_sv" is taken from [4], given in Appendix A.8.
- Ground track of the rocket is plotted by plotting longitude vs latitude.
- Orbit of the rocket is plotted in a 3-dim space in ECEF coordinate system.
- The highest point of the orbit is found by finding the maximum value of height.
- The fastest point of the orbit is found by calculating the time change of velocity length in ECI coordinate and finding the maximum point of it.
- The maximum dynamic pressure point of the orbit is found by the maximum value of the pDyn parameter taken from Simulink given in Figure 13. Time vs dynamic pressure is plotted.
- Landing point of the rocket is found by converting ECI into LLA coordinate system (See Appendix A.6) and taking the last latitude and longitude values of LLA parameters from MATLAB workspace.
- Range of the flight is found by calculating the angle between initial and final position vectors of the rocket in ECEF coordinate system and multiplying it with the radius of Earth since the Earth is assumed the be spherical.
- Azimuth and elevation angles of tracking antenna are firstly found by calculating the position vector between the ground station and the rocket in ECEF and then converting it into TH coordinate system. (See Appendix A.3)

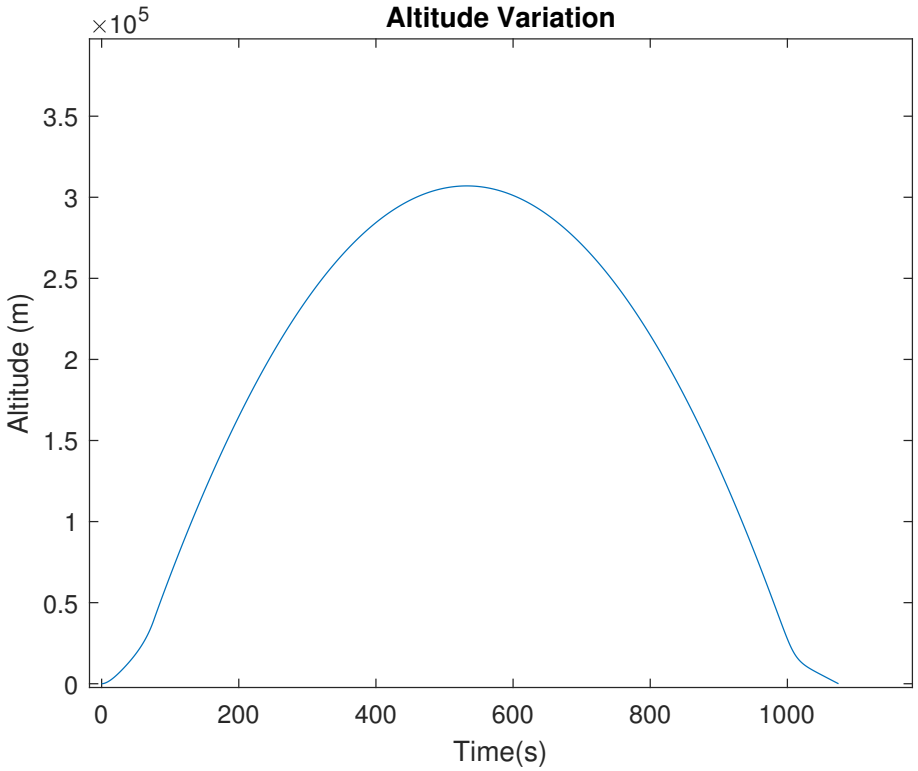
The main MATLAB code is given in A.9. As an example, the main code is ran by using a set of initial parameters. The initial parameters are:

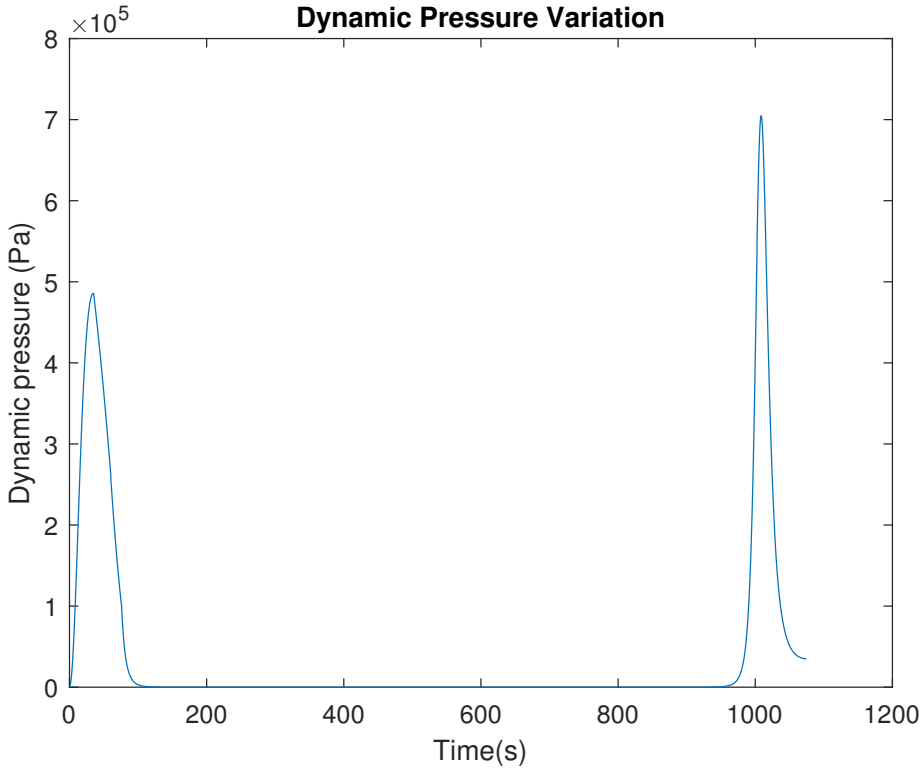
V_0 : 10 $\mathrm{m\,s^{-1}}$
h_0 : 0.001 m
Initial azimuth angle of the rocket: 90°
Initial elevation angle of the rocket: 60°
Launch latitude of the rocket: 41.0605°
Launch longitude of the rocket: 29.0118°
Latitude of the ground station: 41.1015°
Longitude of the ground station: 29.0215°
Altitude of the ground station: 0.001 m
c_d : 0.3

