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 drag 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;
drag = 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;
drag = rho_ref*CD*A*Vc/(2*m0);

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

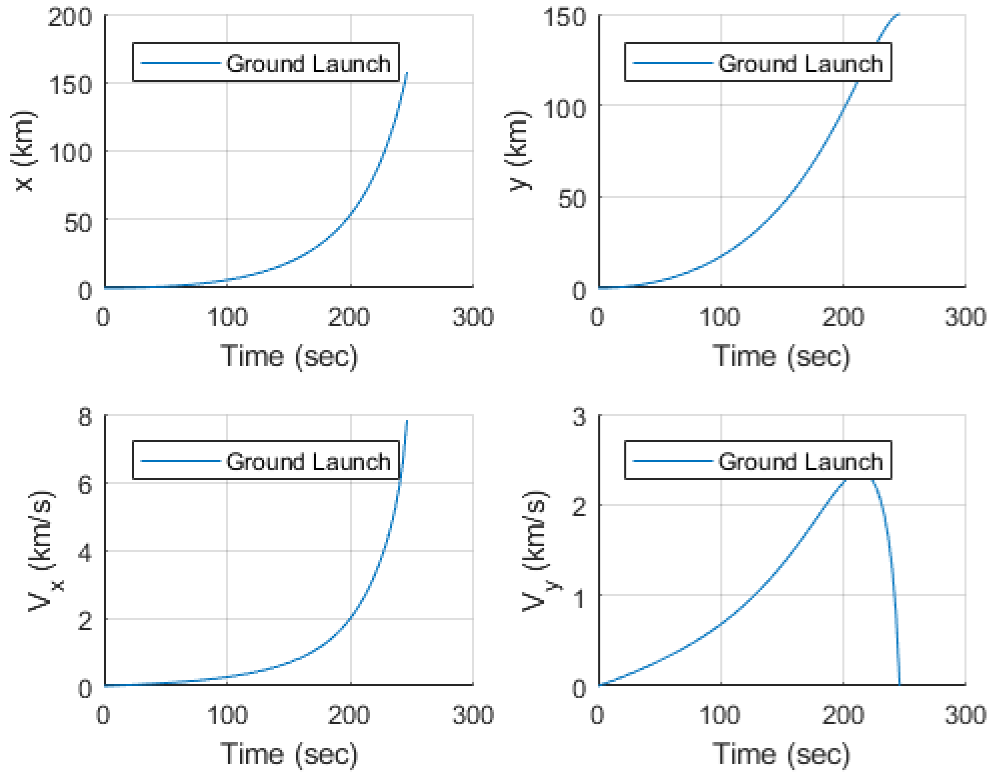
```
solinit_mass.parameters(1) = tf-delta_tf;

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;
drag = 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,:);
```

