UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Industriale

Classe delle Lauree Magistrali in Ingegneria Aerospaziale e Astronautica (LM 20)

Course

Flight Dynamics & Simulation

Project Work

A collection of exercises

Instructor:

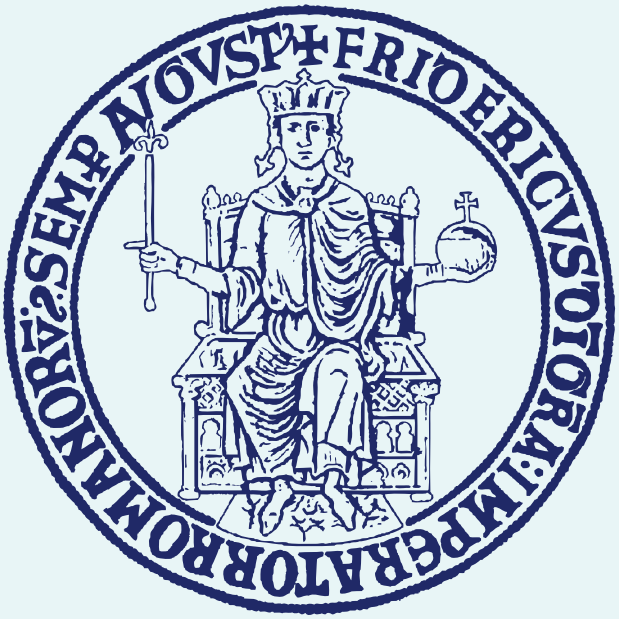
Prof. Agostino De Marco

Student:

Matteo De Luca Picione

ID: M53001994

Anno Accademico 2024/2025



UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

Abstract

1. **Analysis of scenario**. Understanding and accurately simulating aircraft dynamics is fundamental to the aerospace industry as it underpins the design, testing, and certification of safe and efficient aircrafts.
2. **Statement of the problem**. The problem focuses on the dynamic behavior of an aircraft subjected to varying initial conditions and control inputs. By leveraging Euler's equations of motion, this study aims to summarize several mathematical models that captures the complex interactions between aerodynamic forces, propulsion, and gravitational effects.
3. **Adopted methodology**. Numerical integration techniques are employed to solve the resulting system of nonlinear ordinary differential equations (ODEs), enabling the simulation of aircraft trajectories under diverse operational scenarios. Using MATLAB and Simulink, the framework allows for an in-depth evaluation of transient responses and stability, offering insights into the sensitivity of motion to pilot commands and environmental perturbations. Key scenarios include steady-level flight and maneuvering, each analyzed to validate the robustness of the implemented simulation approach.
4. **Main results**. Simulations accurately solve the system of ordinary differential equations governing six-degrees-of-freedom aircraft motions, three-degrees-of-freedom aircraft motions, searching of the trim conditions and studying of the stability, and they demonstrate reliable performances.

Keywords: Aircraft Dynamics, Numerical Simulation, Flight Dynamics, Aircraft Stability and Control, Flight Simulation, Trajectory Prediction

Contents

List of Listings

Chapter 1

KINEMATICS

A review of the kinematics equations used to study aircrafts motion will be presented in this section. Some exercises will follow to show how I implemented such equations in MATLAB to study and visualize different maneuvers. Theory and application discussed will cover all the two possible formulations of the aircraft orientation: *Euler angles* formulation and *Quaternion* formulation.

1.1 Aircraft orientation using Euler angles

Consider two frames of reference $\mathscr{F}_1 = \{C_1,\ x_1,\ y_1,\ z_1\}$ and $\mathscr{F}_2 = \{C_2,\ x_2,\ y_2,\ z_2\}$, where $\mathscr{F}_2$ is obtained from $\mathscr{F}_1$ by a rigid translation of the vector $C_2 - C_1$ and a rotation of ϕ about the z_1 axis. Then, an arbitrary vector $\boldsymbol{u}$, whose components in $\mathscr{F}_1$ are $\left[u_{x_1},\ u_{y_1},\ u_{z_1}\right]^{\mathrm{T}}$, can be expressed in $\mathscr{F}_2$ with a new set of components given by the following

$$\begin{Bmatrix} u_{x_2} \\ u_{y_2} \\ u_{z_2} \end{Bmatrix} = [R_3(\phi)] \begin{Bmatrix} u_{x_1} \\ u_{y_1} \\ u_{z_1} \end{Bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} u_{x_1} \\ u_{y_1} \\ u_{z_1} \end{Bmatrix} \tag{1.1}$$

where $[R_3(\phi)]$ is the *elementary transformation matrix* for a rotation of ϕ about the third axis of $\mathscr{F}_1$.

In the same way the elementary transformation matrices $[R_2(\phi)]$ and $[R_1(\phi)]$ can be defined for rotations of ϕ about the second and third axes of $\mathscr{F}_1$ respectively. In particular,

$$[R_2(\phi)] = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \tag{1.2}$$

$$[R_1(\phi)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{1.3}$$

An arbitrary rigid transformation between two different frames of reference can always be decomposed by at most three consecutive elementary transformations. Given that, every so defined change of basis from $\mathscr{F}_1$ to $\mathscr{F}_2$ can be characterized by a sequence

of appropriately chosen rotations. This translates to mathematically identify the transformation by the matrix $[T]_{21}$, given by sequentially applied elementary rotation matrices, e.g.

$$[T]_{21} = [R_2(\alpha)]\,[R_1(\beta)]\,[R_2(\gamma)] \tag{1.4}$$

for a 212 sequence of rotations. The angles α, β and γ are called Euler angles. It is worth noticing that being $[R_1(\alpha)]$, $[R_2(\alpha)]$ and $[R_3(\alpha)]$ orthogonal matrices trivially implies that $[T]_{21}$ is orthogonal too, namely

$$[T]_{21}^{-1} = [T]_{12} = [T]_{21}^{\mathrm{T}} \tag{1.5}$$

In the study of flight kinematics and under the rigid body assumption, the change of basis between $\mathscr{F}_{\mathrm{E}}$ and $\mathscr{F}_{\mathrm{B}}$ is required to completely define the orientation of the aircraft. In pursuit of such goal the 312 sequence of rotations will be used. In particular we write

$$[T]_{\mathrm{BE}} = [R_1(\varphi)]\,[R_2(\theta)]\,[R_3(\psi)] \tag{1.6}$$

such that, given an arbitrary vector $\boldsymbol{u}$ and considering its components, the following relation holds

$$\{u\}_{\mathrm{B}} = [T]_{\mathrm{BE}}\,\{u\}_{\mathrm{E}} \tag{1.7}$$

1.1.1 Gimbal equations

An important application of the change of basis formulae synthesized by equation (1.4) is the evaluation of the Euler angles rates of change in terms of the angular velocity components. Since $\mathscr{F}_{\mathrm{B}}$ is obtained from $\mathscr{F}_{\mathrm{E}}$ by three different consecutive transformations, each Euler angles rate is an angular velocity component in the respective intermediate frame of reference.

We can obtain the angular velocity components in $\mathscr{F}_{\mathrm{B}}$ by applying the appropriate transformation on each of these rates and combine them all, hence we write

$$\{\Omega_{\mathrm{B}}\}_{\mathrm{B}} = [R_1(\varphi)]\begin{Bmatrix}\dot{\varphi}\\0\\0\end{Bmatrix} + [R_1(\varphi)]\,[R_2(\theta)]\begin{Bmatrix}0\\\dot{\theta}\\0\end{Bmatrix} + [R_1(\varphi)]\,[R_2(\theta)]\,[R_3(\psi)]\begin{Bmatrix}0\\0\\\dot{\psi}\end{Bmatrix} \tag{1.8}$$

which can then be evaluated as shown in the follow

$$\begin{Bmatrix}p\\q\\r\end{Bmatrix} = \begin{bmatrix}1 & 0 & -\sin\theta\\ 0 & \cos\varphi & \sin\varphi\cos\theta\\ 0 & -\sin\varphi & \cos\varphi\cos\theta\end{bmatrix}\begin{Bmatrix}\dot{\varphi}\\\dot{\theta}\\\dot{\psi}\end{Bmatrix} \tag{1.9}$$

or, in a more compact form as

$$\{\Omega_{\mathrm{B}}\}_{\mathrm{B}} = [G]\begin{Bmatrix}\dot{\varphi}\\\dot{\theta}\\\dot{\psi}\end{Bmatrix} \tag{1.10}$$

where $[G]$ is known as the *gimbal matrix*. Its matrix inverse $[G]^{-1}$ is the *inverse gimbal*

matrix and lets us evaluate the following inverse transformation

$$\begin{Bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = [G]^{-1} \{\Omega_{\mathrm{B}}\}_{\mathrm{B}} = \begin{bmatrix} 1 & \dfrac{\sin\varphi \sin\theta}{\cos\theta} & \dfrac{\cos\varphi \sin\theta}{\cos\theta} \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \dfrac{\sin\varphi}{\cos\theta} & \dfrac{\cos\varphi}{\cos\theta} \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{1.11}$$

Equations (1.9) and (1.11) are called *gimbal equations*. During the motion they instantly define the time rates of change of the orientation of the frames of reference in which the Euler angles are defined.

Given some initial conditions, the inverse gimbal equations must be integrated over time in order to determine the Euler angles as functions of time and solve for the aircraft orientation, but they have a singularity. In particular, when the orientation is such that $\theta = \pi/2$ or $\theta = -\pi/2$, the gimbal matrix cannot be inverted, so $[G]^{-1}$ cannot be evaluated and the problem is not solvable anymore. This indetermination is commonly known as *gimbal lock*.

Every possible sequence of Euler angles eventually yields to a singularity of some kind when trying to solve for the orientation. To bypass this inconvenience two different sequences of rotations must be used, switching between them to prevent such singularities.

A more convenient solution requires to change formulation of the aircraft orientation and use quaternions instead of Euler angles. This approach will be later presented in § 1.2.

1.1.2 Navigation equations

Equation (1.7) can be trivially applied to the center of gravity velocity components to write

$$\{V_G\}_{\mathrm{B}} = [T]_{\mathrm{BE}} \{V_G\}_{\mathrm{E}} \tag{1.12}$$

Using $C_\phi \equiv \cos\phi$ and $S_\phi \equiv \sin\phi$, it can be explicitly rewritten as follows

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ S_\varphi S_\theta C_\psi - C_\varphi S_\psi & S_\varphi S_\theta S_\psi + C_\varphi C_\psi & S_\varphi C_\theta \\ C_\varphi S_\theta C_\psi + S_\varphi S_\theta & C_\varphi S_\theta S_\psi - S_\varphi C_\psi & C_\varphi C_\theta \end{bmatrix} \begin{Bmatrix} \dot{x}_{\mathrm{E},G} \\ \dot{y}_{\mathrm{E},G} \\ \dot{z}_{\mathrm{E},G} \end{Bmatrix} \tag{1.13}$$

This equation can be inverted to obtain

$$\begin{Bmatrix} \dot{x}_{\mathrm{E},G} \\ \dot{y}_{\mathrm{E},G} \\ \dot{z}_{\mathrm{E},G} \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & S_\varphi S_\theta C_\psi - C_\varphi S_\psi & C_\varphi S_\theta C_\psi + S_\varphi S_\theta \\ C_\theta S_\psi & S_\varphi S_\theta S_\psi + C_\varphi C_\psi & C_\varphi S_\theta S_\psi - S_\varphi C_\psi \\ -S_\theta & S_\varphi C_\theta & C_\varphi C_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{1.14}$$

which can be integrated over time to determine the center of gravity components in $\mathscr{F}_{\mathrm{E}}$ as functions of time, when some initial conditions are given.

Equations in (1.14) are called *navigation equations*. Together with the gimbal equations they constitute the system of equations which must be solved in order to completely determine every arbitrary six degrees of freedom Aircraft motion.

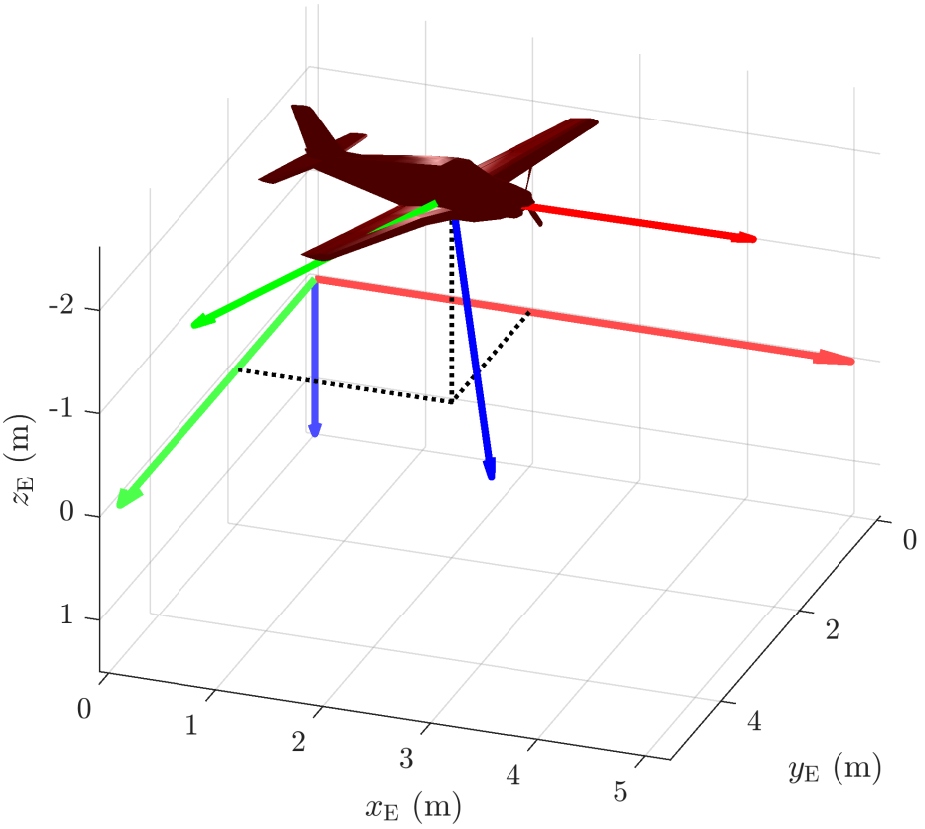


Figure 1.1 3D visualization of an Aircraft STL model with Earth and Body axes.

1.1.3 Exercises

Aircraft orientation in Earth axes

This exercise aims to show the usage of some pre-coded MATLAB functions to display a model of the aircraft body in a 3D figure. The figure will also contain the Earth axes and the Body axes displayed using the *rgb order convention*.

The following MATLAB script takes and reads a 3D STL model of the aircraft, scales it by a `shapeScaleFactor` and writes its *vertices*, *faces* and *connectivity* data into the fields `V`, `F` and `C` of the struct `shape`. Then, the MATLAB binary file `aircraft_pa24-250.mat` is saved to be later loaded by other scripts.

Listing 1.1 Script for saving an STL 3D model as a MATLAB binary file.

```
close all; clc; clear;

%% Load the A/C 3D model
shapeScaleFactor = 1.0;
[V, F, C] = loadAircraftSTL('aircraft_pa24-250.stl', shapeScaleFactor);
shape.V = V; shape.F = F; shape.C = C;
save('aircraft_pa24-250.mat', "shape");
```

This other following script displays the figure with the aircraft 3D model and the Earth and Body axes. It starts setting up the figure and the Earth axes directions then loading and scaling the binary file `aircraft_pa24-250.mat` previously saved. Then, the center of gravity position $[x_{E,G},\ y_{E,G},\ z_{E,G}] = [2\text{ m},\ 2\text{ m},\ -2\text{ m}]$ and the Euler angles $[\psi,\ \theta,\ \varphi] = [20\text{ deg},\ 10\text{ deg},\ 0\text{ deg}]$ are assigned.

The struct `bodyAxesOptions` is defined to contain the fields `show`, `magX`, `magY`, `magZ` and `lineWidth` which are used as input options to the pre-coded function `plotBodyE`. It

plots the aircraft oriented shape and Body axes and sets them in place.

The function `plotEarthAxes` plots the Earth axes with origin in position `vXYZ0` and axis lengths `vExtent`, while `plot3DHelperLines` displays the dotted helper lines visible in the output figure 1.1.

Listing 1.2 Script that shows and exports a 3D figure containing the aircraft model set in place.

```
close all; clc; clear;

%% Setup the figure
h_fig1 = figure(1);
light('Position', [1, 0, -2], 'Style', 'local')
set(gca, 'XDir', 'reverse'); set(gca, 'ZDir', 'reverse')
grid on; hold on

%% Load A/C shape
shapeScaleFactor = 1.75;
shape = loadAircraftMAT('aircraft_pa24-250.mat', shapeScaleFactor);

%% Set the A/C in place
vXYZe = [2, 2, -2];
vEulerAngles = convang([20, 10, 0], 'deg', 'rad');
theView = [200, 25];
bodyAxesOptions.show = true;
bodyAxesOptions.magX = 2 * shapeScaleFactor;
bodyAxesOptions.magY = 2 * shapeScaleFactor;
bodyAxesOptions.magZ = 1.5 * shapeScaleFactor;
bodyAxesOptions.lineWidth = 2.5;
plotBodyE(h_fig1, shape, vXYZe, vEulerAngles, bodyAxesOptions, theView)

%% Plot Earth axes
hold on
xMax = max([abs(vXYZe(1)), 5]);
yMax = max([abs(vXYZe(2)), 5]);
zMax = .3 * xMax;
vXYZ0 = [0, 0, 0];
vExtent = [xMax, yMax, zMax];
plotEarthAxes(h_fig1, vXYZ0, vExtent)

%% Draw CoG coordinate helper lines
hold on
plotPoint3DHelperLines(h_fig1, vXYZe)

set(gca, 'TickLabelInterpreter', 'latex')
xlabel("$x_\mathrm{E}$ (m)", 'Interpreter', 'latex')
ylabel("$y_\mathrm{E}$ (m)", 'Interpreter', 'latex')
zlabel("$z_\mathrm{E}$ (m)", 'Interpreter', 'latex')

exportgraphics(gca, 'ex2_1.pdf', 'Resolution', 450)
```

Weight components in Body axes

First, aircraft mass is assigned. Then, the aircraft shape, Body axes and Earth axes are displayed by the function `plotACBodyEarth` that repeats the same procedure shown in the listing 1.2.

The transformation matrix between $\mathcal{F}_\mathrm{E}$ and $\mathcal{F}_\mathrm{B}$ is evaluated using the function `angle2dcm` of the MATLAB *Aerospace Toolbox*. Its numerical value is displayed in the terminal window, obtaining

$$[T]_{\mathrm{BE}} = \begin{bmatrix} 0.9254 & 0.3368 & -0.1736 \\ -0.3420 & 0.9397 & 0 \\ 0.1632 & 0.0594 & 0.9848 \end{bmatrix}$$

The weight components in Earth axes are evaluated and then used to compute the weight components in Body axes. The function `quiver3` ends the figure 1.2 by plotting

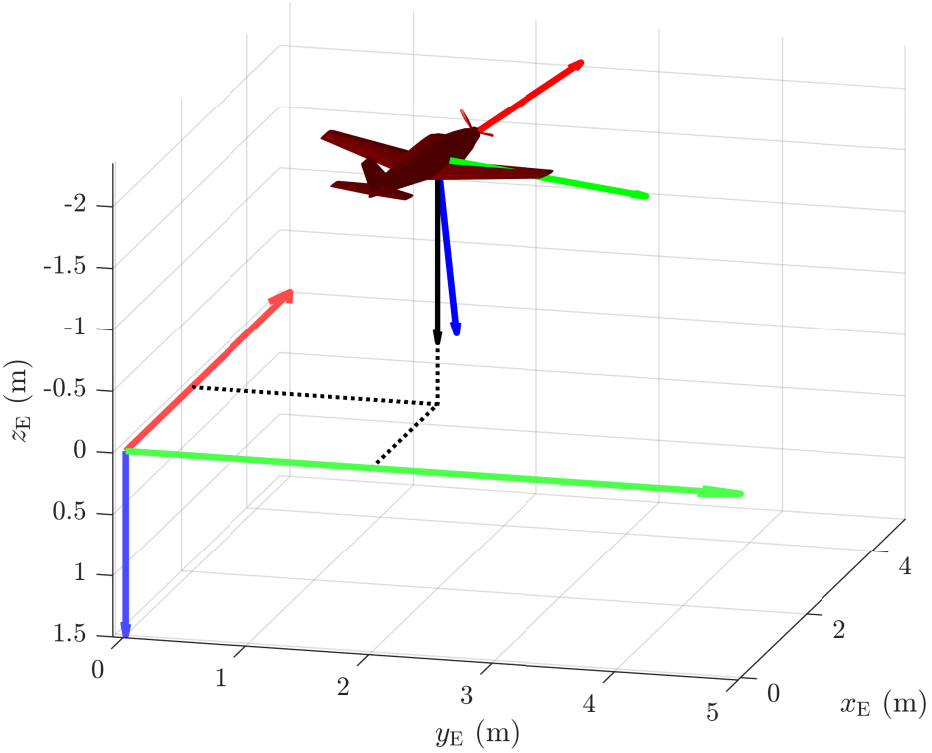


Figure 1.2 3D visualization of the Aircraft model, Earth and Body axes and the weight vector.

the weight vector $\boldsymbol{W}$. Its components values are displayed in the terminal window.

$$\{W\}_E = \begin{Bmatrix} 0\,\text{N} \\ 0\,\text{N} \\ 11772\,\text{N} \end{Bmatrix}$$

$$\{W\}_B = \begin{Bmatrix} -2044.0\,\text{N} \\ 0\,\text{N} \\ 11593\,\text{N} \end{Bmatrix}$$

Listing 1.3 Script that evaluates weight components in Body axes and plot the aircraft and the weight vector.

```
close all; clc; clear;

%% Mass data
mass = 1200; % kg
g = 9.81; % m / s^2

%% Euler angles
[vEulerAngles, vXYZe] = plotACBodyEarth([20, 10, 0]);
psi = vEulerAngles(1); theta = vEulerAngles(2); phi = vEulerAngles(3);

%% DCM
TBE = angle2dcm(psi, theta, phi, 'ZYX')

%% Weight
hold on
vWeight_E = [0; 0; mass*g]
vWeight_B = TBE * vWeight_E
quiver3(vXYZe(1), vXYZe(2), vXYZe(3), ...
    vWeight_E(1), vWeight_E(2), vWeight_E(3), ...
    1.5/mass/g, 'color', 'k', 'linewidth', 2)

```

```
set(gca, 'TickLabelInterpreter', 'latex')
xlabel("$x_\mathrm{E}$ (m)", 'Interpreter', 'latex')
ylabel("$y_\mathrm{E}$ (m)", 'Interpreter', 'latex')
zlabel("$z_\mathrm{E}$ (m)", 'Interpreter', 'latex')
exportgraphics(gca, 'ex23main.pdf', 'Resolution', 450)
```

Banking turn maneuver

This exercise aims to show a procedure for visualizing an arbitrary assigned maneuver.

The aircraft trajectory is displayed from the input time laws of aircraft center of gravity coordinates and Euler angles. In this particular case a *banking turn maneuver* is assigned.

After the figure has been set up and the aircraft shape has been loaded, the time domain is defined and the time samples used for the model plot is assigned. That and all the other options are saved as fields of the struct `trajectoryOptions`, to later be used for the actual plot.

Given the time discretization defined, the following time laws for the center of gravity components can be assigned

$$x_{E,G}(t) = (2250\ \mathrm{m})\ \left[1 - \cos\left(\frac{\pi}{10\ \mathrm{s}}t\right)\right]$$

$$y_{E,G}(t) = (2250\ \mathrm{m})\ \sin\left(\frac{\pi}{10\ \mathrm{s}}t\right)$$

$$z_{E,G}(t) = -1000\ \mathrm{m}$$

and the time laws for the Euler angles as follows

$$\varphi(t) = -\frac{\pi\ \mathrm{rad}}{3}$$

$$\theta(t) = 0\ \mathrm{rad}$$

$$\psi(t) = (\pi\ \mathrm{rad})\ \left(\frac{1}{2} - \frac{t}{10\ \mathrm{s}}\right)$$

The pre-coded MATLAB function `plotTrajectoryAndBodyE` plots the trajectory and the aircraft shape at each time sample previously assigned. The result is shown in figure 1.3.

Listing 1.4 Script that displays a banking maneuver trajectory.

```
close all; clc; clear;

%% Setup the figure
h_fig1 = figure(1);
light('Position', [1, 0, -2], 'Style', 'local')
set(gca, 'XDir', 'reverse'); set(gca, 'ZDir', 'reverse')
grid on; hold on

%% Load the A/C shape
shapeScaleFactor = 200;
shape = loadAircraftMAT('aircraft_mig29.mat', shapeScaleFactor);

%% Define the time domain
timeSteps = 100;
duration = 10;
t = linspace(0, duration, timeSteps);

```

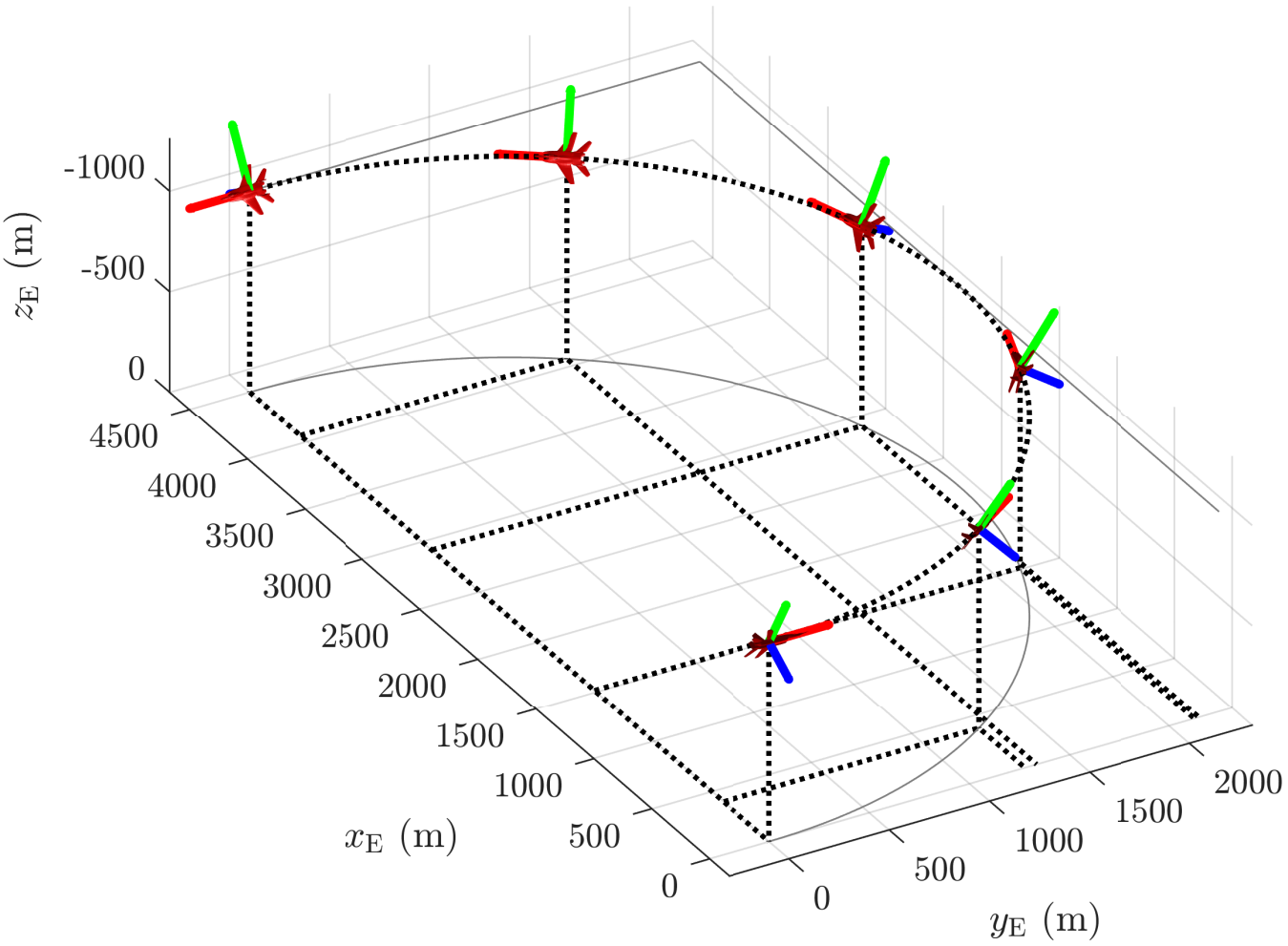


Figure 1.3 3D visualization of a banking turn maneuver.

```
%% Define the samples and the options
numberOfStages = 6;
trajectoryOptions.samples = floor(linspace(1, timeSteps,
    ↪ numberOfStages));
trajectoryOptions.theView = [60, 30];
trajectoryOptions.bodyAxes.show = true;
trajectoryOptions.bodyAxes.magX = 300;
trajectoryOptions.bodyAxes.magY = 300;
trajectoryOptions.bodyAxes.magZ = 200;
trajectoryOptions.bodyAxes.lineWidth = 2.5;
trajectoryOptions.helperLines.show = true;
trajectoryOptions.helperLines.lineColor = 'k';
trajectoryOptions.helperLines.lineWidth = 1.5;
trajectoryOptions.helperLines.lineStyle = ':';
trajectoryOptions.trajectory.show = true;
trajectoryOptions.trajectory.lineColor = 'k';
trajectoryOptions.trajectory.lineWidth = 1.5;
trajectoryOptions.trajectory.lineStyle = ':';

%% Define the CoG positions
mXYZe = zeros(timeSteps, 3);
mXYZe(:, 1) = 2250 - 2250 * cos(pi / duration * t);
mXYZe(:, 2) = 2250 * sin(pi / duration * t);
mXYZe(:, 3) = -1000;


%% Define the Euler angles
mEulerAngles = zeros(timeSteps, 3);
mEulerAngles(:, 1) = pi/2 - pi / duration * t;
```

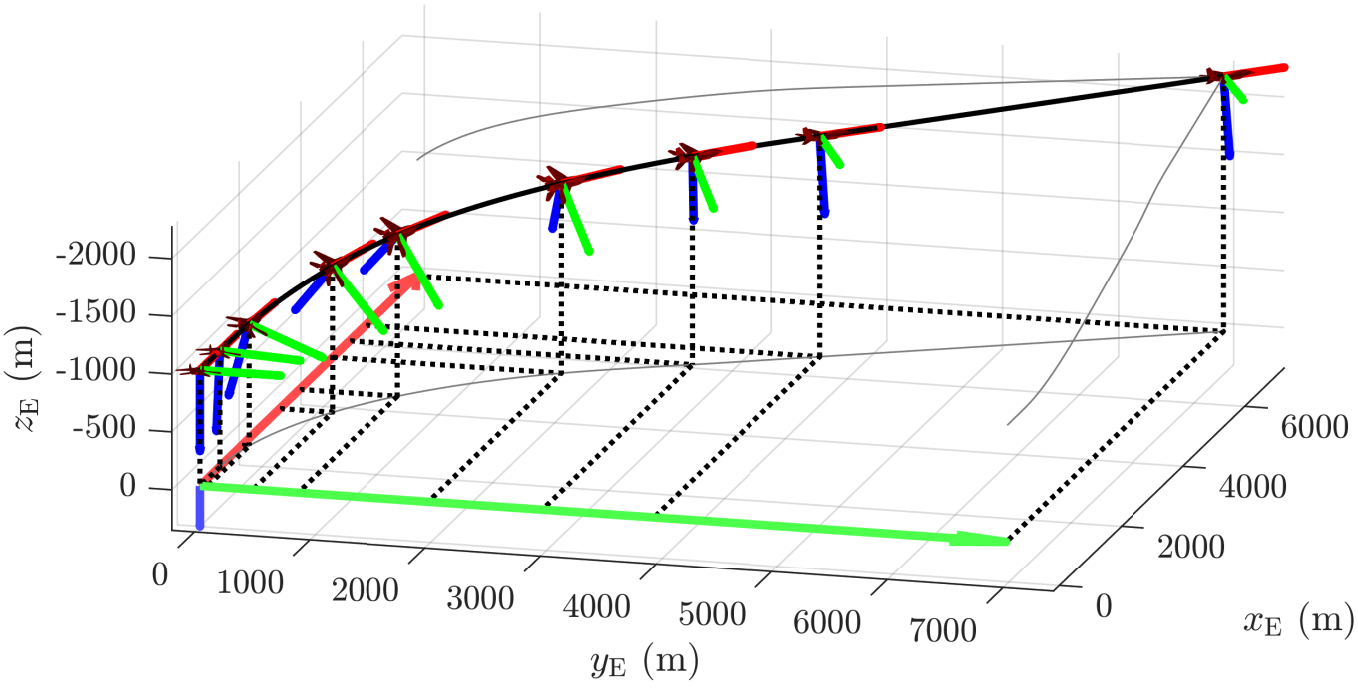


Figure 1.4 3D visualization of the maneuver obtained from the integration of the kinematic equations.

```
mEulerAngles(:, 2) = 0;
mEulerAngles(:, 3) = -pi/3;

%% Plot the trajectory and the body
plotTrajectoryAndBodyE(h_fig1, shape, mXYZe, mEulerAngles, ...
                                                  trajectoryOptions);

set(gca, 'TickLabelInterpreter', 'latex')
xlabel("$x_\mathrm{E}$ (m)", 'Interpreter', 'latex')
ylabel("$y_\mathrm{E}$ (m)", 'Interpreter', 'latex')
zlabel("$z_\mathrm{E}$ (m)", 'Interpreter', 'latex')
exportgraphics(gca, 'ex24main.pdf', 'Resolution', 450)
```

Numerical integration of the kinematic equations

In this exercise, for some given initial conditions, velocity components time law and angular velocity components time law, the aircraft motion will be completely solved.

Ending time of the maneuver and initial conditions are assigned. The time laws $p(t)$, $q(t)$, $r(t)$, $u(t)$, $v(t)$ and $w(t)$ are defined using the `interp1` pre-built MATLAB function, for some given desired breakpoints.

The right hand side of the gimbal equations (1.11) are defined with an *anonymous function* to be used to solve for the Euler angles using the `ode45` routine. The discretizations of the Euler angles so obtained are used as breakpoints for the time laws $\varphi(t)$, $\theta(t)$ and $\psi(t)$, that are necessary to solve with `ode45`.

The navigation equations (1.14) are numerically integrated and the resulting figure 1.4 is obtained through the same procedure as in listing 1.4 for the trajectory and listing 1.2 for the Earth axes.

Listing 1.5 Algorithm for the integration of the kinematics equations.

```
close all; clc; clear;

%% Problem data
```

```
tEnd = 100;
psi0 = 0;
theta0 = 0;
phi0 = 0;
z0 = -1000;

%% Maximum magnitudes for the angular velocities
pMax = convangvel(2, 'deg/s', 'rad/s');
qMax = convangvel(1, 'deg/s', 'rad/s');
rMax = convangvel(1.2, 'deg/s', 'rad/s');

%% p law
vBreakPointsP(1, :) = [0, .03, .08, .2, .25, .35, .6, .67, .75, 1] * tEnd;
vBreakPointsP(2, :) = [0, .025, .7, 1, 1, 0, -1, -1, 0, 0] * pMax;
p = @(t) interp1(vBreakPointsP(1, :), vBreakPointsP(2, :), t, 'pchip');

%% q law
vBreakPointsQ(1, :) = [0, .03, .1, .2, .48, .6, .7, 1] * tEnd;
vBreakPointsQ(2, :) = [0, .025, .75, 1, 1, -.4, 0, 0] * qMax;
q = @(t) interp1(vBreakPointsQ(1, :), vBreakPointsQ(2, :), t, 'pchip');

%% r law
vBreakPointsR(1, :) = [0, .03, .1, .2, .5, .6, 1] * tEnd;
vBreakPointsR(2, :) = [0, .025, .75, 1, 1, 0, 0] * qMax;
r = @(t) interp1(vBreakPointsR(1, :), vBreakPointsR(2, :), t, 'pchip');

%% Reference value for CoG velocity components
u0 = convvel(380, 'km/h', 'm/s');
v0 = convvel(0, 'km/h', 'm/s');
w0 = convvel(0, 'km/h', 'm/s');

%% u law
vBreakPointsU(1, :) = [0, 1/30, 1/10, 1/5, .8, 1] * tEnd;
vBreakPointsU(2, :) = [1, .8, .7, 1, 1.1, 1] * u0;
u = @(t) interp1(vBreakPointsU(1, :), vBreakPointsU(2, :), t, 'pchip');

%% v law
v = @(t) 0;

%% w law
w = @(t) 0;

%% RHS of Gimbal equations
dPhiThetaPsidt = @(t, x) [ ...
    1, sin(x(1)).*sin(x(2))./cos(x(2)), cos(x(1)).*sin(x(2))./cos(x(2)); ...
    0, cos(x(1)), -sin(x(1)); ...
    0, sin(x(1))./cos(x(2)), cos(x(1))./cos(x(2)) ...
    ] * [p(t); q(t); r(t)];

%% Solution of Gimbal equations
gimbalOptions = odeset('RelTol', 1e-9, 'AbsTol', 1e-9 * ones(1, 3));
vPhiThetaPsi0 = [phi0, theta0, psi0];
[vTime, vPhiThetaPsi] = ode45(dPhiThetaPsidt, [0, tEnd], ...
                                              vPhiThetaPsi0, ...
    gimbalOptions);

%% Arrays of velocity components
for i = 1 : numel(vTime)
    vU(i) = u(vTime(i));
    vV(i) = v(vTime(i));
    vW(i) = w(vTime(i));
end

%% Euler angles functions
fPhi = @(t) interp1(vTime, vPhiThetaPsi(:, 1), t);
fTheta = @(t) interp1(vTime, vPhiThetaPsi(:, 2), t);
fPsi = @(t) interp1(vTime, vPhiThetaPsi(:, 3), t);

%% RHS of Navigation equations
dPosEdt = @(t, pos) ...
```

```
    transpose(angle2dcm(fPsi(t), fTheta(t), fPhi(t), 'ZYX')) * ...
                                                    [u(t); v(t);
    ↪ w(t)];

%% Solution of Navigation equations
navigationOptions = odeset('RelTol', 1e-3, 'AbsTol', 1e-3 * ones(1, 3));
vPosE0 = [0; 0; 0];
[~, vPosE] = ode45(dPosEdt, vTime, vPosE0, navigationOptions);

%% Arrays of CoG positions
vXe = vPosE(:, 1);
vYe = vPosE(:, 2);
vZe = vPosE(:, 3);

%% Setup the figure
h_fig1 = figure(1);
grid on; hold on
light('Position', [1, 0, -4], 'Style', 'local');
set(gca, 'XDir', 'reverse'); set(gca, 'ZDir', 'reverse')
daspect([1, 1, 1])

%% Load the A/C shape
shapeScaleFactor = 350;
shape = loadAircraftMAT('aircraft_mig29.mat', shapeScaleFactor);

%% Sequence of positions and Euler angles
mXYZe = [vXe, vYe, vZe+z0];
mEulerAngles = ...
            [vPhiThetaPsi(:, 3), vPhiThetaPsi(:, 2), vPhiThetaPsi(:, 1)];

%% Settings
plotOptions.samples = 1:50:numel(vTime);
plotOptions.theView = [105, 15];
plotOptions.bodyAxes.show = true;
plotOptions.bodyAxes.magX = 1.5 * shapeScaleFactor;
plotOptions.bodyAxes.magY = 2 * shapeScaleFactor;
plotOptions.bodyAxes.magZ = 2 * shapeScaleFactor;
plotOptions.bodyAxes.lineWidth = 2.5;
plotOptions.helperLines.show = true;
plotOptions.helperLines.lineColor = 'k';
plotOptions.helperLines.lineStyle = ':';
plotOptions.helperLines.lineWidth = 1.5;
plotOptions.trajectory.show = true;
plotOptions.trajectory.lineColor = 'k';
plotOptions.trajectory.lineStyle = '-';
plotOptions.trajectory.lineWidth = 1.5;

%% Plot body and trajectory
plotTrajectoryAndBodyE(h_fig1, shape, mXYZe, mEulerAngles, plotOptions);

%% Plot Earth axes
hold on
xMax = max([max(abs(mXYZe(:, 1))), 5]);
yMax = max([max(abs(mXYZe(:, 2))), 5]);
zMax = .05 * xMax;
vXYZ0 = [0, 0, 0];
vExtent = [xMax, yMax, zMax];
plotEarthAxes(h_fig1, vXYZ0, vExtent);
xlabel('$x_\mathrm{E}$ (m)', 'interpreter', 'latex')
ylabel('$y_\mathrm{E}$ (m)', 'interpreter', 'latex')
zlabel('$z_\mathrm{E}$ (m)', 'interpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
hold off
exportgraphics(gca, 'ex25main.pdf', 'Resolution', 450)
```

1.2 Aircraft orientation using quaternions

From the *Euler's Theorem* it is known that the instantaneous orientation of a rigid body can always be described by only one angle of rotation μ about an axis known as *Euler axis*.

Defining the Euler axis upon its direction versor $\boldsymbol{e}$, it is possible to write the transformation of reference frames from $\mathscr{F}_\mathrm{E}$ to $\mathscr{F}_\mathrm{B}$ for any given vector $\boldsymbol{u}$ as

$$\{u\}_\mathrm{B} = [T(\boldsymbol{e}, \mu)]_\mathrm{BE}\{u\}_\mathrm{E} \tag{1.15}$$

where $[T(\boldsymbol{e}, \mu)]_\mathrm{BE}$ is $[T]_\mathrm{BE}$, but now it has been written with emphasis on its dependencies, that are formally different when using this parametrization instead of the Euler angles one.

In this case, the usage of four parameters instead of three requires us to consider an additional equation as a constraint. In particular, the magnitude of $\boldsymbol{e}$ is set to be 1, since it is a versor. Therefore, the following constraint is obtained

$$e_x^2 + e_y^2 + e_z^2 = 1 \tag{1.16}$$

To eliminate the presence of the singularity discussed in § 1.1.1, a new parametrization is introduced, i.e. the *Euler-Rodrigues parametrization*. It is defined as follows.

$$\begin{Bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{Bmatrix} = \begin{Bmatrix} \cos\dfrac{\mu}{2} \\ e_x \sin\dfrac{\mu}{2} \\ e_y \sin\dfrac{\mu}{2} \\ e_z \sin\dfrac{\mu}{2} \end{Bmatrix} \tag{1.17}$$

New parameters $\left[q_0,\ q_x,\ q_y,\ q_z\right]^\mathrm{T}$ can be seen as components of a quaternion defined as

$$\mathbb{q} = (q_0,\ q_x,\ q_y,\ q_z) = \left(\cos\frac{\mu}{2},\ \boldsymbol{e}\sin\frac{\mu}{2}\right) \tag{1.18}$$

With this parametrization the constraints in equation (1.16) now becomes

$$q_0^2 + q_x^2 + q_y^2 + q_z^2 = 1 \tag{1.19}$$

and the components transformation matrix can be written as

$$[T]_\mathrm{BE} = \begin{bmatrix} q_0^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y + q_0 q_z) & 2(q_z q_x - q_0 q_y) \\ 2(q_x q_y - q_0 q_z) & q_0^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z + q_0 q_x) \\ 2(q_z q_x + q_0 q_y) & 2(q_y q_z - q_0 q_x) & q_0^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{1.20}$$

1.2.1 Quaternion evolution equations

In order to solve for the aircraft orientation, equations of the time rates of change of the quaternion components are required. These equations are analogous to the inverse

gimbal equations. They can be written in terms of the angular velocity components and the qaternion components as

$$\begin{Bmatrix} \dot{q}_0 \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_0 & -q_z & q_y \\ q_z & q_0 & -q_x \\ -q_y & q_x & q_0 \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{1.21}$$

or in the following alternative form.

$$\begin{Bmatrix} \dot{q}_0 \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{Bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{Bmatrix} \tag{1.22}$$

1.2.2 From the quaternion to the Euler angles

The quaternion formulation of the orientation, using the Euler-Rodrigues parametrization, is far more convenient to use when integrating the equations of motion. Nevertheless, Euler angles are more suitable to visualize and eventually comprehend the motion of the aircraft from a practical perspective.

Finding a relation between quaternion components and Euler angles is therefore crucial. We can write the components of the quaternion in terms of the Euler angles as

$$\begin{Bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{Bmatrix} = \begin{Bmatrix} \cos\frac{\varphi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\varphi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \sin\frac{\varphi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\varphi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\varphi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\varphi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\varphi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\varphi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \end{Bmatrix} \tag{1.23}$$

that can be inverted to get

$$\begin{Bmatrix} \varphi \\ \theta \\ \psi \end{Bmatrix} = \begin{Bmatrix} \operatorname{atan2}\left[2(q_0q_x + q_yq_z), (q_0^2 - q_x^2 - q_y^2 + q_z^2)\right] \\ \arcsin\left[2(q_0q_y - q_xq_z)\right] \\ \operatorname{atan2}\left[2(q_0q_z + q_xq_y), (q_0^2 + q_x^2 - q_y^2 - q_z^2)\right] \end{Bmatrix} \tag{1.24}$$

from which it is worth noticing that in gimbal lock condition, when $\theta = \pm\pi/2$, Euler angles become

$$\begin{Bmatrix} \varphi \\ \theta \\ \psi \end{Bmatrix}_{\text{gimbal lock}} = \begin{Bmatrix} 2\arcsin\left(\sqrt{2}q_x\right) \pm \psi \\ \pm\frac{\pi}{2} \\ \text{arbitrary} \end{Bmatrix} \tag{1.25}$$

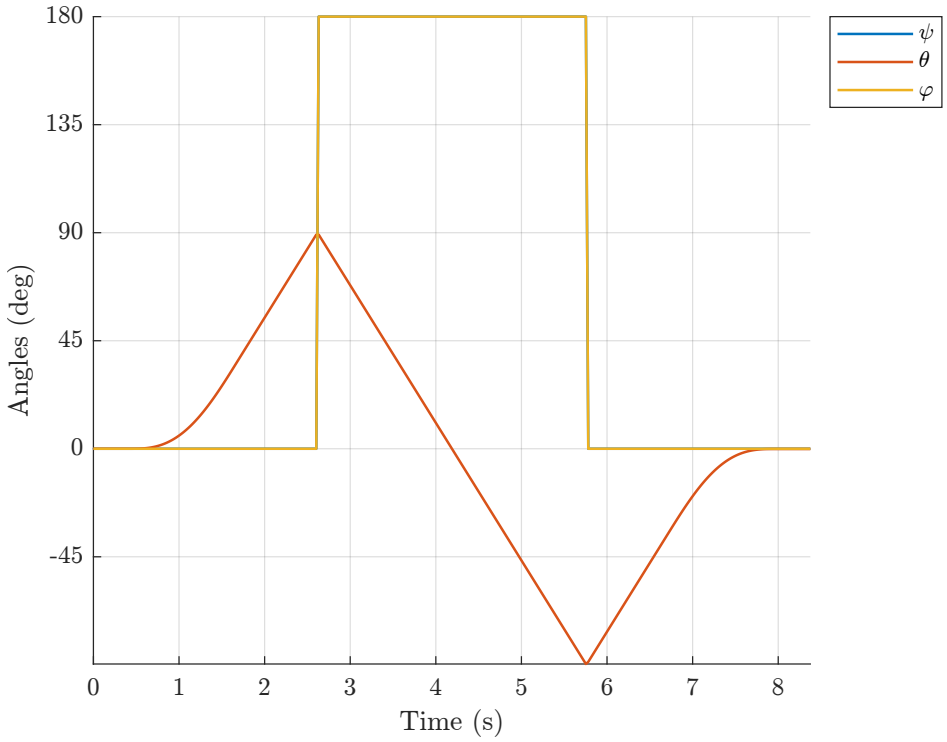


Figure 1.5 Time histories of the Euler angles during a looping maneuver.

1.2.3 Exercises

Kinematics of a looping maneuver

During a looping maneuver the gimbal lock condition $\theta = \pm\pi/2$ is reached two times. In this exercise the kinematics of such maneuver is completely solved using the quaternion formulation to show this approach in action in case of more complicated motion.

The time laws $p(t)$, $q(t)$ and $r(t)$ are assigned as follows. The initial conditions are given and the right hand side of equations in (1.22) is defined using an anonymous function.

Listing 1.6 Script that displays a looping maneuver kinematics integrating the kinematic equations.

```
clc; close all; clear

%% Time and pitch constraints
t_end = 8.375; % final time (s)
q_max = 1; % max pitch angular rate (rad/s)

%% p, q and r laws
p = @(t) 0;
q = @(t) interp1([0, 0.05, 0.2, 0.8, 0.95, 1] * t_end, ...
                 [0,  0,   1,   1,   0,  0] * q_max, ...
                     t, 'pchip');
r = @(t) 0;

%% Initial conditions
quat_0 = [1; 0; 0; 0];

%% ODE RHS
dquat_dt = @(t, quat) 0.5 * [ 0,   -p(t), -q(t), -r(t); ...
                              p(t),   0,    r(t), -q(t); ...
                              q(t), -r(t),   0,    p(t); ...
                              r(t),  q(t), -p(t),   0  ] * quat;

%% ODE solution
```

```
ODE_options = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);
[v_time, m_quat] = ode45(dquat_dt, [0, t_end], quat_0, ODE_options);

%% Euler angles
[v_psi, v_theta, v_phi] = quat2angle(m_quat);
v_psi_deg = convang(v_psi, 'rad', 'deg');
v_theta_deg = convang(v_theta, 'rad', 'deg');
v_phi_deg = convang(v_phi, 'rad', 'deg');

%% Plots
h_fig1 = figure(1);
hold on
plot(v_time, v_psi_deg, 'LineWidth', 1, 'LineStyle', '-');
plot(v_time, v_theta_deg, 'LineWidth', 1, 'LineStyle', '--');
plot(v_time, v_phi_deg, 'LineWidth', 1, 'LineStyle', '-');
hold off
grid on
xlabel("Time (s)", 'Interpreter', 'latex'); ylabel("Angles (deg)",
    ↪ 'Interpreter', 'latex')
%title("\textbf{Euler angles during a Looping}", 'Interpreter', 'latex')
legend({"$\psi$", "$\theta$", "$\varphi$"}, 'Interpreter', 'latex',
    ↪ 'Location', 'northeastoutside')
set(gca, 'TickLabelInterpreter', 'latex')
yticks([-90, -45, 0, 45, 90, 135, 180]); axis tight
exportgraphics(gca, 'ex31main.pdf', 'Resolution', 300)
```

The results of the numerical integration and conversions are finally shown in 1.5.

Trajectory of a looping maneuver

In this exercise the procedure of the listing 1.6 is repeated to display the trajectory of the looping maneuver.

In addition, the center of gravity velocity components are computed using the transformation matrix $[T]_{\mathrm{EB}} = [T]_{\mathrm{BE}}^{\mathrm{T}}$, where $[T]_{\mathrm{BE}}$ is defined as specified in equation (1.20) by using the `quat2dcm` function of the Aerospace Toolbox.

The aircraft center of gravity components are obtained by numerical integration of the velocity components in Earth axes. The trajectory is displayed resulting in figure 1.6 as output.

Listing 1.7 Script that displays a looping maneuver trajectory integrating the kinematic equations.

```
clc; close all; clear

%% Time and kinematics constraints
t_end = 8.375; % final time (s)
q_max = 1; % max pitch angular rate (rad/s)
u_0 = convvel(250, 'km/h', 'm/s');

%% p, q and r laws
p = @(t) 0;
q = @(t) interp1([0, 0.05, 0.2, 0.8, 0.95, 1] * t_end, ...
                 [0,  0,    1,   1,   0,   0] * q_max, ...
                     t, 'pchip');
r = @(t) 0;

%% u, v and w laws
u = @(t) interp1([0, 0.1,  0.2,  1  ] * t_end, ...
                 [1,  1,  0.75, 0.75] * u_0  , ...
                     t, 'pchip');
v = @(t) 0;
w = @(t) 0;

%% Initial conditions
quat_0 = [1; 0; 0; 0];

```

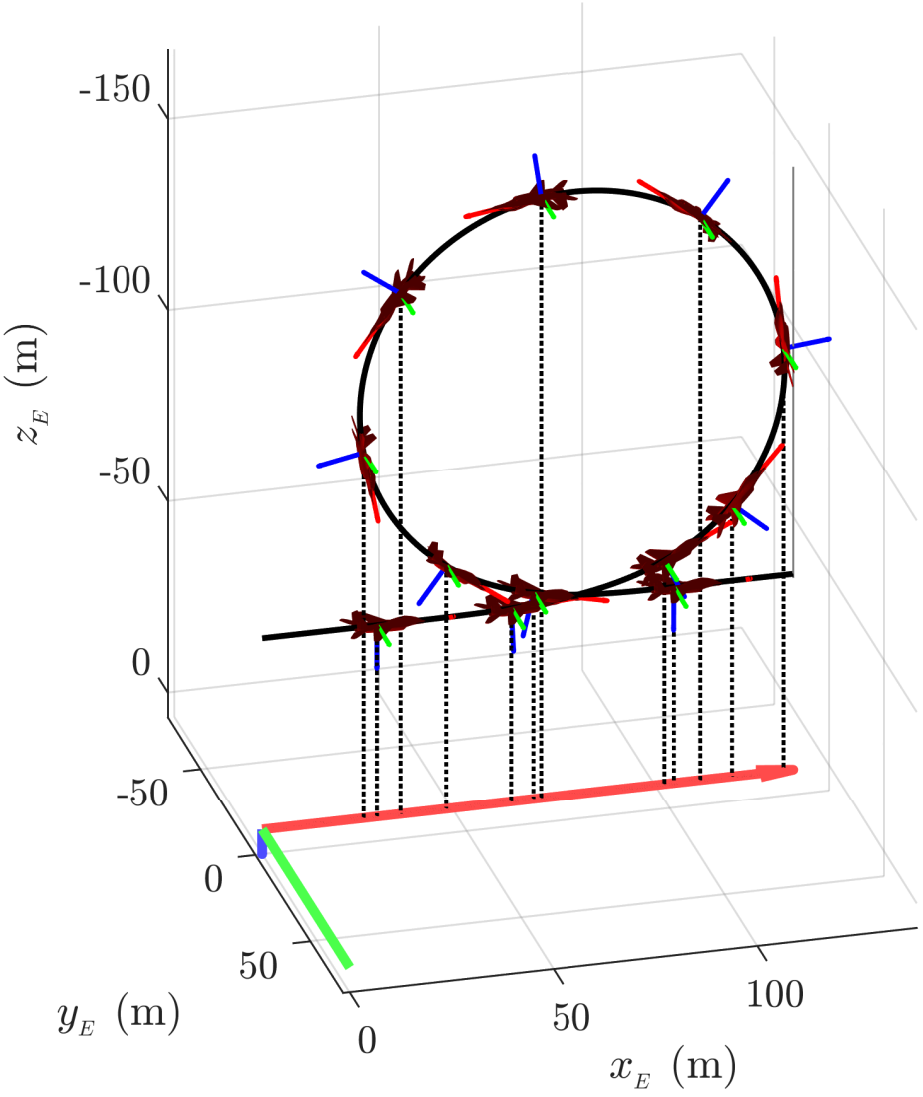


Figure 1.6 3D visualization of a looping maneuver.

```
%% ODE RHS
dquat_dt = @(t, quat) 0.5 * [ 0,   -p(t), -q(t), -r(t); ...
                              p(t),   0,    r(t), -q(t); ...
                              q(t), -r(t),   0,    p(t); ...
                              r(t),  q(t), -p(t),   0  ] * quat;

%% ODE solution
ODE_options = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);
[v_time, m_quat] = ode45(dquat_dt, [0, t_end], quat_0, ODE_options);

%% Euler angles
[v_psi, v_theta, v_phi] = quat2angle(m_quat);
v_psi_deg = convang(v_psi, 'rad', 'deg');
v_theta_deg = convang(v_theta, 'rad', 'deg');
v_phi_deg = convang(v_phi, 'rad', 'deg');

%% Velocity components in Earth FoR
T_EB = @(quat) quat2dcm(quat).';

v_u_E = zeros(length(v_time), 1);
v_v_E = zeros(length(v_time), 1);
v_w_E = zeros(length(v_time), 1);
for it = 1 : length(v_time)
    V_E = T_EB(m_quat(it, :)) * ...
                              [u(v_time(it)); v(v_time(it));
    ↪ w(v_time(it))];
```

```
    v_u_E(it) = V_E(1);
    v_v_E(it) = V_E(2);
    v_w_E(it) = V_E(3);
end

u_E = @(t) interp1(v_time, v_u_E, t, 'pchip');
v_E = @(t) interp1(v_time, v_v_E, t, 'pchip');
w_E = @(t) interp1(v_time, v_w_E, t, 'pchip');

%% CoG coordinates in Earth FoR
CoG_0 = [0, 0, -50];
dCoG_dt = @(t, CoG) [u_E(t); v_E(t); w_E(t)];
[~, m_CoG] = ode45(dCoG_dt, v_time, CoG_0, ODE_options);

v_x_EG = m_CoG(:, 1);
v_y_EG = m_CoG(:, 2);
v_z_EG = m_CoG(:, 3);

%% Trajectory
h_fig1 = figure(1);
grid on; hold on
light('Position', [1, 0, -4], 'Style', 'local');
set(gca, 'XDir', 'reverse'); set(gca, 'ZDir', 'reverse')
daspect([1, 1, 1])

% Load the A/C shape
shape_scale_factor = 15;
shape = loadAircraftMAT('aircraft_mig29.mat', shape_scale_factor);

% Sequence of positions and Euler angles
m_XYZ_EG = [v_x_EG, v_y_EG, v_z_EG];
m_euler_angles = [v_psi, v_theta, v_phi];

% Settings
plot3d_options.samples = 31 : 30 : length(v_time);
plot3d_options.theView = [165, 25];
plot3d_options.bodyAxes.show = true;
plot3d_options.bodyAxes.magX = 1.25 * shape_scale_factor;
plot3d_options.bodyAxes.magY = 0.75 * shape_scale_factor;
plot3d_options.bodyAxes.magZ = 0.75 * shape_scale_factor;
plot3d_options.bodyAxes.lineWidth = 1.25;
plot3d_options.helperLines.show = true;
plot3d_options.helperLines.lineColor = 'k';
plot3d_options.helperLines.lineStyle = ':';
plot3d_options.helperLines.lineWidth = 1;
plot3d_options.trajectory.show = true;
plot3d_options.trajectory.lineColor = 'k';
plot3d_options.trajectory.lineStyle = '-';
plot3d_options.trajectory.lineWidth = 1.5;

% Plot body and trajectory
plotTrajectoryAndBodyE(h_fig1, shape, ...
                                m_XYZ_EG, m_euler_angles,
    ↪ plot3d_options);

% Plot Earth axes
hold on
x_max = max([max(abs(m_XYZ_EG(:, 1))), 100]);
y_max = max([max(abs(m_XYZ_EG(:, 2))), 100]);
z_max = 0.05 * x_max;
v_XYZ_0 = [0, 0, 0];
v_extent = [x_max, y_max, z_max];
plotEarthAxes(h_fig1, v_XYZ_0, v_extent);
hold off

% Figure add-ons
ylim([-80, 80])
xlabel("$x_{_E}$ (m)", 'Interpreter', 'latex'); ylabel("$y_{_E}$ (m)",
    ↪ 'Interpreter', 'latex'); zlabel("$z_{_E}$ (m)", 'Interpreter',
    ↪ 'latex')
%title("\textbf{3D View of the trajectory}", 'Interpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
```

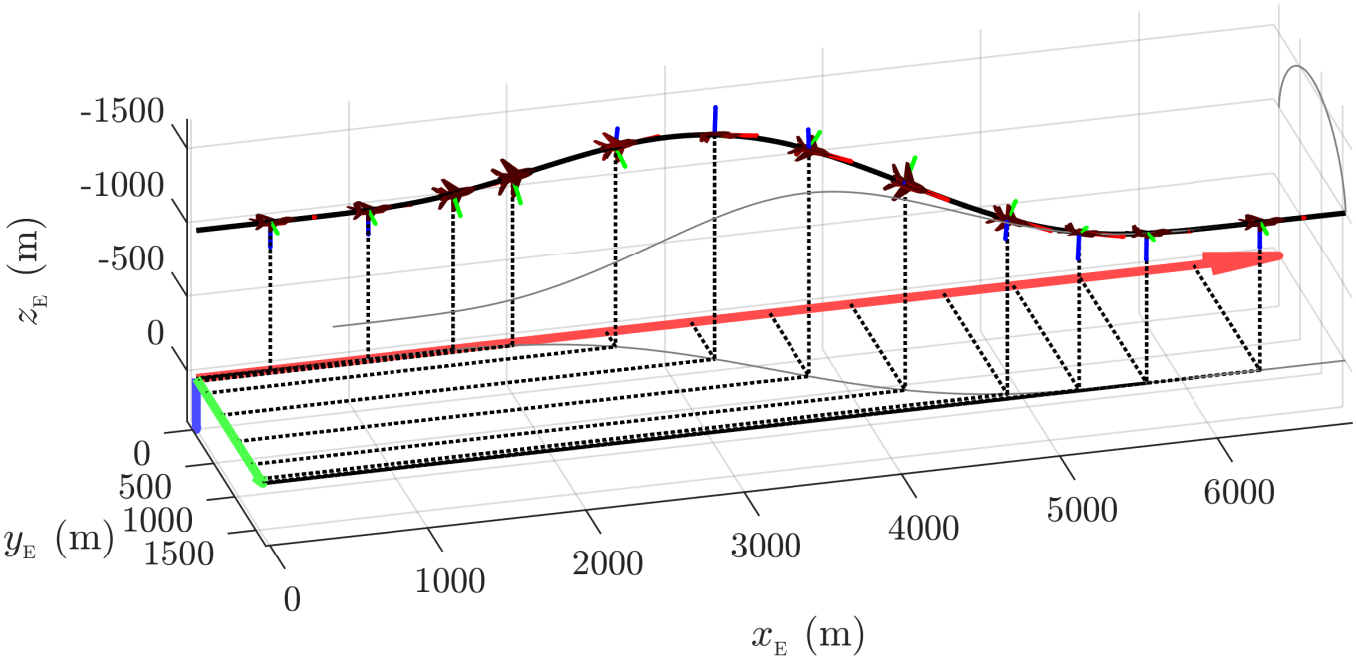


Figure 1.7 3D visualization of a tonneau maneuver.

```
119 exportgraphics(gca, 'ex32main.pdf', 'Resolution', 450)
```

Kinematics and trajectory of a tonneau maneuver

This exercise aims to apply the same procedure of the listing 1.7 to a more complex maneuver, such as a tonneau maneuver. The assigned nonzero time laws $p(t)$ and $q(t)$ are plotted in figure 1.8.

The kinematics is then solved for the Euler angles and the time histories of the Euler angles are plotted in figure 1.9. The resulting trajectory is shown in figure 1.7.

Listing 1.8 Script that solves the kinematics and the trajectory of a tonneau maneuver.

```
clc; close all; clear

%% Time and kinematics constraints
t_end = 90.5; % final time (s)
p_max = convangvel(5, 'deg/s', 'rad/s'); % max roll angular rate (rad/s)
q_max = convangvel(1.75, 'deg/s', 'rad/s'); % max pitch angular rate
    ↪ (rad/s)
u_0 = convvel(380, 'km/h', 'm/s');

%% p, q and r laws
p = @(t) interp1([0, 0.05, 0.2, 0.8, 0.95, 1] * t_end, ...
                 [0,  0,   1,   1,   0,  0] * p_max, ...
                     t, 'pchip');
q = @(t) interp1([0, 0.05, 0.2, 0.8, 0.95, 1] * t_end, ...
                 [0,  0,   1,   1,   0,  0] * q_max, ...
                     t, 'pchip');
r = @(t) 0;

figure
hold on
fplot(@(t) convangvel(p(t), 'rad/s', 'deg/s'), [0, t_end], 'LineWidth',
    ↪ 1)
fplot(@(t) convangvel(q(t), 'rad/s', 'deg/s'), [0, t_end], 'LineWidth',
    ↪ 1)
hold off
grid on
```

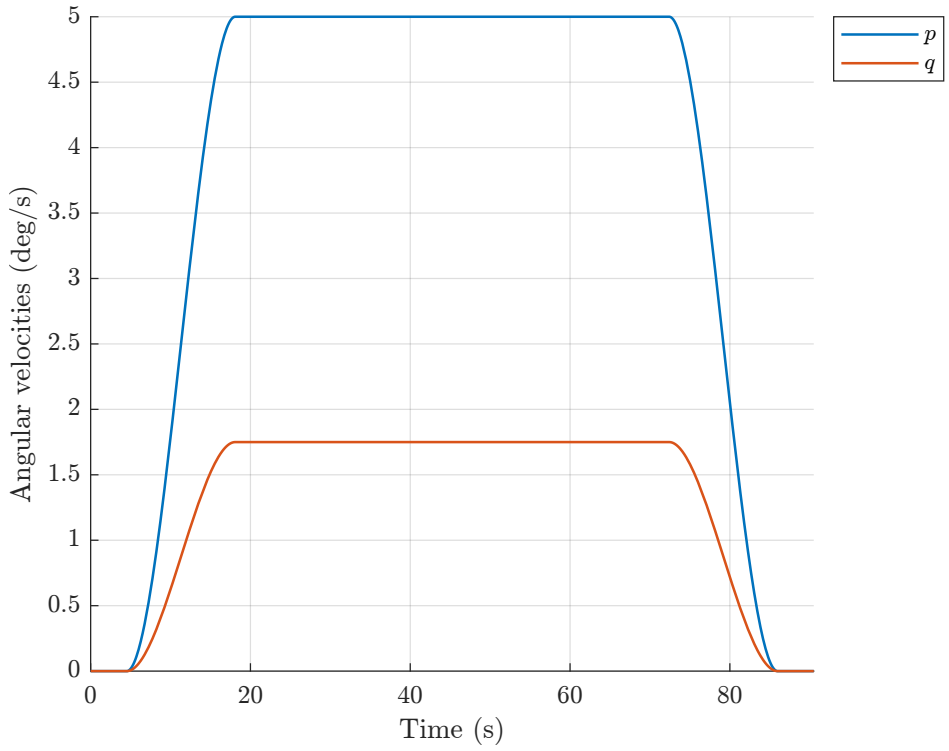


Figure 1.8 Angular velocity components assigned.

```
xlabel("Time (s)", 'Interpreter', 'latex'); ylabel("Angular velocities (deg/s)", 'Interpreter', 'latex')
legend("$p$", "$q$", 'Location', 'northeastoutside', 'Interpreter', 'latex')
axis tight
set(gca, 'TickLabelInterpreter', 'latex')
exportgraphics(gca, 'ex33main1.pdf', 'Resolution', 300)

%% u, v and w laws
u = @(t) interp1([0, 0.05, 0.2, 0.8, 0.95, 1] * t_end, ...
                 [1,  1,  0.7, 0.7, 1, 1] * u_0  , ...
                     t, 'pchip');
v = @(t) 0;
w = @(t) 0;

%% Initial conditions
quat_0 = angle2quat(0, 0, 0);

%% ODE RHS
dquat_dt = @(t, quat) 0.5 * [ 0,   -p(t), -q(t), -r(t); ...
                              p(t),   0,    r(t), -q(t); ...
                              q(t), -r(t),   0,    p(t); ...
                              r(t),  q(t), -p(t),   0  ] * quat;

%% ODE solution
ODE_options = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);
[v_time, m_quat] = ode45(dquat_dt, [0, t_end], quat_0, ODE_options);

%% Euler angles
[v_psi, v_theta, v_phi] = quat2angle(m_quat);
v_psi_deg = convang(v_psi, 'rad', 'deg');
v_theta_deg = convang(v_theta, 'rad', 'deg');
v_phi_deg = convang(v_phi, 'rad', 'deg');

%% Plots
figure
hold on
plot(v_time, v_psi_deg, 'LineWidth', 1, 'LineStyle', '-');
plot(v_time, v_theta_deg, 'LineWidth', 1, 'LineStyle', '--');
plot(v_time, v_phi_deg, 'LineWidth', 1, 'LineStyle', '-');
```

```
hold off
grid on
xlabel("Time (s)", 'Interpreter', 'latex'); ylabel("Angles (deg)", 'Interpreter', 'latex')
%title("\textbf{Euler angles during a Looping}", 'Interpreter', 'latex')
legend({"$\psi$", "$\theta$", "$\varphi$"}, 'Interpreter', 'latex', 'Location', 'northeastoutside')
set(gca, 'TickLabelInterpreter', 'latex')
yticks([-90, -45, 0, 45, 90, 135, 180])
axis tight
exportgraphics(gca, 'ex33main2.pdf', 'Resolution', 300)

%% Velocity components in Earth FoR
T_EB = @(quat) quat2dcm(quat).';

v_u_E = zeros(length(v_time), 1);
v_v_E = zeros(length(v_time), 1);
v_w_E = zeros(length(v_time), 1);
for it = 1 : length(v_time)
    V_E = T_EB(m_quat(it, :)) * [u(v_time(it)); v(v_time(it)); w(v_time(it))];
    v_u_E(it) = V_E(1);
    v_v_E(it) = V_E(2);
    v_w_E(it) = V_E(3);
end

u_E = @(t) interp1(v_time, v_u_E, t, 'pchip');
v_E = @(t) interp1(v_time, v_v_E, t, 'pchip');
w_E = @(t) interp1(v_time, v_w_E, t, 'pchip');

%% CoG coordinates in Earth FoR
CoG_0 = [0, 0, -1000];
dCoG_dt = @(t, CoG) [u_E(t); v_E(t); w_E(t)];
[~, m_CoG] = ode45(dCoG_dt, v_time, CoG_0, ODE_options);

v_x_EG = m_CoG(:, 1);
v_y_EG = m_CoG(:, 2);
v_z_EG = m_CoG(:, 3);

%% Trajectory
figure
grid on; hold on
light('Position', [1, 0, -4], 'Style', 'local');
set(gca, 'XDir', 'reverse'); set(gca, 'ZDir', 'reverse')
daspect([1, 1, 1])

% Load the A/C shape
shape_scale_factor = 225;
shape = loadAircraftMAT('aircraft_mig29.mat', shape_scale_factor);

% Sequence of positions and Euler angles
m_XYZ_EG = [v_x_EG, v_y_EG, v_z_EG];
m_euler_angles = [v_psi, v_theta, v_phi];

% Settings
plot3d_options.samples = 31 : 30 : length(v_time);
plot3d_options.theView = [165, 25];
plot3d_options.bodyAxes.show = true;
plot3d_options.bodyAxes.magX = 1.25 * shape_scale_factor;
plot3d_options.bodyAxes.magY = 0.75 * shape_scale_factor;
plot3d_options.bodyAxes.magZ = 0.75 * shape_scale_factor;
plot3d_options.bodyAxes.lineWidth = 1.25;
plot3d_options.helperLines.show = true;
plot3d_options.helperLines.lineColor = 'k';
plot3d_options.helperLines.lineStyle = ':';
plot3d_options.helperLines.lineWidth = 1;
plot3d_options.trajectory.show = true;
plot3d_options.trajectory.lineColor = 'k';
plot3d_options.trajectory.lineStyle = '-';
plot3d_options.trajectory.lineWidth = 1.5;

% Plot body and trajectory
```

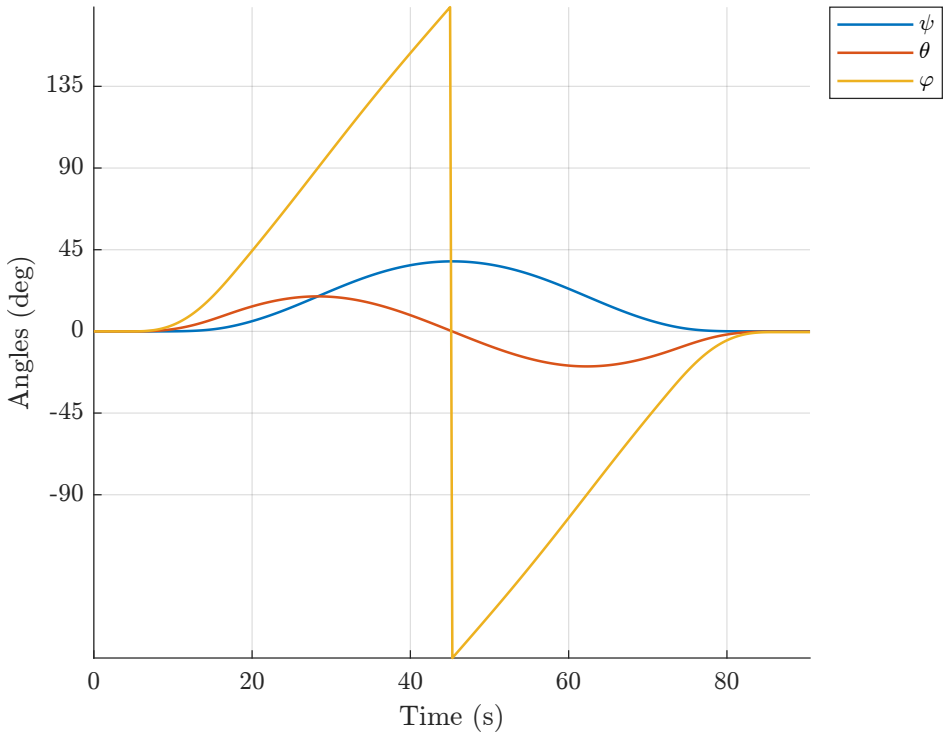


Figure 1.9 Time histories of the Euler angles during a tonneau maneuver.

```
plotTrajectoryAndBodyE(gcf, shape, ...
                                m_XYZ_EG, m_euler_angles,
    ↪ plot3d_options);

% Plot Earth axes
hold on
x_max = max([max(abs(m_XYZ_EG(:, 1))), 100]);
y_max = max([max(abs(m_XYZ_EG(:, 2))), 100]);
z_max = 0.05 * x_max;
v_XYZ_0 = [0, 0, 0];
v_extent = [x_max, y_max, z_max];
plotEarthAxes(gcf, v_XYZ_0, v_extent);
hold off

% Figure add-ons
% ylim([-80, 80])
xlabel("$x_{_\mathrm{E}}$ (m)", 'Interpreter', 'latex');
hyL = ylabel("$y_{_\mathrm{E}}$ (m)", 'Interpreter', 'latex');
    ↪ hyL.Position = hyL.Position + [-450, -2250, 0.];
zlabel("$z_{_\mathrm{E}}$ (m)", 'Interpreter', 'latex')
%title("\textbf{3D View of the trajectory}", 'Interpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
exportgraphics(gca, 'ex33main3.pdf', 'Resolution', 450)
```

Kinematics and trajectory of a lazy eight maneuver

In this exercise is done the same as the two previous ones, but for another acrobatic maneuver.