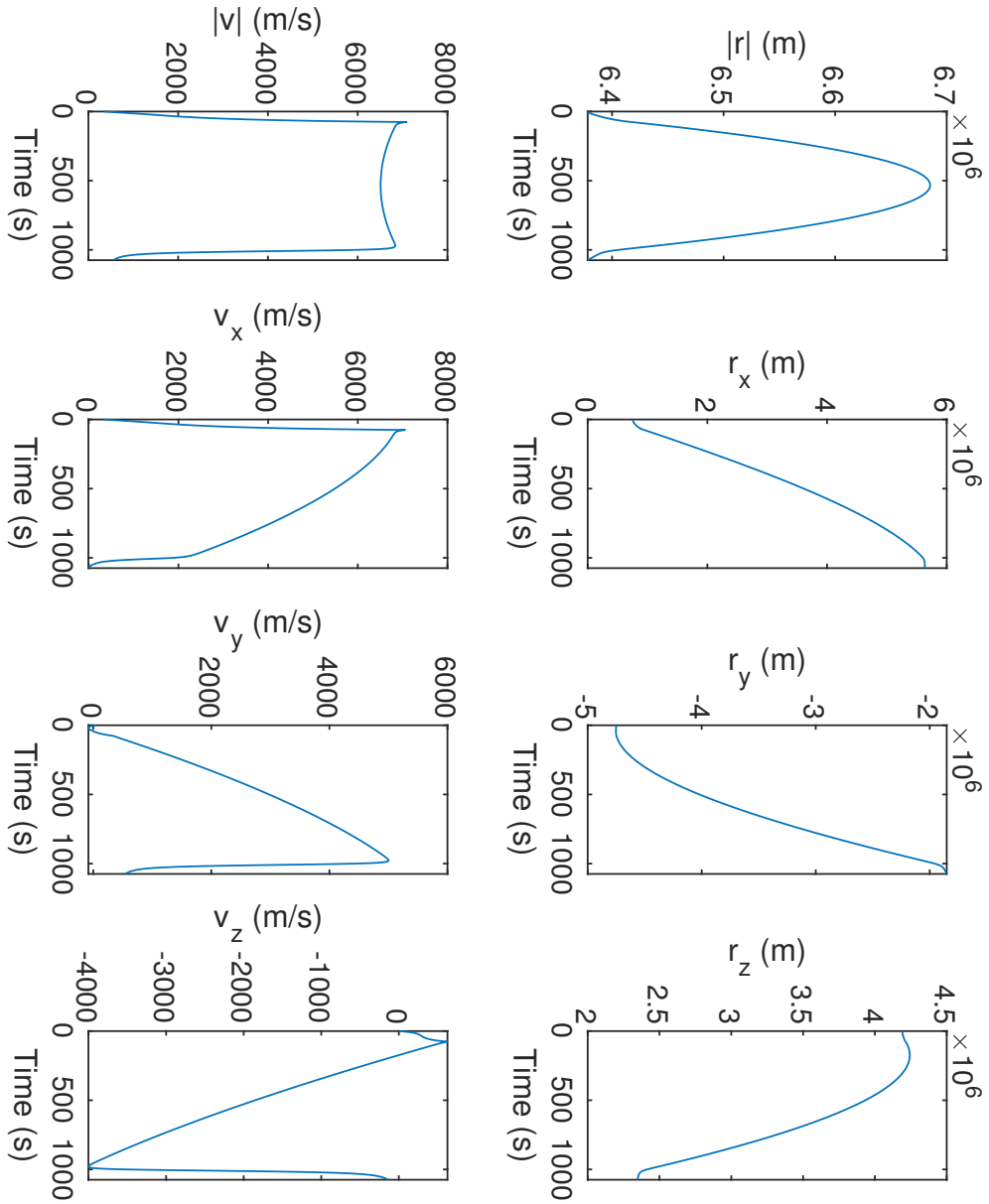
A_{ref} : 0.78 m
v_{exh} : 5000 $\mathrm{m\,s^{-1}}$
m_0 : 4500 kg
m_{fuel} : 3800 kg
$\dot{m}$: 50 $\mathrm{kg\,s^{-1}}$
Year: 2021
Month: 1
Day: 11
Hour: 9
Minute: 15
Second: 30

