3801 Lab 5 – Fixed Wing Dynamics

December 8th, 2023

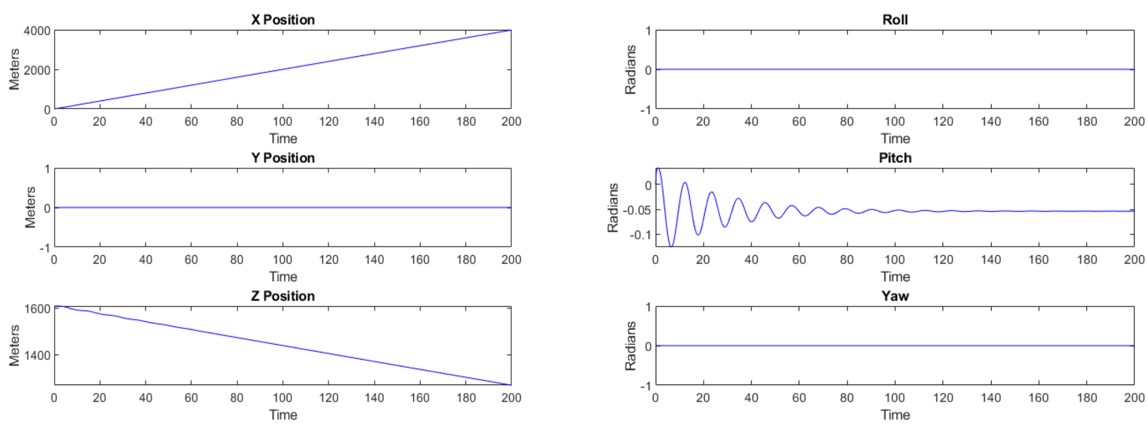
Kirin Kawamoto
Jared Steffen
Mark Turner
Andrew Vo

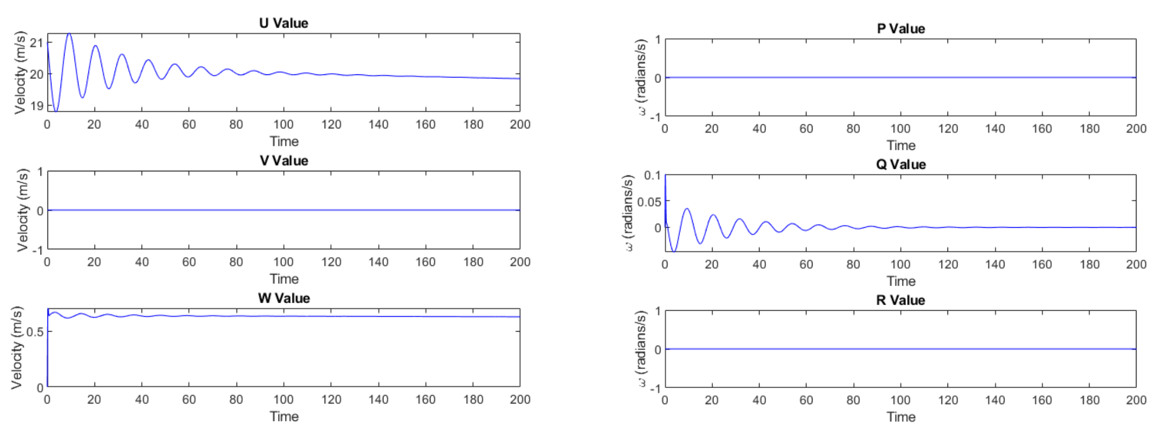
Lab Task 1: PlotSimulation

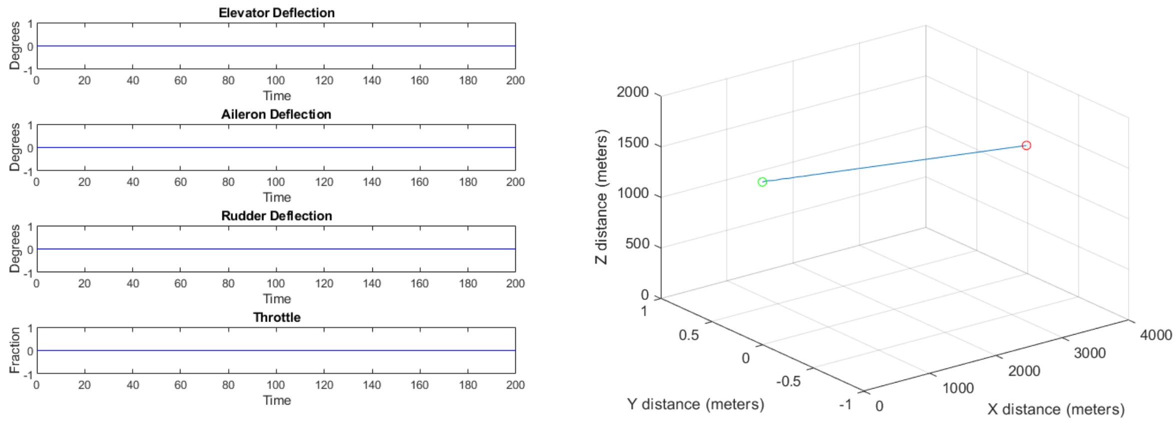
See Appendix for PlotSimulation function

Lab Task 2: Aircraft Equations of Motions

2.1



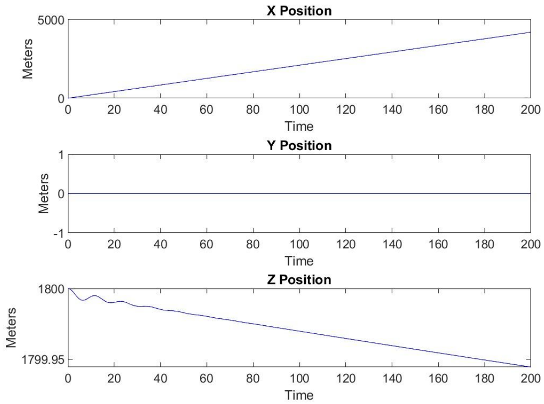


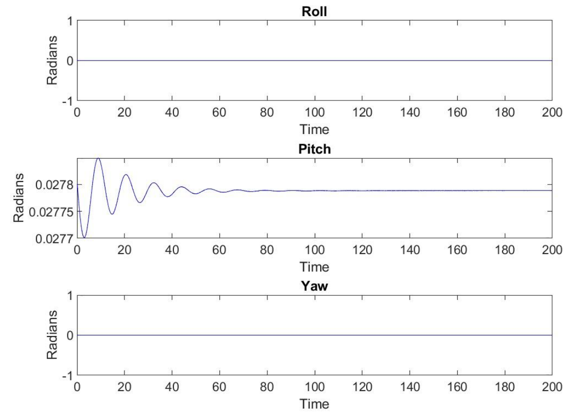


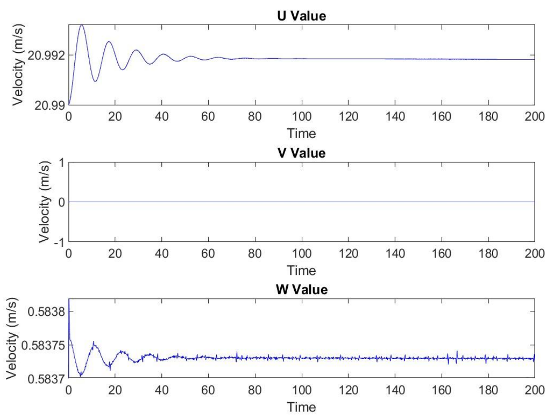
The initial conditions for the above plots are an altitude of 1609.34 meters, initial x- and y- positions of 0 meters; body x-velocity of 21 meters per second, body x- and y-velocities of 0 m/s; and roll, pitch, and yaw rates of zero radians per second. The control inputs are all zero for the elevator, aileron, and rudder deflections and the throttle input. The controls, as well as the roll and yaw angles and rates and body y-direction position and velocity remain at their initial values of zero during the simulation. Because the throttle input is zero and the angle of attack is zero, the aircraft is gliding rather than a steady, level trim condition. The body x-velocity starts at an initial value of 21 m/s and has decreasing oscillations that initially have a magnitude slightly larger than 1 m/s and decrease in magnitude over time to approach an oscillation magnitude of zero. The initial oscillations have a period of around 10 seconds. By 160 seconds the oscillation is minimal and can be approximated as zero, so the body x-velocity decreases linearly by approximately -0.001646 m/s.