It starts assigning the time laws of the angular velocity components $p(t)$, $q(t)$ and $r(t)$. The diagram in figure 1.11 shows them. The time laws of the center of gravity velocity components $u(t)$, $v(t)$ and $w(t)$ are assigned as well in the following.

With the same approach as in listings 1.6, 1.7 and 1.8, the evolution equations (1.22) can be integrated using the MATLAB ode45 routine.

Euler angles are obtained from the quaternion components. Their time histories are

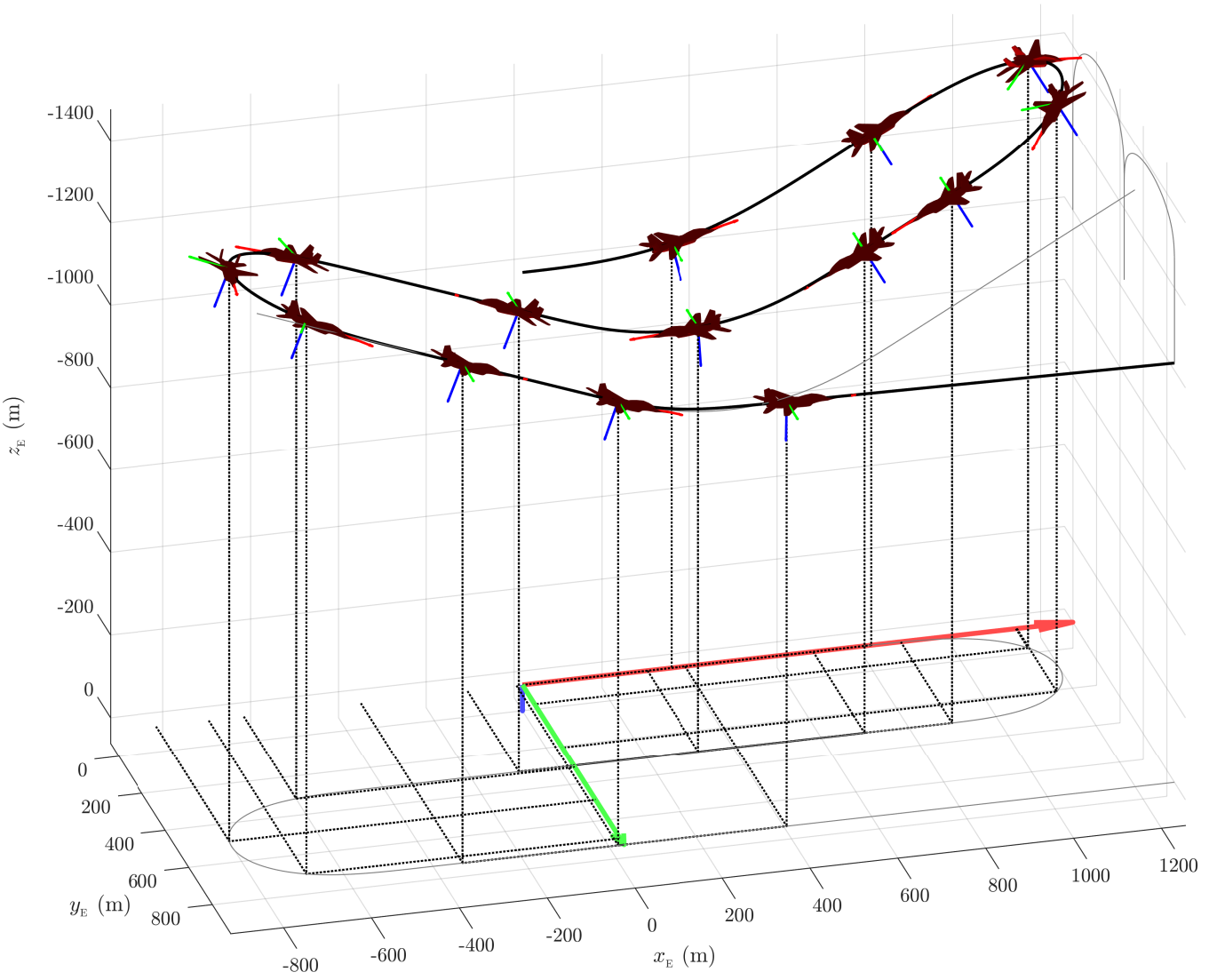


Figure 1.10 3D visualization of a lazy eight maneuver.

shown in figure 1.12.

The velocity components in Earth axes are computed to be integrated to get the center of gravity coordinates as functions of time. The trajectory is plotted. Results shown in figure 1.10.

Listing 1.9 Script that solves the kinematics and the trajectory of a lazy eight maneuver.

```
clc; close all; clear

%% Time and kinematics constraints
t_end = 100; % final time (s)
p_max = convangvel(5, 'deg/s', 'rad/s'); % max roll angular rate (rad/s)
q_max = convangvel(5, 'deg/s', 'rad/s'); % max pitch angular rate (rad/s)
r_max = convangvel(18, 'deg/s', 'rad/s');
u_0 = convvel(280, 'km/h', 'm/s');

%% p, q and r laws
p = @(t) interp1([0, 0.1, 0.15, 0.18, 0.23, 1] * t_end, ...
                 [0,  0,   0,    0,   0,  0] * p_max, ...
                 t, 'pchip');
q = @(t) interp1([0, 0.05, 0.1, 0.25, 0.3, 0.325, 0.375, 0.425, 0.475,
    ↪ 0.8, 0.84, 0.88, 1] * t_end, ...
                 [0,   1,   0,    0,   0,    0,    1,     1,      0,   0,
    ↪    1,   0,  0] * q_max, ...
```

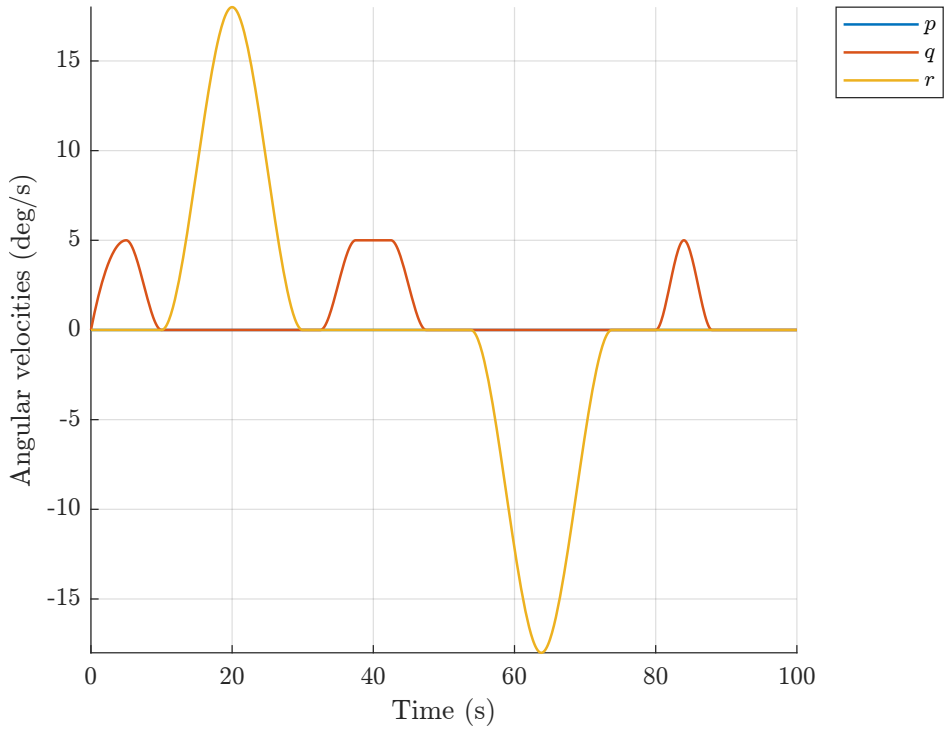


Figure 1.11 Time laws of the angular velocity components.

```
                    t, 'pchip');
r = @(t) interp1([0, 0.1, 0.2, 0.3, 0.35, 0.538, 0.638, 0.738,  1] *
    ↪ t_end, ...
                [0,   0,  1,   0,    0,    0,    -1,       0,  0] *
    ↪ r_max, ...
                t, 'pchip');

figure
hold on
fplot(@(t) convangvel(p(t), 'rad/s', 'deg/s'), [0, t_end], 'LineWidth',
    ↪ 1)
fplot(@(t) convangvel(q(t), 'rad/s', 'deg/s'), [0, t_end], 'LineWidth',
    ↪ 1)
fplot(@(t) convangvel(r(t), 'rad/s', 'deg/s'), [0, t_end], 'LineWidth',
    ↪ 1)
hold off
grid on
xlabel("Time (s)", 'Interpreter', 'latex'); ylabel("Angular velocities
    ↪ (deg/s)", 'Interpreter', 'latex')
legend("$p$", "$q$", "$r$", 'Location', 'northeastoutside',
    ↪ 'Interpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
axis tight
exportgraphics(gca, 'ex34main1.pdf', 'Resolution', 300)

%% u, v and w laws
u = @(t) interp1([0, 0.05, 0.2, 0.8, 0.95, 1] * t_end, ...
                 [1,  1,  0.7, 0.7, 1, 1] * u_0  , ...
                     t, 'pchip');
v = @(t) 0;
w = @(t) 0;

%% Initial conditions
quat_0 = angle2quat(0, 0, 0);

%% ODE RHS
dquat_dt = @(t, quat) 0.5 * [ 0,   -p(t), -q(t), -r(t); ...
                              p(t),   0,    r(t), -q(t); ...
                              q(t), -r(t),   0,    p(t); ...
                              r(t),  q(t), -p(t),   0  ] * quat;

```

```
%% ODE solution
ODE_options = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);
[v_time, m_quat] = ode45(dquat_dt, [0, t_end], quat_0, ODE_options);

%% Euler angles
[v_psi, v_theta, v_phi] = quat2angle(m_quat);
v_psi_deg = convang(v_psi, 'rad', 'deg');
v_theta_deg = convang(v_theta, 'rad', 'deg');
v_phi_deg = convang(v_phi, 'rad', 'deg');

%% Plots
figure
hold on
plot(v_time, v_psi_deg, 'LineWidth', 1, 'LineStyle', '-');
plot(v_time, v_theta_deg, 'LineWidth', 1, 'LineStyle', '-');
plot(v_time, v_phi_deg, 'LineWidth', 1, 'LineStyle', '-');
hold off
grid on
xlabel("Time (s)", 'Interpreter', 'latex'); ylabel("Angles (deg)",
    ↪ 'Interpreter', 'latex')
% title("\textbf{Euler angles during a Looping}", 'Interpreter', 'latex')
legend({"$\psi$ (deg)", "$\theta$ (deg)", "$\varphi$ (deg)"},
    ↪ 'Interpreter', 'latex', 'Location', 'northeastoutside')
set(gca, 'TickLabelInterpreter', 'latex')
yticks([-90, -45, 0, 45, 90, 135, 180])
axis tight
exportgraphics(gca, 'ex34main2.pdf', 'Resolution', 300)

%% Velocity components in Earth FoR
T_EB = @(quat) quat2dcm(quat).';

v_u_E = zeros(length(v_time), 1);
v_v_E = zeros(length(v_time), 1);
v_w_E = zeros(length(v_time), 1);
for it = 1 : length(v_time)
    V_E = T_EB(m_quat(it, :)) * [u(v_time(it)); v(v_time(it));
    ↪ w(v_time(it))];
    v_u_E(it) = V_E(1);
    v_v_E(it) = V_E(2);
    v_w_E(it) = V_E(3);
end

u_E = @(t) interp1(v_time, v_u_E, t, 'pchip');
v_E = @(t) interp1(v_time, v_v_E, t, 'pchip');
w_E = @(t) interp1(v_time, v_w_E, t, 'pchip');

%% CoG coordinates in Earth FoR
CoG_0 = [0, 0, -1000];
dCoG_dt = @(t, CoG) [u_E(t); v_E(t); w_E(t)];
[~, m_CoG] = ode45(dCoG_dt, v_time, CoG_0, ODE_options);

v_x_EG = m_CoG(:, 1);
v_y_EG = m_CoG(:, 2);
v_z_EG = m_CoG(:, 3);

%% Trajectory
h = figure; set(h, 'Position', 0.9*get(0, 'Screensize'))
grid on; hold on
light('Position', [1, 0, -4], 'Style', 'local');
set(gca, 'XDir', 'reverse'); set(gca, 'ZDir', 'reverse')
daspect([1, 1, 1])

% Load the A/C shape
shape_scale_factor = 125;
shape = loadAircraftMAT('aircraft_mig29.mat', shape_scale_factor);

% Sequence of positions and Euler angles
m_XYZ_EG = [v_x_EG, v_y_EG, v_z_EG];
m_euler_angles = [v_psi, v_theta, v_phi];

% Settings
plot3d_options.samples = 31 : 60 : length(v_time);
```

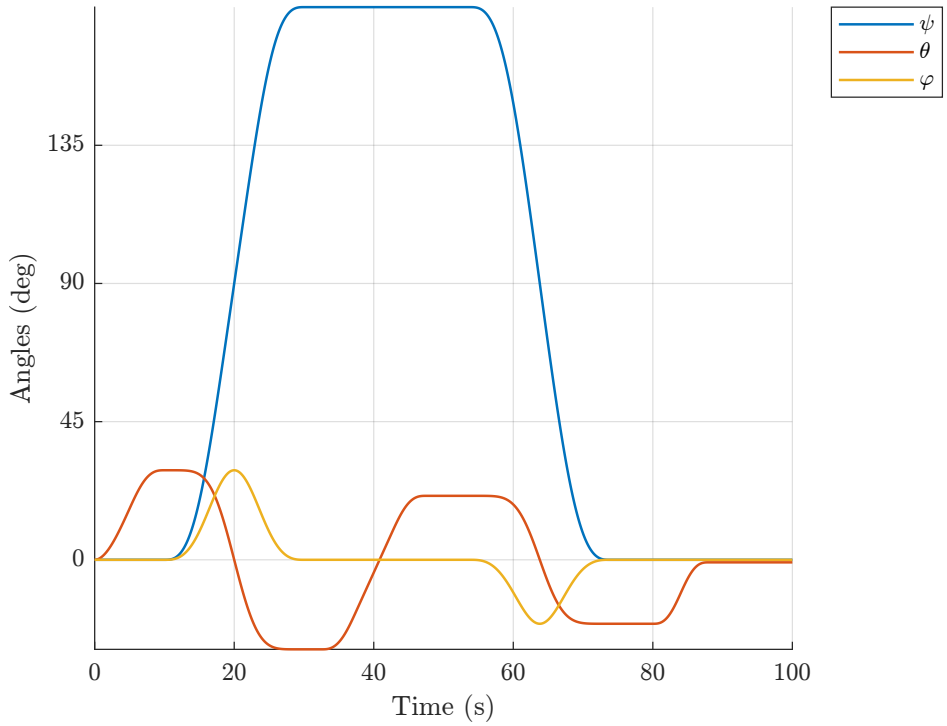


Figure 1.12 Time histories of the Euler angles during the maneuver.

```
plot3d_options.theView = [165, 25];
plot3d_options.bodyAxes.show = true;
plot3d_options.bodyAxes.magX = 1.25 * shape_scale_factor;
plot3d_options.bodyAxes.magY = 0.75 * shape_scale_factor;
plot3d_options.bodyAxes.magZ = 0.75 * shape_scale_factor;
plot3d_options.bodyAxes.lineWidth = 1.25;
plot3d_options.helperLines.show = true;
plot3d_options.helperLines.lineColor = 'k';
plot3d_options.helperLines.lineStyle = ':';
plot3d_options.helperLines.lineWidth = 1;
plot3d_options.trajectory.show = true;
plot3d_options.trajectory.lineColor = 'k';
plot3d_options.trajectory.lineStyle = '-';
plot3d_options.trajectory.lineWidth = 1.5;

% Plot body and trajectory
plotTrajectoryAndBodyE(gcf, shape, m_XYZ_EG, m_euler_angles,
    ↪ plot3d_options);

% Plot Earth axes
hold on
x_max = max([max(abs(m_XYZ_EG(:, 1))), 100]);
y_max = max([max(abs(m_XYZ_EG(:, 2))), 100]);
z_max = 0.05 * x_max;
v_XYZ_0 = [0, 0, 0];
v_extent = [x_max, y_max, z_max];
plotEarthAxes(gcf, v_XYZ_0, v_extent);
hold off

% Figure add-ons
% ylim([-80, 80])
hxL = xlabel("$x_{_\mathrm{E}}$ (m)", 'Interpreter', 'latex');
    ↪ hxL.Position = hxL.Position + [0, -200, 0];
hyL = ylabel("$y_{_\mathrm{E}}$ (m)", 'Interpreter', 'latex');
    ↪ hyL.Position = hyL.Position + [0, -600, 0];
zlabel("$z_{_\mathrm{E}}$ (m)", 'Interpreter', 'latex')
% title("\textbf{3D View of the trajectory}", 'Interpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
exportgraphics(gca, 'ex34main3.pdf', 'Resolution', 450)
```

Chapter 2

Dynamics

A review of the dynamics equations used to study aircrafts motion will be presented in this section. Some exercises will follow to show how such equations can be implemented in MATLAB to solve and simulate, given some initial conditions and input commands.

Theory and application discussed will cover both 6-DoF (six degrees of freedom) and 3-DoF (three degrees of freedom) motion dynamical models.

2.1 Relative rigid motions

Consider two frames of reference $\mathscr{F}_\mathrm{f} = \{O,\ x_\mathrm{f},\ y_\mathrm{f},\ z_\mathrm{f}\}$ and $\mathscr{F}_\mathrm{m} = \{C,\ x_\mathrm{m},\ y_\mathrm{m},\ z_\mathrm{m}\}$. We define $\mathscr{F}_\mathrm{f}$ to be fixed and $\mathscr{F}_\mathrm{m}$ to be mobile, subjected to a rigid motion. Its kinematics can be completely described by $C(t) \equiv C(t) - O$, $\boldsymbol{V}_C(t)$ and $\boldsymbol{\Omega}(t)$.

The mobile reference frame is assumed to characterize a motion of a rigid body $\mathscr{B}$. Therefore, given an arbitrary point $P \in \mathscr{B}$, the following relation holds

$$P(t) - C(t) = x_\mathrm{m}\boldsymbol{i}_\mathrm{m}(t) + y_\mathrm{m}\boldsymbol{j}_\mathrm{m}(t) + z_\mathrm{m}\boldsymbol{k}_\mathrm{m}(t) \tag{2.1}$$

where it is worth noticing that the coordinates of P in the mobile axes are not time-dependent, since $\mathscr{B}$ is rigid, so its *relative motion* is said to be zero. On the other hand, mobile axes versors $\boldsymbol{i}_\mathrm{m}$, $\boldsymbol{j}_\mathrm{m}$ and $\boldsymbol{k}_\mathrm{m}$ depends on time and they describe the instantaneous orientation of $\mathscr{B}$ with respect to $\mathscr{F}_\mathrm{f}$.

Furthermore, *absolute motion* of P (with respect to $\mathscr{F}_\mathrm{f}$) can be mathematically identified by the vector $P(t) \equiv P(t) - O$, given by

$$P(t) = x_\mathrm{f}(t)\boldsymbol{i}_\mathrm{f} + y_\mathrm{f}(t)\boldsymbol{j}_\mathrm{f} + z_\mathrm{f}(t)\boldsymbol{k}_\mathrm{f} \tag{2.2}$$

In addition, we can write

$$P(t) = C(t) - O + P(t) - C(t) = C(t) + x_\mathrm{m}\boldsymbol{i}_\mathrm{m}(t) + y_\mathrm{m}\boldsymbol{j}_\mathrm{m}(t) + z_\mathrm{m}\boldsymbol{k}_\mathrm{m}(t) \tag{2.3}$$

from which we notice that the position of any point of a rigid body can be instantaneous determined by knowing the position of the origin of $\mathscr{F}_\mathrm{m}$, hence $\boldsymbol{V}_C$ and the orientation of its axes versor only, hence $\boldsymbol{\Omega}$.

Using a more general approach, we can consider an arbitrary vector $\boldsymbol{u}$, such that it

can be written as $\boldsymbol{u}(t) = u_{x_\mathrm{m}}(t)\boldsymbol{i}_\mathrm{m}(t) + u_{y_\mathrm{m}}(t)\boldsymbol{j}_\mathrm{m}(t) + u_{z_\mathrm{m}}(t)\boldsymbol{k}_\mathrm{m}(t) = u_{x_\mathrm{f}}(t)\boldsymbol{i}_\mathrm{f} + u_{y_\mathrm{f}}(t)\boldsymbol{j}_\mathrm{f} + u_{z_\mathrm{f}}(t)\boldsymbol{k}_\mathrm{f}$. We can now define the derivative of such vector with respect to the mobile frame of reference as follows,

$$\frac{\partial \boldsymbol{u}}{\partial t} = \dot{u}_{x_\mathrm{m}}\boldsymbol{i}_\mathrm{m} + \dot{u}_{y_\mathrm{m}}\boldsymbol{j}_\mathrm{m} + \dot{u}_{z_\mathrm{m}}\boldsymbol{k}_\mathrm{m} \tag{2.4}$$

and the derivative with respect to the fixed reference frame as

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \dot{u}_{x_\mathrm{f}}\boldsymbol{i}_\mathrm{f} + \dot{u}_{y_\mathrm{f}}\boldsymbol{j}_\mathrm{f} + \dot{u}_{z_\mathrm{f}}\boldsymbol{k}_\mathrm{f} \tag{2.5}$$

Keeping in mind that

$$\frac{\mathrm{d}\boldsymbol{i}_\mathrm{m}}{\mathrm{d}t} = \boldsymbol{\Omega} \times \boldsymbol{i}_\mathrm{m} \tag{2.6}$$

$$\frac{\mathrm{d}\boldsymbol{j}_\mathrm{m}}{\mathrm{d}t} = \boldsymbol{\Omega} \times \boldsymbol{j}_\mathrm{m} \tag{2.7}$$

$$\frac{\mathrm{d}\boldsymbol{k}_\mathrm{m}}{\mathrm{d}t} = \boldsymbol{\Omega} \times \boldsymbol{k}_\mathrm{m} \tag{2.8}$$

we can easily obtain the *Poisson formula*

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{\Omega} \times \boldsymbol{u} \tag{2.9}$$

that is valid for any given vector $\boldsymbol{u}$.

2.2 Equations of motion

Laws just described in § 2.1 will be applied to the aircraft body $\mathscr{B}$ motion, considering $\mathscr{F}_\mathrm{B}$ as the mobile frame of reference and $\mathscr{F}_\mathrm{E}$ as the fixed frame of reference.

In order to eventually reach the complete dynamical formulation, based upon *Newtonian mechanics*, few quantities must be defined, starting from the *center of gravity*, defined as

$$G \equiv G - O_\mathrm{E} = \frac{1}{m}\int_{\mathscr{B}} (P - O_\mathrm{E})\ \mathrm{d}m \tag{2.10}$$

Then, the *total linear momentum* of the body as

$$\boldsymbol{Q} = \int_{\mathscr{B}} \frac{\mathrm{d}P}{\mathrm{d}t}\,\mathrm{d}m \tag{2.11}$$

from which we can notice that, since

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}(P - G) + \frac{\mathrm{d}G}{\mathrm{d}t} \tag{2.12}$$

we obtain

$$\boldsymbol{Q} = \boldsymbol{Q}^{(C)} + \boldsymbol{Q}_C \tag{2.13}$$

where $\boldsymbol{Q}^{(C)}$ is the total linear momentum seen from a point $C \in \mathscr{B}$ and $\boldsymbol{Q}_C$ is the linear

momentum of the point C as if it was a material particle of mass m. In particular

$$\boldsymbol{Q}^{(C)} = \int_{\mathscr{B}} \frac{\mathrm{d}}{\mathrm{d}t}(P - G)\ \mathrm{d}m \tag{2.14}$$

$$\boldsymbol{Q}_C = m\frac{\mathrm{d}C}{\mathrm{d}t} = \frac{W}{g}\boldsymbol{V}_C \tag{2.15}$$

Lastly, the *total angular momentum* about G of the aircraft body is defined as

$$\boldsymbol{\mathscr{K}} = \int_{\mathscr{B}} (P - G) \times \frac{\mathrm{d}P}{\mathrm{d}t}\,\mathrm{d}m \tag{2.16}$$

From *Newton's law of motion* we obtain the following system of six differential equations

$$\int_{\mathscr{B}} \frac{\mathrm{d}^2P}{\mathrm{d}t^2}\,\mathrm{d}m = \boldsymbol{F} \tag{2.17}$$

$$\int_{\mathscr{B}} (P - G) \times \frac{\mathrm{d}^2P}{\mathrm{d}t^2}\,\mathrm{d}m = \boldsymbol{\mathscr{M}} \tag{2.18}$$

which results in the following alternative formulation known as *Euler's law of motion*

$$\frac{\mathrm{d}\boldsymbol{Q}}{\mathrm{d}t} = \boldsymbol{F} \tag{2.19}$$

$$\frac{\mathrm{d}\boldsymbol{\mathscr{K}}}{\mathrm{d}t} = \boldsymbol{\mathscr{M}} \tag{2.20}$$

By differentiate equation (2.3) for $C \equiv G$ to evaluate $\mathrm{d}P/\mathrm{d}t$ and then applying the Poisson's formula (2.9), we easily obtain

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \frac{\mathrm{d}G}{\mathrm{d}t} + \boldsymbol{\Omega}_\mathrm{B} \times (P - G) \tag{2.21}$$

so that

$$\boldsymbol{Q} = \boldsymbol{Q}_G = \frac{W}{g}\boldsymbol{V}_G \tag{2.22}$$

$$\boldsymbol{\mathscr{K}} = I_\mathrm{B}\boldsymbol{\Omega}_\mathrm{B} \tag{2.23}$$

that can be substituted into equations (2.19) and (2.20) to get

$$\frac{W}{g}\frac{\partial \boldsymbol{V}_G}{\partial t} + \frac{W}{g}\boldsymbol{\Omega}_\mathrm{B} \times \boldsymbol{V}_G = \boldsymbol{F} \tag{2.24}$$

$$I_\mathrm{B}\frac{\partial \boldsymbol{\Omega}_\mathrm{B}}{\partial t} + \boldsymbol{\Omega}_\mathrm{B} \times I_\mathrm{B}\boldsymbol{\Omega}_\mathrm{B} = \boldsymbol{\mathscr{M}} \tag{2.25}$$

2.3 6-DoF dynamical system

Rewriting equation (2.24) in Body axes gives us the following expression, written in terms of the center of gravity velocity components

$$\frac{W}{g}\begin{Bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{Bmatrix}+\frac{W}{g}\begin{bmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{bmatrix}\begin{Bmatrix}u\\ v\\ w\end{Bmatrix}=\begin{Bmatrix}X\\ Y\\ Z\end{Bmatrix} \tag{2.26}$$

Similarly, rewriting equation (2.25) in Body axes lets us obtain

$$\begin{bmatrix}I_{xx} & -I_{xy} & -I_{xz}\\ -I_{xy} & I_{yy} & -I_{yz}\\ -I_{xz} & -I_{xz} & I_{zz}\end{bmatrix}\begin{Bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{Bmatrix}+\begin{bmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{bmatrix}\begin{bmatrix}I_{xx} & -I_{xy} & -I_{xz}\\ -I_{xy} & I_{yy} & -I_{yz}\\ -I_{xz} & -I_{xz} & I_{zz}\end{bmatrix}\begin{Bmatrix}p\\ q\\ r\end{Bmatrix}=\begin{Bmatrix}\mathscr{L}\\ \mathscr{M}\\ \mathscr{N}\end{Bmatrix} \tag{2.27}$$

These two equations can be solved for $[\dot{u},\ \dot{v},\ \dot{w}]^{\mathrm{T}}$ and $[\dot{p},\ \dot{q},\ \dot{r}]^{\mathrm{T}}$ to get the following system of six differential equations for the unknowns $\boldsymbol{x}_{\mathrm{d}} = [u,\ v,\ w,\ p,\ q,\ r]^{\mathrm{T}}$

$$\begin{Bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{Bmatrix}=-\begin{bmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{bmatrix}\begin{Bmatrix}u\\ v\\ w\end{Bmatrix}+\frac{g}{W}\begin{Bmatrix}X\\ Y\\ Z\end{Bmatrix} \tag{2.28}$$

$$\begin{Bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{Bmatrix}=\begin{bmatrix}I_{xx} & -I_{xy} & -I_{xz}\\ -I_{xy} & I_{yy} & -I_{yz}\\ -I_{xz} & -I_{xz} & I_{zz}\end{bmatrix}^{-1}\left(-\begin{bmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{bmatrix}\begin{bmatrix}I_{xx} & -I_{xy} & -I_{xz}\\ -I_{xy} & I_{yy} & -I_{yz}\\ -I_{xz} & -I_{xz} & I_{zz}\end{bmatrix}\begin{Bmatrix}p\\ q\\ r\end{Bmatrix}+\begin{Bmatrix}\mathscr{L}\\ \mathscr{M}\\ \mathscr{N}\end{Bmatrix}\right) \tag{2.29}$$

that can be joined with the system of seven differential equations for the unknowns $\boldsymbol{x}_{\mathrm{k}}$ given by the inverse navigation equations (1.14) rewritten with $[T]_{\mathrm{BE}}$ in terms of quaternions and the quaternion evolution equations (1.22), where

$$\boldsymbol{x}_{\mathrm{k}} = \left[x_{\mathrm{E},G},\ y_{\mathrm{E},G},\ z_{\mathrm{E},G},\ q_0,\ q_x,\ q_y,\ q_z\right]^{\mathrm{T}} \tag{2.30}$$

This yields to the complete dynamical system, identified by the following *state propagation equation* written in the classical form $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u})$, where the *state vector* is

$$\boldsymbol{x} = \begin{Bmatrix}\boldsymbol{x}_{\mathrm{d}}\\ \boldsymbol{x}_{\mathrm{k}}\end{Bmatrix} \tag{2.31}$$

and the *input vector* is

$$\boldsymbol{u} = \begin{Bmatrix}\delta_{\mathrm{T}}\\ \delta_{\mathrm{e}}\\ \delta_{\mathrm{s}}\\ \delta_{\mathrm{a}}\\ \delta_{\mathrm{r}}\end{Bmatrix} \tag{2.32}$$

It is worth noticing that X, Y, Z, $\mathscr{L}$, $\mathscr{M}$ and $\mathscr{N}$ may in general depend on the state $\boldsymbol{x}$ and its derivative $\dot{\boldsymbol{x}}$. In that case, when the dependencies on $\dot{\boldsymbol{x}}$ can be linearly expressed,

$$
\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{x}_{\mathrm{E},G} \\ \dot{y}_{\mathrm{E},G} \\ \dot{z}_{\mathrm{E},G} \\ \dot{q}_0 \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{Bmatrix} = \begin{bmatrix} -\left[\tilde{\Omega}_{\mathrm{B}}\right]_{\mathrm{B}} & [O]_{3\times3} & [O]_{3\times3} & [O]_{3\times4} \\ [O]_{3\times3} & -[I_{\mathrm{B}}]_{\mathrm{B}}^{-1}\left[\tilde{\Omega}_{\mathrm{B}}\right]_{\mathrm{B}}[I_{\mathrm{B}}]_{\mathrm{B}} & [O]_{3\times3} & [O]_{3\times4} \\ [T]_{\mathrm{EB}} & [O]_{3\times3} & [O]_{3\times3} & [O]_{3\times4} \\ [O]_{4\times3} & [O]_{4\times3} & [O]_{4\times3} & [Q]_{\mathrm{evol}} \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \\ p \\ q \\ r \\ x_{\mathrm{E},G} \\ y_{\mathrm{E},G} \\ z_{\mathrm{E},G} \\ q_0 \\ q_x \\ q_y \\ q_z \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ 0 \\ [I_{\mathrm{B}}]_{\mathrm{B}}^{-1} \begin{Bmatrix} \mathscr{L} \\ \mathscr{M} \\ \mathscr{N} \end{Bmatrix} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \begin{Bmatrix} \frac{g}{W} \begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} \end{Bmatrix}
\tag{2.33}
$$

the following form of the state propagation equation is eventually assumed

$$M(t, \boldsymbol{x}; \boldsymbol{u})\, \dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u}) \tag{2.34}$$

where $M = M(t, \boldsymbol{x}; \boldsymbol{u})$ is the *generalized mass matrix* of the dynamical system.

The system of equations (2.33) identifies the *6-DoF dynamical model* of the aircraft, since it assumes complete translational and rotational rigid motion. When needed, it can be simplified to obtain a reduced dynamical system, namely the *3-DoF dynamical model* of the aircraft, that will be later presented (see § 2.4).

2.3.1 Exercises

F/A-18 HARV 6-DoF simulation

This exercise aims to show a six degrees of freedom simulation of the *F/A-18 HARV (High Alpha Research Vehicle)* [4]. To obtain such simulation a general method will be presented, together with some helper MATLAB functions I wrote. By changing some inputs of these functions other simulations can be easily carried out when needed.

The first MATLAB routine is `saveAircraftStruct.m` (see listing 2.1). It takes as input data from the aircraft its name, wing surface area, wingspan, mean aerodynamic chord, mass, inertia tensor and maximum thrust. To obtain a useful data structure of the aircraft a propulsion and aerodynamic models must be provided as MATLAB functions. In particular,

- The propulsion model `ThrustModel` must be a function of the *generalized inputs* $\boldsymbol{u}^*$ and of the aircraft geometry and mass parameters $\boldsymbol{p}$ and must give $T = T(\boldsymbol{u}^*, \boldsymbol{p})$.
- The aerodynamic models `LiftCoeffModel`, `DragCoeffModel`, `CrossforceCoeffModel`, `RollCoeffModel`, `PitchCoeffModel` and `YawCoeffModel` must be functions of the generalized inputs $\boldsymbol{u}^*$ and must give $C_L = C_L(\boldsymbol{u}^*)$, $C_D = C_D(\boldsymbol{u}^*)$, $C_{Y_\mathrm{A}} = C_{Y_\mathrm{A}}(\boldsymbol{u}^*)$, $C_{\mathscr{L}} = C_{\mathscr{L}}(\boldsymbol{u}^*)$, $C_{\mathscr{M}} = C_{\mathscr{M}}(\boldsymbol{u}^*)$ and $C_{\mathscr{N}} = C_{\mathscr{N}}(\boldsymbol{u}^*)$ respectively.

where

$$\boldsymbol{u}^* = [\alpha_\mathrm{B},\ \beta,\ \delta_\mathrm{T},\ \delta_\mathrm{e},\ \delta_\mathrm{s},\ \delta_\mathrm{a},\ \delta_\mathrm{r},\ p,\ q,\ r,\ h,\ V_G]^\mathrm{T} \tag{2.35}$$

Then, it saves a MATLAB struct named after the aircraft containing its data stored as variables and the assumed models as function handles.

Listing 2.1 Function for saving an aircraft struct.

```
function saveAircraftStruct(aircraftName, ...
                S, b, c, m, I_B, ...
                T_max, ThrustModel, ...
                LiftCoeffModel, DragCoeffModel, CrossforceCoeffModel, ...
                RollCoeffModel, PitchCoeffModel, YawCoeffModel)

    myAircraft.S = S;
    myAircraft.b = b;
    myAircraft.c = c;
    myAircraft.m = m;
    myAircraft.W = m * 9.81;
    myAircraft.I_B = I_B;
    myAircraft.T_max = T_max;
    myAircraft.ThrustModel = @(inputs) ThrustModel(inputs, myAircraft);
    myAircraft.DragCoeffModel = @(inputs) DragCoeffModel(inputs);
    myAircraft.LiftCoeffModel = @(inputs) LiftCoeffModel(inputs);
```

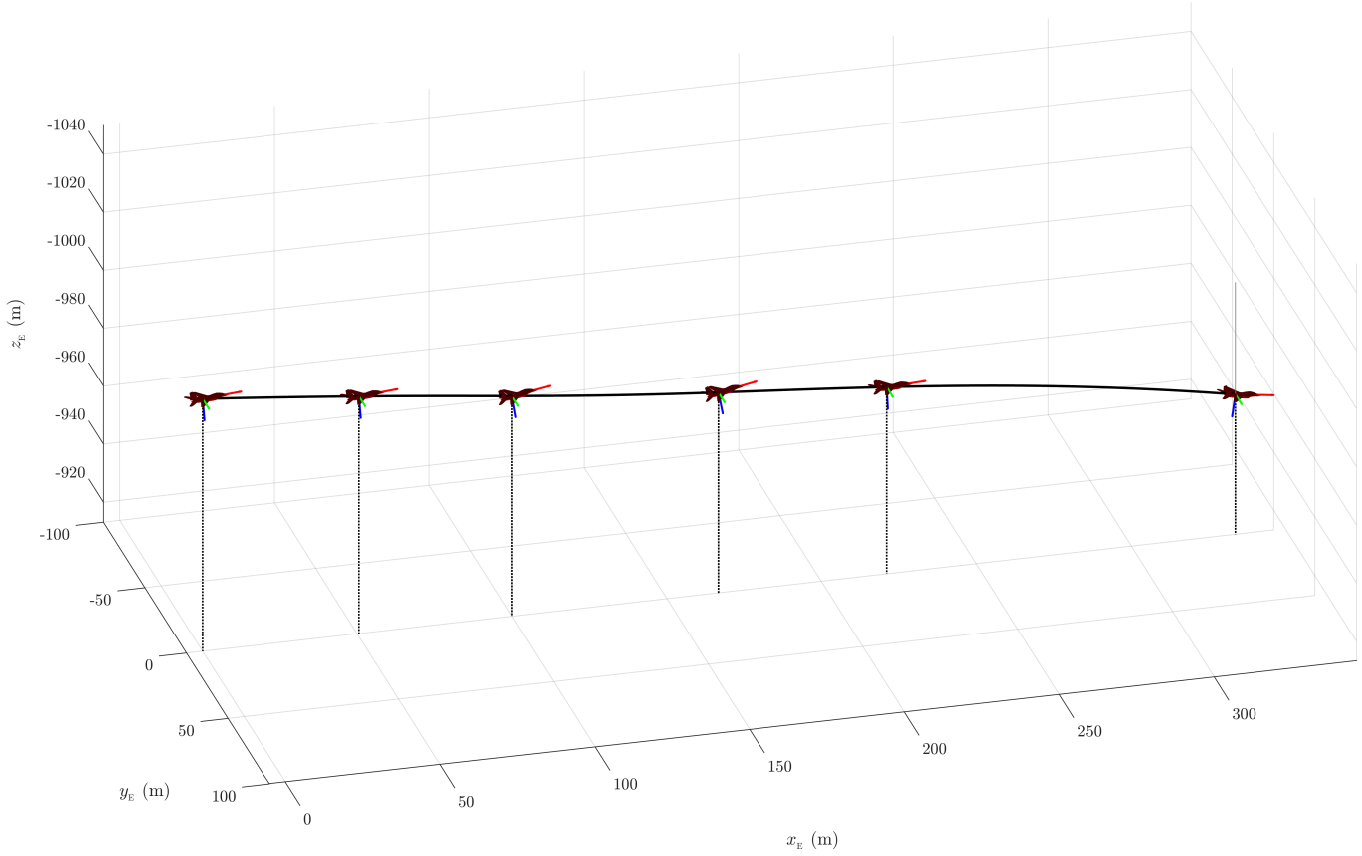


Figure 2.1 3D visualization of the trajectory obtained from the simulation.

```
    myAircraft.CrossforceCoeffModel = @(inputs)
    ↪ CrossforceCoeffModel(inputs);
    myAircraft.RollCoeffModel = @(inputs) RollCoeffModel(inputs);
    myAircraft.PitchCoeffModel = @(inputs) PitchCoeffModel(inputs);
    myAircraft.YawCoeffModel = @(inputs) YawCoeffModel(inputs);

    save(strcat(aircraftName, '.mat'), '-struct', 'myAircraft')

end
```

To successfully save the aircraft struct, the aerodynamic coefficients must be assumed, estimated or experimentally determined. For this aircraft, the following lift coefficient is taken. Its diagram is shown in figure 2.2 and its MATLAB definition is shown in listing 2.2.

$$C_L = \begin{cases} 0.732 + 0.0751\alpha_B + 0.0144\delta_e & -5 \text{ deg} \leq \alpha_B \leq 10 \text{ deg} \\ 0.569 + 0.106\alpha_B - 0.00148\alpha_B^2 + 0.0144\delta_e & 10 \text{ deg} < \alpha_B \leq 40 \text{ deg} \end{cases}$$

Listing 2.2 F/A-18 HARV lift coefficient aerodynamic model.

```
function CL = LiftCoeffModel(inputs)

   alpha_deg = inputs(1);
   beta_deg = inputs(2);
   delta_T = inputs(3);
   delta_e_deg = inputs(4);
   delta_s_deg = inputs(5);
   delta_a_deg = inputs(6);
   delta_r_deg = inputs(7);
   p_degps = inputs(8);
   q_degps = inputs(9);
   r_degps = inputs(10);
```

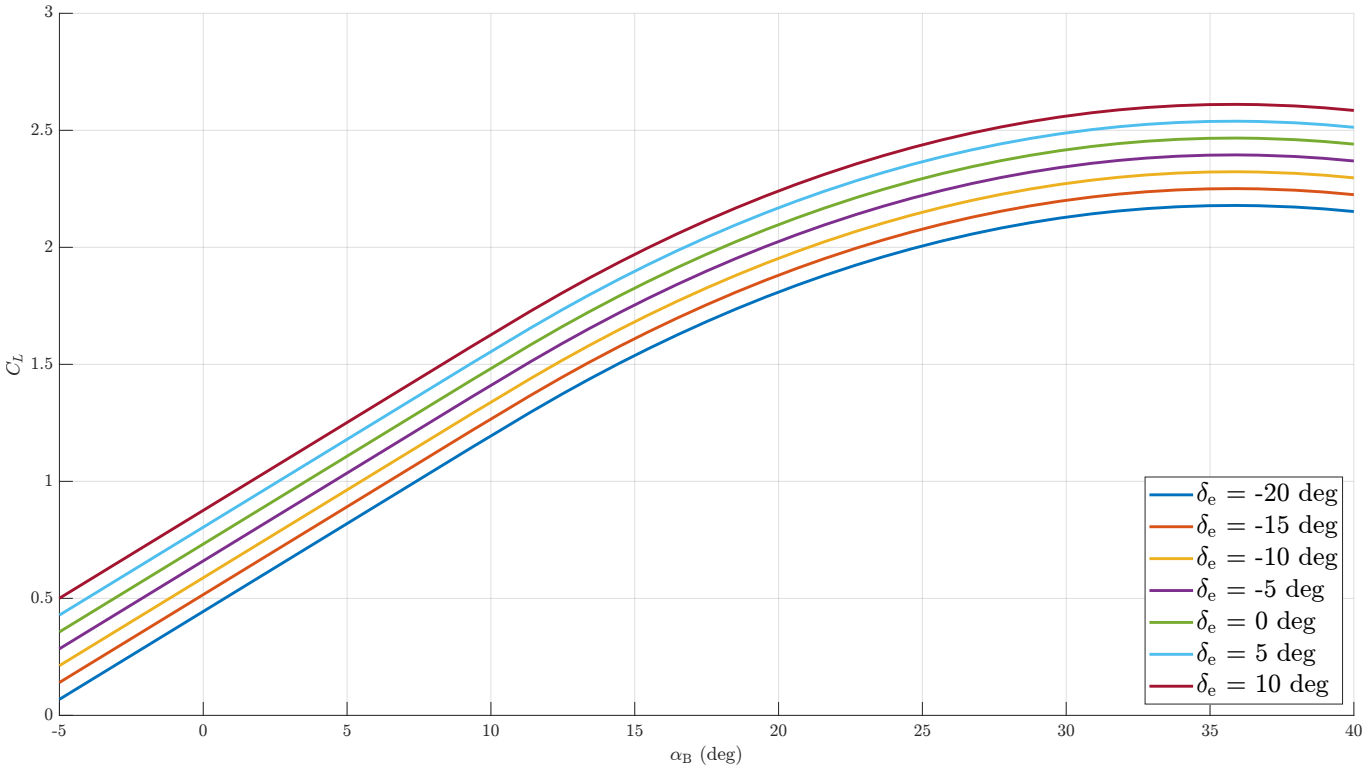


Figure 2.2 Lift coefficient curve of the aircraft.

```
    altitude = inputs(11);
    airspeed = inputs(12);

    if alpha_deg >= -5 && alpha_deg <= 10
        CL = 0.732 + 0.0751 * alpha_deg + 0.0144 * delta_e_deg;
    elseif alpha_deg > 10 && alpha_deg <= 40
        CL = 0.569 + 0.106 * alpha_deg - 0.00148 * alpha_deg^2 + 0.0144 *
    ↪ delta_e_deg;
    elseif alpha_deg < 5
        CL = 0.732 - 0.0751 * 5 + 0.0144 * delta_e_deg;
    else
        CL = 0.569 + 0.106 * 40 - 0.00148 * 40^2 + 0.0144 * delta_e_deg;
    end

end
```

In the same way, the drag coefficient is defined as in the following. Its diagram and aircraft polar is shown in figure 2.3. The code for C_D is in figure 2.3.

$$C_D = \begin{cases} 0.1426 - 0.00438\alpha_B + 0.0013\alpha_B^2 & -5 \text{ deg} \leq \alpha_B \leq 20 \text{ deg} \\ -0.358 + 0.0473\alpha_B - 0.0000348\alpha_B^2 & 20 \text{ deg} < \alpha_B \leq 40 \text{ deg} \end{cases}$$

Listing 2.3 F/A-18 HARV drag coefficient aerodynamic model.

```
function CD = DragCoeffModel(inputs)

    alpha_deg = inputs(1);
    beta_deg = inputs(2);
    delta_T = inputs(3);
    delta_e_deg = inputs(4);
    delta_s_deg = inputs(5);
    delta_a_deg = inputs(6);
    delta_r_deg = inputs(7);
    p_degps = inputs(8);
    q_degps = inputs(9);
```

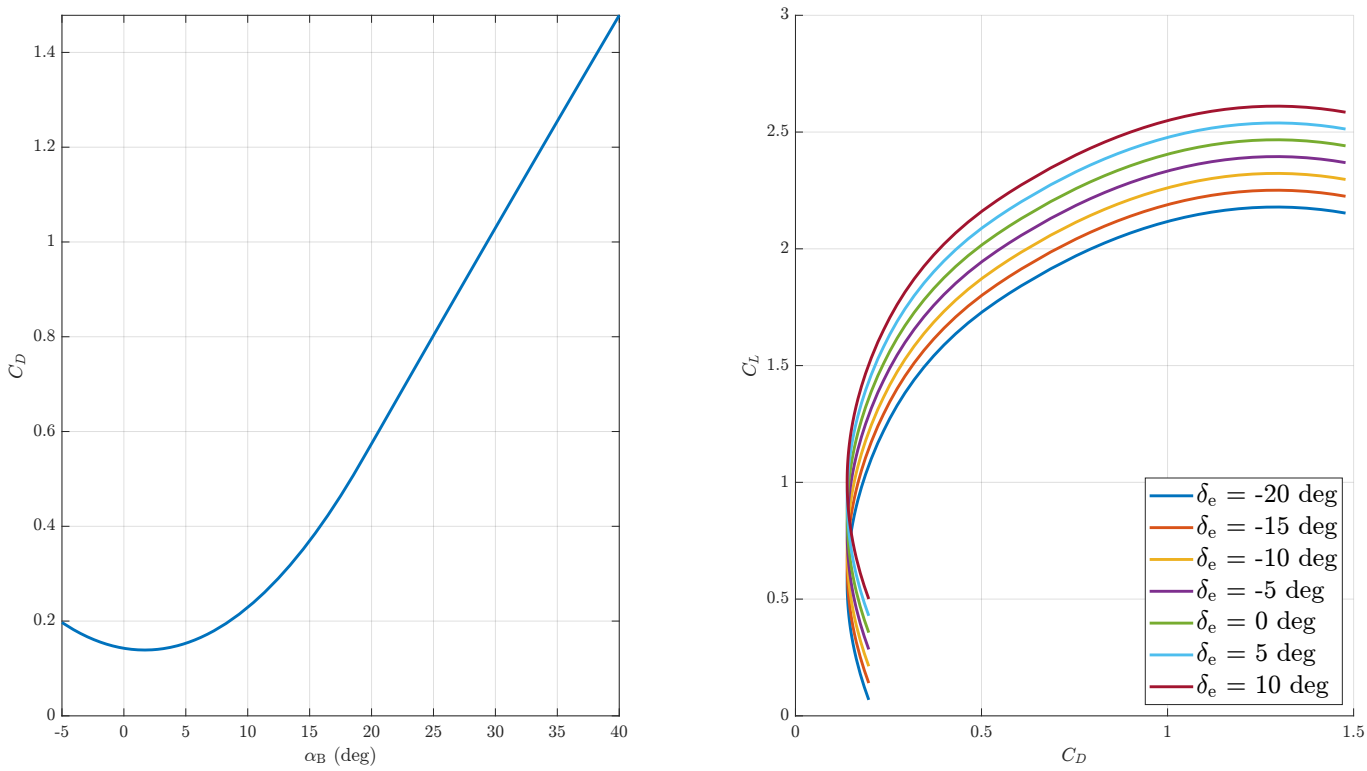


Figure 2.3 Drag characteristics of the aircraft.

```
    r_degps = inputs(10);
    altitude = inputs(11);
    airspeed = inputs(12);

    if alpha_deg >= -5 && alpha_deg <= 20
        CD = 0.1426 - 0.00438 * alpha_deg + 0.0013 * alpha_deg^2;
    elseif alpha_deg > 20 && alpha_deg <= 40
        CD = -0.3580 + 0.0473 * alpha_deg - 0.0000348 * alpha_deg^2;
    elseif alpha_deg < -5
        CD = 0.1426 + 0.00438 * 5 + 0.0013 * 5^2;
    else
        CD = -0.3580 + 0.0473 * 40 - 0.0000348 * 40^2;
    end

end
```

The aircraft crossforce coefficient model used is the following.

$$C_{Y_A} = -0.0186\beta + (0.039 - 0.00227\alpha_B)\frac{\delta_a}{25} + (0.141 - 0.00265\alpha_B)\frac{\delta_r}{30}$$

Listing 2.4 F/A-18 HARV crossforce coefficient aerodynamic model.

```
function CC = CrossforceCoeffModel(inputs)

    alpha_deg = inputs(1);
    beta_deg = inputs(2);
    delta_T = inputs(3);
    delta_e_deg = inputs(4);
    delta_s_deg = inputs(5);
    delta_a_deg = inputs(6);
    delta_r_deg = inputs(7);
    p_degps = inputs(8);
    q_degps = inputs(9);
    r_degps = inputs(10);
    altitude = inputs(11);
    airspeed = inputs(12);
```

```
15
16    CC = -0.0186 * beta_deg + ...
17          delta_a_deg * (0.039 - 0.00227 * alpha_deg) / 25 + ...
18          delta_r_deg * (0.141 - 0.00265 * alpha_deg) / 30;
19
20 end
```

The $C_{\mathscr{L}}$ model assumed is

$$C_{\mathscr{L}} = C_{\mathscr{L}_0} - 0.0315p + 0.0126r + (-0.0628 + 0.00121\alpha_{\mathrm{B}})\frac{\delta_{\mathrm{a}}}{25} - (-0.0124 + 0.000351\alpha_{\mathrm{B}})\frac{\delta_{\mathrm{r}}}{30}$$

where

$$C_{\mathscr{L}_0} = \begin{cases} (-0.00092 - 0.00012\alpha_{\mathrm{B}})\beta & -5 \text{ deg} \leq \alpha_{\mathrm{B}} \leq 15 \text{ deg} \\ (-0.006 + 0.00022\alpha_{\mathrm{B}})\beta & 15 \text{ deg} < \alpha_{\mathrm{B}} \leq 25 \text{ deg} \end{cases}$$

In the following listing the code for its MATLAB definition.

Listing 2.5 F/A-18 HARV aerodynamic roll coefficient model.

```
1  function CRoll = RollCoeffModel(inputs)
2
3     alpha_deg = inputs(1);
4     beta_deg = inputs(2);
5     delta_T = inputs(3);
6     delta_e_deg = inputs(4);
7     delta_s_deg = inputs(5);
8     delta_a_deg = inputs(6);
9     delta_r_deg = inputs(7);
10    p_degps = inputs(8);
11    q_degps = inputs(9);
12    r_degps = inputs(10);
13    altitude = inputs(11);
14    airspeed = inputs(12);
15
16    p = convangvel(p_degps, 'deg/s', 'rad/s');
17    r = convangvel(r_degps, 'deg/s', 'rad/s');
18
19    if alpha_deg >= -5 && alpha_deg <= 15
20        CRoll_0 = (-0.00092 - 0.00012 * alpha_deg) * beta_deg;
21    elseif alpha_deg > 15 && alpha_deg <= 25
22        CRoll_0 = (-0.006 + 0.00022 * alpha_deg) * beta_deg;
23    elseif alpha_deg < -5
24        CRoll_0 = (-0.00092 + 0.00012 * 5) * beta_deg;
25    else
26        CRoll_0 = (-0.006 + 0.00022 * 25) * beta_deg;
27    end
28    CRoll = CRoll_0 - 0.0315 * p + 0.0126 * r + ...
29        delta_a_deg * (-0.0628 + 0.00121 * alpha_deg) / 25 - ...
30        delta_r_deg * (-0.0124 + 0.000351 * alpha_deg) / 30;
31
32 end
```

The pitching characteristics of the aircraft is given by the following. Diagram shown in figure 2.4 and code shown in listing 2.6.

$$C_{\mathscr{M}} = -0.1885 - 0.00437\alpha_{\mathrm{B}} - 0.123q - 0.0196\delta_{\mathrm{e}}$$

Listing 2.6 F/A-18 HARV aerodynamic pitch coefficient model.

```
1  function CPitch = PitchCoeffModel(inputs)
2
3     alpha_deg = inputs(1);
4     beta_deg = inputs(2);
5     delta_T = inputs(3);
6     delta_e_deg = inputs(4);
7     delta_s_deg = inputs(5);
8     delta_a_deg = inputs(6);
```

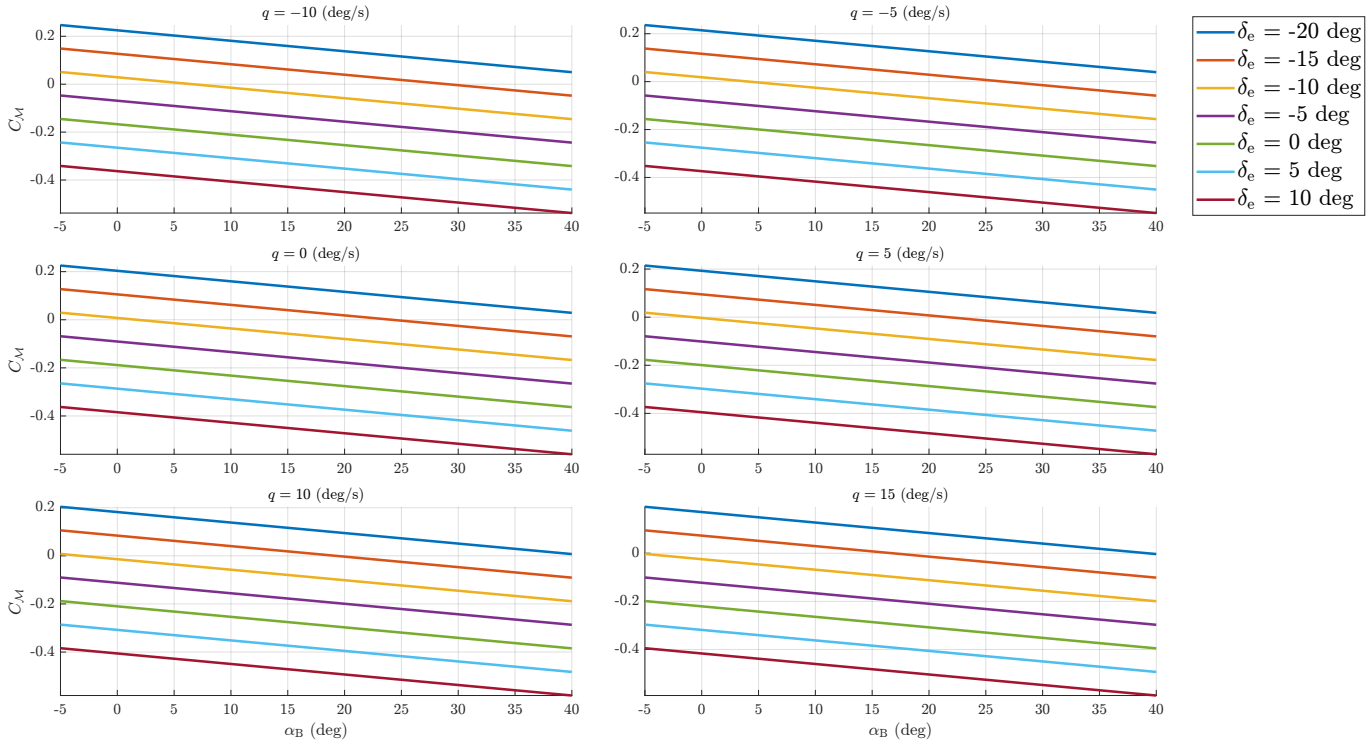


Figure 2.4 Pitching characteristics of the aircraft.

```
    delta_r_deg = inputs(7);
    p_degps = inputs(8);
    q_degps = inputs(9);
    r_degps = inputs(10);
    altitude = inputs(11);
    airspeed = inputs(12);

    q = convangvel(q_degps, 'deg/s', 'rad/s');

    CPitch = -0.1885 - 0.00437 * alpha_deg - ...
        0.123 * q - 0.0196 * delta_e_deg;

end
```

The yaw moment coefficient is defined as follows.

$$C_{\mathcal{N}} = C_{\mathcal{N}_0} - 0.0142r + (0.00128 + 0.000213\alpha_B)\frac{\delta_a}{25} + (-0.0474 + 0.000804\alpha_B)\frac{\delta_r}{30}$$

where

$$C_{\mathcal{N}_0} = \begin{cases} 0.00125\beta & -5 \text{ deg} \leq \alpha_B \leq 10 \text{ deg} \\ (0.00342 - 0.00022\alpha_B)\beta & 10 \text{ deg} < \alpha_B \leq 25 \text{ deg} \\ -0.00201\beta & 25 \text{ deg} < \alpha_B \leq 35 \text{ deg} \end{cases}$$

Listing 2.7 F/A-18 HARV aerodynamic yaw coefficient model.

```
function CYaw = YawCoeffModel(inputs)

    alpha_deg = inputs(1);
    beta_deg = inputs(2);
    delta_T = inputs(3);
    delta_e_deg = inputs(4);
    delta_s_deg = inputs(5);
    delta_a_deg = inputs(6);
    delta_r_deg = inputs(7);
```

```
    p_degps = inputs(8);
    q_degps = inputs(9);
    r_degps = inputs(10);
    altitude = inputs(11);
    airspeed = inputs(12);

    r = convangvel(r_degps, 'deg/s', 'rad/s');

    if alpha_deg >= -5 && alpha_deg <= 10
        CYaw_0 = 0.00125 * beta_deg;
    elseif alpha_deg > 10 && alpha_deg <= 25
        CYaw_0 = (0.00342 - 0.00022 * alpha_deg) * beta_deg;
    elseif alpha_deg > 25 && alpha_deg <= 35
        CYaw_0 = -0.00201 * beta_deg;
    elseif alpha_deg < -5
        CYaw_0 = 0.00125 * beta_deg;
    else
        CYaw_0 = -0.00201 * beta_deg;
    end

    CYaw = CYaw_0 - 0.0142 * r + ...
        delta_a_deg * (0.00128 + 0.000213 * alpha_deg) / 25 + ...
        delta_r_deg * (-0.0474 + 0.000804 * alpha_deg) / 30;

end
```

Finally, the simplest possible form for a thrust model is chosen. The dependencies of $T_{\max}$ from the altitude h and the airspeed V_G are assumed as negligible for this simple model.

$$T = \delta_{\mathrm{T}} T_{\max}$$

Listing 2.8 Trivial propulsion model.

```
function T = ThrustModel(inputs, myAircraft)

    alpha_deg = inputs(1);
    beta_deg = inputs(2);
    delta_T = inputs(3);
    delta_e_deg = inputs(4);
    delta_s_deg = inputs(5);
    delta_a_deg = inputs(6);
    delta_r_deg = inputs(7);
    p_degps = inputs(8);
    q_degps = inputs(9);
    r_degps = inputs(10);
    altitude = inputs(11);
    airspeed = inputs(12);

    T_max = myAircraft.T_max;

    T = delta_T * T_max;

end
```

After all of that has been set up, the following script can be executed. It inserts the data into the `saveAircraftStruct` routine so that it can properly save the aircraft struct `f18harv.mat`. In particular, this aircraft geometrical and mass parameters are

- Mass $m = 1036$ slug
- Maximum thrust $T_{\max} = 11200$ lbf
- Wing reference surface $S = 400$ ft^2
- Wingspan $b = 37.42$ ft
- Reference mean aerodynamic chord $c = 11.52$ ft

- Tensor of inertia components in Body axes $I_{xx} = 23000$ slug ft^2, $I_{yy} = 151293$ slug ft^2, $I_{zz} = 169945$ slug ft^2 and $I_{xy} = I_{xz} = I_{yz} \approx 0$

Listing 2.9 Script for saving the F/A-18 HARV struct.

```
close all; clear; clc

m = convmass(1036, 'slug', 'kg');
T_max = convforce(11200, 'lbf', 'N');
S = convlength(convlength(400, 'ft', 'm'), 'ft', 'm');
b = convlength(37.42, 'ft', 'm');
c = convlength(11.52, 'ft', 'm');

Ixx = convmass(23000, 'slug', 'kg');
Ixx = convlength(convlength(Ixx, 'ft', 'm'), 'ft', 'm');

Iyy = convmass(151293, 'slug', 'kg');
Iyy = convlength(convlength(Iyy, 'ft', 'm'), 'ft', 'm');

Izz = convmass(169945, 'slug', 'kg');
Izz = convlength(convlength(Izz, 'ft', 'm'), 'ft', 'm');

Ixy = 0;
Iyz = 0;
Ixz = 0;

I_B = [Ixx, Ixy, Ixz; ...
       Ixy, Iyy, Iyz; ...
       Ixz, Iyz, Izz];

aircraftName = 'f18harv';

saveAircraftStruct(aircraftName, S, b, c, m, I_B, ...
                   T_max, @ThrustModel, ...
                   @LiftCoeffModel, @DragCoeffModel, @CrossforceCoeffModel, ...
                   @RollCoeffModel, @PitchCoeffModel, @YawCoeffModel);
```

Now, the simulation of this aircraft can be carried out. For this sake, I wrote a general MATLAB routine, suitable for any aircraft struct defined as previously explained. This function (see listing 2.10) takes the simulation duration, initial conditions, the struct and the input commands functions as input parameters and outputs discretized vectors of the simulation time, input commands and state variables. It does that by integrating the system of ordinary differential equations (2.33).

It starts by saving the aircraft geometric and mass parameters into local scoped variables. Then, the matrix operators $\left[\tilde{\Omega}_{\mathrm{B}}\right]_{\mathrm{B}}$, $[T]_{\mathrm{EB}}$ and $[Q]_{\mathrm{evol}}$ and some helper anonymous functions are defined as anonymous functions.

The generalized inputs vector $\boldsymbol{u}^*$ is defined as an anonymous function to later be used as an input parameter to the propulsion and aerodynamic coefficients models that will be locally defined.

The use of the models lets us evaluate the components of the resultant force and moment in Body axes. In particular, X, Y, Z, $\mathscr{L}$, $\mathscr{M}$ and $\mathscr{N}$ are computed.

The function terminates evaluating the solution for state variables using the ode45 routine and puts all the requested outputs into column vectors.