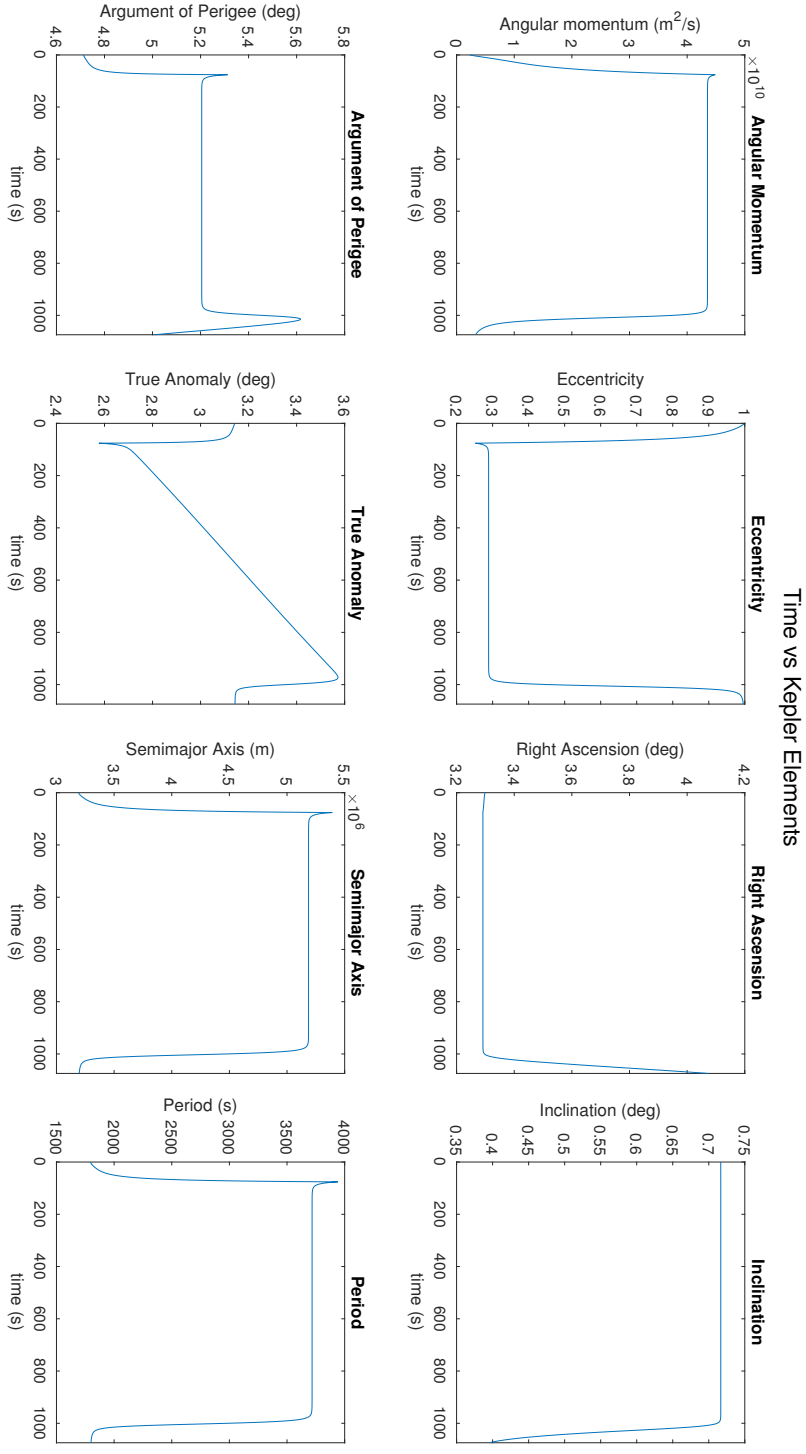
Finally, outputs of the code are given below. Red regions of orbit and ground track shows the powered flight:

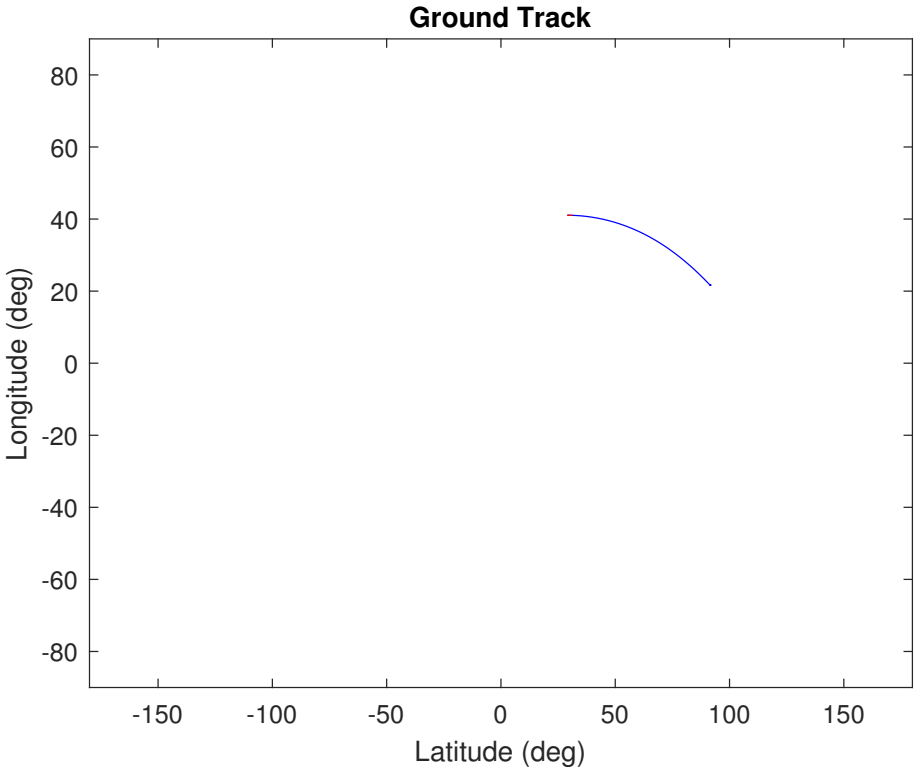
```
Total flight time is 1074.55 seconds

Maximum altitude of the orbit: 307.0065 km
Corresponding time: 532.50 seconds

The maximum velocity during flight: 7091.8105 m/s
Corresponding time: 76.00 seconds

The maximum dynamic pressure during flight: 704731.0249 Pa
Corresponding time: 1008.59 seconds

Landing coordinates: 21.595291 N , 91.833789 E

Range of the flight: 6212.1348 km
>>
```