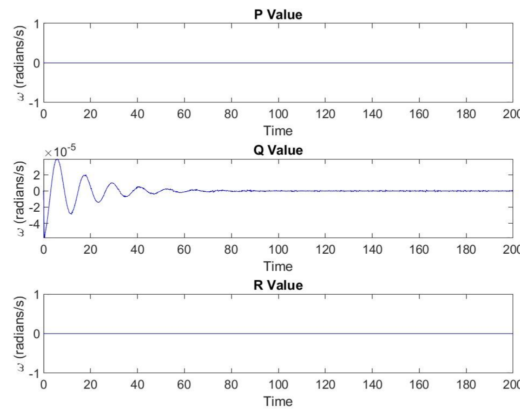
The cause for the initial oscillations in the body x-velocity (as well as the body z-velocity, pitch angle, and pitch rate) is that the aircraft is not at trim because the throttle input is zero, so aerodynamic drag slows down the aircraft, causing the lift force to decrease and the aircraft to descend. The decrease in height increases the velocity due to conservation of energy (exchanging potential energy stored in height for kinetic energy in the form of velocity and a small loss of energy due to drag), which increases the lift force and causes the aircraft to climb. As the aircraft climbs the velocity is converted back to height, which decreases the velocity and causes the cycle to repeat. The change in attitude of the aircraft can also be seen in the pitch angle and rate both oscillate. The magnitude of the oscillations decreases each cycle because the system loses energy due to drag. After the oscillations die out, the aircraft reaches a constant downward glide with a body z-velocity of 0.63 m/s and an inertial z-velocity of -1.698 m/s

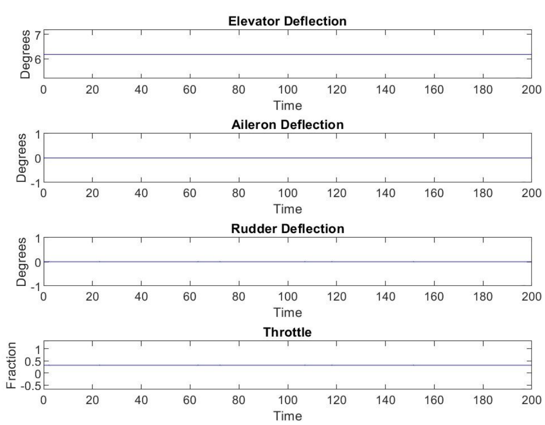
2.2

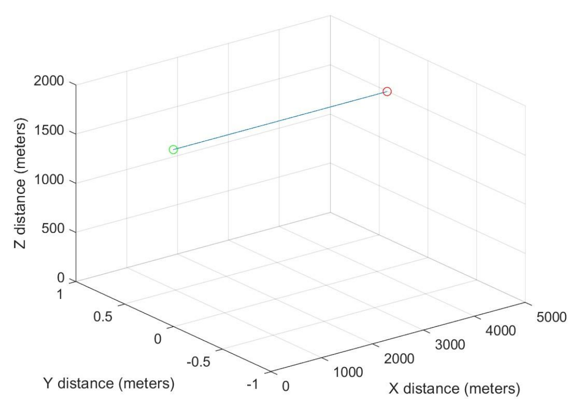








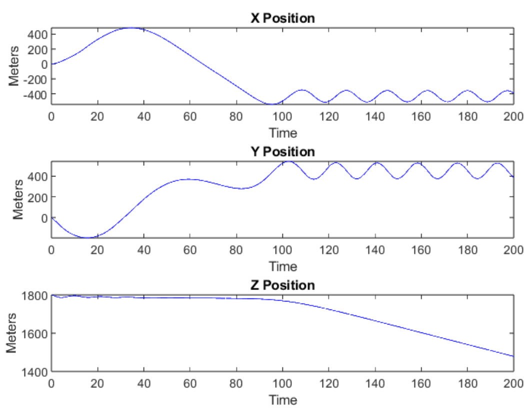


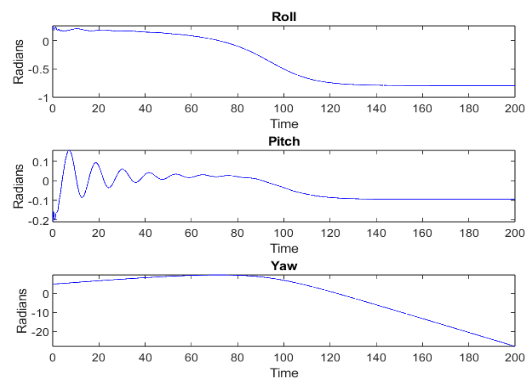


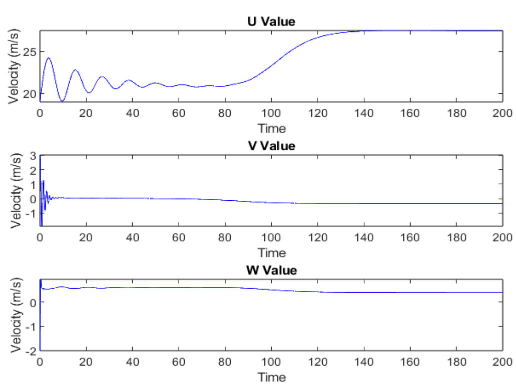
For problem 2.2, the aircraft is now in trim and is moving in the inertial x direction. With the given initial conditions, the aircraft experiences some very minor oscillations in the parameters: inertial z-direction pitch angle, u^E, w^E, and q that are on the order of 10^{-3} to 10^{-5}.

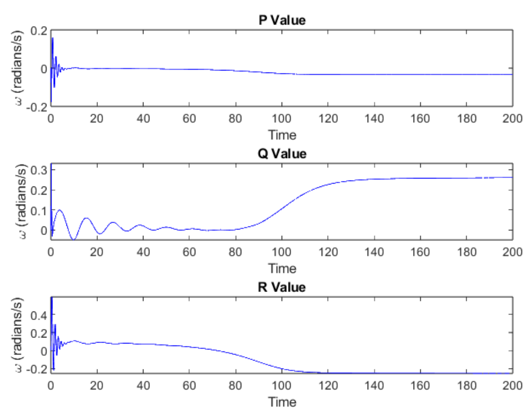
All of the oscillations settle at around 60 seconds and have a period of around 12 seconds. Due to the magnitude of the oscillations, they are most likely not very noticeable and do not impact the flight trim. The main indications that the aircraft is now in trim is that the velocities are constant (with the exception of the very minor oscillations for the first 60 seconds), that the roll and yaw angles are held steady at zero, and that the height is kept constant at 1800 meters. With only a pitch angle, no acceleration, and a steady height, we know that the aircraft is in trim. The differences between this section and section 2.1 is that the elevator and thrust values are non zero. The addition of thrust makes it so the aircraft is no longer gliding and the elevator controls gives the aircraft an angle of attack, which in turn helps the aircraft generate lift. With this increase in lift and addition of thrust, the drag and gravity force is now balanced out so the net force on the aircraft is approximately zero.

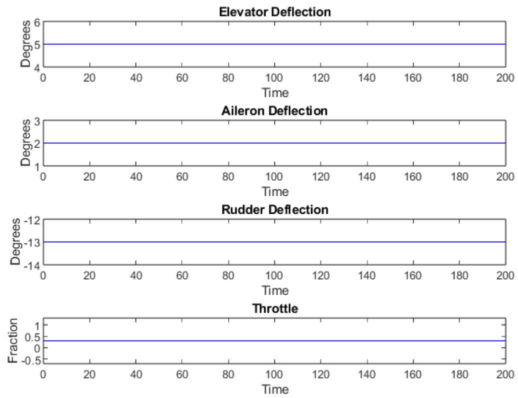
2.3

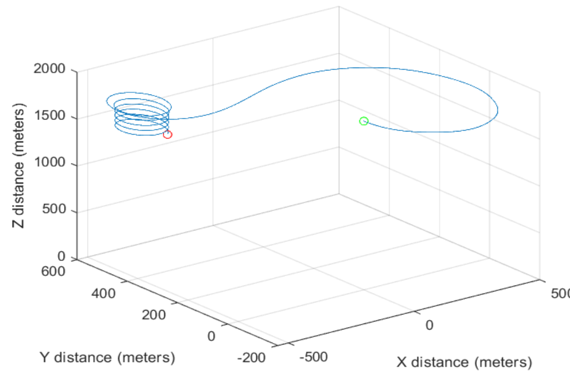












The simulation above appears to demonstrate the dynamics of the spiral mode of the aircraft, illustrated by the "spiraling" motion of the aircraft in the 3 dimensional path plot. The dynamics of most of the states can be characterized by 3 stages: the initial response, the transition stage, and the steady state. However, the deflection angles through the simulation stay constant.

The roll and yaw rates, along with the velocity in the body y direction, experience a fast oscillatory response from 0 to 5 seconds, each with a period of 1 second. The roll and pitch angle experience a fast oscillatory response, but smaller in magnitude and duration in comparison to their magnitudes in later stages. Their oscillations have magnitudes no more than 10 percent of the maximum amplitudes experienced by each state over the entire simulation. Velocity in the body z direction and pitch rate both experience a very large step response, settling to small oscillations in less than a second.

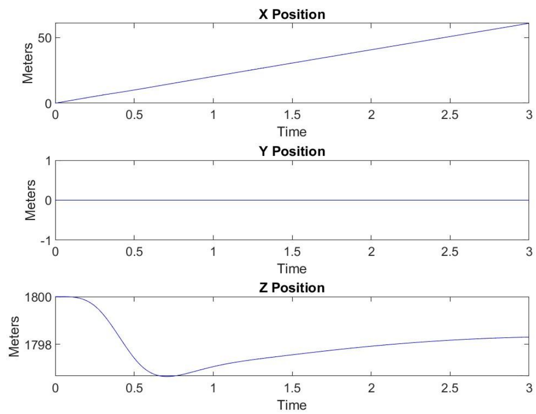
After the initial short period response, many states, like pitch, velocity in the body y direction, and pitch rate experience decreasing amplitude oscillations, with periods of around 12 seconds, settling to a temporary steady state around 60 seconds; this can be characterized as phugoid mode.. Other states, like roll angle, velocity in the body y direction, velocity in the body z direction, pitch rate, and yaw rate, settle to the temporary steady state immediately after the initial response. States like the inertial z position and yaw angle did not experience a significant initial response, settling to the temporary state from the beginning.