Listing 2.10 Routine for carrying out 6-DoF simulations.

```
function [time, ...
   delta_T, delta_e_deg, delta_s_deg, delta_a_deg, delta_r_deg, ...
   u, v, w, p, q, r, ...
   x_EG, y_EG, z_EG, quat0, quatx, quaty, quatz] = ...
                  SixDoFQuat(t_end, state0, ...
                             myAircraft, ...
```

```
                                delta_T_law, ...
                                delta_e_deg_law, delta_s_deg_law, ...
                                delta_a_deg_law, delta_r_deg_law)

S = myAircraft.S;
b = myAircraft.b;
c = myAircraft.c;
m = myAircraft.m;
W = myAircraft.W;
I_B = myAircraft.I_B;

O_3x3 = zeros(3);
O_4x4 = zeros(4);
O_4x3 = zeros(4, 3);
O_3x4 = zeros(3, 4);
O_7x1 = zeros(7, 1);

omega_B_tilde = @(state) [   0,     -state(6),  state(5); ...
                         state(6),     0,     -state(4); ...
                        -state(5),  state(4),     0    ];

T_EB = @(state) ...
         quat2dcm([state(10), state(11), state(12), state(13)]).';

Quat_evol = @(state) 0.5 * [-state(11), -state(12), -state(13); ...
                            state(10), -state(13),  state(12); ...
                            state(13),  state(10), -state(11); ...
                           -state(12),  state(11),  state(10)];

function rho = density(h)
 [~, ~, ~, rho] = atmosisa(h);
end
rho = @(state) density(-state(9));

airspeed = @(state) sqrt(state(1)^2 + state(2)^2 + state(3)^2);
alpha_deg = @(state) atand(state(3) / state(1));
beta_deg = @(state) asind(state(2) / airspeed(state));
p_degps = @(state) convangvel(state(4), 'rad/s', 'deg/s');
q_degps = @(state) convangvel(state(5), 'rad/s', 'deg/s');
r_degps = @(state) convangvel(state(6), 'rad/s', 'deg/s');
altitude = @(state) -state(9);

quaternion = @(state) [state(10), state(11), state(12), state(13)];

function psi = quat2psi(quat)
 [psi, ~, ~] = quat2angle(quat);
end
psi = @(state) quat2psi(quaternion(state));

function theta = quat2theta(quat)
 [~, theta, ~] = quat2angle(quat);
end
theta = @(state) quat2theta(quaternion(state));

function phi = quat2phi(quat)
 [~, ~, phi] = quat2angle(quat);
end
phi = @(state) quat2phi(quaternion(state));

InputCommands = @(t) ...
 [delta_T_law(t), ...
 delta_e_deg_law(t), delta_s_deg_law(t), ...
 delta_a_deg_law(t), delta_r_deg_law(t)];

ModelsInputs = @(t, state) ...
 [alpha_deg(state), beta_deg(state), ...
  InputCommands(t), ...
  p_degps(state), q_degps(state), r_degps(state), ...
  altitude(state), airspeed(state)];

ThrustModel = @(t, state) myAircraft.ThrustModel(ModelsInputs(t, state));
DragCoeffModel = @(t, state) myAircraft.DragCoeffModel(ModelsInputs(t,
```

```
    ↪ state));
LiftCoeffModel = @(t, state) myAircraft.LiftCoeffModel(ModelsInputs(t,
    ↪ state));
CrossforceCoeffModel = @(t, state)
    ↪ myAircraft.CrossforceCoeffModel(ModelsInputs(t, state));
RollCoeffModel = @(t, state) myAircraft.RollCoeffModel(ModelsInputs(t,
    ↪ state));
PitchCoeffModel = @(t, state) myAircraft.PitchCoeffModel(ModelsInputs(t,
    ↪ state));
YawCoeffModel = @(t, state) myAircraft.YawCoeffModel(ModelsInputs(t,
    ↪ state));

LiftModel = @(t, state) ...
 LiftCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S;

DragModel = @(t, state) ...
 DragCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S;

CrossforceModel = @(t, state) ...
 CrossforceCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S;

X = @(t, state) ...
 ThrustModel(t, state) - ...
 DragModel(t, state) * cosd(alpha_deg(state)) + ...
 LiftModel(t, state) * sind(alpha_deg(state)) - ...
 W * sin(theta(state));

Y = @(t, state) ...
 CrossforceModel(t, state) + ...
 W * sin(phi(state)) * cos(theta(state));

Z = @(t, state) ...
-DragModel(t, state) * sind(alpha_deg(state)) - ...
 LiftModel(t, state) * cosd(alpha_deg(state)) + ...
 W * cos(phi(state)) * cos(theta(state));

Roll = @(t, state) ...
 RollCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S * b;

Pitch = @(t, state) ...
 PitchCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S * c;

Yaw = @(t, state) ...
 YawCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S * b;

dstate_dt = @(t, state) ...
 [-omega_B_tilde(state),           O_3x3,              O_3x3, O_3x4; ...
         O_3x3,          -I_B\omega_B_tilde(state)*I_B, O_3x3, O_3x4; ...
       T_EB(state),                O_3x3,              O_3x3, O_3x4; ...
         O_4x3,              Quat_evol(state),        O_4x3, O_4x4] *
    ↪ ...
   state + ...
 [ 1/m * [ X(t, state); ...
           Y(t, state); ...
           Z(t, state) ]; ...
   I_B \ [ Roll(t, state); ...
           Pitch(t, state); ...
           Yaw(t, state) ]; ...
   O_7x1 ];

ODEoptions = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);

[time, state] = ode45(dstate_dt, [0, t_end], state0, ODEoptions);

delta_T = delta_T_law(time);         delta_T = delta_T(:);
delta_e_deg = delta_e_deg_law(time); delta_e_deg = delta_e_deg(:);
```

```
delta_s_deg = delta_s_deg_law(time); delta_s_deg = delta_s_deg(:);
delta_a_deg = delta_a_deg_law(time); delta_a_deg = delta_a_deg(:);
delta_r_deg = delta_r_deg_law(time); delta_r_deg = delta_r_deg(:);
u = state(:, 1);
v = state(:, 2);
w = state(:, 3);
p = state(:, 4);
q = state(:, 5);
r = state(:, 6);
x_EG = state(:, 7);
y_EG = state(:, 8);
z_EG = state(:, 9);
quat0 = state(:, 10);
quatx = state(:, 11);
quaty = state(:, 12);
quatz = state(:, 13);

end
```

The following runnable script plots the aircraft lift, drag and pitching characteristics previously shown in figures 2.2, 2.3 and 2.4 and executes the `SixDoFQuat` routine to then plots the results.

After the initial plots have been displayed to show the models used in the simulation, the duration of that can be assigned as $t_{\text{end}} = 5$ s. The initial conditions $\boldsymbol{x}_0$ are set to the following,

$$\boldsymbol{x}_0 = \left[u_0,\ v_0,\ w_0,\ p_0,\ q_0,\ r_0,\ x_{\text{E},G,0},\ y_{\text{E},G,0},\ z_{\text{E},G,0},\ \boldsymbol{q}_0\right]^{\text{T}}$$

evaluated as in listing 2.11, given the following chosen parameters

- $h_0 = 1000$ m and $M_0 = 0.20$
- $\varphi_0 = 0$ deg, $\theta_0 = 4.46$ deg and $\psi_0 = 0$ deg
- $\alpha_{\text{B},0} = 9.96$ deg and $\beta_0 = 0$ deg
- $p_0 = 0$ deg/s, $q_0 = 0$ deg/s and $r_0 = 0$ deg/s

The input commands δ_{T}, δ_{e}, δ_{s}, δ_{a} and δ_{r} are assigned as anonymous functions of the time t using the `interp1` routine. Their nonzero diagrams are shown in figure 2.5.

The solver function `SixDoFQuat` is called to obtain the column vectors of the time, input commands and state variables histories. `multiPlot`, `stackedPlot2`, `stackedPlot3`, `stackedPlot4` and `trajectoryView` are some helper functions I wrote that are used to obtain the plots that will later be shown.

In particular, the histories of the center of gravity velocity and angular velocity components in Body axes are displayed in figures 2.6 and 2.7. The time histories of the center of gravity coordinates in Earth axes and of the airspeed are shown in figures 2.8 and 2.9. Euler angles and aerodynamic angles time histories are displayed in figures 2.10 and 2.11. Time histories of the aerodynamic coefficients are shown in figures 2.12 and 2.13.

The load factor components in aerodynamic axes, defined as in equation (2.36) are obtained from the MATLAB function I wrote, later shown in listing 2.12. The time history of the normal load factor is lastly displayed in figure 2.14.

At the end, the trajectory plot firstly shown in figure 2.1 is obtained from the script.

Listing 2.11 F/A-18 HARV simulation.

```
close all; clear; clc

myAircraft = load('f18harv.mat');

```

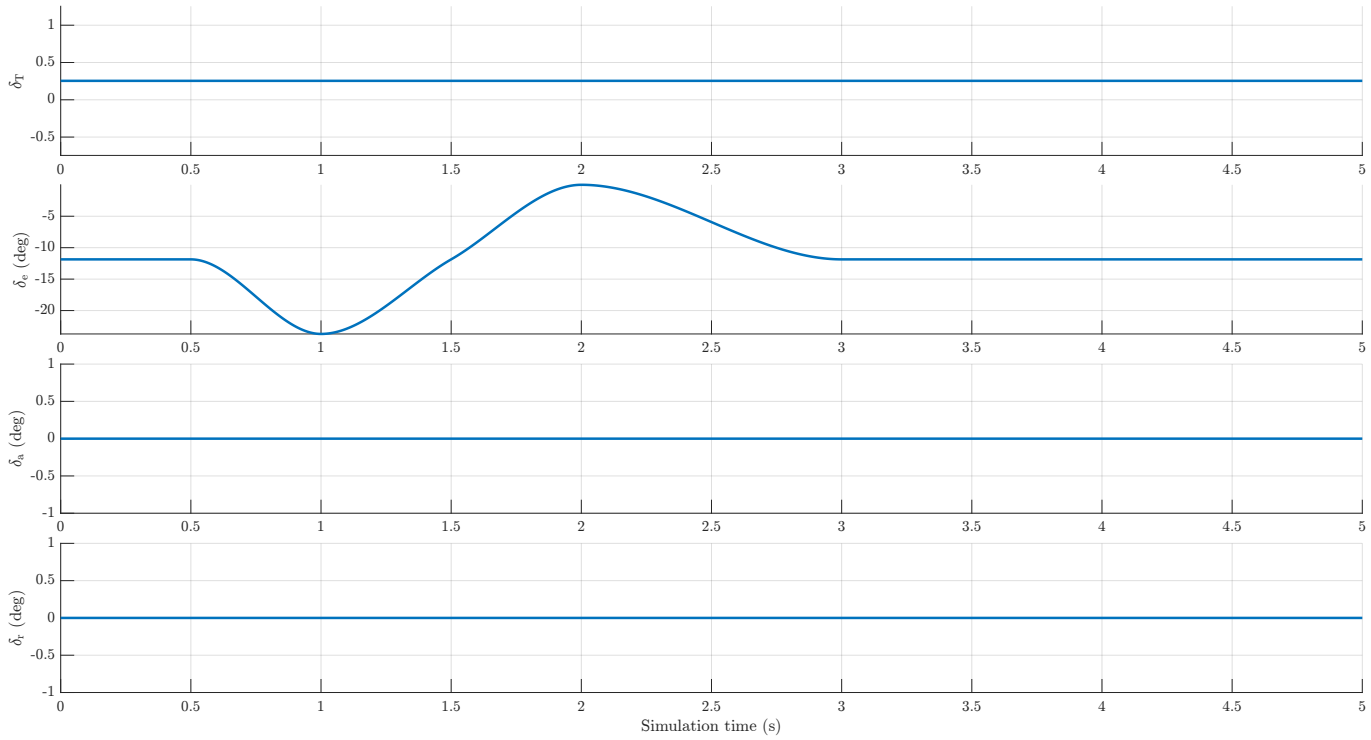


Figure 2.5 Time histories of the nonzero input commands assigned laws.

```
ModelsInputs = zeros(1, 11);
delta_e_deg_cases = -20 : 5 : 10;

liftCharacteristicsPlot(myAircraft, ModelsInputs, [-5, 40], ...
                delta_e_deg_cases, 'ex72liftcharacteristics.pdf')

staticPolarPlot(myAircraft, ModelsInputs, [-5, 40], ...
                delta_e_deg_cases, 'ex72dragcharacteristics.pdf')

q_degps_cases = linspace(-10, 15, 6);
pitchCharacteristicsPlot(myAircraft, ModelsInputs, [-5, 40], ...
                    delta_e_deg_cases, q_degps_cases, ...
                    'ex72pitchcharacteristics.pdf')

t_end = 5;

x_EG0 = 0;
y_EG0 = 0;
z_EG0 = -1000;

phi0 = 0;
theta0 = convang(4.46, 'deg', 'rad');
psi0 = 0;
Quat0 = angle2quat(psi0, theta0, phi0);

alpha0_deg = 9.96;
beta0_deg = 0;
Mach0 = 0.20;
[temp0, sound0, press0, dens0] = atmosisa(-z_EG0);
V0 = Mach0 * sound0;
u0 = V0 * cosd(beta0_deg) * cosd(alpha0_deg);
v0 = V0 * sind(beta0_deg);
w0 = V0 * cosd(beta0_deg) * sind(alpha0_deg);

p0 = 0;
q0 = 0;
r0 = 0;

state0 = [u0, v0, w0, p0, q0, r0, x_EG0, y_EG0, z_EG0, Quat0].';
```

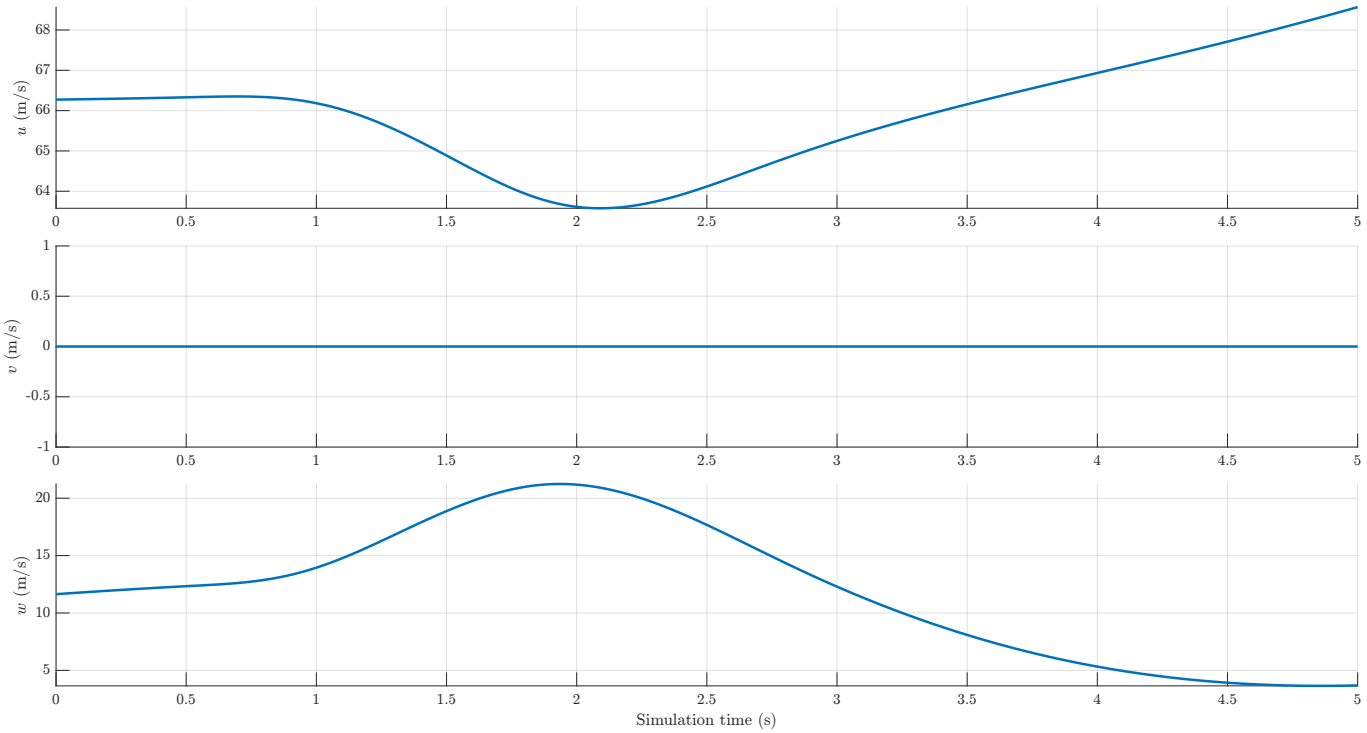


Figure 2.6 Time histories of the center of gravity velocity components in Body axes.

```
44
45 delta_T0 = 0.254;
46 delta_T_law = @(t) interp1([0, 1] * t_end, ...
47                           [1, 1] * delta_T0, ...
48                           t, 'linear');
49
50 delta_e0_deg = -11.86;
51 delta_e_deg_law = @(t) interp1( ...
52                 [0, 1/10, 2/10, 3/10, 4/10, 6/10, 1] * t_end, ...
53                 [1,   1,    2,    1,    0,    1,  1] * delta_e0_deg, ...
54                 t, 'pchip');
55
56 delta_s0_deg = 0;
57 delta_s_deg_law = @(t) interp1([0, 1] * t_end, ...
58                                [1, 1] * delta_s0_deg, ...
59                                t, 'linear');
60
61 delta_a0_deg = 0;
62 delta_a_deg_law = @(t) interp1([0, 1] * t_end, ...
63                                [1, 1] * delta_a0_deg, ...
64                                t, 'linear');
65
66 delta_r0_deg = 0;
67 delta_r_deg_law = @(t) interp1([0, 1] * t_end, ...
68                                [1, 1] * delta_r0_deg, ...
69                                t, 'linear');
70
71 [time, ...
72 delta_T, delta_e_deg, delta_s_deg, delta_a_deg, delta_r_deg, ...
73 u, v, w, p, q, r, ...
74 x_EG, y_EG, z_EG, quat0, quatx, quaty, quatz] = ...
75     SixDoFQuat(t_end, state0, ...
76                myAircraft, ...
77                delta_T_law, ...
78                delta_e_deg_law, delta_s_deg_law, ...
79                delta_a_deg_law, delta_r_deg_law);
80
81 Nt = length(time);
82
83 stackedPlot4(time, ...
```

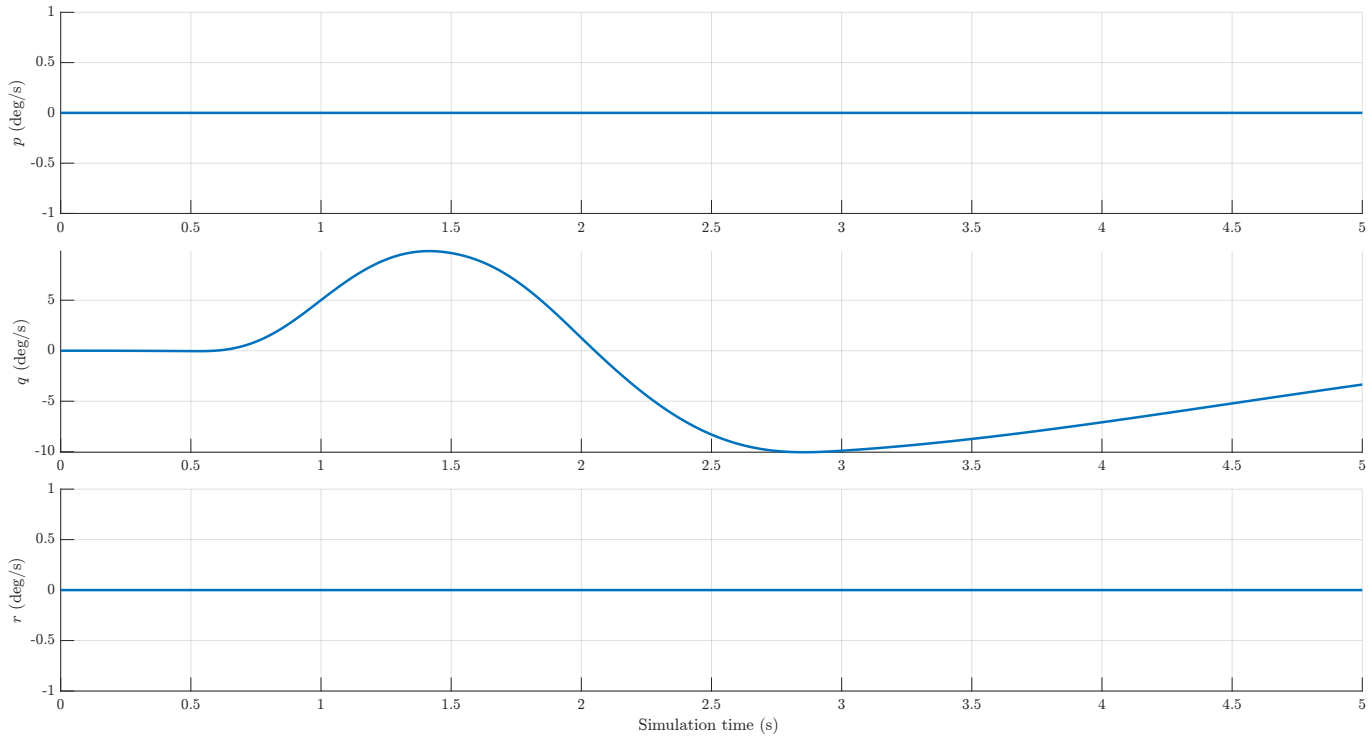


Figure 2.7 Time histories of the center of gravity angular velocity components in Body axes.

```
            delta_T, delta_e_deg, delta_a_deg, delta_r_deg, ...
            {"Simulation time (s)", ...
                "$\delta_\mathrm{T}$", "$\delta_\mathrm{e}$ (deg)", ...
                "$\delta_\mathrm{a}$ (deg)", "$\delta_\mathrm{r}$
    ↪ (deg)"}, ...
            {}, {}, 'ex72inputcommands.pdf')

stackedPlot3(time, ...
            u, v, w, ...
            {"Simulation time (s)", ...
                "$u$ (m/s)", "$v$ (m/s)", "$w$ (m/s)"}, ...
            {}, {}, 'ex72velcomponents.pdf')

p_degps = convangvel(p, 'rad/s', 'deg/s');
q_degps = convangvel(q, 'rad/s', 'deg/s');
r_degps = convangvel(r, 'rad/s', 'deg/s');
stackedPlot3(time, ...
            p_degps, q_degps, r_degps, ...
            {"Simulation time (s) ", ...
                "$p$ (deg/s)", "$q$ (deg/s)", "$r$ (deg/s)"}, ...
            {}, {}, 'ex72angvelcomponents.pdf')

stackedPlot2(time, ...
            x_EG, y_EG, ...
            {"Simulation time (s)", ...
                "$x_{\mathrm{E},G}$ (m)", "$y_{\mathrm{E},G}$ (m)"}, ...
            {}, {}, 'ex72cogxy.pdf')

altitude = -z_EG;
airspeed = sqrt(u.^2 + v.^2 + w.^2);
stackedPlot2(time, ...
            altitude, airspeed, ...
            {"Simulation time (s)", ...
                "Altitude (m)", "Airspeed (m/s)"}, ...
            {}, {}, 'ex72altitudeairspeed.pdf')

[psi, theta, phi] = quat2angle([quat0, quatx, quaty, quatz]);
phi_deg = convang(phi, 'rad', 'deg');
theta_deg = convang(theta, 'rad', 'deg');
```

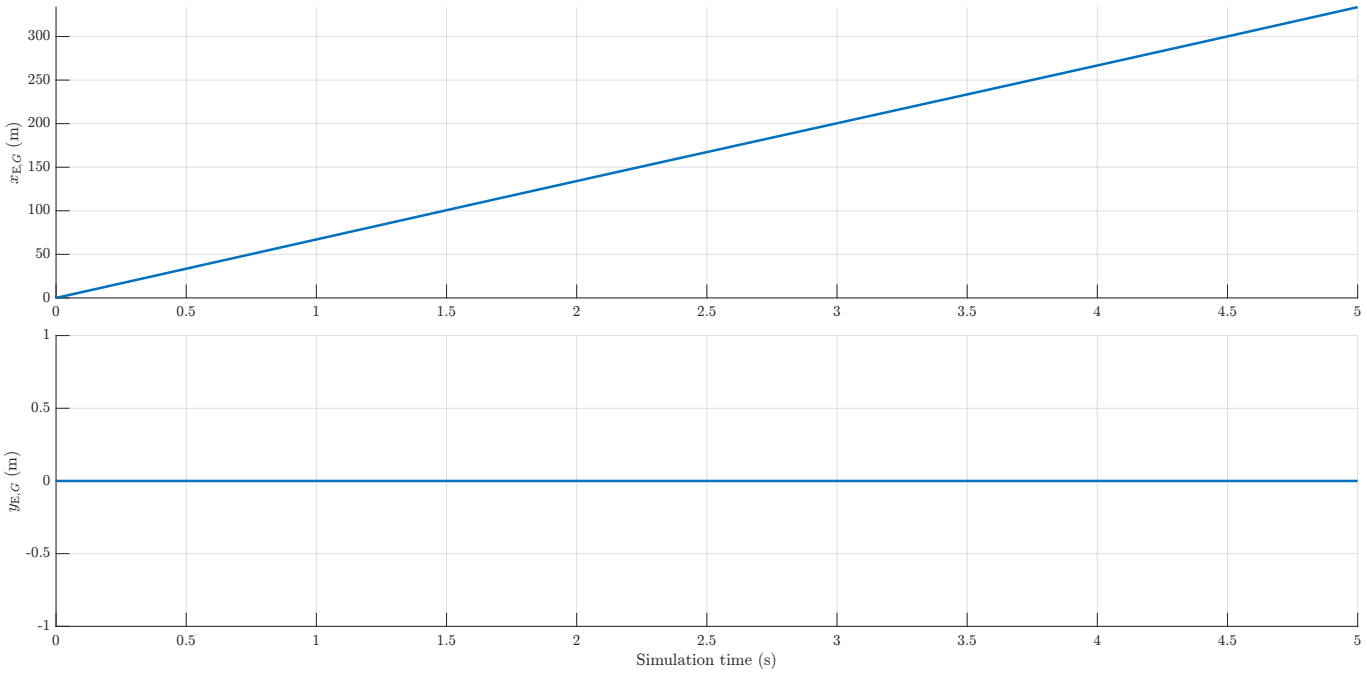


Figure 2.8 Time histories of the center of gravity components.

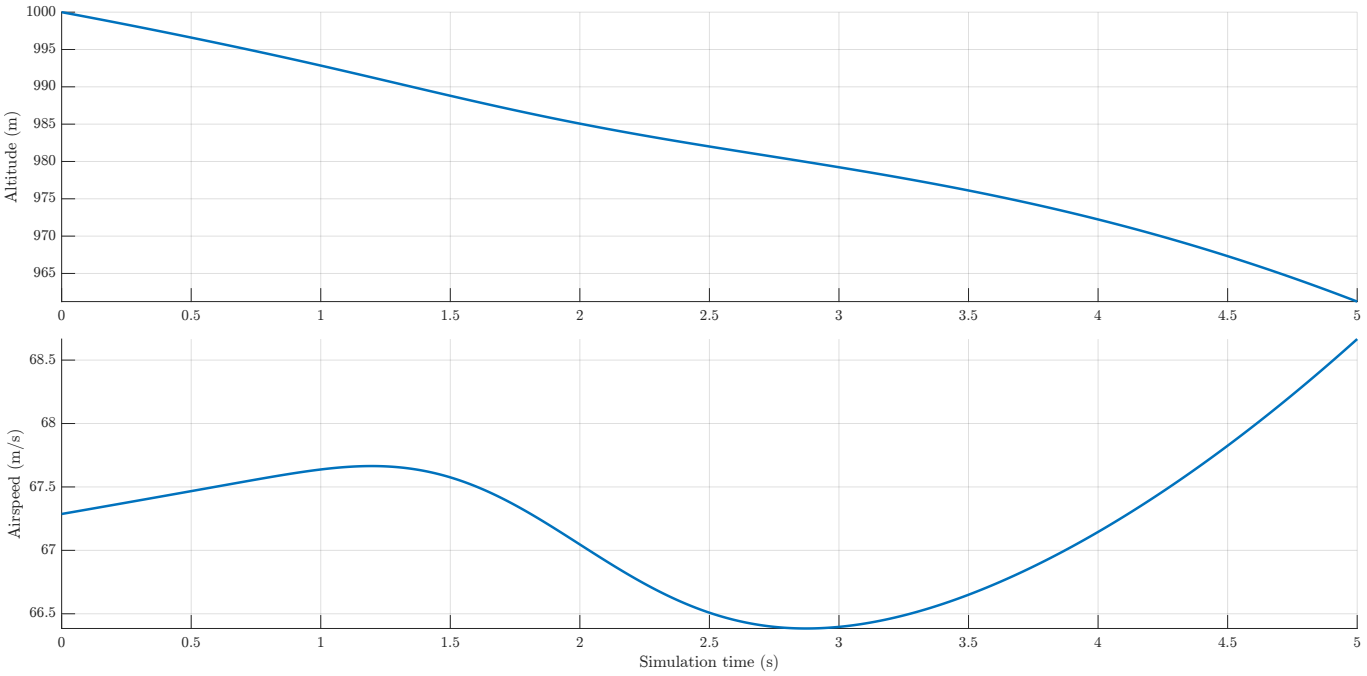


Figure 2.9 Altitude and airspeed time histories.

```
122 psi_deg = convang(psi, 'rad', 'deg');
123 stackedPlot3(time, ...
124             phi_deg, theta_deg, psi_deg, ...
125             {"Simulation time (s)", ...
126                 "$\varphi$ (deg)", "$\theta$ (deg)", "$\psi$ (deg)"}, ...
127             {}, {}, 'ex72eulangles.pdf')
128
129 [alpha_deg, beta_deg] = AlphaBetaDegHistories(u, v, w, airspeed);
130 stackedPlot2(time, ...
131             alpha_deg, beta_deg, ...
132             {"Simulation time (s)", ...
133                 "$\alpha_\mathrm{B}$ (deg)", "$\beta$ (deg)"}, ...
```

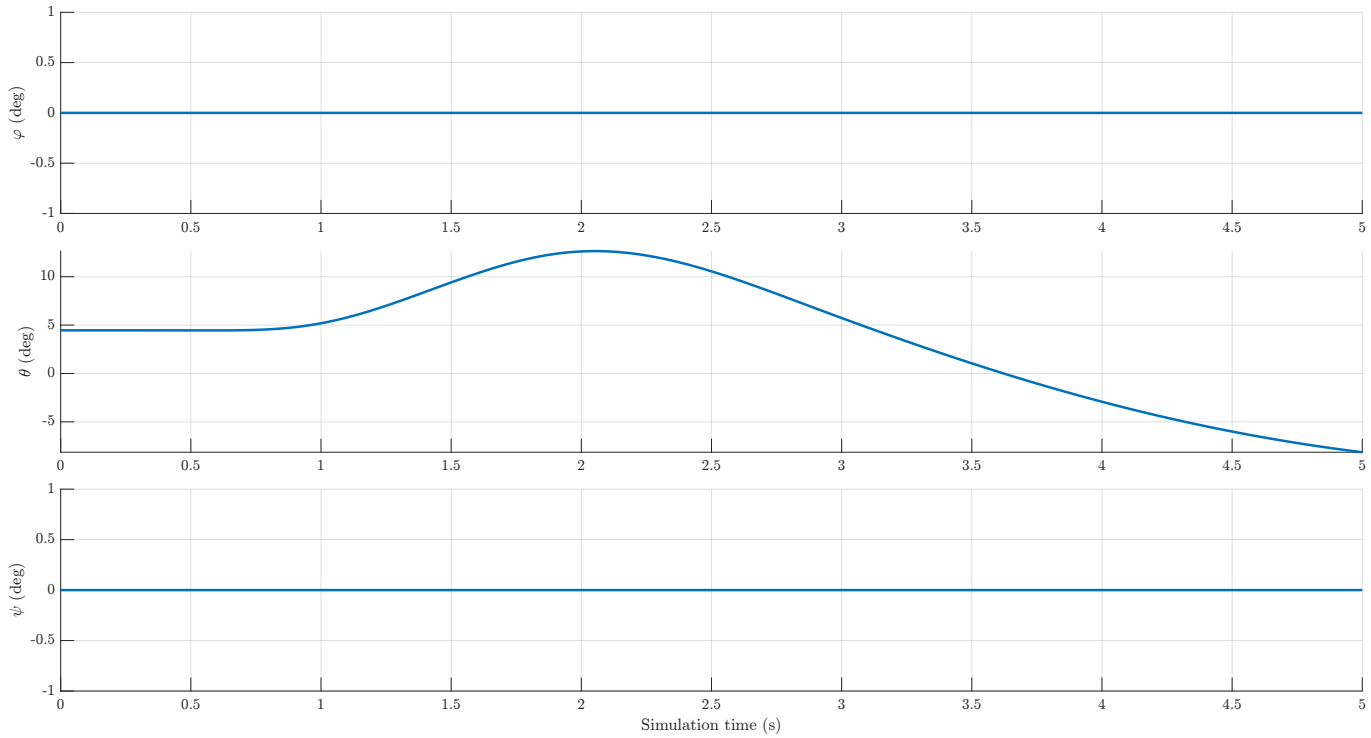


Figure 2.10 Time histories of the Euler angles.

```
            {}, {}, 'ex72aeroangles.pdf')

ModelsInputs = [alpha_deg, beta_deg, ...
                delta_T, delta_e_deg, delta_s_deg, delta_a_deg,
    ↪ delta_r_deg, ...
                p_degps, q_degps, r_degps, ...
                altitude, airspeed];
CL = zeros(Nt, 1);
CD = CL;
CC = CL;
CRoll = CL;
CPitch = CL;
CYaw = CL;
for it = 1 : Nt
    CL(it) = myAircraft.LiftCoeffModel(ModelsInputs(it, :));
    CD(it) = myAircraft.DragCoeffModel(ModelsInputs(it, :));
    CC(it) = myAircraft.CrossforceCoeffModel(ModelsInputs(it, :));
    CRoll(it) = myAircraft.RollCoeffModel(ModelsInputs(it, :));
    CPitch(it) = myAircraft.PitchCoeffModel(ModelsInputs(it, :));
    CYaw(it) = myAircraft.YawCoeffModel(ModelsInputs(it, :));
end
stackedPlot3(time, ...
            CL, CD, CC, ...
            {"Simulation time (s)", ...
                "$C_L$", "$C_D$", "$C_C$"}, ...
            {}, {}, 'ex72aeroforcecoeffhistories.pdf')
stackedPlot3(time, ...
            CRoll, CPitch, CYaw, ...
            {"Simulation time (s)", ...
                "$C_\mathcal{L}$", "$C_\mathcal{M}$",
    ↪ "$C_\mathcal{N}$"}, ...
            {}, {}, 'ex72aeromomcoeffhistories.pdf')

[f_xA, f_yA, f_zA] = LoadFactor(time, u, v, w, alpha_deg, quat0, quatx,
    ↪ quaty, quatz);
multiPlot(time, ...
            "Simulation time (s)", "$f_{z_\mathrm{A}}$", ...
            {}, ...
            'ex72loadfactor.pdf', ...
```

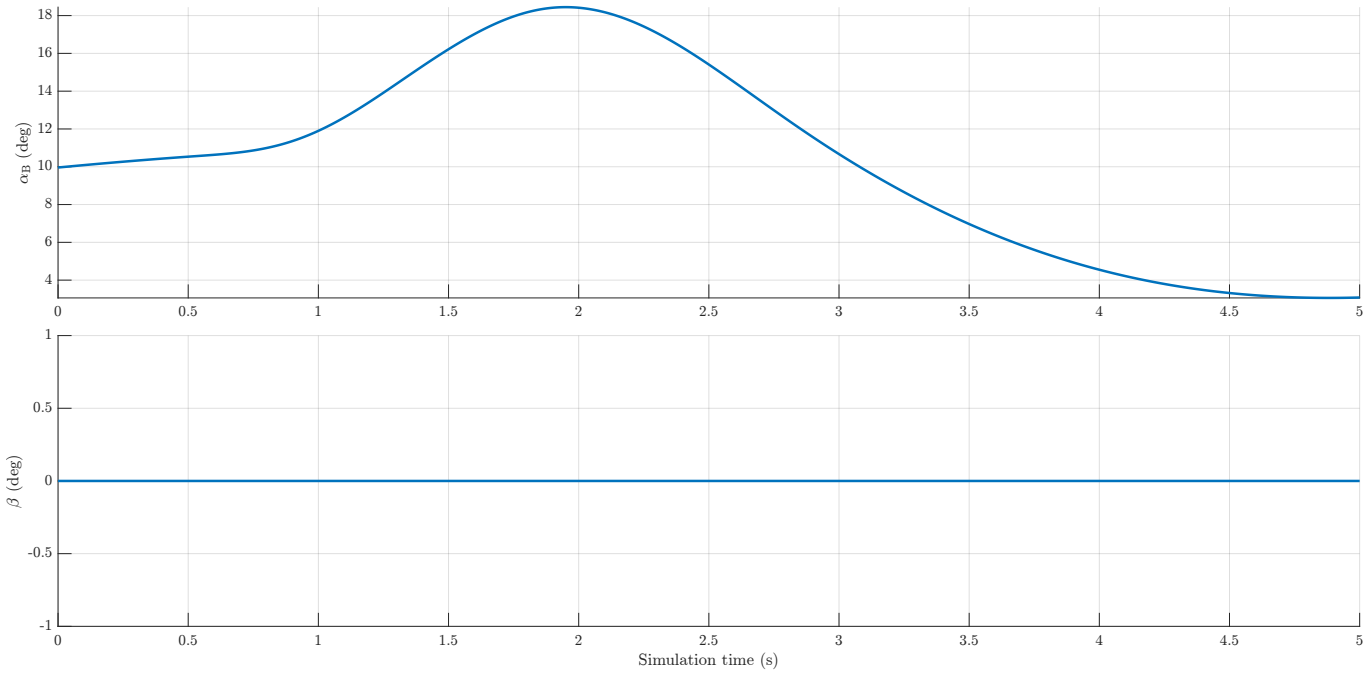


Figure 2.11 Time histories of the aerodynamic angles.

```
            f_zA)

trajectoryView(time, ...
                x_EG, y_EG, z_EG, ...
                phi, theta, psi, ...
                10, 6, ...
                'ex72trajectory.pdf')
```

Given the acceleration vector of the aircraft center of gravity $\boldsymbol{a}$, the load factor $\boldsymbol{f}$ is defined as

$$\boldsymbol{f} = \frac{\boldsymbol{g} - \boldsymbol{a}}{g} \tag{2.36}$$

and its components in aerodynamic axes are

$$\{f\}_{\mathrm{A}} = \begin{Bmatrix} f_{x_{\mathrm{A}}} \\ f_{y_{\mathrm{A}}} \\ f_{z_{\mathrm{A}}} \end{Bmatrix} \tag{2.37}$$

The following MATLAB routine computes all of these components. $\boldsymbol{a}$ is evaluated as a second order accuracy numerical time derivative of the velocity vector using the `timeDerivative` function I wrote. Its components and $\boldsymbol{g}$ components in aerodynamic axes are obtained using the transformation matrix

$$[T]_{\mathrm{AE}} = [T]_{\mathrm{AB}} [T]_{\mathrm{BE}}$$

Listing 2.12 Computes the load factor components.

```
function [f_xA, f_yA, f_zA] = LoadFactor(time, u, v, w, alpha_deg,
    ↪ quat0, quatx, quaty, quatz)

   Nt = length(time);

   T_AB = @(alpha_deg) [cosd(-alpha_deg), 0, -sind(-alpha_deg); ...
                                0        , 1,          0        ; ...
```

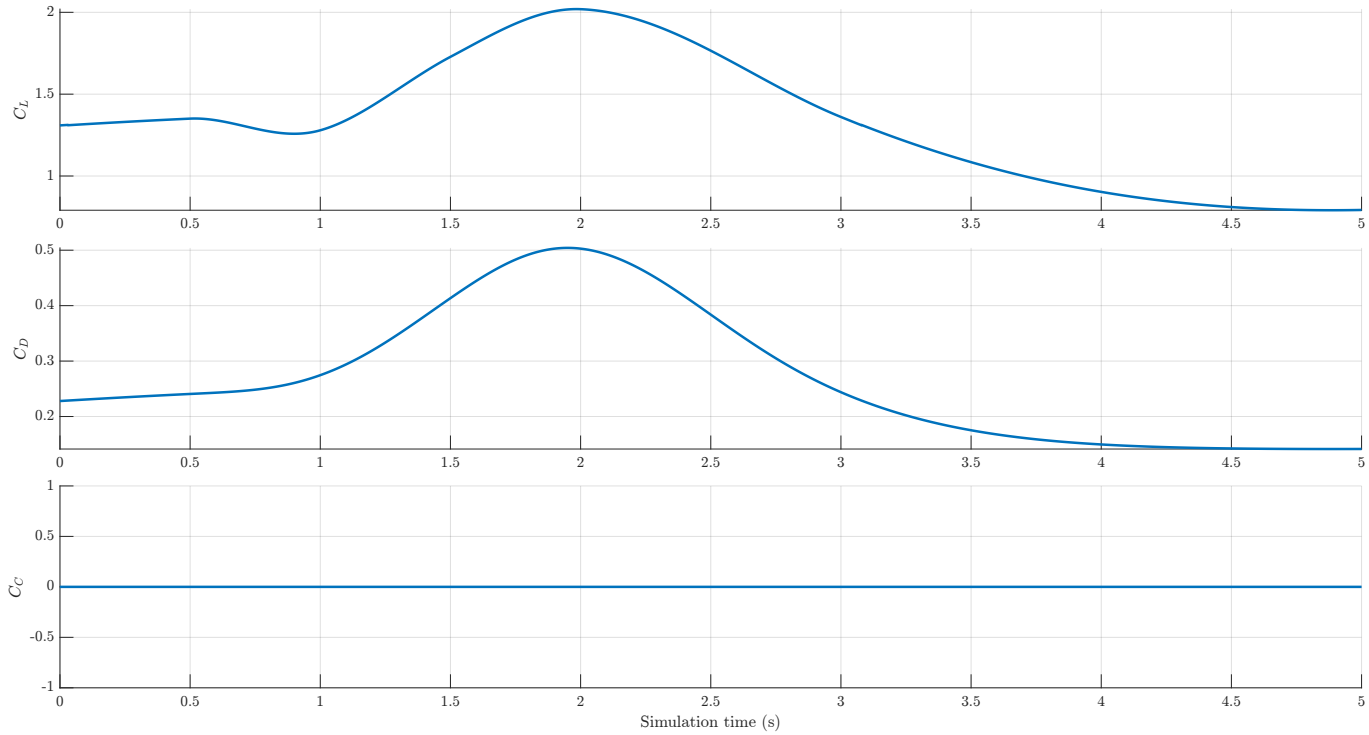


Figure 2.12 Time histories of the aerodynamic force coefficients.

```
                        sind(-alpha_deg), 0,  cosd(-alpha_deg)];

    V_E = zeros(Nt, 3);
    g_A = V_E;
    for it = 1 : Nt
        T_BE = quat2dcm([quat0(it), quatx(it), quaty(it), quatz(it)]);
        T_EB = T_BE.';
        V_E(it, :) = (T_EB * [u(it); v(it); w(it)]).';
        g_A(it, :) = (T_AB(alpha_deg(it)) * T_BE * [0; 0; 9.81]).';
    end

    a_E = timeDerivative(time, V_E);
    a_A = zeros(Nt, 3);
    for it = 1 : Nt
        T_BE = quat2dcm([quat0(it), quatx(it), quaty(it), quatz(it)]);
        a_A(it, :) = (T_AB(alpha_deg(it)) * T_BE * a_E(it, :).').';
    end

    f_A = (g_A - a_A) ./ 9.81;

    f_xA = f_A(:, 1);
    f_yA = f_A(:, 2);
    f_zA = f_A(:, 3);

end
```

6-DoF simulation using a different state vector

By choosing a different state vector we can solve for the state variables integrating a different system of differential equations, yielding another approach to the problem setup. One of the most typical choices is the use of V_G, α_B and β as state variables instead of u, v and w as in equations (2.33).

In this exercise this approach will be presented. The new state vector is defined as previously done in equation (2.31), but with the mentioned change into its dynamical

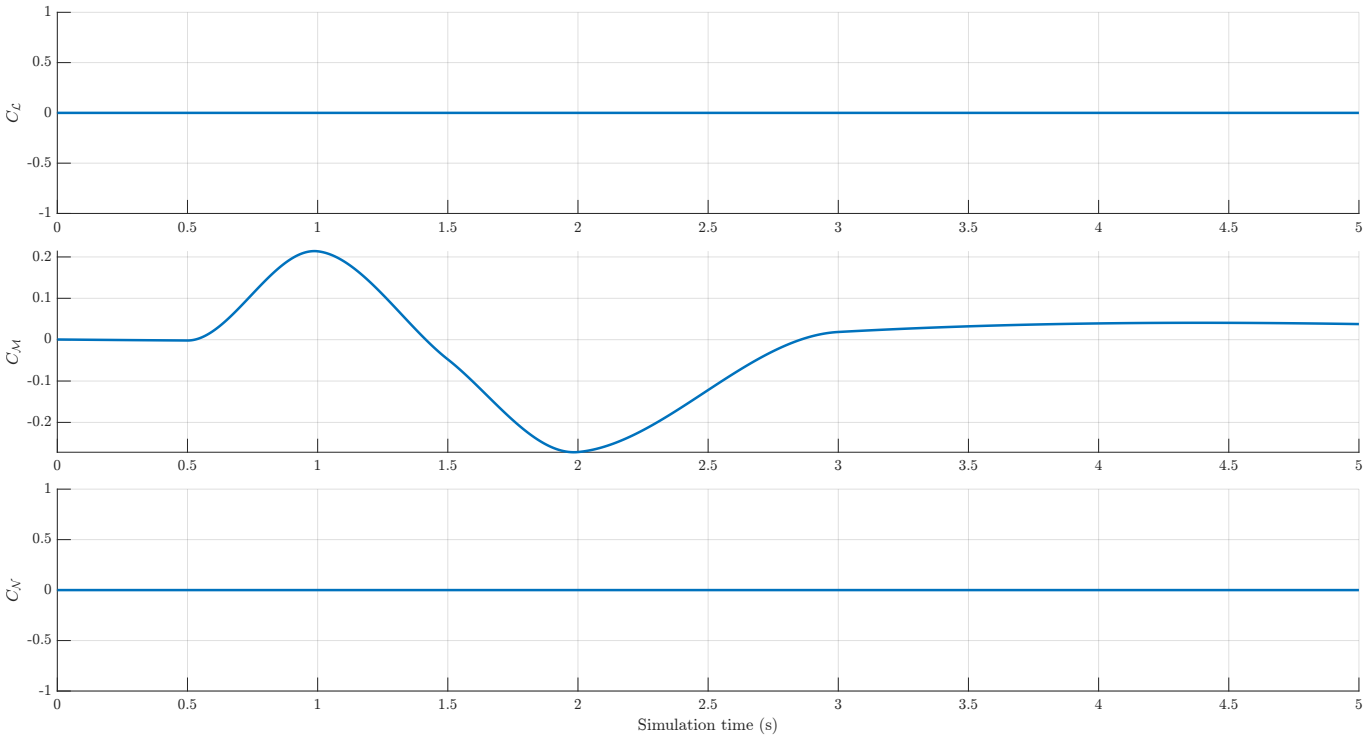


Figure 2.13 Time histories of the aerodynamic moment coefficients.

part $\boldsymbol{x}_{\mathrm{d}}$. In particular,

$$\boldsymbol{x}_{\mathrm{d}} = \begin{Bmatrix} V_G \\ \alpha_{\mathrm{B}} \\ \beta \\ p \\ q \\ r \end{Bmatrix} \tag{2.38}$$

The first three equations of the system (2.33) must be changed, in order to have the proper dynamical system to solve. These equations must be replaced by the one for $\dot{V}_G$, $\dot{\alpha}_{\mathrm{B}}$ and $\dot{\beta}$ hence they must be evaluated. The starting point for this computation are the following relations between u, v, w and V_G, α_{B}, β.

$$u = V_G \cos\beta \cos\alpha_{\mathrm{B}} \tag{2.39}$$

$$v = V_G \sin\beta \tag{2.40}$$

$$w = V_G \cos\beta \sin\alpha_{\mathrm{B}} \tag{2.41}$$

that can be inverted to obtain

$$V_G = \sqrt{u^2 + v^2 + w^2} \tag{2.42}$$

$$\alpha_{\mathrm{B}} = \arctan\frac{w}{u} \tag{2.43}$$

$$\beta = \arcsin\frac{u}{\sqrt{u^2 + v^2 + w^2}} \tag{2.44}$$

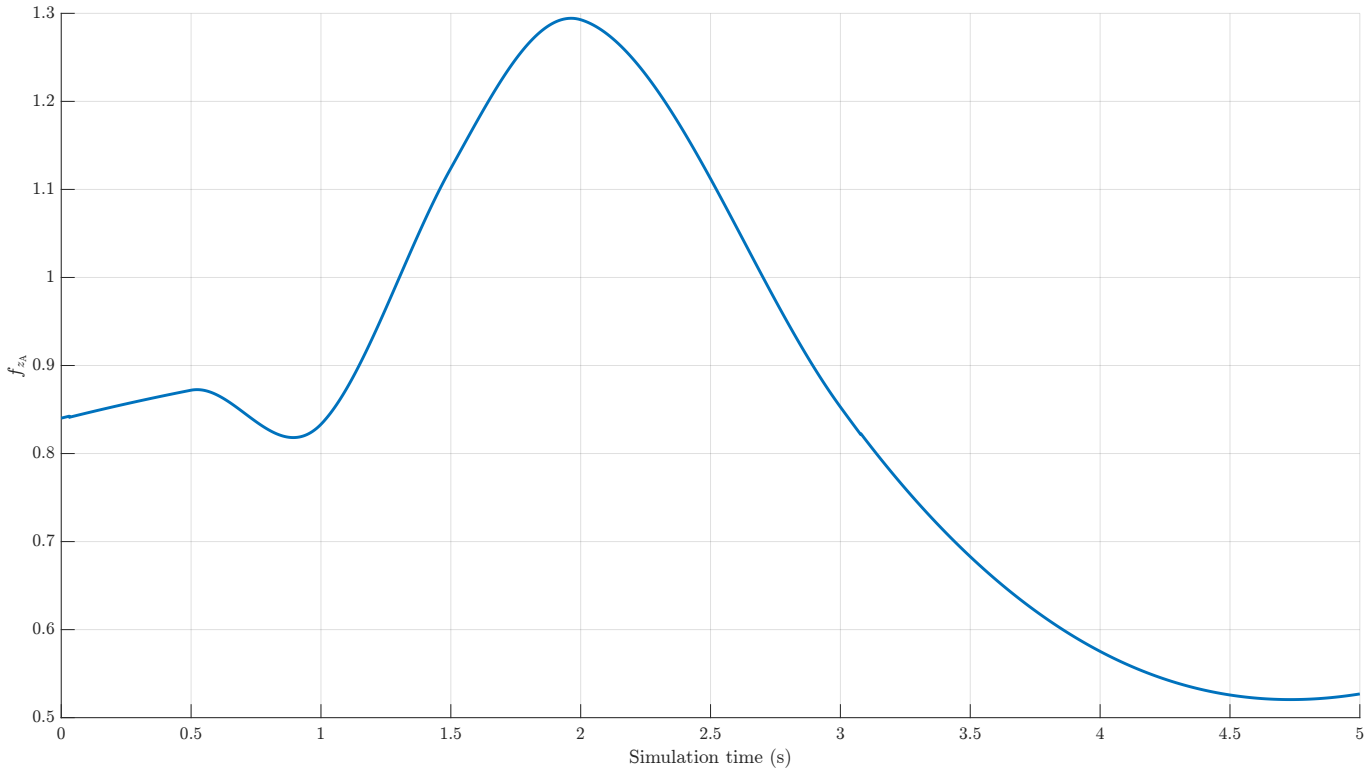


Figure 2.14 Time history of the normal load factor.

By differentiating these relations and using (2.28), together with

$$X = X_T + X_A - W \sin\theta \tag{2.45}$$

$$Y = Y_T + Y_A + W \sin\varphi \cos\theta \tag{2.46}$$

$$Z = Z_T + Z_A + W \cos\varphi \cos\theta \tag{2.47}$$

where

$$X_A = -D \cos\alpha_B + L \sin\alpha_B \tag{2.48}$$

$$Y_A = -Y_A \tag{2.49}$$

$$Z_A = -D \sin\alpha_B - L \cos\alpha_B \tag{2.50}$$

we obtain the following time rate of change of the airspeed and aerodynamic angles

$$\begin{aligned}\dot{V}_G =& \frac{g}{W}\Big[- D\cos\beta + Y_A \sin\beta + X_T \cos\alpha_B \cos\beta + Y_T \sin\beta + Z_T \sin\alpha_B \cos\beta \\ & - W\left(\cos\alpha_B \cos\beta \sin\theta - \sin\beta \sin\varphi \cos\theta - \sin\alpha_B \cos\beta \cos\varphi \cos\theta\right)\Big]\end{aligned} \tag{2.51}$$

$$\begin{aligned}\dot{\alpha}_B =& \frac{g}{W V_G \cos\beta}\Big[- L + Z_T \cos\alpha_B - X_T \sin\alpha_B + W\left(\cos\alpha_B \cos\varphi \cos\theta + \sin\alpha_B \sin\theta\right)\Big] \\ & + q - \tan\beta\left(p\cos\alpha_B + r\sin\alpha_B\right)\end{aligned} \tag{2.52}$$

$$\dot{\beta} = \frac{g}{WV_G}\Big[D\sin\beta + Y_A\cos\beta - X_T\cos\alpha_B\sin\beta + Y_T\cos\beta - Z_T\sin\alpha_B\sin\beta + W\left(\cos\alpha_B\sin\beta\sin\theta + \cos\beta\sin\varphi\cos\theta - \sin\alpha_B\sin\beta\cos\varphi\cos\theta\right)\Big] + p\sin\alpha_B - r\cos\alpha_B \tag{2.53}$$

These equations will be implemented in the following MATLAB routine written with the same idea of the one in listing 2.10.

The main differences from the previous approach are the definition of the output vectors and the declaration of the right hand side of the state propagation equation. In particular, it is set to

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u}) \tag{2.54}$$

that is a more general form of a system of differential equations, since each one of them can be nonlinear.

In fact, equations (2.51), (2.52) and (2.53) are explicitly written and then the right hand side of the dynamical system is assigned as a single column vector.

Listing 2.13 Function for carrying out 6-DoF simulations with V_G, α_B and β state variables.

```
function [time, ...
   delta_T, delta_e_deg, delta_s_deg, delta_a_deg, delta_r_deg, ...
   V, alpha, sideslip, p, q, r, ...
   x_EG, y_EG, z_EG, quat0, quatx, quaty, quatz] = ...
                  SixDoFQuatVab(t_end, state0, ...
                                myAircraft, ...
                                delta_T_law, ...
                                delta_e_deg_law, delta_s_deg_law, ...
                                delta_a_deg_law, delta_r_deg_law)

S = myAircraft.S;
b = myAircraft.b;
c = myAircraft.c;
m = myAircraft.m;
W = myAircraft.W;
I_B = myAircraft.I_B;

omega_B_tilde = @(state) [   0,     -state(6),  state(5); ...
                         state(6),     0,     -state(4); ...
                        -state(5),  state(4),     0    ];

T_EB = @(state) ...
         quat2dcm([state(10), state(11), state(12), state(13)]).';

Quat_evol = @(state) 0.5 * [-state(11), -state(12), -state(13); ...
                            state(10), -state(13),  state(12); ...
                            state(13),  state(10), -state(11); ...
                           -state(12),  state(11),  state(10)];

function rho = density(h)
 [~, ~, ~, rho] = atmosisa(h);
end
rho = @(state) density(-state(9));
function a = sound(h)
 [~, a, ~, ~] = atmosisa(h);
end
a = @(state) sound(-state(9));

u = @(state) state(1) * cos(state(3)) * cos(state(2));
v = @(state) state(1) * sin(state(3));
w = @(state) state(1) * cos(state(3)) * sin(state(2));

airspeed = @(state) state(1);
alpha_deg = @(state) convang(state(2), 'rad', 'deg');
beta_deg = @(state) convang(state(3), 'rad', 'deg');
p_degps = @(state) convangvel(state(4), 'rad/s', 'deg/s');
q_degps = @(state) convangvel(state(5), 'rad/s', 'deg/s');
```

```
r_degps = @(state) convangvel(state(6), 'rad/s', 'deg/s');
altitude = @(state) -state(9);
mach = @(state) airspeed(state) / a(state);

quaternion = @(state) [state(10), state(11), state(12), state(13)];

function psi = quat2psi(quat)
 [psi, ~, ~] = quat2angle(quat);
end
psi = @(state) quat2psi(quaternion(state));

function theta = quat2theta(quat)
 [~, theta, ~] = quat2angle(quat);
end
theta = @(state) quat2theta(quaternion(state));

function phi = quat2phi(quat)
 [~, ~, phi] = quat2angle(quat);
end
phi = @(state) quat2phi(quaternion(state));

InputCommands = @(t) ...
 [delta_T_law(t), ...
 delta_e_deg_law(t), delta_s_deg_law(t), ...
 delta_a_deg_law(t), delta_r_deg_law(t)];

ModelsInputs = @(t, state) ...
 [alpha_deg(state), beta_deg(state), ...
  InputCommands(t), ...
  p_degps(state), q_degps(state), r_degps(state), ...
  altitude(state), airspeed(state)];

ThrustModel = @(t, state) delta_T_law(t) *
    ↪ myAircraft.ThrustModel(altitude(state), mach(state));
DragCoeffModel = @(t, state) myAircraft.DragCoeffModel(ModelsInputs(t,
    ↪ state));
LiftCoeffModel = @(t, state) myAircraft.LiftCoeffModel(ModelsInputs(t,
    ↪ state));
CrossforceCoeffModel = @(t, state)
    ↪ myAircraft.CrossforceCoeffModel(ModelsInputs(t, state));
RollCoeffModel = @(t, state) myAircraft.RollCoeffModel(ModelsInputs(t,
    ↪ state));
PitchCoeffModel = @(t, state) myAircraft.PitchCoeffModel(ModelsInputs(t,
    ↪ state));
YawCoeffModel = @(t, state) myAircraft.YawCoeffModel(ModelsInputs(t,
    ↪ state));

LiftModel = @(t, state) ...
 LiftCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S;

DragModel = @(t, state) ...
 DragCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S;

CrossforceModel = @(t, state) ...
 CrossforceCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S;

Roll = @(t, state) ...
 RollCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S * b;

Pitch = @(t, state) ...
 PitchCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S * c;

Yaw = @(t, state) ...
 YawCoeffModel(t, state) * ...
 0.5 * rho(state) * airspeed(state)^2 * S * b;

Vdot = @(t, state) 1/m * (-DragModel(t, state)*cos(state(3)) + ...
```

```
 CrossforceModel(t, state)*sin(state(3)) + ThrustModel(t,
    ↪ state)*cos(state(2))*cos(state(3)) - ...
 W * (cos(state(2))*cos(state(3))*sin(theta(state)) - ...
      sin(state(3))*sin(phi(state))* ...
                                  cos(theta(state)) - ...
      sin(state(2))*cos(state(3))*cos(phi(state))* ...
                                  cos(theta(state))));

alphadot = @(t,state) 1/(m*state(1)*cos(state(3))) * (-LiftModel(t,
    ↪ state) - ...
     ThrustModel(t, state)*sin(state(2)) + W * (cos(state(2))* ...
     cos(phi(state))*cos(theta(state)) + ...
     sin(state(2))*sin(theta(state)))) + ...
 state(5) - tan(state(3)) * ...
                 (state(4)*cos(state(2)) + state(6)*sin(state(2)));

betadot = @(t, state) 1/(m*state(1)) * (DragModel(t,
    ↪ state)*sin(state(3)) + ...
 CrossforceModel(t, state)*cos(state(3)) - ThrustModel(t,
    ↪ state)*cos(state(2))*sin(state(3)) + ...
 W * (cos(state(2))*sin(state(3))*sin(theta(state)) + ...
      cos(state(3))*sin(phi(state))* ...
                                  cos(theta(state)) - ...
      sin(state(2))*sin(state(3))*cos(phi(state))* ...
                                  cos(theta(state)))) + ...
 state(4)*sin(state(2)) - state(6)*cos(state(2));

dstate_dt = @(t, state) ...
 [ Vdot(t, state); ...
   alphadot(t, state); ...
   betadot(t, state); ...
   I_B \ ([ Roll(t, state); ...
            Pitch(t, state); ...
            Yaw(t, state) ] - ...
              omega_B_tilde(state) * I_B * state(4:6)); ...
   T_EB(state) * [ u(state); ...
                   v(state); ...
                   w(state)]; ...
   Quat_evol(state) * state(4:6) ];

ODEoptions = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);

[time, state] = ode45(dstate_dt, [0, t_end], state0, ODEoptions);

delta_T = delta_T_law(time);          delta_T = delta_T(:);
delta_e_deg = delta_e_deg_law(time); delta_e_deg = delta_e_deg(:);
delta_s_deg = delta_s_deg_law(time); delta_s_deg = delta_s_deg(:);
delta_a_deg = delta_a_deg_law(time); delta_a_deg = delta_a_deg(:);
delta_r_deg = delta_r_deg_law(time); delta_r_deg = delta_r_deg(:);
V = state(:, 1);
alpha = state(:, 2);
sideslip = state(:, 3);
p = state(:, 4);
q = state(:, 5);
r = state(:, 6);
x_EG = state(:, 7);
y_EG = state(:, 8);
z_EG = state(:, 9);
quat0 = state(:, 10);
quatx = state(:, 11);
quaty = state(:, 12);
quatz = state(:, 13);

end
```

The above discussed setting of the flight dynamics and simulation will be the most used from now on, since it gives the useful flight parameters airspeed and aerodynamic angles as state variables solutions.

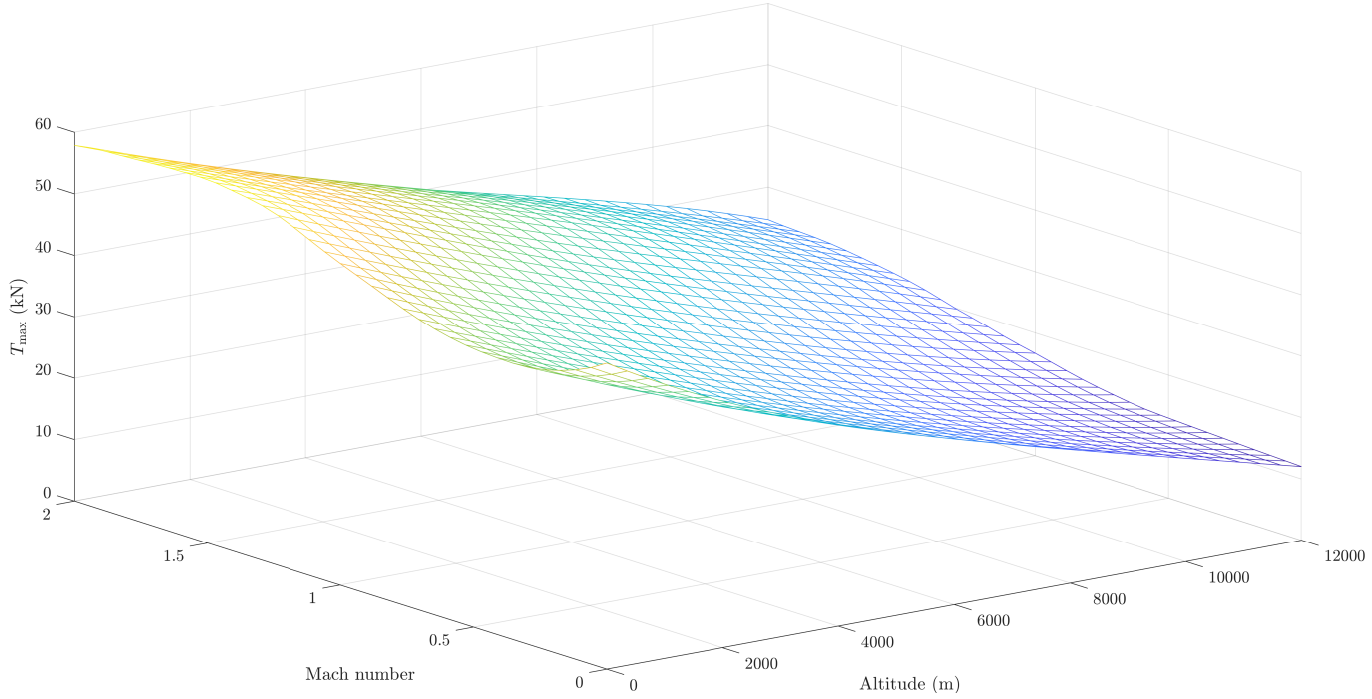


Figure 2.15 Maximum available thrust with respect to altitude and Mach number.

6-DoF simulation with an appropriate Thrust model

In this exercise has been carried out a simulation of the same aircraft of the F/A-18 HARV exercise in § 2.3.1, but with a significal change in the thrust model. An example of a more accurate thrust model is shown here. It has the expected dependencies on altitude and Mach number, as shown by the following data.

```
--- File 1          --- File 2          --- File 3          --- File 4          --- File 5
h = 0 m             h = 3000 m          h = 6000 m          h = 9000 m          h = 12000 m
Mach, T (kgf)       Mach, T (kgf)       Mach, T (kgf)       Mach, T (kgf)       Mach, T (kgf)
0.00311, 3081.6     0.003, 2244.8       0.00040, 1669.5     0.00088, 1192.4     0.00192, 742.7
0.09430, 2945.1     0.09787, 2189.9     0.09845, 1627.1     0.10046, 1158.5     0.10064, 752.6
0.19748, 2846.2     0.19923, 2147.5     0.20058, 1609.9     0.20008, 1141.2     0.20103, 764.6
0.30306, 2791.4     0.30470, 2134.5     0.30439, 1594.7     0.30637, 1136.6     0.30643, 778.8
0.39679, 2788.7     0.39758, 2138.1     0.39894, 1598.3     0.40009, 1140.2     0.40096, 790.8
0.49800, 2807.1     0.49959, 2171.1     0.49847, 1616.6     0.49962, 1160.5     0.49967, 804.9
0.59997, 2852.6     0.60239, 2225.0     0.60127, 1668.4     0.60246, 1195.6     0.60256, 823.2
0.70438, 2929.6     0.70431, 2293.5     0.70655, 1730.8     0.70611, 1241.2     0.70622, 862.5
0.79954, 3025.2     0.80118, 2372.4     0.80343, 1807.6     0.80300, 1313.9     0.80486, 905.9
0.89968, 3135.5     0.90136, 2466.1     0.90197, 1890.7     0.90325, 1380.3     0.90178, 966.0
0.99816, 3241.7     0.99983, 2576.4     1.00046, 1990.6     1.00178, 1467.6     1.00371, 1030.4
1.09828, 3360.4     1.09911, 2695.1     1.10229, 2094.7     1.10363, 1563.4     1.10398, 1090.5
1.20005, 3487.6     1.20425, 2816.0     1.20409, 2211.4     1.20713, 1667.5     1.20672, 1165.4
1.29774, 3577.0     1.29937, 2924.2     1.30010, 2302.9     1.30146, 1761.0     1.29944, 1231.7
1.39469, 3624.5     1.39625, 3003.2     1.39776, 2400.6     1.39911, 1863.0     1.40050, 1308.6
1.49167, 3659.5     1.49233, 3065.3     1.49377, 2490.0     1.49433, 1935.6     1.49742, 1370.8
1.59042, 3659.0     1.59101, 3089.9     1.59152, 2554.3     1.59292, 1995.7     1.59269, 1418.3
1.68918, 3650.0     1.69056, 3101.9     1.69188, 2576.8     1.69242, 2028.7     1.69301, 1459.6
1.79049, 3630.7     1.79270, 3084.7     1.79398, 2574.2     1.79529, 2051.2     1.79502, 1492.6
1.89178, 3617.7     1.89152, 3052.8     1.89530, 2548.6     1.89573, 2044.4     1.89540, 1508.8
1.98558, 3587.8     1.98957, 2995.7     1.98915, 2497.8     1.99028, 2045.9     1.98992, 1525.0
```

The maximum available thrust force $T_{max} = T_{max}(h, M)$ is obtained from the data shown above by a 2D spline interpolation using MATLAB `interp2` function. Figure 2.15 shows its resulted diagram. See also figures 2.16 and 2.17 for more detailed representa-

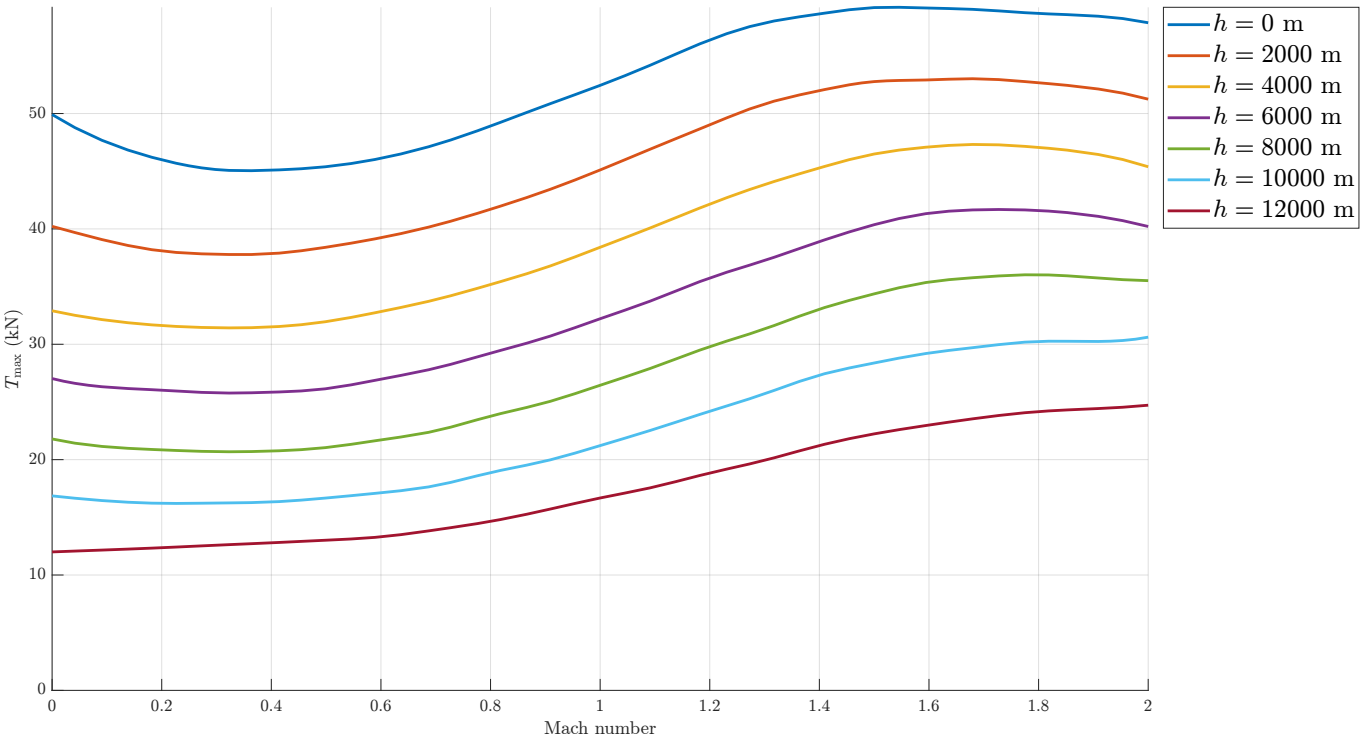


Figure 2.16 Maximum available thrust with respect to the Mach number for different assigned altitude values.

tions.