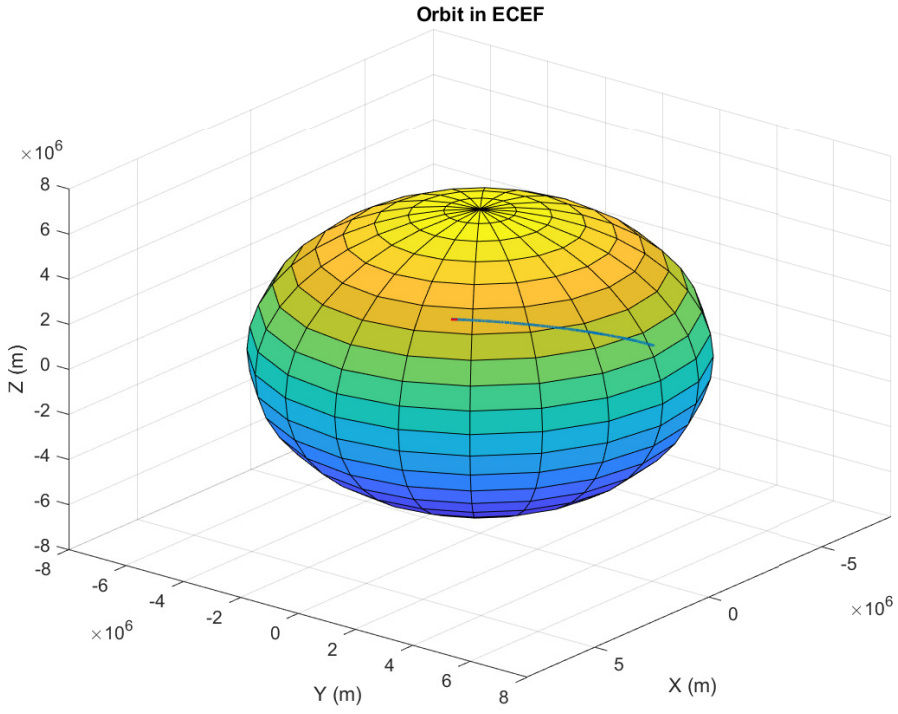

Altitude Variation
×10^5
3.5
3
2.5
2
1.5
1
0.5
0
Altitude (m)
0
200
400
600
800
1000
Time(s)

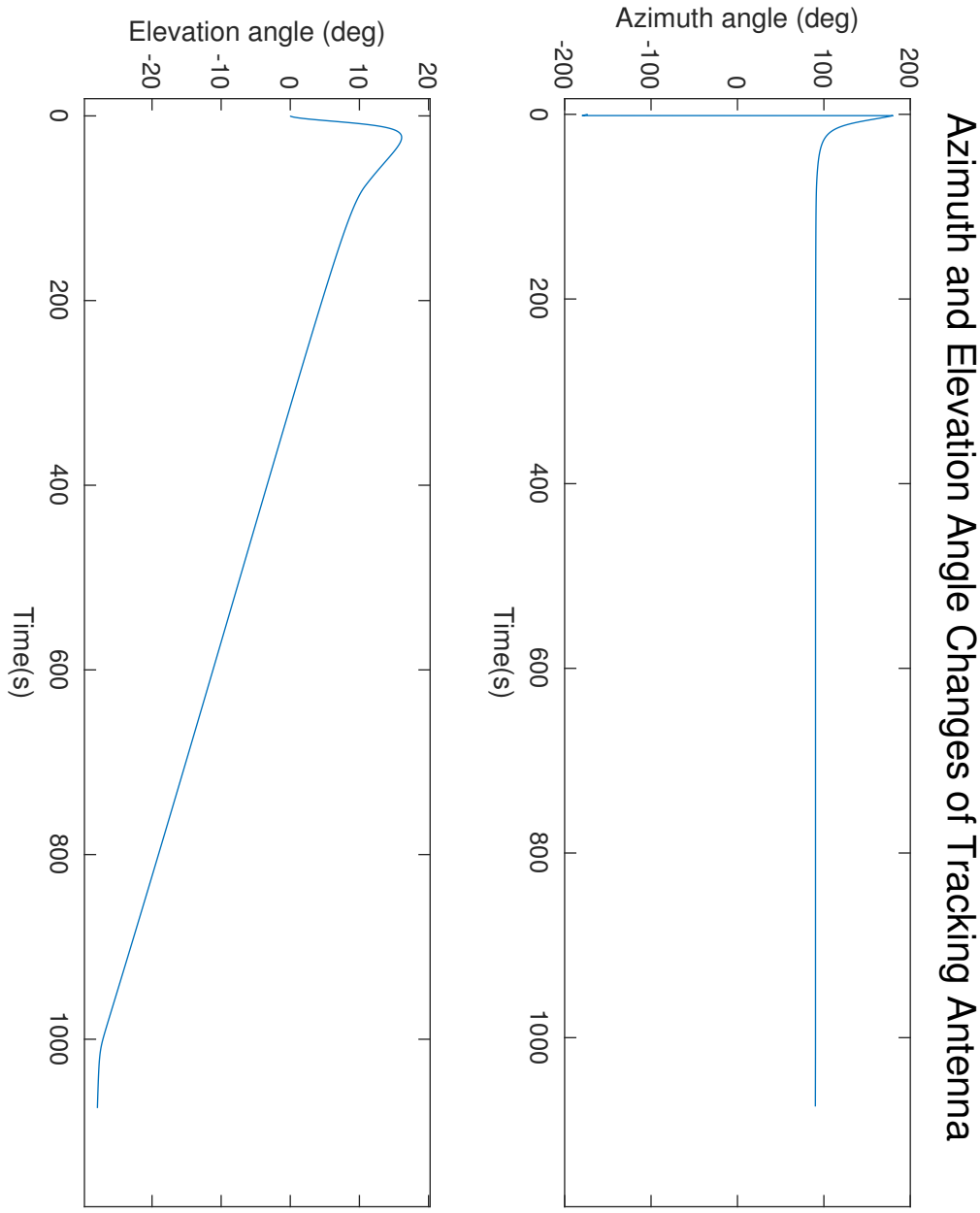

Dynamic Pressure Variation
×10^5
8
7
6
5
4
3
2
1
0
Dynamic pressure (Pa)
0
200
400
600
800
1000
1200
Time(s)


Time vs Position and Velocity in ECI
|r| (m)
r_x (m)
r_y (m)
r_z (m)
|v| (m/s)
v_x (m/s)
v_y (m/s)
v_z (m/s)
Time (s)
×10^6


Time vs Kepler Elements
Angular momentum (m^2/s)
×10^10
Angular Momentum
Eccentricity
Eccentricity
Right Ascension (deg)
Right Ascension
Inclination (deg)
Inclination
Argument of Perigee (deg)
Argument of Perigee
True Anomaly (deg)
True Anomaly
Semimajor Axis (m)
×10^6
Semimajor Axis
Period (s)
Period
time (s)


Ground Track
Longitude (deg)
Latitude (deg)
80
60
40
20
0
-20
-40
-60
-80
-150
-100
-50
0
50
100
150


Orbit in ECEF
Z (m)
Y (m)
X (m)
×10^6


Azimuth and Elevation Angle Changes of Tracking Antenna
Azimuth angle (deg)
200
100
0
-100
-200
0
200
400
600
800
1000
Time(s)
Elevation angle (deg)
20
10
0
-10
-20
0
200
400
600
800
1000
Time(s)

8 Conclusion

As shown by Kepler elements output, at the beginning of the flight, Kepler elements change dramatically since there are drag and thrust force that can be assumed as disturbances that effect the orbital flight. However, after a time, when the thrust force vanishes and drag force nearly becomes zero (since density of the atmosphere converges zero at high altitudes), the Kepler elements behave linear (behave constant except true anomaly), which is consistent with the expectation since in space, rules of orbital mechanics are valid and there are no disturbances affecting angular momentum etc. After entering the atmosphere, the behavior becomes nonlinear since the effect of drag force occurs again. Furthermore, for this example flight, eccentricity is about 0.3 in constant region, which shows that the orbit of flight is elliptical and close to circular orbit than a parabolic orbit.