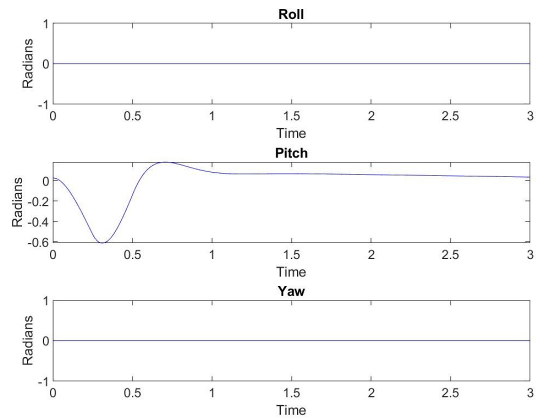
Around 80 to 100 seconds, the states transition from their temporary state to the final steady state, which is a constant value for some states, or a constant slope for others. Roll angle, pitch angle, all velocities, and all rotation rates settle to a constant value by 140 seconds, while inertial z position and yaw settle to maintain a constant negative slope, steadily decreasing.

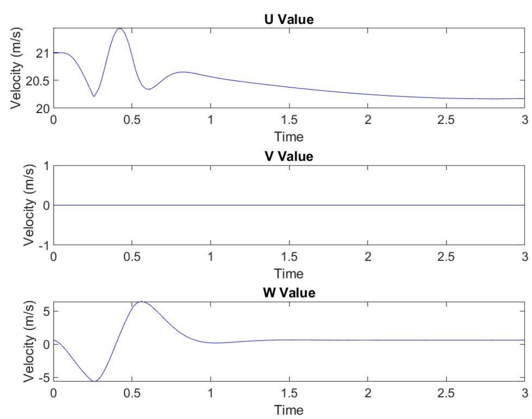
Two states that stray from the 3 stage response are the inertial x and y positions. From 0 seconds to 100 seconds, the states experience large amplitude (400 m), slow-frequency oscillations. The inertial x position has an oscillation with a period of 100 seconds, while the inertial y position has an oscillation with a period of 75 seconds. After 100 seconds, the x and y inertial positions settle into a steady state oscillation with amplitudes of 100 meters and periods of 20 seconds.

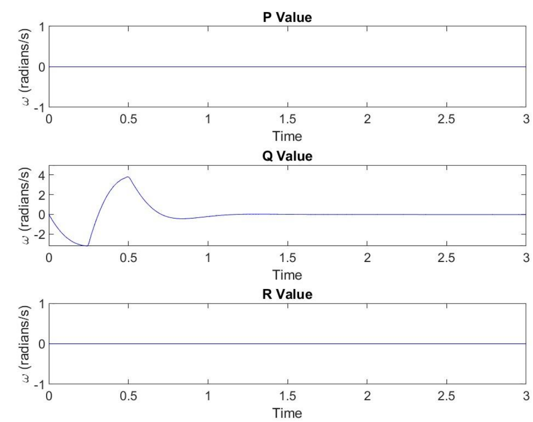
Lab Task 3: Aircraft Equations of Motions Doublet

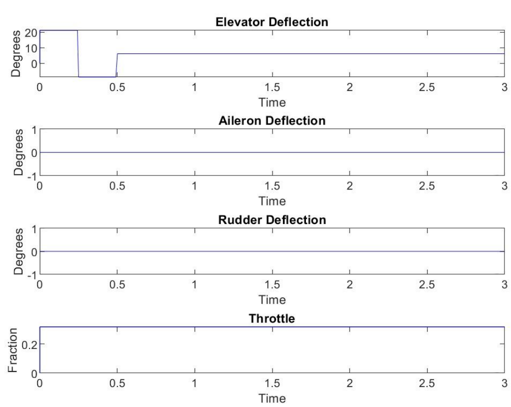
3.1

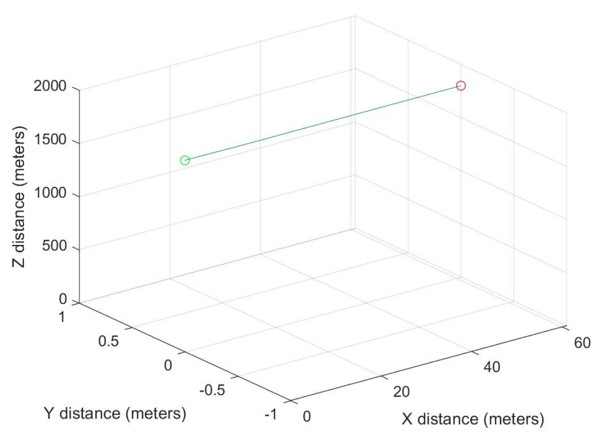










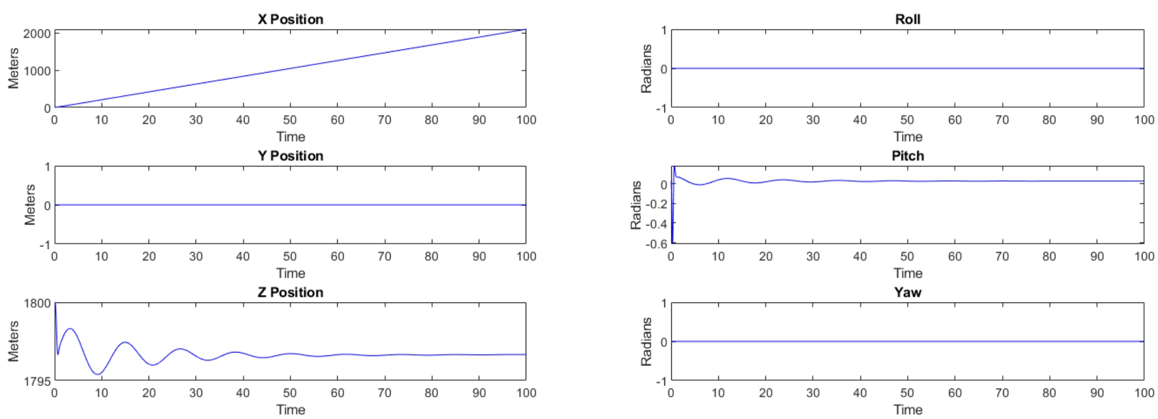


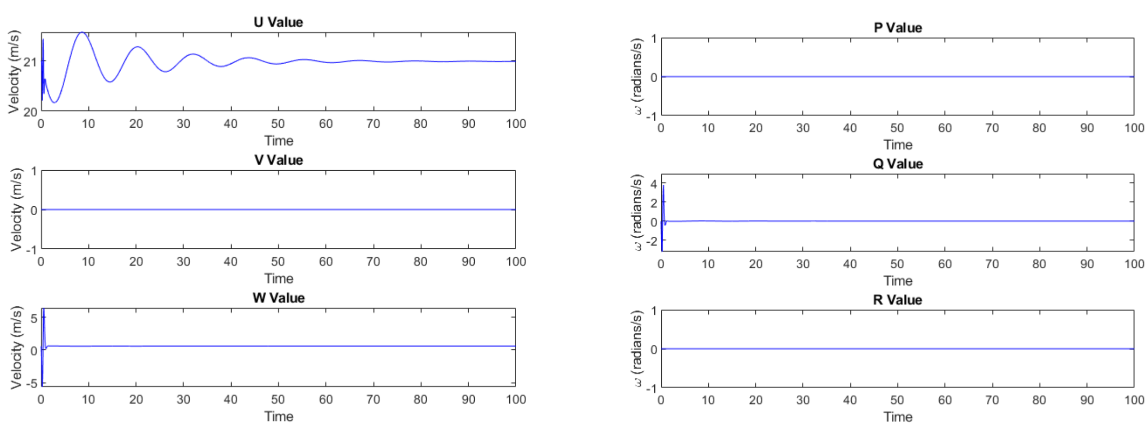
The damping ratio and natural frequency can be estimated graphically from the following equations:

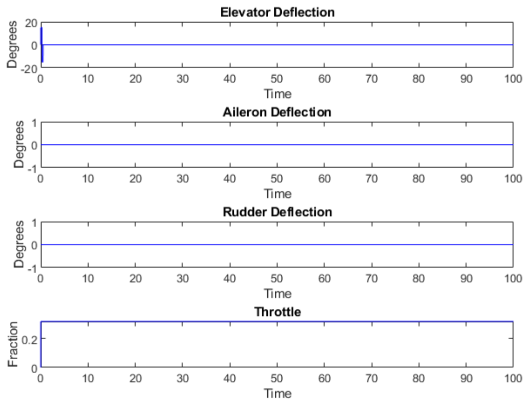
$$\sigma = ln(\frac{y(t)}{y(t+nT)}) \; ; \; \zeta = \frac{1}{\sqrt{1+(\frac{2\pi}{\sigma})^2}} \; ; \omega_d = \frac{1}{T} \; ; \; \omega_n = \frac{\omega_d}{\sqrt{1-\zeta^2}}$$