It is worth noticing that for different fixed Mach number values the T_{max} curve is approximately linear and its slope does not vary much with h or M. On the other hand, for different fixed altitude values the dependency of T_{max} on the Mach number is no more easily predictable. At most, it can be approximated with a globally increasing function of M that decreases when the altitude increases.

These considerations suggest that, given some chosen assumptions on the dependencies of the T_{max} curve and on its slope, a simpler model can be written as typically done with the aerodynamic coefficients models.

The following MATLAB script I wrote tests a simulation using this data-driven thrust model just introduced. The simulation is obtained from the following initial conditions.

- $M_0 = 0.80$
- $\alpha_{B,0} = 9.5$ deg and $\beta_0 = 0$ deg
- $p_0 = 0$ deg/s, $p_0 = 0$ deg/s and $r_0 = 0$ deg/s
- $x_{E,G,0} = 0$ m, $y_{E,G,0} = 0$ m and $z_{E,G,0} = -8000$ m
- $\varphi_0 = 0$ deg, $\theta_0 = -2.5$ deg and $\psi_0 = 0$ deg

and assigning the input commands time laws shown in figure 2.18.

Listing 2.14 F/A-18 HARV 6-DoF simulation with a non-trivial thrust model.

```
close all; clear; clc

    myAircraft = load('f18harvthrust.mat');

    ModelsInputs = zeros(1, 11);
```

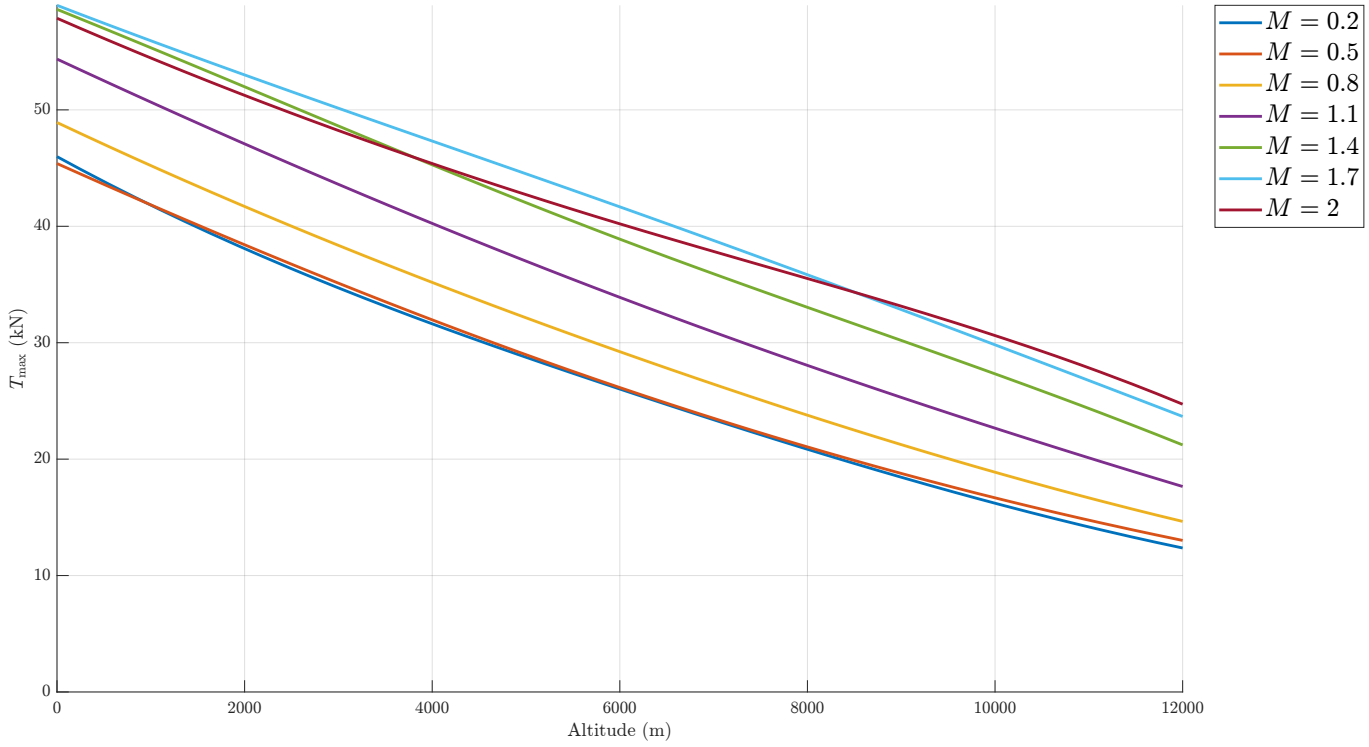


Figure 2.17 Maximum available thrust with respect to the altitude for different assigned Mach numbers.

```
delta_e_deg_cases = -20 : 5 : 10;

liftCharacteristicsPlot(myAircraft, ModelsInputs, [-5, 40], ...
                delta_e_deg_cases,
↪ 'exf18harvthrustliftcharacteristics.pdf')

staticPolarPlot(myAircraft, ModelsInputs, [-5, 40], ...
                delta_e_deg_cases,
↪ 'exf18harvthrustdragcharacteristics.pdf')

q_degps_cases = linspace(-10, 15, 6);
pitchCharacteristicsPlot(myAircraft, ModelsInputs, [-5, 40], ...
                    delta_e_deg_cases, q_degps_cases, ...
                    'exf18harvthrustpitchcharacteristics.pdf')

h_cases = 0 : 2000 : 12000;
Mach_cases = 0.20 : 0.30 : 2.0;
thrustCharacteristicsPlot(myAircraft, [0, 12000], [0, 2], ...
                          h_cases, Mach_cases, ...

↪ 'exf18harvthrustthrustcharacteristics1.pdf', ...

↪ 'exf18harvthrustthrustcharacteristics2.pdf', ...
                          'exf18harvthrustthrustcharacteristics3.pdf')

t_end = 10;

x_EG0 = 0;
y_EG0 = 0;
z_EG0 = -8000;

Mach0 = 0.8;
[temp0, sound0, press0, dens0] = atmosisa(-z_EG0);
V0 = Mach0 * sound0;

phi0 = 0;
theta0 = convang(-2.5, 'deg', 'rad');
psi0 = 0;
Quat0 = angle2quat(psi0, theta0, phi0);
```

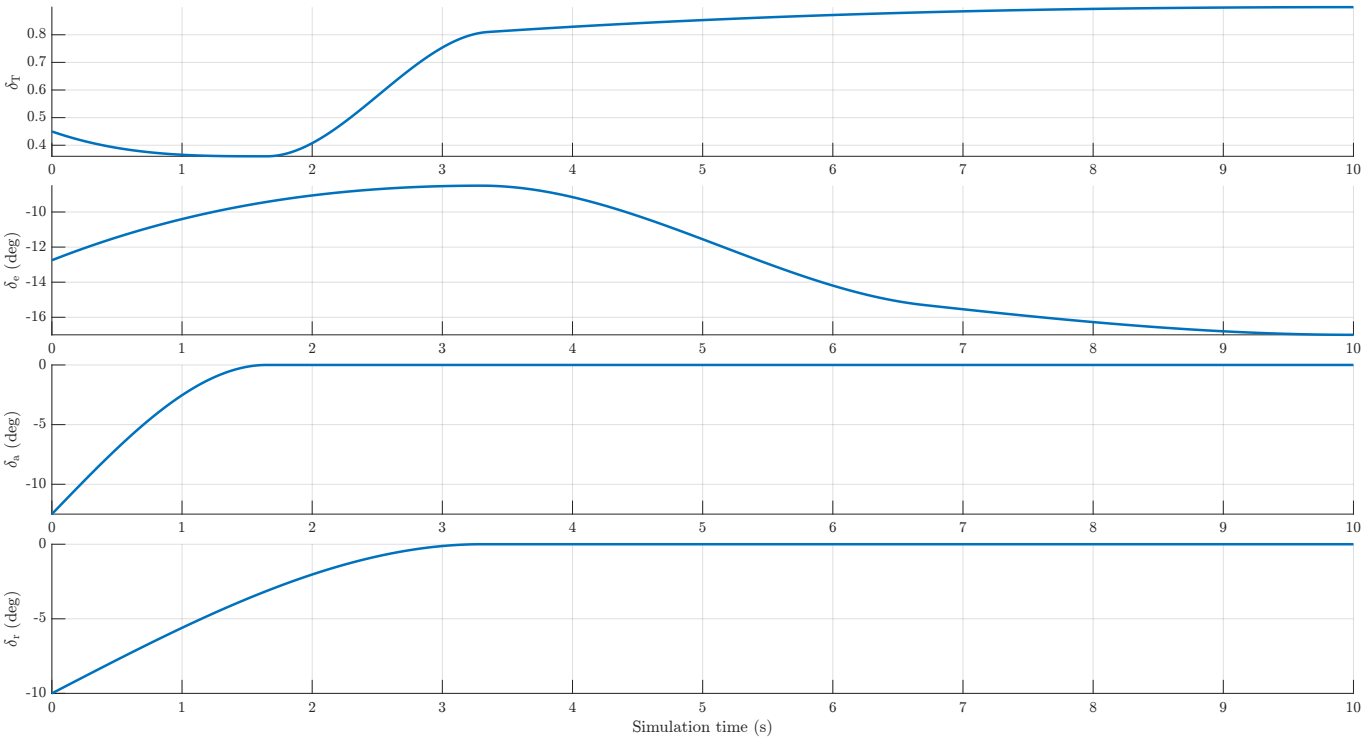


Figure 2.18 Time histories of the assigned input commands laws.

```
41
42  alpha0_deg = 9.5; alpha0 = convang(alpha0_deg, 'deg', 'rad');
43  beta0_deg = 0; beta0 = convang(beta0_deg, 'deg', 'rad');
44
45  p0 = 0;
46  q0 = 0;
47  r0 = 0;
48
49  state0 = [V0, alpha0, beta0, p0, q0, r0, x_EG0, y_EG0, z_EG0,
    ↪ Quat0].';
50
51  delta_T0 = 0.45;
52  delta_T_law = @(t) interp1([0, 0.33, 0.67, 2] * t_end/2, ...
53                             [1, 0.8, 1.8, 2] * delta_T0, ...
54                              t, 'pchip');
55
56  delta_e0_deg = -8.5;
57  delta_e_deg_law = @(t) interp1( ...
58                  [0, 0.33, 0.67,  1] * t_end, ...
59                  [1.5, 1, 1.8,  2] * delta_e0_deg, ...
60                   t, 'pchip');
61
62  delta_s0_deg = 0;
63  delta_s_deg_law = @(t) interp1([0, 1] * t_end, ...
64                                 [1, 1] * delta_s0_deg, ...
65                                  t, 'linear');
66
67  delta_a0_deg = -12.5;
68  delta_a_deg_law = @(t) interp1([0, 0.33, 0.67, 2] * t_end/2, ...
69                                 [1, 0, 0, 0] * delta_a0_deg, ...
70                                  t, 'pchip');
71
72  delta_r0_deg = -10;
73  delta_r_deg_law = @(t) interp1([0, 0.33, 0.67, 1] * t_end, ...
74                                 [1, 0, 0, 0] * delta_r0_deg, ...
75                                  t, 'pchip');
76
77  [time, ...
78   delta_T, delta_e_deg, delta_s_deg, delta_a_deg, delta_r_deg, ...
```

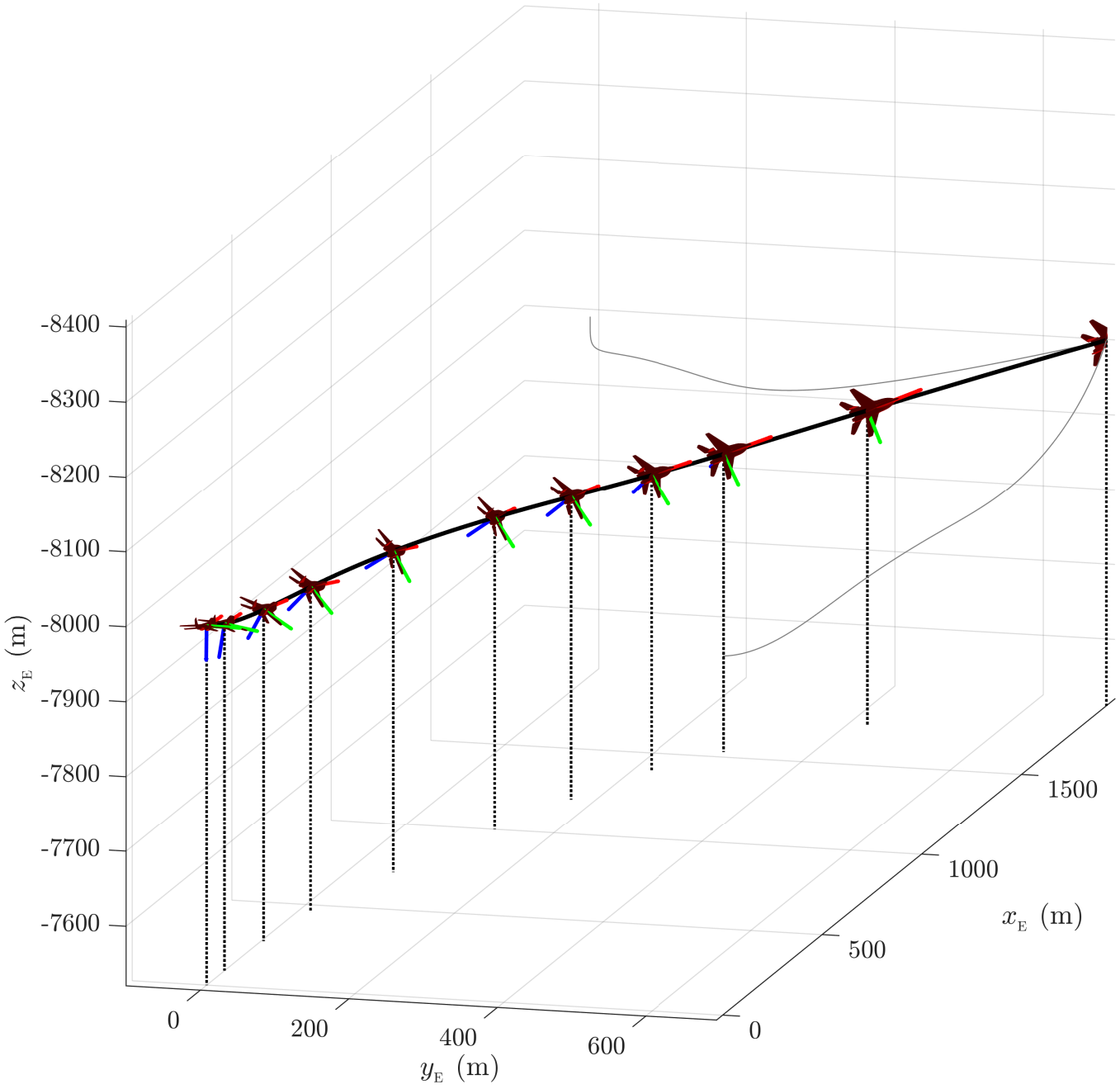


Figure 2.19 3D visualization of the simulated trajectory.

```
    V, alpha, beta, p, q, r, ...
    x_EG, y_EG, z_EG, quat0, quatx, quaty, quatz] = ...
        SixDoFQuatVab(t_end, state0, ...
                      myAircraft, ...
                      delta_T_law, ...
                      delta_e_deg_law, delta_s_deg_law, ...
                      delta_a_deg_law, delta_r_deg_law);

Nt = length(time);

stackedPlot4(time, ...
             delta_T, delta_e_deg, delta_a_deg, delta_r_deg, ...
             {"Simulation time (s)", ...
              "$\delta_\mathrm{T}$", "$\delta_\mathrm{e}$ (deg)", ...
              "$\delta_\mathrm{a}$ (deg)", "$\delta_\mathrm{r}$
↪ (deg)"}, ...
             {}, {}, 'exf18harvthrustinputcommands.pdf')

```

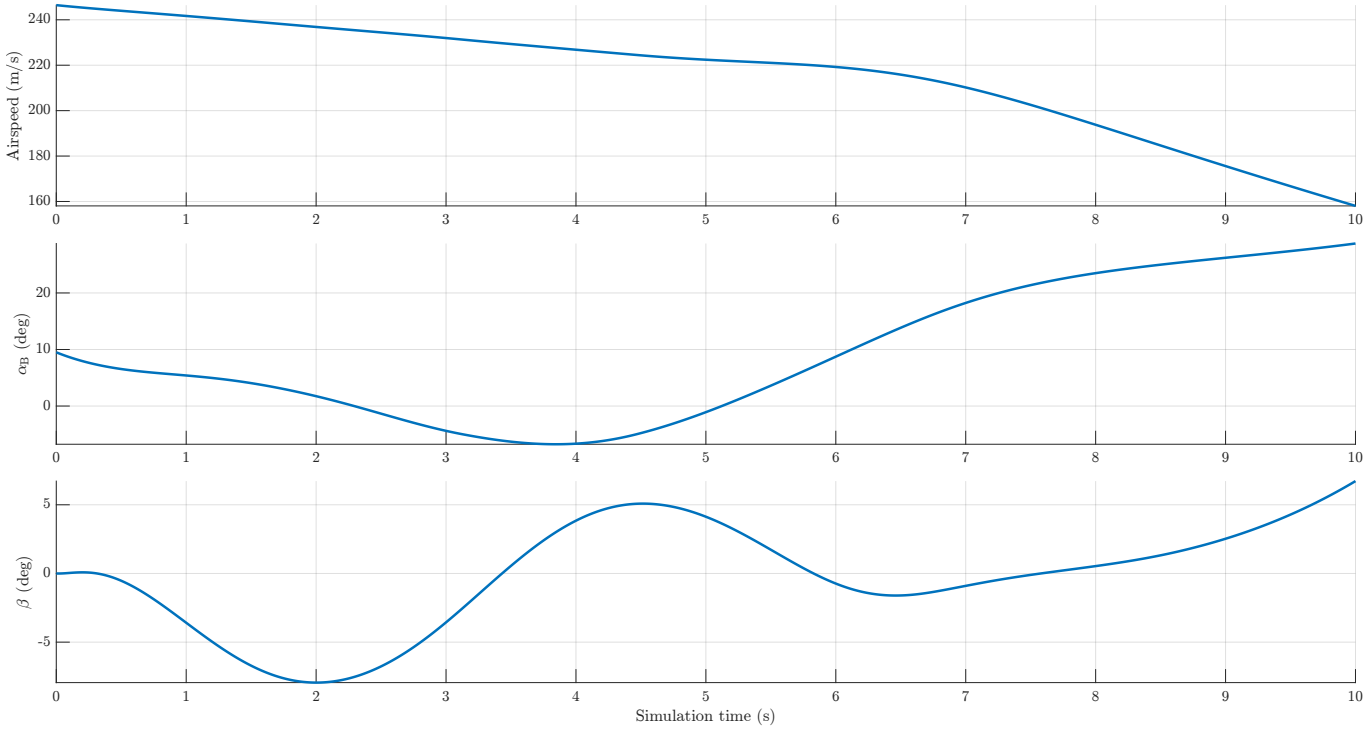


Figure 2.20 Time histories of the simulated airspeed and aerodynamic angles.

```
alpha_deg = convang(alpha, 'rad', 'deg');
beta_deg = convang(beta, 'rad', 'deg');
stackedPlot3(time, ...
             V, alpha_deg, beta_deg, ...
             {"Simulation time (s)", ...
              "Airspeed (m/s)", "$\alpha_\mathrm{B}$ (deg)", "$\beta$ (deg)"}, ...
             {}, {}, 'exf18harvthrustVab.pdf')

p_degps = convangvel(p, 'rad/s', 'deg/s');
q_degps = convangvel(q, 'rad/s', 'deg/s');
r_degps = convangvel(r, 'rad/s', 'deg/s');
stackedPlot3(time, ...
             p_degps, q_degps, r_degps, ...
             {"Simulation time (s) ", ...
              "$p$ (deg/s)", "$q$ (deg/s)", "$r$ (deg/s)"}, ...
             {}, {}, 'exf18harvthrustangvelcomponents.pdf')

altitude = -z_EG;
stackedPlot3(time, ...
             x_EG, y_EG, altitude, ...
             {"Simulation time (s)", ...
              "$x_{\mathrm{E},G}$ (m)", "$y_{\mathrm{E},G}$ (m)", "Altitude (m)"}, ...
             {}, {}, 'exf18harvthrustcogxyh.pdf')

[psi, theta, phi] = quat2angle([quat0, quatx, quaty, quatz]);
phi_deg = convang(phi, 'rad', 'deg');
theta_deg = convang(theta, 'rad', 'deg');
psi_deg = convang(psi, 'rad', 'deg');
stackedPlot3(time, ...
             phi_deg, theta_deg, psi_deg, ...
             {"Simulation time (s)", ...
              "$\varphi$ (deg)", "$\theta$ (deg)", "$\psi$ (deg)"}, ...
             {}, {}, 'exf18harvthrusteulangles.pdf')

ModelsInputs = [alpha_deg, beta_deg, ...
                delta_T, delta_e_deg, delta_s_deg, delta_a_deg, delta_r_deg, ...
```

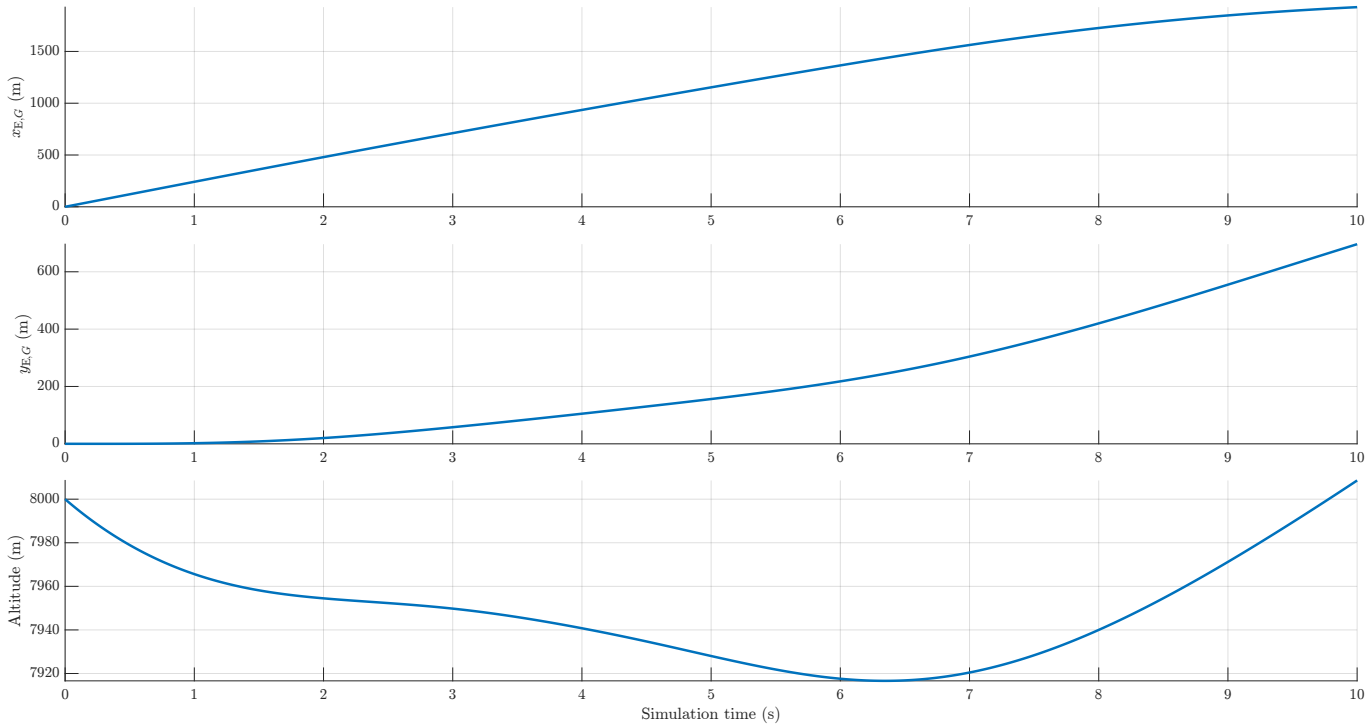


Figure 2.21 Center of gravity simulated components over time.

```
                    p_degps, q_degps, r_degps, ...
                    altitude, V];
    CL = zeros(Nt, 1);
    CD = CL;
    CC = CL;
    CRoll = CL;
    CPitch = CL;
    CYaw = CL;
    Thrust = CL;
    [~, sound, ~, ~] = atmosisa(altitude);
    Mach = V ./ sound;
    for it = 1 : Nt
        CL(it) = myAircraft.LiftCoeffModel(ModelsInputs(it, :));
        CD(it) = myAircraft.DragCoeffModel(ModelsInputs(it, :));
        CC(it) = myAircraft.CrossforceCoeffModel(ModelsInputs(it, :));
        CRoll(it) = myAircraft.RollCoeffModel(ModelsInputs(it, :));
        CPitch(it) = myAircraft.PitchCoeffModel(ModelsInputs(it, :));
        CYaw(it) = myAircraft.YawCoeffModel(ModelsInputs(it, :));
        Thrust(it) = myAircraft.ThrustModel(altitude(it), Mach(it));
    end
    stackedPlot3(time, ...
                 CL, CD, CC, ...
                 {"Simulation time (s)", ...
                  "$C_L$", "$C_D$", "$C_C$"}, ...
                 {}, {}, 'exf18harvthrustaeroforcecoeffhistories.pdf')
    stackedPlot3(time, ...
                 CRoll, CPitch, CYaw, ...
                 {"Simulation time (s)", ...
                  "$C_\mathcal{L}$", "$C_\mathcal{M}$",
    ↪ "$C_\mathcal{N}$"}, ...
                 {}, {}, 'exf18harvthrustaeromomcoeffhistories.pdf')
    stackedPlot2(time, ...
                 Mach, 1e-3*Thrust, ...
                 {"Simulation time (s)", ...
                  "Mach number", "Thrust (kN)"}, ...
                 {}, {}, 'exf18harvthrustmachthrust.pdf')

    u = V .* cos(beta) .* cos(alpha);
    v = V .* sin(beta);
    w = V .* cos(beta) .* sin(alpha);
```

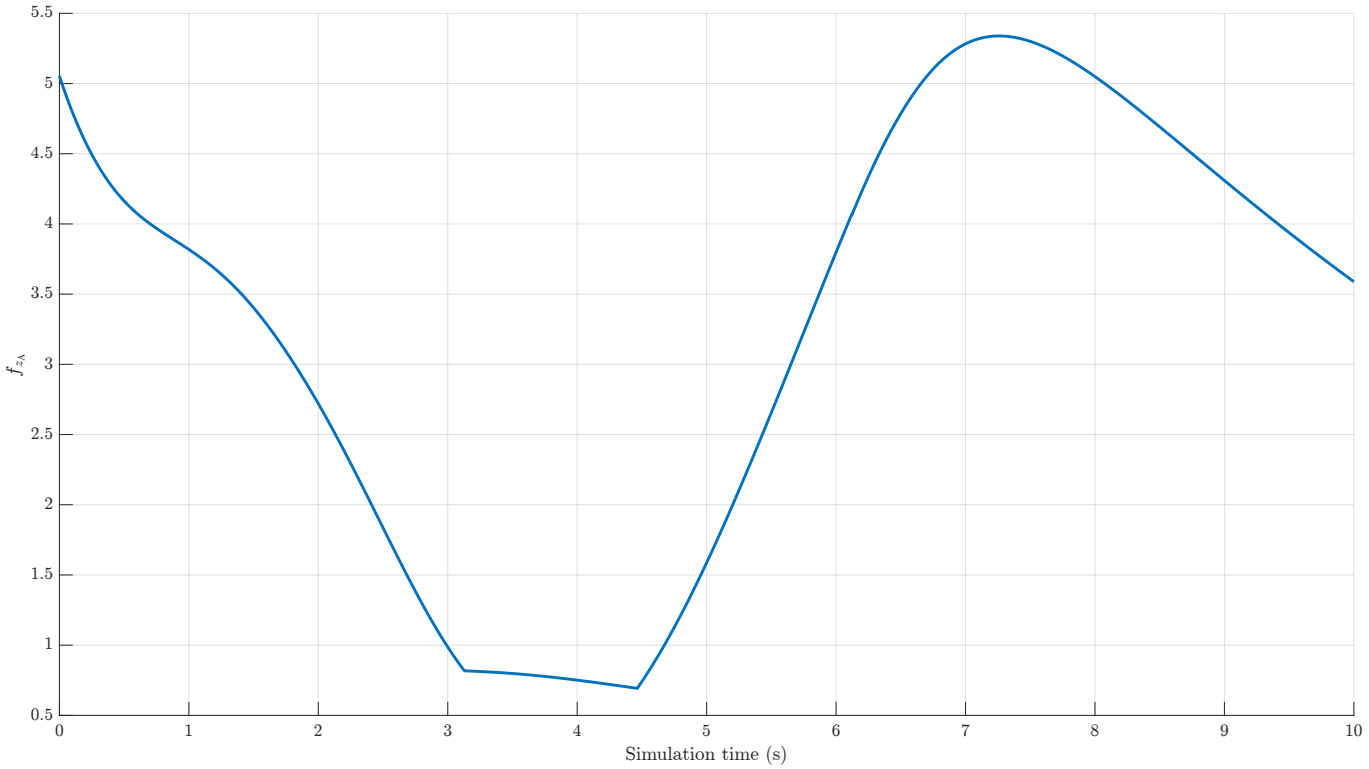


Figure 2.22 Time history of the normal load factor.

```
[f_xA, f_yA, f_zA] = LoadFactor(time, u, v, w, alpha_deg, quat0,
↪ quatx, quaty, quatz);
multiPlot(time, ...
          "Simulation time (s)", "$f_{z_\mathrm{A}}$", ...
          {}, ...
          'exf18harvthrustloadfactor.pdf', ...
          f_zA)

Thrust = zeros(Nt, 1);
[~, sound, ~, ~] = atmosisa(altitude);
Mach = V ./ sound;
for it = 1 : Nt
    Thrust(it) = myAircraft.ThrustModel(altitude(it), Mach(it));
end

trajectoryView(time, ...
               x_EG, y_EG, z_EG, ...
               phi, theta, psi, ...
               60, 11, ...
               'exf18harvthrusttrajectory.pdf')
```

As expected, the time history of the thrust force (see figure 2.23) matches the dependencies of the propulsion model. In particular, the thrust decreases by about 2.75 kN despite the increment in the throttle. Since the altitude is approximately constant throughout the simulation, this happens because of the rapid decrease in the Mach number during the simulation, which causes the drop in the thrust.

2.4 3-DoF dynamical system

The three degrees of freedom simplification of the aircraft motions is particularly useful when dealing with *Longitudinal-and-Symmetric flight*. In this scenario, the aircraft plane of symmetry is always vertical and the trajectory of its center of gravity completely

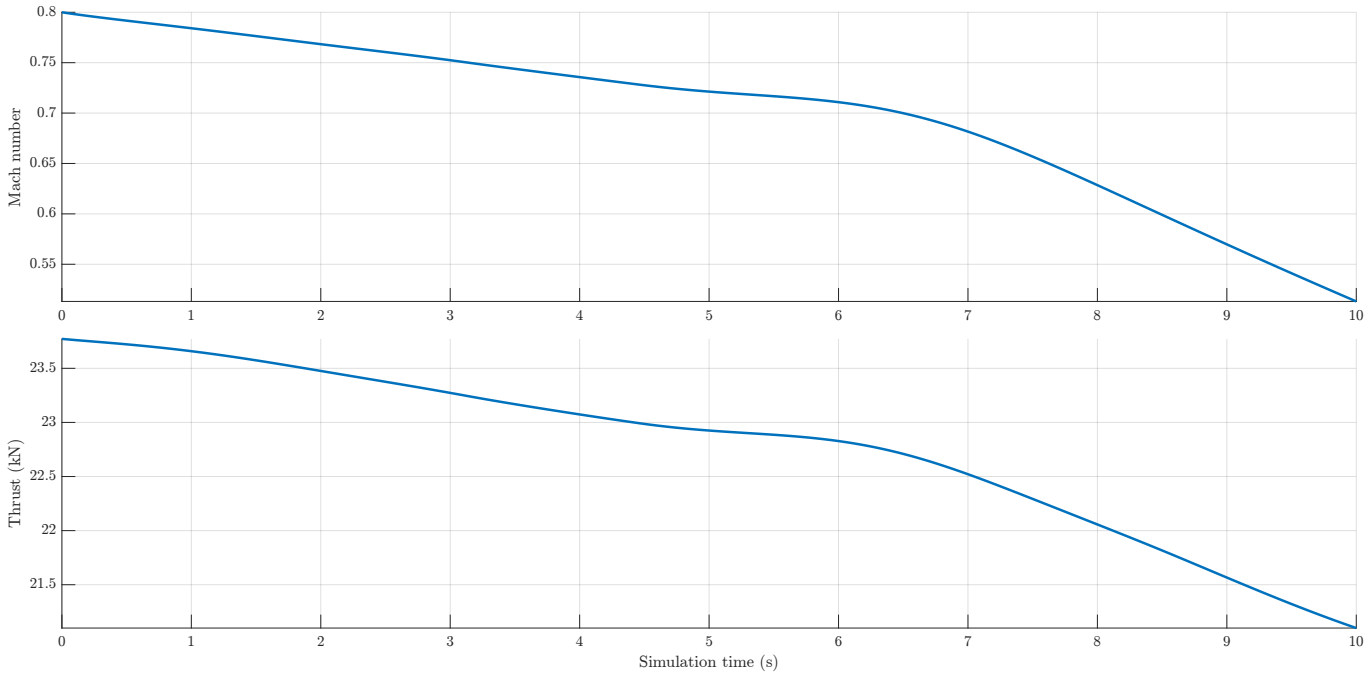


Figure 2.23 Time histories of the simulated Mach number and thrust force.

lays in it.

Under these assumptions there are conclusions that can be stated upon some state variables. In particular, the bank angle must be zero in each instant of time and the heading angle must not change with time. We write

$$\varphi(t) = 0 \tag{2.55}$$

$$\psi(t) = \text{const.} \equiv 0 \tag{2.56}$$

where ψ is conventionally set to be zero.

In addition, to ensure such simplified motion this orientation must not change. This implies that the related velocity component, angular velocity components, force component and moment components must all be zero. In particular,

$$\beta(t) = 0 \tag{2.57}$$

$$v(t) = 0 \tag{2.58}$$

$$p(t) = 0 \tag{2.59}$$

$$r(t) = 0 \tag{2.60}$$

$$Y(t) = 0 \tag{2.61}$$

$$\mathscr{L}(t) = 0 \tag{2.62}$$

$$\mathscr{N}(t) = 0 \tag{2.63}$$

that trivially results in

$$\dot{\beta} = \dot{\varphi} = \dot{\psi} = 0 \tag{2.64}$$

$$\dot{p} = \dot{r} = 0 \tag{2.65}$$

$$\dot{y}_{E,G} = 0 \tag{2.66}$$

and in specific constraints on the input commands time laws. The ailerons and the rudder must not be deflected not to induce any undesired force or moment

$$\delta_a(t) = \delta_r(t) = 0 \tag{2.67}$$

and in case of a multi-engine aircraft all of the engine throttles must be equals to the assigned δ_T time law.

All these assumptions significantly reduce the number of unknowns we have, hence the number of differential equations to integrate. Now, the state vector

$$\boldsymbol{x} = \begin{Bmatrix} \boldsymbol{x}_d \\ \boldsymbol{x}_k \end{Bmatrix} \tag{2.68}$$

is a smaller vector, given that

$$\boldsymbol{x}_d = \begin{Bmatrix} V_G \\ \alpha_B \\ q \end{Bmatrix} \tag{2.69}$$

$$\boldsymbol{x}_k = \begin{Bmatrix} x_{E,G} \\ z_{E,G} \\ \theta \end{Bmatrix} \tag{2.70}$$

where the last state variable is such that $\dot{\theta} = q$ and it is not the elevation angle as previously defined because for this model it may take values from 0 to 2π. Given its unchanged geometrical meaning, the same symbol as the elevation angle is used.

In the same way, because of equations (2.67), the input vector is smaller than it is for the six degrees of freedom model. In particular,

$$\boldsymbol{u} = \begin{Bmatrix} \delta_T \\ \delta_e \\ \delta_s \end{Bmatrix} \tag{2.71}$$

The new set of differential equations which constitutes the three degrees of freedom dynamical system can be obtained from equations (2.51) and (2.52) and from the fifth, seventh and ninth rows of the system of equations (2.33) after applying all the conditions discussed above. In addition, the orientation angle equation is taken as last equation of the system. We obtain

$$\dot{V}_G = \frac{g}{W}\left[-D + X_T \cos\alpha_B + Z_T \sin\alpha_B - W(\cos\alpha_B \sin\theta - \sin\alpha_B \cos\theta)\right] \tag{2.72}$$

$$\dot{\alpha}_B = \frac{g}{V_G W}\left[-L + Z_T \cos\alpha_B - X_T \sin\alpha_B + W(\cos\alpha_B \cos\theta + \sin\alpha_B \sin\theta)\right] \tag{2.73}$$

$$\dot{q} = \frac{\mathcal{M}}{I_{yy}} \tag{2.74}$$

$$\dot{x}_{E,G} = V_G \cos\alpha_B \cos\theta + V_G \sin\alpha_B \sin\theta \tag{2.75}$$

$$\dot{z}_{E,G} = -V_G \cos\alpha_B \sin\theta + V_G \sin\alpha_B \cos\theta \tag{2.76}$$

$$\dot{\theta} = q \tag{2.77}$$

that can be rewritten in the already known form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u})$$

Soon it will be implemented in MATLAB for a symmetric flight simulation. From this system of equations, for different choices of the propulsion model and aerodynamic models, it is possible to obtain specific flight dynamical models.

A simple yet useful one which is frequently used for three degrees of freedom simulations can be obtained from the linearization of the aerodynamic models. In particular, given the aerodynamic loads evaluated as usual

$$L = C_L \frac{1}{2}\rho V_G^2 S \tag{2.78}$$

$$D = C_D \frac{1}{2}\rho V_G^2 S \tag{2.79}$$

$$\mathscr{M} = C_{\mathscr{M}} \frac{1}{2}\rho V_G^2 S \bar{c} \tag{2.80}$$

we can assume a linear formulation for C_L, C_D and $C_{\mathscr{M}}$. In particular, these are the following

$$C_L = C_{L_\alpha}\alpha + \left(C_{L_{\dot{\alpha}}}\dot{\alpha} + C_{L_q} q\right)\frac{\bar{c}}{2V_G} + C_{L_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{L_{\delta_\mathrm{s}}}\delta_\mathrm{s} \tag{2.81}$$

$$C_D = C_{D_0} + kC_L^m \approx C_{D_0} + k\left(C_{L_\alpha}\alpha + C_{L_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{L_{\delta_\mathrm{s}}}\delta_\mathrm{s}\right)^m \tag{2.82}$$

$$C_{\mathscr{M}} = C_{\mathscr{M}_0} + C_{\mathscr{M}_\alpha}\alpha + \left(C_{\mathscr{M}_{\dot{\alpha}}}\dot{\alpha} + C_{\mathscr{M}_q} q\right)\frac{\bar{c}}{2V_G} + C_{\mathscr{M}_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{\mathscr{M}_{\delta_\mathrm{s}}}\delta_\mathrm{s} \tag{2.83}$$

where k is a parameter that depends on mach and in subsonic condition it is

$$k \approx \frac{1}{\pi \mathscr{A} e} \tag{2.84}$$

and m is a parameter approximately constant that depends on the aerodynamic of the aircraft. Typically $m \approx 2$.

These chosen models can be put into equations (2.78), (2.79) and (2.80) and then the resulted loads into equations (2.72) to (2.77) to obtain the following system of differential equations.

$$\begin{aligned}\dot{V}_G = &-\frac{1}{2}\frac{S}{W}\rho V_G^2 g\left[C_{D_0} + k\left(C_{L_\alpha}\alpha + C_{L_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{L_{\delta_\mathrm{s}}}\delta_\mathrm{s}\right)^m\right] \\ &+ \frac{T}{W} g\cos(\alpha - \mu_x - \mu_\mathrm{T}) + g\sin(\alpha - \mu_x - \theta)\end{aligned} \tag{2.85}$$

$$\begin{aligned}\dot{\alpha} = \frac{1}{1 + \frac{\bar{c}/b}{4\mu}C_{L_{\dot{\alpha}}}}\Bigg\{ &-\frac{1}{2}\frac{S}{W}\rho V_G g\left[C_{L_\alpha}\alpha + C_{L_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{L_{\delta_\mathrm{s}}}\delta_\mathrm{s}\right] + q\left(1 - \frac{\bar{c}/b}{4\mu}C_{L_q}\right) \\ &- \frac{T}{W}\frac{g}{V_G}\sin(\alpha - \mu_x - \mu_\mathrm{T}) + \frac{g}{V_G}\cos(\alpha - \mu_x - \theta)\Bigg\}\end{aligned} \tag{2.86}$$

$$M_{32}\dot{\alpha} + \dot{q} = \frac{V_G^2}{\kappa_y^2}\frac{\bar{c}/b}{2\mu}\left[C_{\mathscr{M}_0} + C_{\mathscr{M}_\alpha}\alpha + C_{\mathscr{M}_q}\frac{q\bar{c}}{2V_G} + C_{\mathscr{M}_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{\mathscr{M}_{\delta_\mathrm{s}}}\delta_\mathrm{s} + C_{\mathscr{M},\mathrm{T}}\right] \tag{2.87}$$

$$\dot{x}_{\mathrm{E},G} = V_G \cos(\alpha - \mu_x - \theta) \tag{2.88}$$

$$\dot{z}_{\mathrm{E},G} = V_G \sin(\alpha - \mu_x - \theta) \tag{2.89}$$

$$\dot{\theta} = q \tag{2.90}$$

where κ_y is the aircraft *gyration radius* in the y_B direction, μ_T is the angle between the thrust line and the fuselage reference line, $\mu = \mu(z_{\mathrm{E},G})$ is the *relative density* of the vehicle, defined as

$$\mu = \frac{W}{S}\frac{1}{\rho b g} \tag{2.91}$$

and $C_{\mathscr{M},\mathrm{T}}$ is the coefficient of the pitching moment due to the propulsion of the aircraft. In this model it can be evaluated as

$$C_{\mathscr{M},\mathrm{T}} = \frac{T e_\mathrm{T}}{\frac{1}{2}\rho V_G^2 S \bar{c}} \tag{2.92}$$

where e_T is the eccentricity of the thrust direction with respect to the center of gravity.

It is worth noticing the presence of M_{32} at the third row of the system. It is the only non-trivial element of the mass matrix M. In fact, it has always been equal to the identity matrix, $\mathbb{1}_6$ for a six-rows dynamical system, up to this case. Now, it is an almost sparse matrix, i.e.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & M_{32} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.93}$$

where

$$M_{32} = -\frac{\bar{c}/b}{4\mu}\frac{V_G \bar{c}}{\kappa_y^2} C_{\mathscr{M}_{\dot{\alpha}}} \tag{2.94}$$

Therefore, the dynamical system can be written in the following form as usual

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & M_{32} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} V_G \\ \alpha \\ q \\ x_{\mathrm{E},G} \\ z_{\mathrm{E},G} \\ \theta \end{Bmatrix} = \begin{Bmatrix} f_1(V_G, \alpha, \quad x_{\mathrm{E},G}, z_{\mathrm{E},G}, \theta; \delta_\mathrm{T}, \delta_\mathrm{e}, \delta_\mathrm{s}) \\ f_2(V_G, \alpha, q, x_{\mathrm{E},G}, z_{\mathrm{E},G}, \theta; \delta_\mathrm{T}, \delta_\mathrm{e}, \delta_\mathrm{s}) \\ f_3(V_G, \alpha, q, x_{\mathrm{E},G}, z_{\mathrm{E},G} \quad ; \delta_\mathrm{T}, \delta_\mathrm{e}, \delta_\mathrm{s}) \\ f_4(V_G, \alpha, \qquad\qquad \theta \qquad\quad) \\ f_5(V_G, \alpha, \qquad\qquad \theta \qquad\quad) \\ f_6(\qquad q \qquad\qquad\quad) \end{Bmatrix} \tag{2.95}$$

after having explicitly written the dependencies of the right hand sides of the equations on the state and input variables for each row of the system. To see a MATLAB implementation of such dynamical model see the 3-DoF linearized model exercise in § 2.4.1.

2.4.1 Exercises

F/A-18 HARV 3-DoF simulation

The following MATLAB routine takes the necessary inputs and returns the column vectors of the state variables values over time.

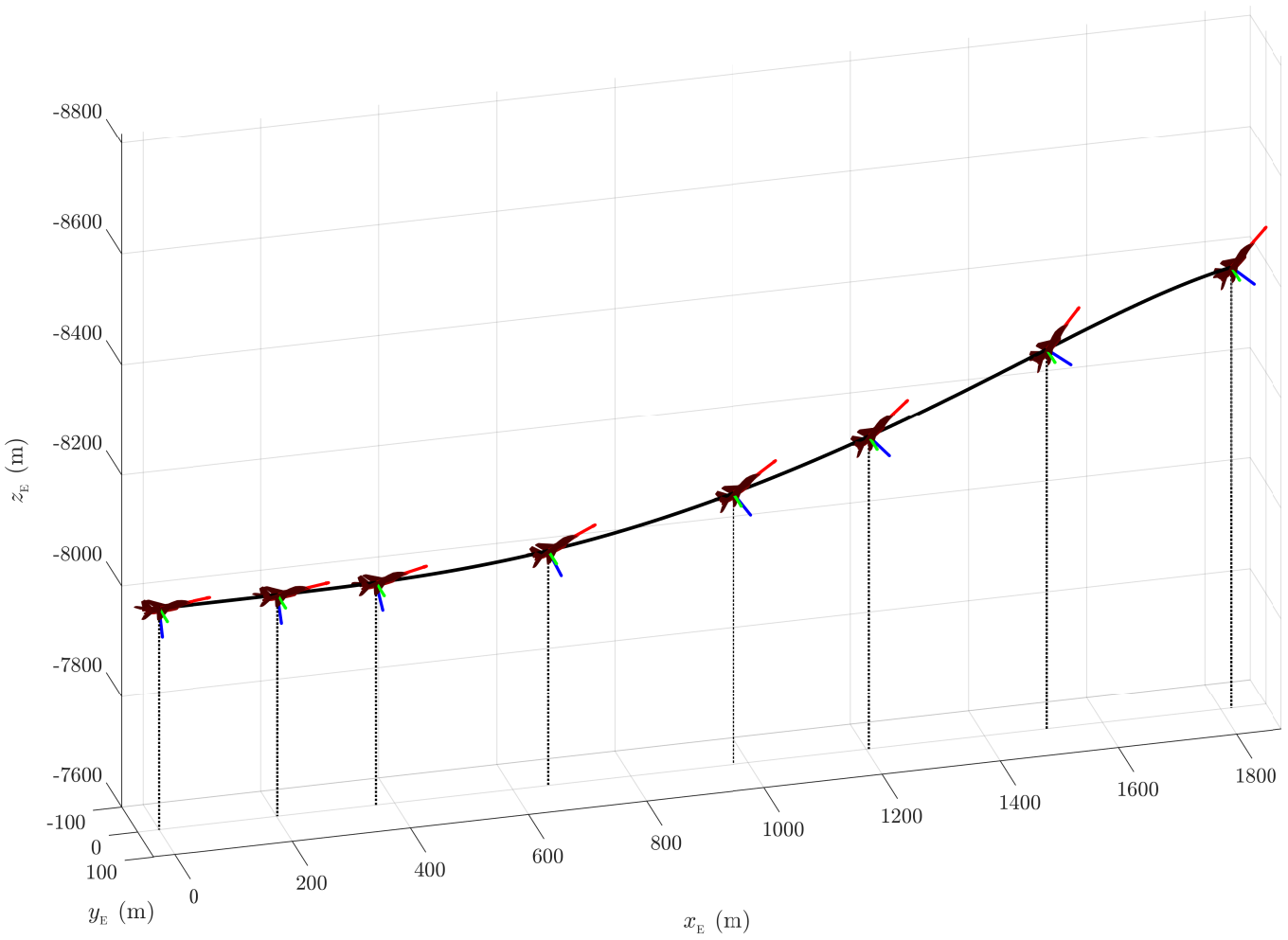


Figure 2.24 Trajectory of the pull-up maneuver resulting from the 3-DoF simulation.

It is worth noticing the reuse of the same aircraft data structure previously introduced through listing 2.1. To do so the generalized inputs vector must be defined as specified in equation (2.35) as required by that function. In particular, to declare the `ThrustModel`, `DragCoeffModel`, `LiftCoeffModel` and `PitchCoeffModel` without errors it must be

$$\boldsymbol{u}^* = \begin{Bmatrix} \alpha_B \\ \beta \\ \delta_T \\ \delta_e \\ \delta_s \\ \delta_a \\ \delta_r \\ p \\ q \\ r \\ h \\ V_G \end{Bmatrix} = \begin{Bmatrix} \alpha_B \\ 0 \\ \delta_T \\ \delta_e \\ \delta_s \\ 0 \\ 0 \\ 0 \\ q \\ 0 \\ h \\ V_G \end{Bmatrix} \tag{2.96}$$

In this script the right hand side of the system of equations (2.72) to (2.77) is declared row-by-row.

Listing 2.15 Function for carrying out 3-DoF simulations.

```
function [time, ...
   delta_T, delta_e_deg, delta_s_deg, ...
   V, alpha, q, x_EG, z_EG, theta] = ...
                  ThreeDoFQuatVab(t_end, state0, ...
                                  myAircraft, ...
                                  delta_T_law, ...
                                  delta_e_deg_law, delta_s_deg_law)

S = myAircraft.S;
c = myAircraft.c;
m = myAircraft.m;
W = myAircraft.W;
I_B = myAircraft.I_B;

function rho = density(h)
[~, ~, ~, rho] = atmosisa(h);
end
rho = @(state) density(-state(5));

airspeed = @(state) state(1);
alpha_deg = @(state) convang(state(2), 'rad', 'deg');
elevation = @(state) state(6);
q_degps = @(state) convangvel(state(3), 'rad/s', 'deg/s');
altitude = @(state) -state(5);

InputCommands = @(t) [delta_T_law(t), ...
               delta_e_deg_law(t), delta_s_deg_law(t), ...
               0, 0];

ModelsInputs = @(t, state) [alpha_deg(state), 0, ...
                     InputCommands(t), ...
                     0, q_degps(state), 0, ...
                     altitude(state), airspeed(state)];

ThrustModel = @(t, state) myAircraft.ThrustModel(ModelsInputs(t, state));
DragCoeffModel = @(t, state) myAircraft.DragCoeffModel(ModelsInputs(t,
    ↪ state));
LiftCoeffModel = @(t, state) myAircraft.LiftCoeffModel(ModelsInputs(t,
    ↪ state));
PitchCoeffModel = @(t, state) myAircraft.PitchCoeffModel(ModelsInputs(t,
    ↪ state));

LiftModel = @(t, state) ...
LiftCoeffModel(t, state) * ...
0.5 * rho(state) * airspeed(state)^2 * S;

DragModel = @(t, state) ...
DragCoeffModel(t, state) * ...
0.5 * rho(state) * airspeed(state)^2 * S;

Pitch = @(t, state) ...
PitchCoeffModel(t, state) * ...
0.5 * rho(state) * airspeed(state)^2 * S * c;

Vdot = @(t, state) 1/m * (-DragModel(t, state) + ...
ThrustModel(t, state) * cos(state(2)) - W * ...
(cos(state(2)) * sin(elevation(state)) - ...
sin(state(2)) * cos(elevation(state))));

alphadot = @(t, state) 1/(m*airspeed(state)) * (-LiftModel(t, state) -
    ↪ ...
ThrustModel(t, state) * sin(state(2)) + W * ...
(cos(state(2)) * cos(elevation(state)) + ...
sin(state(2)) * sin(elevation(state)))) + state(3);

qdot = @(t, state) Pitch(t, state) / I_B(2, 2);

xEGdot = @(t, state) airspeed(state) * (cos(state(2)) * ...
cos(elevation(state)) + sin(state(2)) * ...
sin(elevation(state)));
```

```
67
68 zEGdot = @(t, state) airspeed(state) * (sin(state(2)) * ...
69 cos(elevation(state)) - cos(state(2)) * ...
70 sin(elevation(state)));
71
72 thetadot = @(t, state) state(3);
73
74 dstate_dt = @(t, state) [ Vdot(t, state); ...
75                   alphadot(t, state); ...
76                   qdot(t, state); ...
77                   xEGdot(t, state); ...
78                   zEGdot(t, state); ...
79                   thetadot(t, state) ];
80
81 ODEoptions = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);
82
83 [time, state] = ode45(dstate_dt, [0, t_end], state0, ODEoptions);
84
85 delta_T = delta_T_law(time);          delta_T = delta_T(:);
86 delta_e_deg = delta_e_deg_law(time); delta_e_deg = delta_e_deg(:);
87 delta_s_deg = delta_s_deg_law(time); delta_s_deg = delta_s_deg(:);
88 V = state(:, 1);
89 alpha = state(:, 2);
90 q = state(:, 3);
91 x_EG = state(:, 4);
92 z_EG = state(:, 5);
93 theta = state(:, 6);
94
95 end
```

This routine is used in the next listing. It simulates for 20 s the pull-up maneuver shown in 2.24. The initial conditions assigned for this simulation are

- $M_0 = 0.3895$
- $\alpha_{B,0} = 6.1825$ deg
- $q_0 = 0$ deg/s
- $h_0 = 8000$ m
- $\theta_0 = 6.1825$ deg

The input commands time laws are displayed in figure 2.25.

In figure 2.26 we can notice how the F/A-18 HARV can flight at very high angle of attacks when compared against other conventional aircrafts. During the simulated pull-up maneuver it has increased its angle of attack by about 30 deg as shown in the mentioned figure. At the end of the simulation also the altitude increased by about 400 m as shown in figure 2.28.

Listing 2.16 F/A-18 HARV 3-DoF simulation.

```
1 close all; clear; clc
2
3 myAircraft = load('f18harv.mat');
4
5 ModelsInputs = zeros(1, 11);
6 delta_e_deg_cases = -20 : 5 : 10;
7
8 liftCharacteristicsPlot(myAircraft, ModelsInputs, [-5, 40], ...
9                 delta_e_deg_cases, 'ex3dofliftcharacteristics.pdf')
10
11 staticPolarPlot(myAircraft, ModelsInputs, [-5, 40], ...
12                 delta_e_deg_cases, 'ex3dofdragcharacteristics.pdf')
13
14 q_degps_cases = linspace(-10, 15, 6);
```

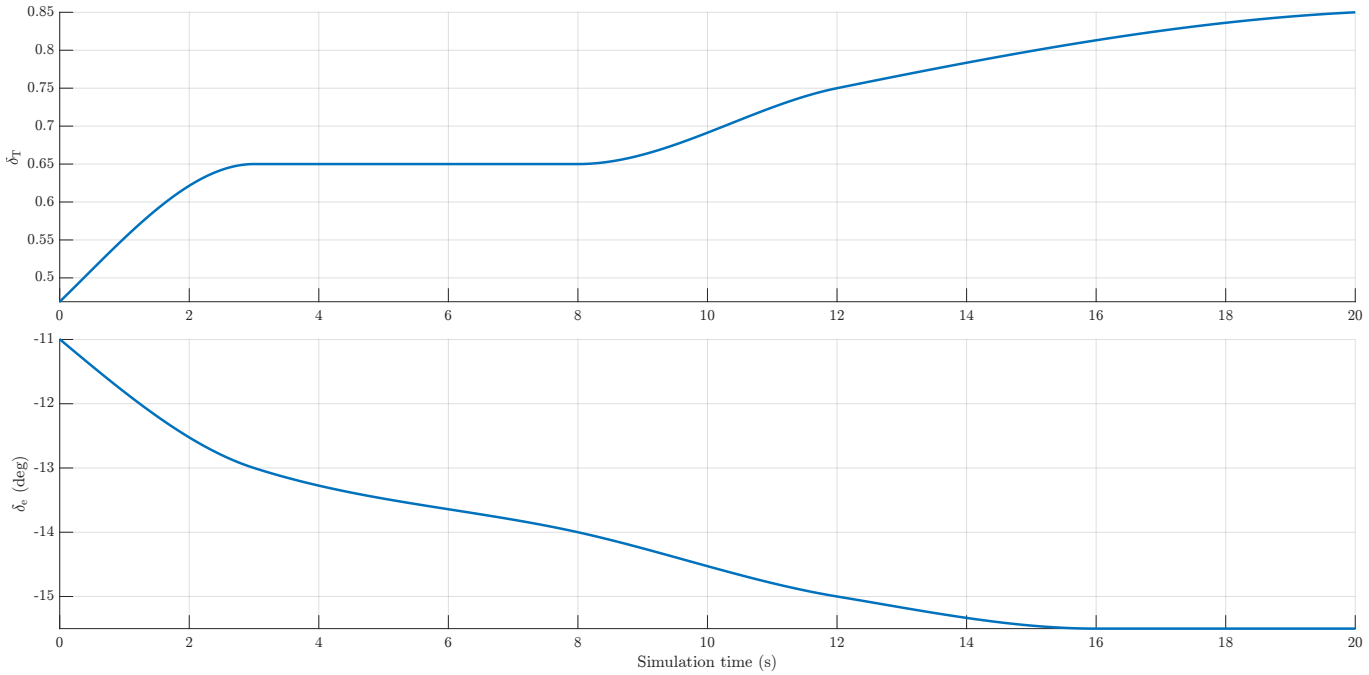


Figure 2.25 Input commands time law during the maneuver.

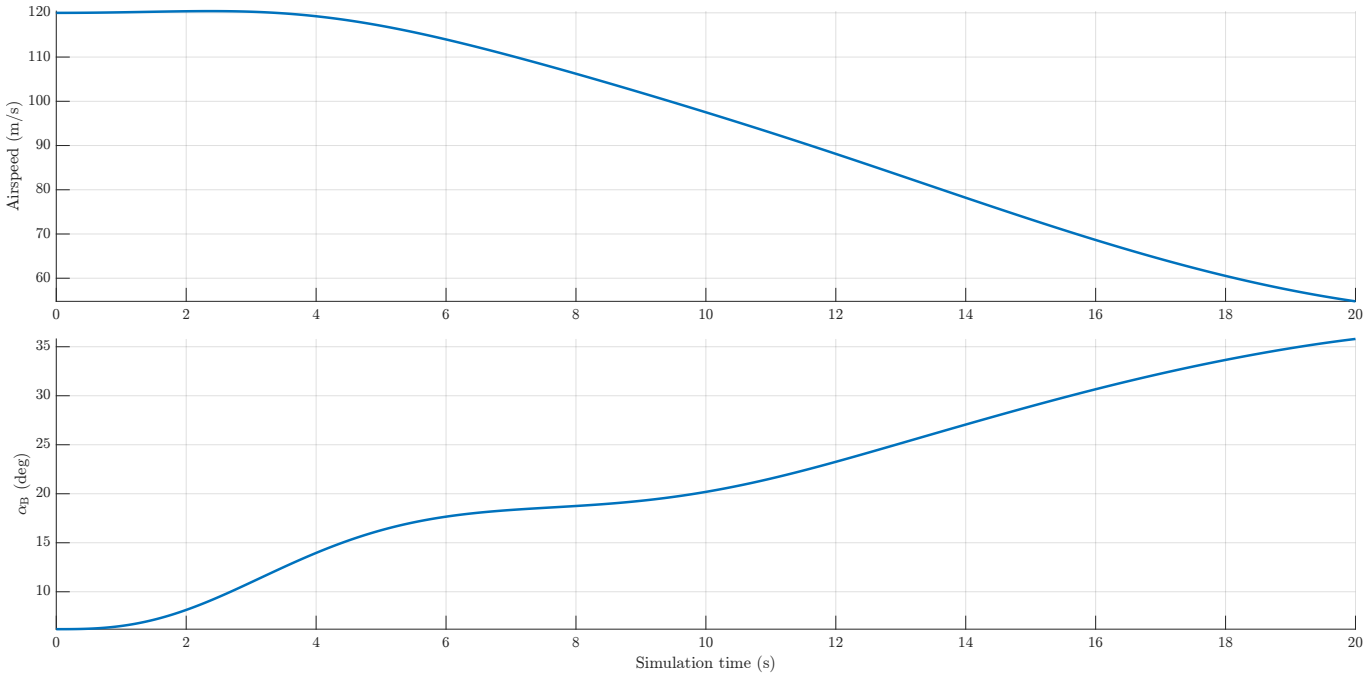


Figure 2.26 Time histories of the aircraft center of gravity airspeed and of its angle of attack.

```
pitchCharacteristicsPlot(myAircraft, ModelsInputs, [-5, 40], ...
                    delta_e_deg_cases, q_degps_cases, ...
                    'ex3dofpitchcharacteristics.pdf')

t_end = 20;

x_EG0 = 0;
z_EG0 = -8000;

elevation0 = convang(6.1825, 'deg', 'rad');

alpha0 = convang(6.1825, 'deg', 'rad');
```

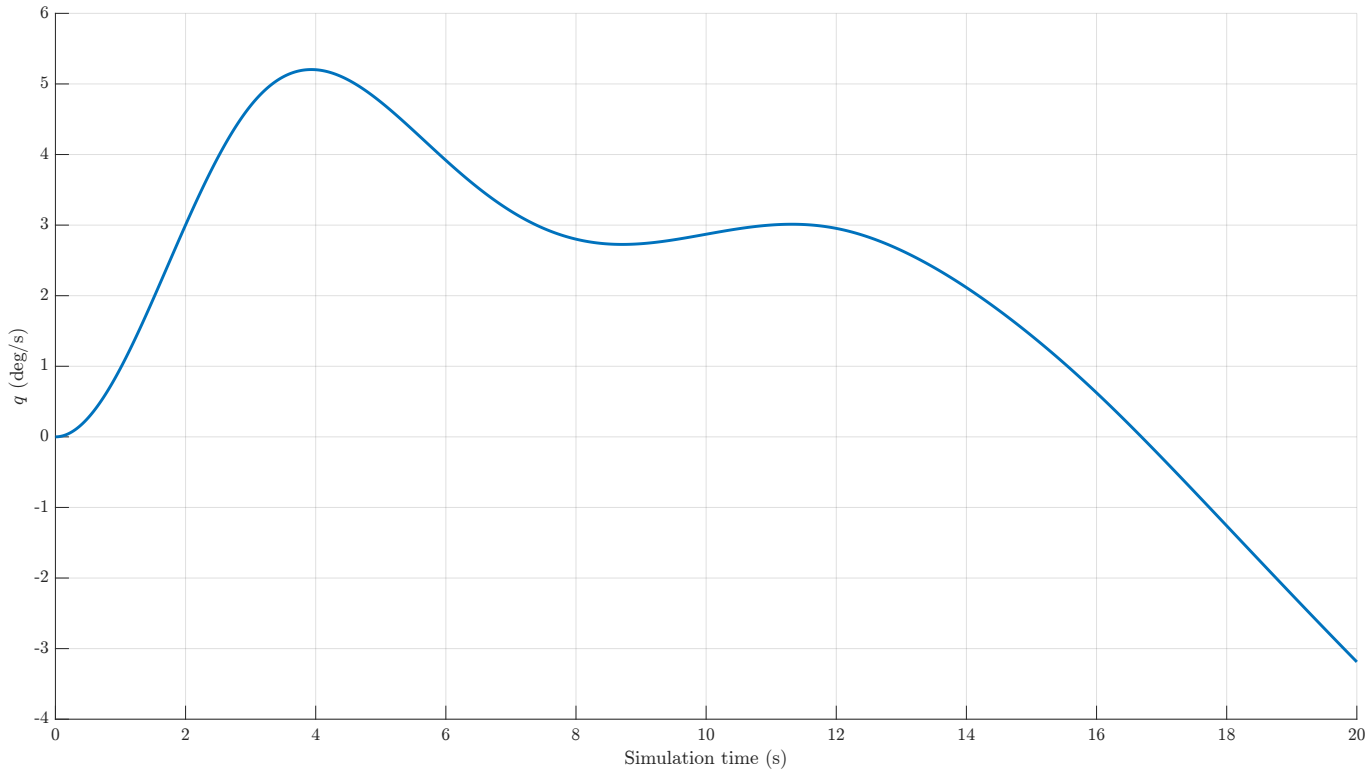


Figure 2.27 Pitch angular velocity time history during the simulation.

```
Mach0 = 0.3895;
[temp0, sound0, press0, dens0] = atmosisa(-z_EG0);
V0 = Mach0 * sound0;

q0 = 0;

state0 = [V0, alpha0, q0, x_EG0, z_EG0, elevation0].';

delta_T0 = 0.4686;
delta_T_law = @(t) interp1([0, 3, 8, 12, t_end], ...
                           [delta_T0, 0.65, 0.65, 0.75, 0.85], ...
                           t, 'pchip');

delta_e0_deg = -10.9958;
delta_e_deg_law = @(t) interp1( ...
                [0, 3, 8, 12, 16, t_end], ...
                [delta_e0_deg,   -13, -14, -15, -15.5, -15.5], ...
                t, 'pchip');

delta_s0_deg = 0;
delta_s_deg_law = @(t) interp1([0, 1] * t_end, ...
                               [1, 1] * delta_s0_deg, ...
                               t, 'linear');

[time, ...
delta_T, delta_e_deg, delta_s_deg, ...
V, alpha, q, x_EG, z_EG, elevation] = ThreeDoFQuatVab(t_end, state0, ...
                                                      myAircraft, ...
                                                      delta_T_law, ...
                                                      delta_e_deg_law, ...
    ↪ ...
                                                      delta_s_deg_law);

Nt = length(time);

stackedPlot2(time, ...
             delta_T, delta_e_deg, ...
             {"Simulation time (s)", ...
```

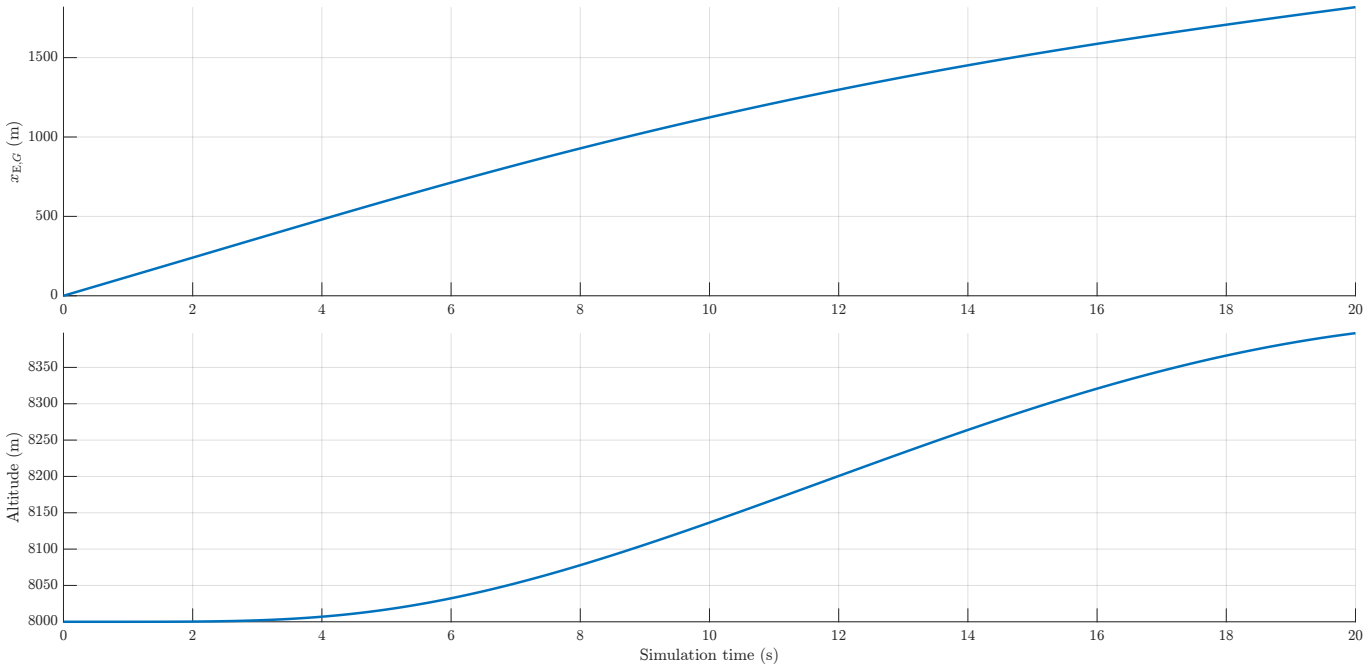


Figure 2.28 Aircraft center of gravity coordinates over time.

```
                "$\delta_\mathrm{T}$", "$\delta_\mathrm{e}$ (deg)"}, ...
            {}, {}, 'ex3dofinputcommands.pdf')

alpha_deg = convang(alpha, 'rad', 'deg');
stackedPlot2(time, ...
            V, alpha_deg, ...
            {"Simulation time (s)", ...
                "Airspeed (m/s)", "$\alpha_\mathrm{B}$ (deg)"}, ...
            {}, {}, 'ex3dofairspeedAoA.pdf')

q_degps = convangvel(q, 'rad/s', 'deg/s');
multiPlot(time, ...
            "Simulation time (s)", "$q$ (deg/s)", ...
            {}, ...
            'ex3dofqdegps.pdf', ...
            q_degps)

stackedPlot2(time, ...
            x_EG, -z_EG, ...
            {"Simulation time (s)", ...
                "$x_{\mathrm{E},G}$ (m)", "Altitude (m)"}, ...
            {}, {}, 'ex3dofcogxz.pdf')

elevation_deg = convang(elevation, 'rad', 'deg');
multiPlot(time, ...
            "Simulation time (s)", "$\theta$ (deg)", ...
            {}, ...
            'ex3dofelevation.pdf', ...
            elevation_deg)

0 = zeros(Nt, 1);
ModelsInputs = [alpha_deg, 0, ...
                delta_T, delta_e_deg, delta_s_deg, 0, 0, ...
                0, q_degps, 0, ...
                -z_EG, V];
CL = zeros(Nt, 1);
CD = CL;
CPitch = CL;
for it = 1 : Nt
    CL(it) = myAircraft.LiftCoeffModel(ModelsInputs(it, :));
    CD(it) = myAircraft.DragCoeffModel(ModelsInputs(it, :));
    CPitch(it) = myAircraft.PitchCoeffModel(ModelsInputs(it, :));
```

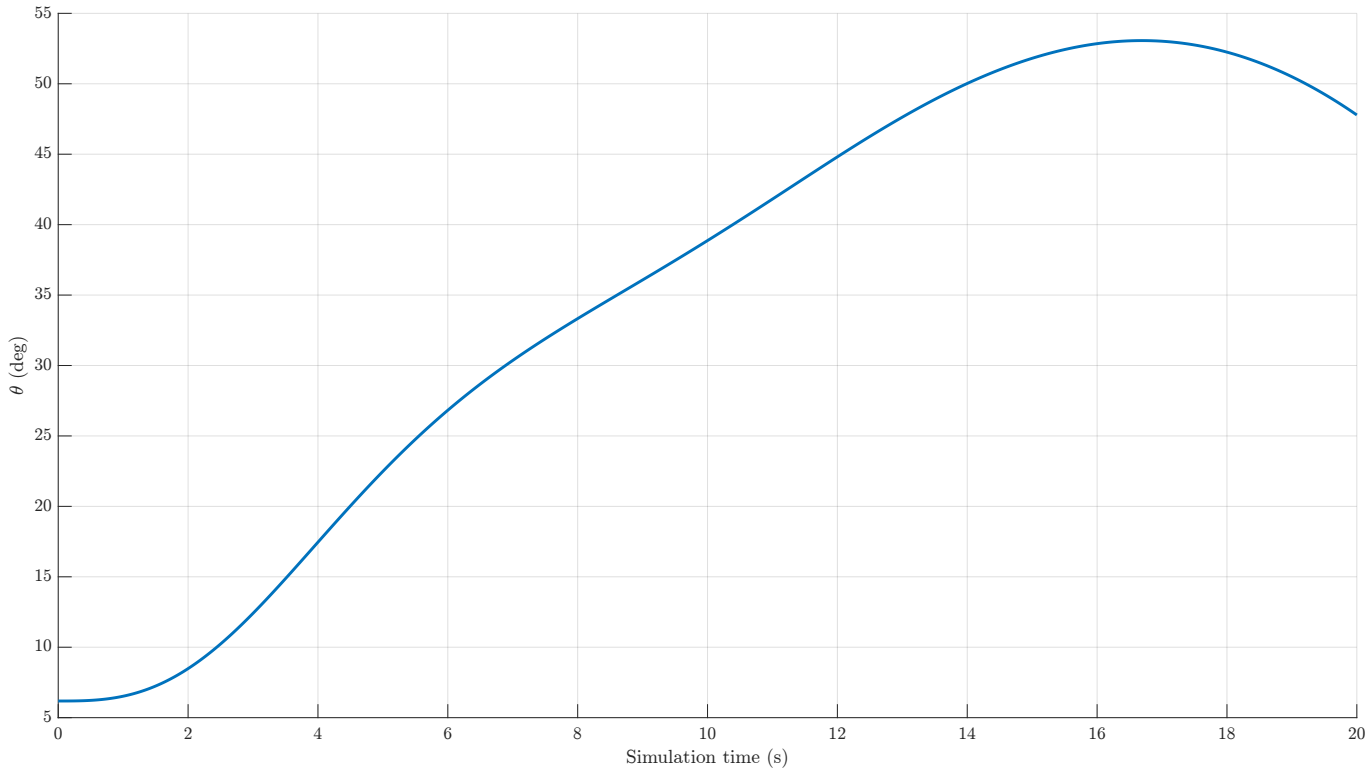


Figure 2.29 Time history of the aircraft body longitudinal orientation angle.

```
end
stackedPlot3(time, ...
            CL, CD, CPitch, ...
            {"Simulation time (s)", ...
                "$C_L$", "$C_D$", "$C_\mathcal{M}$"}, ...
            {}, {}, 'ex3dofaerocoeffhistories.pdf')

u = V .* cos(alpha);
v = 0;
w = V .* sin(alpha);
quat = angle2quat(0, elevation, 0);
quat0 = quat(:, 1);
quatx = quat(:, 2);
quaty = quat(:, 3);
quatz = quat(:, 4);
[f_xA, f_yA, f_zA] = LoadFactor(time, u, v, w, alpha_deg, quat0, quatx,
    ↪ quaty, quatz);
multiPlot(time, ...
            "Simulation time (s)", "$f_{z_\mathrm{A}}$", ...
            {}, ...
            'ex3dofloadfactor.pdf', ...
            f_zA)

trajectoryView(time, ...
                x_EG, 0, z_EG, ...
                0, elevation, 0, ...
                70, 8, ...
                'ex3doftrajectory.pdf')

dens = zeros(Nt, 1);
for it = 1 : Nt
    [~, ~, ~, dens(it)] = atmosisa(-z_EG(it));
end
lift = 1e-3 * CL .* 0.5 .* dens .* V.^2 .* myAircraft.S;
drag = 1e-3 * CD .* 0.5 .* dens .* V.^2 .* myAircraft.S;
stackedPlot2(time, ...
            lift, drag, ...
            {"Simulation time (s)", ...
```

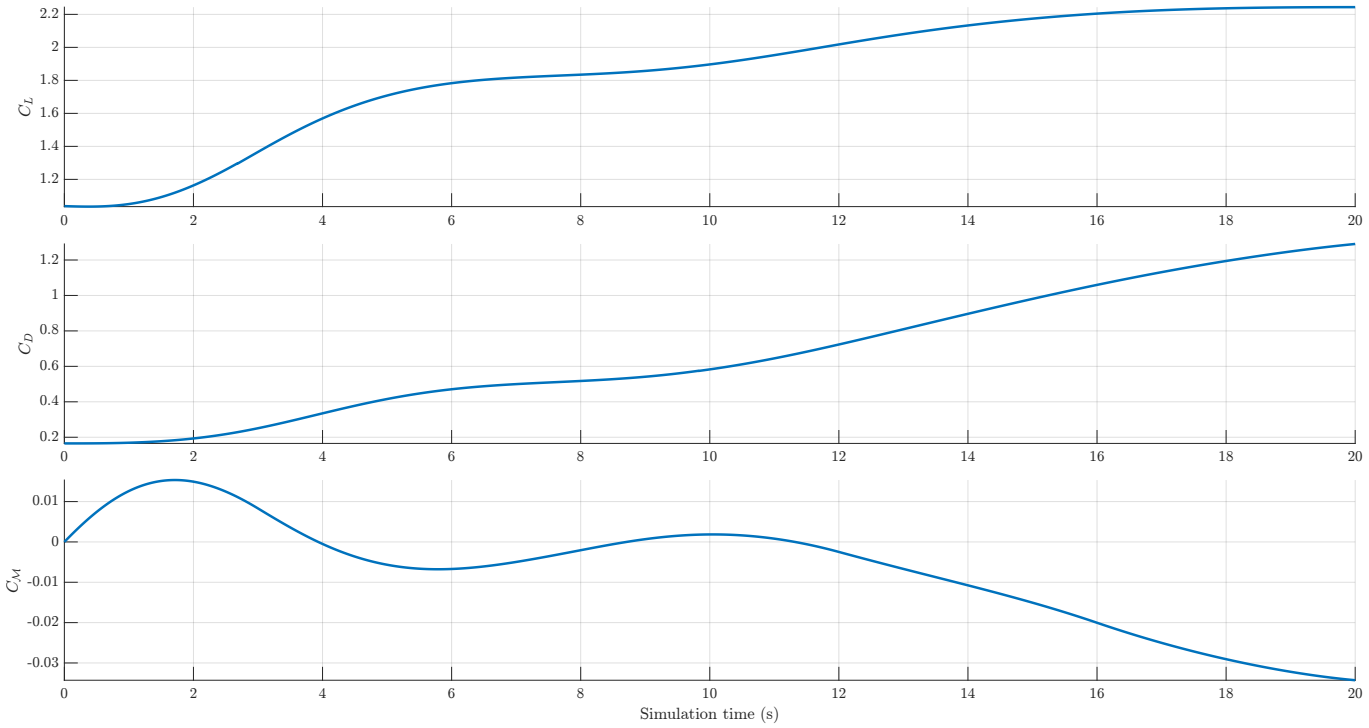


Figure 2.30 Aerodynamic coefficients time histories.

```
                "Lift (kN)", "Drag (kN)"}, ...
            {}, {}, 'ex3dofliftdrag.pdf')
```

It is worth noticing the time history diagram of the normal load factor in figure 2.31. It shows a peak when the aircraft center of gravity has the maximum vertical acceleration during the simulation. After that it decreases to then become less than 1 since the aircraft is decelerating at the end of the maneuver.

