Table of Contents


```
clc; clear all; close all;
```

PART ONE: Time Optimal Titan II launch from ground, air, and balloon.

```
% This script is derived from Appendix B from "Optimal Controls With
% Aerospace Applications".
% Drag on the vehicle and varying mass are calculated. Nozzle
 efficiency is
% considered for the ground launch only.

for iter = 1:3

    global g Vc h K1 scale F m0 mdot variable_thrust xbar0 ybar0
 Vxbar0 Vybar0 ybarf Vxbarf Vybarf
```

Boundary Conditions for Three Launch Types

```
    %Ground Launch Boundary Conditions
    if iter == 1
        variable_thrust = 0;
        h = 150e3; % m, final altitude
        Vc = sqrt(3.9860044e5/(6378.14+h/1000))*1000; % m/s, final
 circular velocity
        g = 9.80665; % m/s^2, gravity
        F = 2.1e6; % N, constant thrust
        h_scale = 8440; % m, atmospheric scale-height
        scale = h/h_scale; % constant for EOM simplification
        rho_ref = 1.225; % reference density
        A = 7.069; % m^2, cross-sectional area

        % Initial conditions
        xbar0 = 0; % initial x-position
```

```
        ybar0 = 0; % initial y-position
        Vxbar0 = 0; % initial downrange velocity
        Vybar0 = 0; % initial vertical velocity

        % Final conditions
        ybarf = h/h; % final altitude
        Vxbarf = Vc/Vc; % final downrange velocity
        Vybarf = 0; % final vertical velocity
        Hf = -1; %final value of Hamiltonian
    end
    if iter == 2
        variable_thrust = 0;
        h_init = 15000; % m ,initial altitude
        V_init = 220; % m/s, initial velocity
        h = 150000; % m, final altitude
        Vc = sqrt(3.9860044e5/(6378.14+h/1000))*1000; % m/s, final
circular speed
        g = 9.80665; % m/s^2, gravity
        F = 2.1e6; % N, constant thrust
        h_scale = 8440; % m, atmospheric scale-height
        scale = h/h_scale; % constant to simplify EOM
        rho_ref = 1.225; % reference density
        A = 7.069; % m^22, cross-sectional area

        % Initial conditions
        xbar0 = 0; % initial x-position
        ybar0 = h_init/h; % initial y-position
        Vxbar0 = V_init/Vc * sqrt(2)/2; % initial downrange velocity
        Vybar0 = V_init/Vc * sqrt(2)/2; % initial vertical velocity

        % Final conditions
        ybarf = h/h; % final altitude
        Vxbarf = Vc/Vc; % final downrange velocity
        Vybarf = 0; % final vertical velocity
        Hf = -1; %final value of Hamiltonian
    end
    if iter == 3
        variable_thrust = 0;
        h_init = 30000; % m ,initial altitude
        V_init = 0; % m/s, initial velocity
        h = 150000; % m, final altitude
        Vc = sqrt(3.9860044e5/(6378.14+h/1000))*1000; % m/s, final
circular speed
        g = 9.80665; % m/s^2, gravity
        F = 2.1e6; % N, constant thrust
        h_scale = 8440; % m, atmospheric scale-height
        scale = h/h_scale; % constant to simplify EOM
        rho_ref = 1.225; % reference density
        A = 7.069; % m^22, cross-sectional area

        % Initial conditions
        xbar0 = 0; % initial x-position
        ybar0 = h_init/h; % initial y-position
        Vxbar0 = V_init/Vc * sqrt(2)/2; % initial downrange velocity
```

```
        Vybar0 = V_init/Vc * sqrt(2)/2; % initial vertical velocity

        % Final conditions
        ybarf = h/h; % final altitude
        Vxbarf = Vc/Vc; % final downrange velocity
        Vybarf = 0; % final vertical velocity
        Hf = -1; %final value of Hamiltonian
    end
```

Solution

CONSTANT MASS / NO DRAG Parameters

```
m0 = 60880; % kg, average mass of rocket
CD = 0; % no drag case has Cd = 0
mdot = 0;
K1 = rho_ref*CD*A*Vc/(2*m0); %constant to simplify EOM (drag
constants * Vc / m)

% Initial Guesses
t0 = 0;
yinit = [xbar0 ybar0 Vxbar0 Vybar0 0 -1 0]; %lambda20 lambda30
lambda40
tf_guess = 500; % sec, initial guess for final time
Nt = 80;
tau = linspace(0,1,Nt)'; % nondimensional time vector

solinit = bvpinit(tau,yinit,tf_guess);

% Solution - CONSTANT MASS / NO DRAG

sol = bvp4c(@ascent_odes_tf, @ascent_bcs_tf, solinit);
% Extract the final time from the solution:
tf = sol.parameters(1);
% Evaluate the solution
Z = deval(sol,tau);
% Convert back to dimensional time for plotting
time = t0 + tau.*(tf-t0);

x_sol = Z(1,:)*h/1000;
y_sol = Z(2,:)*h/1000;
vx_sol = Z(3,:)*Vc/1000;
vy_sol = Z(4,:)*Vc/1000;
lambda2_bar_sol = Z(5,:);
lambda3_bar_sol = Z(6,:);
lambda4_bar_sol = Z(7,:);

% Solution - VARIABLE MASS / NO DRAG
m0 = 117020;
mdot = 807.5;
delta_tf = 114;
CD = 0;
K1 = rho_ref*CD*A*Vc/(2*m0);
```

```matlab
solinit_mass = solinit;
solinit_mass.y = Z;
tf = sol.parameters(1);
solinit_mass.parameters(1) = tf-delta_tf;

sol_mass = bvp4c(@ascent_odes_tf, @ascent_bcs_tf, solinit_mass);
% Extract the final time from the solution:
tf_mass = sol_mass.parameters(1)
% Evaluate the solution
Z_mass = deval(sol_mass,tau);
% Convert back to dimensional time for plotting
time_mass = t0 + tau.*(tf-t0);

x_sol_mass = Z_mass(1,:)*h/1000;
y_sol_mass = Z_mass(2,:)*h/1000;
vx_sol_mass = Z_mass(3,:)*Vc/1000;
vy_sol_mass = Z_mass(4,:)*Vc/1000;
lambda2_bar_sol_mass = Z_mass(5,:);
lambda3_bar_sol_mass = Z_mass(6,:);
lambda4_bar_sol_mass = Z_mass(7,:);


% Solution - VARIABLE MASS / WITH DRAG
m0 = 117020;
mdot = 807.5;
delta_tf = 114;
CD = 0.5;
K1 = rho_ref*CD*A*Vc/(2*m0);

solinit_mass_drag = solinit_mass;
solinit_mass_drag.y = Z_mass;
tf_mass = sol_mass.parameters(1);
solinit_mass_drag.parameters(1) = tf_mass;

sol_mass_drag = bvp4c(@ascent_odes_tf, @ascent_bcs_tf,
solinit_mass_drag);
% Extract the final time from the solution:
tf_mass_drag = sol_mass_drag.parameters(1);
% Evaluate the solution
Z_mass_drag = deval(sol_mass_drag,tau);
% Convert back to dimensional time for plotting
time_mass_drag = t0 + tau.*(tf-t0);

x_sol_mass_drag = Z_mass_drag(1,:)*h/1000;
y_sol_mass_drag = Z_mass_drag(2,:)*h/1000;
vx_sol_mass_drag = Z_mass_drag(3,:)*Vc/1000;
vy_sol_mass_drag = Z_mass_drag(4,:)*Vc/1000;
lambda2_bar_sol_mass_drag = Z_mass_drag(5,:);
lambda3_bar_sol_mass_drag = Z_mass_drag(6,:);
lambda4_bar_sol_mass_drag = Z_mass_drag(7,:);
```

tf_mass =

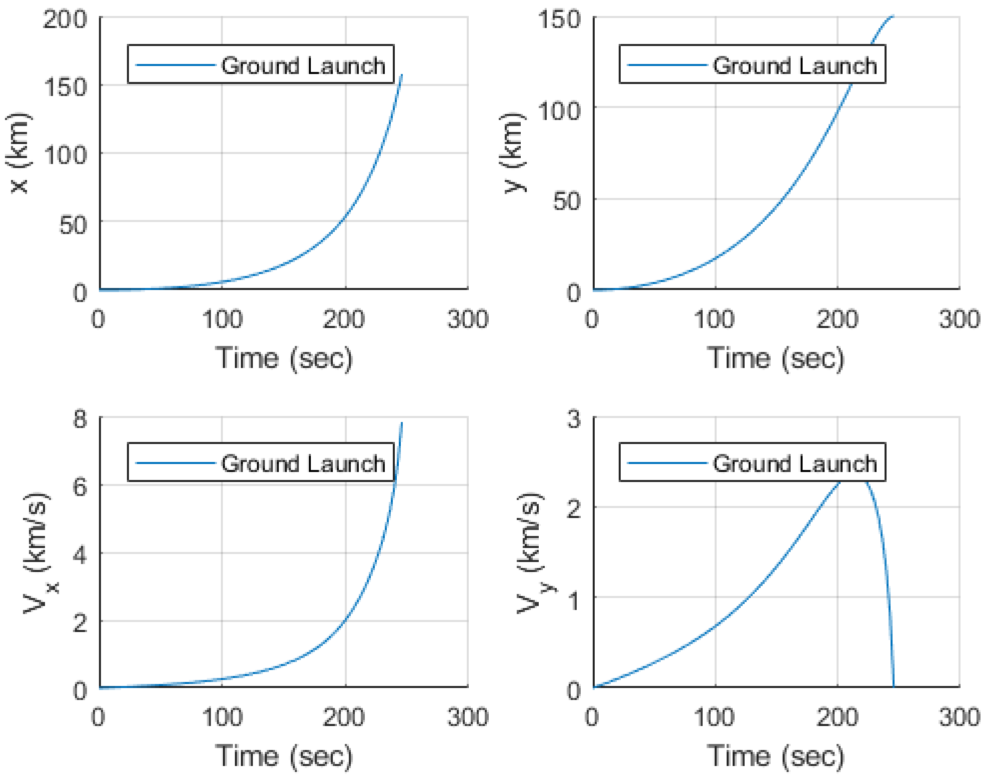
```
  140.8825


tf_mass =

  141.2180
```