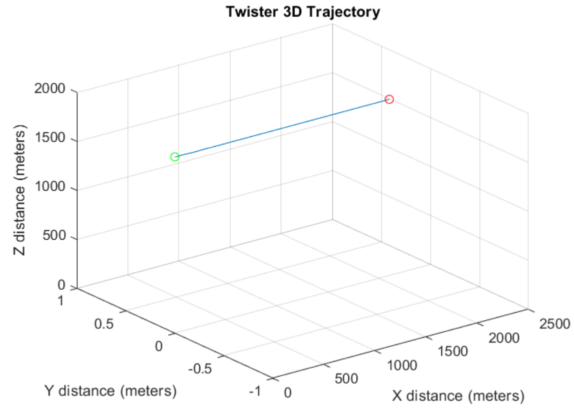
Specifically, the data was pulled from the u^E graph. For short period mode, the first peak is at (0.4315, 21.4359) and the second peak was estimated to be at (0.8222, 20.6459). Using these values, a damping ratio of 0.005948 and a natural frequency of 2.5595 rad/s was determined. While the damping ratio seems a little low, it does end up being double the value of the damping ratio of the phugoid mode (more on that in section 3.2). This however could be explained by the fact that the Ttwistor aircraft is not a full sized aircraft, so it wouldn't need as high of a damping ratio as a normal aircraft to have the oscillations damped out. The short period oscillations have a period of about half a second and settle out within 1.5 seconds.

3.2









Using the same method outlined in section 3.1 for short period, the first peak for phugoid mode was estimated to be at (20.1528, 21.568) and the second peak was estimated at (8.568, 21.2768). These values give a damping ratio of 0.002163 and a natural frequency of 0.08632 rad/s. These values for damping ratio and natural frequency make sense when compared to those of short period. We would expect the damping ratio and natural frequencies of phugoid mode to be lower than those of short period as the oscillations continue on for a much longer duration and have higher magnitudes. This is the case for our graphically estimated values. The phugoid mode oscillations have a period of about 11.5 seconds and settle out around 80 seconds.

Team Participation Table

Name	Plan	Model	Experiment	Results	Report	Code	ACK
Kirin Kawamoto	1	1	1	1	2	1	X
Jared Steffen	1	1	1	2	1	1	X
Mark Turner	2	1	1	1	1	1	X
Andrew Vo	1	1	1	1	1	2	X

Appendix – MATLAB Functions

main.m

```
clc; clear; close all
% Calls another script that sets all the aircraft parameters
ttwistor;
% Setting the inertial wind velocity vector
wind_inertial = [0;0;0];
% Setting the time span for the simulations
tfinal = 200;
TSPAN = [0 tfinal];
%% Problem 2_1
% Initial aicraft state and control values for 2.1
aircraft_state_2_1 = [0;0;-1609.34;0;0;0;21;0;0;0;0;0];
control_input_2_1 = [0;0;0;0];
% Runs simulation from initial state
[TOUT2_1,YOUT2_1] = ode45(@(t,y) AircraftEOM(...
   t,y,control_input_2_1,wind_inertial,aircraft_parameters),...
                                           TSPAN,aircraft_state_2_1,[]);
% Makes control input array for plotting
UOUT2_1 = zeros(4,length(TOUT2_1));
for i=1:length(TOUT2_1)
   UOUT2_1(:,i) = control_input_2_1;
end
% Plottting simulation
PlotSimulation(TOUT2_1,YOUT2_1,UOUT2_1,'b')
%% Problem 2_2
% Initial aicraft state and control values for 2.2
aircraft_state_2_2 = [0;0;-1800;0;0.02780;0;20.99;0;0.5837;0;0;0];
control_input_2_2 = [0.1079;0;0;0.3182];
% Runs simulation from initial state
[TOUT2_2,YOUT2_2] = ode45(@(t,y) AircraftEOM(...
   t,y,control_input_2_2,wind_inertial,aircraft_parameters),...
                                           TSPAN,aircraft_state_2_2,[]);
% Makes control input array for plotting
UOUT2_2 = zeros(4,length(TOUT2_2));
for i=1:length(TOUT2_2)
   UOUT2_2(:,i) = control_input_2_2;
end
% Plottting simulation
PlotSimulation(TOUT2_2,YOUT2_2,UOUT2_2,'b')
%% Problem 2_3
% Initial aicraft state and control values for 2.2
aircraft_state_2_3 = [0;0;-1800;...
                     deg2rad(15);deg2rad(-12);deg2rad(270);...
                     19;3;-2;...
```

```
                     deg2rad(0.08);deg2rad(-0.2);0];
control_input_2_3 = [deg2rad(5);deg2rad(2);deg2rad(-13);0.3];
% Runs simulation from initial state
[TOUT2_3,YOUT2_3] = ode45(@(t,y) AircraftEOM(...
   t,y,control_input_2_3,wind_inertial,aircraft_parameters),...
                                       TSPAN,aircraft_state_2_3,[]);
% Makes control input array for plotting
UOUT2_3 = zeros(4,length(TOUT2_3));
for i=1:length(TOUT2_3)
   UOUT2_3(:,i) = control_input_2_3;
end
% Plottting simulation
PlotSimulation(TOUT2_3,YOUT2_3,UOUT2_3,'b')
%% Problem 3_1
% Initial aicraft state and control values for 3.1
aircraft_state_3_1 = [0;0;-1800;0;0.02780;0;20.99;0;0.5837;0;0;0];
control_input_3_1 = [0.1079;0;0;0.3182];
% Doublet size and duration
d_size_3_1 = 15; % degrees
d_time_3_1 = 0.25; % seconds
% Runs simulation from initial state using doublet EOM for 3 seconds
[TOUT3_1,YOUT3_1] = ...
   ode45(@(t,y) AircraftEOMDoublet(t,y,control_input_3_1,d_size_3_1,...
       d_time_3_1,wind_inertial,aircraft_parameters),[0 3],...
                                                 aircraft_state_3_1,[]);
% Creating control input array for plotting controls over time
UOUT3_1 = zeros(4,length(TOUT3_1));
for i=1:length(TOUT3_1)
   % If within 1 doublet time from beginning
   if TOUT3_1(i) > 0 && TOUT3_1(i) <= d_time_3_1
       % Subtract doublet deflection from initial  elevator deflection
       UOUT3_1(1,i) = control_input_3_1(1) + deg2rad(d_size_3_1);
   % After after 1 doublet time but before 2 doublet times
   elseif TOUT3_1(i) > d_time_3_1 && TOUT3_1(i) <= 2*d_time_3_1
       % Subtract doublet deflection from initial  elevator deflection
       UOUT3_1(1,i) = control_input_3_1(1) - deg2rad(d_size_3_1);
   % After doublet
   elseif TOUT3_1(i) > 2*d_time_3_1
       % Set controls to initial state after doublet
       UOUT3_1(1,i) = control_input_3_1(1);
   end
   % Keep controls other than elevator deflection the same
   UOUT3_1(2:4,i) = control_input_3_1(2:4);
end
% Plottting simulation
PlotSimulation(TOUT3_1,YOUT3_1,UOUT3_1,'b')
%% Problem 3_2
% Runs simulation from initial state using doublet EOM for 100 seconds
```

```
[TOUT3_2,YOUT3_2] = ...
   ode45(@(t,y) AircraftEOMDoublet(t,y,control_input_3_1,d_size_3_1,...
       d_time_3_1,wind_inertial,aircraft_parameters),[0 100],...
                                                   aircraft_state_3_1,[]);
% Creating control input array for plotting controls over time
UOUT3_2 = zeros(4,length(TOUT3_2));
for i=1:length(TOUT3_2)
   % If within 1 doublet time from beginning
   if TOUT3_2(i) > 0 && TOUT3_2(i) <= d_time_3_1
       % Subtract doublet deflection from initial  elevator deflection
       UOUT3_2(1,i) = control_input_3_1(1) + deg2rad(d_size_3_1);
   % After after 1 doublet time but before 2 doublet times
   elseif TOUT3_2(i) > d_time_3_1 && TOUT3_2(i) <= 2*d_time_3_1
       % Subtract doublet deflection from initial  elevator deflection
       UOUT3_2(1,i) = control_input_3_1(1) - deg2rad(d_size_3_1);
   % After doublet
   elseif TOUT3_2(i) > 2*d_time_3_1
       % Set controls to initial state after doublet
       UOUT3_2(1,i) = control_input_3_1(1);
   end
   % Keep controls other than elevator deflection the same
   UOUT3_2(2:4,i) = control_input_3_1(2:4);
end
% Plottting simulation
PlotSimulation(TOUT3_2,YOUT3_2,UOUT3_2,'b')
```