If the initial parameters are changed, the orbit of the rocket will obviously different than the orbit of this example. Since the Earth rotates during the flight, initial azimuth and elevation angles of the rocket will change the landing point of the rocket if one changes these initial angles.

Appendices

A MATLAB Codes

A.1 Function: LLAtoECEF

```
function [r_ecef] = LLAtoECEF (degLat, degLon, h)

Re = 6378136.6;

r = h + Re;

lat = deg2rad(degLat);
lon = deg2rad(degLon);

[ecef_x, ecef_y, ecef_z] = sph2cart(lon,lat,r);

r_ecef = [ecef_x; ecef_y; ecef_z];

end
```

A.2 Function: topo2geo

```
function vGeoEqu = topo2geo(vLength,latDeg,lonDeg,azm,elv)
%Azimuth: Angle from the north (positive clockwise)
%Elevation: Angle from line of sight

lat = deg2rad(latDeg);
lon = deg2rad(lonDeg);

%Velocity vector of a body in topocentric horizon system:
vTopoHorz = [vLength*cosd(elv)*sind(azm);
vLength*cosd(elv)*cosd(azm);
vLength*sind(elv)   ];

%Transf. matrix from topocentric horizon into Geocentric Equatorial:
Q = [ -sin(lon)  -sin(lat)*cos(lon)    cos(lat)*cos(lon);
cos(lon)  -sin(lat)*sin(lon)    cos(lat)*sin(lon);
0           cos(lat)             sin(lat)];

%Velocity vector transformed into Geocentric Equatorial (ECI)
vGeoEqu = Q*vTopoHorz;
end
```

A.3 Function: ECEF2topo

```
function [azm, elv] = ECEF2topo(gndstLatDeg, gndstLonDeg, ECEF_pos)

ECEF_pos = transpose(ECEF_pos);  %Conver it into 3x1
lat = deg2rad(gndstLatDeg);
lon = deg2rad(gndstLonDeg);

%Transformation matrix from geocentric to topocentric horizon
Q = [    -sin(lon)               cos(lon)                0 ;
-sin(lat)*cos(lon)    -sin(lat)*sin(lon)        cos(lat);
cos(lat)*cos(lon)     cos(lat)*sin(lon)        sin(lat)];

rTopo = Q*ECEF_pos;

sin_elv = rTopo(3,1)/norm(rTopo); %Sine of elevation angle
elv = asind(sin_elv);  %Calculate elevation angle in deg

sin_azm = rTopo(1,1) / (cosd(elv)*norm(rTopo)); %Sine of azimuth angle
cos_azm = rTopo(2,1) / (cosd(elv)*norm(rTopo)); %Cosine of azimuth angle

azm = atan2d(sin_azm,cos_azm); %Azimuth angle in deg
end
```

A.4 Function: ECEFtoECI

```
% Programed by Darin Koblick  07-17-2010                                   %
% Modified on 03/01/2012 to add acceleration vector support
% Modified on 06/01/2021 by Emre SAYIN for Spacecraft Dynamics Lecture Project

function [r_ECI, v_ECI] = ECEFtoECI(JD, r_ECEF, v_ECEF)
%Enforce JD to be [N x 1]
JD = JD(:);

THETA = JD2GMST(JD);
%Average inertial rotation rate of the earth radians per second
omega_e = 7.292115e-5;

%Assemble the transformation matricies to go from ECEF to ECI
r_ECI = squeeze(MultiDimMatrixMultiply(MultiDimMatrixTranspose(T3D(THETA)),r_ECEF));
v_ECI = squeeze(MultiDimMatrixMultiply(MultiDimMatrixTranspose(T3D(THETA)),v_ECEF) ...
    + ...
MultiDimMatrixMultiply(MultiDimMatrixTranspose(Tdot3D(THETA,omega_e)),r_ECEF));

function C = MultiDimMatrixMultiply(A,B)
C = sum(bsxfun(@times,A,repmat(permute(B',[3 2 1]),[size(A,2) 1 1])),2);
function A = MultiDimMatrixTranspose(A)
A = permute(A,[2 1 3]);
function T = T3D(THETA)
T = zeros([3 3 length(THETA)]);
T(1,1,:) = cosd(THETA);
```

```
T(1,2,:) = sind(THETA);
T(2,1,:) = -T(1,2,:);
T(2,2,:) = T(1,1,:);
T(3,3,:) = ones(size(THETA));
function Tdot = Tdot3D(THETA,omega_e)
Tdot = zeros([3 3 length(THETA)]);
Tdot(1,1,:) = -omega_e.*sind(THETA);
Tdot(1,2,:) = omega_e.*cosd(THETA);
Tdot(2,1,:) = -Tdot(1,2,:);
Tdot(2,2,:) = Tdot(1,1,:);
```