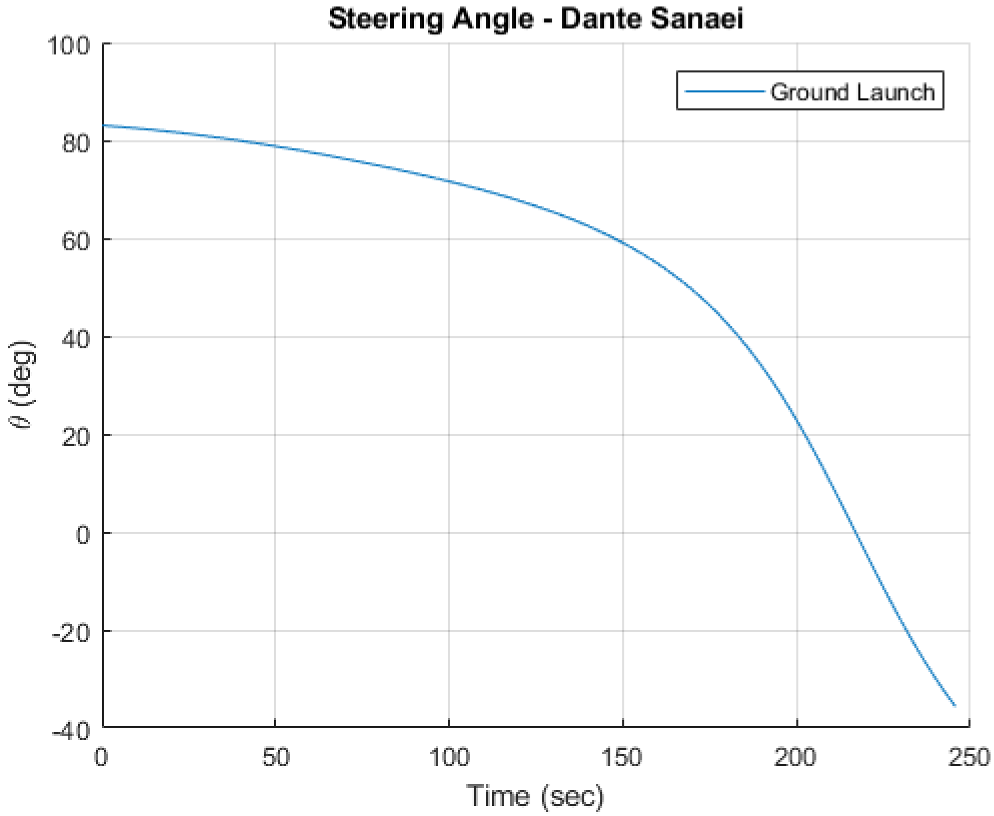
Plots

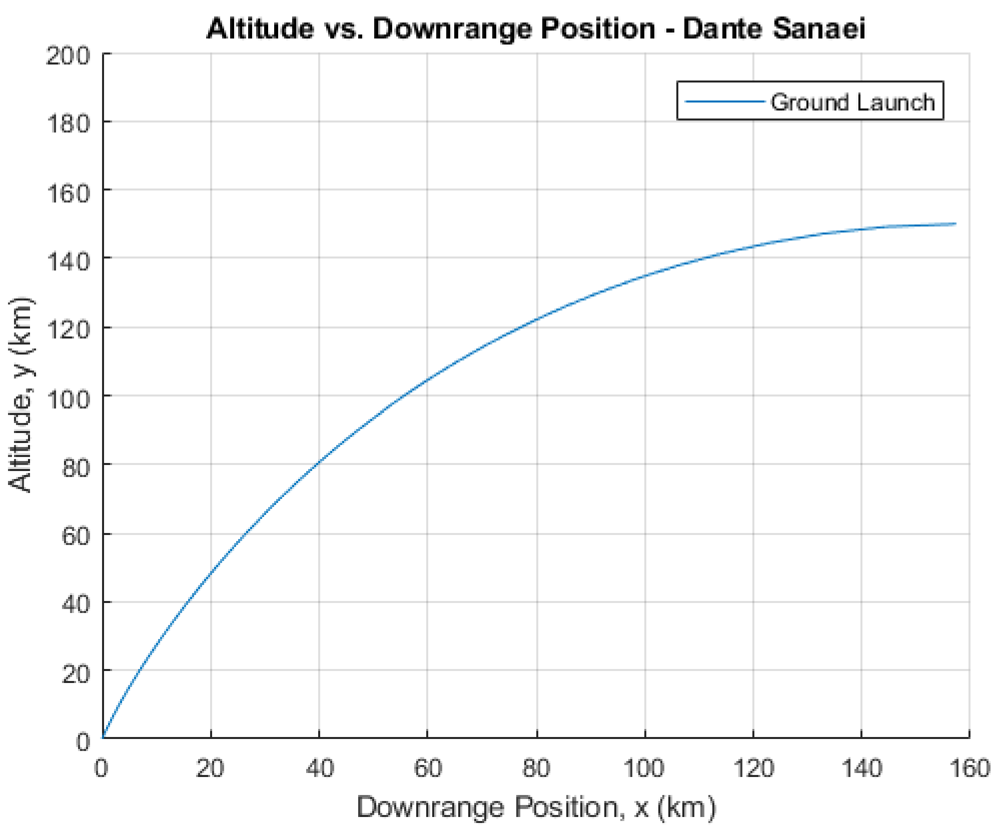
```
figure(1)
subplot(2,2,1); hold on
plot(time_mass_drag,x_sol_mass_drag); grid on
xlabel('Time (sec)')
ylabel('x (km)')
%xlim([t0 tf])
legend('Ground Launch', 'Air Launch', 'Balloon
Launch', 'location', 'northwest')

subplot(2,2,2); hold on
plot(time_mass_drag,y_sol_mass_drag); grid on
xlabel('Time (sec)')
ylabel('y (km)')
%xlim([t0 tf])
legend('Ground Launch', 'Air Launch', 'Balloon
Launch', 'location', 'northwest')

subplot(2,2,3); hold on
plot(time_mass_drag,vx_sol_mass_drag); grid on
xlabel('Time (sec)')
ylabel('V_x (km/s)')
%xlim([t0 tf])
legend('Ground Launch', 'Air Launch', 'Balloon
Launch', 'location', 'northwest')

subplot(2,2,4); hold on
plot(time_mass_drag,vy_sol_mass_drag); grid on
xlabel('Time (sec)')
ylabel('V_y (km/s)')
%xlim([t0 tf])
legend('Ground Launch', 'Air Launch', 'Balloon
Launch', 'location', 'northwest')

figure(2); hold on; grid on
plot(time_mass_drag,atand(lambda4_bar_sol_mass_drag./
lambda3_bar_sol_mass_drag)); grid on
xlabel('Time (sec)')
ylabel('\theta (deg)')
title('Steering Angle - Dante Sanaei');
%xlim([t0 tf])
legend('Ground Launch', 'Air Launch', 'Balloon Launch')
```

```
figure(3); hold on; grid on
plot(x_sol_mass_drag,y_sol_mass_drag);
xlabel('Downrange Position, x (km)');
ylabel('Altitude, y (km)');
title('Altitude vs. Downrange Position - Dante Sanaei');
%xlim([x_sol_mass_drag(1) x_sol_mass_drag(end)]);
ylim([0 200]);
legend('Ground Launch', 'Air Launch', 'Balloon Launch')
```

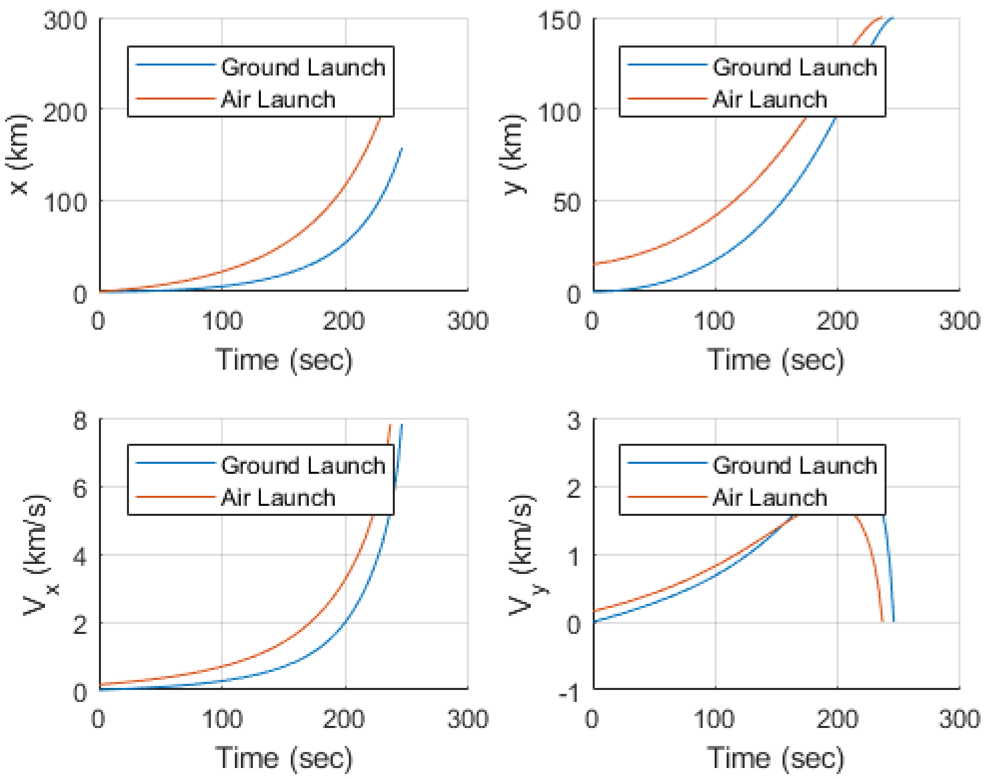
```
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
```

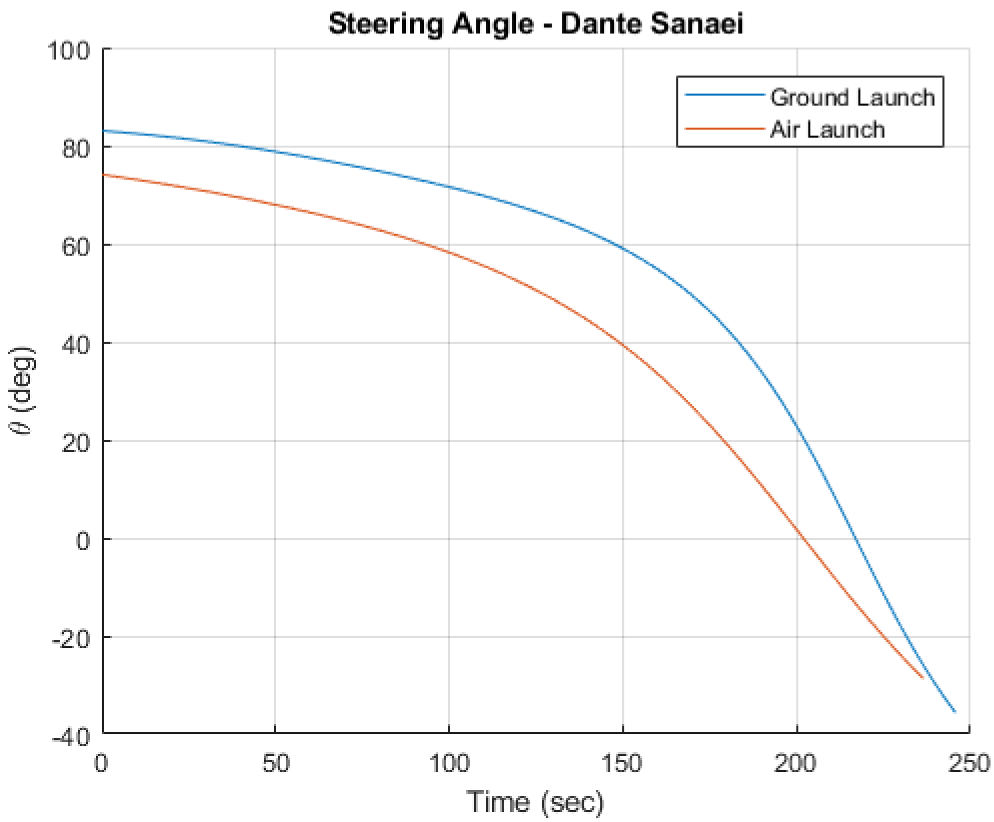



Steering Angle - Dante Sanaei
Ground Launch
θ (deg)
Time (sec)
100
80
60
40
20
0
-20
-40
0
50
100
150
200
250