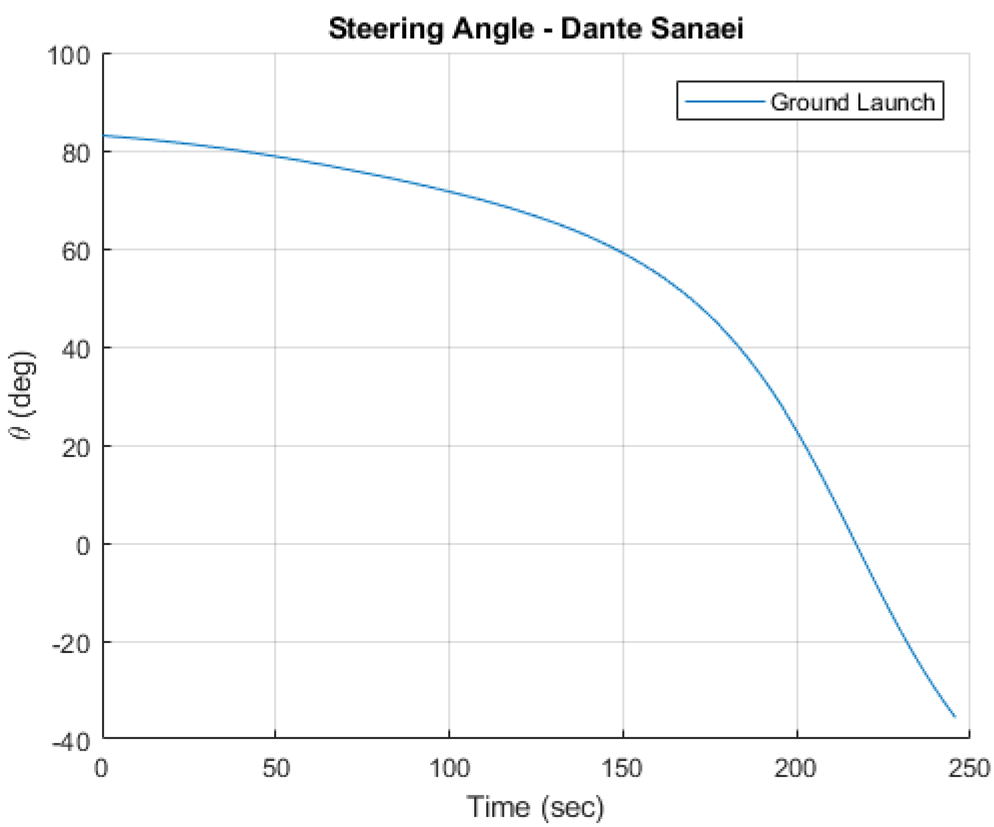
Plots

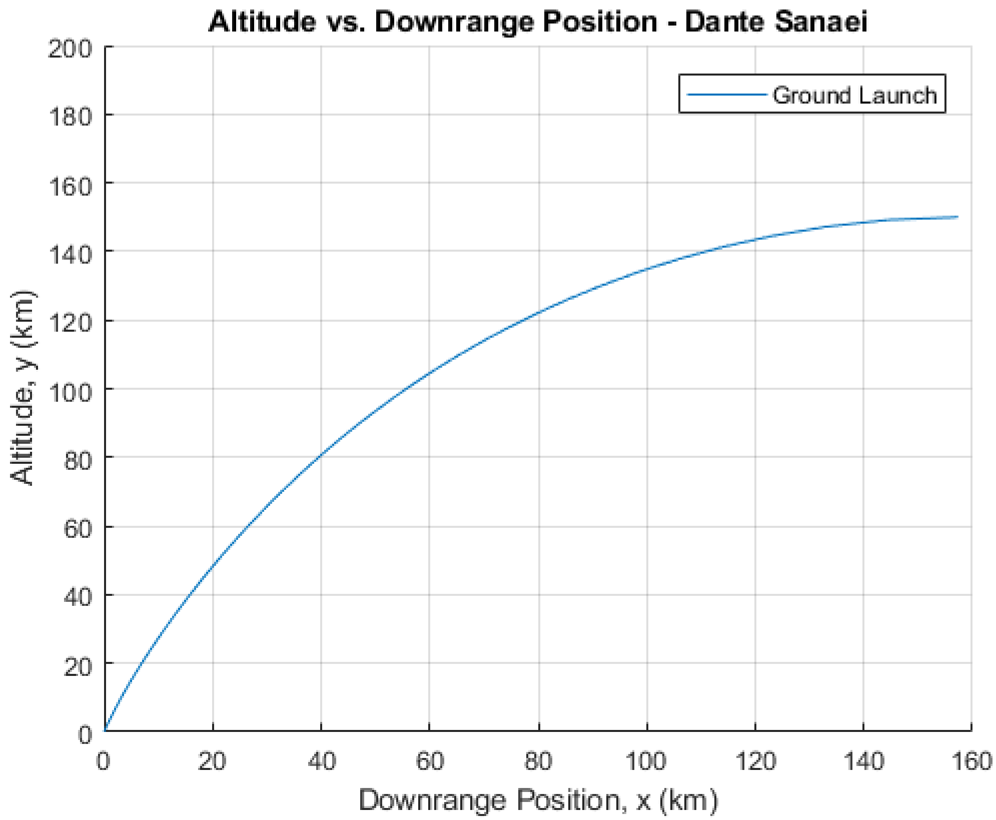
```
figure(1)
subplot(2,2,1); hold on
plot(time_mass_drag,x_sol_mass_drag); grid on
```

```
    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')
```