PlotSimulation.m

```
function PlotSimulation(time, aircraft_state_array, control_input_array, col)
% PlotSimulation: Plots the states of an aircraft over time from a
%                 simulation
% Inputs:             time = n X 1 vector of time values from simulation
%     aircraft_state_array = n x 12 matrix of state values; each column
%                            represents a state over time
%      control_input_array = 4 x n matrix of control inputs; each row
%                            represents a control input over time
%                      col = character array for plotting line-style
% Convert control surface deflections from radians to degrees
control_input_array = [rad2deg(control_input_array(1,:)); % Elevator
                      rad2deg(control_input_array(2,:)); % Aileron
                      rad2deg(control_input_array(3,:)); % Rudder
                      control_input_array(4,:)];         % Throttle
%%%%%%%%%% Plotting Inertial positions %%%%%%%%%%
figure;
subplot(311);
plot(time, aircraft_state_array(:,1), col); hold on;
title('X Position')
xlabel('Time')
```

```
ylabel('Meters')
subplot(312);
plot(time, aircraft_state_array(:,2), col); hold on;
title('Y Position')
xlabel('Time')
ylabel('Meters')
subplot(313);
plot(time, -aircraft_state_array(:,3), col); hold on;
title('Z Position')
xlabel('Time')
ylabel('Meters')
%%%%%%%%%% Plotting Inertial Attitude Angles %%%%%%%%%%
figure;
subplot(311);
plot(time, aircraft_state_array(:,4), col); hold on;
title('Roll')
xlabel('Time')
ylabel('Radians')
subplot(312);
plot(time, aircraft_state_array(:,5), col); hold on;
title('Pitch')
xlabel('Time')
ylabel('Radians')
subplot(313);
plot(time, aircraft_state_array(:,6), col); hold on;
title('Yaw')
xlabel('Time')
ylabel('Radians')
%%%%%%%%%% Plotting Inertial Velocities in Body Coordinates %%%%%%%%%%
figure;
subplot(311);
plot(time, aircraft_state_array(:,7), col); hold on;
title('U Value')
xlabel('Time')
ylabel('Velocity (m/s)')
subplot(312);
plot(time, aircraft_state_array(:,8), col); hold on;
title('V Value')
xlabel('Time')
ylabel('Velocity (m/s)')
subplot(313);
plot(time, aircraft_state_array(:,9), col); hold on;
title('W Value')
xlabel('Time')
ylabel('Velocity (m/s)')
%%%%%%%%%% Plotting angular velocities in body coordinates %%%%%%%%%%
figure;
subplot(311);
```

```matlab
plot(time, aircraft_state_array(:,10), col); hold on;
title('P Value')
xlabel('Time')
ylabel('\omega (radians/s)')
subplot(312);
plot(time, aircraft_state_array(:,11), col); hold on;
title('Q Value')
xlabel('Time')
ylabel('\omega (radians/s)')
subplot(313);
plot(time, aircraft_state_array(:,12), col); hold on;
title('R Value')
xlabel('Time')
ylabel('\omega (radians/s)')
%%%%%%%%%% Plotting Control Inputs %%%%%%%%%%
figure;
subplot(411);
plot(time,control_input_array(1,:), col); hold on;
title('Elevator Deflection')
xlabel('Time')
ylabel('Degrees')
subplot(412);
plot(time,control_input_array(2,:), col); hold on;
title('Aileron Deflection')
xlabel('Time')
ylabel('Degrees')
subplot(413);
plot(time,control_input_array(3,:), col); hold on;
title('Rudder Deflection')
xlabel('Time')
ylabel('Degrees')
subplot(414);
plot(time,control_input_array(4,:), col); hold on;
title('Throttle')
xlabel('Time')
ylabel('Fraction')
%%%%%%%%%% Plotting Inertial 3 dimensional path %%%%%%%%%%
figure;
hold on
grid on
plot3(aircraft_state_array(:,1),aircraft_state_array(:,2),...
                                            -aircraft_state_array(:,3))
plot3(aircraft_state_array(1,1),aircraft_state_array(1,2),...
                                       -aircraft_state_array(1,3),'og')
plot3(aircraft_state_array(end,1),aircraft_state_array(end,2),...
                                     -aircraft_state_array(end,3),'or')
zlim([0 2000])
view(3)
```

```
xlabel('X distance (meters)')
ylabel('Y distance (meters)')
zlabel('Z distance (meters)')
end
```

AircraftEOM.m

```
function xdot = ...
   AircraftEOM(time, aircraft_state, aircraft_surfaces, wind_inertial, ...
                                                        aircraft_parameters)
% AircraftEOM: calculates the derivative of the aircraft states; used by
%              ode45 to simulate response to an aircrafts initial
%              conditions
% Inputs:                 time = current time (seconds)
%        aircraft_state_array = 12 x 1 vector of current state values
%         control_input_array = 4 x 1 vector of control inputs
%               wind_inertial = 3 x 1 vector of inertial velocity
%                               components
%         aircraft parameters = structure of flight parameters for aircraft
% Output:                 xdot = 12 x 1 vector of state derivatives
% Extracting inertial positions in inertial coordinates
pos_inertial = aircraft_state(1:3,1);
% Extracting inertial attitude angles
euler_angles = aircraft_state(4:6,1);
% Extracting inertial velocities in body coordinates
vel_body = aircraft_state(7:9,1);
% Extracting angular velocities in body coordinates
omega_body = aircraft_state(10:12,1);
%%% Kinematics
% Calculating velocities in inertial coordinates using rotation
vel_inertial = TransformFromBodyToInertial(vel_body, euler_angles);
% Calculating angular velocities in inertial coordinates using rotation
euler_rates = EulerRatesFromOmegaBody(omega_body, euler_angles);
%%% Aerodynamic force and moment
% Getting air density using current height
density = stdatmo(-pos_inertial(3,1));
% Calculating aerodynamics forces and moments (including controls)
[fa_body, ma_body] = ...
   AeroForcesAndMoments(aircraft_state, aircraft_surfaces, ...
                           wind_inertial, density, aircraft_parameters);
%%% Gravity
% Calculating acceleration of gravity in body coordinates
fg_body = (aircraft_parameters.g)*...
   [-sin(euler_angles(2));...
     sin(euler_angles(1))*cos(euler_angles(2));...
     cos(euler_angles(2))*cos(euler_angles(1))];
%%% Dynamics
% Calculates inertial velocity derivatives in body coordinates
```

```
vel_body_dot = -cross(omega_body, vel_body) ...
                           + fg_body+(fa_body)/aircraft_parameters.m;
% Creating Inertia Matrix from aircraft parameters
inertia_matrix = [aircraft_parameters.Ix 0 -aircraft_parameters.Ixz;...
                   0 aircraft_parameters.Iy 0;...
                   -aircraft_parameters.Ixz 0 aircraft_parameters.Iz];
% Calculates inertial angular velocity derivatives in body coordinates
omega_body_dot = inv(inertia_matrix)...
           *(-cross(omega_body, inertia_matrix*omega_body) + ma_body);
%%% State derivative
xdot = [vel_inertial; euler_rates; vel_body_dot; omega_body_dot];
end
```

AircraftEOMDoublet.m

```
function xdot = ...
   AircraftEOMDoublet(time, aircraft_state, aircraft_surfaces, ...
       doublet_size, doublet_time, wind_inertial, aircraft_parameters)
% AircraftEOMDoublet: calculates the derivative of the aircraft states;
%                     used by ode45 to simulate response to an aircrafts
%                     initial conditions
% Inputs:                time = current time (seconds)
%        aircraft_state_array = 12 x 1 vector of current state values
%         control_input_array = 4 x 1 vector of control inputs
%                doublet_size = magnitude of doublet
%                doublet_time = duration of doublet
%               wind_inertial = 3 x 1 vector of inertial velocity
%                               components
%         aircraft parameters = structure of flight parameters for aircraft
% Output:                xdot = 12 x 1 vector of state derivatives
% Extracting states to their individual vectors:
% Inertial position in inertial coordinates
pos_inertial = aircraft_state(1:3,1);
% Inertial attitude in inertial coordinates
euler_angles = aircraft_state(4:6,1);
% Inertial velocity in body coordinates
vel_body = aircraft_state(7:9,1);
% Inertial angular velocity in body coordinates
omega_body = aircraft_state(10:12,1);
% During doublet time, change the elevator deflection by doublet size
if time > 0 && time <= doublet_time
   aircraft_surfaces(1) = aircraft_surfaces(1) + doublet_size;
elseif time > doublet_time && time <= 2*doublet_time
   aircraft_surfaces(1) = aircraft_surfaces(1) - doublet_size;
% After doublet time, keep initial elevator angle
elseif time > 2*doublet_time
   aircraft_surfaces(1) = aircraft_surfaces(1) ;
end
```

```
%%% Kinematics
% Calculating velocities in inertial coordinates using rotation
vel_inertial = TransformFromBodyToInertial(vel_body, euler_angles);
% Calculating angular velocities in inertial coordinates using rotation
euler_rates = EulerRatesFromOmegaBody(omega_body, euler_angles);
%%% Aerodynamic force and moment
% Getting air density using current height
density = stdatmo(-pos_inertial(3,1));
% Calculating aerodynamics forces and moments (including controls)
[fa_body, ma_body] = ...
   AeroForcesAndMoments(aircraft_state, aircraft_surfaces, ...
                         wind_inertial, density, aircraft_parameters);
%%% Gravity
% Calculating acceleration of gravity in body coordinates
fg_body = (aircraft_parameters.g)*...
   [-sin(euler_angles(2));...
     sin(euler_angles(1))*cos(euler_angles(2));...
     cos(euler_angles(2))*cos(euler_angles(1))];
%%% Dynamics
% Calculates inertial velocity derivatives in body coordinates
vel_body_dot = -cross(omega_body, vel_body) ...
                          + fg_body+(fa_body)/aircraft_parameters.m;
% Creating Inertia Matrix from aircraft parameters
inertia_matrix = [aircraft_parameters.Ix 0 -aircraft_parameters.Ixz;...
                   0 aircraft_parameters.Iy 0;...
                   -aircraft_parameters.Ixz 0 aircraft_parameters.Iz];
% Calculates inertial angular velocity derivatives in body coordinates
omega_body_dot = inv(inertia_matrix)...
           *(-cross(omega_body, inertia_matrix*omega_body) + ma_body);
%%% State derivative
xdot = [vel_inertial; euler_rates; vel_body_dot; omega_body_dot];
end
```