The causes for this time evolution of the normal load factor can be investigated in the aerodynamic coefficients time history diagrams shown in figure 2.30. Despite the lift coefficient increases up to twice its equilibrium value, the drag coefficient increases from 200 drag counts to 1200 drag counts due to the high change in the angle of attack. It makes the airspeed decrease and so does the dynamic pressure $\bar{q} = 0.5\rho V_G^2$.

Therefore, the lift itself decreases and so does the normal acceleration. This evolution just deduced can be clearly seen in figure 2.32.

A 3-DoF linearized model

In this exercise I implemented in MATLAB the system of equations (2.85) to (2.90) as shown in equation (2.95). This implementation has been done inside the MATLAB routine appended at this exercise that can be used to easily carry out three degrees of freedom simulations.

It is important noticing that for this kind of simulations the aircraft data must be stored in a MATLAB struct different from the one discussed from the F/A-18 exercise seen in § 2.3.1 on. In particular, the so called `myAircraft` object must at least contain the following fields.

- Aircraft mass (kg), `mass`

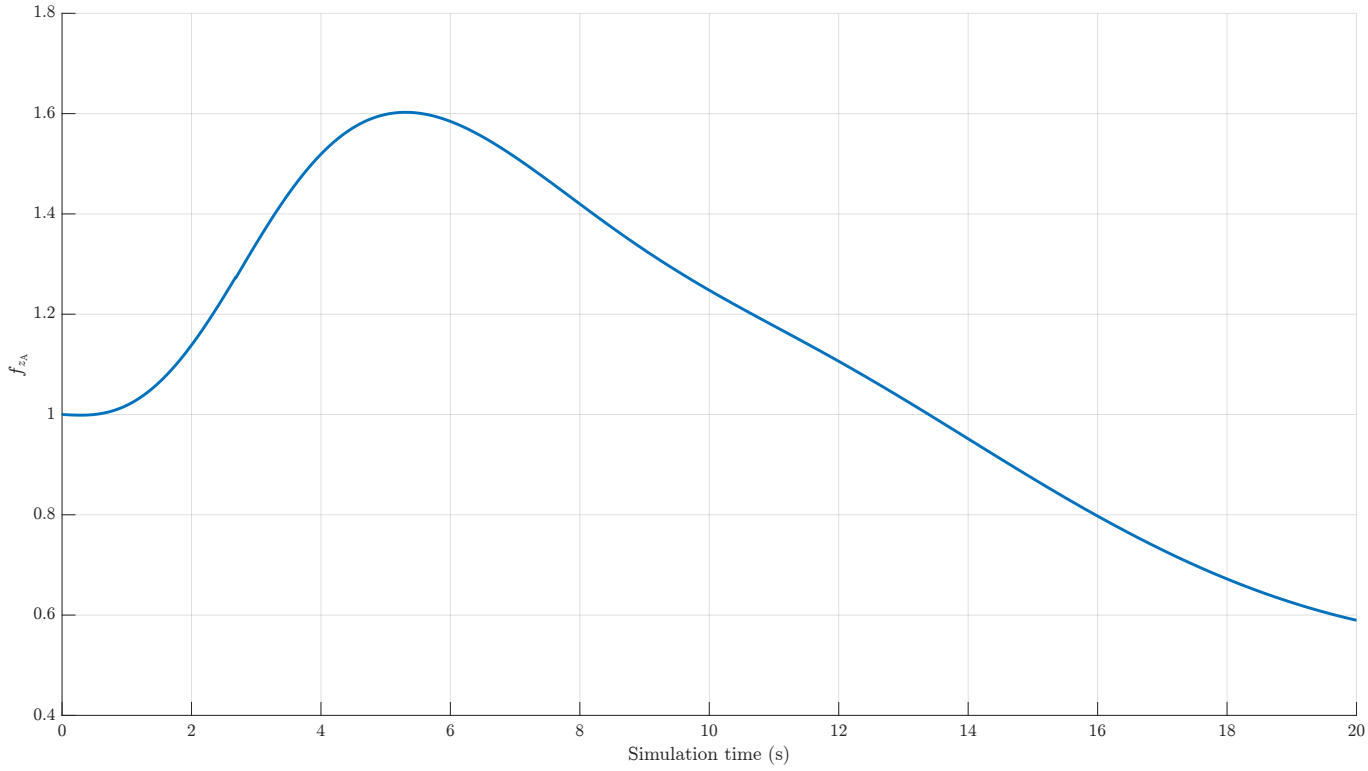


Figure 2.31 Normal load factor evolution during the simulation.

- Wing surface (m^2), `S`
- Wing mean aerodynamic chord (m), `mac`
- Wingspan (m), `b`
- Zero lift angle μ_x (rad), `mu_x`
- Thrust line and fuselage reference line angle μ_T (rad), `mu_T`
- Parameters in the lift-drag polar C_{D_0}, k and m, `CD_0`, `K` and `m` respectively
- Aircraft gyration radius in the pitching line direction κ_y (m), `k_y`
- Parameters in the lift coefficient linearization C_{L_α}, $C_{L_{\delta_e}}$, $C_{L_{\delta_s}}$, C_{L_q} and $C_{L_{\dot\alpha}}$ (1/rad), `CL_alpha`, `CL_delta_e`, `CL_delta_s`, `CL_q` and `CL_alpha_dot` respectively
- Parameters in the pitching moment coefficient linearization $C_{\mathcal{M}_0}$, $C_{\mathcal{M}_\alpha}$, $C_{\mathcal{M}_{\delta_e}}$, $C_{\mathcal{M}_{\delta_s}}$, $C_{\mathcal{M}_q}$ and $C_{\mathcal{M}_{\dot\alpha}}$ (1/rad), `Cm_0`, `Cm_alpha`, `Cm_delta_e`, `Cm_delta_s`, `Cm_q` and `Cm_alpha_dot` respectively

It is worth noticing that here the thrust model must not be contained into the object `myAircraft`, but must be present in the script directory as an external routine. It must be defined with the same convention discussed in the F/A-18 exercise in § 2.3.1.

The aircraft relative density μ is defined as an anonymous function of the state variables. The non-trivial element M_{32} and so the mass matrix M are assigned as an anonymous function of time and state variables as shown in equation (2.93).

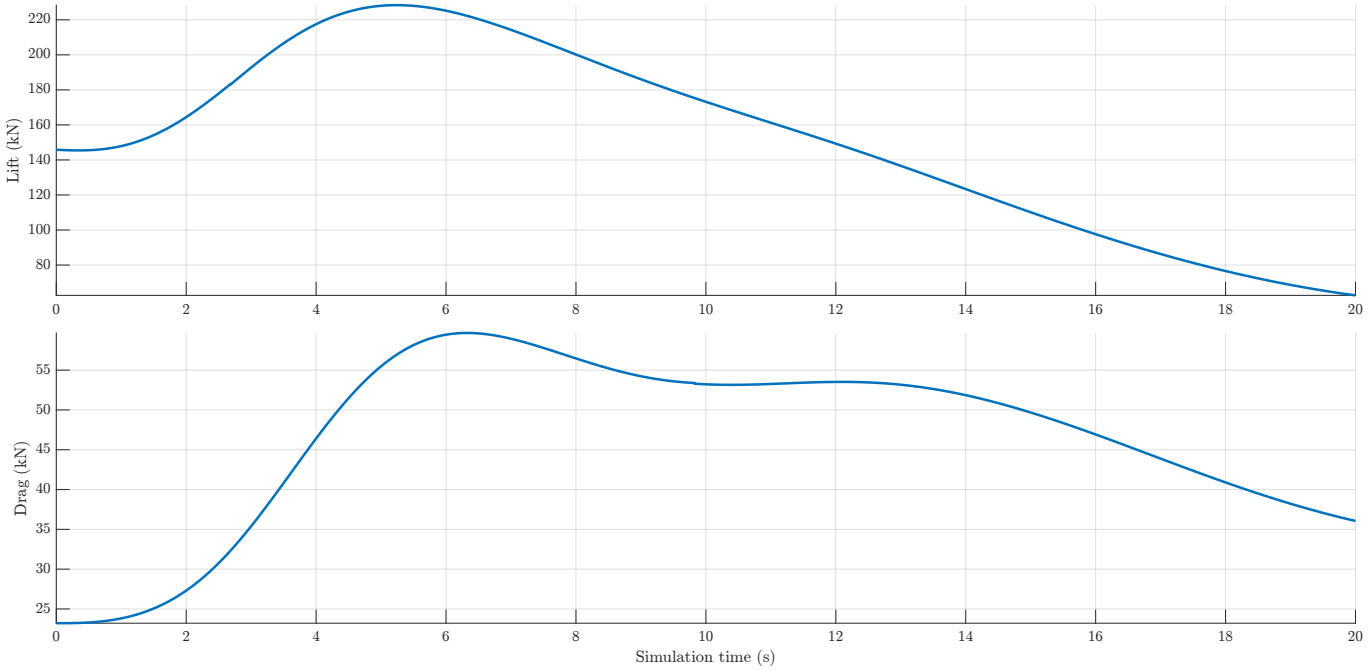


Figure 2.32 Lift and drag forces time histories during the simulation. As previously mentioned, the lift first arises and then decreases significantly due to the drop in arispeed caused by the high drag reached.

The right hand side of the linearized three degrees of freedom dynamical system in explicitly defined to be able to solve the simulation using the ode45 MATLAB routine. In this case it is assigned as the explicit form

$$\dot{\boldsymbol{x}} = M(t, \boldsymbol{x})^{-1} \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u})$$

Listing 2.17 Function for carrying out 3-DoF simulations with linearized aerodynamic coefficients.

```
function [time, ...
   delta_T, delta_e_deg, delta_s_deg, ...
   V, alpha, q, x_EG, z_EG, theta] = ...
               ThreeDoFVabLin(t_end, state0, ...
                              myAircraft, ...
                              delta_T_law, ...
                              delta_e_deg_law, delta_s_deg_law)

delta_e_law = @(t) convang(delta_e_deg_law(t), 'deg', 'rad');
delta_s_law = @(t) convang(delta_s_deg_law(t), 'deg', 'rad');

g = 9.81;
W = myAircraft.mass * g;
S = myAircraft.S;
c = myAircraft.mac;
b = myAircraft.b;
mu_x = myAircraft.mu_x;
mu_T = myAircraft.mu_T;
k = myAircraft.K;
m = myAircraft.m;
k_y = myAircraft.k_y;
CL_alpha = myAircraft.CL_alpha;
CL_delta_e = myAircraft.CL_delta_e;
CL_delta_s = myAircraft.CL_delta_s;
CL_alpha_dot = myAircraft.CL_alpha_dot;
CL_q = myAircraft.CL_q;
CD0 = myAircraft.CD_0;
Cm0 = myAircraft.Cm_0;
```

```
Cm_alpha = myAircraft.Cm_alpha;
Cm_delta_e = myAircraft.Cm_delta_e;
Cm_delta_s = myAircraft.Cm_delta_s;
Cm_alpha_dot = myAircraft.Cm_alpha_dot;
Cm_q = myAircraft.Cm_q;
Cm_T_0 = myAircraft.Cm_T_0;
Cm_T_alpha = myAircraft.Cm_T_alpha;

airspeed = @(state) state(1);
alpha_deg = @(state) convang(state(2), 'rad', 'deg');
elevation = @(state) state(6);
q_degps = @(state) convangvel(state(3), 'rad/s', 'deg/s');
altitude = @(state) -state(5);

InputCommands = @(t) [delta_T_law(t), ...
               delta_e_deg_law(t), delta_s_deg_law(t), ...
               0, 0];

ModelsInputs = @(t, state) [alpha_deg(state), 0, ...
                     InputCommands(t), ...
                     0, q_degps(state), 0, ...
                     altitude(state), airspeed(state)];

Thrust = @(t, state) ThrustModel(ModelsInputs(t, state), myAircraft);
Cm_T = @(t, state) delta_T_law(t) * (Cm_T_0 + Cm_T_alpha * state(2));

function rho = density(h)
[~, ~, ~, rho] = atmosisa(h);
end
rho = @(state) density(altitude(state));
mu = @(state) W / S / (rho(state) * b * g);

M32 = @(state) -c/b / (4 * mu(state)) * airspeed(state)*c/k_y^2 *
    ↪ Cm_alpha_dot;
massmatrix = @(t, state) [1,      0,     0, 0, 0, 0; ...
                  0,      1,     0, 0, 0, 0; ...
                  0, M32(state), 1, 0, 0, 0; ...
                  0,      0,     0, 1, 0, 0; ...
                  0,      0,     0, 0, 1, 0; ...
                  0,      0,     0, 0, 0, 1];

Vdot = @(t, state) ...
-0.5 * S/W * rho(state) * airspeed(state)^2 * g * ( ...
 CD0 + k * (CL_alpha*state(2) + CL_delta_e*delta_e_law(t) + ...
                                CL_delta_s*delta_s_law(t))^m) ...
+Thrust(t, state)/W * g * cos(state(2) - mu_x - mu_T) ...
+g * sin(state(2) - mu_x - elevation(state));

alphadot = @(t, state) 1 / (1 + c/b / (4 * mu(state)) * CL_alpha_dot) *
    ↪ ( ...
-0.5 * S/W * rho(state) * airspeed(state) * g * ( ...
 CL_alpha*state(2) + CL_delta_e*delta_e_law(t) +
    ↪ CL_delta_s*delta_s_law(t)) ...
+state(3) * (1 - c/b / (4 * mu(state)) * CL_q) ...
-Thrust(t, state)/W * g/airspeed(state) * sin(state(2) - mu_x - mu_T) ...
+g/airspeed(state) * cos(state(2) - mu_x - elevation(state)));

qdot = @(t, state) airspeed(state)^2/k_y^2 * c/b / (2 * mu(state)) * (
    ↪ ...
Cm0 + Cm_alpha*state(2) + c/(2*airspeed(state))*(Cm_q*state(3) ) ...
+Cm_delta_e*delta_e_law(t) + Cm_delta_s*delta_s_law(t) + Cm_T(t, state));

xEGdot = @(t, state) airspeed(state) * cos(state(2) - mu_x -
    ↪ elevation(state));

zEGdot = @(t, state) airspeed(state) * sin(state(2) - mu_x -
    ↪ elevation(state));

thetadot = @(t, state) state(3);

dstate_dt = @(t, state) massmatrix(t, state) \ ...
                [ Vdot(t, state); ...
```

```
                    alphadot(t, state); ...
                    qdot(t, state); ...
                    xEGdot(t, state); ...
                    zEGdot(t, state); ...
                    thetadot(t, state) ];

ODEoptions = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);

[time, state] = ode45(dstate_dt, [0, t_end], state0, ODEoptions);

delta_T = delta_T_law(time);          delta_T = delta_T(:);
delta_e_deg = delta_e_deg_law(time); delta_e_deg = delta_e_deg(:);
delta_s_deg = delta_s_deg_law(time); delta_s_deg = delta_s_deg(:);
V = state(:, 1);
alpha = state(:, 2);
q = state(:, 3);
x_EG = state(:, 4);
z_EG = state(:, 5);
theta = state(:, 6);

end
```

2.5 Trim

The *trim problem* is the necessity to find specific values to assign at the input commands to guarantee equilibrium during flight. Where in this context *equilibrium* stands for having a set of state variables or flight variables that are kept stationary or at a constant assigned value over time.

The set of variables that must be stationary depends on the desired flight condition. For instance, during *steady level flight* the airspeed V_G, the angle of attack α_B and the altitude h are constant over time, while the sideslip angle β and the angular velocity components p, q and r must all be zero. An other example is a *steady and coordinate turn*, in which the airspeed V_G, the angle of attack α_B, the sideslip angle β the altitude h and the bank angle φ are stationary.

To reach the required conditions for a specific flight configuration, such as the two just mentioned, the input commands and some other variables must be tweaked accordingly. The variables to tweak in pursuit of the steady state are conventionally grouped in a vector $\boldsymbol{\xi}$ called *design vector*. It will be defined for each case depending on the needs of the problem. It typically contains the input commands and other variables as needed.

The design vector must be evaluated and the values obtained are the ones that guarantee the searched equilibrium condition. To successfully determine these values a function of the design vector $J = J(\boldsymbol{\xi})$ called *cost function* is defined as follows

$$J = \sum_k f_k^2 + \sum_r (x_r - \tilde{x}_r)^2 \tag{2.97}$$

where f_k are the components of the right hand side $\boldsymbol{f}$ of the dynamical system corresponding to the stationary state variables, while x_r are the state variables that are assigned to the desired values $\tilde{x}_r$.

The so defined cost function must then be minimized in order to reach the imposed conditions. To do so a different optimization algorithms can be used through direct implementation, different tools or programming language libraries. For the sake of simplicity we will apply the pre-built MATLAB routine `fmincon`. See the next exercises for some applied examples.

When a trim condition has been found, the obtained state variables, input variables and observed variables are each one labelled with a subscript 0. Respectively $\boldsymbol{x}_0$, $\boldsymbol{u}_0$ and $\boldsymbol{y}_0$. This notation refers to the practical usage of such values. In fact, searching for a trimmed flight condition means searching for some initial conditions of the state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and for some constant input commands laws $\boldsymbol{u}(t) = \boldsymbol{u}_0$ such that the requested (stationary or assigned value) constraints are guaranteed over time.

2.5.1 Exercises

3-DoF Trim

The following MATLAB routine I wrote takes the aircraft struct defined as in the 3-DoF linearized model exercise in § 2.4.1, the desired flight parameters $V_{G,0}$, h_0, γ_0, the assigned $\delta_{s,0}$ and the limit values of the design variables as inputs, to return the design vector evaluated at the trim condition and the cost function value. The design variables are stored in the design vector, here defined as

$$\boldsymbol{\xi} = \begin{Bmatrix} \alpha \\ \delta_\mathrm{T} \\ \delta_\mathrm{e} \\ \delta_\mathrm{s} \end{Bmatrix}$$

For a three degrees of freedom flight trim the desired stationary variables are V_G, α and θ. Therefore, in this case the cost function is defined as

$$J = \dot{V}_G^2 + \dot{\alpha}^2 + \dot{q}^2 + \dot{\theta}^2$$

where the presence of both $\dot{q}$ and $\dot{\theta}$ is redundant, since in this routine is set $q = 0$ and $\dot{\theta} = q = 0$.

Listing 2.18 Function for solving the 3-DoF trim with linearized aerodynamic coefficients.

```
function [design, Jval] = ThreeDoFTrimLin(myAircraft, ...
   airspeed, altitude0, ...
   delta_s_deg, gamma0_deg, ...
   design0, lowerBounds, upperBounds)

g = 9.81;
W = myAircraft.mass * g;
S = myAircraft.S;
c = myAircraft.mac;
b = myAircraft.b;
mu_x = myAircraft.mu_x;
mu_T = myAircraft.mu_T;
k = myAircraft.K;
m = myAircraft.m;
k_y = myAircraft.k_y;
CL_alpha = myAircraft.CL_alpha;
CL_delta_e = myAircraft.CL_delta_e;
CL_delta_s = myAircraft.CL_delta_s;
CL_alpha_dot = myAircraft.CL_alpha_dot;
CL_q = myAircraft.CL_q;
CD0 = myAircraft.CD_0;
Cm0 = myAircraft.Cm_0;
Cm_alpha = myAircraft.Cm_alpha;
Cm_delta_e = myAircraft.Cm_delta_e;
Cm_delta_s = myAircraft.Cm_delta_s;
Cm_alpha_dot = myAircraft.Cm_alpha_dot;
Cm_q = myAircraft.Cm_q;
```

```
Cm_T_0 = myAircraft.Cm_T_0;
Cm_T_alpha = myAircraft.Cm_T_alpha;

alpha = @(xi) xi(1);
delta_T = @(xi) xi(2);
delta_e = @(xi) xi(3);
delta_s = @(xi) xi(4);

gamma0 = convang(gamma0_deg, 'deg', 'rad');

q = 0;

[~, ~, ~, rho] = atmosisa(altitude0);
mu = W / S / (rho * b * g);

InputCommands = @(xi) [delta_T(xi), ...
convang(delta_e(xi), 'rad', 'deg'), ...
convang(delta_s(xi), 'rad', 'deg'), ...
0, 0];

alpha_deg = @(xi) convang(alpha(xi), 'rad', 'deg');
mu_x_deg = convang(mu_x, 'rad', 'deg');
ModelsInputs = @(xi) [alpha_deg(xi)-mu_x_deg, 0, ...
InputCommands(xi), ...
0, 0, 0, ...
altitude0, airspeed];

Thrust = @(xi) ThrustModel(ModelsInputs(xi), myAircraft);
Cm_T = @(xi) delta_T(xi) * (Cm_T_0 + Cm_T_alpha * alpha(xi));

M32 = -c/b / (4 * mu) * airspeed*c/k_y^2 * Cm_alpha_dot;
massmatrix = [1,  0,  0, 0, 0, 0; ...
0,  1,  0, 0, 0, 0; ...
0, M32, 1, 0, 0, 0; ...
0,  0,  0, 1, 0, 0; ...
0,  0,  0, 0, 1, 0; ...
0,  0,  0, 0, 0, 1];

Vdot = @(xi) ...
-0.5 * S/W * rho * airspeed^2 * g * ( ...
CD0 + k * (CL_alpha*alpha(xi) + CL_delta_e*delta_e(xi) + ...
CL_delta_s*delta_s(xi))^m) ...
+Thrust(xi)/W * g * cos(alpha(xi) - mu_x - mu_T) ...
-g * sin(gamma0);

alphadot = @(xi) 1 / (1 + c/b / (4 * mu) * CL_alpha_dot) * ( ...
-0.5 * S/W * rho * airspeed * g * ( ...
CL_alpha*alpha(xi) + CL_delta_e*delta_e(xi) + ...
         CL_delta_s*delta_s(xi)) ...
+q * (1 - c/b / (4 * mu) * CL_q) ...
-Thrust(xi)/W * g/airspeed * sin(alpha(xi) - mu_x - mu_T) ...
+g/airspeed* cos(gamma0));

qdot = @(xi) airspeed^2/k_y^2 * c/b / (2 * mu) * ( ...
Cm0 + Cm_alpha*alpha(xi) + Cm_q*q*c/(2*airspeed) ...
+Cm_delta_e*delta_e(xi) + Cm_delta_s*delta_s(xi) + Cm_T(xi));

xEGdot = @(xi) airspeed * cos(gamma0);

zEGdot = @(xi) -airspeed * sin(gamma0);

thetadot = @(xi) q;

dstate_dt = @(xi) massmatrix \ [Vdot(xi); ...
alphadot(xi); ...
qdot(xi); ...
xEGdot(xi); ...
zEGdot(xi); ...
thetadot(xi)];

function J = cost(dstate_dt)
J = dstate_dt(1)^2 + dstate_dt(2)^2 + dstate_dt(3)^2 + dstate_dt(6)^2;
```

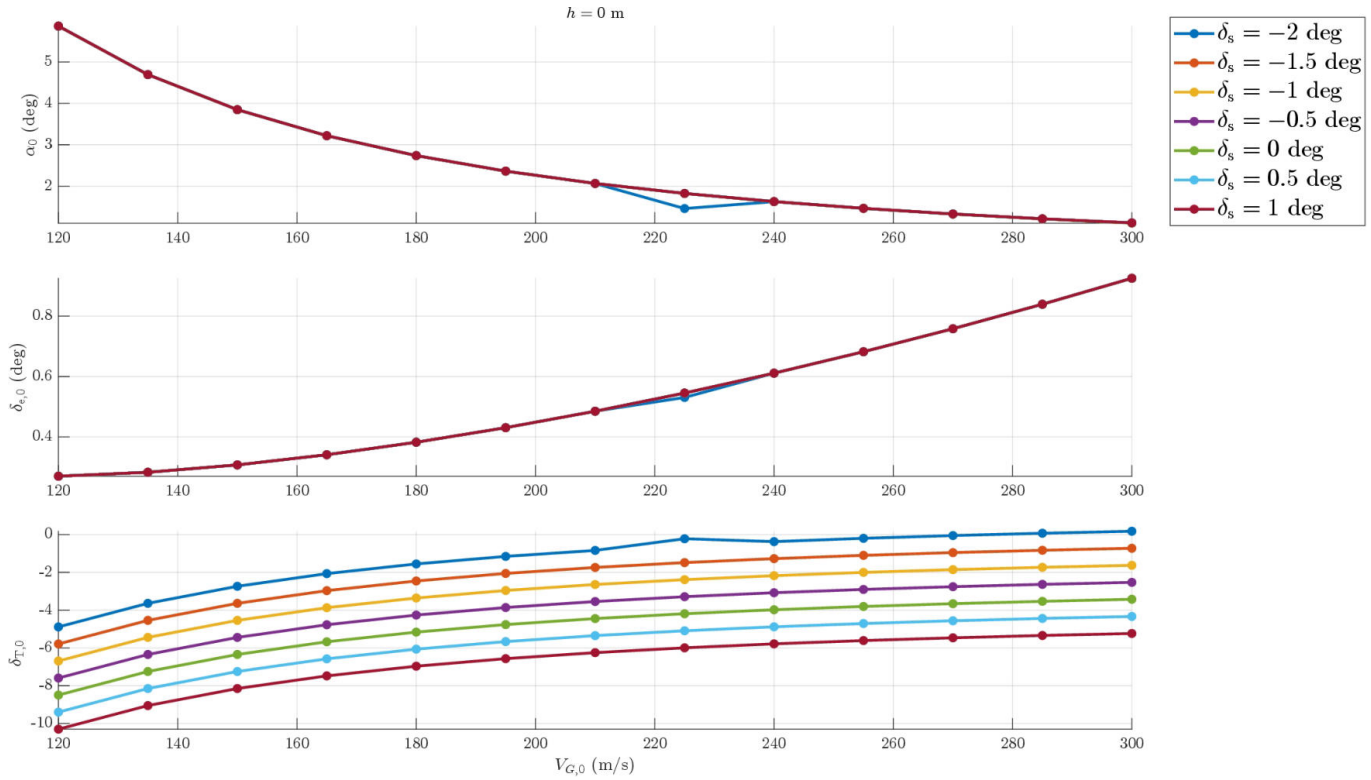


Figure 2.33 Trim results of the absolute angle of attack, the throttle and the elevator deflection angle at sea level for different values of the stabilator deflection angle and with respect to the airspeed.

```
end

function [c, ceq] = nonLinCon(~)
c = [];
ceq = [];
end

Aeq = zeros(4); Aeq(4, 4) = 1;
beq = zeros(4, 1); beq(4) = convang(delta_s_deg, 'deg', 'rad');

trimOptions = optimset('tolfun', 1e-9, 'Algorithm', 'interior-point');
[design, Jval] = fmincon(@(xi) cost(dstate_dt(xi)), ...
design0, ...
[], [], ...
Aeq, beq, ...
lowerBounds, upperBounds, ...
nonLinCon, ...
trimOptions);

end
```

The next script, based upon the above introduced function `ThreeDoFTrimLin.m`, evaluates the design vector for given values of the airspeed, altitude and stabilator deflection angle. In particular, the following samples are assigned

$$V_G \longrightarrow [120,\ 135,\ 150,\ 165,\ 180,\ 195,\ 210,\ 225,\ 240,\ 255,\ 270,\ 285,\ 300]\ \text{m/s}$$

$$h \longrightarrow [0,\ 3000,\ 6000,\ 9000]\ \text{m}$$

$$\delta_s \longrightarrow [-2.0,\ -1.5,\ -1.0,\ -0.5,\ 0,\ 0.5,\ 1.0]\ \text{deg}$$

Listing 2.19 3-DoF trim for different flight parameters.

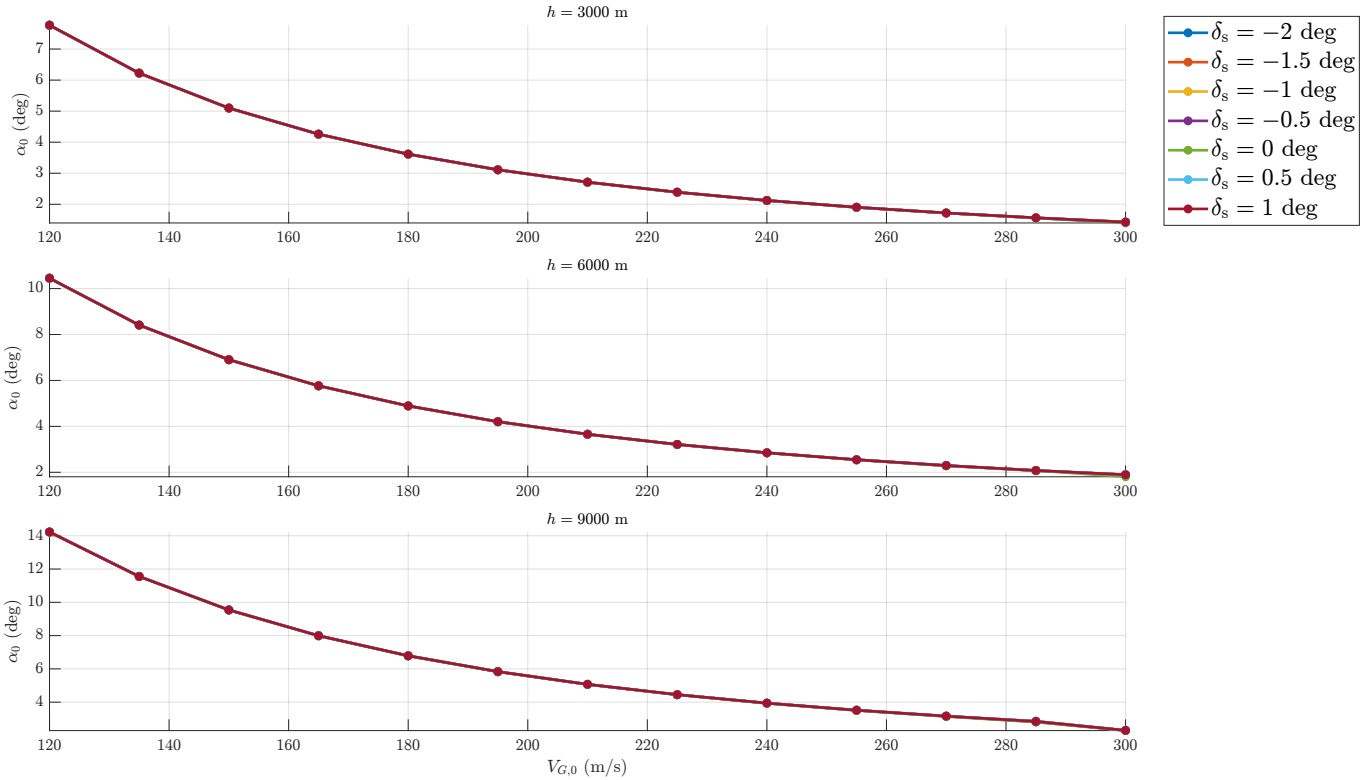


Figure 2.34 Trim results of the absolute angle of attack at different altitudes, for different values of the stabilator deflection angle and with respect to the airspeed.

```
close all; clear; clc

aircraftDataFileName = 'DSV_Aircraft_data.txt';
myAircraft = DSVAircraft(aircraftDataFileName);

airspeeds = 120 : 15 : 300;
Nv = length(airspeeds);

altitudes0 = 0 : 3000 : 9000;
Nh = length(altitudes0);

deltas_s_deg = -2.0 : 0.5 : 1.0;

gamma0_deg = 0;

design0 = [0; 0.5; 0; 0];
lowerBounds = [convang(-15, 'deg', 'rad'); ...
               0.0; ...
               convang(-24, 'deg', 'rad'); ...
               convang(-3, 'deg', 'rad')];
upperBounds = [convang(18, 'deg', 'rad'); ...
               1.0; ...
               convang(16, 'deg', 'rad'); ...
               convang(3, 'deg', 'rad')];

alphas0 = zeros(Nv, Nh, Nd);
deltas_T0 = alphas0;
deltas_e0 = alphas0;
Ntot = Nv * Nh * Nd;
iteration = 0;
for iv = 1 : Nv
    for ih = 1 : Nh
        for id = 1 : Nd
            [design, cost] = ...
                ThreeDoFTrimLin(myAircraft, ...
                                airspeeds(iv), altitudes0(ih), ...
```

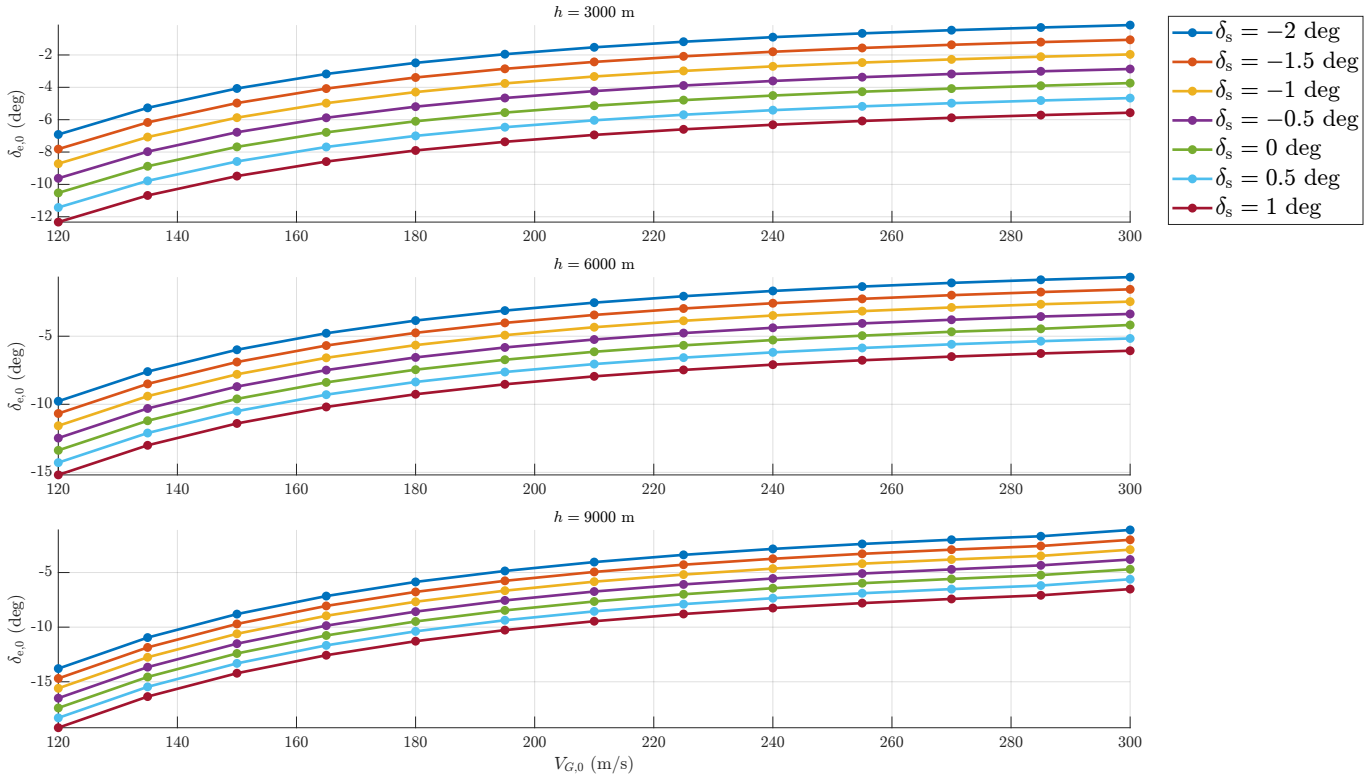


Figure 2.35 Trim results of the elevator deflection angle at different altitudes, for different values of the stabilator deflection angle and with respect to the airspeed.

```
                                deltas_s_deg(id), gamma0_deg, ...
                                design0, lowerBounds, upperBounds);
            alphas0(iv, ih, id) = design(1);
            deltas_T0(iv, ih, id) = design(2);
            deltas_e0(iv, ih, id) = design(3);
            iteration = iteration + 1;
            fprintf("Iteration %d/%d\n", iteration, Ntot)
        end
    end
end

alphas0_deg = convang(alphas0, 'rad', 'deg');
deltas_e0_deg = convang(deltas_e0, 'rad', 'deg');

altitude_titles = cell(Nh-1, 1);
for ih = 1 : Nh-1
    altitude_titles{ih} = strcat( ...
        "$h = ", num2str(altitudes0(ih+1)), "$ m");
end

delta_s_labels = cell(Nd, 1);
for id = 1 : Nd
    delta_s_labels{id} = strcat( ...
        "$\delta_\mathrm{s} = ", num2str(deltas_s_deg(id)), "$ deg");
end

stackedPlot3(airspeeds, ...
            alphas0_deg(:, 1, :), ...
            deltas_T0(:, 1, :), ...
            deltas_e0_deg(:, 1, :), ...
            {"$V_{G,0}$ (m/s)", "$\alpha_0$ (deg)", ...
                "$\delta_{\mathrm{e},0}$ (deg)", ...
                "$\delta_{\mathrm{T},0}$ (deg)"}, ...
            {strcat("$h = ", num2str(altitudes0(1)), "$ m"), "", ""}, ...
            delta_s_labels, 'ex3dofTtrimsh0.pdf')

stackedPlot3(airspeeds, ...
```

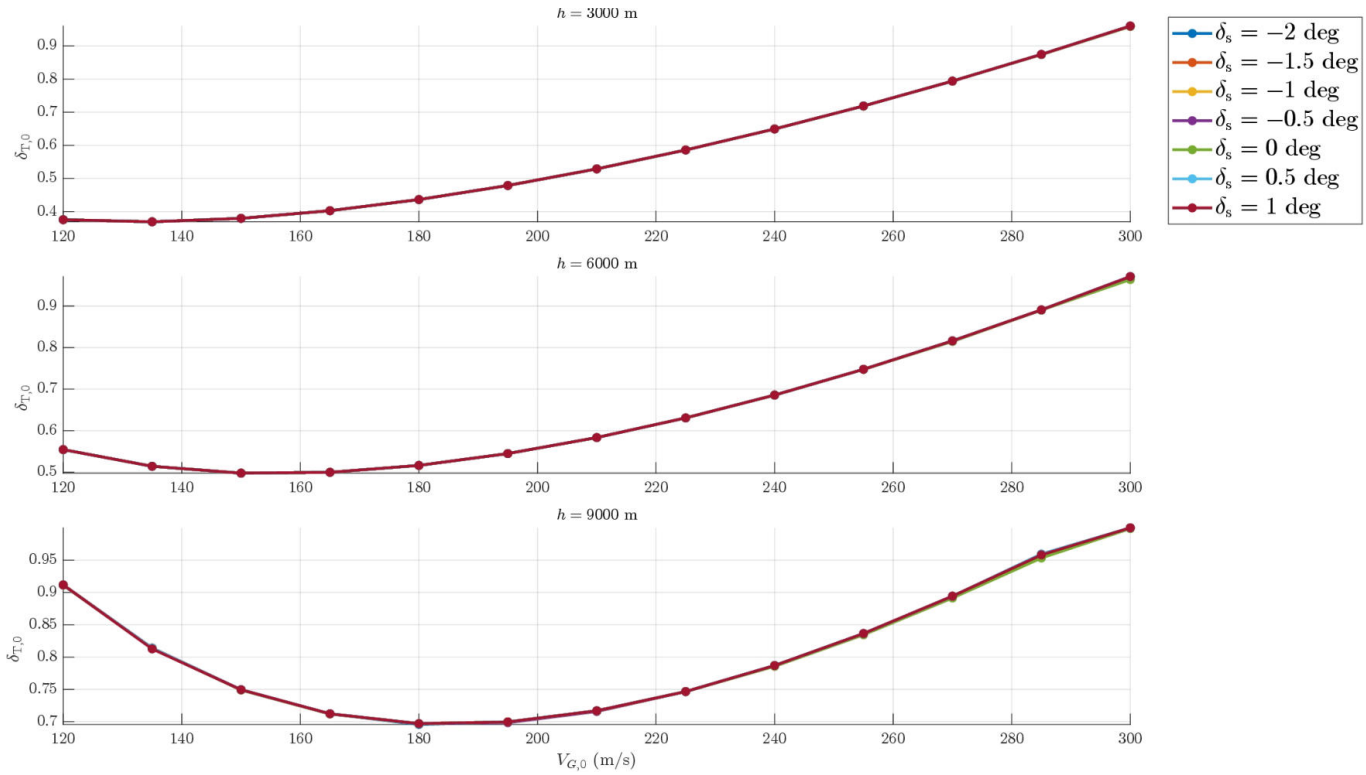


Figure 2.36 Trim results of the throttle at different altitudes, for different values of the stabilator deflection angle and with respect to the airspeed.

```
            alphas0_deg(:, 2, :), ...
            alphas0_deg(:, 3, :), ...
            alphas0_deg(:, 4, :), ...
            {"$V_{G,0}$ (m/s)", "$\alpha_0$ (deg)", ...
                "$\alpha_0$ (deg)", "$\alpha_0$ (deg)"}, ...
            altitude_titles, delta_s_labels, 'ex3dofTalphas.pdf')

stackedPlot3(airspeeds, ...
            deltas_T0(:, 2, :), ...
            deltas_T0(:, 3, :), ...
            deltas_T0(:, 4, :), ...
            {"$V_{G,0}$ (m/s)", "$\delta_{\mathrm{T},0}$ (deg)", ...
                "$\delta_{\mathrm{T},0}$ (deg)", ...
                "$\delta_{\mathrm{T},0}$ (deg)"}, ...
            altitude_titles, delta_s_labels, 'ex3dofTdeltast.pdf')

stackedPlot3(airspeeds, ...
            deltas_e0_deg(:, 2, :), ...
            deltas_e0_deg(:, 3, :), ...
            deltas_e0_deg(:, 4, :), ...
            {"$V_{G,0}$ (m/s)", "$\delta_{\mathrm{e},0}$ (deg)", ...
                "$\delta_{\mathrm{e},0}$ (deg)", ...
                "$\delta_{\mathrm{e},0}$ (deg)"}, ...
            altitude_titles, delta_s_labels, 'ex3dofTdeltase.pdf')

t_end = 30;

alpha0 = design(1);
delta_T0 = design(2);
delta_e0_deg = convang(design(3), 'rad', 'deg');
delta_s0_deg = convang(design(4), 'rad', 'deg');

elevation0 = alpha0 - myAircraft.mu_x + ...
                              convang(gamma0_deg, 'deg', 'rad');
state0 = [airspeed, alpha0, 0, 0, -altitude0, elevation0].';

delta_T_law = @(t) interp1([0, 1] * t_end, ...
```

```
                              [1, 1] * delta_T0, ...
                              t, 'linear');

delta_e_deg_law = @(t) interp1([0, 1] * t_end, ...
                                [1, 1] * delta_e0_deg, ...
                                t, 'linear');

delta_s_deg_law = @(t) interp1([0, 1] * t_end, ...
                                [1, 1] * delta_s0_deg, ...
                                t, 'linear');

[time, ...
delta_T, delta_e_deg, delta_s_deg, ...
V, alpha, q, x_EG, z_EG, elevation] = ThreeDoFVabLin(t_end, state0, ...
                                                     myAircraft, ...
                                                     delta_T_law, ...
                                                     delta_e_deg_law, ...
    ↪ ...
                                                     delta_s_deg_law);

Nt = length(time);

stackedPlot2(time, ...
             delta_T, delta_e_deg, ...
             {"Simulation time (s)", ...
                 "$\delta_\mathrm{T}$", "$\delta_\mathrm{e}$ (deg)"}, ...
             {}, {}, 'ex3dofTinputcommands.pdf')

alpha_deg = convang(alpha, 'rad', 'deg');
stackedPlot2(time, ...
             V, alpha_deg, ...
             {"Simulation time (s)", ...
                 "Airspeed (m/s)", "$\alpha$ (deg)"}, ...
             {}, {}, 'ex3dofTairspeedAoA.pdf')

elevation_deg = convang(elevation, 'rad', 'deg');
q_degps = convangvel(q, 'rad/s', 'deg/s');
stackedPlot2(time, ...
             elevation_deg, q_degps, ...
             {"Simulation time (s)", ...
                 "$\theta$ (deg)", "q (deg/s)"}, ...
             {}, {}, 'ex3dofTelevationq.pdf')

stackedPlot2(time, ...
             x_EG, -z_EG, ...
             {"Simulation time (s)", ...
                 "$x_{\mathrm{E},G}$ (m)", "Altitude (m)"}, ...
             {}, {}, 'ex3dofTcogxz.pdf')

gamma_deg = elevation_deg - ...
    (alpha_deg - convang(myAircraft.mu_x, 'rad', 'deg'));
multiPlot(time, ...
             "Simulation time (s)", "$\gamma$ (deg)", {}, ...
             'ex3dofgamma.pdf', gamma_deg);

O = zeros(Nt, 1);
trajectoryView(time, ...
                x_EG, O, z_EG, ...
                O, elevation, O, ...
                75, 6, ...
                'ex3dofTtrajectory.pdf')
```

The results are shown in the above figures 2.33 to 2.36. As expected, only the trimmed elevator deflection angle $\delta_{e,0}$ shows an actual dependence on the stabilator deflection angle.

It is also worth noticing the increments in the trimmed throttle values when the altitude increases. This trend is due to the thrust model used in this exercise, which is the same as the one used in the 6-DoF F/A-18 thrust model eercise in § 2.3.1. In fact, as already discussed, the maximum thrust available decreases when the altitude increases

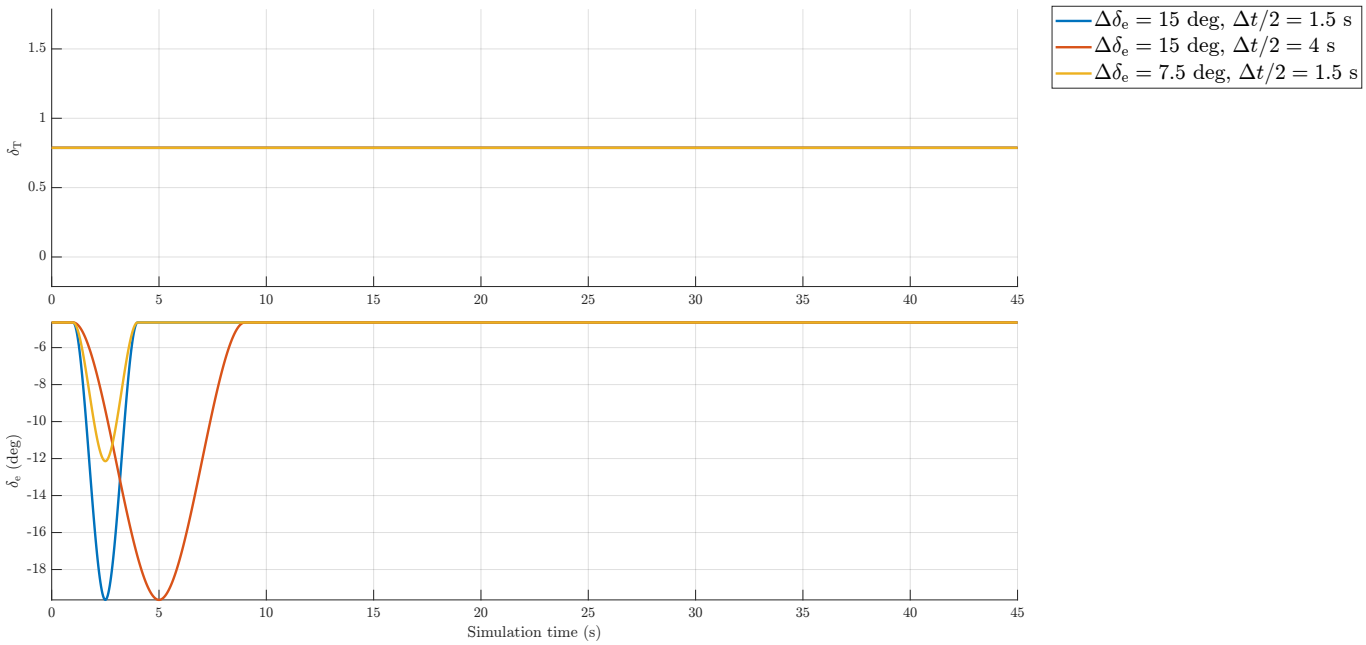


Figure 2.37 Assigned time laws of the input commands.

and this requires an higher throttle value.

Impulsive climbing perturbation

In this exercise it is discussed the aircraft motion evolution when an impulsive perturbation from a trimmed flight condition is done. In particular, the case of an impulsive climbing command can be seen here.

From a trim condition, the elevator is rapidly deflected by an amount of $-\Delta\delta_e$ and then it returns at the previous deflection. The full duration Δt of such perturbation is small when compared with the duration of the simulation.

The simulation can then be parametrized using $\Delta\delta_e$ and $\Delta t/2$. Therefore, different simulations can be carried out to observe slightly different perturbed evolutions. In this case the following three simulations have been done, giving the input commands shown in figure 2.37.

- $\Delta\delta_e = 15$ deg and $\Delta t/2 = 1.5$ s.
- $\Delta\delta_e = 15$ deg and $\Delta t/2 = 4$ s.
- $\Delta\delta_e = 7.5$ deg and $\Delta t/2 = 1.5$ s.

These simulations differ from each other either by the duration of the perturbation or by its intensity.

The exercise starts with researching the trim conditions for the following desired flight parameters and stabilator deflection angle. Observe the necessity to assign a fixed value for δ_s, since it and δ_e are not independent in the longitudinal equilibrium optimization.

- $V_{G,0} = 240$ m/s, $h_0 = 9000$ m and $\gamma_0 = 0$ deg
- $\delta_{s,0} = -1$ deg

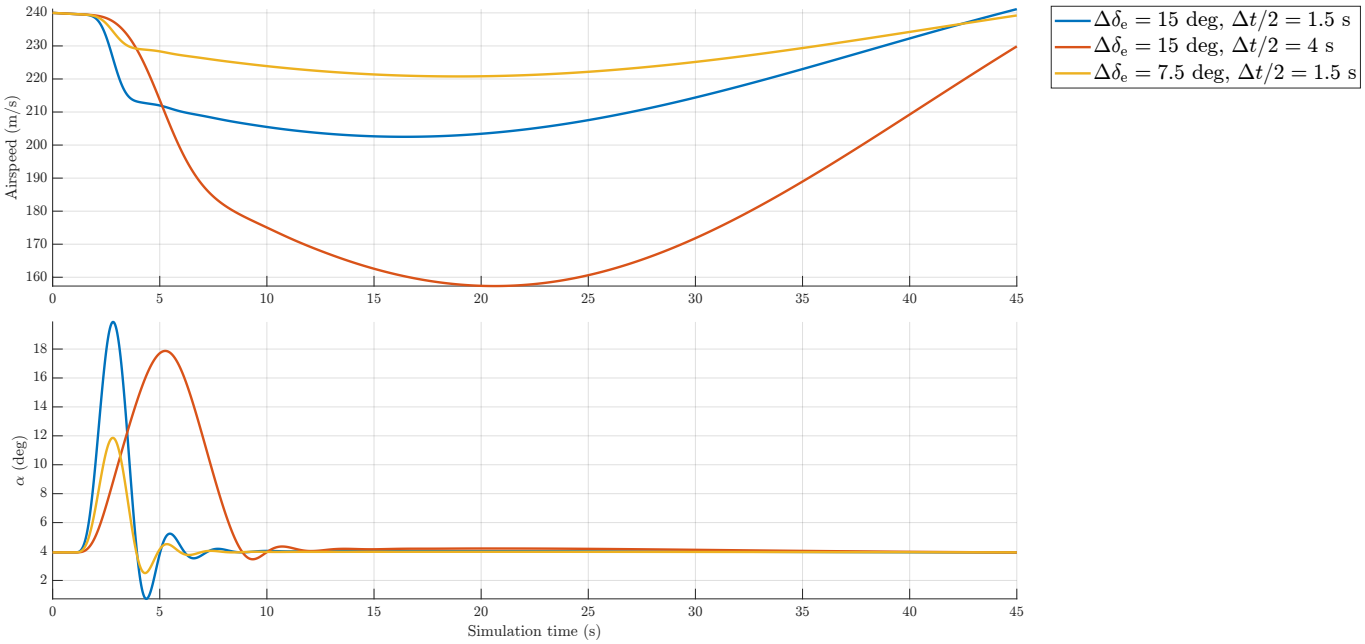


Figure 2.38 Time histories of the airspeed and the absolute angle of attack.

To sum up the results of the flight trim, data are written into `excabrapicchiatrimTable.txt` file. It is worth noticing the minimized cost value $J_{\min} = 1.87 \times 10^{-10} \approx 0$ that ensures the equilibrium conditions have been satisfied.

```
Trimmed leveled symmetrical flight
h = 9000 m, V = 240 m/s
---------------------------------------
Absolute AoA (deg)              3.94
Throttle                        0.787
Elevator deflection (deg)       -4.64
Stabilator deflection (deg)     -1
Cost value                      1.87e-10
```

Now, all the three simulations can be carried out. Each one of them has the trim data as initial conditions, but a different elevator deflection angle time law. See figures 2.38, 2.39 and 2.40 for the resulted time histories of the state variables and figures 2.41, 2.42 and 2.43 for the three different trajectories obtained.

Listing 2.20 Different simulations of an impulsive climbing perturbation from a trim condition.

```
close all; clear; clc

aircraftDataFileName = 'DSV_Aircraft_data.txt';
myAircraft = DSVAircraft(aircraftDataFileName);

V0 = 240;
h0 = 9000;
gamma0_deg = 0;

delta_s_deg = -1;

design0 = [0; 0.5; 0; 0];
lowerBounds = [convang(-15, 'deg', 'rad'); ...
                0.0; ...
                convang(-24, 'deg', 'rad'); ...
                convang(-3, 'deg', 'rad')];
upperBounds = [convang(18, 'deg', 'rad'); ...
                1.0; ...
```

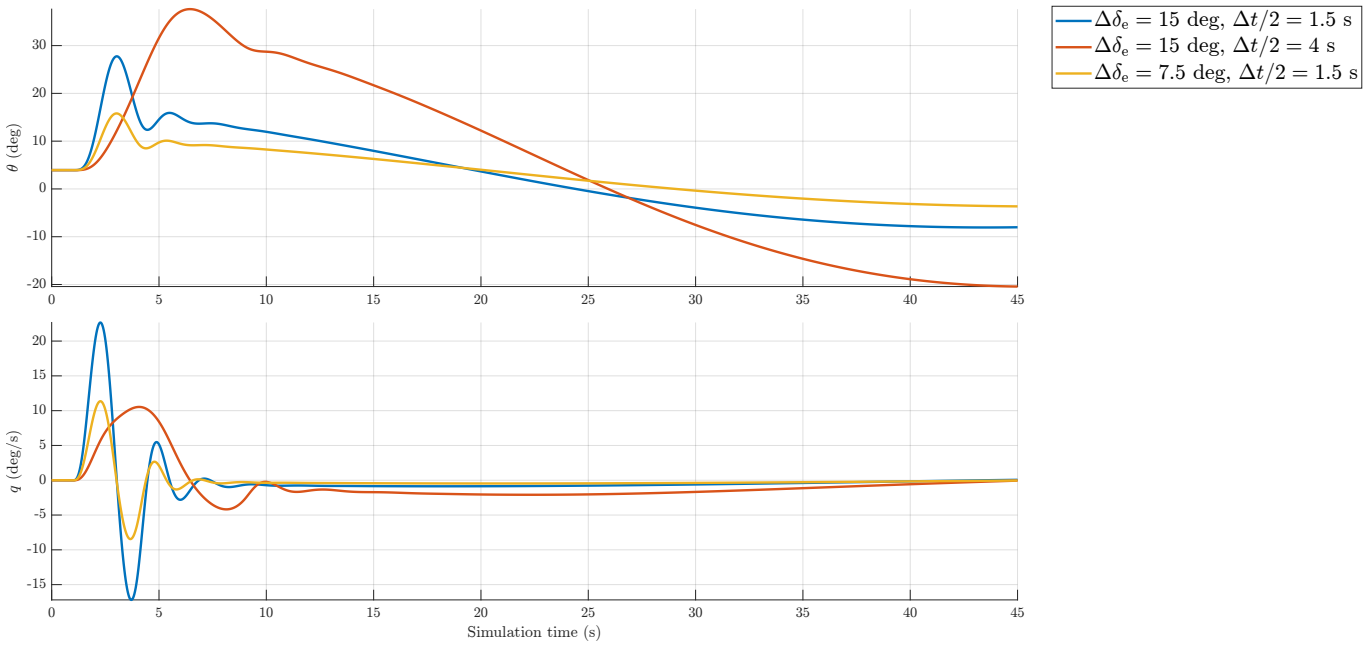


Figure 2.39 Angular orientation and pitch rate of the aircraft time histories.