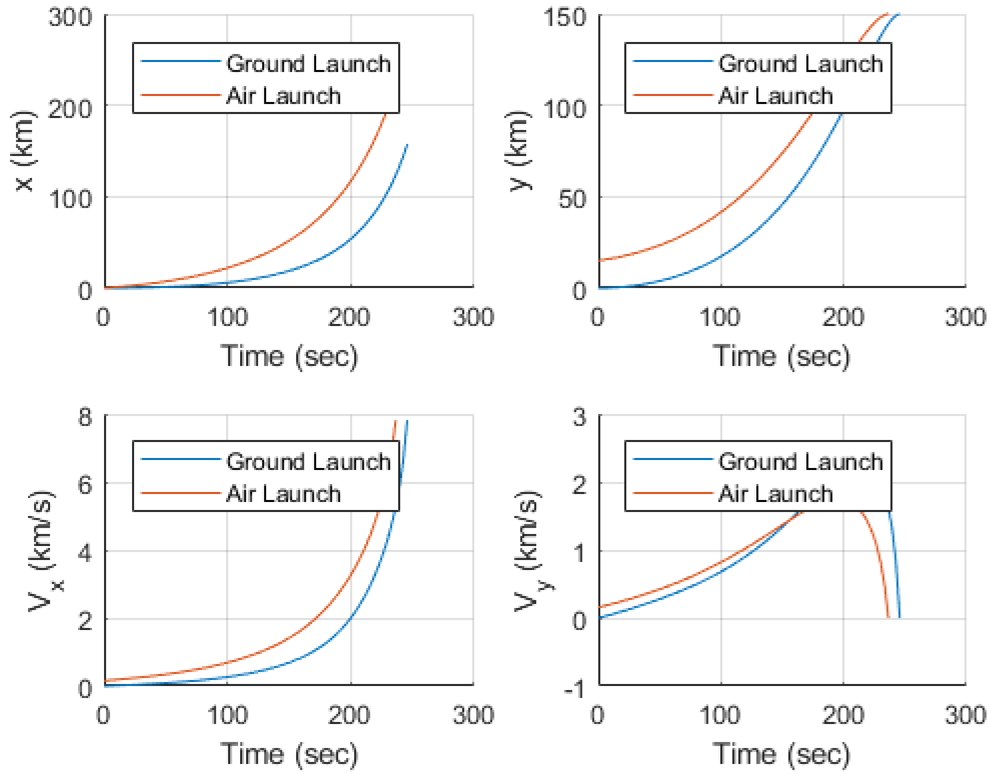
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.

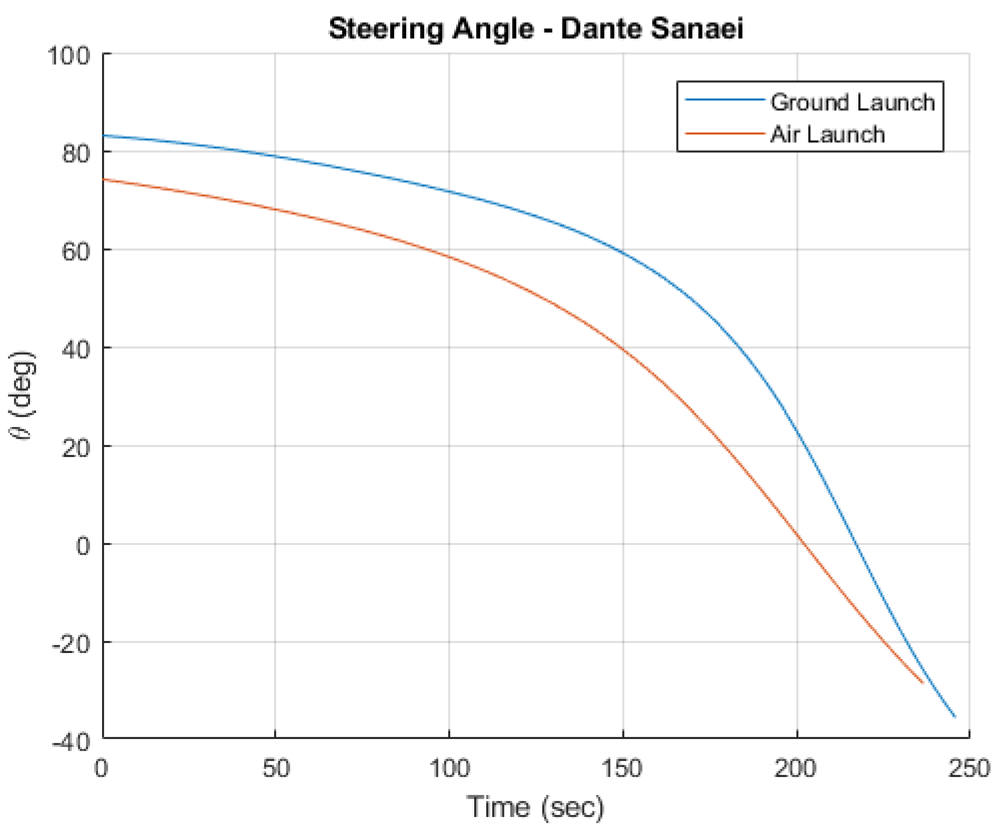

Ground Launch
x (km)
y (km)
V_x (km/s)
V_y (km/s)
Time (sec)