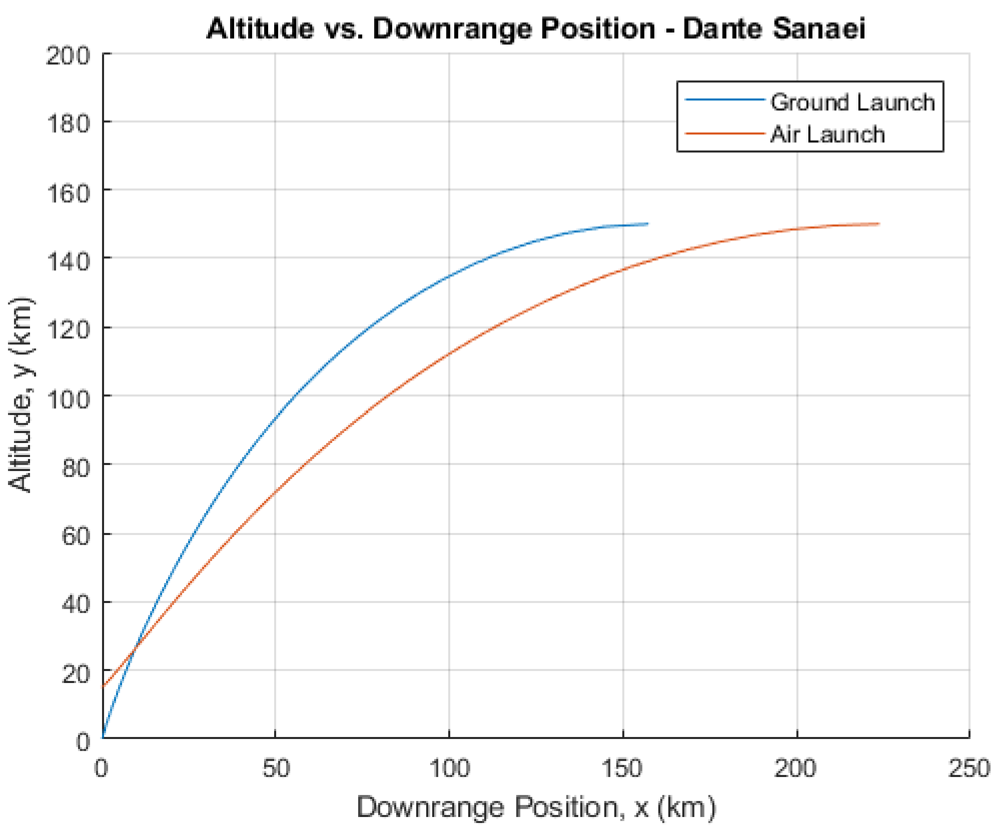

Altitude vs. Downrange Position - Dante Sanaei
Ground Launch
Altitude, y (km)
Downrange Position, x (km)
200
180
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140
160

```
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
```

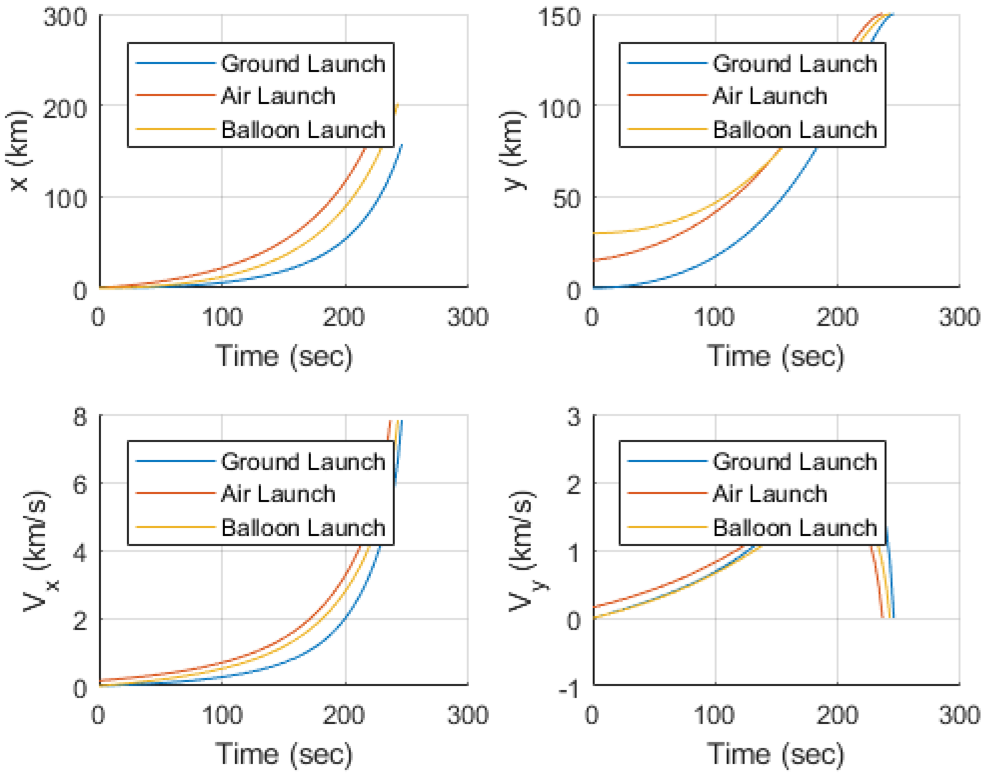


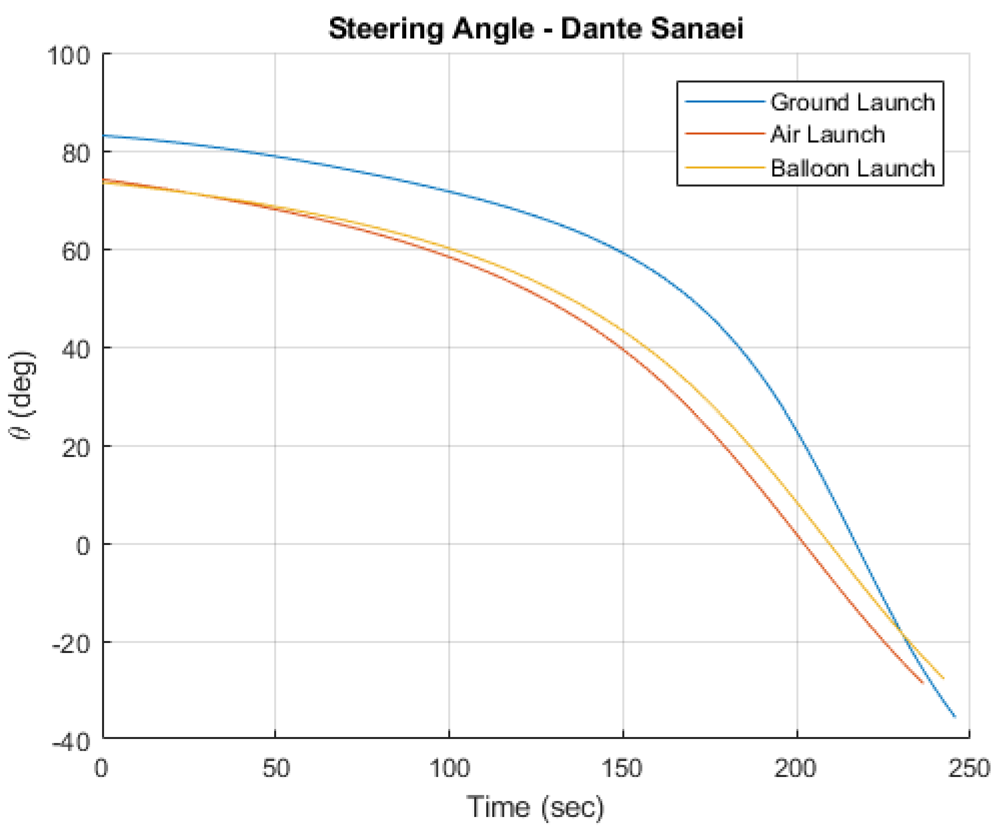


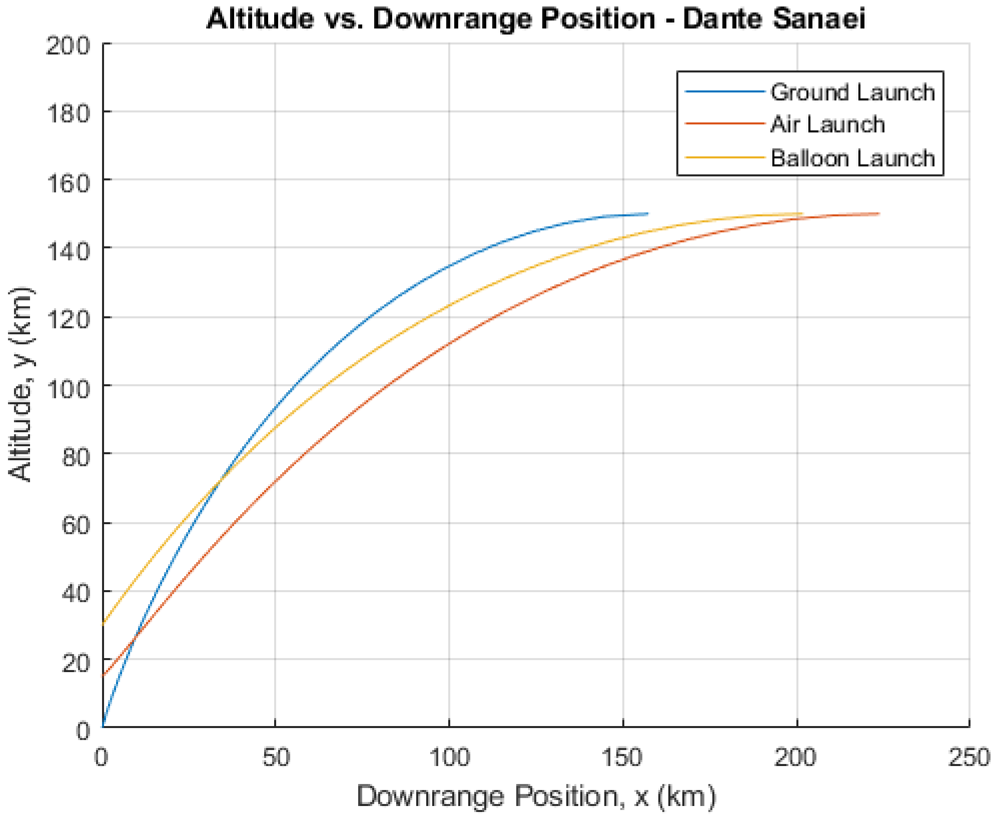
Steering Angle - Dante Sanaei
Ground Launch
Air Launch
θ (deg)
Time (sec)
100
80
60
40
20
0
-20
-40
0
50
100
150
200
250