EulerRatesFromOmegaBody.m

```
function euler_rates = EulerRatesFromOmegaBody(omega_body, euler_angles)
% Transforms angular velocity in the body frame to inertial frame
% Transformation matrix from body to inertial for rotation rates
col1 = [1;...
       0;...
       0;];
col2 = [sin(euler_angles(1))*tan(euler_angles(2));...
       cos(euler_angles(1));...
       sin(euler_angles(1))*sec(euler_angles(2))];
col3 = [cos(euler_angles(1))*tan(euler_angles(2));...
       -sin(euler_angles(1));...
       cos(euler_angles(1))*sec(euler_angles(2))];
```

```
transform_matrix = [col1, col2, col3];
% Compute inertial rates by multiplying transformation matrix to rotations
% rates in body coordinates
euler_rates = transform_matrix * omega_body;
end
```

RotationMatrix321.m

```
function R321 = RotationMatrix321(attitude)
% Standard 321 Euler Angle Rotation
% If xI is inertial coordinates of vector
% then XB = R321 * XI are body coordinates
% cphi = cos(attitude(1));
% sphi = sin(attitude(1));
%
% cth = cos(attitude(2));
% sth = sin(attitude(2));
%
% cpsi = cos(attitude(3));
% spsi = sin(attitude(3));
%
%
% R3 = [cpsi spsi 0;...
%     -spsi cpsi 0;...
%     0 0 1];
%
% R2 = [cth 0 -sth;...
%     0 1 0;...
%     sth 0 cth];
%
% R1 = [1 0 0;...
%     0 cphi sphi;...
%     0 -sphi cphi];
%
% R321 = R1*R2*R3;
phi = attitude(1);
theta = attitude(2);
psi = attitude(3);
R3 = [cos(psi) sin(psi) 0;...
   -sin(psi) cos(psi) 0;...
   0 0 1];
R2 = [cos(theta) 0 -sin(theta);...
   0 1 0;...
   sin(theta) 0 cos(theta)];
R1 = [1 0 0;...
   0 cos(phi) sin(phi);...
   0 -sin(phi) cos(phi)];

R321 = R1*R2*R3;
```

TransformFromInertialToBody.m

```
function vector_body = ...
               TransformFromInertialToBody(vector_inertial, euler_angles)
% Rotates vector from inertial to body coordinates
vector_body = RotationMatrix321(euler_angles)*vector_inertial;
end
```

TransformFromBodyToInertial.m

```
function vector_inertial = ...
                   TransformFromBodyToInertial(vector_body, euler_angles)
% Rotates vector from body to inertial coordinates
vector_inertial = RotationMatrix321(euler_angles)'*vector_body;
end
```

WindAnglesFromVelocityBody.m

```
function [wind_angles] = WindAnglesFromVelocityBody(velocity_body)
V = norm(velocity_body);
alpha = atan2(velocity_body(3,1),velocity_body(1,1));
beta = asin(velocity_body(2,1)/V);
wind_angles = [V; beta; alpha];
```

AeroForcesAndMoments.m

```
function [aero_forces, aero_moments] = AeroForcesAndMoments(aircraft_state,
aircraft_surfaces, wind_inertial, density, aircraft_parameters)
%
%
% aircraft_state = [xi, yi, zi, roll, pitch, yaw, u, v, w, p, q, r]
%   NOTE: The function assumes the veolcity is the air relative velocity
%   vector. When used with simulink the wrapper function makes the
%   conversion.
%
% aircraft_surfaces = [de da dr dt];
%
% Lift and Drag are calculated in Wind Frame then rotated to body frame
% Thrust is given in Body Frame
% Sideforce calculated in Body Frame
%%% redefine states and inputs for ease of use
ap = aircraft_parameters;
wind_body = TransformFromInertialToBody(wind_inertial, aircraft_state(4:6,1));
air_rel_vel_body = aircraft_state(7:9,1) - wind_body;
[wind_angles] = WindAnglesFromVelocityBody(air_rel_vel_body);
V = wind_angles(1,1);
beta = wind_angles(2,1);
alpha = wind_angles(3,1);
p = aircraft_state(10,1);
q = aircraft_state(11,1);
r = aircraft_state(12,1);
de = aircraft_surfaces(1,1);
```

```
da = aircraft_surfaces(2,1);
dr = aircraft_surfaces(3,1);
dt = aircraft_surfaces(4,1);
alpha_dot = 0;
%Q = ap.qbar;
Q = 0.5*density*V*V;
sa = sin(alpha);
ca = cos(alpha);
%%% determine aero force coefficients
CL = ap.CL0 + ap.CLalpha*alpha + ap.CLq*q*ap.c/(2*V) + ap.CLde*de;
%CD = ap.CD0 + ap.CDalpha*alpha + ap.CDq*q*ap.c/(2*V) + ap.CDde*de;
CD = ap.CDpa + ap.K*CL*CL;
CX = -CD*ca + CL*sa;
CZ = -CD*sa - CL*ca;
CY = ap.CY0 + ap.CYbeta*beta + ap.CYp*p*ap.b/(2*V) + ap.CYr*r*ap.b/(2*V) +
ap.CYda*da + ap.CYdr*dr;
%%Thrust = .5*density*ap.Sprop*ap.Cprop*((ap.kmotor*dt)^2 - V^2);
Thrust = density*ap.Sprop*ap.Cprop*(V + dt*(ap.kmotor - V))*dt*(ap.kmotor-V);
%%Changed 5/2/15;New model as described in
http://uavbook.byu.edu/lib/exe/fetch.php?media=shared:propeller_model.pdf
%%% determine aero forces from coeffficients
X = Q*ap.S*CX + Thrust;
Y = Q*ap.S*CY;
Z = Q*ap.S*CZ;
aero_forces = [X;Y;Z];
%%% determine aero moment coefficients
Cl = ap.b*[ap.Cl0 + ap.Clbeta*beta + ap.Clp*p*ap.b/(2*V) + ap.Clr*r*ap.b/(2*V)
+ ap.Clda*da + ap.Cldr*dr];
Cm = ap.c*[ap.Cm0 + ap.Cmalpha*alpha + ap.Cmq*q*ap.c/(2*V) + ap.Cmde*de];
Cn = ap.b*[ap.Cn0 + ap.Cnbeta*beta + ap.Cnp*p*ap.b/(2*V) + ap.Cnr*r*ap.b/(2*V)
+ ap.Cnda*da + ap.Cndr*dr];
%%% determine aero moments from coeffficients
aero_moments = Q*ap.S*[Cl; Cm; Cn];%[l;m;n];
```