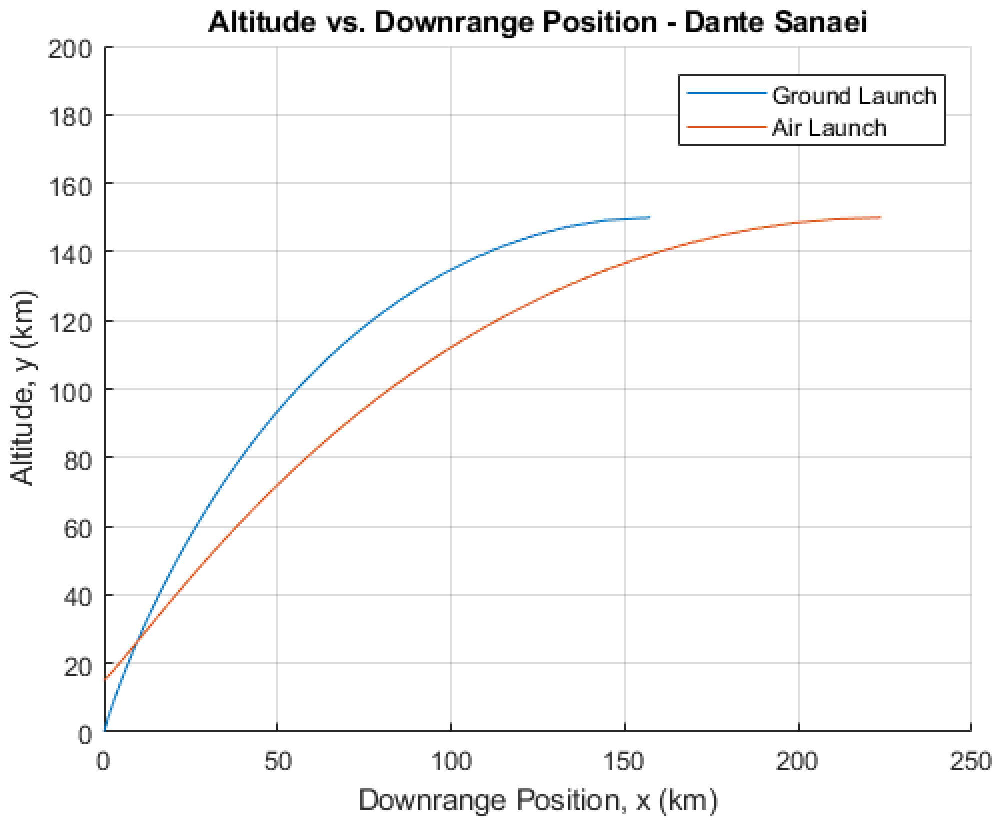

Steering Angle - Dante Sanaei
Ground Launch
θ (deg)
Time (sec)