A.5 MATLAB Function Block: ECItoECEF

```
% Programed by Darin Koblick  07-05-2010                                   %
% Modified on 03/01/2012 to add acceleration vector support                %
% Modified on 06/01/2021 by Emre SAYIN for Spacecraft Dynamics Lecture Project

%--------------------------------------------------------------------------%
function [r_ECEF, v_ECEF ] = ECItoECEF(JD,r_ECI,v_ECI)
%Enforce JD to be [N x 1]
JD = JD(:);

%Calculate the Greenwich Apparent Sideral Time (THETA)
THETA = JD2GAST(JD);

%Average inertial rotation rate of the earth radians per second
omega_e = 7.292115e-5;

%Assemble the transformation matricies to go from ECI to ECEF
r_ECEF = squeeze(MultiDimMatrixMultiply(T3D(THETA),r_ECI));
v_ECEF = squeeze(MultiDimMatrixMultiply(T3D(THETA),v_ECI) + ...
MultiDimMatrixMultiply(Tdot3D(THETA,omega_e),r_ECI));

%----------------------------- End Code-----------------------------------%

function C = MultiDimMatrixMultiply(A,B)
C = sum(bsxfun(@times,A,repmat(permute(B',[3 2 1]),[size(A,2) 1 1])),2);

function T = T3D(THETA)
T = zeros([3 3 length(THETA)]);
T(1,1,:) = cosd(THETA);
T(1,2,:) = sind(THETA);
T(2,1,:) = -T(1,2,:);
T(2,2,:) = T(1,1,:);
T(3,3,:) = ones(size(THETA));

function Tdot = Tdot3D(THETA,omega_e)
Tdot = zeros([3 3 length(THETA)]);
Tdot(1,1,:) = -omega_e.*sind(THETA);
Tdot(1,2,:) = omega_e.*cosd(THETA);
Tdot(2,1,:) = -Tdot(1,2,:);
Tdot(2,2,:) = Tdot(1,1,:);
```

A.6 MATLAB Function Block: ECIconvLLA

```
function [degLat,degLon,h] = ECIconvLLA(JD,r_ECI)

Re = 6378136.6;

%Enforce JD to be [N x 1]
JD = JD(:);

%Calculate the Greenwich Apparent Sideral Time (THETA)
THETA = JD2GAST(JD);

%Assemble the transformation matrix to go from ECI to ECEF
r_ECEF = squeeze(MultiDimMatrixMultiply(T3D(THETA),r_ECI));

ecef_x = r_ECEF(1);
ecef_y = r_ECEF(2);
ecef_z = r_ECEF(3);

%ECI to LLA conversion
[lon, lat, r] = cart2sph(ecef_x,ecef_y,ecef_z);

degLon = rad2deg(lon);
degLat = rad2deg(lat);
h = r-Re;

function C = MultiDimMatrixMultiply(A,B)
    C = sum(bsxfun(@times,A,repmat(permute(B',[3 2 1]),[size(A,2) 1 1])),2);

function T = T3D(THETA)
    T = zeros([3 3 length(THETA)]);
    T(1,1,:) = cosd(THETA);
    T(1,2,:) = sind(THETA);
    T(2,1,:) = -T(1,2,:);
    T(2,2,:) = T(1,1,:);
    T(3,3,:) = ones(size(THETA));
```

A.7 MATLAB Function Block: getVectorLength

```
    function L = getVectorLength(u)

    L = sqrt( u(1)^2 + u(2)^2 + u(3)^2 );

    end
```

A.8 Function: coe_from_sv

```
function coe = coe_from_sv(R,V,mu)
%{
% This function computes the classical orbital elements (coe)
% from the state vector (R,V)
%
mu - gravitational parameter (km^3/s^2)
R - position vector in the geocentric equatorial frame (km)
V - velocity vector in the geocentric equatorial frame (km)
r, v - the magnitudes of R and V
vr - radial velocity component (km/s)
H - the angular momentum vector (km^2/s)
h - the magnitude of H (km^2/s)
incl - inclination of the orbit (rad)
N - the node line vector (km^2/s)
n - the magnitude of N
cp - cross product of N and R
RA - right ascension of the ascending node (rad)
E - eccentricity vector
e - eccentricity (magnitude of E)
eps - a small number below which the eccentricity is considered
to be zero
w - argument of perigee (rad)
TA - true anomaly (rad)
a - semimajor axis (km)
pi - 3.1415926...
coe - vector of orbital elements [h e RA incl w TA a]
User M-functions required: None
%}

eps = 1.e-10;
r = norm(R);
v = norm(V);

vr = dot(R,V)/r;
H = cross(R,V);
h = norm(H);

incl = acos(H(3)/h);

N = cross([0 0 1],H);
n = norm(N);

if n ≠ 0
RA = acos(N(1)/n);
if N(2) < 0
RA = 2*pi - RA;
end
else
RA = 0;
end

E = 1/mu*((v^2 - mu/r)*R - r*vr*V);
```

```
e = norm(E);

if n ≠ 0
if e > eps
w = acos(dot(N,E)/n/e);
if E(3) < 0
w = 2*pi - w;
end
else
w = 0;
end
else
w = 0;
end

if e > eps
TA = acos(dot(E,R)/e/r);
if vr < 0
TA = 2*pi - TA;
end
else

cp = cross(N,R);
if cp(3) ≥ 0
TA = acos(dot(N,R)/n/r);
else
TA = 2*pi - acos(dot(N,R)/n/r);
end
end

a = h^2/mu/(1 - e^2);
coe = [h e RA incl w TA a];
end %coe_from_sv
```