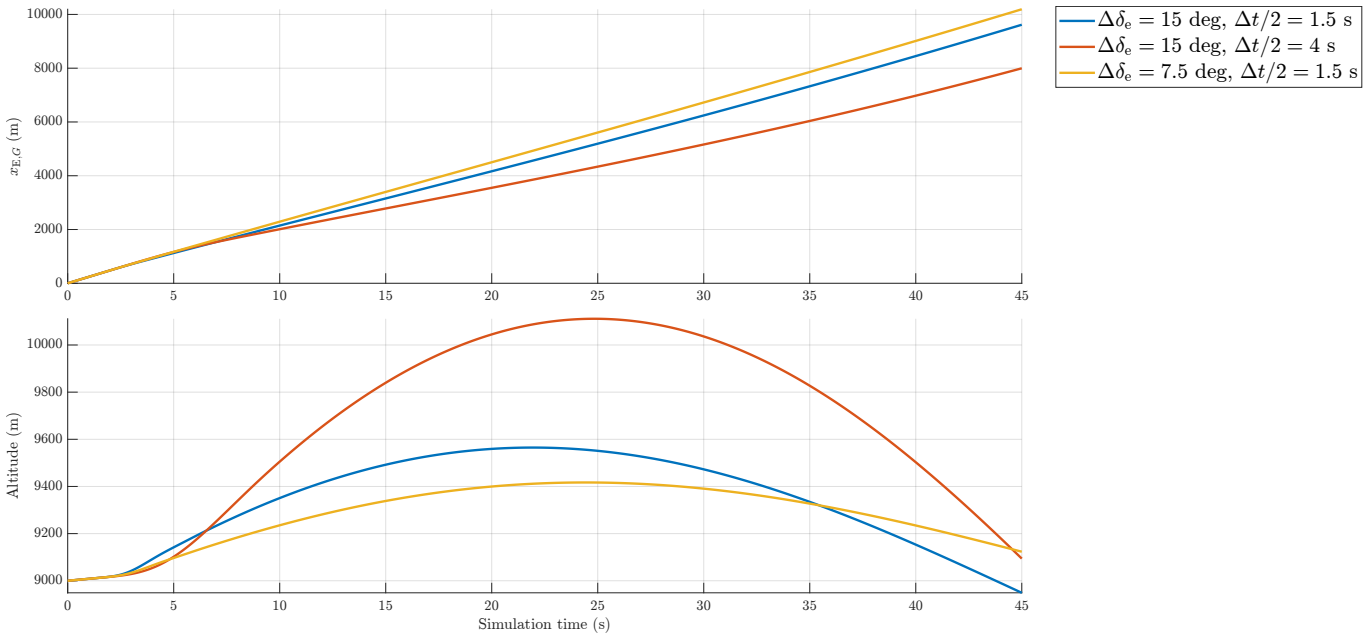


Figure 2.40 Center of gravity position over time.

```
                convang(16, 'deg', 'rad'); ...
                convang(3, 'deg', 'rad')];

[design, cost] = ThreeDoFTrim(myAircraft, ...
                              V0, h0, ...
                              delta_s_deg, gamma0_deg, ...
                              design0, lowerBounds, upperBounds);

alpha0 = design(1);
alpha0_deg = convang(alpha0, 'rad', 'deg');

delta_T0 = design(2);

delta_e0 = design(3);
delta_e0_deg = convang(delta_e0, 'rad', 'deg');
```

```
34
35 delta_s0 = design(4);
36 delta_s0_deg = convang(delta_s0, 'rad', 'deg');
37
38 trimTable = table({" "; ...
39                     strcat("V = ", num2str(V0), " m/s"); ...
40                     round(alpha0_deg, 3, 'Significant'); ...
41                     round(delta_T0, 3, 'Significant'); ...
42                     round(delta_e0_deg, 3, 'Significant'); ...
43                     round(delta_s0_deg, 3, 'Significant'); ...
44                     round(cost, 3, 'Significant')}, ...
45                     'RowNames', ...
46                     ["Trimmed leveled symmetrical flight", ...
47                     strcat("h = ", num2str(h0), " m"), ...
48                     "Absolute AoA (deg)", "Throttle", ...
49                     "Elevator deflection (deg)", ...
50                     "Stabilator deflection (deg)", "Cost value"]);
51 writetable(trimTable, 'excabrapicchiatrimTable.txt', 'Delimiter', ',',
       ↪ ...
52             'WriteVariableNames', false, 'WriteRowNames', true);
53
54 t_end = 45;
55 delta_e_deg_impulse = [15, 15, 7.5];
56 half_duration = [1.5, 4, 1.5];
57
58 state0 = [V0; alpha0; 0; 0; -h0; alpha0];
59
60 time_cell = cell(3, 1);
61 delta_T_cell = time_cell;
62 delta_e_deg_cell = time_cell;
63 delta_s_deg_cell = time_cell;
64 V_cell = time_cell;
65 alpha_cell = time_cell;
66 q_cell = time_cell;
67 x_EG_cell = time_cell;
68 z_EG_cell = time_cell;
69 elevation_cell = time_cell;
70 Nt_cell = time_cell;
71 for is = 1 : 3
72 delta_T_law = @(t) interp1([0, 1] * t_end, ...
73                            [1, 1] * delta_T0, ...
74                            t, 'linear');
75
76 delta_e_deg_law = @(t) interp1(...
77                     [0, 1, ...
78                         1+half_duration(is), 1+2*half_duration(is), ...
79                         t_end], ...
80                     [delta_e0_deg, delta_e0_deg, ...
81                         delta_e0_deg-delta_e_deg_impulse(is), ...
82                         delta_e0_deg, delta_e0_deg], ...
83                     t, 'pchip');
84
85 delta_s_deg_law = @(t) interp1([0, 1] * t_end, ...
86                                [1, 1] * delta_s0_deg, ...
87                                t, 'linear');
88
89 [time, ...
90 delta_T, delta_e_deg, delta_s_deg, ...
91 V, alpha, q, x_EG, z_EG, elevation] = ...
92             ThreeDoFVabLin(t_end, state0, myAircraft, ...
93                            delta_T_law, delta_e_deg_law,
       ↪ delta_s_deg_law);
94
95 time_cell{is} = time;
96 Nt_cell{is} = length(time);
97 delta_T_cell{is} = delta_T;
98 delta_e_deg_cell{is} = delta_e_deg;
99 delta_s_deg_cell{is} = delta_s_deg;
100 V_cell{is} = V;
101 alpha_cell{is} = alpha;
102 q_cell{is} = q;
103 x_EG_cell{is} = x_EG;
```

```
z_EG_cell{is} = z_EG;
elevation_cell{is} = elevation;
end

Nt = min([Nt_cell{1}, Nt_cell{2}, Nt_cell{3}]);
time = linspace(0, t_end, Nt).';

interpFunc = @(data1, data2) interp1(data1, data2, time, 'pchip');

delta_T = [interpFunc(time_cell{1}, delta_T_cell{1}), ...
           interpFunc(time_cell{2}, delta_T_cell{2}), ...
           interpFunc(time_cell{3}, delta_T_cell{3})];

delta_e_deg = [interpFunc(time_cell{1}, delta_e_deg_cell{1}), ...
               interpFunc(time_cell{2}, delta_e_deg_cell{2}), ...
               interpFunc(time_cell{3}, delta_e_deg_cell{3})];

delta_s_deg = [interpFunc(time_cell{1}, delta_s_deg_cell{1}), ...
               interpFunc(time_cell{2}, delta_s_deg_cell{2}), ...
               interpFunc(time_cell{3}, delta_s_deg_cell{3})];

V = [interpFunc(time_cell{1}, V_cell{1}), ...
    interpFunc(time_cell{2}, V_cell{2}), ...
    interpFunc(time_cell{3}, V_cell{3})];

alpha = [interpFunc(time_cell{1}, alpha_cell{1}), ...
        interpFunc(time_cell{2}, alpha_cell{2}), ...
        interpFunc(time_cell{3}, alpha_cell{3})];
alpha_deg = convang(alpha, 'rad', 'deg');

q = [interpFunc(time_cell{1}, q_cell{1}), ...
    interpFunc(time_cell{2}, q_cell{2}), ...
    interpFunc(time_cell{3}, q_cell{3})];
q_degps = convangvel(q, 'rad/s', 'deg/s');

x_EG = [interpFunc(time_cell{1}, x_EG_cell{1}), ...
        interpFunc(time_cell{2}, x_EG_cell{2}), ...
        interpFunc(time_cell{3}, x_EG_cell{3})];

z_EG = [interpFunc(time_cell{1}, z_EG_cell{1}), ...
        interpFunc(time_cell{2}, z_EG_cell{2}), ...
        interpFunc(time_cell{3}, z_EG_cell{3})];

elevation = [interpFunc(time_cell{1}, elevation_cell{1}), ...
           interpFunc(time_cell{2}, elevation_cell{2}), ...
           interpFunc(time_cell{3}, elevation_cell{3})];
elevation_deg = convang(elevation, 'rad', 'deg');

lgnd = {sprintf("$\\Delta\\delta_\\mathrm{e} = %s$ deg, $\\Delta t/2 =
    ↪ %s$ s", ...
                num2str(delta_e_deg_impulse(1)), ...
                num2str(half_duration(1))), ...
        sprintf("$\\Delta\\delta_\\mathrm{e} = %s$ deg, $\\Delta t/2 =
    ↪ %s$ s", ...
                num2str(delta_e_deg_impulse(2)), ...
                num2str(half_duration(2))), ...
        sprintf("$\\Delta\\delta_\\mathrm{e} = %s$ deg, $\\Delta t/2 =
    ↪ %s$ s", ...
                num2str(delta_e_deg_impulse(3)), ...
                num2str(half_duration(3)))};

stackedPlot2(time, delta_T, delta_e_deg, ...
           {"Simulation time (s)", ...
               "$\delta_\mathrm{T}$", "$\delta_\mathrm{e}$ (deg)"}, ...
           {}, lgnd, 'excabrapicchiainputcommands.pdf')

stackedPlot2(time, V, alpha_deg, ...
           {"Simulation time (s)", "Airspeed (m/s)", "$\alpha$ (deg)"},
    ↪ ...
           {}, lgnd, 'excabrapicchiaVa.pdf')

stackedPlot2(time, elevation_deg, q_degps, ...
```

```
172             {"Simulation time (s)", "$\theta$ (deg)", "$q$ (deg/s)"}, ...
173             {}, lgnd, 'excabriapicchiathetaq.pdf')
174
175 stackedPlot2(time, x_EG, -z_EG, ...
176             {"Simulation time (s)", ...
177                 "$x_{\mathrm{E},G}$ (m)", "Altitude (m)"}, ...
178             {}, lgnd, 'excabrapicchiacogxh.pdf')
179
180 y_EG = zeros(Nt, 1);
181 phi = y_EG;
182 psi = y_EG;
183 for is = 1 : 3
184     trajectoryView(time, x_EG(:, is), y_EG, z_EG(:, is), ...
185                    phi, elevation(:, is), psi, ...
186                    650, 6, ...
187                    sprintf('excabrapicchiatrajectory%s.pdf',
    ↪ num2str(is)))
188 end
```

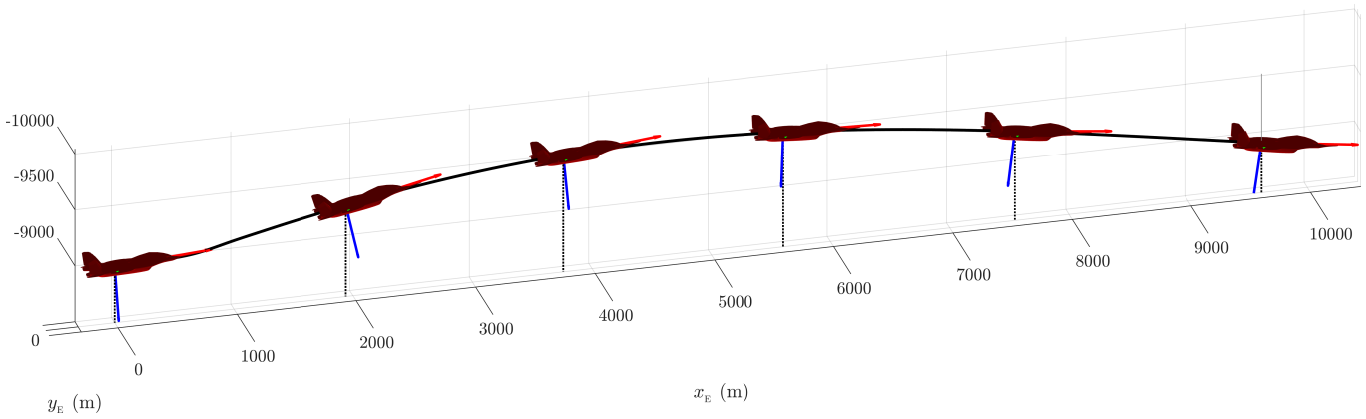


Figure 2.41 Trajectory of the simulation obtained from the input commands given by the assigned parameters $\Delta\delta_e = 15$ deg and $\Delta t/2 = 1.5$ s.

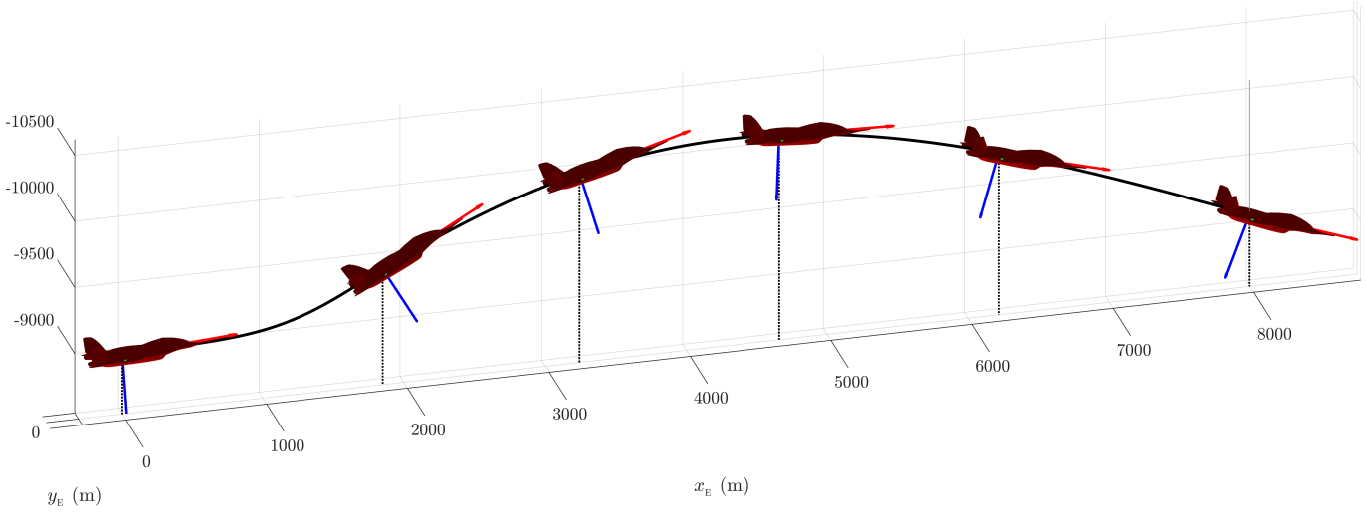


Figure 2.42 Trajectory of the simulation obtained from the input commands given by the assigned parameters $\Delta\delta_e = 15$ deg and $\Delta t/2 = 4$ s.

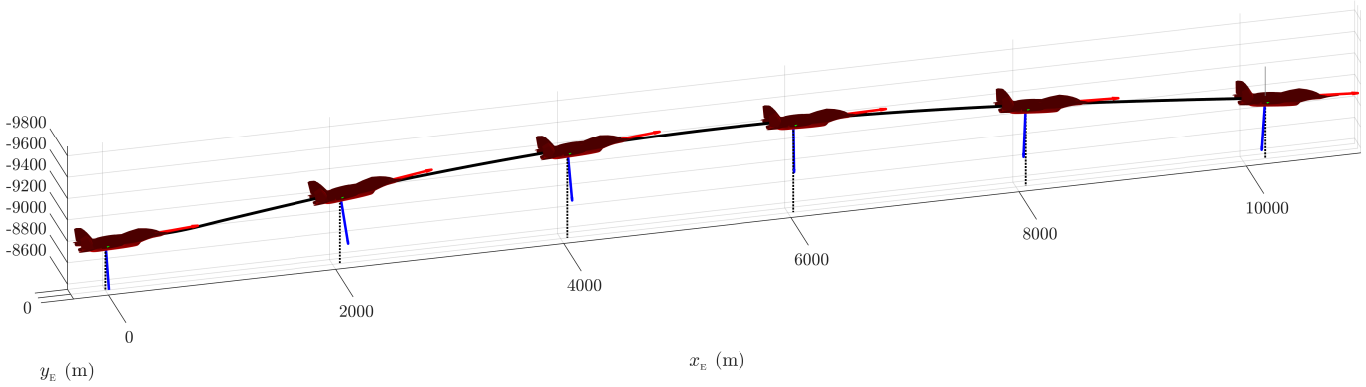


Figure 2.43 Trajectory of the simulation obtained from the input commands given by the assigned parameters $\Delta\delta_e = 7.5$ deg and $\Delta t/2 = 1.5$ s.

2.6 Control surfaces

Up to this point we have considered the aircraft motion as a rigid motion despite the reciprocal motions between its control surfaces and its body. This approximation is not problematic, since control surfaces motions are negligible when compared with the aircraft body motion, during any maneuver. Nevertheless, it is useful to consider such motions, to fully understand how control surfaces behave by a dynamical point of view.

In pursuit of a dynamical characterization of the control surfaces, some terminology and conventions are defined. Everything relating to a generic control surface will be subscripted with letters CS, that stands for *control surface*. A new frame of reference is also defined to be solidal with the generic control surface motion, it is $\mathcal{F}_{\text{CS}} = \{C,\ c,\ t,\ n\}$, where c is the hinge axis, t is the chordwise axis orthogonal to c and n is chosen accordingly. The center of the reference frame C is centered along c. Defining $\mathcal{F}_{\text{CS}}$ like that makes the control surface center of gravity G_{CS} lying on t, so its distance from C is defined as e_{CS}.

With this setting, it is possible to introduce the hinge moment $\mathscr{H}_{\text{CS}}$ as the component of the resultant moment acting on the control surface along the c axis. In particular,

$$\mathscr{H}_{\text{CS}} = \mathscr{H}_{\text{CS,A}} + \mathscr{H}_{\text{CS,C}} + m_{\text{CS}} g_n e_{\text{CS}} \tag{2.98}$$

where $\mathscr{H}_{\text{CS,A}}$ is due to the aerodynamic moment, $\mathscr{H}_{\text{CS,C}}$ is caused by the imposed command and $m_{\text{CS}} g_n e_{\text{CS}}$ is the moment produced by the weight of the control surface applied on its center of gravity. It can be trivially deduced that to obtain static equilibrium, the condition $e_{\text{CS}} = 0$ is required. It can be reached through *balancing dummy masses* accurately placed along the chordwise direction.

By recalling equation (2.25) it is possible to obtain a differential equation for the control surface deflection. In particular we can rewrite it as

$$\left(I_{\text{CS}} \frac{\partial \boldsymbol{\Omega}_{\text{CS}}}{\partial t} + \boldsymbol{\Omega}_{\text{CS}} \times I_{\text{CS}} \boldsymbol{\Omega}_{\text{CS}} \right)_c = \mathscr{H}_{\text{CS}} \tag{2.99}$$

where I_{CS} is the tensor of inertia of the control surface and $\boldsymbol{\Omega}_{\text{CS}}$ is its angular velocity vector with respect to the Earth reference frame.

Since control surfaces typically do not have a significant camber, it is appropriate to assume a symmetrical shape for the control surface. This, together with the t axis being baricentric, yields all the products of inertia to be zero. Therefore, the matrix representation of the tensor of inertia is diagonal. In addition, we can approximate the control surface as a thin walled plate lying on the ct plane. This last assumption gives

$$I_n = I_c + I_t \tag{2.100}$$

Putting all together yields to the following differential equation

$$I_c \left[\dot{\Omega}_{\text{CS}_c} + \Omega_{\text{CS}_t} \Omega_{\text{CS}_n} \right] + m_{\text{CS}} a_{C_n} e_{\text{CS}} = \mathscr{H}_{\text{CS}} \tag{2.101}$$

We can express the angular velocity vector of the control surface as the sum of its relative motion with respect to the aircraft and the aircraft motion itself, to obtain the

following.

$$\boldsymbol{\Omega}_{\mathrm{CS}} = \boldsymbol{\omega}_{\mathrm{CS}} + \boldsymbol{\Omega}_{\mathrm{B}} = \dot{\delta}_{\mathrm{CS}}\boldsymbol{i}_{\mathrm{CS}} + p\boldsymbol{i}_{\mathrm{B}} + q\boldsymbol{j}_{\mathrm{B}} + r\boldsymbol{k}_{\mathrm{B}} \tag{2.102}$$

where, in case of a control surface having the plane ct parallel to the plane $x_{\mathrm{B}}y_{\mathrm{B}}$,

$$\begin{cases} \boldsymbol{i}_{\mathrm{B}} = -\boldsymbol{i}_{\mathrm{CS}} \sin \Lambda_{\mathrm{CS}} - \boldsymbol{j}_{\mathrm{CS}} \cos \Lambda_{\mathrm{CS}} \\ \boldsymbol{j}_{\mathrm{B}} = \boldsymbol{i}_{\mathrm{CS}} \cos \Lambda_{\mathrm{CS}} - \boldsymbol{j}_{\mathrm{CS}} \sin \Lambda_{\mathrm{CS}} \\ \boldsymbol{k}_{\mathrm{B}} = \boldsymbol{k}_{\mathrm{CS}} \end{cases} \tag{2.103}$$

with $\boldsymbol{i}_{\mathrm{CS}}$, $\boldsymbol{j}_{\mathrm{CS}}$ and $\boldsymbol{k}_{\mathrm{CS}}$ being the versor of the c, t and n axes respectively and Λ_{CS} being the control surface sweep angle taken on the c axis.

Substituting equations (2.103) into the angular velocity vector equation (2.102) we can obtain its projection onto $\mathcal{F}_{\mathrm{CS}}$ axes. These and equation (2.98) into equation (2.99) yields the following.

$$\begin{aligned} I_c \ddot{\delta}_{\mathrm{CS}} - (\dot{p} + qr) I_c \sin \Lambda_{\mathrm{CS}} + (\dot{q} - pr) I_c \cos \Lambda_{\mathrm{CS}} \\ + m_{\mathrm{CS}} (a_{C_{z_{\mathrm{B}}}} - g_{z_{\mathrm{B}}}) e_{\mathrm{CS}} = \mathscr{H}_{\mathrm{CS,A}} + \mathscr{H}_{\mathrm{CS,C}} \end{aligned} \tag{2.104}$$

Using

$$\boldsymbol{a}_C = \boldsymbol{a}_G + \dot{\boldsymbol{\Omega}}_{\mathrm{B}} \times (C - G) + \dot{\boldsymbol{\Omega}}_{\mathrm{B}} \times \left[\dot{\boldsymbol{\Omega}}_{\mathrm{B}} \times (C - G) \right] \tag{2.105}$$

we obtain

$$a_{C_{z_{\mathrm{B}}}} = a_{G_{z_{\mathrm{B}}}} + (\dot{p} + qr) y_{\mathrm{B},C} - (\dot{q} - pr) x_{\mathrm{B},C} \tag{2.106}$$

that can be substituted into equation (2.104) to get

$$\begin{aligned} I_c \ddot{\delta}_{\mathrm{CS}} + (\dot{p} + qr) I_{cx_{\mathrm{B}}} - (\dot{q} - pr) I_{cy_{\mathrm{B}}} \\ + m_{\mathrm{CS}} (a_{G_{z_{\mathrm{B}}}} - g_{z_{\mathrm{B}}}) e_{\mathrm{CS}} = \mathscr{H}_{\mathrm{CS,A}} + \mathscr{H}_{\mathrm{CS,C}} \end{aligned} \tag{2.107}$$

where

$$I_{cx_{\mathrm{B}}} = m_{\mathrm{CS}} e_{\mathrm{CS}} y_{\mathrm{B},C} - I_c \sin \Lambda_{\mathrm{CS}} \tag{2.108}$$

$$I_{cy_{\mathrm{B}}} = m_{\mathrm{CS}} e_{\mathrm{CS}} x_{\mathrm{B},C} - I_c \cos \Lambda_{\mathrm{CS}} \tag{2.109}$$

With a similar approach, changing from equation (2.103), it is possible to obtain the same equation but for a control surface whose ct plane is parallel to the $x_{\mathrm{B}}z_{\mathrm{B}}$ one. In particular, it is possible to operate the substitutions $y_{\mathrm{B}} \longrightarrow -z_{\mathrm{B}}$, $z_{\mathrm{B}} \longrightarrow y_{\mathrm{B}}$, $q \longrightarrow -r$, $r \longrightarrow q$, $I_{cx_{\mathrm{B}}} \longrightarrow -I_{cx_{\mathrm{B}}}$, $I_{cy_{\mathrm{B}}} \longrightarrow I_{cz_{\mathrm{B}}}$, $\delta_{\mathrm{CS}} \longrightarrow -\delta_{\mathrm{CS}}$ and $\mathscr{H}_{\mathrm{CS}} \longrightarrow -\mathscr{H}_{\mathrm{CS}}$. By doing so the equation for such control surface becomes

$$\begin{aligned} I_c \ddot{\delta}_{\mathrm{CS}} + (\dot{p} - qr) I_{cx_{\mathrm{B}}} - (\dot{r} + pq) I_{cz_{\mathrm{B}}} \\ - m_{\mathrm{CS}} (a_{G_{y_{\mathrm{B}}}} - g_{y_{\mathrm{B}}}) e_{\mathrm{CS}} = \mathscr{H}_{\mathrm{CS,A}} + \mathscr{H}_{\mathrm{CS,C}} \end{aligned} \tag{2.110}$$

It is useful pointing out that one can always evaluate the aerodynamic term in the resultant hinge moment expression (2.98) using the aerodynamic hinge moment coefficient of the control surface $C_{\mathscr{H}_{\mathrm{CS}}}$. In particular, it is possible to write

$$\mathscr{H}_{\mathrm{CS,A}} = C_{\mathscr{H}_{\mathrm{CS}}} \frac{1}{2} \rho V_G^2 S_{\mathrm{CS}} \bar{c}_{\mathrm{CS}} \tag{2.111}$$

2.6.1 Elevator, stick free dynamics

The dynamical modeling of the elevator control surface is of particular interest when simulating three degrees of freedom motions. By making the symbolic substitutions CS $\longrightarrow$ e and $c \longrightarrow h_\mathrm{e}$ into equation (2.107) it is possible to obtain the elevator dynamics equation

$$\begin{aligned} I_{h_\mathrm{e}}\ddot{\delta}_\mathrm{e} + (\dot{p} + qr)I_{h_\mathrm{e}x_\mathrm{B}} - (\dot{q} - pr)I_{h_\mathrm{e}y_\mathrm{B}} \\ + m_\mathrm{e}(a_{G_{z_\mathrm{B}}} - g_{z_\mathrm{B}})e_\mathrm{e} = \mathscr{H}_{\mathrm{e,A}} + \mathscr{H}_{\mathrm{e,C}} \end{aligned} \tag{2.112}$$

where

$$I_{h_\mathrm{e}x_\mathrm{B}} = m_\mathrm{e}e_\mathrm{e}y_{\mathrm{B},C} - I_{h_\mathrm{e}} \sin \Lambda_\mathrm{e} \tag{2.113}$$

$$I_{h_\mathrm{e}y_\mathrm{B}} = m_\mathrm{e}e_\mathrm{e}x_{\mathrm{B},C} - I_{h_\mathrm{e}} \cos \Lambda_\mathrm{e} \tag{2.114}$$

To carry out stick free three degrees of freedom simulations, we must recall and expand the dynamical system of equations (2.95) by adding two other differential equations. In fact, during a stick free motion δ_e is no longer an input variable but is indeed a state variable. Moreover, also its time derivative is a state variable.

The stick free three degrees of freedom dynamical system becomes a system of eight differential equations, that can be expressed in the canonical form

$$M(t, \boldsymbol{x})\,\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}) \tag{2.115}$$

where the state vector and the input vector are

$$\boldsymbol{x} = \begin{Bmatrix} V_G \\ \alpha_\mathrm{B} \\ q \\ x_{\mathrm{E},G} \\ z_{\mathrm{E},G} \\ \theta \\ \dot{\delta}_\mathrm{e} \\ \delta_\mathrm{e} \end{Bmatrix}, \qquad \boldsymbol{u} = \begin{Bmatrix} \delta_\mathrm{T} \\ \delta_\mathrm{s} \\ \delta_\mathrm{t} \end{Bmatrix} \tag{2.116}$$

To specify the differential equations of the system appropriate models for the aerodynamic forces, moments and elevator hinge moment are required. In this case, the same linearized models introduced in § 2.4 can be chosen. For the stick free dynamics that set of models must be completed by a similar linearized model of the hinge moment.

Since $\mathscr{H}_{\mathrm{e,C}} \equiv 0$ by definition during stick free motions, it can trivially be stated that $\mathscr{H}_\mathrm{e} \equiv \mathscr{H}_{\mathrm{e,A}} + m_\mathrm{e}g_n e_\mathrm{e}$ in this case. Therefore, the resultant hinge moment can be expressed as

$$\mathscr{H}_\mathrm{e} = C_{\mathscr{H}_\mathrm{e}} \frac{1}{2}\rho V_G^2 S_\mathrm{e}\bar{c}_\mathrm{e} + m_\mathrm{e}g_n e_\mathrm{e} \tag{2.117}$$

where, as said above, the linearized coefficient can be used, i.e.

$$\begin{aligned} C_{\mathscr{H}_\mathrm{e}} = C_{\mathscr{H}_0} + C_{\mathscr{H}_\alpha}\alpha_\mathrm{H} + C_{\mathscr{H}_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{\mathscr{H}_{\delta_\mathrm{s}}}\delta_\mathrm{s} \\ + C_{\mathscr{H}_{\delta_\mathrm{t}}}\delta_\mathrm{t} + \frac{\bar{c}_\mathrm{e}}{2V_G}\left(C_{\mathscr{H}_{\dot{\alpha}}}\dot{\alpha}_\mathrm{H} + C_{\mathscr{H}_q}q + C_{\mathscr{H}_{\dot{\delta}_\mathrm{e}}}\dot{\delta}_\mathrm{e}\right) \end{aligned} \tag{2.118}$$

with α_{H} being the angle of attack seen from the horizontal tail plane, that is affected by the downwas angle ε. In particular it can be evauated as

$$\alpha_{\mathrm{H}} = \left(1 - \frac{\mathrm{d}\varepsilon}{\mathrm{d}\alpha}\right)\alpha_{\mathrm{B}} - \varepsilon_0 + \delta_{\mathrm{s}} + \mu_x \tag{2.119}$$

With the above discussed setting, the three degrees of freedom stick free dynamical system can be obtained from the system of equations (2.95) by considering δ_{e} as a state variable and by adding the following two equations.

$$\begin{aligned}\ddot{\delta}_{\mathrm{e}} = &\left(\frac{e_{\mathrm{e}} x_{\mathrm{B},C_{\mathrm{e}}}}{\kappa_{\mathrm{e}}^2} - \cos\Lambda_{\mathrm{e}}\right)\dot{q} - \frac{e_{\mathrm{e}}}{\kappa_{\mathrm{e}}^2}\left[\dot{V}_G \sin(\alpha - \mu_x)\right.\\ &\left. - V_G q\cos(\alpha - \mu_x) + V_G\dot{\alpha}\cos(\alpha - \mu_x) - g\cos\theta\right]\\ &+ \frac{\rho V_G^2 S_{\mathrm{e}}\bar{c}_{\mathrm{e}}}{2m_{\mathrm{e}}\kappa_{\mathrm{e}}^2}\left\{C_{\mathscr{H}_0} + C_{\mathscr{H}_\alpha}\left[\left(1 - \frac{\mathrm{d}\varepsilon}{\mathrm{d}\alpha}\right)(\alpha - \mu_x) - \varepsilon_0 + \delta_{\mathrm{s}} + \mu_x\right]\right.\\ &\left. + C_{\mathscr{H}_{\delta_{\mathrm{e}}}}\delta_{\mathrm{e}} + C_{\mathscr{H}_{\delta_{\mathrm{s}}}}\delta_{\mathrm{s}} + C_{\mathscr{H}_{\delta_{\mathrm{t}}}}\delta_{\mathrm{t}} + \frac{\bar{c}_{\mathrm{e}}}{2V_G}\left[C_{\mathscr{H}_{\dot{\alpha}}}\left(1 - \frac{\mathrm{d}\varepsilon}{\mathrm{d}\alpha}\right)\dot{V}_G + C_{\mathscr{H}_q} q + C_{\mathscr{H}_{\dot{\delta}_{\mathrm{e}}}}\dot{\delta}_{\mathrm{e}}\right]\right\}\end{aligned} \tag{2.120}$$

$$\dot{\delta}_{\mathrm{e}} = \dot{\delta}_{\mathrm{e}} \qquad (\dot{x}_8 = x_7) \tag{2.121}$$

The dynamical system can be explicitely written in the form recalled in equation (2.115) as

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & M_{32} & 1 & 0 & 0 & 0 & 0 & M_{38}\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ M_{71} & M_{72} & M_{73} & 0 & 0 & M_{76} & 1 & M_{78}\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \dot{V}_G\\ \dot{\alpha}\\ \dot{q}\\ \dot{x}_{\mathrm{E},G}\\ \dot{z}_{\mathrm{E},G}\\ \dot{\theta}\\ \ddot{\delta}_{\mathrm{e}}\\ \dot{\delta}_{\mathrm{e}} \end{Bmatrix} = \begin{Bmatrix} f_1\\ f_2\\ f_3\\ f_4\\ f_5\\ f_6\\ f_7\\ f_8 \end{Bmatrix} \tag{2.122}$$

where

$$M_{38} = -\frac{1}{4\mu}\frac{\bar{c}^2}{\kappa_y^2}\frac{V_G}{b}C_{\mathscr{M}_{\dot{\delta}_{\mathrm{e}}}} \tag{2.123}$$

$$M_{71} = \frac{e_{\mathrm{e}}}{\kappa_{\mathrm{e}}^2}\sin(\alpha - \mu_x) \tag{2.124}$$

$$M_{72} = \frac{e_{\mathrm{e}}}{\kappa_{\mathrm{e}}^2}V_G\cos(\alpha - \mu_x) - \frac{\rho V_G S_{\mathrm{e}}\bar{c}_{\mathrm{e}}^2}{4m_{\mathrm{e}}\kappa_{\mathrm{e}}^2}\left(1 - \frac{\mathrm{d}\varepsilon}{\mathrm{d}\alpha}\right)C_{\mathscr{H}_{\dot{\alpha}}} \tag{2.125}$$

$$M_{73} = -\left(\frac{e_{\mathrm{e}} x_{\mathrm{B},C_{\mathrm{e}}}}{\kappa_{\mathrm{e}}^2} - \cos\Lambda_{\mathrm{e}}\right) \tag{2.126}$$

$$M_{76} = -\frac{e_{\mathrm{e}}}{\kappa_{\mathrm{e}}^2}V_G\cos(\alpha - \mu_x) \tag{2.127}$$

$$M_{78} = -\frac{\rho V_G S_{\mathrm{e}}\bar{c}_{\mathrm{e}}^2}{4m_{\mathrm{e}}\kappa_{\mathrm{e}}^2}C_{\mathscr{H}_{\dot{\delta}_{\mathrm{e}}}} \tag{2.128}$$

and each f_i can be trivially obtained from equations (2.72) to (2.77) and from equations (2.120) and (2.121).

2.6.2 Exercises

Stick free evolution after an impulsive climbing perturbation

A simulation of the aircraft motion after an impulsive climbing perturbation will be carried out in this exercise. It will be used the same data and initial conditions as in the equivalent exercise in § 2.5.1.

To do so, it is necessary a routine, analogous to the one in listing 2.17, but for stick free simulations. It exactly replicates the system of differential equations (2.122) previously discussed to integrate it and finally solve for the unknown state variables. The routine is appended here.

Listing 2.21 Routine for carrying out 3-DoF stick free simulations.

```
function [time, delta_T, delta_s_deg, delta_tab_deg, ...
   V, alpha, q, x_EG, z_EG, theta, delta_e_dot, delta_e, ...
   CH_e, HingeMom] = ...
       ThreeDoFVabLinStickfree(t_end, state0, myAircraft, ...
                               delta_T_law, ...
                               delta_s_deg_law, delta_tab_deg_law)

delta_s_law = @(t) convang(delta_s_deg_law(t), 'deg', 'rad');
delta_tab_law = @(t) convang(delta_tab_deg_law(t), 'deg', 'rad');

g = 9.81;
W = myAircraft.mass * g;
S = myAircraft.S;
c = myAircraft.mac;
b = myAircraft.b;
mu_x = myAircraft.mu_x;
mu_T = myAircraft.mu_T;
k = myAircraft.K;
m = myAircraft.m;
k_y = myAircraft.k_y;
CL_alpha = myAircraft.CL_alpha;
CL_delta_e = myAircraft.CL_delta_e;
CL_delta_s = myAircraft.CL_delta_s;
CL_alpha_dot = myAircraft.CL_alpha_dot;
CL_q = myAircraft.CL_q;
CD0 = myAircraft.CD_0;
Cm0 = myAircraft.Cm_0;
Cm_alpha = myAircraft.Cm_alpha;
Cm_delta_e = myAircraft.Cm_delta_e;
Cm_delta_s = myAircraft.Cm_delta_s;
Cm_alpha_dot = myAircraft.Cm_alpha_dot;
Cm_delta_e_dot = myAircraft.Cm_delta_e_dot;
Cm_q = myAircraft.Cm_q;
Cm_T_0 = myAircraft.Cm_T_0;
Cm_T_alpha = myAircraft.Cm_T_alpha;
S_e = myAircraft.S_e;
Lambda_e = myAircraft.Lambda_e;
e_e = myAircraft.ec_adim;
x_C_e = myAircraft.x_C_e;
mac_e = myAircraft.mac_e;
mass_e = myAircraft.mass_e;
k_e = myAircraft.k_e;
CH_e_0 = myAircraft.Ch_e_0;
CH_e_alpha = myAircraft.Ch_e_alpha;
CH_e_delta_e = myAircraft.Ch_e_delta_e;
CH_e_delta_s = myAircraft.Ch_e_delta_s;
CH_e_delta_tab = myAircraft.Ch_e_delta_tab;
CH_e_delta_e_dot = myAircraft.Ch_e_delta_e_dot;
CH_e_alpha_dot = myAircraft.Ch_e_alpha_dot;
CH_e_q = myAircraft.Ch_e_q;
eps0 = myAircraft.eps_0;
Deps_Dalpha = myAircraft.DepsDalpha;

airspeed = @(state) state(1);
```

```
AoA = @(state) state(2);
AoA_deg = @(state) convang(state(2), 'rad', 'deg');
pitchrate = @(state) state(3);
pitchrate_degps = @(state) convang(state(3), 'rad/s', 'deg/s');
altitude = @(state) -state(5);
elevation = @(state) state(6);
elevation_deg = @(state) convang(state(6), 'rad', 'deg');
elevator = @(state) state(8);
elevator_dot = @(state) state(7);

Cm_T = @(t, state) delta_T_law(t) * (Cm_T_0 + Cm_T_alpha * AoA(state));

function a = sound(h)
[~, a, ~, ~] = atmosisa(h);
end
a = @(state) sound(altitude(state));
mach = @(state) airspeed(state) / a(state);

function rho = density(h)
[~, ~, ~, rho] = atmosisa(h);
end
rho = @(state) density(altitude(state));
mu = @(state) W / S / (rho(state) * b * g);

M32 = @(state) ...
-c/b / (4 * mu(state)) * airspeed(state)*c/k_y^2 * Cm_alpha_dot;
M38 = @(state) ...
-c/b / (4 * mu(state)) * airspeed(state)*c/k_y^2 * Cm_delta_e_dot;
M71 = @(state) ...
e_e / k_e^2 * sin(AoA(state) - mu_x);
M72 = @(state) ...
e_e / k_e^2 * airspeed(state) * cos(AoA(state) - mu_x) ...
-rho(state) * airspeed(state) * S_e * mac_e^2 / (4 * mass_e * k_e^2) ...
             * (1 - Deps_Dalpha) * CH_e_alpha_dot;
M73 = @(state) ...
-(e_e * x_C_e / k_e^2 - cos(Lambda_e));
M76 = @(state) ...
-e_e / k_e^2 * airspeed(state) * cos(AoA(state) - mu_x);
M78 = @(state) ...
-rho(state) * airspeed(state) * S_e * mac_e^2 / (4 * mass_e * k_e^2) ...
             * CH_e_delta_e_dot;
massmatrix = @(state) ...
[    1,          0,          0,      0, 0,     0,      0,     0; ...
0,          1,          0,      0, 0,     0,      0,     0; ...
0,      M32(state),     1,      0, 0,     0,      0, M38(state); ...
0,          0,          0,      1, 0,     0,      0,     0; ...
0,          0,          0,      0, 1,     0,      0,     0; ...
0,          0,          0,      0, 0,     1,      0,     0; ...
M71(state), M72(state), M73(state), 0, 0, M76(state), 1, M78(state); ...
0,          0,          0,      0, 0,     0,      0,     1     ];

Thrust = @(t, state) ...
delta_T_law(t) * ThrustModel(altitude(state), mach(state));

f1 = @(t, state) ...
-0.5 * S/W * rho(state) * airspeed(state)^2 * g * ( ...
 CD0 + k * (CL_alpha*AoA(state) + CL_delta_e*elevator(state) + ...
                                  CL_delta_s*delta_s_law(t))^m) ...
+Thrust(t, state)/W * g * cos(AoA(state) - mu_x - mu_T) ...
+g * sin(AoA(state) - mu_x - elevation(state));

f2 = @(t, state) ...
1 / (1 + c/b / (4 * mu(state)) * CL_alpha_dot) * ( ...
-0.5 * S/W * rho(state) * airspeed(state) * g * ( ...
CL_alpha*AoA(state) + ...
         CL_delta_e*elevator(state) + CL_delta_s*delta_s_law(t)) ...
+pitchrate(state) * (1 - c/b / (4 * mu(state)) * CL_q) ...
-Thrust(t, state)/W * g/airspeed(state) * ...
                                  sin(AoA(state) - mu_x - mu_T) ...
+g/airspeed(state) * cos(AoA(state) - mu_x - elevation(state)));

f3 = @(t, state) ...
```

```
airspeed(state)^2 * k_y^2 * c/b / (2 * mu(state)) * (...
Cm_T(t, state) + Cm0 + Cm_alpha * AoA(state) + ...
Cm_delta_e * elevator(state) + Cm_delta_s*delta_s_law(t) + ...
Cm_q * pitchrate(state) * c / (2 * airspeed(state)));

f4 = @(t, state) ...
airspeed(state) * cos(AoA(state) - mu_x - elevation(state));

f5 = @(t, state) ...
airspeed(state) * sin(AoA(state) - mu_x - elevation(state));

f6 = @(t, state) ...
pitchrate(state);

f7 = @(t, state) ...
e_e/k_e^2 * g * cos(elevation(state)) + ...
rho(state)*airspeed(state)^2*S_e*mac_e / (2 * mass_e * k_e^2) * ( ...
CH_e_0 + CH_e_alpha * ((1 - Deps_Dalpha) * (AoA(state) - mu_x) ...
-eps0 + delta_s_law(t) + mu_x) + Cm_delta_e * elevator(state) + ...
CH_e_delta_s * delta_s_law(t) + CH_e_delta_tab * delta_tab_law(t) + ...
CH_e_q * pitchrate(state) * mac_e / (2 * airspeed(state)));

f8 = @(t, state) ...
state(7);

rhs = @(t, state) ...
[f1(t, state); ...
f2(t, state); ...
f3(t, state); ...
f4(t, state); ...
f5(t, state); ...
f6(t, state); ...
f7(t, state); ...
f8(t, state)];

dstate_dt = @(t, state) ...
massmatrix(state) \ rhs(t, state);

ODEoptions = odeset('AbsTol', 1e-9, 'RelTol', 1e-9);

[time, state] = ode45(dstate_dt, [0, t_end], state0, ODEoptions);

delta_T = delta_T_law(time);              delta_T = delta_T(:);
delta_s_deg = delta_s_deg_law(time);      delta_s_deg = delta_s_deg(:);
delta_tab_deg = delta_tab_deg_law(time); delta_tab_deg =
    ↪ delta_tab_deg(:);
V = state(:, 1);
alpha = state(:, 2);
q = state(:, 3);
x_EG = state(:, 4);
z_EG = state(:, 5);
theta = state(:, 6);
delta_e_dot = state(:, 7);
delta_e = state(:, 8);

delta_s = convang(delta_s_deg, 'deg', 'rad');
delta_tab = convang(delta_tab_deg, 'deg', 'rad');
alpha_H = ( 1 - Deps_Dalpha) * (alpha - mu_x) - eps0 + delta_s + mu_x;
[~, ~, ~, dens] = atmosisa(-z_EG);

CH_e = CH_e_0 + CH_e_alpha * alpha_H + CH_e_delta_e * delta_e + ...
CH_e_delta_s * delta_s + CH_e_delta_tab * delta_tab + ...
mac_e./(2*V) .* ( ...
CH_e_alpha_dot * (1 - Deps_Dalpha) * timeDerivative(time, alpha) + ...
CH_e_q * q + CH_e_delta_e_dot * delta_e_dot);

HingeMom = CH_e .* 0.5 .* dens .* V.^2 .* S_e .* mac_e;

end
```

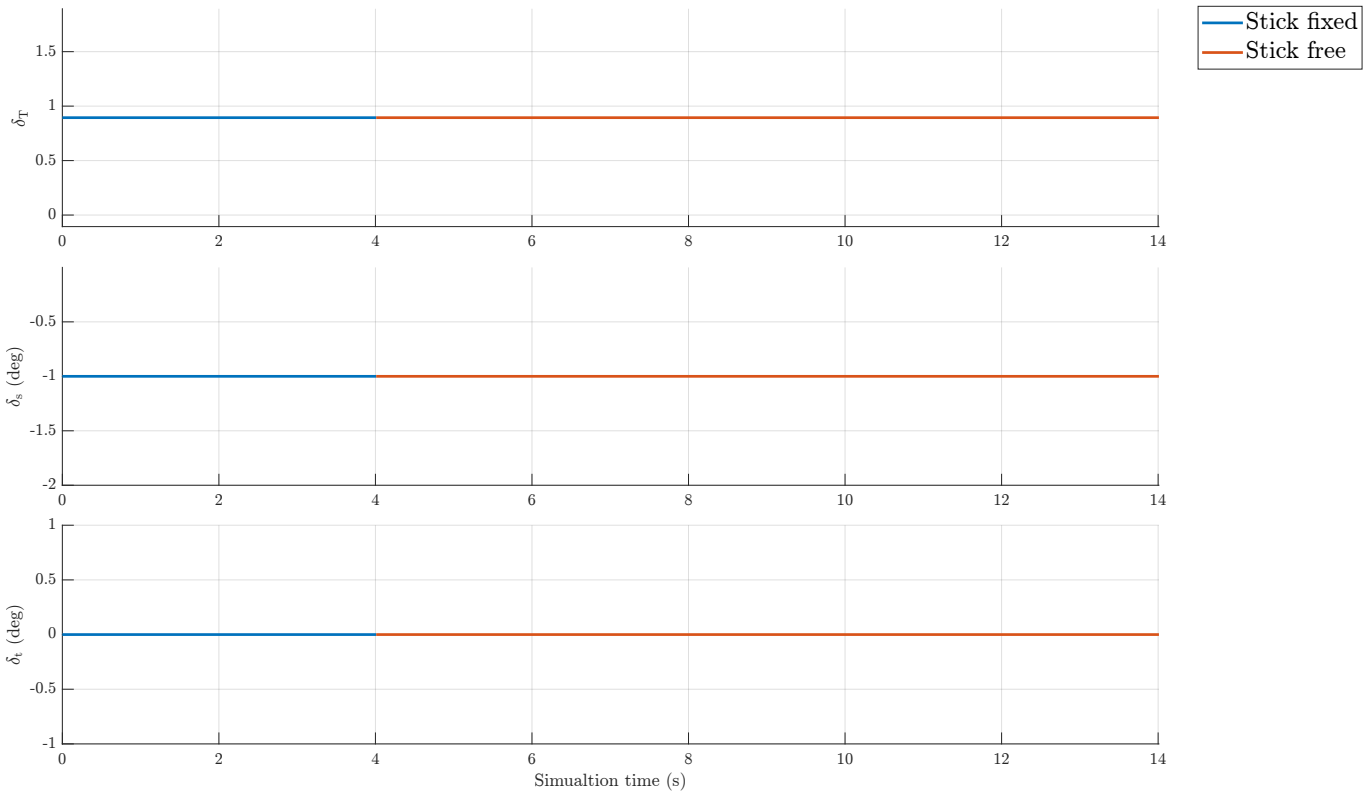


Figure 2.44 Input commands time histories.

Now, the simulation can be carried out. The chosen perturbation parameters are

$$\Delta\delta_e = 15 \text{ deg}, \qquad \Delta t/2 = 1.5 \text{ s}$$

After the stick fixed perturbation the aircraft motion will evolve with a stick free motion. It is worth noticing the hinge moment coefficient and hinge moment time histories in figure 2.49 and the stick free evolution of the elevator position in figure 2.45.

Observe the very much higher difficulty the aircraft has to return in an equilibrium flight condition, as visible in figures 2.46, 2.47 and 2.48.

Listing 2.22 Stick free evolution after a stick fixed impulsive perturbation.

```
close all; clear; clc

aircraftDataFileName = 'DSV_Aircraft_data.txt';
myAircraft = DSVAircraft(aircraftDataFileName);

V0 = 240;
h0 = 9000;
gamma0_deg = 0;

delta_s_deg = -1;

design0 = [0; 0.5; 0; 0];
lowerBounds = [convang(-15, 'deg', 'rad'); ...
               0.0; ...
               convang(-24, 'deg', 'rad'); ...
               convang(-3, 'deg', 'rad')];
upperBounds = [convang(18, 'deg', 'rad'); ...
               1.0; ...
               convang(16, 'deg', 'rad'); ...
               convang(3, 'deg', 'rad')];

[design, cost] = ThreeDoFTrim(myAircraft, ...
```

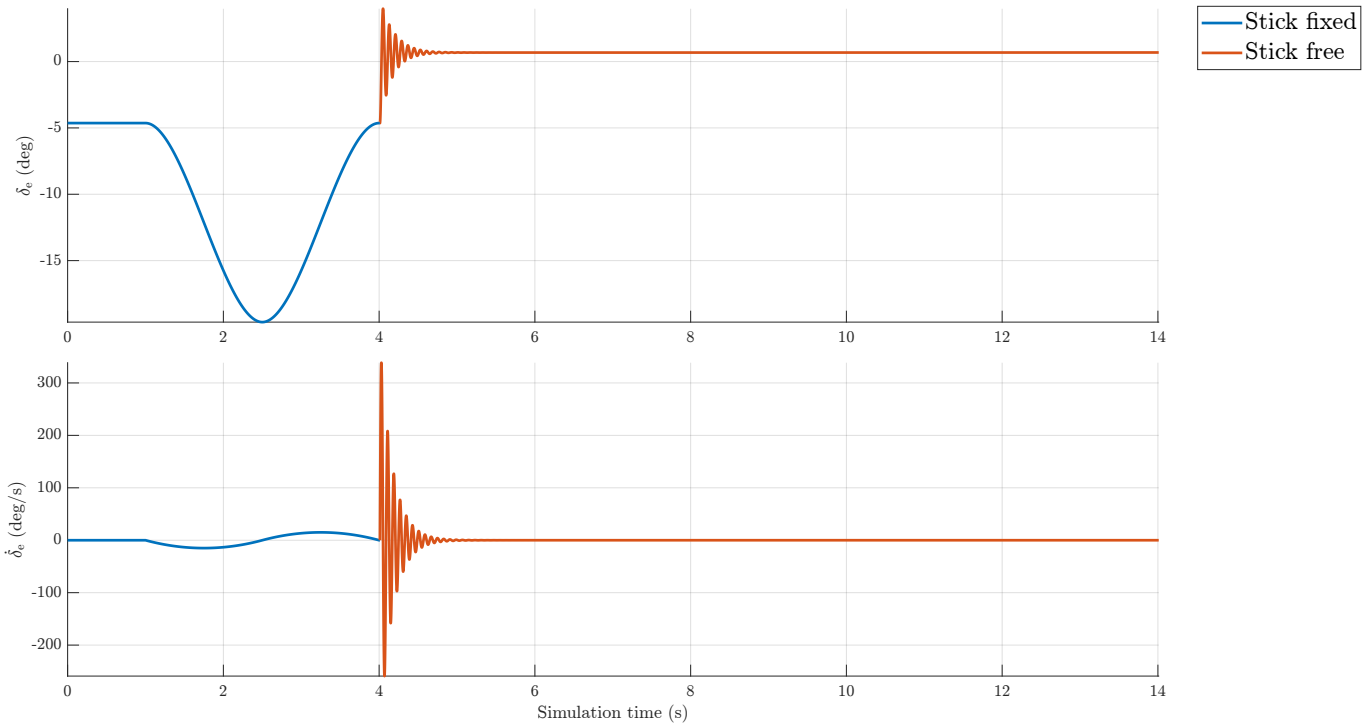


Figure 2.45 Time histories of the elevator deflection angle and its derivative.

```
                                V0, h0, ...
                                delta_s_deg, gamma0_deg, ...
                                design0, lowerBounds, upperBounds);

alpha0 = design(1);
alpha0_deg = convang(alpha0, 'rad', 'deg');

delta_T0 = design(2);

delta_e0 = design(3);
delta_e0_deg = convang(delta_e0, 'rad', 'deg');

delta_s0 = design(4);
delta_s0_deg = convang(delta_s0, 'rad', 'deg');

state0 = [V0; alpha0; 0; 0; -h0; alpha0];

half_duration = 1.5;
delta_e_deg_impulse = 15;
t_end = 1 + 2 * half_duration + 0.01;
delta_e_deg_law = @(t) interp1(...
    [0, 1, 1+half_duration, 1 + 2 * half_duration, t_end], ...
    [delta_e0_deg, delta_e0_deg, delta_e0_deg-delta_e_deg_impulse ...
    delta_e0_deg, delta_e0_deg], ...
    t, 'pchip');

delta_T_law = @(t) interp1([0, 1] * t_end, ...
                           [1, 1] * delta_T0, ...
                           t, 'linear');

delta_s_deg_law = @(t) interp1([0, 1] * t_end, ...
                               [1, 1] * delta_s0_deg, ...
                               t, 'linear');

[time1, ...
delta_T1, delta_e_deg1, delta_s_deg1, ...
V1, alpha1, q1, x_EG1, z_EG1, elevation1] = ...
            ThreeDoFVabLin(t_end, state0, myAircraft, ...
                           delta_T_law, delta_e_deg_law,
    ↪ delta_s_deg_law);
```

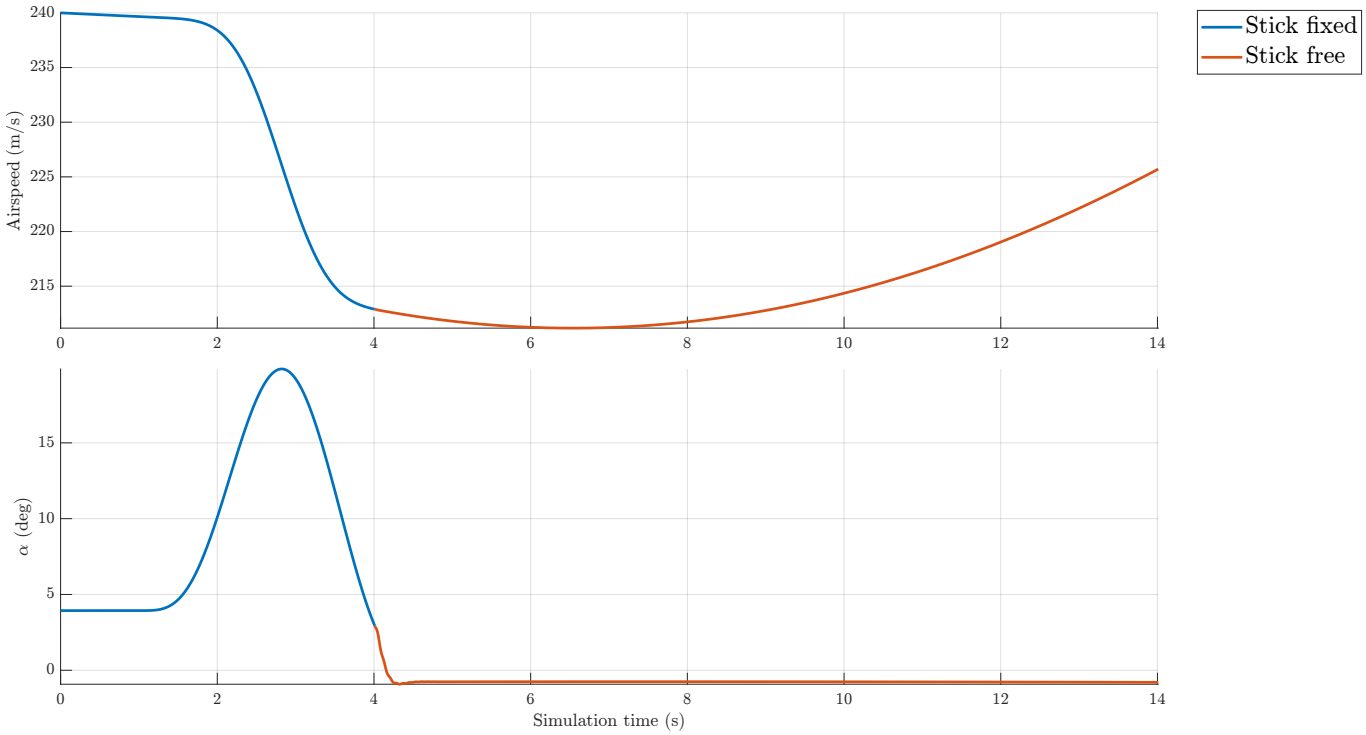


Figure 2.46 Airspeed and angle of attack time histories.

```
62
63 delta_e1 = convang(delta_e_deg1, 'deg', 'rad');
64 state0 = [V1(end); ...
65             alpha1(end); ...
66             q1(end); ...
67             x_EG1(end); ...
68             z_EG1(end); ...
69             elevation1(end); ...
70             0; ...
71             delta_e1(end)];
72
73 t_end = 10;
74
75 delta_T_law = @(t) interp1([0, 1] * t_end, ...
76                            [1, 1] * delta_T0, ...
77                            t, 'linear');
78
79 delta_s_deg_law = @(t) interp1([0, 1] * t_end, ...
80                                [1, 1] * delta_s0_deg, ...
81                                t, 'linear');
82
83 delta_tab_deg_law = @(t) interp1([0, 1] * t_end, ...
84                                [1, 1] * 0, ...
85                                t, 'linear');
86
87 [time2, delta_T2, delta_s_deg2, delta_tab_deg2, ...
88 V2, alpha2, q2, x_EG2, z_EG2, elevation2, delta_e_dot2, delta_e2, ...
89 CH_e2, HingeMom2] = ...
90     ThreeDoFVabLinStickfree(t_end, state0, myAircraft, delta_T_law, ...
91                            delta_s_deg_law, delta_tab_deg_law);
92
93 delta_e_dot1 = timeDerivative(time1, delta_e1);
94 delta_tab_deg1 = zeros(length(time1), 1);
95 nan1 = NaN(length(time1), 1);
96 nan2 = NaN(length(time2), 1);
97
98 mu_x = myAircraft.mu_x;
99 S_e = myAircraft.S_e;
100 Lambda_e = myAircraft.Lambda_e;
```

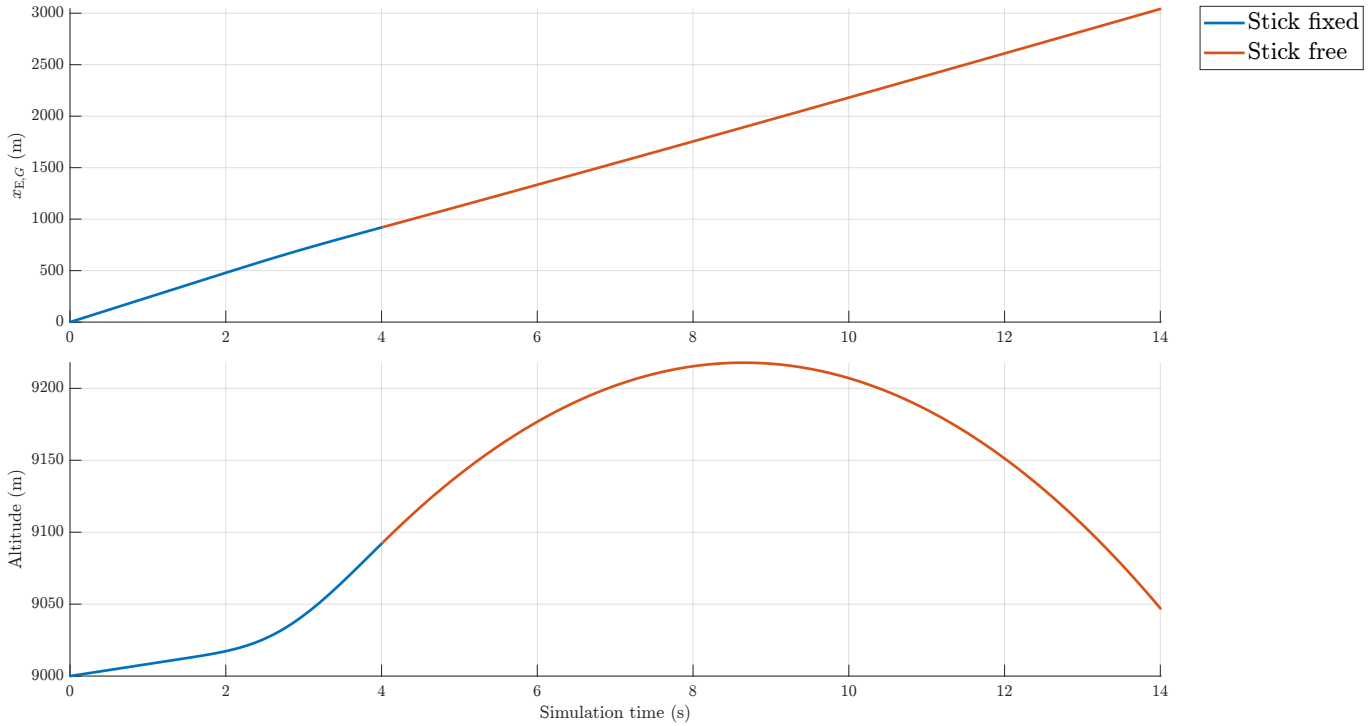


Figure 2.47 Aircraft center of gravity coordinates over time.

```
e_e = myAircraft.ec_adim;
x_C_e = myAircraft.x_C_e;
mac_e = myAircraft.mac_e;
mass_e = myAircraft.mass_e;
k_e = myAircraft.k_e;
CH_e_0 = myAircraft.Ch_e_0;
CH_e_alpha = myAircraft.Ch_e_alpha;
CH_e_delta_e = myAircraft.Ch_e_delta_e;
CH_e_delta_s = myAircraft.Ch_e_delta_s;
CH_e_delta_tab = myAircraft.Ch_e_delta_tab;
CH_e_delta_e_dot = myAircraft.Ch_e_delta_e_dot;
CH_e_alpha_dot = myAircraft.Ch_e_alpha_dot;
CH_e_q = myAircraft.Ch_e_q;
eps0 = myAircraft.eps_0;
Deps_Dalpha = myAircraft.DepsDalpha;

delta_s1 = convang(delta_s_deg1, 'deg', 'rad');
delta_tab1 = convang(delta_tab_deg1, 'deg', 'rad');
alpha_H1 = (1 - Deps_Dalpha) * (alpha1 - mu_x) - eps0 + delta_s1 + mu_x;
[~, ~, ~, dens1] = atmosisa(-z_EG1);

CH_e1 = CH_e_0 + CH_e_alpha * alpha_H1 + CH_e_delta_e * delta_e1 + ...
    CH_e_delta_s * delta_s1 + CH_e_delta_tab * delta_tab1 + ...
    mac_e./(2*V1) .* ( ...
    CH_e_alpha_dot * (1 - Deps_Dalpha) * timeDerivative(time1, alpha1) + ...
    CH_e_q * q1 + CH_e_delta_e_dot * delta_e_dot1);

HingeMom1 = CH_e1 .* 0.5 .* dens1 .* V1.^2 .* S_e .* mac_e;

x_EG_traj = [x_EG1; x_EG2];
z_EG_traj = [z_EG1; z_EG2];
elevation_traj = [elevation1; elevation2];

delta_T1 = [delta_T1; nan2];
delta_s_deg1 = [delta_s_deg1; nan2];
delta_tab_deg1 = [delta_tab_deg1; nan2];
V1 = [V1; nan2];
alpha1 = [alpha1; nan2];
x_EG1 = [x_EG1; nan2];
```

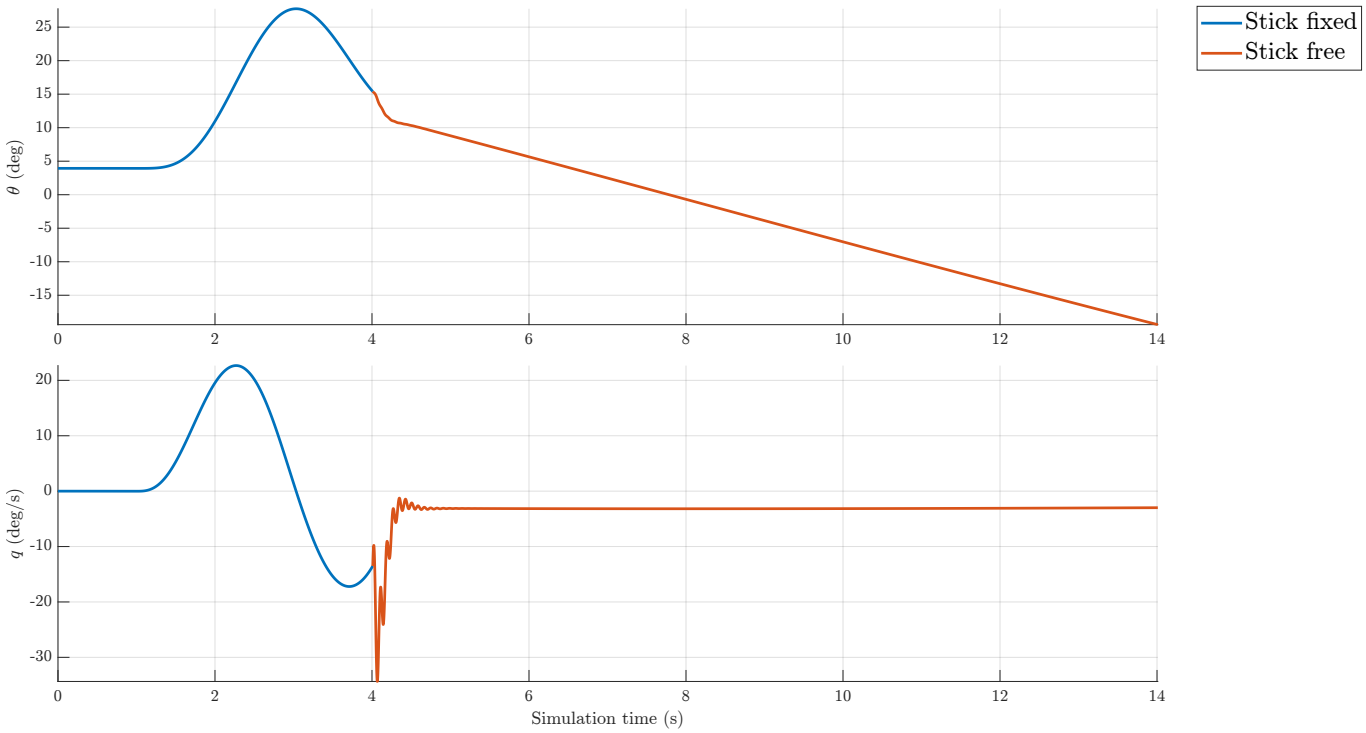


Figure 2.48 Time histories of the aircraft angular position and its pitch rate.

```
z_EG1 = [z_EG1; nan2];
elevation1 = [elevation1; nan2];
q1 = [q1; nan2];
delta_e1 = [delta_e1; nan2];
delta_e_dot1 = [delta_e_dot1; nan2];
CH_e1 = [CH_e1; nan2];
HingeMom1 = [HingeMom1; nan2];

delta_T2 = [nan1; delta_T2];
delta_s_deg2 = [nan1; delta_s_deg2];
delta_tab_deg2 = [nan1; delta_tab_deg2];
V2 = [nan1; V2];
alpha2 = [nan1; alpha2];
x_EG2 = [nan1; x_EG2];
z_EG2 = [nan1; z_EG2];
elevation2 = [nan1; elevation2];
q2 = [nan1; q2];
delta_e2 = [nan1; delta_e2];
delta_e_dot2 = [nan1; delta_e_dot2];
CH_e2 = [nan1; CH_e2];
HingeMom2 = [nan1; HingeMom2];

time = [time1; time2+time1(end)];
delta_T = [delta_T1, delta_T2];
delta_s_deg = [delta_s_deg1, delta_s_deg2];
delta_tab_deg = [delta_tab_deg1, delta_tab_deg2];
V = [V1, V2];
alpha = [alpha1, alpha2];
x_EG = [x_EG1, x_EG2];
z_EG = [z_EG1, z_EG2];
elevation = [elevation1, elevation2];
q = [q1, q2];
delta_e = [delta_e1, delta_e2];
delta_e_dot = [delta_e_dot1, delta_e_dot2];
CH_e = [CH_e1, CH_e2];
HingeMom = [HingeMom1, HingeMom2];

lgnd = {"Stick fixed", "Stick free"};

stackedPlot3(time, delta_T, delta_s_deg, delta_tab_deg, ...
```

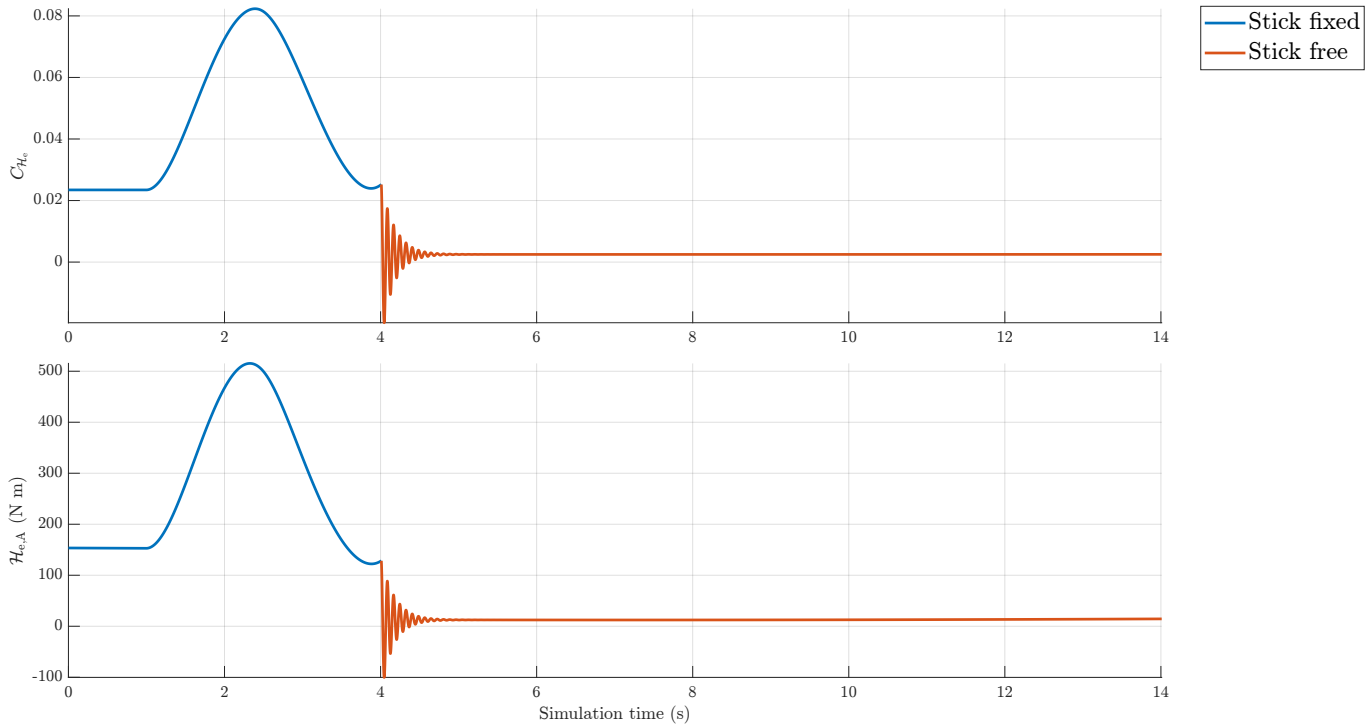


Figure 2.49 Time histories of the hinge moment coefficient and of the hinge moment during both the stick fixed part and the stick free part of the motion. Notice that since $e_e = 0$ for this aircraft data, during the stick fixed evolution the displayed hinge moment is equal to the moment the pilot must exercise in order to mantain the command. On the other hand, during the stick free evolution the displayed hinge moment is exactly the resultant moment acting on the elevator.

```
    {"Simualtion time (s)", "$\delta_\mathrm{T}$", ...
    "$\delta_\mathrm{s}$ (deg)", "$\delta_\mathrm{t}$ (deg)"}, ...
    {}, lgnd, 'ex3doflinstickfreeinputcommands.pdf')

alpha_deg = convang(alpha, 'rad', 'deg');
stackedPlot2(time, V, alpha_deg, ...
    {"Simulation time (s)", "Airspeed (m/s)", "$\alpha$ (deg)"}, ...
    {}, lgnd, 'ex3doflinstickfreeVa.pdf')

stackedPlot2(time, x_EG, -z_EG, ...
    {"Simulation time (s)", "$x_{\mathrm{E},G}$ (m)", "Altitude (m)"},
    ↪ ...
    {}, lgnd, 'ex3doflinstickfreecogxh.pdf')

elevation_deg = convang(elevation, 'rad', 'deg');
q_degps = convangvel(q, 'rad/s', 'deg/s');
stackedPlot2(time, elevation_deg, q_degps, ...
    {"Simulation time (s)", "$\theta$ (deg)", "$q$ (deg/s)"}, ...
    {}, lgnd, 'ex3doflinstickfreethetaq.pdf')

delta_e_deg = convang(delta_e, 'rad', 'deg');
delta_e_dot_degps = convangvel(delta_e_dot, 'rad/s', 'deg/s');
stackedPlot2(time, delta_e_deg, delta_e_dot_degps, ...
    {"Simulation time (s)", "$\delta_\mathrm{e}$ (deg)", ...
    "$\dot\delta_\mathrm{e}$ (deg/s)"}, ...
    {}, lgnd, 'ex3doflinstickfreedeltae.pdf')

stackedPlot2(time, CH_e, HingeMom, ...
    {"Simulation time (s)", "$C_{\mathcal{H}_\mathrm{e}}$", ...
    "$\mathcal{H}_\mathrm{e,A}$ (N m)"}, ...
    {}, lgnd, 'ex3doflinstickfreeCH.pdf')

Nt = length(time);
y_EG = zeros(Nt, 1);
```

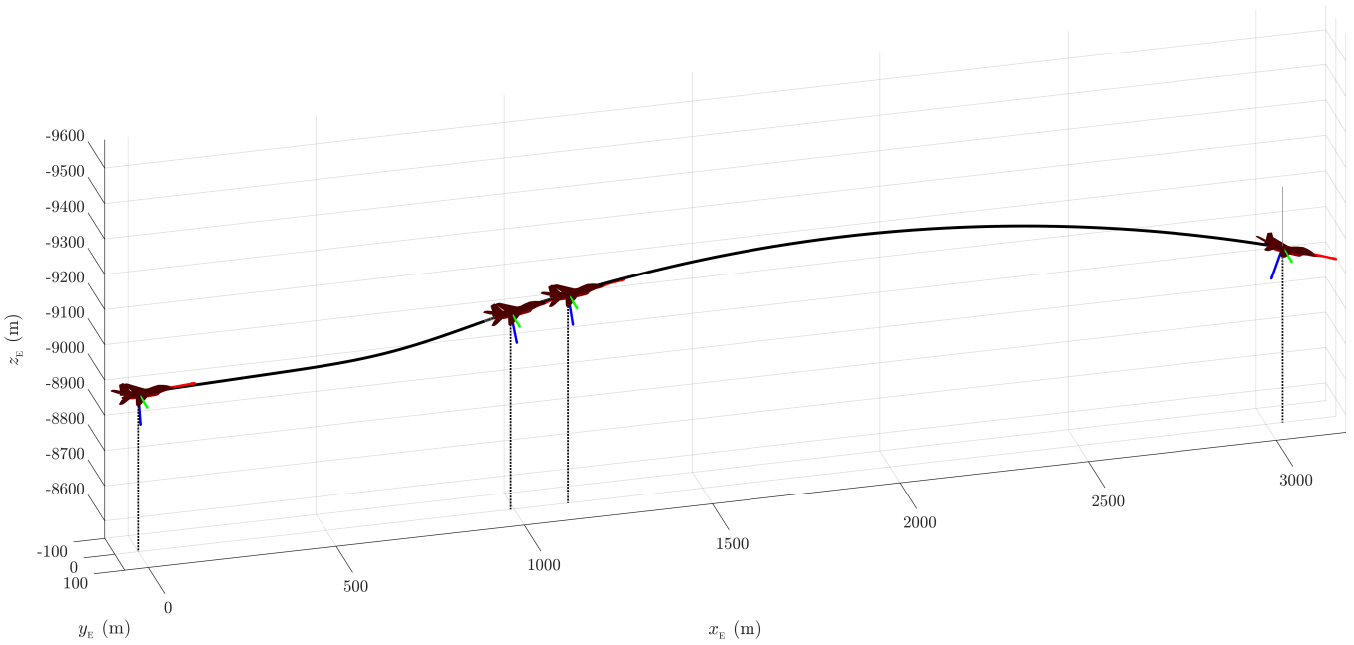


Figure 2.50 Visualization of the trajectory obtained from the simulation.

```
213 phi = y_EG;
214 psi = y_EG;
215 trajectoryView(time, x_EG_traj, y_EG, z_EG_traj, ...
216                 phi, elevation_traj, psi, ...
217                 120, 4, ...
218                 'ex3doflinstickfreetrajectory.pdf')
```

Chapter 3

Stability

The study of stability consists of finding out if a given equilibrium condition is a stable one or not. In this chapter we will discuss the stability of aircrafts and then solve for the response of the aircraft motion to given perturbations from trimmed conditions.