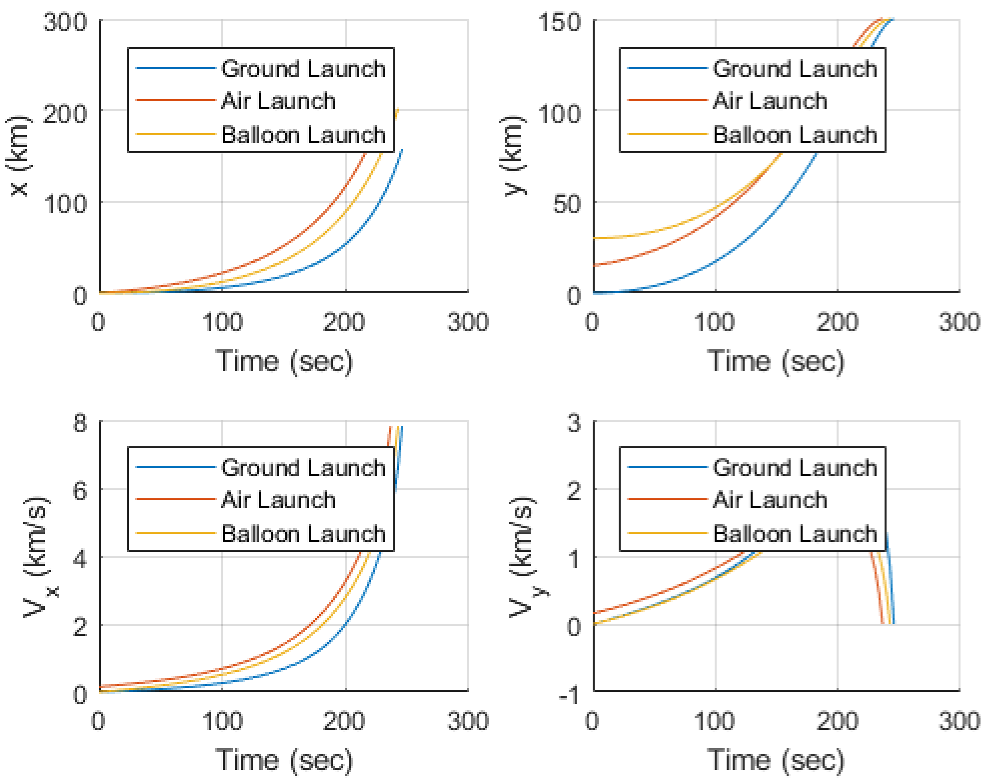


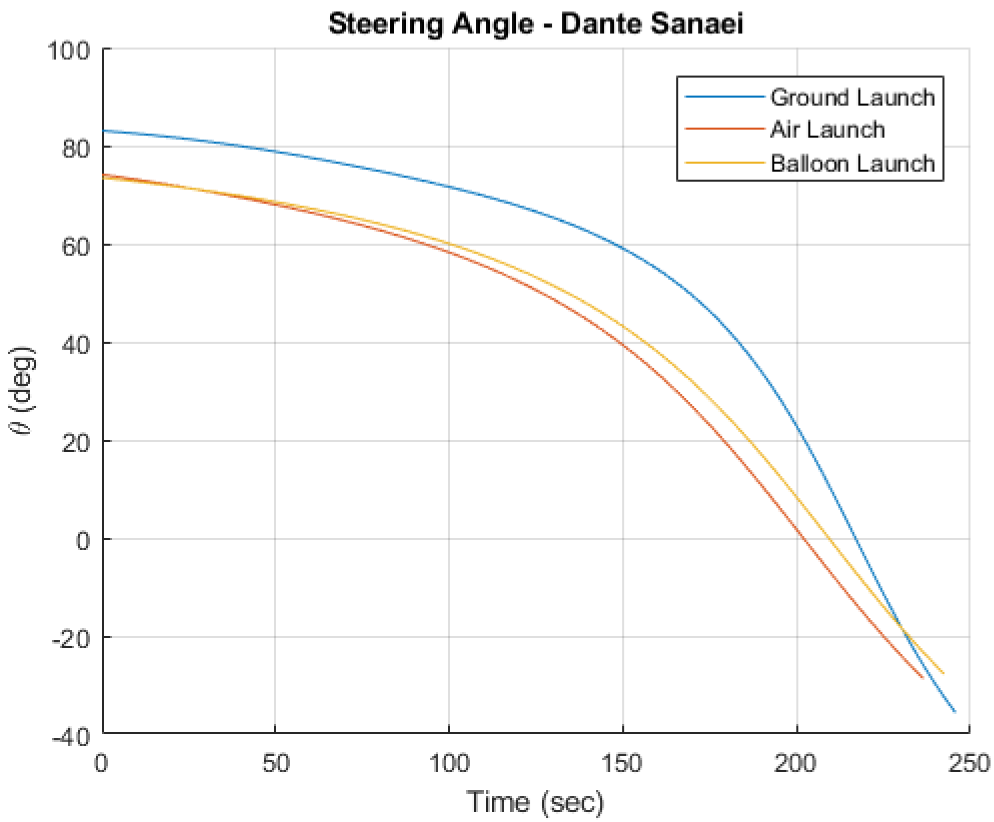

```
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
Warning: Ignoring extra legend entries.
```