Altitude vs. Downrange Position - Dante Sanaei
Ground Launch
Air Launch
Altitude, y (km)
Downrange Position, x (km)
200
180
160
140
120
100
80
60
40
20
0
0
50
100
150
200
250


Ground Launch
Air Launch
Balloon Launch
x (km)
y (km)
V_x (km/s)
V_y (km/s)
Time (sec)


Steering Angle - Dante Sanaei
Ground Launch
Air Launch
Balloon Launch
θ (deg)
Time (sec)



Nozzle Efficiency (only for ground launch)

```
    if iter == 1
        % Variable Thrust
        variable_thrust = 1;
        m0 = 117020;
        mdot = 807.5;
        delta_tf = 114;
        CD = 0.5;
        K1 = rho_ref*CD*A*Vc/(2*m0);

        solinit_mass_drag = solinit_mass;
        solinit_mass_drag.y = Z_mass;
        tf_mass = sol_mass.parameters(1)
        solinit_mass_drag.parameters(1) = tf_mass;

        sol_mass_drag = bvp4c(@ascent_odes_tf, @ascent_bcs_tf,
solinit_mass_drag);
        % Extract the final time from the solution:
        tf_mass_drag = sol_mass_drag.parameters(1)
        % Evaluate the solution
        Z_mass_drag = deval(sol_mass_drag,tau);
        % Convert back to dimensional time for plotting
        time_mass_drag_vtrhust = t0 + tau.*(tf-t0);
```

```
        x_sol_mass_drag_vtrhust = Z_mass_drag(1,:)*h/1000;
        y_sol_mass_drag_vtrhust = Z_mass_drag(2,:)*h/1000;
        vx_sol_mass_drag_vtrhust = Z_mass_drag(3,:)*Vc/1000;
        vy_sol_mass_drag_vtrhust = Z_mass_drag(4,:)*Vc/1000;
        lambda2_bar_sol_mass_drag_vtrhust = Z_mass_drag(5,:);
        lambda3_bar_sol_mass_drag_vtrhust = Z_mass_drag(6,:);
        lambda4_bar_sol_mass_drag_vtrhust = Z_mass_drag(7,:);

        % Variable Thrust Comparison
        Alt_diff= mean(abs(y_sol_mass_drag_vtrhust-y_sol_mass_drag))
        SA_diff = mean(abs(atand(lambda4_bar_sol_mass_drag_vtrhust./
lambda3_bar_sol_mass_drag_vtrhust)-atand(lambda4_bar_sol_mass_drag./
lambda3_bar_sol_mass_drag)))
        Vx_diff = mean(abs(vx_sol_mass_drag_vtrhust-vx_sol_mass_drag))
        Vy_diff =mean(abs(vy_sol_mass_drag_vtrhust-vy_sol_mass_drag))
    end
```

```
tf_mass =

  142.2460


tf_mass_drag =

  142.4434


Alt_diff =

     0


SA_diff =

     0


Vx_diff =

     0


Vy_diff =

     0
```

Final Parameters

```
    final_x = x_sol_mass_drag(end)
    final_y = y_sol_mass_drag(end)
    final_vx = vx_sol_mass_drag(end)
```

```
    final_vy = vy_sol_mass_drag(end)
    final_time = tf


    if iter ~= 3
       clear all
    end
```

```
final_x =

  157.3430


final_y =

   150


final_vx =

    7.8140


final_vy =

   4.2484e-19


final_time =

  245.6584



final_x =

  223.8041


final_y =

   150


final_vx =

    7.8140


final_vy =

  -6.2051e-22
```

```
final_time =

  236.4157


final_x =

  201.8151


final_y =

   150


final_vx =

    7.8140


final_vy =

   4.1367e-22


final_time =

  242.4526

end
```

PART TWO: Time Optimal and Range Optimal Aircraft Launches of SpaceShipTwo and GO-Launcher One

```
% The time optimal simulation portion of this script is derived from
 Appendix B from "Optimal Controls With
% Aerospace Applications".
% Drag on the vehicle and varying mass are calculated for time optimal
 trajectory.
```

Time-Optimal Trajectory to LEO for SpaceShipTwo and GOLauncher

```
for iter = 1:2
    global g Vc h K1 scale F m0 mdot xbar0 ybar0 Vxbar0 Vybar0 ybarf
 Vxbarf Vybarf
```

```
% Boundary Conditions for SpaceShipTwo and GOLauncher
%SpaceShipTwo Conditions
if iter == 1
    h_init = 15000; % m ,initial altitude
    V_init = 55; % m/s, initial velocity
    h = 150000; % m, final altitude
    Vc = sqrt(3.9860044e5/(6378.14+h/1000))*1000; % m/s, final
circular speed
    g = 9.80665; % m/s^2, gravity
    F = 2.7e6; % N, constant thrust
    h_scale = 8440; % m, atmospheric scale-height
    scale = h/h_scale; % constant to simplify EOM
    rho_ref = 1.225; % reference density
    r = 2.3/2; %m, radius of crew cabin
    A = pi*r^2; % m^22, cross-sectional area

    % Initial conditions
    xbar0 = 0; % initial x-position
    ybar0 = h_init/h; % initial y-position
    Vxbar0 = V_init/Vc; % initial downrange velocity
    Vybar0 = 0; % initial vertical velocity

    % Final conditions
    ybarf = h/h; % final altitude
    Vxbarf = Vc/Vc; % final downrange velocity
    Vybarf = 0; % final vertical velocity
    Hf = -1; %final value of Hamiltonian
end
%GOLauncher Conditions
if iter == 2
    h_init = 10e3; % m ,initial altitude
    V_init = 220; % m/s, initial velocity
    h = 150000; % m, final altitude
    Vc = sqrt(3.9860044e5/(6378.14+h/1000))*1000; % m/s, final
circular speed
    g = 9.80665; % m/s^2, gravity
    F = 2.5e6; % N, constant thrust
    h_scale = 8440; % m, atmospheric scale-height
    scale = h/h_scale; % constant to simplify EOM
    rho_ref = 1.225; % reference density
    r = .5/2; %m, radius of crew cabin
    A = pi*r^2; % m^22, cross-sectional area

    % Initial conditions
    xbar0 = 0; % initial x-position
    ybar0 = h_init/h; % initial y-position
    Vxbar0 = V_init/Vc; % initial downrange velocity
    Vybar0 = 0; % initial vertical velocity

    % Final conditions
    ybarf = h/h; % final altitude
    Vxbarf = Vc/Vc; % final downrange velocity
    Vybarf = 0; % final vertical velocity
    Hf = -1; %final value of Hamiltonian
```

```
    end


% Solution
    % CONSTANT MASS / NO DRAG Parameters
    m0 = 60880; % kg, average mass of rocket
    CD = 0; % no drag case has Cd = 0
    mdot = 0;
    K1 = rho_ref*CD*A*Vc/(2*m0); %constant to simplify EOM

    % Initial Guesses
    t0 = 0;
    yinit = [xbar0 ybar0 Vxbar0 Vybar0 0 -1 0]; %lambda20 lambda30
 lambda40
    tf_guess = 500; % sec, initial guess for final time
    Nt = 80;
    tau = linspace(0,1,Nt)'; % nondimensional time vector

    solinit = bvpinit(tau,yinit,tf_guess);

    % Solution - CONSTANT MASS / NO DRAG

    sol = bvp4c(@ascent_odes_tf, @ascent_bcs_tf, solinit);
    % Extract the final time from the solution:
    tf = sol.parameters(1);
    % Evaluate the solution
    Z = deval(sol,tau);
    % Convert back to dimensional time for plotting
    time = t0 + tau.*(tf-t0);

    x_sol = Z(1,:)*h/1000;
    y_sol = Z(2,:)*h/1000;
    vx_sol = Z(3,:)*Vc/1000;
    vy_sol = Z(4,:)*Vc/1000;
    lambda2_bar_sol = Z(5,:);
    lambda3_bar_sol = Z(6,:);
    lambda4_bar_sol = Z(7,:);

    % Solution - VARIABLE MASS / NO DRAG
    if iter == 1
        m0 = 13154.179;
        mdot = 112.65;
        delta_tf = 114;
        CD = 0;
        K1 = rho_ref*CD*A*Vc/(2*m0);
    end
    if iter == 2
        m0 = 5700;
        mdot = 65;
        delta_tf = 154;
        CD = 0;
        K1 = rho_ref*CD*A*Vc/(2*m0);
    end
```

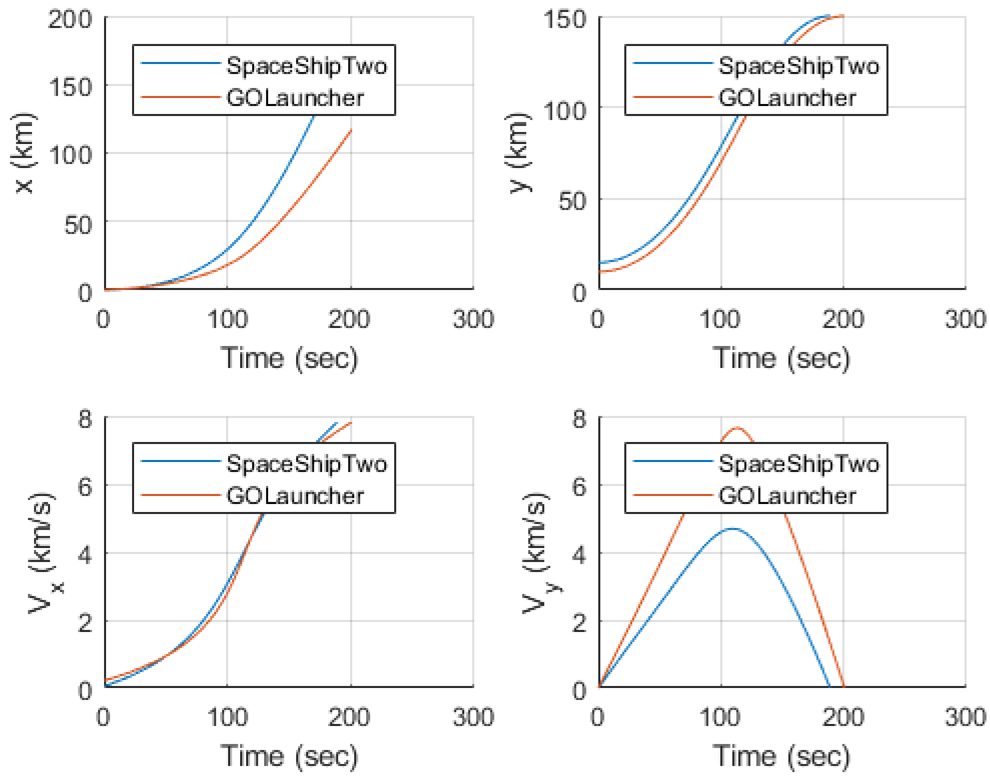
```
solinit_mass = solinit;
solinit_mass.y = Z;
tf = sol.parameters(1);
solinit_mass.parameters(1) = tf-delta_tf;

sol_mass = bvp4c(@ascent_odes_tf, @ascent_bcs_tf, solinit_mass);
% Extract the final time from the solution:
tf_mass = sol_mass.parameters(1);
% Evaluate the solution
Z_mass = deval(sol_mass,tau);
% Convert back to dimensional time for plotting
time_mass = t0 + tau.*(tf-t0);

x_sol_mass = Z_mass(1,:)*h/1000;
y_sol_mass = Z_mass(2,:)*h/1000;
vx_sol_mass = Z_mass(3,:)*Vc/1000;
vy_sol_mass = Z_mass(4,:)*Vc/1000;
lambda2_bar_sol_mass = Z_mass(5,:);
lambda3_bar_sol_mass = Z_mass(6,:);
lambda4_bar_sol_mass = Z_mass(7,:);


% Solution - VARIABLE MASS / WITH DRAG
if iter == 1
    m0 = 13154.179;
    mdot = 112.65;
    delta_tf = 114;
    CD = .3;
    K1 = rho_ref*CD*A*Vc/(2*m0);
end
if iter == 2
    m0 = 5700;
    mdot = 65;
    delta_tf = 154;
    CD = .1;
    K1 = rho_ref*CD*A*Vc/(2*m0);
end

solinit_mass_drag = solinit_mass;
solinit_mass_drag.y = Z_mass;
tf_mass = sol_mass.parameters(1);
solinit_mass_drag.parameters(1) = tf_mass;

sol_mass_drag = bvp4c(@ascent_odes_tf, @ascent_bcs_tf,
solinit_mass_drag);
% Extract the final time from the solution:
tf_mass_drag = sol_mass_drag.parameters(1);
% Evaluate the solution
Z_mass_drag = deval(sol_mass_drag,tau);
% Convert back to dimensional time for plotting
time_mass_drag = t0 + tau.*(tf-t0);

x_sol_mass_drag = Z_mass_drag(1,:)*h/1000;
y_sol_mass_drag = Z_mass_drag(2,:)*h/1000;
```