3.1 Dynamic model

In pursuit of the aircraft stability characterization we will use the following model for its dynamics from now on.

$$\begin{cases} m\left(\dot{u} + qw - rv\right) = X \\ m\left(\dot{v} + ru - pw\right) = Y \\ m\left(\dot{w} + pv - qu\right) = Z \end{cases} \tag{3.1}$$

$$\begin{cases} I_{xx}\dot{p} - I_{xz}\left(\dot{r} + pq\right) - \left(I_{yy} - I_{zz}\right)qr = \mathscr{L} \\ I_{yy}\dot{q} - I_{xz}\left(r^2 - p^2\right) - \left(I_{zz} - I_{xx}\right)rp = \mathscr{M} \\ I_{zz}\dot{r} - I_{xz}\left(\dot{p} - qr\right) - \left(I_{xx} - I_{yy}\right)pq = \mathscr{N} \end{cases} \tag{3.2}$$

$$\begin{cases} \dot{\varphi} = p + \left(q\sin\varphi + r\cos\varphi\right)\tan\theta \\ \dot{\theta} = \quad q\cos\varphi - r\sin\varphi \\ \dot{\psi} = \quad \left(q\sin\varphi + r\cos\varphi\right)\sec\theta \end{cases} \tag{3.3}$$

$$\begin{Bmatrix} \dot{x}_{\mathrm{E},G} \\ \dot{y}_{\mathrm{E},G} \\ \dot{z}_{\mathrm{E},G} \end{Bmatrix} = [T]_{\mathrm{EB}} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{3.4}$$

All these twelve equations can be rearranged into a system of differential equations to constitute the usual expression for the dynamical system $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u})$, where the state vector and the input vector are the following.

$$\boldsymbol{x} = \left[u,\ v,\ w,\ p,\ q,\ r,\ x_{\mathrm{E},G},\ y_{\mathrm{E},G},\ z_{\mathrm{E},G},\ \varphi,\ \theta,\ \psi\right]^{\mathrm{T}} \tag{3.5}$$

$$\boldsymbol{u} = \left[\delta_{\mathrm{T}},\ \delta_{\mathrm{e}},\ \delta_{\mathrm{s}},\ \delta_{\mathrm{a}},\ \delta_{\mathrm{r}}\right]^{\mathrm{T}} \tag{3.6}$$

3.2 Small perturbations notation

In this chapter we will make a distinction between trimmed values and small perturbation values of the variables. For this sake we will use a different notation from now on. The actual value of a variable can be expressed as the sum of its constant trimmed value and its small perturbation. Therefore, we can write

$$\boldsymbol{x}(t) = \boldsymbol{x}_{\mathrm{N}}(t) + \Delta\boldsymbol{x}(t), \qquad \boldsymbol{u}(t) = \boldsymbol{u}_0 + \Delta\boldsymbol{u}(t) \tag{3.7}$$

where $\boldsymbol{x}_{\mathrm{N}}(t)$ is the state vector in nominal flight condition, $\boldsymbol{u}_0$ is the constant trimmed input vector, and $\Delta\boldsymbol{x}(t)$ and $\Delta\boldsymbol{u}(t)$ are small perturbations of the state and the input respectively from their trimmed reference values.

From now on uppercase letters will be used for the actual value of each variable, and lowercase letters for the relative perturbation value of each variable. Trimmed variables will be denoted with the same uppercase letter as the actual value of the variable, but with a subscripted 0 for each variable. In particular,

$$\boldsymbol{x}_{\mathrm{N}} = \left[U_0,\ V_0,\ W_0,\ P_0,\ Q_0,\ R_0,\ X_{\mathrm{E},G_0},\ Y_{\mathrm{E},G_0},\ Z_{\mathrm{E},G_0},\ \Phi_0,\ \Theta_0,\ \Psi_0\right]^{\mathrm{T}} \tag{3.8}$$

$$\Delta\boldsymbol{x} = \left[u,\ v,\ w,\ p,\ q,\ r,\ x_{\mathrm{E},G},\ y_{\mathrm{E},G},\ z_{\mathrm{E},G},\ \varphi,\ \theta,\ \psi\right]^{\mathrm{T}} \tag{3.9}$$

$$\boldsymbol{u}_0 = \left[\delta_{\mathrm{T}0},\ \delta_{\mathrm{e}0},\ \delta_{\mathrm{s}0},\ \delta_{\mathrm{a}0},\ \delta_{\mathrm{r}0}\right]^{\mathrm{T}} \tag{3.10}$$

$$\Delta\boldsymbol{u} = \left[\delta_{\mathrm{T}},\ \delta_{\mathrm{e}},\ \delta_{\mathrm{s}},\ \delta_{\mathrm{a}},\ \delta_{\mathrm{r}}\right]^{\mathrm{T}} \tag{3.11}$$

and the same notation will be applied to other variables. Few only exceptions are the Mach number M, the angle of attack α and the sideslip angle β. For these variables we will adopt the following notation

$$M = M_0 + \Delta M, \qquad \alpha = \alpha_0 + \Delta\alpha, \qquad \beta = \beta_0 + \Delta\beta \tag{3.12}$$

Furthermore, we will choose a *stability reference frame*, defined upon the trim condition of the aircraft. This allows us to assume $V_0 = 0$, $P_0 = Q_0 = R_0 = 0$, $Y_{\mathrm{E},G_0} = 0$, $\Phi_0 = \Psi_0 = 0$ and $\alpha_0 = \beta_0 = 0$. Therefore, it is possible to write

$$V = v \tag{3.13}$$

$$P = p, \qquad Q = q, \qquad R = r \tag{3.14}$$

$$Y_{\mathrm{E},G} = y_{\mathrm{E},G} \tag{3.15}$$

$$\Phi = \varphi, \qquad \Psi = \psi \tag{3.16}$$

$$\alpha = \Delta\alpha, \qquad \beta = \Delta\beta \tag{3.17}$$

when a symmetric longitudinal three degrees of freedom has being chosen as the reference trim condition.

3.3 Linearized aerodynamic model

Each aerodynamic function $f_{\mathrm{A}} = f_{\mathrm{A}0} + \Delta f_{\mathrm{A}}$, such as the aerodynamic forces coefficients and the aerodynamic moments coefficients, can be linearized by a *Taylor's expansion* at

$\frac{\partial \rightarrow}{\partial \downarrow}$	X_A	Y_A	Z_A	$\mathscr{L}_A$	$\mathscr{M}_A$	$\mathscr{N}_A$
u	■	0	■	0	■	0
v	0	●	0	●	0	●
w	■	0	■	0	■	0
$\dot{w}$	≈ 0	0	■	0	■	0
p	0	≈ 0	0	●	0	●
q	≈ 0	0	■	0	■	0
r	0	●	0	●	0	●
δ_a	0	≈ 0	0	●	0	●
δ_e	≈ 0	0	■	0	■	0
δ_r	0	●	0	●	0	●

Table 3.1 Dependencies of the aerodynamic forces and moments on the state and input variables. ■ denotes a dependency on a symmetric variable and ● denotes a dependency on a non-symmetric variable. Every non-stationary effect has been neglected here, the only exception is constituted by $\dot{w}$ that represents the non-negligible $\dot{\alpha}$ effect as discussed below in § 3.3.1.

first order as follows (see reference [1])

$$\begin{aligned} f_A = \quad & C_0 \\ & + C_1 u + C_2 \dot{u} + C_3 v + C_4 \dot{v} + C_5 w + C_6 \dot{w} \\ & + C_7 p + C_8 \dot{p} + C_9 q + C_{10} \dot{q} + C_{11} r + C_{12} \dot{r} \\ & + C_{13} z_{E,G} \\ & + C_{14} \delta_a + C_{15} \dot{\delta}_a + C_{16} \delta_e + C_{17} \dot{\delta}_e + C_{18} \delta_r + C_{19} \dot{\delta}_r \\ & + C_{20} \delta_T + C_{21} \dot{\delta}_T \end{aligned} \tag{3.18}$$

where each coefficient C_i is a first order derivative of f_A evaluated in trim condition. In particular they can be expressed as

$$C_0 = f_A\Big|_0 \tag{3.19}$$

$$C_1 = \frac{\partial f_A}{\partial u}\Big|_0,\ C_2 = \frac{\partial f_A}{\partial \dot{u}}\Big|_0,\ C_3 = \frac{\partial f_A}{\partial v}\Big|_0,\ C_4 = \frac{\partial f_A}{\partial \dot{v}}\Big|_0,\ C_5 = \frac{\partial f_A}{\partial w}\Big|_0,\ C_6 = \frac{\partial f_A}{\partial \dot{w}}\Big|_0 \tag{3.20}$$

$$C_7 = \frac{\partial f_A}{\partial p}\Big|_0,\ C_8 = \frac{\partial f_A}{\partial \dot{p}}\Big|_0,\ C_9 = \frac{\partial f_A}{\partial q}\Big|_0,\ C_{10} = \frac{\partial f_A}{\partial \dot{q}}\Big|_0,\ C_{11} = \frac{\partial f_A}{\partial r}\Big|_0,\ C_{12} = \frac{\partial f_A}{\partial \dot{r}}\Big|_0 \tag{3.21}$$

$$C_{13} = \frac{\partial f_A}{\partial z_{E,G}}\Big|_0 \tag{3.22}$$

$$C_{14} = \frac{\partial f_A}{\partial \delta_a}\Big|_0,\ C_{15} = \frac{\partial f_A}{\partial \dot{\delta}_a}\Big|_0,\ C_{16} = \frac{\partial f_A}{\partial \delta_e}\Big|_0,\ C_{17} = \frac{\partial f_A}{\partial \dot{\delta}_e}\Big|_0,\ C_{18} = \frac{\partial f_A}{\partial \delta_r}\Big|_0,\ C_{19} = \frac{\partial f_A}{\partial \dot{\delta}_r}\Big|_0 \tag{3.23}$$

$$C_{20} = \frac{\partial f_A}{\partial \delta_T}\Big|_0,\ C_{21} = \frac{\partial f_A}{\partial \dot{\delta}_T}\Big|_0 \tag{3.24}$$

where the notation $\big|_0$ means $\big|_{\text{trim}}$.

It is worth pointing out that it is conventionally possible to neglect some of those C_i coefficients. In particular, the aerodynamic loads X_A, Z_A and $\mathscr{M}_A$ are assumed to depend on symmetric longitudinal variables only, while the aerodynamic loads Y_A, $\mathscr{L}_A$ and $\mathscr{N}_A$

are assumed to depend on non-symmetric variables only. In addition, some of the cross dependencies between lateral and directional dynamics can be also neglected by first approximation. All of these relations are summed up in table 3.1.

3.3.1 Bryan's hypotheses

The Bryan's hypotheses state that it is possible to neglect derivatives of symmetric quantities like X_A, Z_A and $\mathscr{M}_A$ with respect to non-symmetric variables. In the same way, it is possible to neglect derivatives of non-symmetric quantities like Y_A, $\mathscr{L}_A$ and $\mathscr{N}_A$ with respect to symmetric variables. In addition, empirical experience has proved that it is possible to neglect some dependencies of some symmetric quantities with respect to some symmetric variables and the same for some non-symmetric quantities.

Furthermore, Bryan's hypotheses also state that all the non-stationary effects during the aircraft motion can be neglected. The only one that is typically not neglected is the one relative to the angle of attack rate of change $\dot{\alpha}$.

3.4 Separation between longitudinal and lateral-directional motion

As suggested by Bryan's hypotheses it is possible to separate longitudinal motion and lateral-directional motion from each other. In particular we can define

$$\boldsymbol{x}_{\text{LON}} = \begin{Bmatrix} u \\ w \\ q \\ x_{E,G} \\ z_{E,G} \\ \theta \end{Bmatrix}, \qquad \boldsymbol{u}_{\text{LON}} = \begin{Bmatrix} \delta_T \\ \delta_e \end{Bmatrix} \tag{3.25}$$

and

$$\boldsymbol{x}_{\text{LD}} = \begin{Bmatrix} v \\ p \\ r \\ y_{E,G} \\ \varphi \\ \psi \end{Bmatrix}, \qquad \boldsymbol{u}_{\text{LD}} = \begin{Bmatrix} \delta_a \\ \delta_r \end{Bmatrix} \tag{3.26}$$

The linearized state propagation equations can be expressed as

$$\begin{Bmatrix} \dot{\boldsymbol{x}}_{\text{LON}} \\ \dot{\boldsymbol{x}}_{\text{LD}} \end{Bmatrix} = F \begin{Bmatrix} \boldsymbol{x}_{\text{LON}} \\ \boldsymbol{x}_{\text{LD}} \end{Bmatrix} + G \begin{Bmatrix} \boldsymbol{u}_{\text{LON}} \\ \boldsymbol{u}_{\text{LD}} \end{Bmatrix} \tag{3.27}$$

where F and G are *Jacobian matrices* defined as follows

$$F = \frac{\partial \boldsymbol{f}}{\partial (\boldsymbol{x}_{\text{LON}}, \boldsymbol{x}_{\text{LD}})} = \begin{bmatrix} F_{\text{LON}} & O_{6\times6} \\ O_{6\times6} & F_{\text{LD}} \end{bmatrix}, \qquad G = \frac{\partial \boldsymbol{f}}{\partial (\boldsymbol{u}_{\text{LON}}, \boldsymbol{u}_{\text{LD}})} = \begin{bmatrix} G_{\text{LON}} & O_{6\times4} \\ O_{6\times4} & G_{\text{LD}} \end{bmatrix} \tag{3.28}$$

The separation of the longitudinal and lateral-directional motions is possible be-

cause of the hypothesis that the cross-terms into matrices F and G are all zero. Therefore, these two motions can be studied separately.

3.5 Longitudinal dynamics

The system of differential equations of the longitudinal dynamics can be expressed as the typical form of a dynamic linear time-invariant (LTI) system. In particular,

$$\dot{\boldsymbol{x}}_{\text{LON}} = A_{\text{LON}}\boldsymbol{x}_{\text{LON}} + B_{\text{LON}}\boldsymbol{u}_{\text{LON}} \tag{3.29}$$

with characteristic polynomial $\Delta_{\text{LON}}(s) = \det(s\,\mathbb{1}_6 - A_{\text{LON}})$ and the following characteristic equation

$$(s-\lambda_{\text{RANGE}})\,(s-\lambda_{\text{HEIGHT}})\,(s-\lambda_{\text{PH}})\left(s-\lambda^*_{\text{PH}}\right)(s-\lambda_{\text{SP}})\left(s-\lambda^*_{\text{SP}}\right) = 0 \tag{3.30}$$

Since the only two real eigenvalues λ_{RANGE} and λ_{HEIGHT} are approximately zero and their corresponding modes are negligible and non-relevant, it is convenient to neglect them and their associated state variables $x_{\text{E},G}$ and $z_{\text{E},G}$.

Therefore, we will consider the same state propagation equation (3.29) mentioned above, but with a restriction from 6 to 4 equations and hence variables. In particular,

$$\boldsymbol{x}_{\text{LON}} = \begin{Bmatrix} u \\ w \\ q \\ \theta \end{Bmatrix} \tag{3.31}$$

with characteristic equation

$$(s-\lambda_{\text{PH}})\left(s-\lambda^*_{\text{PH}}\right)(s-\lambda_{\text{SP}})\left(s-\lambda^*_{\text{SP}}\right) = 0 \tag{3.32}$$

The matrices of the dynamical system can be evaluated as

$$A_{\text{LON}} = \begin{bmatrix} \hat{X}_u & \hat{X}_w & 0 & -g\cos\Theta_0 \\ \hat{Z}_u & \hat{Z}_w & U_0 & -g\sin\Theta_0 \\ \hat{\mathcal{M}}_u + \hat{\mathcal{M}}_{\dot{w}}\hat{Z}_u & \hat{\mathcal{M}}_w + \hat{\mathcal{M}}_{\dot{w}}\hat{Z}_w & \hat{\mathcal{M}}_q + \hat{\mathcal{M}}_{\dot{w}}U_0 & -\hat{\mathcal{M}}_{\dot{w}}g\sin\Theta_0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.33}$$

$$B_{\text{LON}} = \begin{bmatrix} \hat{X}_{\delta_{\text{T}}} & \hat{X}_{\delta_{\text{e}}} \\ \hat{Z}_{\delta_{\text{T}}} & \hat{Z}_{\delta_{\text{e}}} \\ \hat{\mathcal{M}}_{\delta_{\text{T}}} + \hat{\mathcal{M}}_{\dot{w}}\hat{Z}_{\delta_{\text{T}}} & \hat{\mathcal{M}}_{\delta_{\text{e}}} + \hat{\mathcal{M}}_{\dot{w}}\hat{Z}_{\delta_{\text{e}}} \\ 0 & 0 \end{bmatrix} \tag{3.34}$$

where the dimensional derivatives inside the matrices can be computed as shown in table 3.2 at the next page.

Derivative	Formula	Unit
$\hat{X}_u$	$= -\frac{\frac{1}{2}\rho_0 U_0^2 S}{mU_0}\left(2C_D\Big\|_0 + M_0\frac{\partial C_D}{\partial M}\Big\|_0\right)$	s^{-1}
$\hat{X}_w$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S}{mU_0}\left(C_L\Big\|_0 - C_{D\alpha}\Big\|_0\right)$	s^{-1}
$\hat{X}_{\dot{w}}$	≈ 0	—
$\hat{X}_q$	≈ 0	ms^{-1}
$\hat{X}_{\delta_\mathrm{T}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S}{m}\left(C_{T_\mathrm{fix}} + k_V/U_0^2\right)$	ms^{-2}
$\hat{X}_{\delta_\mathrm{e}}$	≈ 0	ms^{-2}
$\hat{Z}_w$	$= -\frac{\frac{1}{2}\rho_0 U_0^2 S}{mU_0}\left(2C_L\Big\|_0 + \frac{M_0^2}{1-M_0^2}C_L\Big\|_0\right)$	s^{-1}
$\hat{Z}_w$	$= -\frac{\frac{1}{2}\rho_0 U_0^2 S}{mU_0}\left(C_D\Big\|_0 + C_{L\alpha}\Big\|_0\right)$	s^{-1}
$\hat{Z}_{\dot{w}}$	$= -\frac{\frac{1}{2}\rho_0 S\bar{c}}{2m}C_{L\dot{\alpha}}\Big\|_0 = -\frac{1}{2\mu_0}C_{L\dot{\alpha}} \approx 0$	—
$\hat{Z}_q$	$= -\frac{U_0}{2\mu_0}C_{Lq}\Big\|_0 \approx 0$	ms^{-1}
$\hat{Z}_{\delta_\mathrm{e}}$	$= -\frac{\frac{1}{2}\rho_0 S}{m}C_{L\delta_\mathrm{e}}\Big\|_0 = -\frac{\frac{1}{2}\rho_0 S}{m}\eta_\mathrm{H}\frac{S_\mathrm{H}}{S}C_{L\alpha,\mathrm{H}}\tau_\mathrm{e}$	ms^{-2}
$\hat{\mathscr{M}}_u$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S\bar{c}}{I_{yy}U_0}C_{\mathscr{M}u}\Big\|_0 = \frac{\frac{1}{2}\rho_0 U_0^2 S\bar{c}}{I_{yy}U_0}M_0 C_{\mathscr{M}M}\Big\|_0$	$\mathrm{m}^{-1}\mathrm{s}^{-1}$
$\hat{\mathscr{M}}_w$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S\bar{c}}{I_{yy}U_0}C_{\mathscr{M}\alpha}\Big\|_0$	$\mathrm{m}^{-1}\mathrm{s}^{-1}$
$\hat{\mathscr{M}}_{\dot{w}}$	$= \frac{\rho_0 S\bar{c}^2}{4I_{yy}}C_{\mathscr{M}\dot{\alpha}}\Big\|_0 = -2\frac{\rho_0 S\bar{c}^2}{4I_{yy}}K\eta_\mathrm{H}\bar{\mathscr{V}}_\mathrm{H}\frac{l_\mathrm{H}}{\bar{c}}\frac{\mathrm{d}\varepsilon}{\mathrm{d}\alpha}C_{L\alpha,\mathrm{H}}$	m^{-1}
$\hat{\mathscr{M}}_q$	$= \frac{\rho_0 U_0 S\bar{c}^2}{4I_{yy}}C_{\mathscr{M}q}\Big\|_0 = -2\frac{\rho_0 U_0 S\bar{c}^2}{4I_{yy}}K\eta_\mathrm{H}\bar{\mathscr{V}}_\mathrm{H}\frac{l_\mathrm{H}}{\bar{c}}C_{L\alpha,\mathrm{H}}$	s^{-1}
$\hat{\mathscr{M}}_{\delta_\mathrm{e}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S\bar{c}}{I_{yy}}C_{\mathscr{M}\delta_\mathrm{e}}\Big\|_0 = -\frac{\frac{1}{2}\rho_0 U_0^2 S\bar{c}}{I_{yy}}\eta_\mathrm{H}\bar{\mathscr{V}}_\mathrm{H}C_{L\alpha,\mathrm{H}}\tau_\mathrm{e}$	s^{-2}

Table 3.2 Dimensional derivatives of stability of the longitudinal motion. These formulae are valid for an aircraft whose propulsion systems are modeled under the constant thrust assumption.

3.5.1 Exercises

Short period and phugoid eigenvectors as initial conditions

Listing 3.1 Short period and phugoid fasors and response.

```
clear; close all; clc

global  g AC myAC...
        V_0 q_0 gamma_0 ...
        rho_0

load('dati_velivolo.mat');
Weight  = myAC.W;
g       = 9.81;
mass    = myAC.mass;

h_0     = 4000;
[T_0, a_0, p_0, rho_0] = atmosisa(h_0);
V_0     = 100;
Mach_0  = V_0/a_0;
Iyy     = myAC.Iyy;

S       = myAC.S;
mac     = myAC.mac;
qbar_0  = 0.5*rho_0*(V_0^2);
mu_0    = mass/(0.5*rho_0*S*mac);
gamma_0 = 0.0;
q_0     = 0;

myAC.T_max = ...
    AC.AircraftDataAdvanced.Propulsion.CRUISE.f_Thrust(h_0, Mach_0);

xi0 = [0; ...               % alpha_0
       0; ...                % delta_e_0
       0.5];                 % delta_T_0

lb =[convang(-8,'deg','rad'), ...
    convang(-30,'deg','rad'), ...
    0.2 ...
    ];
ub =[convang( 20,'deg','rad'), ...
    convang( 10,'deg','rad'), ...
    1.0 ...
    ];

options = optimset('tolfun', 1e-9, 'Algorithm', 'interior-point');

[xi, cost] = ...
    fmincon(@costLongEquilibriumStaticStickFixedVel, xi0, ...
            [], [], [], [], ...
            lb, ub, @myNonLinearConstraint, options);

alpha_0 = xi(1);
delta_e_0 = xi(2);
delta_T_0 = xi(3);
theta_0 = gamma_0 + alpha_0 - myAC.mu_x;

alpha_0_deg = convang(alpha_0, 'rad', 'deg');
delta_e_0_deg = convang(delta_e_0, 'rad', 'deg');

C_L             = ...
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL(alpha_0, ...
    delta_e_0, 0, 0);
C_D             = ...
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CD.f_CD(...
    alpha_0, delta_e_0);
C_L_Mach        = 0;
Delta_alpha     = deg2rad(1);
Delta_delta_e   =  deg2rad(1);
C_L_alpha       = ...
```

```
    (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL( ...
    alpha_0+Delta_alpha, delta_e_0, 0, 0) - ...
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL( ...
    alpha_0-Delta_alpha, delta_e_0, 0, 0)) / (2*Delta_alpha);
C_L_alphadot    = myAC.CL_alphadot;
C_L_q           = myAC.CL_q;
C_D_alpha       = ...
    (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CD.f_CD( ...
    alpha_0+Delta_alpha, delta_e_0) - ...
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CD.f_CD( ...
    alpha_0-Delta_alpha, delta_e_0)) / (2*Delta_alpha);
C_D_alphadot    = 0.0;
C_D_q           = 0.0;
C_D_Mach        = 0.0;

C_m_alpha    = ...
    (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
    alpha_0+Delta_alpha, delta_e_0, 0, 0) - ...
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
    alpha_0-Delta_alpha, delta_e_0, 0, 0)) / (2*Delta_alpha);
C_m_alphadot = myAC.Cpitch_alphadot;
C_m_q        = myAC.Cpitch_q;
C_m_Mach     = 0.0;
C_L_de       = ...
    (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL( ...
    alpha_0, delta_e_0+Delta_delta_e, 0, 0) - ...
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL( ...
    alpha_0, delta_e_0-Delta_delta_e, 0, 0)) / (2*Delta_delta_e);
C_m_de       = ...
    (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
    alpha_0, delta_e_0 +Delta_delta_e, 0, 0) - ...
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
    alpha_0, delta_e_0-Delta_delta_e, 0, 0)) / (2*Delta_alpha);

X_u    = -(qbar_0*S/(mass*V_0))...
           *(2*C_D + Mach_0*C_D_Mach);
X_w    = (qbar_0*S/(mass*V_0))...
           *(C_L - C_D_alpha);
X_wdot = 0;
X_q    = 0;
X_de   = 0;
X_dT   = 0;
Z_u    = -(qbar_0*S/(mass*V_0))*( ...
           2*C_L+(Mach_0^2/(1-Mach_0^2))...
           *C_L_Mach);
Z_w    = -(qbar_0*S/(mass*V_0))...
           *(C_D + C_L_alpha);
Z_wdot = -(1/(2*mu_0))*C_L_alphadot;
Z_q    = -(V_0/(2*mu_0))*C_L_q;
Z_de   = 0;
Z_dT   = 0;
M_u    = (qbar_0*S*mac/(Iyy*V_0))...
           *Mach_0*C_m_Mach;
M_w    = (qbar_0*S*mac/(Iyy*V_0))...
           *C_m_alpha;
M_wdot = (rho_0*S*(mac^2)/(4*Iyy))...
           *C_m_alphadot;
M_q    = (rho_0*V_0*S...
           *(mac^2)/(4*Iyy))*C_m_q;
M_de   = 0;
M_dT   = 0;

k_hat  = M_wdot/(1-Z_wdot);

M_a = M_w*V_0;
M_adot = M_wdot*V_0;

A_lon = zeros(4);

A_lon(1,1) = X_u;
A_lon(1,2) = X_w;
A_lon(1,3) = 0;
```

```
A_lon(1,4) = -g*cos(gamma_0);

A_lon(2,1) = Z_u/(1 - Z_wdot);
A_lon(2,2) = Z_w/(1 - Z_wdot);
A_lon(2,3) = (Z_q + V_0)/(1 - Z_wdot);
A_lon(2,4) = -g*sin(gamma_0)/(1 - Z_wdot);

A_lon(3,1) = M_u + k_hat*Z_u;
A_lon(3,2) = M_w + k_hat*Z_w;
A_lon(3,3) = M_q + k_hat*(Z_q+V_0);
A_lon(3,4) = -k_hat*g*sin(gamma_0);

A_lon(4,1) = 0;
A_lon(4,2) = 0;
A_lon(4,3) = 1;
A_lon(4,4) = 0;

B_lon = zeros(4, 2);

B_lon(1, 1) = X_de/mass;
B_lon(1, 2) = X_dT/mass;

B_lon(2, 1) = Z_de/(mass-Z_wdot);
B_lon(2, 2) = Z_dT/(mass-Z_wdot);

B_lon(3, 1) = (M_de + Z_de*M_wdot/(mass-Z_wdot))/Iyy;
B_lon(3, 2) = (M_dT + Z_dT*M_wdot/(mass-Z_wdot))/Iyy;

C_lon = eye(3,4);
D_lon = zeros(3,2);

syms lambda

characteristic_sym_polynomial_lon = det(A_lon-lambda*eye(4));
char_poly_lon = sym2poly(characteristic_sym_polynomial_lon);

a1 = char_poly_lon(1);
a2 = char_poly_lon(2); a2 = a2/a1;
a3 = char_poly_lon(3); a3 = a3/a1;
a4 = char_poly_lon(4); a4 = a4/a1;
a5 = char_poly_lon(5); a5 = a5/a1; a1 = 1;

[V,D] = eig(A_lon);

W = inv(V);

V_SP      = V(:,1);
V_SP      = V_SP/V_SP(4);
V_SP(1,1) = V_SP(1,1)/V_0;
V_SP(2,1) = V_SP(2,1)/V_0;
V_SP(3,1) = V_SP(3,1)/(2*V_0/mac);

V_Ph = V(:,3);
V_Ph = V_Ph/V_Ph(4);
V_Ph(1,1) = V_Ph(1,1)/V_0;
V_Ph(2,1) = V_Ph(2,1)/V_0;
V_Ph(3,1) = V_Ph(3,1)/(2*V_0/mac);

syms a11 a12 a13 a14 a21 a22 a23 a24 a31 a32 a33 a34 a41 a42 a43 a44
AA = [ ...
        a11, a12, a13, a14; ...
        a21, a22, a23, a24; ...
        a31, a32, a33, a34; ...
        a41, a42, a43, a44];

syms U0 Cbar
TT = diag([1/U0, 1/U0, Cbar/(2*U0), 1]);
AA1 = TT * AA / TT;

A_lon1 = ...
    double( ...
        subs( ...
```

```
            AA1, ...
                [a11, a12, a13, a14, ...
                a21, a22, a23, a24, ...
                a31, a32, a33, a34, ...
                    a41, a42, a43, a44 ...
            U0, Cbar], ...
        [A_lon(1,1), A_lon(1,2), A_lon(1,3), A_lon(1,4), ...
        A_lon(2,1), A_lon(2,2), A_lon(2,3), A_lon(2,4), ...
        A_lon(3,1), A_lon(3,2), A_lon(3,3), A_lon(3,4), ...
        A_lon(4,1), A_lon(4,2), A_lon(4,3), A_lon(4,4), ...
        V_0, mac] ...
        ) ...
    );

[V1,D1] = eig(A_lon1);

W1 = inv(V1);

sys = ss( ...
            A_lon1, ...
            zeros(size(A_lon1,1),1), ...
            eye(4,4), ...
            zeros(4,1) ...
            );


V1SP = V1(:,1);
V1SP = V1SP/V1SP(4,1);

V1Ph = V1(:,3);
V1Ph = V1Ph/V1Ph(4,1);

x_init_1 = real(V1SP);
[y1,t1] = initial(sys,x_init_1);

x_init_3 = real(V1Ph);
[y3,t3] = initial(sys,x_init_3);

h = figure;
compassplot(V_SP(1), 'LineWidth', 1.5); hold on
compassplot(V_SP(2), 'LineWidth', 1.5);
compassplot(V_SP(3), 'LineWidth', 1.5);
compassplot(V_SP(4), 'LineWidth', 1.5);
legend( ...
    '\it{u}/\it{U}_0','\it{w}/\it{U}_0','0.5 \it{qc}/\it{U}_0','\theta', ...
    'Location', 'best');
grid on
exportgraphics(h, 'ex161SPfasor.pdf', 'Resolution', 300)

h = figure;
compassplot(V_Ph(1), 'LineWidth', 1.5); hold on
compassplot(V_Ph(2), 'LineWidth', 1.5);
compassplot(V_Ph(3), 'LineWidth', 1.5);
compassplot(V_Ph(4), 'LineWidth', 1.5);
legend( ...
    '\it{u}/\it{U}_0','\it{w}/\it{U}_0','0.5 \it{qc}/\it{U}_0','\theta', ...
    'Location', 'best');
grid on
exportgraphics(h, 'ex161PHfasor.pdf', 'Resolution', 300)

h = figure;
plot( ...
        t1,y1(:,1),...
        t1,y1(:,2),...
        t1,(mac/(2*V_0))*y1(:,3), ...
        t1,y1(:,4), 'LineWidth', 1.5  ...
    );
legend( ...
    '\it{u}/\it{U}_0','\it{w}/\it{U}_0','0.5 \it{qc}/\it{U}_0','\theta', ...
```

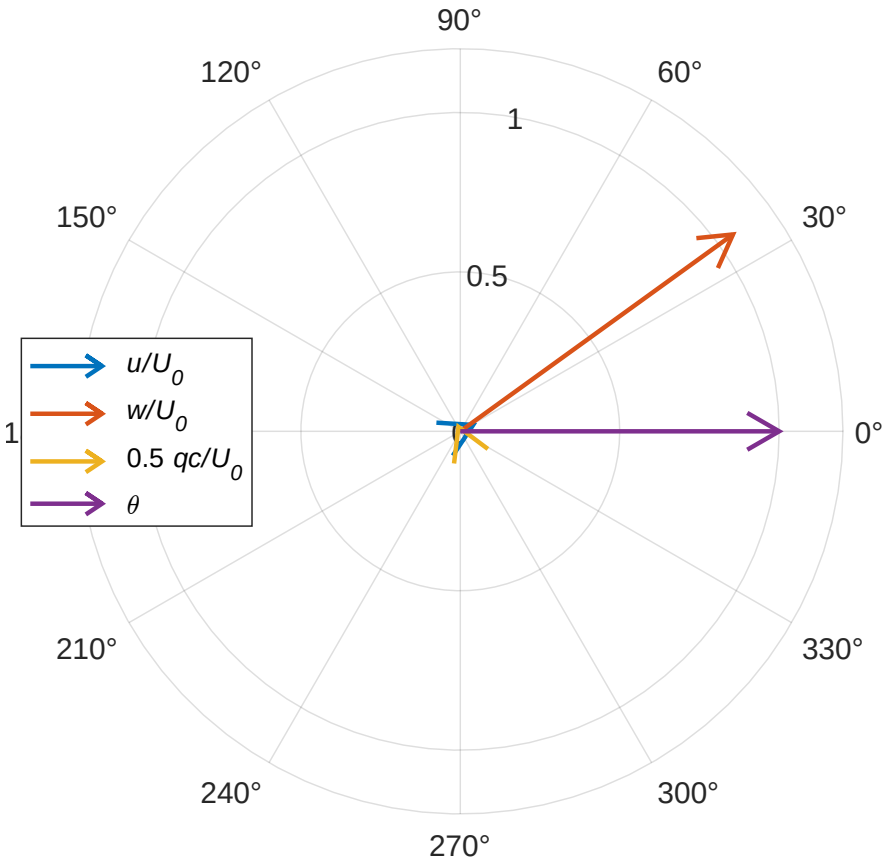


Figure 3.1 Fasor representation of the normalized eigenvectors of short period mode.

```
    'Location', 'best');
ylabel('Perturbation States \it{u}, \it{w}, \it{q}');
xlabel('\it{t} (s)');
grid on
axis square
axis tight
exportgraphics(h, 'ex161SPresponse.pdf', 'Resolution', 300)

h = figure;
plot( ...
    t3,y3(:,1), ...
    t3,y3(:,2), ...
    t3,(mac/(2*V_0))*y3(:,3), ...
    t3,y3(:,4), 'LineWidth', 1 ...
    );
legend( ...
    '\it{u}/\it{U}_0','\it{w}/\it{U}_0','0.5 \it{qc}/\it{U}_0','\theta', ...
    'Location', 'best');
ylabel('Perturbation States \it{u}, \it{w}, \it{q}');
xlabel('\it{t} (s)');
grid on
axis square
axis tight
exportgraphics(h, 'ex161PHresponse.pdf', 'Resolution', 300)
```

Non-linear and linearized responses to an impulsive climbing perturbation

Listing 3.2 Comparison between non-linear and linearized responses.

```
clear all; close all; clc;

global ...
    g ...
```

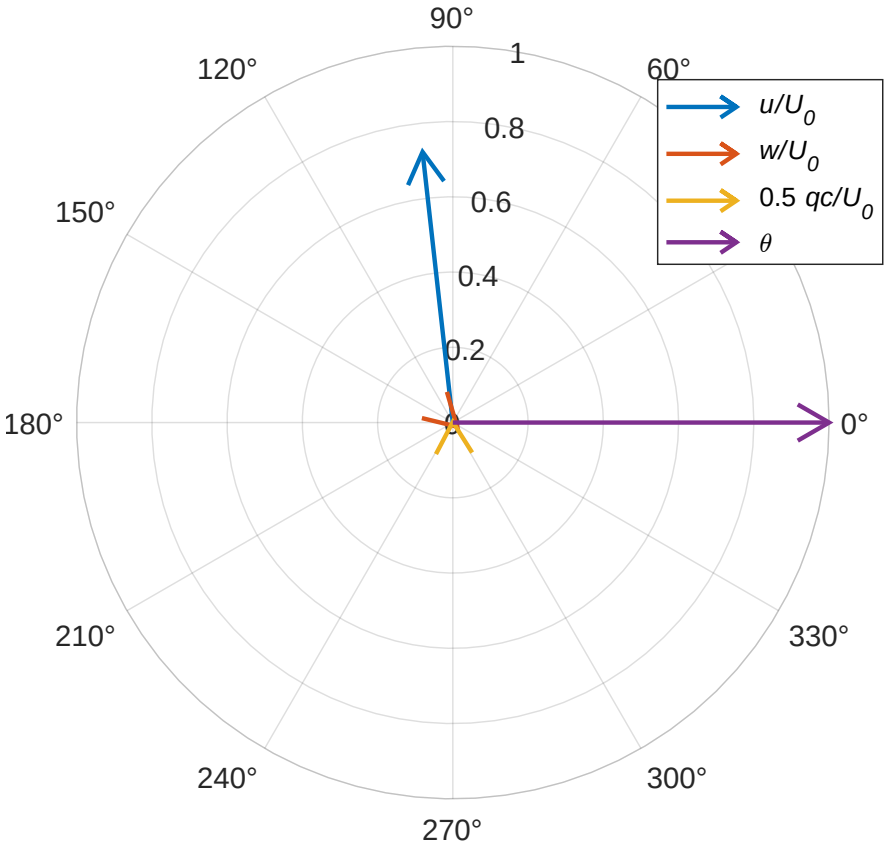


Figure 3.2 Fasor representation of the normalized eigenvectors of phugoid mode.

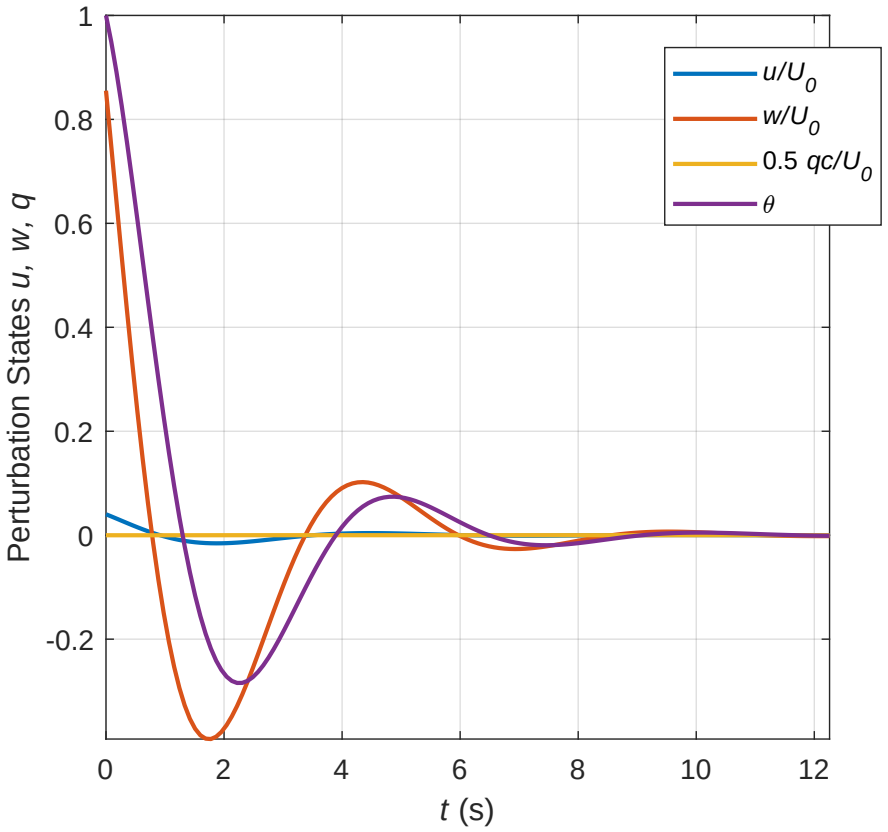


Figure 3.3 Time histories of the short period mode perturbation state variables as a result of the dynamic response.

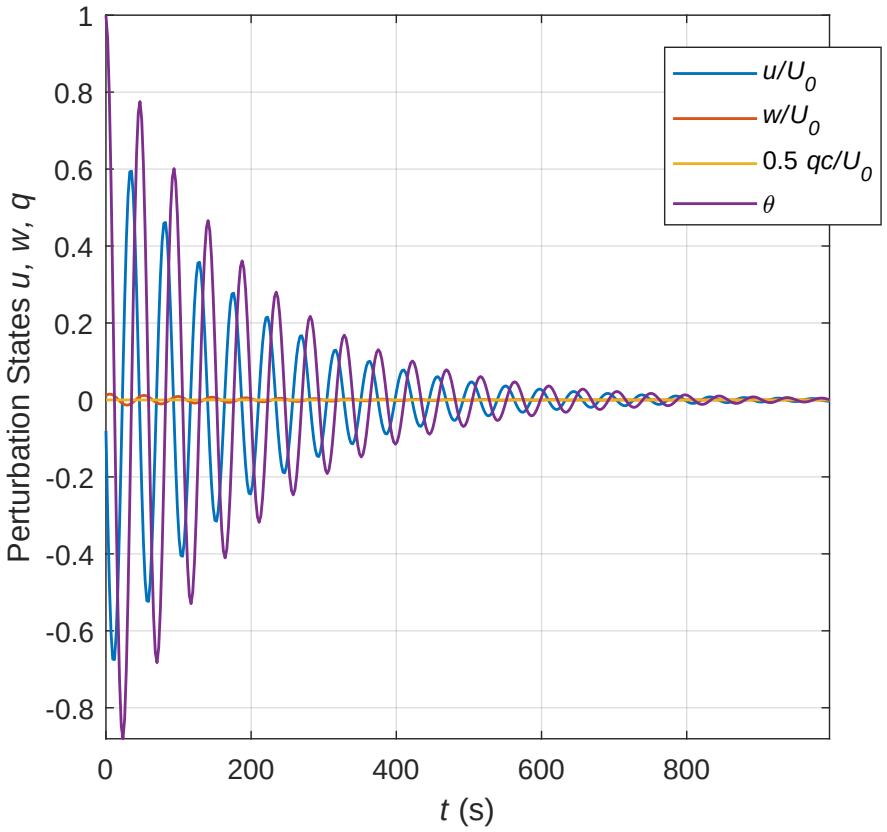


Figure 3.4 Time histories of the phugoid mode perturbation state variables as a result of the dynamic response.

```
    z_0 V_0 q_0 gamma_0 ...
    rho_0 ...
    delta_e ...
    delta_T ...
    myAC
load('dati_velivolo.mat')

x_0 = 0;
z_0 = -4000.0;
V_0 = 100;
q_0 = 0.0;
gamma_0 = 0.0;
[~, sound_speed_0, ~, rho_0] = atmosisa(-z_0);

g = 9.81;

myAC.T_max = AC.AircraftDataAdvanced.Propulsion.CRUISE.f_Thrust(-z_0,
    ↪ V_0/sound_speed_0);
x0 = [ ...
        0; ...                  % alpha_0
        0; ...                  % delta_e_0
        0.5];                   % delta_T_0

lb =[convang(-8,'deg','rad'), ... % alpha minimo
    convang(-30,'deg','rad'), ... % delta_e minimo
    0.2 ...                       % delta_T minimo
    ];
ub =[convang( 20,'deg','rad'), ... % alpha massimo
    convang( 10,'deg','rad'), ... % delta_e massimo
    1.0 ...                       % delta_T massimo
    ];

options = optimset('tolfun',1e-9,'Algorithm','interior-point');
[x,fval] = fmincon(@costLongEquilibriumStaticStickFixedVel, x0, ...
    [],[],[],[],lb,ub,@myNonLinearConstraint,options);

```

```
alpha_0_rad   = x(1);
alpha_0_deg   = convang(x(1),'rad','deg');
delta_e_0_rad = x(2);
delta_e_0_deg = convang(x(2),'rad','deg');
delta_T_0     = x(3);
theta_trim = gamma_0 + alpha_0_rad - myAC.mu_x;

t_0 = 0;
t_finale = 300;

N = 200;
time = linspace (t_0, t_finale, N);

X0 = [ V_0;
       alpha_0_rad;
       q_0;
       x_0;
       z_0;
       theta_trim];

delta_T = @(t) interp1 ([t_0 t_finale], [delta_T_0 delta_T_0], t);

t1 = 1;
t2 = 2.5;
t3 = 4;

delta_e = @(t) interp1( ...
    [0, t1, t2, t3, t_finale], ...
    [delta_e_0_rad, delta_e_0_rad, delta_e_0_rad-3*pi/180,
    ↪ delta_e_0_rad, delta_e_0_rad], ...
    t, 'linear');

options = odeset( 'RelTol', 1e-9, 'AbsTol',1e-9*ones(1,6));

[vTime, X] = ode45(@eqLongDynamicStickFixed,time,X0',options);

DELTA_V     = X(:,1) - V_0;
DELTA_alpha = convang (X(:,2), 'rad', 'deg') - alpha_0_deg;
DELTA_q     = convangvel (X(:,3), 'rad/s', 'deg/s') - q_0;
X_EG        = X(:,4);
DELTA_h     = -(X(:,5) - z_0);
THETA       = X(:,6);
GAMMA       = THETA + myAC.mu_x - X(:,2);

Weight  = myAC.W;
mass    = myAC.mass;
h_0     = -z_0;
[T_0,a_0,p_0,rho_0] = atmosisa(h_0);
Mach_0  = V_0/a_0;
Iyy     = myAC.Iyy;
S       = myAC.S;
mac     = myAC.mac;
qbar_0  = 0.5*rho_0*(V_0^2);
mu_0    = mass/(0.5*rho_0*S*mac);
gamma_0 = 0.0;

C_L             =
    ↪ AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL(alpha_0_rad,...
                    delta_e_0_rad,0,0);
C_D             =
    ↪ AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CD.f_CD(alpha_0_rad,
    ↪ delta_e_0_rad);
C_L_Mach        = 0.0;
Delta_alpha     = deg2rad(1);
Delta_delta_e   =  deg2rad(1);
C_L_alpha       = (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL(
    ↪ ...
     alpha_0_rad+Delta_alpha,...
                    delta_e_0_rad,0,0)-...
                AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL( ...
                alpha_0_rad-Delta_alpha,...
                    delta_e_0_rad,0,0))/(2*Delta_alpha);
```

```
C_L_alphadot    = myAC.CL_alphadot ;
C_L_q           = myAC.CL_q;
C_D_alpha       = (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CD.f_CD( ...
    alpha_0_rad+Delta_alpha, delta_e_0_rad)-...
                    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CD.f_CD( ...
                    alpha_0_rad-Delta_alpha, delta_e_0_rad))/(2*Delta_alpha);
C_D_alphadot    = 0.0;
C_D_q           = 0.0;
C_D_Mach        = 0.0;

C_m_alpha    = (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
    alpha_0_rad+Delta_alpha, delta_e_0_rad,0,0)...
                -
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
                alpha_0_rad-Delta_alpha, delta_e_0_rad,0,0))/...
                    (2*Delta_alpha);
C_m_alphadot = myAC.Cpitch_alphadot;
C_m_q        = myAC.Cpitch_q;
C_m_Mach     = 0.0;
C_L_de       = (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL( ...
    alpha_0_rad,...
                    delta_e_0_rad+Delta_delta_e,0,0)-...

    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.CL.f_CL(alpha_0_rad,...
                    delta_e_0_rad-Delta_delta_e,0,0))/(2*Delta_delta_e);
C_m_de       =
    (AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
    alpha_0_rad, delta_e_0_rad +Delta_delta_e,0,0)...
                -
    AC.AircraftDataAdvanced.Aerodynamics.CRUISE.Cpitch.f_Cpitch( ...
                alpha_0_rad, delta_e_0_rad-Delta_delta_e,0,0))/...
                (2*Delta_alpha);

X_u    = -(qbar_0*S/(mass*V_0))...
            *(2*C_D + Mach_0*C_D_Mach);
X_w    = (qbar_0*S/(mass*V_0))...
            *(C_L - C_D_alpha);
X_wdot = 0;
X_q    = 0;
X_de   = 0;
X_dT   = 0;

Z_u    = -(qbar_0*S/(mass*V_0))*( ...
            2*C_L+(Mach_0^2/(1-Mach_0^2))...
            *C_L_Mach);
Z_w    = -(qbar_0*S/(mass*V_0))...
            *(C_D + C_L_alpha);
Z_wdot = -(1/(2*mu_0))*C_L_alphadot;
Z_q    = -(V_0/(2*mu_0))*C_L_q;

Z_de   = 0.0;
Z_dT   = 0.0;
M_u    = (qbar_0*S*mac/(Iyy*V_0))...
            *Mach_0*C_m_Mach;
M_w    = (qbar_0*S*mac/(Iyy*V_0))...
            *C_m_alpha;
M_wdot = (rho_0*S*(mac^2)/(4*Iyy))...
            *C_m_alphadot;
M_q    = (rho_0*V_0*S...
            *(mac^2)/(4*Iyy))*C_m_q;
M_de   = -0.5;
M_dT   = 0.0;

M_a    = M_w*V_0; M_adot = M_wdot*V_0;

k_hat  = M_wdot/(1 -Z_wdot);

A_lon(1,1) = X_u;
```

```
A_lon(1,2) = X_w;
A_lon(1,3) = 0;
A_lon(1,4) = -g*cos(gamma_0);

A_lon(2,1) = Z_u/(1 - Z_wdot);
A_lon(2,2) = Z_w/(1 - Z_wdot);
A_lon(2,3) = (Z_q + V_0)/(1 - Z_wdot);
A_lon(2,4) = -g*sin(gamma_0)/(1 - Z_wdot);

A_lon(3,1) = M_u + k_hat*Z_u;
A_lon(3,2) = M_w + k_hat*Z_w;
A_lon(3,3) = M_q + k_hat*(Z_q+V_0);
A_lon(3,4) = -k_hat*g*sin(gamma_0);

A_lon(4,1) = 0;
A_lon(4,2) = 0;
A_lon(4,3) = 1;
A_lon(4,4) = 0;

B_lon = [ ...
        X_de/mass,                                    X_dT/mass; ...
        Z_de/(mass-Z_wdot),                  Z_dT/(mass-Z_wdot); ...
        (M_de + Z_de*M_wdot/(mass-Z_wdot))/Iyy, ...
                            (M_dT + Z_dT*M_wdot/(mass-Z_wdot))/Iyy; ...
        0,                                                 0];

C_lon = eye(3,4);
D_lon = zeros(3,2);

syms lambda

characteristic_sym_polynomial_lon = det(A_lon-lambda*eye(4));
char_poly_lon = sym2poly(characteristic_sym_polynomial_lon);

a1 = char_poly_lon(1);
a2 = char_poly_lon(2); a2 = a2/a1;
a3 = char_poly_lon(3); a3 = a3/a1;
a4 = char_poly_lon(4); a4 = a4/a1;
a5 = char_poly_lon(5); a5 = a5/a1; a1 = 1;

[V,D] = eig(A_lon);

W = inv(V);


V_SP      = V(:,1);
V_SP      = V_SP/V_SP(4);
V_SP(1,1) = V_SP(1,1)/V_0;
V_SP(2,1) = V_SP(2,1)/V_0;
V_SP(3,1) = V_SP(3,1)/(2*V_0/mac);

V_Ph = V(:,3);
V_Ph = V_Ph/V_Ph(4);
V_Ph(1,1) = V_Ph(1,1)/V_0;
V_Ph(2,1) = V_Ph(2,1)/V_0;
V_Ph(3,1) = V_Ph(3,1)/(2*V_0/mac);

sys = ss( ...
                A_lon, ...
                B_lon , ...
                eye(4,4) , ...
                zeros(4,2) ...
            );

t_BP      = [0, t1, t2, t3, t_finale];
de_deg_BP = [delta_e_0_deg, delta_e_0_deg, delta_e_0_deg-3,
    ↪ delta_e_0_deg, delta_e_0_deg];

t_Elevator_Doublet = [0:0.025:t_finale]';

de_rad_Elevator_Doublet = ...
            interp1(t_BP,convang(de_deg_BP-delta_e_0_deg,'deg','rad'),
```

```
    ↪ ...
242             t_Elevator_Doublet);
243 de_deg_Elevator_Doublet = convang(de_rad_Elevator_Doublet,'rad','deg');
244
245 u_Elevator_Doublet = [ ...
246         de_rad_Elevator_Doublet,
    ↪ zeros(length(de_rad_Elevator_Doublet),1)];
247
248 x0_Elevator_Doublet = [0;0;0;0];
249
250 [y_Elevator_Doublet, t_Elevator_Doublet, x_Elevator_Doublet] = ...
251 lsim(sys, u_Elevator_Doublet, t_Elevator_Doublet, x0_Elevator_Doublet);
252
253 h = figure;
254 plot(t_Elevator_Doublet,de_deg_Elevator_Doublet,'LineWidth', 1.5);
255 ylabel('\delta_e (deg)');
256 xlabel('{\it{t}} (s)');
257 grid on;
258 axis square; axis tight;
259 xlim([0, 10])
260 exportgraphics(h, 'ex165elevator.pdf', 'Resolution', 300)
261
262 h = figure;
263 set(h, 'Position', 0.75*get(0, 'MonitorPositions'))
264 positions = [0.05, 0.79, 0.7, 0.20; ...
265             0.05, 0.55, 0.7, 0.20; ...
266             0.05, 0.31, 0.7, 0.20; ...
267             0.05, 0.07, 0.7, 0.20];
268 lgnd = {'Non-linear response', 'Linearized response'};
269 subplot('Position', positions(1, :))
270     plot(time, DELTA_V, ...
271         t_Elevator_Doublet, y_Elevator_Doublet(:,1)*2*10^6, 'LineWidth',
    ↪ 2)
272     ylabel('\Delta{\it{V}} (m/s)', 'FontSize', 18)
273     axis tight
274     grid on
275 subplot('Position', positions(2, :))
276     plot(time,DELTA_alpha, ...
277     t_Elevator_Doublet, y_Elevator_Doublet(:,2)*10^6, 'LineWidth', 2)
278     ylabel('\alpha (deg)', 'FontSize', 18)
279     axis tight
280     grid on
281 subplot('Position', positions(3, :))
282     plot(time, DELTA_q,...
283     t_Elevator_Doublet, y_Elevator_Doublet(:,3)*10^8, 'LineWidth', 2)
284     ylabel('{\it{q}} (deg/s)', 'FontSize', 18)
285     axis tight
286     grid on
287 subplot('Position', positions(4, :))
288     plot(time, convang(THETA-theta_trim, 'rad', 'deg'), ...
289         t_Elevator_Doublet, ...
290         convang(y_Elevator_Doublet(:,4)*10^6, 'rad', 'deg'),
    ↪ 'LineWidth', 2)
291     xlabel('{\it{t}} (s)', 'FontSize', 18)
292     ylabel('\theta (deg)', 'FontSize', 18)
293     axis tight
294     grid on
295 lgnd = legend(lgnd);
296         lgnd.Position(1) = 0.775;
297         lgnd.Position(2) = positions(1, 2)+positions(1,
    ↪ 4)-lgnd.Position(4);
298 exportgraphics(h, 'ex165comparison.pdf', 'Resolution', 400)
```

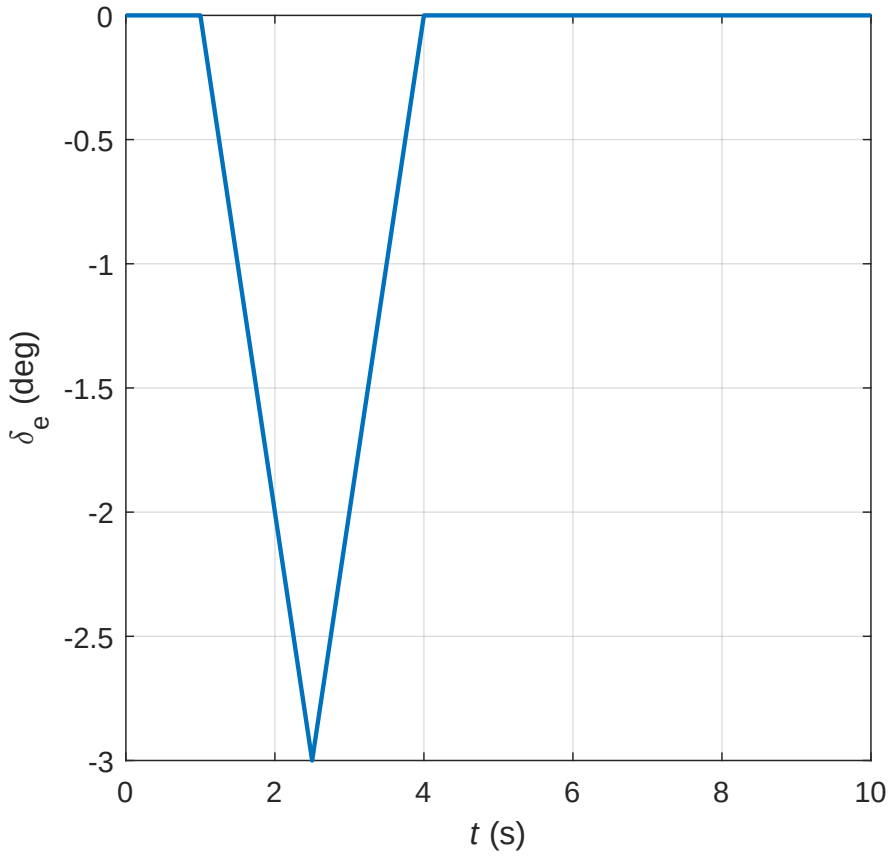


Figure 3.5 Impulsive climbing perturbation time law.

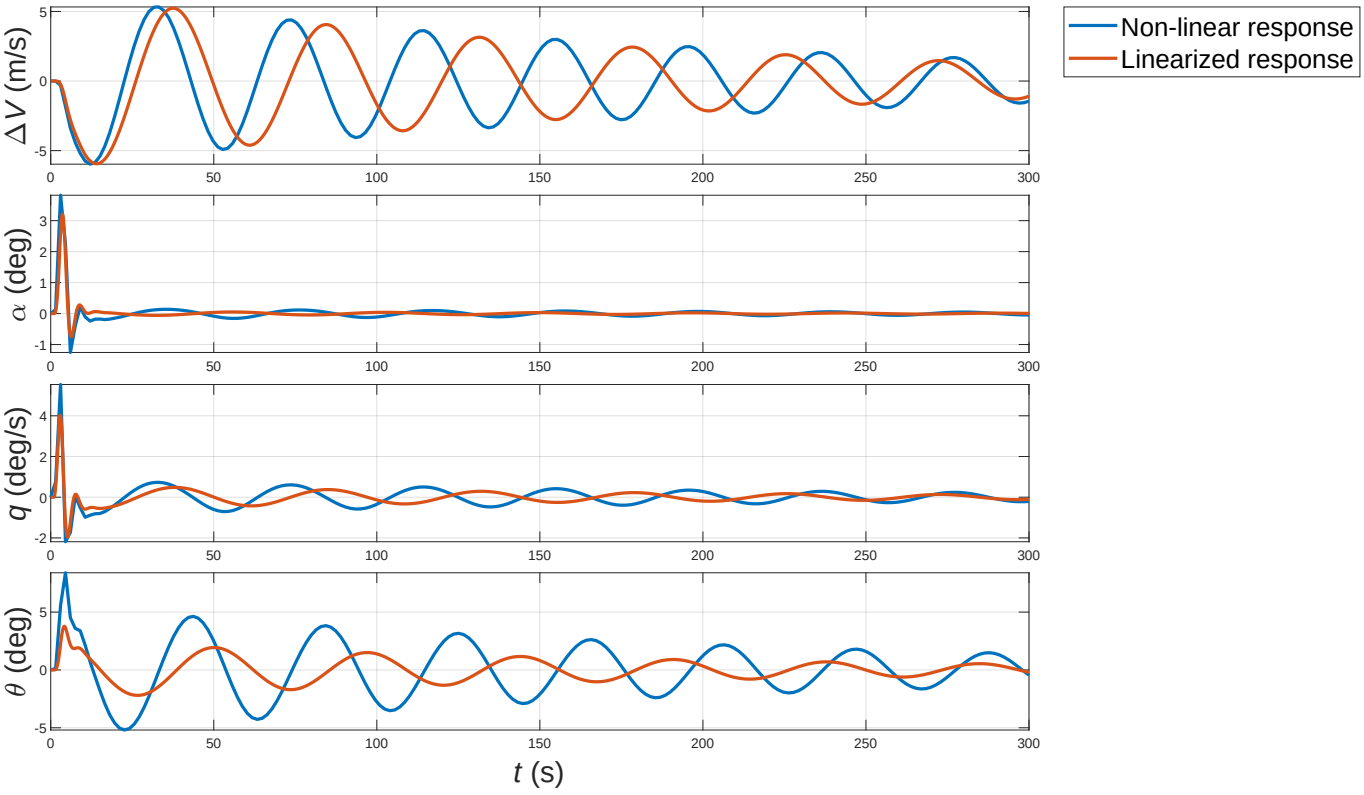


Figure 3.6 Comparison between the non-linear model response and the linearized model response.

3.6 Lateral-directional dynamics

With the same approach used in § 3.5 we can express the lateral-directional dynamical system as a reduced system of differential equations with the position

$$\boldsymbol{x}_{\mathrm{LD}} = \begin{Bmatrix} v \\ p \\ r \\ \varphi \end{Bmatrix} \tag{3.35}$$

This yields the following LTI state propagation equation

$$\dot{\boldsymbol{x}}_{\mathrm{LD}} = A_{\mathrm{LD}}\boldsymbol{x}_{\mathrm{LD}} + B_{\mathrm{LD}}\boldsymbol{u}_{\mathrm{LD}} \tag{3.36}$$

with characteristic equation

$$(s - \lambda_{\mathrm{Spiral}})\,(s - \lambda_{\mathrm{Roll}})\,(s - \lambda_{\mathrm{DR}})\left(s - \lambda^*_{\mathrm{DR}}\right) \tag{3.37}$$

and with the matrices A_{LD} and B_{LD} being

$$A_{\mathrm{LD}} = \begin{bmatrix} Y'_v & Y'_p & Y'_r - U_0 & g\cos\Theta_0 \\ \mathscr{L}'_v & \mathscr{L}'_p & \mathscr{L}'_r & 0 \\ \mathscr{N}'_v & \mathscr{N}'_p & \mathscr{N}'_r & 0 \\ 0 & 1 & \tan\Theta_0 & 0 \end{bmatrix} \tag{3.38}$$

$$B_{\mathrm{LD}} = \begin{bmatrix} Y'_{\delta_{\mathrm{a}}} & Y'_{\delta_{\mathrm{r}}} \\ \mathscr{L}'_{\delta_{\mathrm{a}}} & \mathscr{L}'_{\delta_{\mathrm{r}}} \\ \mathscr{N}'_{\delta_{\mathrm{a}}} & \mathscr{N}'_{\delta_{\mathrm{r}}} \\ 0 & 0 \end{bmatrix} \tag{3.39}$$

where $Y'_\bullet = \hat{Y}_\bullet$ and

$$\mathscr{L}'_\bullet = \frac{\mathscr{L}_\bullet}{I'_{xx}} + \frac{\mathscr{N}_\bullet}{I'_{xz}} = \frac{\hat{\mathscr{L}}_\bullet + i_1\hat{\mathscr{N}}_\bullet}{1 - i_1 i_2}, \qquad \mathscr{N}'_\bullet = \frac{\mathscr{L}_\bullet}{I'_{xz}} + \frac{\mathscr{N}_\bullet}{I'_{zz}} = \frac{i_2\hat{\mathscr{L}}_\bullet + \hat{\mathscr{N}}_\bullet}{1 - i_1 i_2} \tag{3.40}$$

with

$$i_1 = \frac{I_{xz}}{I_{xx}}, \qquad i_2 = \frac{I_{xz}}{I_{zz}} \tag{3.41}$$

and

$$I'_{xx} = I_{xx} - \frac{I^2_{xz}}{I_{zz}}, \qquad I'_{zz} = I_{zz} - \frac{I^2_{xz}}{I_{xx}}, \qquad I'_{xz} = \frac{I_{xx}I_{zz} - I^2_{xz}}{I_{xz}} \tag{3.42}$$

The dimensional derivative inside the matrices can be evaluated as shown in table 3.3 at the next page.

Derivative	Formula	Unit
$\hat{Y}_\beta$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S}{m} C_{Y\beta}$	ms^{-2}
$\hat{Y}_p$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S}{m} C_Y \frac{b}{2U_0} C_{Yp}$	ms^{-1}
$\hat{Y}_r$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S}{m} C_Y \frac{b}{2U_0} C_{Yr}$	ms^{-1}
$\hat{Y}_{\delta_\mathrm{a}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S}{m} C_{Y\delta_\mathrm{a}}$	ms^{-2}
$\hat{Y}_{\delta_\mathrm{r}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 S}{m} C_{Y\delta_\mathrm{r}}$	ms^{-2}
$\hat{\mathscr{L}}_\beta$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{xx}} C_{\mathscr{L}\beta}$	s^{-2}
$\hat{\mathscr{L}}_p$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{xx}} \frac{b}{2U_0} C_{\mathscr{L}p}$	s^{-1}
$\hat{\mathscr{L}}_r$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{xx}} \frac{b}{2U_0} C_{\mathscr{L}r}$	s^{-1}
$\hat{\mathscr{L}}_{\delta_\mathrm{a}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{xx}} C_{\mathscr{L}\delta_\mathrm{a}}$	s^{-2}
$\hat{\mathscr{L}}_{\delta_\mathrm{r}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{xx}} C_{\mathscr{L}\delta_\mathrm{r}}$	s^{-2}
$\hat{\mathscr{N}}_\beta$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{zz}} C_{\mathscr{N}\beta}$	s^{-2}
$\hat{\mathscr{N}}_p$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{zz}} \frac{b}{2U_0} C_{\mathscr{N}p}$	s^{-1}
$\hat{\mathscr{N}}_r$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{zz}} \frac{b}{2U_0} C_{\mathscr{N}r}$	s^{-1}
$\hat{\mathscr{N}}_{\delta_\mathrm{a}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{zz}} C_{\mathscr{N}\delta_\mathrm{a}}$	s^{-2}
$\hat{\mathscr{N}}_{\delta_\mathrm{r}}$	$= \frac{\frac{1}{2}\rho_0 U_0^2 Sb}{I_{zz}} C_{\mathscr{N}\delta_\mathrm{r}}$	s^{-2}

Table 3.3 Dimensional derivatives of stability of the lateral-directional motion.