A.9 The Main Code

```
clear;
clc;

Re      = 6378136.6;   %Earth radius in m
mu      = 3.986004e14; %Standard grav. parameter in  m^3/s^2
omega_E = 7.292115e-5; %Earth angular velocity in rad/s

%----------------------Initial parameters-----------------%

cd      = 0.3;         %Drag coefficient
Aref    = 0.78;        %Reference area in m^2
v_exh   = 5000;        %Exhaust velocity in m/s

m0      = 4500;         %Initial mass in kg
mFuel   = 3800;         %Fuel mass in kg
m_dot   = 50;           %Mass flow rate in kg/s

year    = 2021;
month   = 1;
day     = 11;
hour    = 9;
minute  = 15;
second  = 30;

initLat  = 41.0605;  %Initial latitude in deg
initLong = 29.0118;  %Initial longitude in deg

GndstLat  = 41.1015; %Latitude of ground station in deg
GndstLong = 29.0215; %Longitude of ground station in deg
hGndst    = 0.001;   %Ground station altitude in m

h0 = 0.001; %Initial altitude of the rocket in m

V0  = 10;   %Launch velocity in m/s
azm = 90;   %Launch azimuth angle in deg
elv = 60;   %Launch elevation angle in deg
%---------------------------------------------------------%

v_ECEF_init = topo2geo(V0,initLat,initLong,azm,elv); %Convert Topocentric Horizon ...
    into ECEF
r_ECEF_init = LLAtoECEF(initLat, initLong, h0); %Convert LLA into ECEF

jd = getJDay(year,month,day,hour,minute,second); %Calculate Julian Day
[r_ECI_init, v_ECI_init] = ECEFtoECI(jd,r_ECEF_init,v_ECEF_init); %Convert ECEF ...
    into ECI

omegaE_vect = [0; 0; omega_E];

%Update initial position and velocity vectors as ECI
Vx0 = v_ECI_init(1);
Vy0 = v_ECI_init(2);
Vz0 = v_ECI_init(3);
```

```
rx0 = r_ECI_init(1);
ry0 = r_ECI_init(2);
rz0 = r_ECI_init(3);

r_init_vect = [rx0 ry0 rz0];
r_init = norm(r_init_vect); %Length of initial position vector in ECI

sim('project.slx');

t = ans.tout;
lat = ans.lat;   %Latitude change by time
lon = ans.lon;   %Longitude change by time
h   = ans.h;     %Height change by time

rEcef = ans.rEcef;
vEcef = ans.vEcef;
vEcefLength = ans.vEcefLength;

rECI = ans.rECI;
vECI = ans.vECI;

pDyn = ans.pDyn;

rECI_length = zeros(1,length(t));
vECI_length = zeros(1,length(t));

for j = 1:length(t)
rECI_length(j) = norm(rECI(j,:));
vECI_length(j) = norm(vECI(j,:));
end

thrust_time = mFuel/m_dot;
idx_t = find(t == thrust_time); %Index of thrust time

%Total flight time:
t_flight = t(end); %Flight time in s
fprintf('Total flight time is %.2f seconds\n\n',t_flight);

%Time vs r and v plot
figure;
subplot(2,4,1)
plot(t, rECI_length)
xlabel('Time (s)')
ylabel('|r| (m)')

subplot(2,4,2)
plot(t, rECI(:,1))
xlabel('Time (s)')
ylabel('r_x (m)')

subplot(2,4,3)
plot(t, rECI(:,2))
xlabel('Time (s)')
ylabel('r_y (m)')
```

```
subplot(2,4,4)
plot(t, rECI(:,3))
xlabel('Time (s)')
ylabel('r_z (m)')

subplot(2,4,5)
plot(t, vECI_length)
xlabel('Time (s)')
ylabel('|v| (m/s)')

subplot(2,4,6)
plot(t, vECI(:,1))
xlabel('Time (s)')
ylabel('v_x (m/s)')

subplot(2,4,7)
plot(t, vECI(:,2))
xlabel('Time (s)')
ylabel('v_y (m/s)')

subplot(2,4,8)
plot(t, vECI(:,3))
xlabel('Time (s)')
ylabel('v_z (m/s)')

sgtitle('Time vs Position and Velocity in ECI')


%Time vs Kepler elements

orb_el  = zeros(length(t),7);
ang_mom = zeros(length(t),1);
ecc     = zeros(length(t),1);
ra      = zeros(length(t),1);
inc_d   = zeros(length(t),1);
aop     = zeros(length(t),1);
t_anom  = zeros(length(t),1);
sem_ax  = zeros(length(t),1);
T       = zeros(length(t),1);

for i = 1:length(t)
orb_el(i,:) = coe_from_sv(rECI(i,:),vECI(i,:),mu);

ang_mom(i) = orb_el(i,1); %Angular momentum in m^2/s
ecc(i)     = orb_el(i,2); %Eccentricity
ra(i)      = orb_el(i,3); %Right ascension in deg
inc_d(i)   = orb_el(i,4); %Inclination in deg
aop(i)     = orb_el(i,5); %Argument of perigee in deg
t_anom(i)  = orb_el(i,6); %True anomaly in deg
sem_ax(i)  = orb_el(i,7); %Semimajor axis in m

if ecc(i)<1
T(i) = 2*pi/sqrt(mu)*orb_el(i,7)^1.5; %Period in seconds
end
end
```

```
figure;
subplot(2,4,1)
plot(t,ang_mom)
title('Angular Momentum')
xlabel('time (s)')
ylabel('Angular momentum (m^2/s)')

subplot(2,4,2)
plot(t,ecc)
title('Eccentricity')
xlabel('time (s)')
ylabel('Eccentricity')

subplot(2,4,3)
plot(t,ra)
title('Right Ascension')
xlabel('time (s)')
ylabel('Right Ascension (deg)')

subplot(2,4,4)
plot(t,inc_d)
title('Inclination')
xlabel('time (s)')
ylabel('Inclination (deg)')

subplot(2,4,5)
plot(t,aop)
title('Argument of Perigee')
xlabel('time (s)')
ylabel('Argument of Perigee (deg)')

subplot(2,4,6)
plot(t,t_anom)
title('True Anomaly')
xlabel('time (s)')
ylabel('True Anomaly (deg)')

subplot(2,4,7)
plot(t,sem_ax)
title('Semimajor Axis')
xlabel('time (s)')
ylabel('Semimajor Axis (m)')

subplot(2,4,8)
plot(t,T)
title('Period')
xlabel('time (s)')
ylabel('Period (s)')

sgtitle('Time vs Kepler Elements')


%Ground track plot
figure;
plot(lon,lat,'b');
hold on
```

```
plot(lon(1:idx_t),lat(1:idx_t),'r')