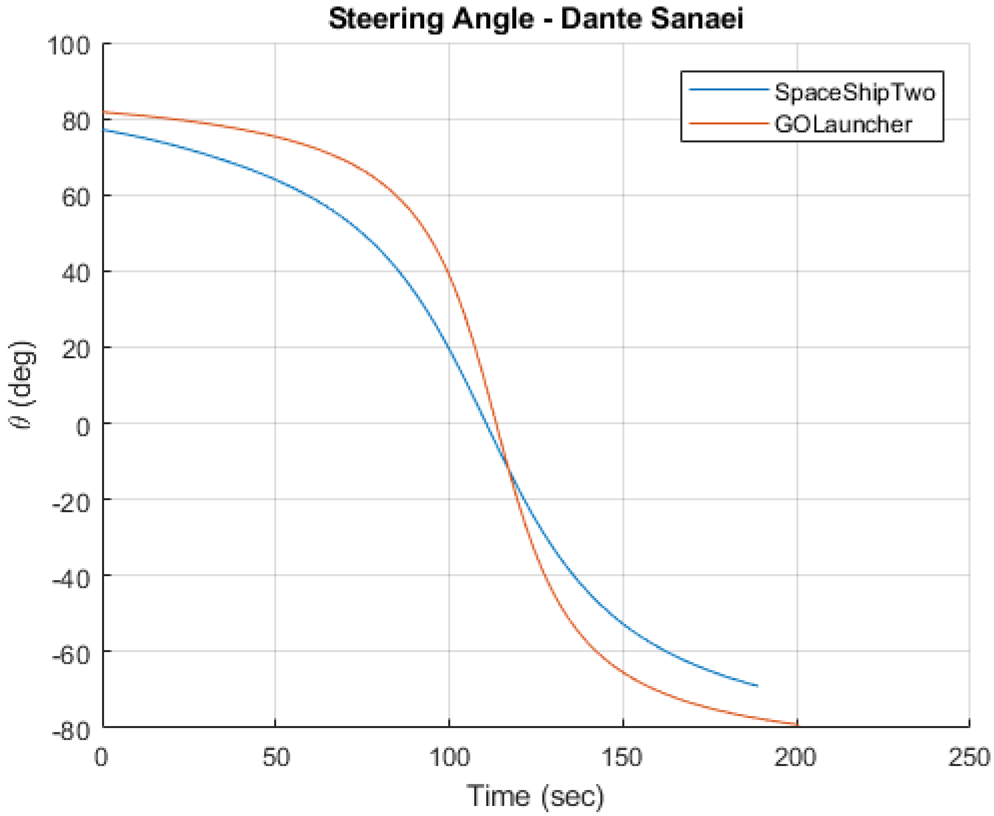
```
    vx_sol_mass_drag = Z_mass_drag(3,:)*Vc/1000;
    vy_sol_mass_drag = Z_mass_drag(4,:)*Vc/1000;
    lambda2_bar_sol_mass_drag = Z_mass_drag(5,:);
    lambda3_bar_sol_mass_drag = Z_mass_drag(6,:);
    lambda4_bar_sol_mass_drag = Z_mass_drag(7,:);

% Plots
    figure(4)
    subplot(2,2,1); hold on
    plot(time_mass_drag,x_sol_mass_drag); grid on
    xlabel('Time (sec)')
    ylabel('x (km)')
    %xlim([t0 tf])
    legend('SpaceShipTwo', 'GOLauncher', 'location', 'northwest')

    subplot(2,2,2); hold on
    plot(time_mass_drag,y_sol_mass_drag); grid on
    xlabel('Time (sec)')
    ylabel('y (km)')
    %xlim([t0 tf])
    legend('SpaceShipTwo', 'GOLauncher', 'location', 'northwest')

    subplot(2,2,3); hold on
    plot(time_mass_drag,vx_sol_mass_drag); grid on
    xlabel('Time (sec)')
    ylabel('V_x (km/s)')
    %xlim([t0 tf])
    legend('SpaceShipTwo', 'GOLauncher', 'location', 'northwest')

    subplot(2,2,4); hold on
    plot(time_mass_drag,vy_sol_mass_drag); grid on
    xlabel('Time (sec)')
    ylabel('V_y (km/s)')
    %xlim([t0 tf])
    legend('SpaceShipTwo', 'GOLauncher', 'location', 'northwest')

    figure(5); hold on; grid on
    plot(time_mass_drag,atand(lambda4_bar_sol_mass_drag./
lambda3_bar_sol_mass_drag)); grid on
    xlabel('Time (sec)')
    ylabel('\theta (deg)')
    title('Steering Angle - Dante Sanaei');
    %xlim([t0 tf])
    legend('SpaceShipTwo', 'GOLauncher')

    figure(6); hold on; grid on
    plot(x_sol_mass_drag,y_sol_mass_drag);
    xlabel('Downrange Position, x (km)');
    ylabel('Altitude, y (km)');
    title('Altitude vs. Downrange Position - Dante Sanaei');
    %xlim([x_sol_mass_drag(1) x_sol_mass_drag(end)]);
    ylim([0 200]);
    legend('SpaceShipTwo', 'GOLauncher')
```

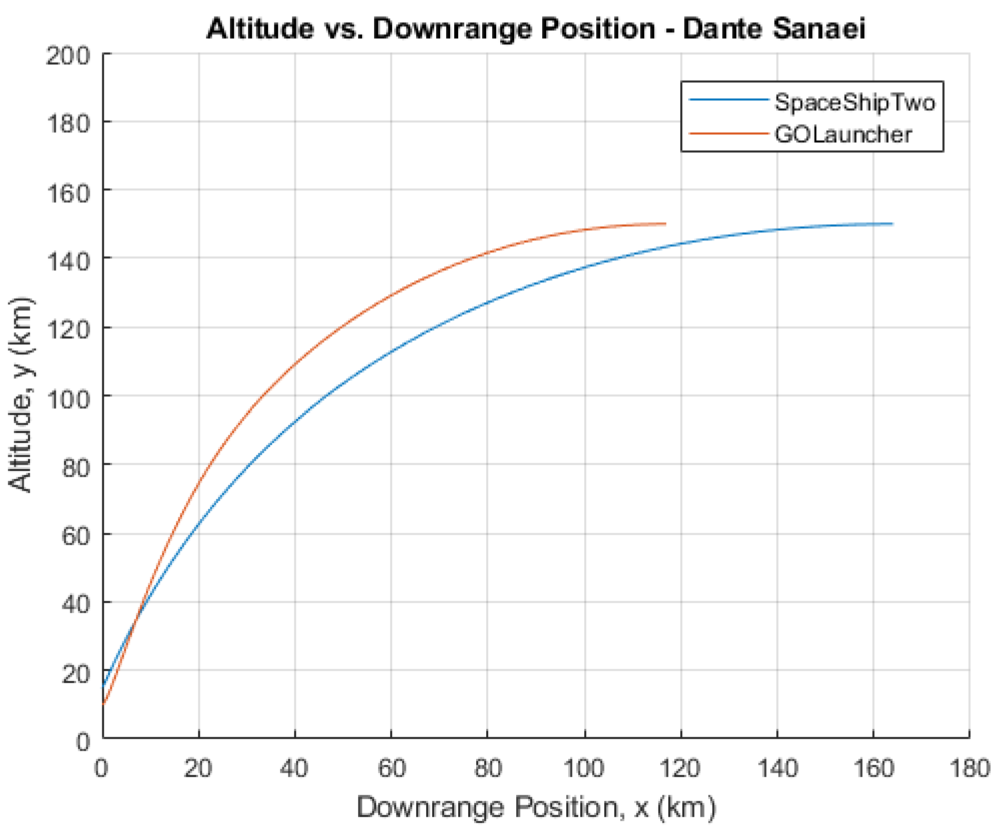
```
% Final Parameters
    final_x = x_sol_mass_drag(end)
    final_y = y_sol_mass_drag(end)
    final_vx = vx_sol_mass_drag(end)
    final_vy = vy_sol_mass_drag(end)
    final_time = tf


    clear all
end
```

```
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.

final_x =

  163.9588


final_y =

   150


final_vx =

    7.8140


final_vy =

   1.0342e-22


final_time =

  188.9313


final_x =

  116.9818


final_y =

   150


final_vx =
```

```
    7.8140

final_vy =

   2.8804e-30

final_time =

  200.6102
```




Steering Angle - Dante Sanaei
SpaceShipTwo
GOLauncher
θ (deg)
Time (sec)
100
80
60
40
20
0
-20
-40
-60
-80
0
50
100
150
200
250


Altitude vs. Downrange Position - Dante Sanaei
SpaceShipTwo
GOLauncher
Altitude, y (km)
Downrange Position, x (km)
200
180
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140
160
180