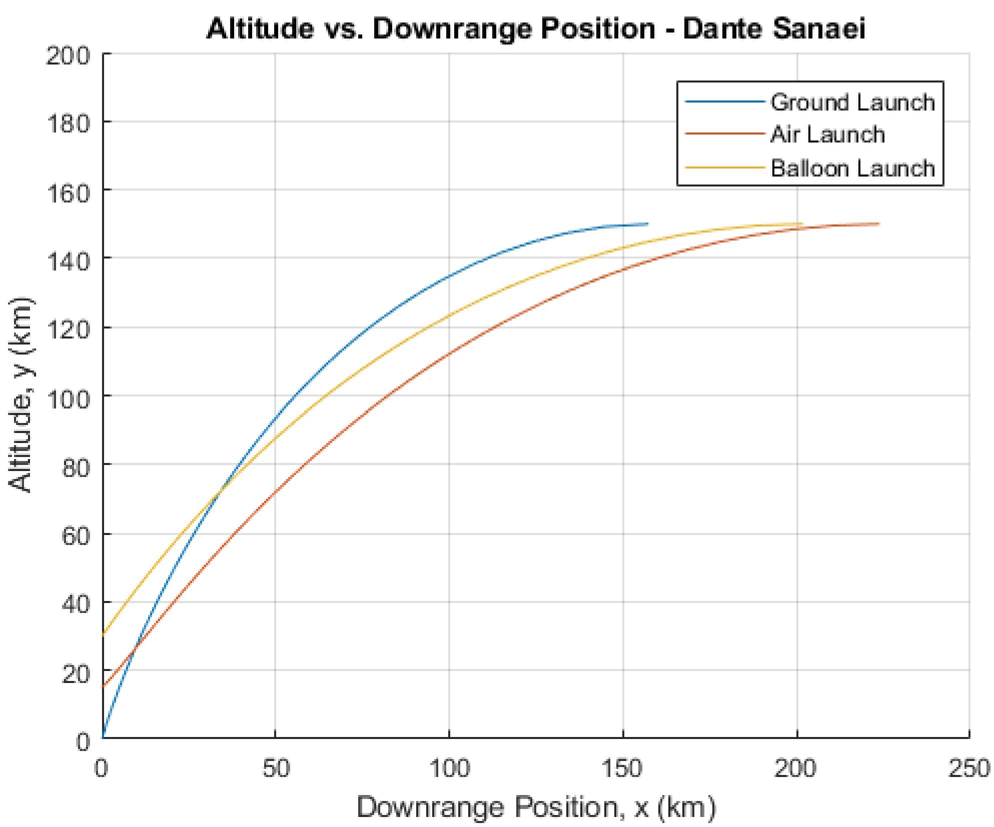

300
200
100
0
x (km)
0
100
200
300
Time (sec)
Ground Launch
Air Launch
150
100
50
0
y (km)
Time (sec)
8
6
4
2
V x (km/s)
3
2
1
0
-1
V y (km/s)


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


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


Steering Angle - Dante Sanaei
Ground Launch
Air Launch
Balloon 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
Balloon 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

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;
        drag = 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 =
          142.246028205757
Alt_diff =
       0.00372030672005718
SA_diff =
       0.00686545547300033
Vx_diff =
      0.000262135418906546
Vy_diff =
      0.000117484259788918
```

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 =
          157.342984774386
final_y =
   150
final_vx =
          7.81401349817147
final_vy =
      4.24839527187442e-19
final_time =
           245.65838037571

final_x =
          223.804089996705
final_y =
   150
final_vx =
          7.81401349817147
final_vy =
     -6.20505621464979e-22
final_time =
          236.415683460426

final_x =
          201.815077714251
final_y =
   150
final_vx =
          7.81401349817147
final_vy =
      4.13670416879971e-22
final_time =
          242.452565104709

end
```

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 drag scale F m0 mdot m variable_thrust


    % Integrate thrust difference due to nozzle efficiency
    if mdot ~= 0 & variable_thrust == 1
        Ve = 2598.43;
        Pe = 55.731;
        Me = 3.667;
        Ae = 27.41;
        gamma = 1.4;
        P = .101 * exp(-X(2)* scale);
        Pfs = P * (1+(gamma-1)/2 * Me^2 ) ^ (-gamma/gamma-1);
        F = mdot * Ve + (Pe - Pfs) * Ae;
    end
    % 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) - drag*exp(-X(2)*scale) *
 X(3) * V_mag;
    Vybardot = (F/(m*Vc)) * (-X(7)/lam_mag) - drag*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) * (-
drag * scale * V_mag);
        lambda_3_bar = drag * 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)) + drag * 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 drag
 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) - drag*exp(-
scale)*Vc*(Yf(6)*Yf(3))*sqrt(Yf(3)^2) - (Yf(7)*g/Vc) - Hf];
    return
end
```

Published with MATLAB® R2017b