title('Ground Track')
xlim([-180 180])
ylim([-90 90])

xlabel('Latitude (deg)')
ylabel('Longitude (deg)')

recycle on
delete('latLonData.xlsx');
latLonData = [lon lat];
xlswrite('latLonData.xlsx',latLonData);


%Orbit Plot
[X,Y,Z] = sphere;
X2 = X * Re;
Y2 = Y * Re;
Z2 = Z * Re;

figure;
s = surf(X2,Y2,Z2);
hold on
plot3(rEcef(:,1),rEcef(:,2),rEcef(:,3),'Linewidth',1.5)
plot3(rEcef(1:idx_t,1),rEcef(1:idx_t,2),rEcef(1:idx_t,3),'r','Linewidth',1.5)
xlabel('X (m)')
ylabel('Y (m)')
zlabel('Z (m)')
title('Orbit in ECEF')
xlim([-8e6 8e6])
ylim([-8e6 8e6])
zlim([-8e6 8e6])

%Highest point of the orbit
h_max = max(h); %Maximum altitude in meters
idx_h_max = find(h == h_max);
t_h_max = t(idx_h_max);
h_max_km = h_max/1e3;
fprintf('Maximum altitude of the orbit: %.4f km\n',h_max_km);
fprintf('Corresponding time: %.2f seconds\n\n',t_h_max);

%Fastest point of the orbit
vECI_max = max(vECI_length);  %Find maximum vECI value in m/s
idx_vECI_max = find(vECI_length == vECI_max); %Find the index of it
t_vECI_max   = t(idx_vECI_max); %Find corresponding time in s
fprintf('The maximum velocity during flight: %.4f m/s\n',vECI_max);
fprintf('Corresponding time: %.2f seconds\n\n',t_vECI_max);

pDynMax = max(pDyn);  %Find the maximum pDyn value in Pa
idx_pDynMax = find(pDyn == pDynMax);  %Find index of max. pDyn
t_pDynMax = t(idx_pDynMax);     %Find Corresponding time in s
fprintf('The maximum dynamic pressure during flight: %.4f Pa\n',pDynMax);
fprintf('Corresponding time: %.2f seconds\n\n',t_pDynMax);

%Landing coordinates
```

```
lat_land = lat(end);
lon_land = lon(end);

if (lat_land > 0)
lat_str = 'N';

elseif (lat_land < 0)
lat_str = 'S';
end

if (lon_land > 0)
lon_str = 'E';

elseif (lon_land < 0)
lon_str = 'W';
end

lat_land = abs(lat_land);
lon_land = abs(lon_land);

fprintf('Landing coordinates: %.6f %s , %.6f ...
    %s\n\n',lat_land,lat_str,lon_land,lon_str);


%Range of the flight
r_Ecef_init  = rEcef(1,:);
r_Ecef_final = rEcef(end,:);

cos_beta = dot(r_Ecef_init, r_Ecef_final) / (norm(r_Ecef_init)*norm(r_Ecef_final));
beta = acos(cos_beta); %Angle between initial and final position vectors in rad
range = Re*beta;
range_km = range/1e3;

fprintf('Range of the flight: %.4f km\n',range_km);


%Azimuth and elevation angles of tracking antenna
rGndst_ECEF = LLAtoECEF(GndstLat,GndstLong, hGndst);
rGndst_ECEF = [rGndst_ECEF(1,:) rGndst_ECEF(2,:) rGndst_ECEF(3,:)];

rhoVect_ECEF = zeros(length(t),3); %Create the vector btw ground station and ...
    rocket in ECEF
gndstazm = zeros(length(t),1); %Create the azimuth angle vector btw ground station ...
    and rocket
gndstelv = zeros(length(t),1); %Create the elevation angle btw ground station and ...
    rocket

for k = 1:length(t)
%Calculate the vector between ground station and rocket in ECEF
rhoVect_ECEF(k,:) = rEcef(k,:) - rGndst_ECEF;
%Convert it into Topocentric Horizontal Coord. Sys.
[gndstazm(k),gndstelv(k)] = ECEF2topo(GndstLat,GndstLong,rhoVect_ECEF(k,:));
end

figure;
subplot(2,1,1)
```

```
plot(t,gndstazm)
xlabel('Time(s)')
ylabel('Azimuth angle (deg)')

subplot(2,1,2)
plot(t,gndstelv)
xlabel('Time(s)')
ylabel('Elevation angle (deg)')

sgtitle('Azimuth and Elevation Angle Changes of Tracking Antenna')


figure;
plot(t,pDyn)
xlabel('Time(s)')
ylabel('Dynamic pressure (Pa)')
title('Dynamic Pressure Variation')


figure;
plot(t,h)
xlabel('Time(s)')
ylabel('Altitude (m)')
title('Altitude Variation')

function j0 = J0(year, month, day)

% This function computes the Julian day number at 0 UT for any
% year between 1900 and 2100.
%
% j0 - Julian day at 0 hr UT (Universal Time)
% year - range: 1901 - 2099
% month - range: 1 - 12
% day - range: 1 - 31

j0 = 367*year - fix(7*(year + fix((month + 9)/12))/4) ...
+ fix(275*month/9) + day + 1721013.5;

end

function jd = getJDay(year,month,day,hour,minute,second)

ut = hour + minute/60 + second/3600;

j0 = J0(year,month,day);

jd = j0 + ut/24;

end
```

References

[1] Earth Atmosphere Model - Metric Units. (n.d.). Retrieved January 08, 2021, from https://www.grc.nasa.gov/www/k-12/airplane/atmosmet.html

[2] Sph2cart. (n.d.). Retrieved January 08, 2021, from https://www.mathworks.com/help/matlab/ref/sph2cart.html

[3] Darin Koblick (2021). Convert ECEF to ECI Coordinates (https://www.mathworks.com/matlabcentral/fileexchange/28234-convert-ecef-to-eci-coordinates), MATLAB Central File Exchange. Retrieved January 8, 2021.

[4] Curtis, H. D. (2013). *Orbital mechanics for engineering students*. Butterworth-Heinemann.

[5] Darin Koblick (2021). Convert ECI to ECEF Coordinates (https://www.mathworks.com/matlabcentral/fileexchange/28233-convert-eci-to-ecef-coordinates), MATLAB Central File Exchange. Retrieved January 8, 2021.

[6] The Astronomical Almanac Online", 2018 Selected Astronomical Constants: http://asa.hmnao.com/static/files/2018/Astronomical_Constants_2018.pdf