Optimal Range Trajectory For SpaceShipTwo and GOLauncher

```
time_in_space_S1 = [];
time_in_space_GO = [];
angles_S1 = [];
angles_GO = [];

for iter = 1:2
    if iter == 1
        g = 9.81;
        mdot = 112;
        F = 270e3;
        Ispm = F/mdot;
        m = 13154.179/1.35;
        f = (mdot * Ispm)/m
        T = 87;
        g2f = g/f;
        y_init = 15e3;
        V0 = 55;
    end
    if iter == 2
        g = 9.81;
        mdot = 65;
        F = 250e3;
        Ispm = F/mdot;
        m = 5700/1.35;
        f = (mdot * Ispm)/m;
        T = 60;
        g2f = g/f;
        y_init = 10e3;
        V0 = 220;
    end

    for i = 0:.001:pi/2
        anglform = g2f*sin(i)^3 - 2*sin(i)^2 + 1;
        i;
        if  anglform <= 0.001 && anglform >= -.001
            anglform;
            optimal_range_theta = i;
            rad2deg(i);
        end
    end
    final_time = 800;

    for angle = 0:1:80
        theta = deg2rad(angle);
        Vx0 = V0*cos(theta);
        Vy0 = V0*sin(theta);
        Vx1 = (f*T*cos(theta)) + Vx0;
        Vy1 = (f*sin(theta)-g)*T + Vy0;
        x1 = .5*f*T^2*cos(theta);
```

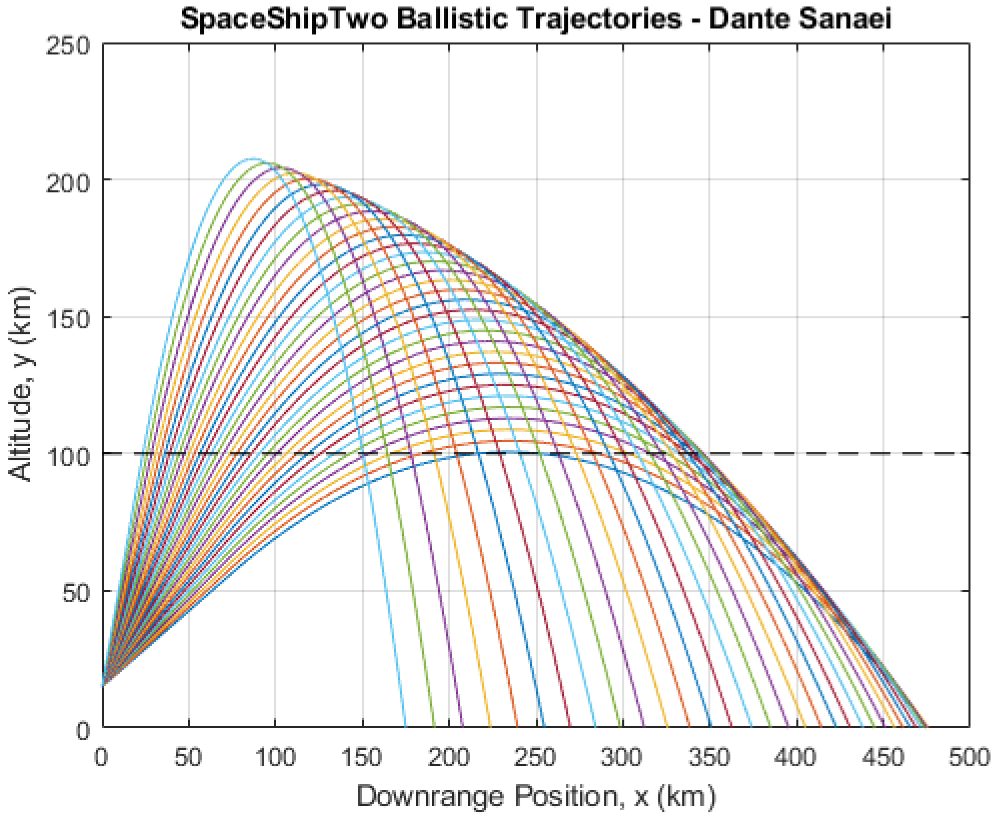
```
        y1 = .5*(f*sin(theta)-g)*T^2 + y_init;

        if  isreal(-(sqrt(Vy1^2 + 2 * g * (y1 - 100e3)) - Vy1) / g)
            u = sqrt(Vx1^2+Vy1^2);
            x_burn = 0:.1:x1;
            slope = (y1-y_init) / x1;
            y_burn = slope * x_burn + y_init;
            time = 0:.1:final_time;
            x_coast = x1 + Vx1*time ;
            y_coast = y1 + Vy1.*time - .5 * g * time.^2;
            k = find(y_coast >-.01,1, 'last');

            x = [x_burn x_coast(1:k)] / 1000;
            y = [y_burn y_coast(1:k)] / 1000;

            if iter == 1
                figure(7)
                plot(x,y); grid on; hold on;
                xlabel('Downrange Position, x (km)');
                ylabel('Altitude, y (km)');
                title('SpaceShipTwo Ballistic Trajectories - Dante
Sanaei');
                plot([min(xlim()),max(xlim())],[100,100], 'k--')

                ylim([0 250])
                coasting_time = (sqrt(Vy1^2 + 2 * g * (y1 - 100e3)) +
Vy1) / g;
                time_in_space_S1 = [time_in_space_S1 coasting_time];
                angles_S1 = [angles_S1 rad2deg(theta)];
                if angle == 65
                    x_optimal_space_S2 = x;
                    y_optimal_space_S2 = y;
                    x1_S2 = x1;
                    y1_S2 = y1;
                    coast_time_OS_S2 = (sqrt(Vy1^2 + 2 * g * (y1 -
100e3)) + Vy1) / g;
                    apogee_S2_OS = max(y)
                    x_coast(k)
                end
            end
            if iter == 2
                figure(8)
                plot(x,y); grid on; hold on;
                xlabel('Downrange Position, x (km)');
                ylabel('Altitude, y (km)');
                title('GOLauncher Ballistic Trajectories - Dante
Sanaei');
                plot([min(xlim()),max(xlim())],[100,100], 'k--')
                ylim([0 800])
                coasting_time = (sqrt(Vy1^2 + 2 * g * (y1 - 100e3)) +
Vy1) / g;
                time_in_space_GO = [time_in_space_GO coasting_time];
                angles_GO = [angles_GO rad2deg(theta)];
                if angle == 70
```

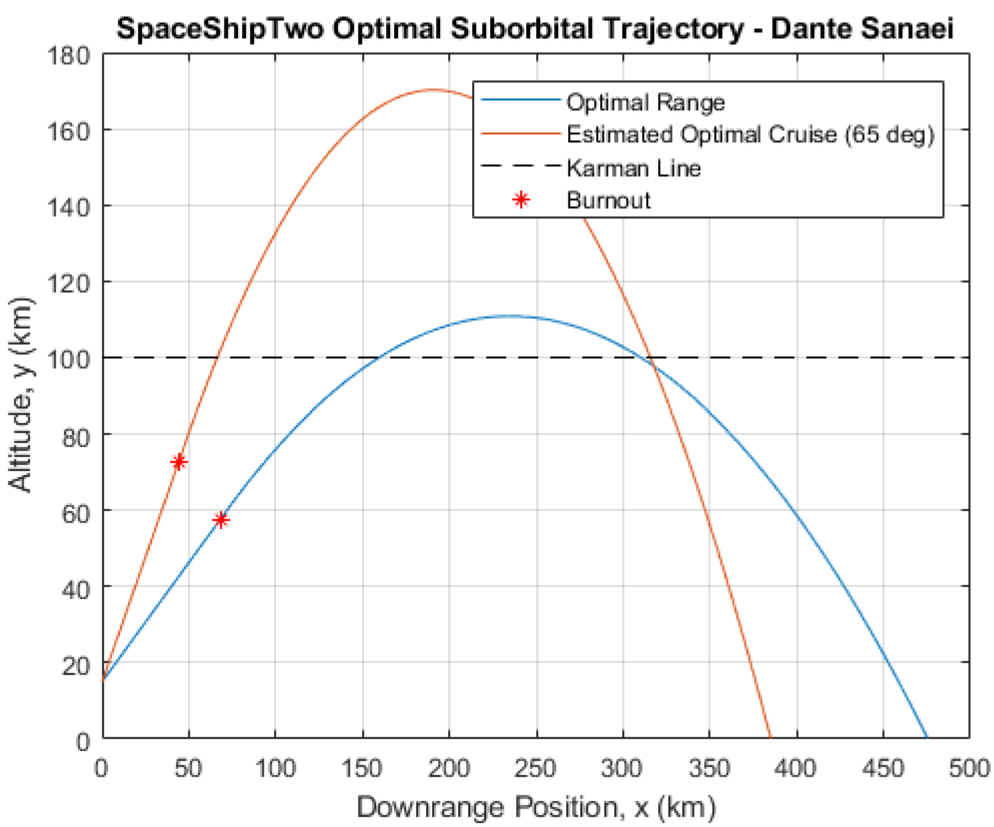
```
                    x_optimal_space_GO = x;
                    y_optimal_space_GO = y;
                    x1_GO = x1;
                    y1_GO = y1;
                    coast_time_OS_GO = (sqrt(Vy1^2 + 2 * g * (y1 -
100e3)) + Vy1) / g;
                    apogee_GO_OS = max(y)
                    x_coast(k)
                end
            end
        end
    end

    figure(9)
    subplot(2,1,1)
    plot(angles_S1, time_in_space_S1); grid on;
    title('SpaceShipTwo Time in Space - Dante Sanaei')
    ylabel('Time (sec)')
    xlabel('Launch Angle (deg)')

    subplot(2,1,2)

    plot(angles_GO, time_in_space_GO); grid on
    title('GOLauncher Time in Space - Dante Sanaei')
    ylabel('Time (sec)')
    xlabel('Launch Angle (deg)')


    theta = optimal_range_theta;
    Vx0 = V0*cos(theta);
    Vy0 = V0*sin(theta);

    Vx1 = (f*T*cos(theta)) + Vx0;
    Vy1 = (f*sin(theta)-g)*T + Vy0;
    x1 = .5*f*T^2*cos(theta);
    y1 = .5*(f*sin(theta)-g)*T^2 + y_init;

    u = sqrt(Vx1^2+Vy1^2);

    x_burn = 0:.1:x1;
    slope = (y1-y_init) / x1;
    y_burn = slope * x_burn + y_init;

    time = 0:.1:final_time;
    x_coast = x1 + Vx1*time ;
    y_coast = y1 + Vy1.*time - .5 * g * time.^2;

    k = find(y_coast >-.01,1, 'last');

    x = [x_burn x_coast(1:k)] / 1000;
    y = [y_burn y_coast(1:k)] / 1000;
    if iter == 1
        figure(10)
        plot(x,y); grid on; hold on
```