ttwistor.m

```
% Flight conditions and atmospheric parameters derived from AVL and other
% sources for the University of Colorado's Ttwistor Unmanned Aircraft, an
% twin-engine version of the Tempest UAS.
%
%   File created by: Eric Frew, eric.frew@colorado.edu
%
%   Data taken from files generated by Jason Roadman.
%       - Derivatives come from AVL analysis
%       - Inertias from Solidworks model
%
%   Data further modified from CFD analysis by Roger Laurence and by
%   adjustments from Eric Frew
%       - modified with new engine model, aircraft geometry and drag model
%
```

```
%
% If using this data for published work please reference:
%
% Jason Roadman, Jack Elston, Brian Argrow, and Eric W. Frew.
%   "Mission Performance of the Tempest UAS in Supercell Storms"."
%   AIAA Journal of Aircraft, 2012.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% All dimensional parameters in SI units
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_parameters.g = 9.81;           % Gravitational acceleration [m/s^2]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aircraft geometry parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_parameters.S = 0.6282; %[m^2]
aircraft_parameters.b = 3.067; %[m]
aircraft_parameters.c = 0.208; %[m]
aircraft_parameters.AR =
aircraft_parameters.b*aircraft_parameters.b/aircraft_parameters.S;
aircraft_parameters.m = 5.74; %[kg]
aircraft_parameters.W = aircraft_parameters.m*aircraft_parameters.g; %[N]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inertias from Solidworks model of Tempest
% These need to be validated, especially for Ttwistor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SLUGFT2_TO_KGM2 = 14.5939/(3.2804*3.2804);
aircraft_parameters.Ix = SLUGFT2_TO_KGM2*4106/12^2/32.2; %[kg m^2]
aircraft_parameters.Iy = SLUGFT2_TO_KGM2*3186/12^2/32.2; %[kg m^2]
aircraft_parameters.Iz = SLUGFT2_TO_KGM2*7089/12^2/32.2; %[kg m^2]
aircraft_parameters.Ixz = SLUGFT2_TO_KGM2*323.5/12^2/32.2; %[kg m^2]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Drag terms determined by curve fit to CFD analysis performed by Roger
% Laurence. Assumes general aircraft drag model
%       CD = CDmin + K(CL-CLmin)^2
% or equivalently
%       CD = CD0 + K1*CL + K*CL^2
% where
%       CD0 = CDmin + K*CLmin^2
%       K1  = -2K*CLmin
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_parameters.CDmin = 0.0240;
aircraft_parameters.CLmin = 0.2052;
aircraft_parameters.K = 0.0549;
aircraft_parameters.e = 1/(aircraft_parameters.K*aircraft_parameters.AR*pi);
```

```
aircraft_parameters.CD0 =
aircraft_parameters.CDmin+aircraft_parameters.K*aircraft_parameters.CLmin*aircr
aft_parameters.CLmin;
aircraft_parameters.K1 = -2*aircraft_parameters.K*aircraft_parameters.CLmin;
aircraft_parameters.CDpa = aircraft_parameters.CD0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Engine parameters, assuming model from Beard and Mclain that gives zero
% thrust for zero throttle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_parameters.Sprop = 0.0707;
aircraft_parameters.Cprop = 1;
aircraft_parameters.kmotor = 30;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Zero angle of attack aerodynamic forces and moments
% - some sources (like text used for ASEN 3128) define the body
%   coordinate system as the one that gives zero total lift at
%   zero angle of attack
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_parameters.CL0 = 0.2219;
aircraft_parameters.Cm0 = 0.0519;
aircraft_parameters.CY0 = 0;
aircraft_parameters.Cl0 = 0;
aircraft_parameters.Cn0 = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Longtidunal nondimensional stability derivatives from AVL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_parameters.CLalpha = 6.196683;
aircraft_parameters.Cmalpha = -1.634010;
aircraft_parameters.CLq = 10.137584;
aircraft_parameters.Cmq = -24.376066;
% Neglected parameters, check units below if incorporated later
aircraft_parameters.CLalphadot = 0;
aircraft_parameters.Cmalphadot = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Lateral-directional nondimensional stability derivatives from AVL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_parameters.CYbeta = -0.367231;
aircraft_parameters.Clbeta = -0.080738;
aircraft_parameters.Cnbeta = 0.080613;
aircraft_parameters.CYp = -0.064992;
aircraft_parameters.Clp = -0.686618;
aircraft_parameters.Cnp = -0.039384;
aircraft_parameters.Clr = 0.119718;
aircraft_parameters.Cnr = -0.052324;
aircraft_parameters.CYr = 0.213412;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Control surface deflection parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Elevator
```

```
aircraft_parameters.CLde =   0.006776;
aircraft_parameters.Cmde =  -0.06;
% Aileron
aircraft_parameters.CYda =  -0.000754;
aircraft_parameters.Clda =  -0.02;
aircraft_parameters.Cnda =  -0.000078;
% Rudder
aircraft_parameters.CYdr =   0.003056;
aircraft_parameters.Cldr =   0.000157;
aircraft_parameters.Cndr =  -0.000856;
```