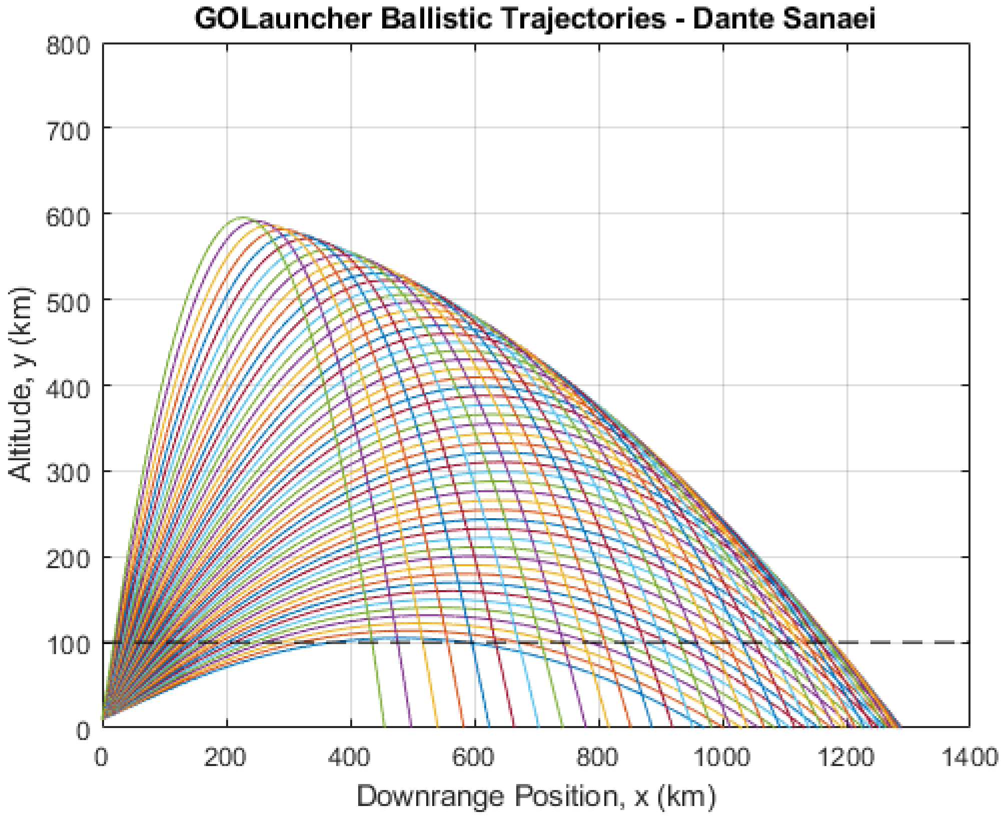
```
        apogee = max(y)
        xlabel('Downrange Position, x (km)');
        ylabel('Altitude, y (km)');
        title('SpaceShipTwo Optimal Suborbital Trajectory - Dante
Sanaei');
        plot(x_optimal_space_S2, y_optimal_space_S2)
        plot([min(xlim()),max(xlim())],[100,100], 'k--');
plot([x1_S2/1000 x1/1000],[y1_S2/1000 y1/1000], 'r*');
        legend('Optimal Range', 'Estimated Optimal Cruise (65
deg)', 'Karman Line', 'Burnout')

        coast_time = (sqrt(Vy1^2 + 2 * g * (y1 - 100e3)) + Vy1) / g
        max_range_formula = f*T^2 * ( f/g * cot(theta) - .5 *
cos(theta))
        max_range_real = x_coast(k)
        coast_time_OS_S2
        rad2deg(optimal_range_theta)

        figure(12)
        plot(x,y); grid on; hold on
        xlabel('Downrange Position, x (km)');
        ylabel('Altitude, y (km)');
        S2fg = f/g
        x_iter1 = x1;
        y_iter1 = y1;

    end
    if iter == 2
        figure(11)
        plot(x,y); grid on; hold on;
        apogee = max(y)
        xlabel('Downrange Position, x (km)');
        ylabel('Altitude, y (km)');
        title('GOLauncher Optimal Suborbital Trajectory - Dante
Sanaei');
        plot(x_optimal_space_GO, y_optimal_space_GO)
        plot([min(xlim()),max(xlim())],[100,100], 'k--');
plot([x1_GO/1000 x1/1000],[y1_GO/1000 y1/1000], 'r*');

        coast_time = (sqrt(Vy1^2 + 2 * g * (y1 - 100e3)) + Vy1) / g
        max_range_formula = f*T^2 * ( f/g * cot(theta) - .5 *
cos(theta))
        max_range_real = x_coast(k)
        coast_time_OS_GO
        rad2deg(optimal_range_theta)

        figure(12)
        plot(x,y); grid on; hold on;
        xlabel('Downrange Position, x (km)');
        ylabel('Altitude, y (km)');
        title('Range Optimized Trajectory - Dante Sanaei');
        plot([min(xlim()),max(xlim())],[100,100], 'k--');
plot([x_iter1/1000 x1/1000],[y_iter1/1000 y1/1000], 'r*');
```

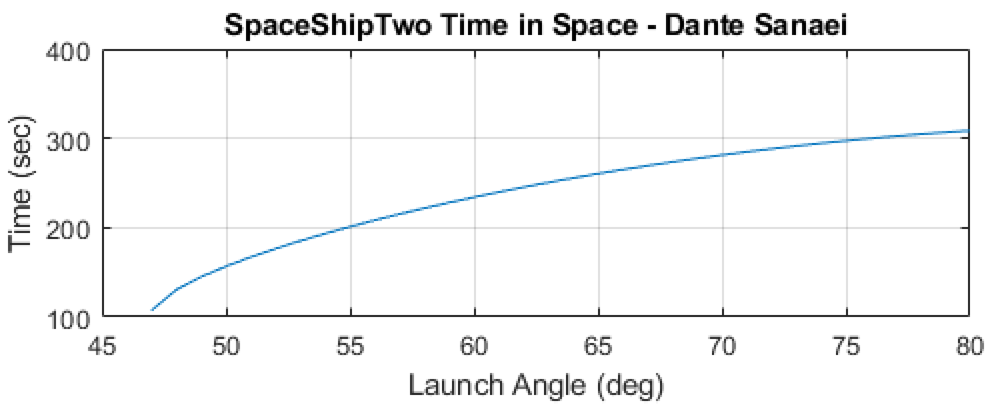
```
        legend('SpaceShipTwo (49.5)', 'GOLauncher (46.87)', 'Karman
 Line', 'Burnout')
        GOfg = f/g

    end

end


f =

   27.7098


apogee_S2_OS =

  170.1585


ans =

   3.8508e+05


apogee =

  110.8148


coast_time =

  151.0950


max_range_formula =

   4.3782e+05


max_range_real =

   4.7546e+05


coast_time_OS_S2 =

  260.3986


ans =

   49.5036
```

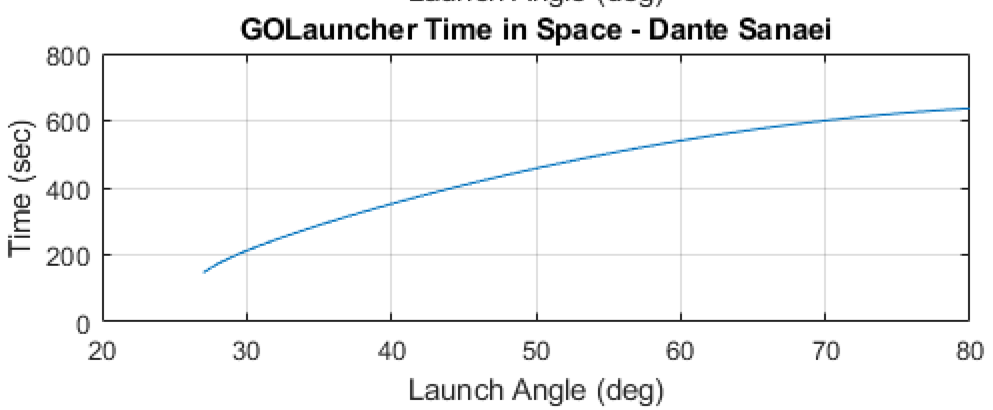
```
S2fg =

    2.8247


apogee_GO_OS =

  538.0070


ans =

   8.5258e+05


apogee =

  308.9312


coast_time =

  427.0384


max_range_formula =

   1.1324e+06


max_range_real =

   1.2893e+06


coast_time_OS_GO =

  600.2054


ans =

   46.8679


GOfg =

    6.0357
```

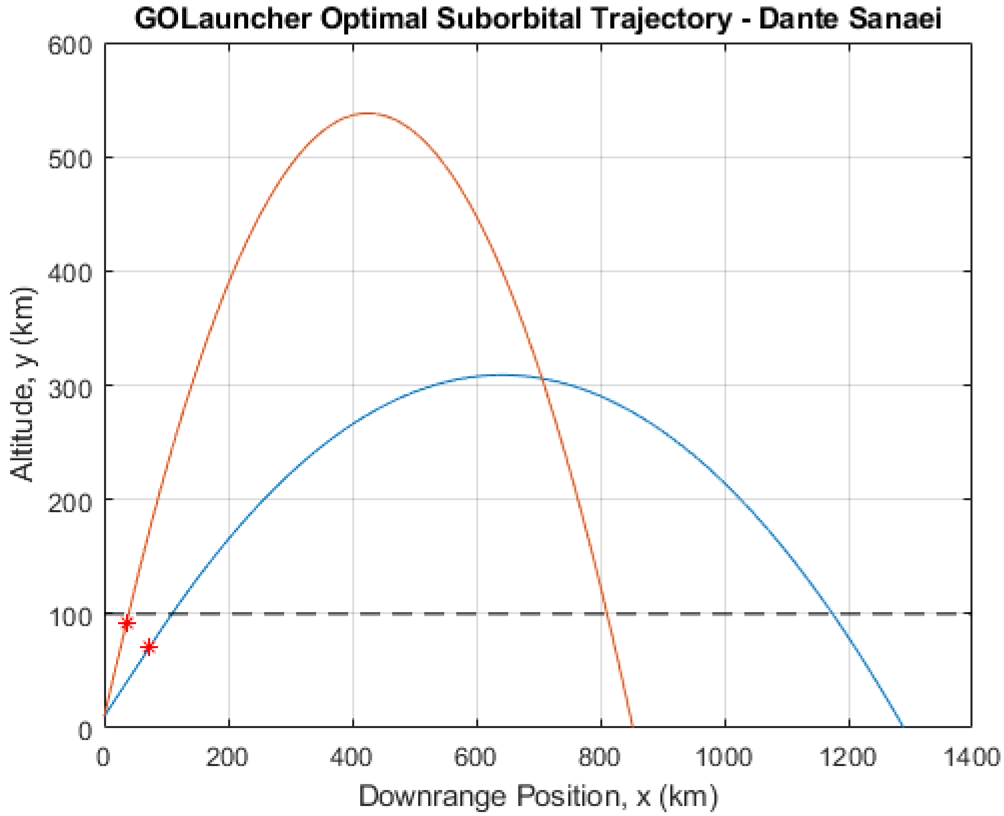

SpaceShipTwo Ballistic Trajectories - Dante Sanaei
Altitude, y (km)
Downrange Position, x (km)
0
50
100
150
200
250
300
350
400
450
500

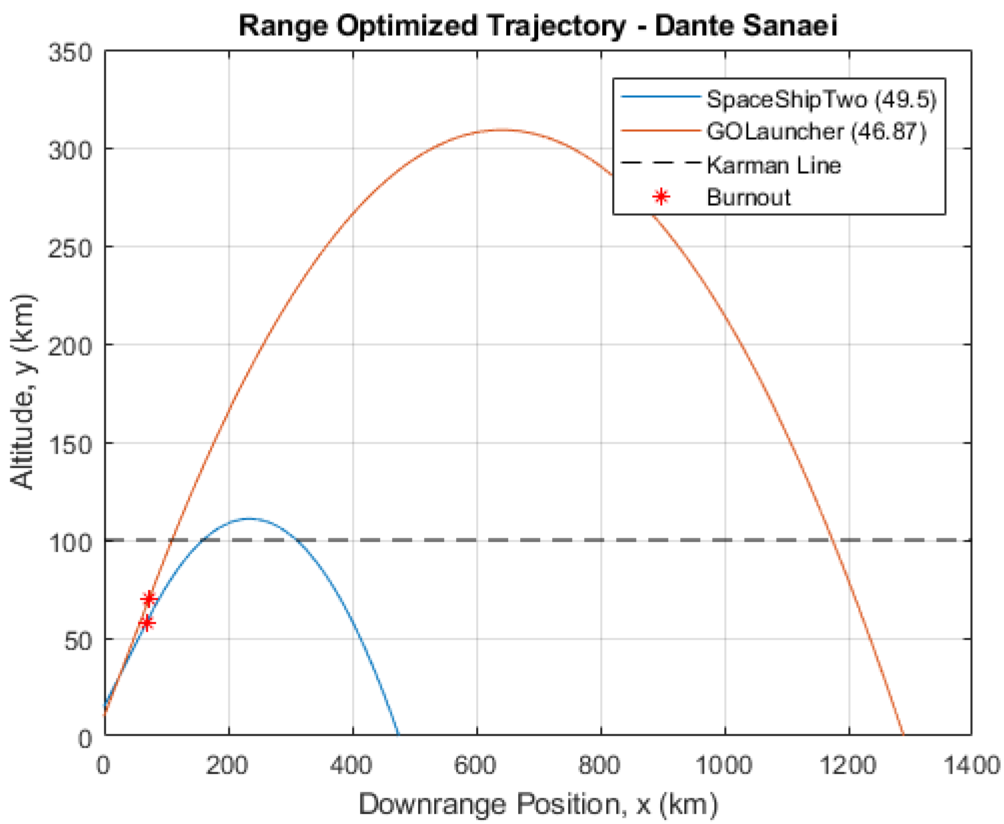

SpaceShipTwo Optimal Suborbital Trajectory - Dante Sanaei
Optimal Range
Estimated Optimal Cruise (65 deg)
Karman Line
Burnout
Altitude, y (km)
Downrange Position, x (km)


GOLauncher Ballistic Trajectories - Dante Sanaei
Altitude, y (km)
Downrange Position, x (km)
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200
1400


SpaceShipTwo Time in Space - Dante Sanaei
Time (sec)
Launch Angle (deg)
100
200
300
400
45
50
55
60
65
70
75
80


GOLauncher Time in Space - Dante Sanaei
Time (sec)
Launch Angle (deg)
0
200
400
600
800
20
30
40
50
60
70
80


GOLauncher Optimal Suborbital Trajectory - Dante Sanaei
Altitude, y (km)
Downrange Position, x (km)
0
100
200
300
400
500
600
0
200
400
600
800
1000
1200
1400


Range Optimized Trajectory - Dante Sanaei
SpaceShipTwo (49.5)
GOLauncher (46.87)
Karman Line
Burnout
Altitude, y (km)
Downrange Position, x (km)
0
50
100
150
200
250
300
350
0
200
400
600
800
1000
1200
1400

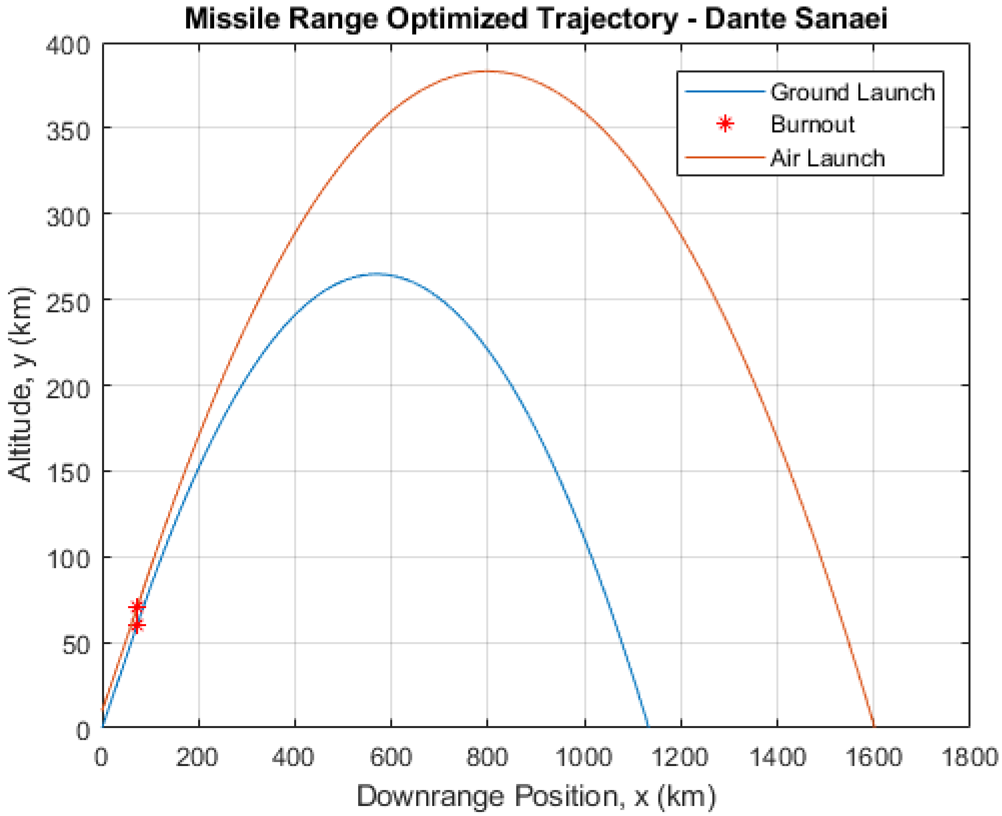
PART THREE: Range Optimal Missile Launch From Ground and Air

```
% Drag on the vehicle and varying mass are not calculated.
```

Range of Ground vs Air Missile Launch

```
final_time=1000;
for groundair = 1:2
    g = 9.81;
    mdot = 65;
    F = 250e3;
    Ispm = F/mdot;
    m = 5700/1.35;
    f = (mdot * Ispm)/m;
    T = 60;
    g2f = g/f;
    if groundair == 1
        y_init = 0;
        V0 = 0;
    end
    if groundair == 2
        y_init = 10e3;
        V0 = 650;
    end
   for i = 0:.001:pi/2
        anglform = g2f*sin(i)^3 - 2*sin(i)^2 + 1;
        i;
        if  anglform <= 0.001 && anglform >= -.001
            anglform;
            optimal_range_theta = i;
            rad2deg(i);
        end
   end
    theta = optimal_range_theta;
    Vx0 = V0*cos(theta);
    Vy0 = V0*sin(theta);

    Vx1 = (f*T*cos(theta)) + Vx0;
    Vy1 = (f*sin(theta)-g)*T + Vy0;
    x1 = .5*f*T^2*cos(theta);
    y1 = .5*(f*sin(theta)-g)*T^2 + y_init;

    u = sqrt(Vx1^2+Vy1^2);

    x_burn = 0:.1:x1;
    slope = (y1-y_init) / x1;
    y_burn = slope * x_burn + y_init;

    time = 0:.1:final_time;
    x_coast = x1 + Vx1*time ;
```

```
    y_coast = y1 + Vy1.*time - .5 * g * time.^2;

    k = find(y_coast >-.01,1, 'last');

    x = [x_burn x_coast(1:k)] / 1000;
    y = [y_burn y_coast(1:k)] / 1000;

    figure(13)
    plot(x,y); grid on; hold on;
    xlabel('Downrange Position, x (km)');
    ylabel('Altitude, y (km)');
    title('Missile Range Optimized Trajectory - Dante Sanaei');
    hold on; plot(x1/1000, y1/1000, 'r*')
    legend('Ground Launch', 'Burnout', 'Air Launch')
    max_range_formula = f*T^2 * ( f/g * cot(theta) - .5 * cos(theta))
    max_range_real = x_coast(k)
    max(y)

end

Warning: Ignoring extra legend entries.

max_range_formula =

   1.1324e+06


max_range_real =

   1.1333e+06


ans =

  264.8191


max_range_formula =

   1.1324e+06


max_range_real =

   1.6017e+06


ans =

  383.1919
```



Functions

```
function dX_dtau = ascent_odes_tf(tau,X,tf)
    % X(1) = xbar, horizontal component of position
    % X(2) = ybar, vertical component of position
    % X(3) = Vxbar, horizontal component of velocity
    % X(4) = Vybar, vertical component of velocity
    % X(5) = lambda_2_bar, first costate
    % X(6) = lambda_3_bar, second costate
    % X(7) = lambda_4_bar, third costate
    global g Vc h K1 scale F m0 mdot m

    % Adjust mass
    m = m0-abs(mdot)*tau*tf;
    % Simplify long equations
    lam_mag = sqrt(X(6)^2+X(7)^2);
    V_mag = sqrt(X(3)^2+X(4)^2);
    %State and Co-state DE's in terms of d/dt:
    xbardot = X(3)*Vc/h;
    ybardot = X(4)*Vc/h;
    Vxbardot = (F/(m*Vc)) * (-X(6)/lam_mag) - K1*exp(-X(2)*scale) *
 X(3) * V_mag;
    Vybardot = (F/(m*Vc)) * (-X(7)/lam_mag) - K1*exp(-X(2)*scale) *
 X(4) * V_mag - (g/Vc);
```

```
    if sqrt(X(3)^2+X(4)^2) == 0
        lambda_2_bar = 0;
        lambda_3_bar = 0;
        lambda_4_bar = -X(5)*Vc/h;
    else
        lambda_2_bar = (X(6)*X(3) + X(7)*X(4)) * exp(-X(2)*scale) * (-
K1 * scale * V_mag);
        lambda_3_bar = K1 * exp(-X(2) * scale) * (X(6) * (V_mag +
 (X(3)^2 / V_mag)) + X(7) * (X(3)*X(4)/V_mag));
        lambda_4_bar = (-X(5) * (Vc/h)) + K1 * exp(-X(2) * scale) *
 (X(7) * (V_mag + (X(4)^2 / V_mag)) + X(6) * (X(3)*X(4)/V_mag));
    end

    dX_dtau = tf*[xbardot; ybardot; Vxbardot; Vybardot;lambda_2_bar;
 lambda_3_bar; lambda_4_bar];
    return
end

function PSI = ascent_bcs_tf(Y0,Yf,tf)
    global xbar0 ybar0 Vxbar0 Vybar0 ybarf Vxbarf Vybarf Vc F K1 scale
 g m0 mdot
    Hf = -1;
    mf = m0-abs(mdot)*tf;

    PSI = [Y0(1) - xbar0; % Initial Condition
           Y0(2) - ybar0; % Initial Condition
           Y0(3) - Vxbar0; % Initial Condition
           Y0(4) - Vybar0; % Initial Condition
           Yf(2) - ybarf; % Final Condition
           Yf(3) - Vxbarf; % Final Condition
           Yf(4) - Vybarf; % Final Condition
           (-F/(mf*Vc))*sqrt(Yf(6)^2 + Yf(7)^2) - K1*exp(-
scale)*Vc*(Yf(6)*Yf(3))*sqrt(Yf(3)^2) - (Yf(7)*g/Vc) - Hf];
    return
end
```

tf_mass =

142.2460

Published with MATLAB® R2017b