stdatmo.m

```
function [rho,a,temp,press,kvisc,ZorH]=stdatmo(H_in,Toffset,Units,GeomFlag)
%  STDATMO Find gas properties in earth's atmosphere.
%   [rho,a,T,P,nu,ZorH]=STDATMO(H,dT,Units,GeomFlag)
%
%   STDATMO by itself gives the atmospheric properties at sea level on a
%   standard day.
%
%   STDATMO(H) returns the properties of the 1976 Standard Atmosphere at
%   geopotential altitude H (meters), where H is a scalar, vector, matrix,
%   or ND array.
%
%   STDATMO(H,dT) returns properties when the temperature is dT degrees
%   offset from standand conditions. H and dT must be the same size or else
%   one must be a scalar.
%
%   STDATMO(H,dT,Units) specifies units for the inputs outputs. Options are
%   SI (default) or US (a.k.a. Imperial, English). For SI, set Units to []
%   or 'SI'. For US, set Units to 'US'. Input and output units may be
%   different by passing a cell array of the form {Units_in Units_out},
%   e.g. {'US' 'SI'}. Keep in mind that dT is an offset, so when converting
%   between Celsius and Fahrenheit, use only the scaling factor (dC/dF =
%   dK/dR = 5/9). Units are as follows:
%       Input:                      SI (default)    US
%           H:     Altitude          m               ft
%           dT:    Temp. offset      °C/°K           °F/°R
%       Output:
%           rho:   Density           kg/m^3          slug/ft^3
%           a:     Speed of sound    m/s             ft/s
%           T:     Temperature       °K              °R
%           P:     Pressure          Pa              lbf/ft^2
%           nu:    Kinem. viscosity  m^2/s           ft^2/s
%           ZorH:  Height or altitude m              ft
%
%   STDATMO(H,dT,u), where u is a structure created by the UNITS function,
%   accepts variables of the DimensionedVariable class as inputs. Outputs
%   are of the DimensionedVariable class. If a DimensionedVariable is not
```

```
%   provided for an input, STDATMO assumes SI input.
%
%   STDATMO(H,dT,Units,GeomFlag) with logical input GeomFlag returns
%   properties at geometric altitude input H instead of the normal
%   geopotential altitude.
%
%   [rho,a,T,P,nu]=STDATMO(H,dT,...) returns atmospheric properties the
%   same size as H and/or dT (P does not vary with temperature offset and
%   is always the size of H)
%
%   [rho,a,T,P,nu,ZorH]=STDATMO(H,...) returns either geometric height, Z,
%   (GeomFlag not set) or geopotential height, H, (GeomFlag set).
%
%   Example 1: Find atmospheric properties at every 100 m of geometric
%   height for an off-standard atmosphere with temperature offset varying
%   +/- 25°C sinusoidally with a period of 4 km.
%       Z = 0:100:86000;
%       [rho,a,T,P,nu,H]=stdatmo(Z,25*sin(pi*Z/2000),'',true);
%       semilogx(rho/stdatmo,H/1000)
%       title('Density variation with sinusoidal off-standard atmosphere')
%       xlabel('\sigma'); ylabel('Altitude (km)')
%
%   Example 2: Create tables of atmospheric properties up to 30000 ft for a
%   cold (-15°C), standard, and hot (+15°C) day with columns
%   [h(ft) Z(ft) rho(slug/ft3) sigma a(ft/s) T(R) P(psf) µ(slug/ft-s)
nu(ft2/s)]
%   using 3-dimensional array inputs.
%       [~,h,dT]=meshgrid(0,-5000:1000:30000,-15:15:15);
%       [rho,a,T,P,nu,Z]=stdatmo(h,dT*9/5,'US',0);
%       Table = [h Z rho rho/stdatmo(0,0,'US') T P nu.*rho nu];
%       format short e
%       ColdTable       = Table(:,:,1)
%       StandardTable   = Table(:,:,2)
%       HotTable        = Table(:,:,3)
%
%   Example 3: Use the unit consistency enforced by the DimensionedVariable
%   class to find the SI dynamic pressure, Mach number, Reynolds number, and
%   stagnation temperature of an aircraft flying at flight level FL500
%   (50000 ft) with speed 500 knots and characteristic length of 80 inches.
%       u = units;
%       V = 500*u.kts; c = 80*u.in;
%       [rho,a,T,P,nu]=stdatmo(50*u.kft,[],u);
%       Dyn_Press = 1/2*rho*V^2;
%       M = V/a;
%       Re = V*c/nu;
%       T0 = T*(1+(1.4-1)/2*M^2);
%
%   This atmospheric model is not recommended for use at altitudes above
%   86 km geometric height (84852 m/278386 ft geopotential) and returns NaN
```

```
%   for altitudes above 90 km geopotential.
%
%   See also DENSITYALT, ATMOSISA, ATMOSNONSTD, DENSITYALT,
%               UNITS, DIMENSIONEDVARIABLE.
%               http://www.mathworks.com/matlabcentral/fileexchange/39325
%               http://www.mathworks.com/matlabcentral/fileexchange/38977
%
%   [rho,a,T,P,nu,ZorH]=STDATMO(H,dT,Units,GeomFlag)
% About:
%{
Author:     Sky Sartorius
           www.mathworks.com/matlabcentral/fileexchange/authors/101715
References: ESDU 77022
          www.pdas.com/atmos.html
%}
if nargin >= 3 && isstruct(Units)
   U = true;
   u = Units;
   if isa(H_in,'DimensionedVariable')
       H_in = H_in/u.m;
   end
   if isa(Toffset,'DimensionedVariable')
       Toffset = Toffset/u.K;
   end

   Units = 'si';
else
   U = false;
end
if nargin == 0
   H_in = 0;
end
if nargin < 2 || isempty(Toffset)
   Toffset = 0;
end
if nargin <= 2 && all(H_in(:) <= 11000) %quick troposphere-only code
   TonTi=1-2.255769564462953e-005*H_in;
   press=101325*TonTi.^(5.255879812716677);
   temp = TonTi*288.15 + Toffset;
   rho = press./temp/287.05287;

   if nargout > 1
       a = sqrt(401.874018 * temp);
       if nargout >= 5
           kvisc = (1.458e-6 * temp.^1.5 ./ (temp + 110.4)) ./ rho;
           if nargout == 6 % Assume Geop in, find Z
               ZorH = 6356766*H_in./(6356766-H_in);
           end
       end
```

```
   end
   return
end
% index    Lapse rate   Base Temp     Base Geopo Alt         Base Pressure
%   i        Ki (°C/m)    Ti (°K)         Hi (m)               P (Pa)
D =[1       -.0065      288.15          0                    101325
   2        0           216.65         11000                 22632.0400950078
   3        .001        216.65         20000                 5474.87742428105
   4        .0028       228.65         32000                 868.015776620216
   5        0           270.65         47000                 110.90577336731
   6        -.0028      270.65         51000                 66.9385281211797
   7        -.002       214.65         71000                 3.9563921603966
   8        0           186.94590831019 84852.0458449057     0.373377173762337
];
% Constants
R=287.05287;%N-m/kg-K; value from ESDU 77022
% R=287.0531;   %N-m/kg-K; value used by MATLAB aerospace toolbox ATMOSISA
gamma=1.4;
g0=9.80665;     %m/sec^2
RE=6356766;     %Radius of the Earth, m
Bs = 1.458e-6;  %N-s/m2 K1/2
S = 110.4;      %K
K=D(:,2);    %°K/m
T=D(:,3);    %°K
H=D(:,4);    %m
P=D(:,5);    %Pa
temp=zeros(size(H_in));
press=temp;
hmax = 90000;
if nargin < 3 || isempty(Units)
   Uin = false;
   Uout = Uin;
elseif isnumeric(Units) || islogical(Units)
   Uin = Units;
   Uout = Uin;
else
   if ischar(Units) %input and output units the same
       Unitsin = Units; Unitsout = Unitsin;
   elseif iscell(Units) && length(Units) == 2
       Unitsin = Units{1}; Unitsout = Units{2};
   elseif iscell(Units) && length(Units) == 1
       Unitsin = Units{1}; Unitsout = Unitsin;
   else
       error('Incorrect Units definition. Units must be ''SI'', ''US'', or
2-element cell array')
   end

   if strcmpi(Unitsin,'si')
       Uin = false;
```

```
   elseif strcmpi(Unitsin,'us')
       Uin = true;
   else error('Units must be ''SI'' or ''US''')
   end

   if strcmpi(Unitsout,'si')
       Uout = false;
   elseif strcmpi(Unitsout,'us')
       Uout = true;
   else error('Units must be ''SI'' or ''US''')
   end
end
% Convert from imperial units, if necessary.
if Uin
   H_in = H_in * 0.3048;
   Toffset = Toffset * 5/9;
end
% Convert from geometric altitude to geopotental altitude, if necessary.
if nargin < 4
   GeomFlag = false;
end
if GeomFlag
   Hgeop=(RE*H_in)./(RE+H_in);
else
   Hgeop=H_in;
end
n1=(Hgeop<=H(2));
n2=(Hgeop<=H(3) & Hgeop>H(2));
n3=(Hgeop<=H(4) & Hgeop>H(3));
n4=(Hgeop<=H(5) & Hgeop>H(4));
n5=(Hgeop<=H(6) & Hgeop>H(5));
n6=(Hgeop<=H(7) & Hgeop>H(6));
n7=(Hgeop<=H(8) & Hgeop>H(7));
n8=(Hgeop<=hmax & Hgeop>H(8));
n9=(Hgeop>hmax);
% Troposphere
if any(n1(:))
   i=1;
   TonTi=1+K(i)*(Hgeop(n1)-H(i))/T(i);
   temp(n1)=TonTi*T(i);
   PonPi=TonTi.^(-g0/(K(i)*R));
   press(n1)=P(i)*PonPi;
end
% Tropopause
if any(n2(:))
   i=2;
   temp(n2)=T(i);
   PonPi=exp(-g0*(Hgeop(n2)-H(i))/(T(i)*R));
   press(n2)=P(i)*PonPi;
```

```
end
% Stratosphere 1
if any(n3(:))
   i=3;
   TonTi=1+K(i)*(Hgeop(n3)-H(i))/T(i);
   temp(n3)=TonTi*T(i);
   PonPi=TonTi.^(-g0/(K(i)*R));
   press(n3)=P(i)*PonPi;
end
% Stratosphere 2
if any(n4(:))
   i=4;
   TonTi=1+K(i)*(Hgeop(n4)-H(i))/T(i);
   temp(n4)=TonTi*T(i);
   PonPi=TonTi.^(-g0/(K(i)*R));
   press(n4)=P(i)*PonPi;
end
% Stratopause
if any(n5(:))
   i=5;
   temp(n5)=T(i);
   PonPi=exp(-g0*(Hgeop(n5)-H(i))/(T(i)*R));
   press(n5)=P(i)*PonPi;
end
% Mesosphere 1
if any(n6(:))
   i=6;
   TonTi=1+K(i)*(Hgeop(n6)-H(i))/T(i);
   temp(n6)=TonTi*T(i);
   PonPi=TonTi.^(-g0/(K(i)*R));
   press(n6)=P(i)*PonPi;
end
% Mesosphere 2
if any(n7(:))
   i=7;
   TonTi=1+K(i)*(Hgeop(n7)-H(i))/T(i);
   temp(n7)=TonTi*T(i);
   PonPi=TonTi.^(-g0/(K(i)*R));
   press(n7)=P(i)*PonPi;
end
% Mesopause
if any(n8(:))
   i=8;
   temp(n8)=T(i);
   PonPi=exp(-g0*(Hgeop(n8)-H(i))/(T(i)*R));
   press(n8)=P(i)*PonPi;
end
if any(n9(:))
   warning('One or more altitudes above upper limit.')
```

```
    temp(n9)=NaN;
    press(n9)=NaN;
end
temp = temp + Toffset;
rho = press./temp/R;
if nargout >= 2
    a = sqrt(gamma * R * temp);
    if nargout >= 5
        kvisc = (Bs * temp.^1.5 ./ (temp + S)) ./ rho; %m2/s
        if nargout == 6
            if GeomFlag % Geometric in, ZorH is geopotential altitude (H)
                ZorH = Hgeop;
            else % Geop in, find Z
                ZorH = RE*Hgeop./(RE-Hgeop);
            end
        end
    end
end
if Uout %convert to imperial units if output in imperial units
    rho = rho / 515.3788;
    if nargout >= 2
        a = a / 0.3048;
        temp = temp * 1.8;
        press = press / 47.88026;
        if nargout >= 5
            kvisc = kvisc / 0.09290304;
            if nargout == 6
                ZorH = ZorH / 0.3048;
            end
        end
    end
end
if U
    rho = rho*u.kg/(u.m^3);
    if nargout >= 2
        a = a*u.m/u.s;
        temp = temp*u.K;
        press = press*u.Pa;
        if nargout >= 5
            kvisc = kvisc*u.m^2/u.s;
            if nargout == 6
                ZorH = ZorH*u.m;
            end
        end
    end
end

end
% Credit for elements of coding scheme:
```

```
% cobweb.ecn.purdue.edu/~andrisan/Courses/AAE490A_S2001/Exp1/
% Revision history:
%{
V1.0    5 July 2010
V1.1    8 July 2010
      Update to references and improved input handling
V2.0    12 July 2010
      Changed input ImperialFlag to Units. Units must now be a string or cell
      array {Units_in Units_out}. Version 1 syntax works as before.
 Two examples added to help
V2.1    15 July 2010
      Changed help formatting
      Sped up code - no longer caclulates a or nu if outputs not specified.
      Also used profiler to speed test against ATMOSISA, which is
      consistently about 5 times slower than STDATMO
         17 July 2010
      Cleaned up Example 1 setup using meshgrid
         26 July 2010
      Switched to logical indexing, which sped up running Example 1
      significantly(running [rho,a,T,P,nu,h]=stdatmo(Z,dT,'US',1) 1000 times:
      ~.67s before, ~.51s after)
V3.0    7 August 2010
      Consolodated some lines for succintness Changed Hgeop output to be
      either geopotential altitude or geometric altitude, depending on which
      was input. Updated help and examples accordingly.
V3.1    27 August 2010
      Added a very quick, troposhere-only section
V3.2    23 December 2010
      Minor changes, tested on R2010a, and sinusoidal example added
V4.0    6 July 2011
      Imperial temp offset now °F/°R instead of °C/°K
V4.1    12 Sep 2012
      Added ZorH output support for quick troposphere calculation
      uploaded
V4.2
 tiny changes to help and input handling
      nov 2012: some :s added to make use of any() better
      added see alsos
 uploaded
V5.0
 STDATMODIM wrapper created that takes DimensionedVariable input
 uploaded 5 Dec 2012
V6.0
 STDATMODIM functionality integrated into STDATMO; example three changed
 for illustration.
%}
```