Chapter 4

Take-Off

In this chapter a model for *Take-Off* simulations will be introduced. The model equations discussed and their implementation into an exercise at the end of the chapter are developed upon the reference [2].

4.1 Take-Off simulation model

The model soon presented will be constituted by a set of differential equations that model the aircraft motion during the whole take-off phase: (i) ground roll, (ii) transition to the airborne phase, (iii) initial climb up to cruise altitude and (iv) leveled trimmed flight.

The aircraft is modeled as a point mass constrained to move in a vertical plane under the action of propulsive, aerodynamic ground contact forces and weight. Therefore, it is treated as a dynamic system in its state-space representation, with state vector $\boldsymbol{x}$, input vector $\boldsymbol{u}$ and state propagation equation $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u})$ defined as follows

$$\boldsymbol{x} = \begin{Bmatrix} s \\ V \\ \gamma \\ h \\ m \end{Bmatrix}, \qquad \boldsymbol{u} = \begin{Bmatrix} \alpha \\ \delta_{\mathrm{T}} \\ \delta_{\mathrm{f}} \end{Bmatrix}, \qquad \boldsymbol{f} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{Bmatrix} \tag{4.1}$$

where

- s is the space the aircraft has traveled.
- V is the true airspeed of the aircraft.
- γ is the flight path angle.
- h is the altitude.
- m is the aircraft mass.

The right hand side of the dynamical system is given by the equations of motion. In particular they can be computed to obtain the following

$$f_1 = V \tag{4.2}$$

$$f_2 = \frac{1}{m}\begin{cases} T - D - \mu(W - L) & \sigma < 1 \text{ (on ground)} \\ T\cos\alpha - W\sin\gamma - D & \sigma \geq 1 \text{ (airborne)} \end{cases} \tag{4.3}$$

$$f_3 = \frac{1}{mV}\begin{cases} 0 & \sigma < 1 \text{ (on ground)} \\ L + T\sin\alpha - W\cos\gamma & \sigma \geq 1 \text{ (airborne)} \end{cases} \tag{4.4}$$

$$f_4 = V\sin\gamma \tag{4.5}$$

$$f_5 = -\dot{m}_\mathrm{f} \tag{4.6}$$

where μ is the friction coefficient with the ground, $\dot{m}_\mathrm{f}$ is the intrinsically positive *fuel mass flow rate* and σ is a *switching function* from ground roll phase to airborne phase, defined as follows

$$\sigma = \frac{L}{W\cos\gamma} \tag{4.7}$$

It is worth discussing the recommended time law to assign to the input variable α. At first, during ground roll phase, it must be assigned to a constant value α_g, i.e. the angle of attack on the ground. When the aircraft reaches an airspeed $V_\mathrm{rot} = k_\mathrm{rot}V_\mathrm{stall}$ the rotation starts and a specific time law for the angle of attack $\alpha_1(t)$ is recommended. After the rotation, at time t_hold, the angle of attack is assigned as constant. Then, at t_climb the climb phase starts and the angle of attack has to be controlled in order to obtain a desired γ_cmd. This can be mathematically expressed in its time law as shown in the following

$$\alpha(t) = \begin{cases} \alpha_\mathrm{g} & t < t_\mathrm{rot} \\ \alpha_1(t) & t_\mathrm{rot} \leq t < t_\mathrm{hold} \\ \alpha_1(t_\mathrm{hold}) & t_\mathrm{hold} \leq t < t_\mathrm{hold} + \Delta t_\mathrm{hold} \\ \text{PID-controlled} & t \geq t_\mathrm{hold} + \Delta t_\mathrm{hold} \end{cases} \tag{4.8}$$

where the time law $\alpha_1(t)$ is given by its diminishing derivative

$$\dot{\alpha}_1(t) = \dot{\alpha}_0(1 - k_\alpha\alpha) \tag{4.9}$$

for appropriates initial slope $\dot{\alpha}_0$ and constant factor k_α.

The time value t_hold depends on the lift coefficient. It is the time at which the lift coefficient is near the maximum lift coefficient in take-off configuration (flaps down). In particular, the hold phase is triggered when $C_L = k_{C_{L_\mathrm{max}}}C_{L_\mathrm{max}}$, where $k_{C_{L_\mathrm{max}}}$ is a safety factor that ensures the stall is never reached.

4.2 Simulink model and full simulation

Since the main idea behind this take-off model has been presented, it is now possible to implement it in Simulink to be able to solve for the unknown variables and to evaluate the main take-off performance parameters.

In the following pages the figures of all the plant will follow. It is worth noticing how each subsystem has been color coded to ease comprehension. (i) Cyan: it is responsible for the dynamics, (ii) Green: computes the input variables, (iii) Orange: evaluates triggered variables and (iv) White: carries out generic computations.

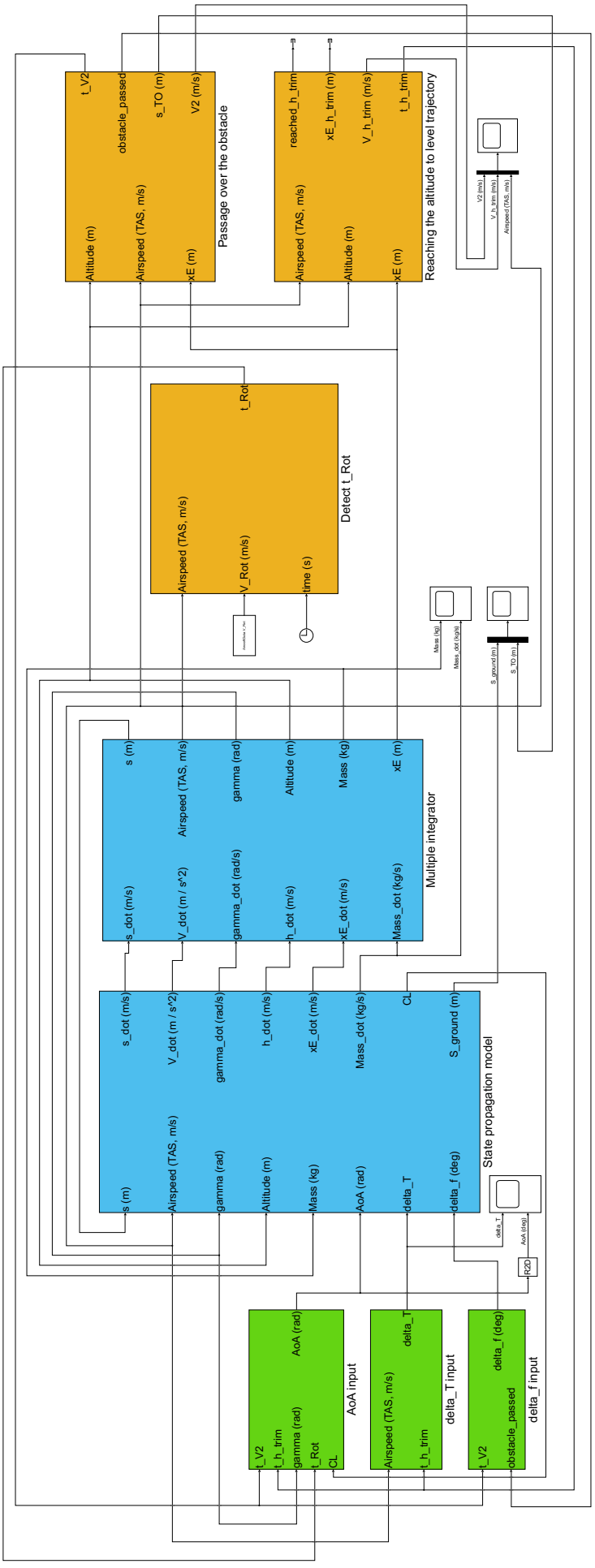


Figure 4.1 Main parent system. It is mainly composed by three groups of subsystems, as the colors suggest.

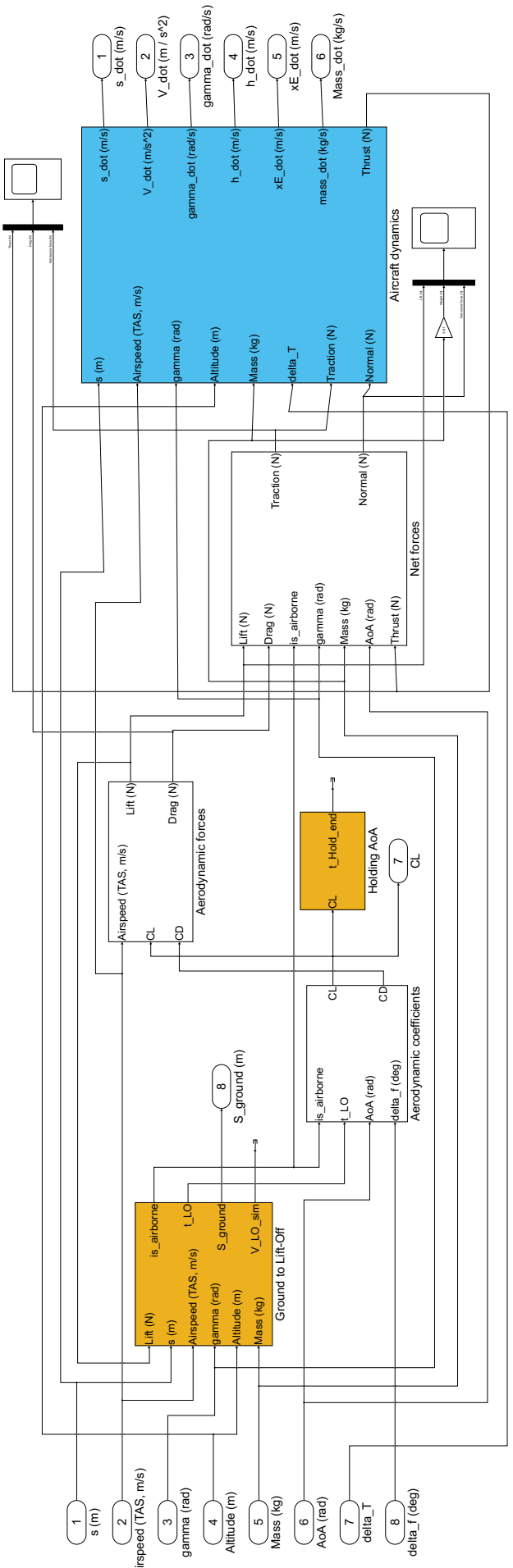


Figure 4.2 Subsystem State propagation model. It is the main subsystem responsible for the aircraft dynamics computation. It evaluates the right hand side function $\boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{u})$ of the dynamical system.

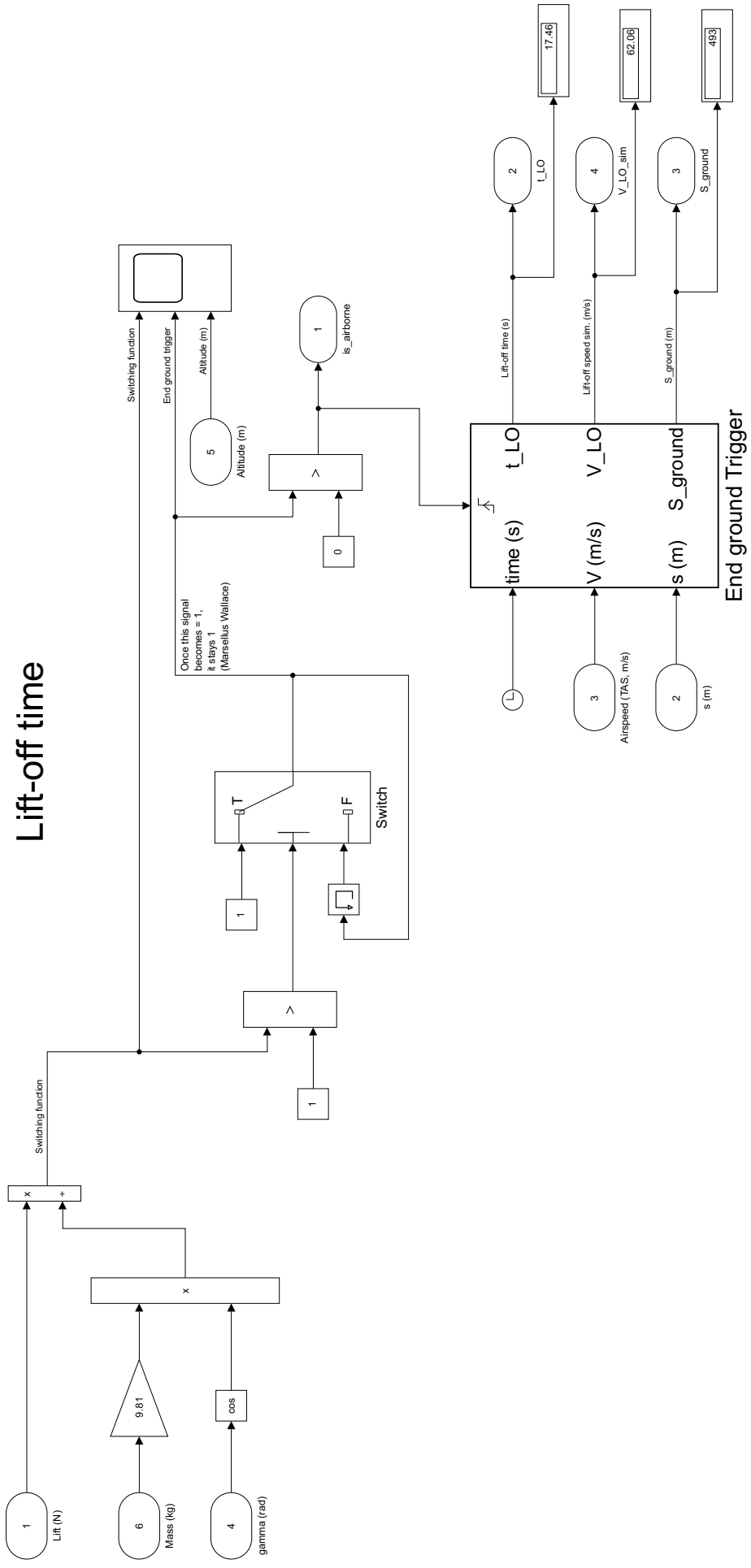


Figure 4.3 Subsystem Ground to Lift-Off. It starts triggers for the ground phase, rotation phase and up to the *lift-off* of the aircraft.

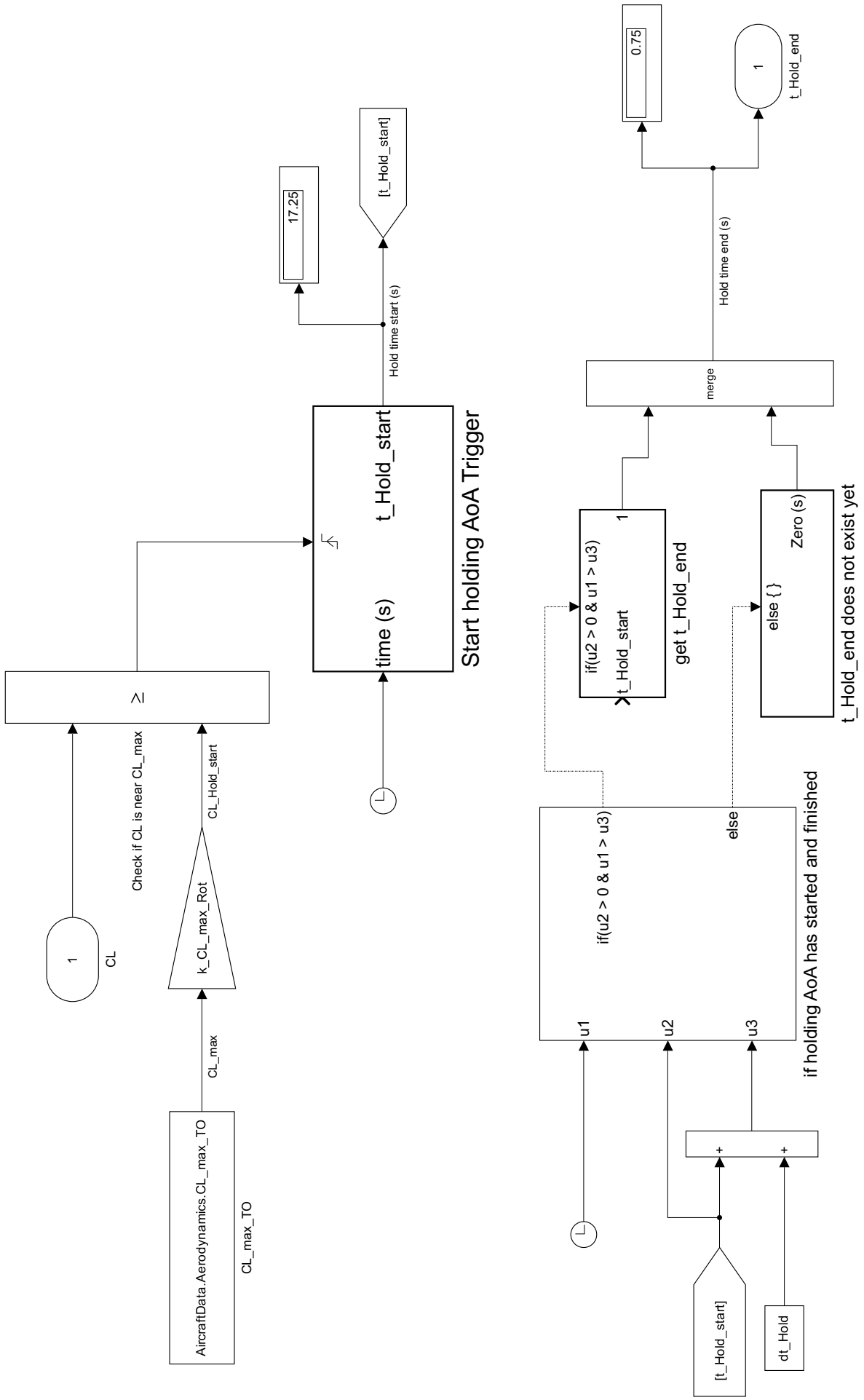


Figure 4.4 Subsystem Holding AoA. It detects when the holding phase starts and ends.

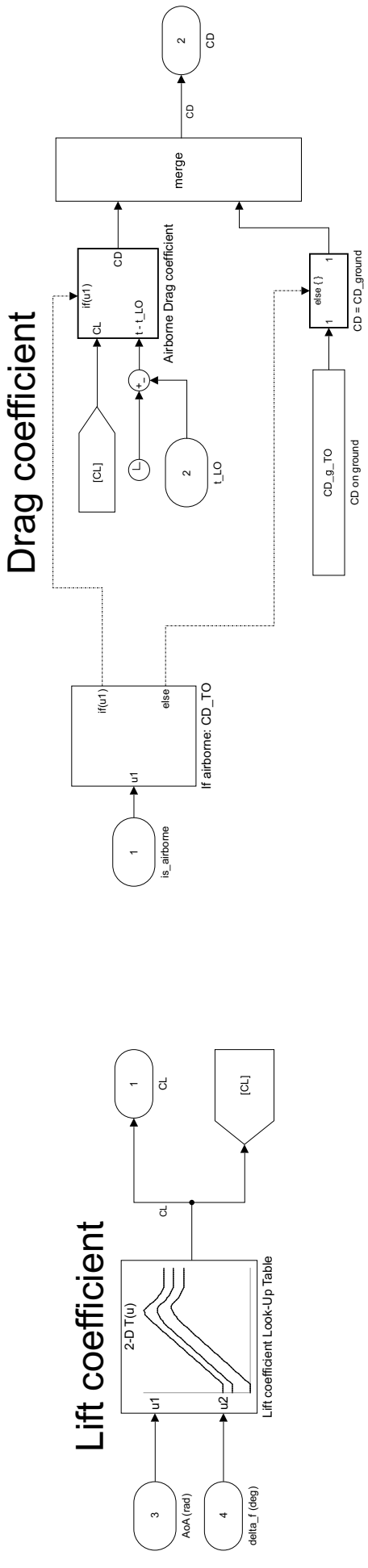


Figure 4.5 Subsystem Aerodynamic coefficients. It computes the aerodynamic coefficients C_L and C_D.

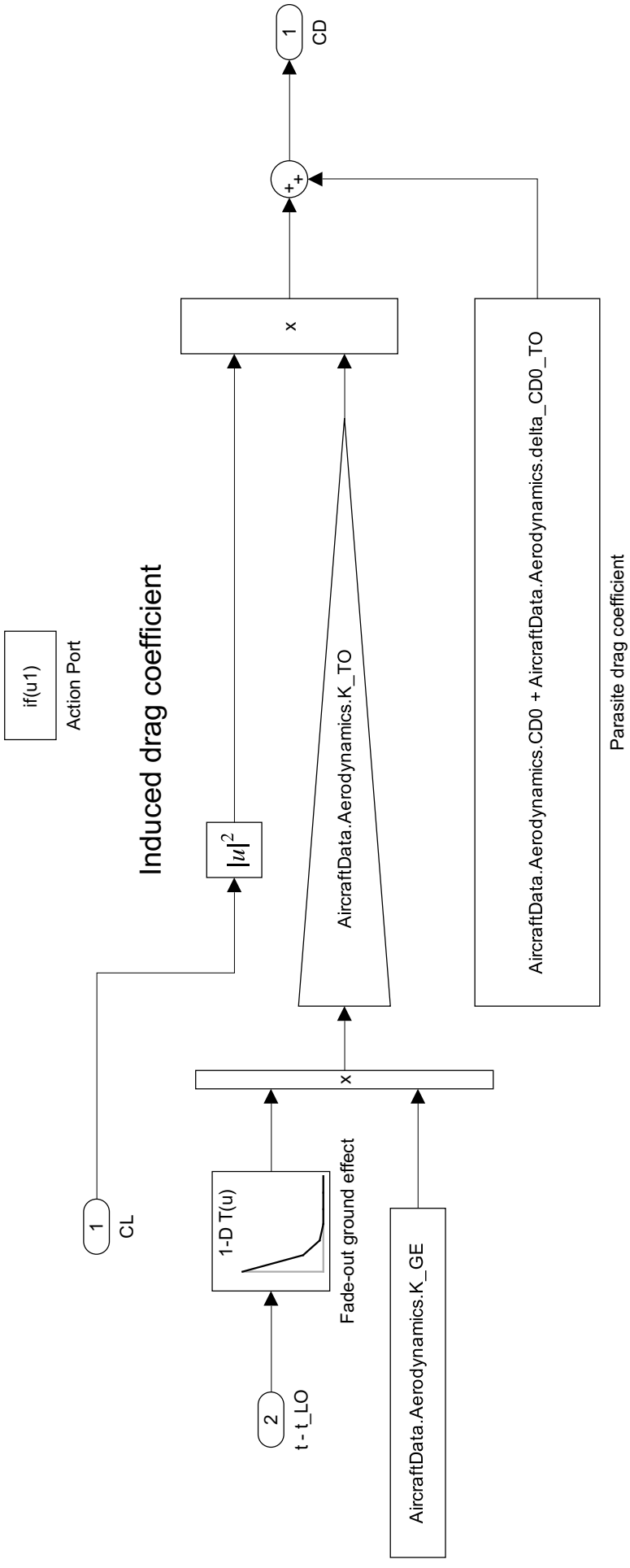


Figure 4.6 Subsystem Airborne Drag coefficient. It evaluates the drag coefficient when the aircraft has passed the ground and rotation phases.

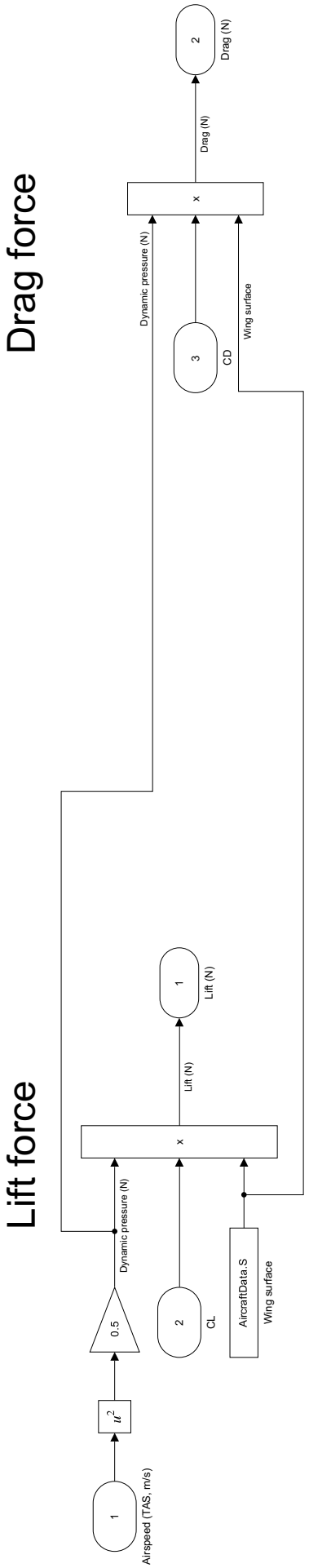


Figure 4.7 Subsystem Aerodynamic forces. It computes the lift L and the drag D.

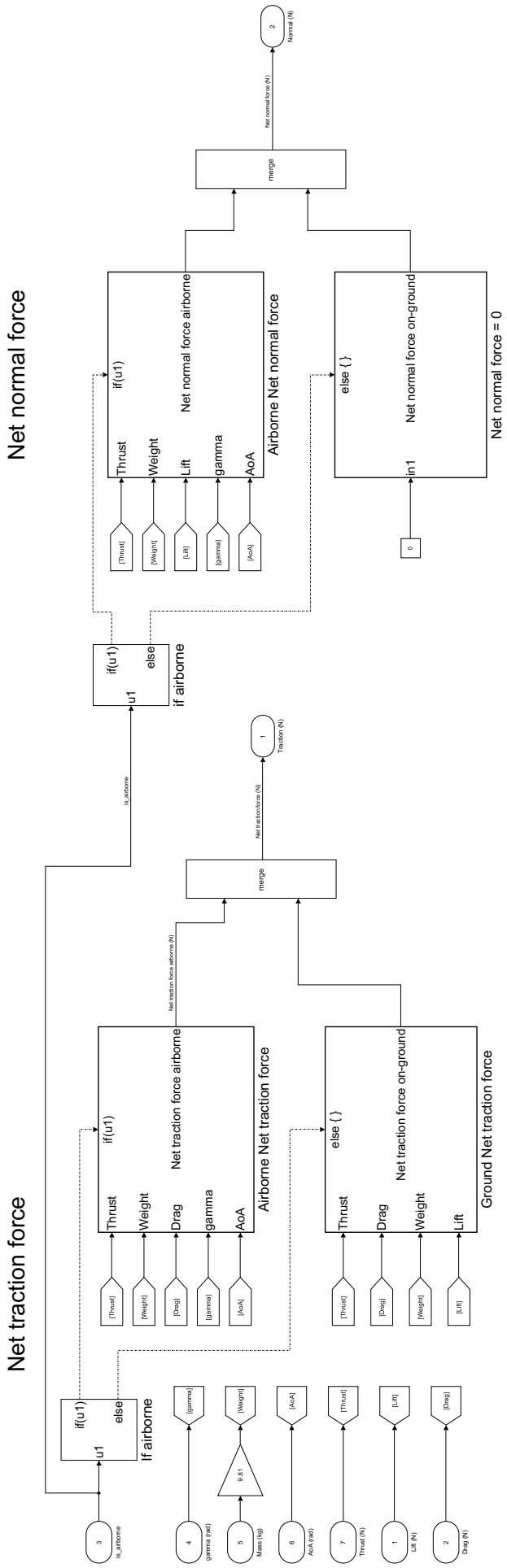


Figure 4.8 Subsystem Net forces. It computes the *net traction force* and the *net normal force* that are required to evaluate f_2 and f_3.

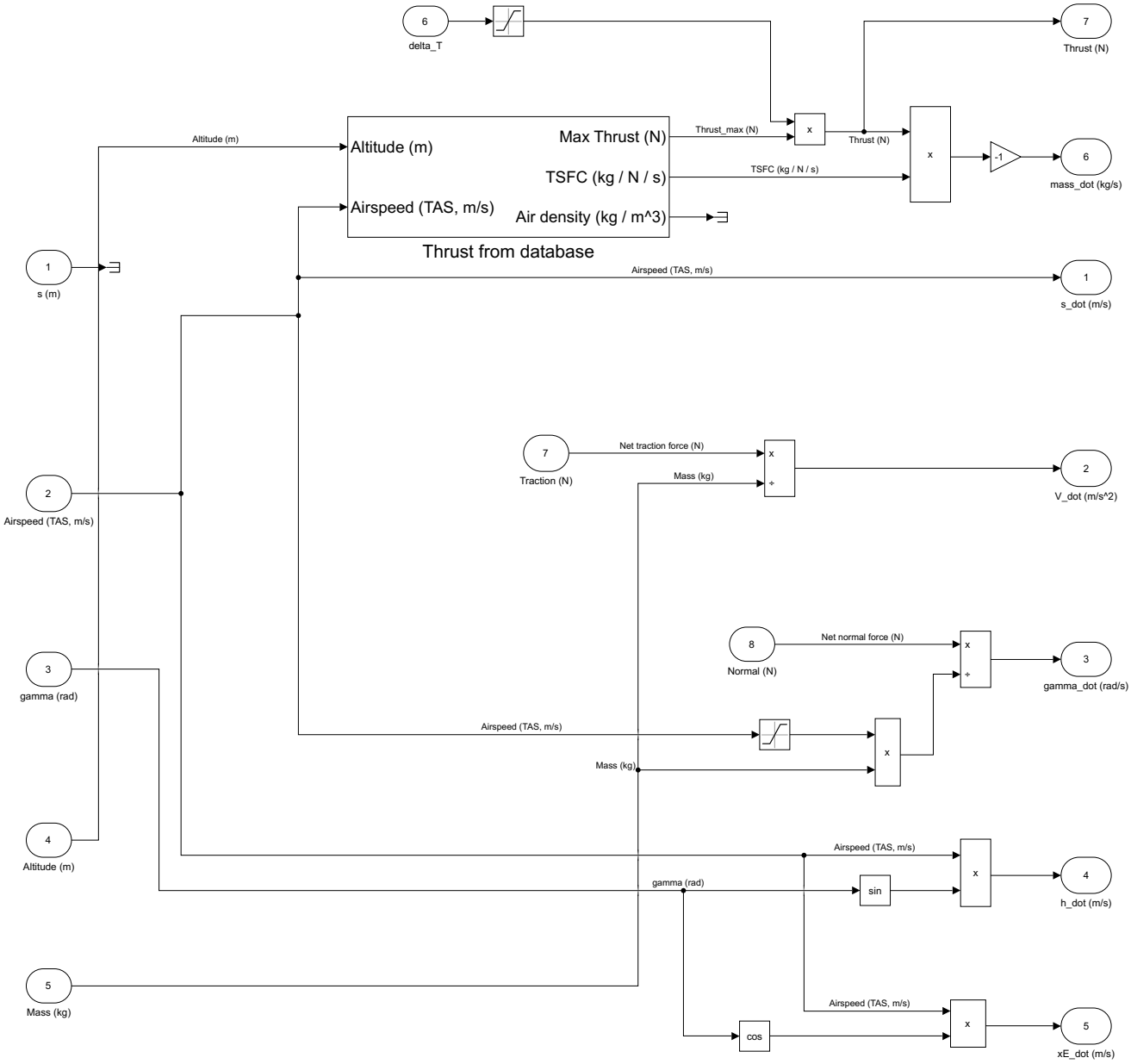


Figure 4.9 Subsystem Aircraft dynamics. It evaluates all the components of the right hand side function of the system f_1, f_2, f_3, f_4 and f_5.

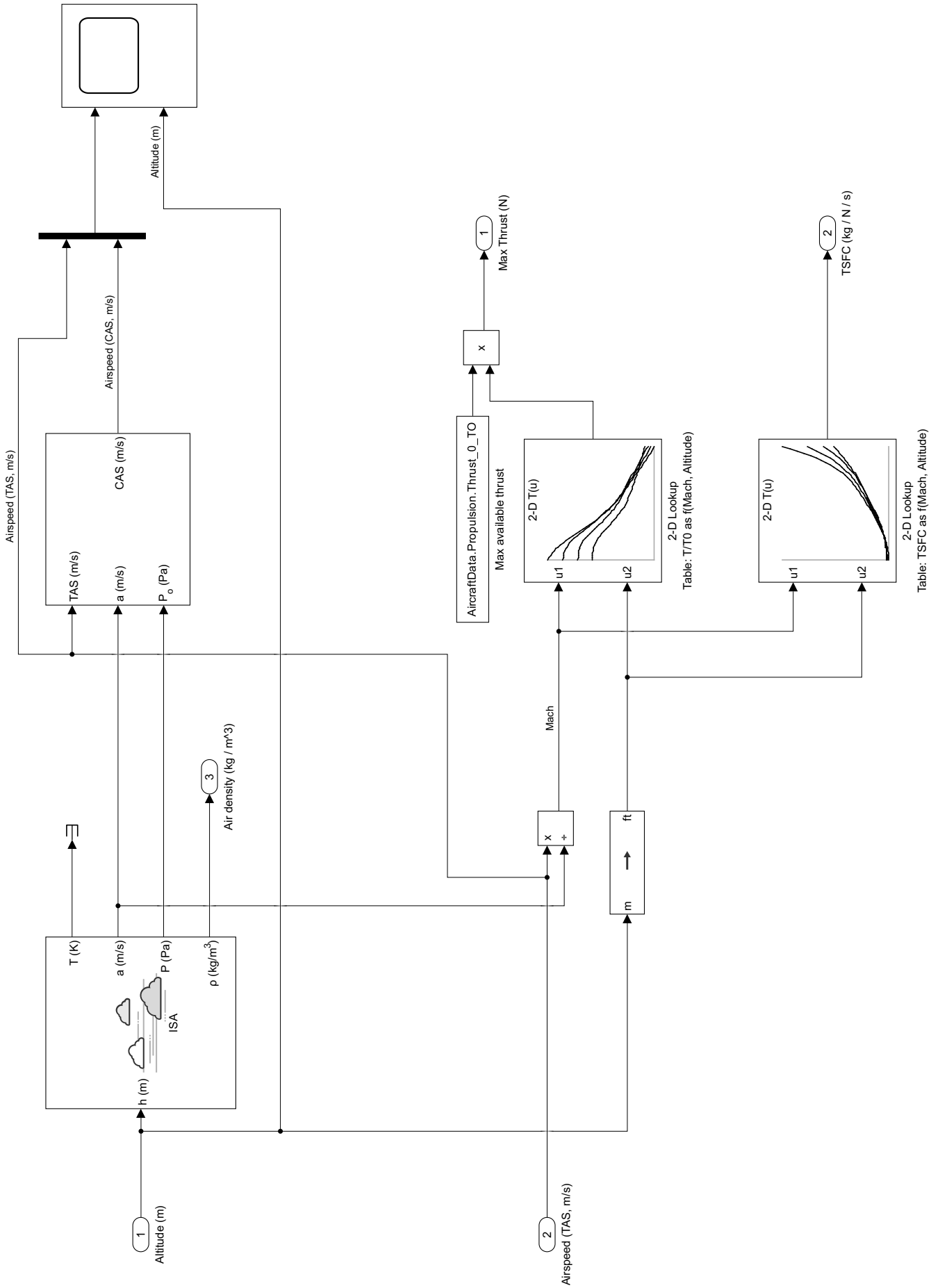


Figure 4.10 Subsystem Thrust from database. It outputs the maximum available thrust $T_{\max}$ and computes the thrust specific fuel consumption TSFC that is necessary to compute the mass flow rate $\dot{m} = -\dot{m}_{\mathrm{f}}$.

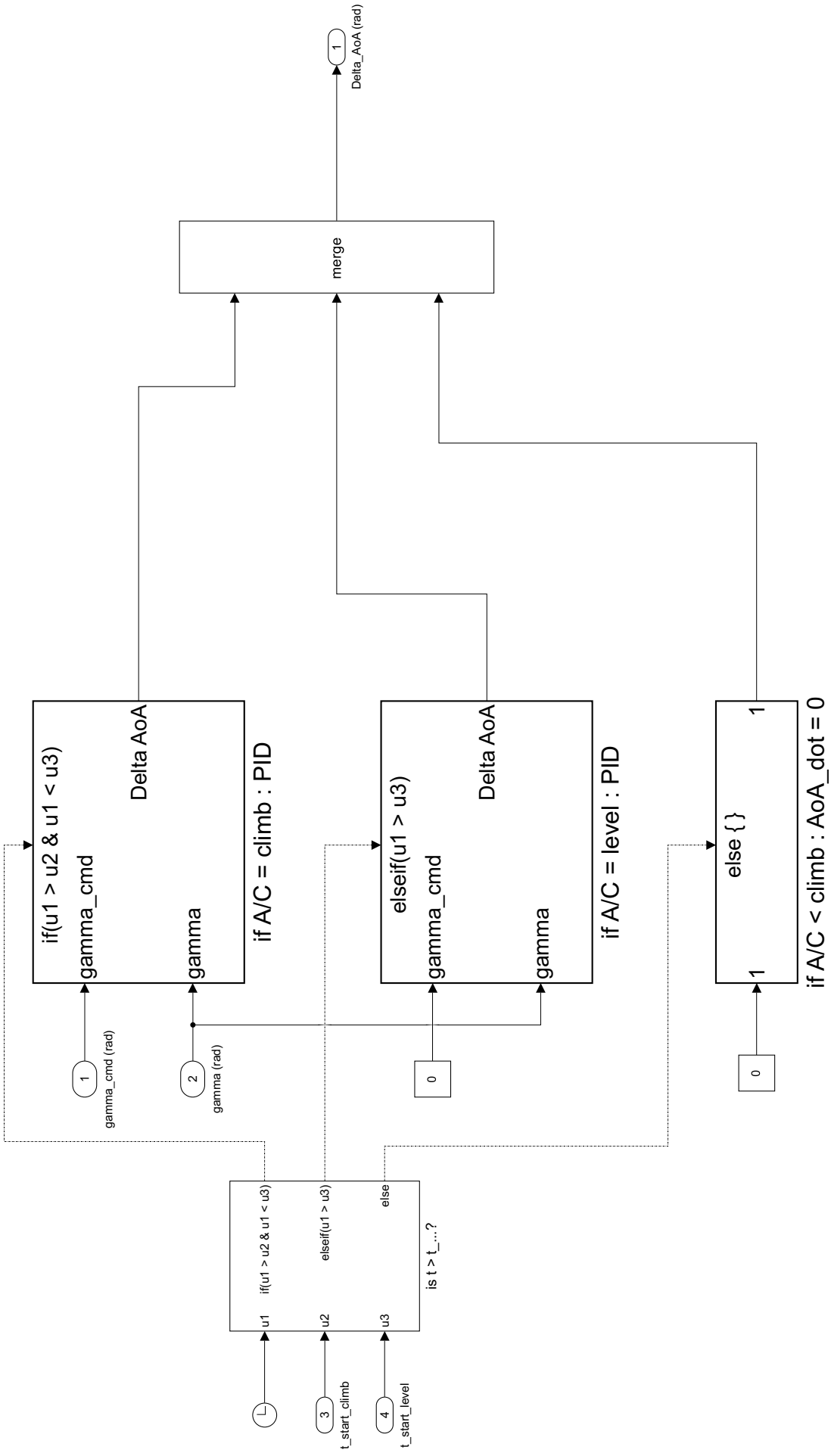


Figure 4.11 Subsystem Climb PID controller. It contains the PID controller responsible for the control law driven by the flight path angle feedback.

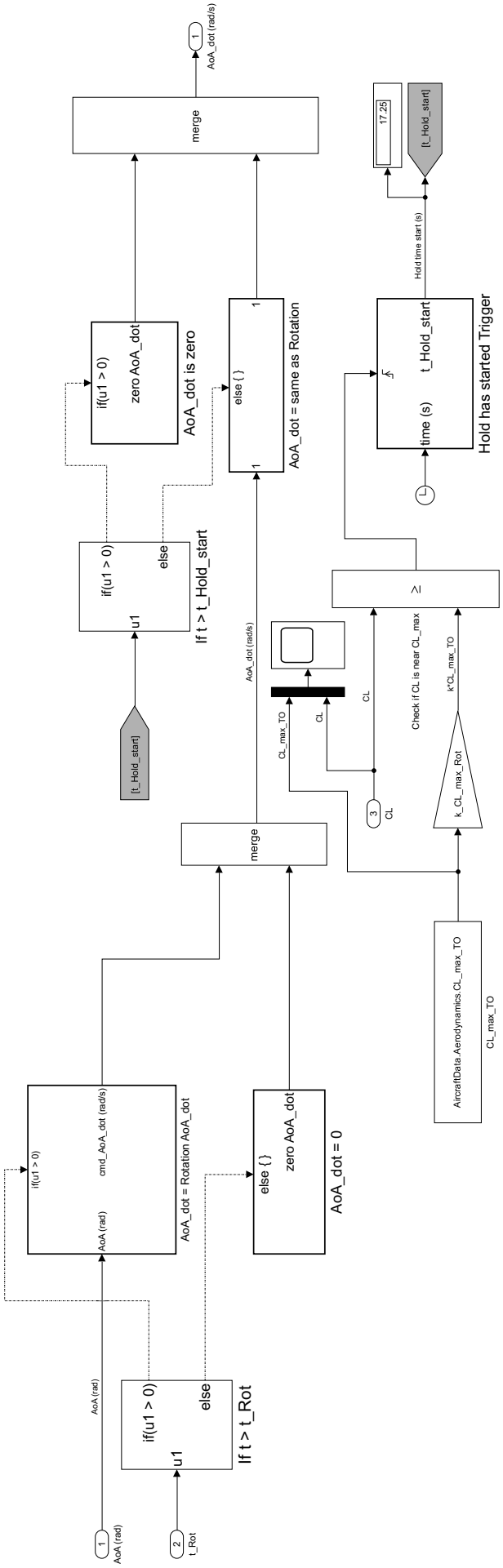


Figure 4.12 Subsystem Ground and Hold command. It outputs the angle of attack time law during ground phase and rotation phase as specified by equations (4.8) and (4.9).

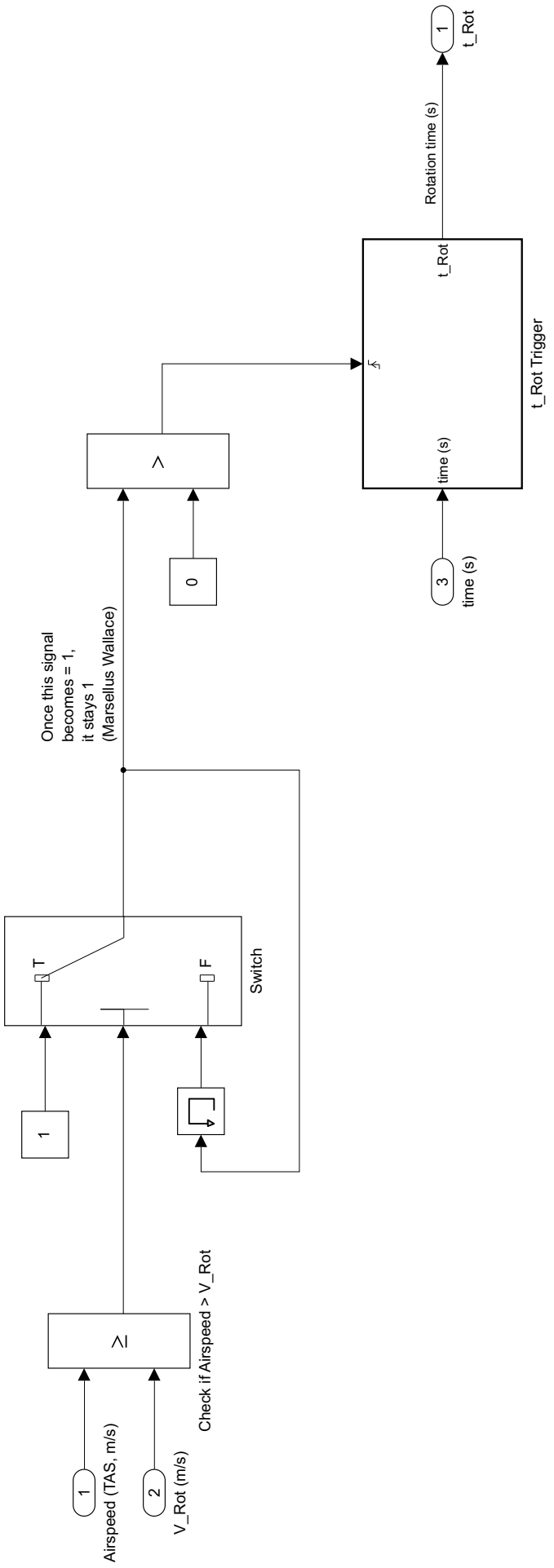


Figure 4.13 Subsystem Detect t_Rot. It detects when the rotation phase can start.

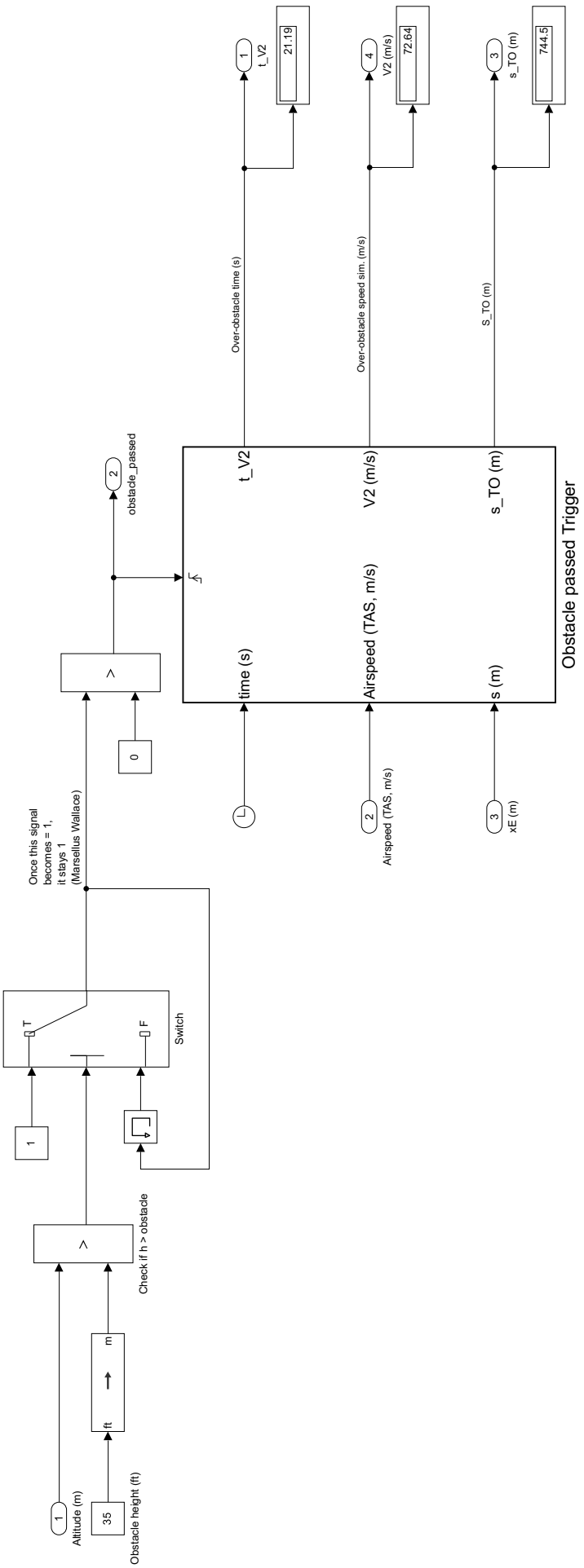


Figure 4.14 Subsystem Passage over the obstacle. It detects when the take-off reaches the end by *passing over the obstacle* and outputs the simulated value of V_2 and s_{TO}. As standardized by the regulations here it is assumed $h_{obstacle} = 35$ ft.

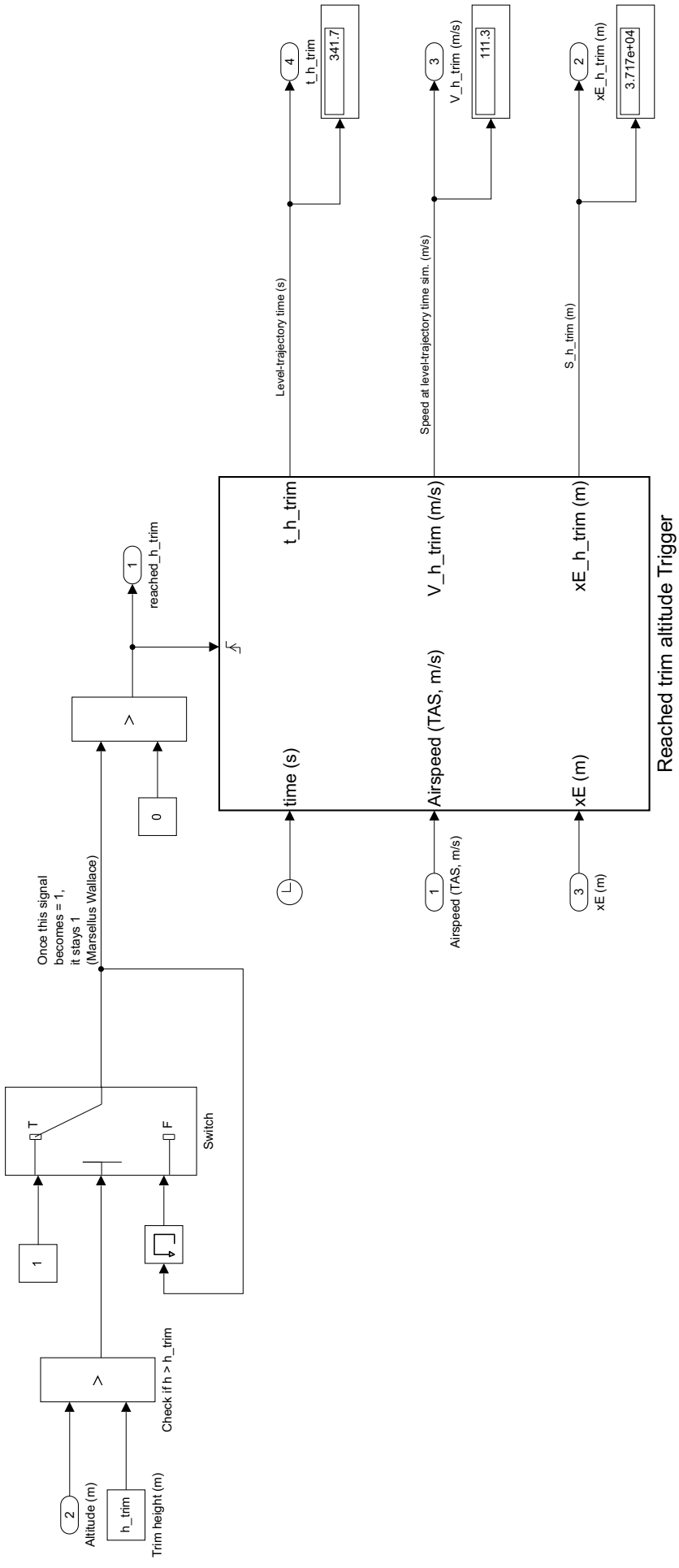


Figure 4.15 Subsystem Reaching the altitude to level trajectory. It detects when the commanded trim altitude is reached.

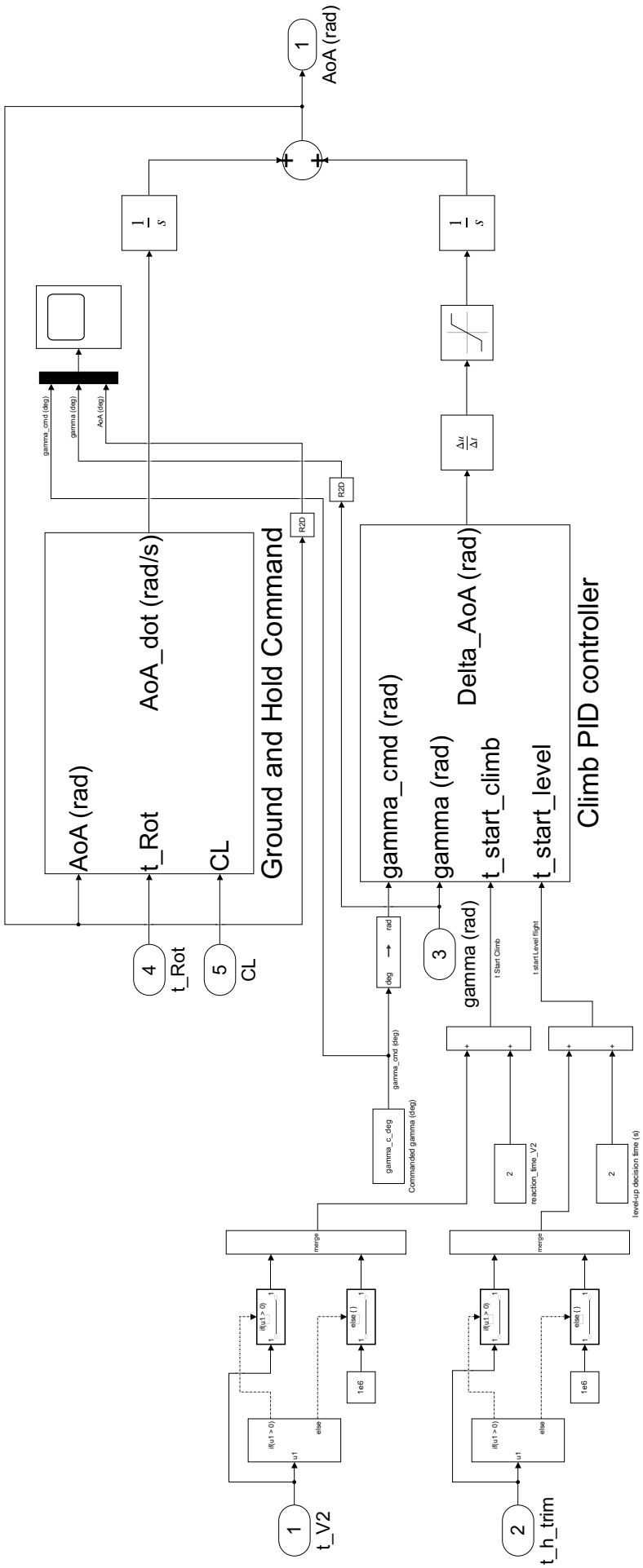


Figure 4.16 Subsystem AoA input. It outputs the angle of attack time law.

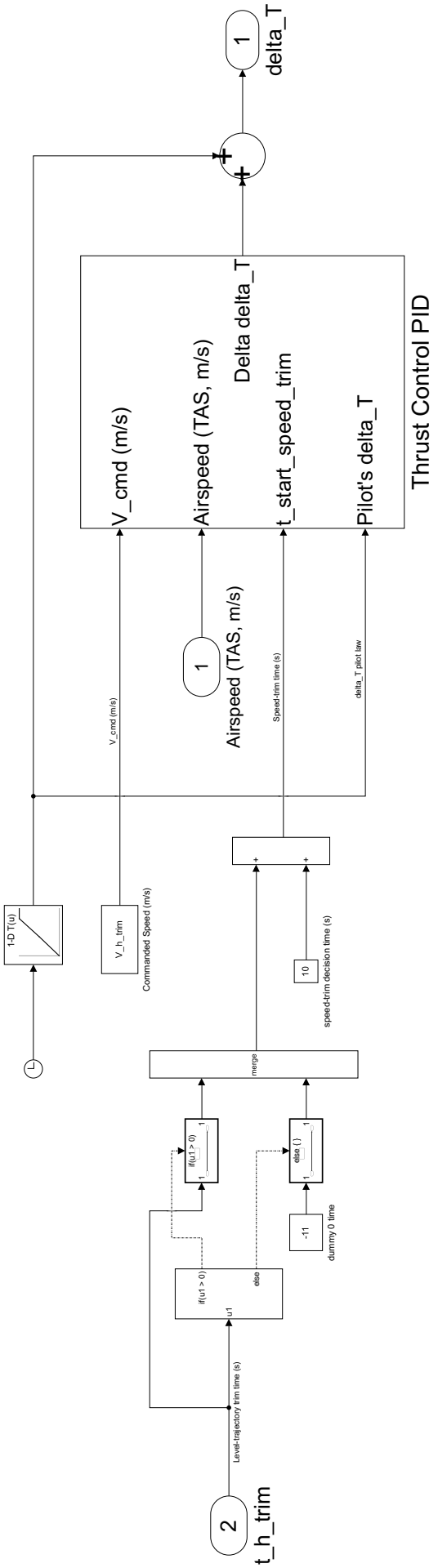


Figure 4.17 Subsystem delta_T input. It is responsible for the time law of the input variable δ_T.

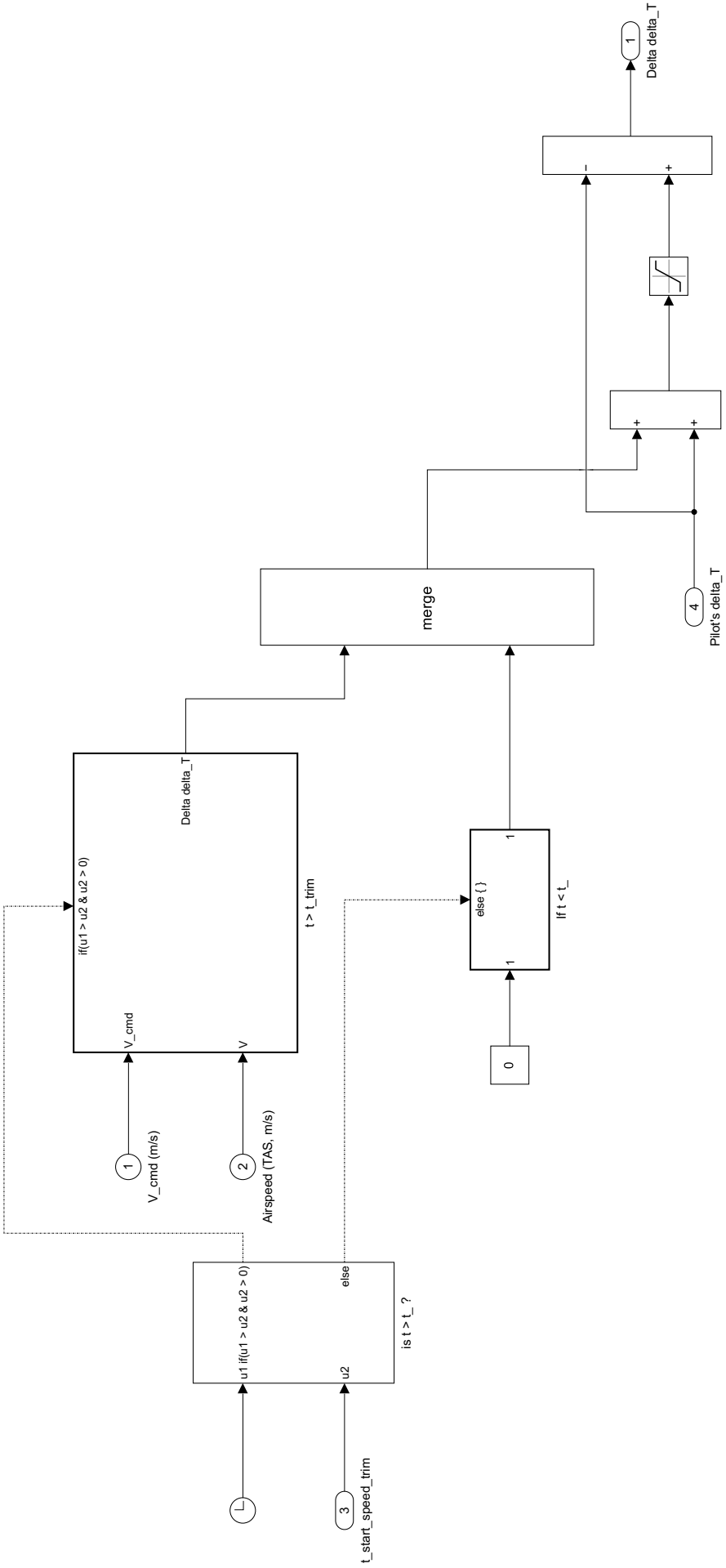


Figure 4.18 Subsystem Thrust control PID. It is responsible for the PID control law of δ_T.

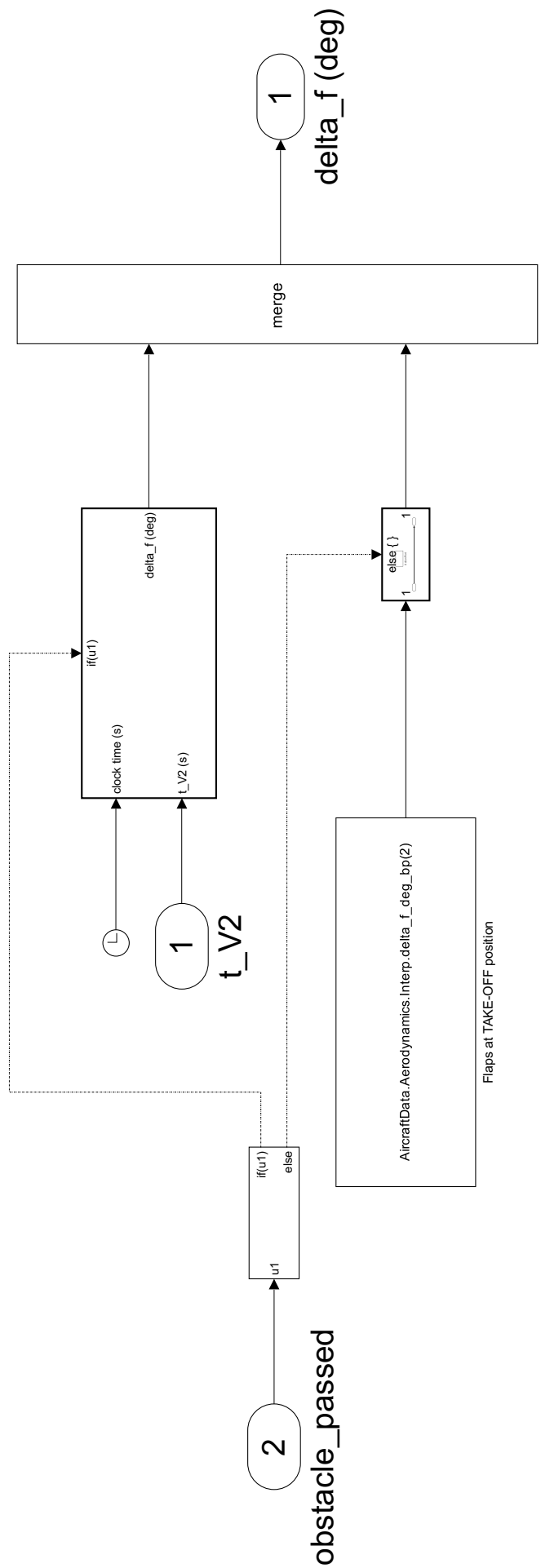


Figure 4.19 Subsystem delta_f input. It is responsible for the time law of the input variable δ_f.

To run the Simulink model few variables into the workspace are needed hence they must be assigned. It is firstly required a data structure containing the geometric, propulsion and aerodynamic data of the aircraft. It must be defined one time only before starting to simulate. Then, each time the Simulink model is run the following MATLAB script is executed.

It can be modified to assign different commanded variables V_{cmd}, h_{cmd} and γ_{cmd} or different parameters such that $\dot{\alpha}_0$, α_g, h_0 and so on.

Listing 4.1 Defines the data required by the simulation.

```
clear; close; clc

warning ('off','all')

load('AircraftData.mat')

alpha_g = convang(0.0,'deg','rad');
CL_g_TO = f_CL_TO(AircraftData, alpha_g);

CD_g_TO = ...
    AircraftData.Aerodynamics.CD0 ...
    + AircraftData.Aerodynamics.delta_CD0_TO ...
    + AircraftData.Aerodynamics.K_TO * ...
                                AircraftData.Aerodynamics.K_GE *
    ↪ CL_g_TO^2;

h_0 = 0;

[~, a_ss_0, ~, rho_0] = atmosisa(h_0);

alpha_dot_0 = convangvel(7.0, 'deg/s', 'rad/s');
dt_Hold = 0.75;
k_CL_max_Rot = 0.8;
k_alpha_Rot = 0.04;
alpha_dot_Red = -0.6 * alpha_dot_0;

V_2 = 1.20 * AircraftData.V_s_TO;
Mach_2 = V_2 / a_ss_0;
Thrust_2 = AircraftData.Propulsion.Interp.f_Thrust_TP_TO(h_0, Mach_2);
n_air = 1.22;
CL_V2 = 0.8 * AircraftData.Aerodynamics.CL_max_TO;
CD_V2 = AircraftData.Aerodynamics.CD0 ...
                + AircraftData.Aerodynamics.delta_CD0_TO ...
                + AircraftData.Aerodynamics.K_TO * CL_V2^2;

Drag_V2 = 0.5 * rho_0*(V_2^2) * AircraftData.S * CD_V2;

gamma_V2 = asin((Thrust_2 - Drag_V2) / AircraftData.W_0);

fun_AoA_from_CL_V2 = @(alpha) f_CL_TO(AircraftData, alpha) - CL_V2;

alpha_V2 = fsolve( ...
    fun_AoA_from_CL_V2, ...
    convang(2.0,'deg','rad') ...
    );

theta_V2 = gamma_V2 + alpha_V2;

gamma_c_deg = 7.0;

h_trim = 4500;
V_h_trim = 125;
```

The Simulink model execution and its outputs are then obtained by this other MATLAB script that saves the desired variables time histories into the workspace to then display them. Results are shown in figures 4.20 to 4.29.

Listing 4.2 Script for carrying out Simulink model simulations.

```
clear; close all; clc

modelName = 'exTakeOffATR42.slx';

simIn = Simulink.SimulationInput(modelName);

simIn = setModelParameter(simIn, 'Solver', 'ode45');

simOut = sim(modelName);

s = simOut.sim_s;
V = simOut.sim_V;
gamma = simOut.sim_gamma;
h = simOut.sim_h;
m = simOut.sim_m;
alpha = simOut.sim_alpha;
delta_T = simOut.sim_delta_T;
delta_f_deg = simOut.sim_delta_f;
xE = simOut.sim_xE;
m_dot = simOut.sim_m_dot;
h_dot = simOut.sim_h_dot;
T = simOut.sim_T;
tsfc = simOut.sim_tsfc;
Traction = simOut.sim_Traction;
Normal = simOut.sim_Normal;
CL = simOut.sim_CL;
CL_max = simOut.sim_CL_max;
CD = simOut.sim_CD;
L = simOut.sim_L;
D = simOut.sim_D;
gamma_c = simOut.sim_gamma_c;
h_c = simOut.sim_h_c;
V_c = simOut.sim_V_c;
T_max = simOut.sim_T_max;
t_Rot = simOut.sim_t_Rot;
t_LO = simOut.sim_t_LO;
t_V2 = simOut.sim_t_V2;
t_h_trim = simOut.sim_t_h_trim;
V_Rot = simOut.sim_V_Rot;
V_LO = simOut.sim_V_LO;
V2 = simOut.sim_V2;
s_g = simOut.sim_s_g;
s_TO = simOut.sim_s_TO;
xE_h_trim = simOut.sim_xE_h_trim;

t_end = s.Time(end);
V_c_new.Time = [0, t_end];
V_c_new.Data = [V_c.Data(1), V_c.Data(1)];
h_c_new.Time = [0, t_end];
h_c_new.Data = [h_c.Data(1), h_c.Data(1)];
gamma_c_new.Time = [0, t_end];
gamma_c_new.Data = [gamma_c.Data(1), gamma_c.Data(1)];
V_Rot_new.Time = [0, t_end];
V_Rot_new.Data = [V_Rot.Data(1), V_Rot.Data(1)];
CL_max_new.Time = [0, t_end];
CL_max_new.Data = [CL_max.Data(1), CL_max.Data(1)];

simVarPlot('exTOtime.pdf', 'Events time (s)', ...
    {'Rotation', 'Lift-Off', 'V_{2} reached'}, ...
    t_Rot, t_LO, t_V2)

simVarPlot('exTOspace.pdf', 'Events space (m)', ...
    {'s_{ground} (m)', 's_{TO} (m)'}, ...
    s_g, s_TO)

simVarPlot('exTOspeed.pdf', 'Airspeed (m/s)', ...
    {'TAS (m/s)', 'V_{Rot} (m/s)', 'V_{LO} (m/s)', ...
    'V_{2} (m/s)', 'V_{cmd} (m/s)', 'southeast'}, ...
    V, V_Rot_new, V_LO, V2, V_c_new)

```

```
gamma_deg = gamma;
gamma_deg.Data = convang(gamma_deg.Data, 'rad', 'deg');
gamma_c_deg = gamma_c_new;
alpha_deg = alpha;
alpha_deg.Data = convang(alpha_deg.Data, 'rad', 'deg');
simVarPlot('exTOangles.pdf', 'Angles (deg)', ...
    {'\gamma (deg)', '\gamma_{cmd} (deg)', '\alpha (deg)'}, ...
    gamma_deg, gamma_c_deg, alpha_deg)

simVarPlot('exTOaltitude.pdf', 'Altitude (m)', ...
    {'h_{cmd} (m)'}, h_c_new, h)

simVarPlot('exTOthrottle.pdf', '\delta_{T}', {}, delta_T)

simVarPlot('exTOflap.pdf', '\delta_{f} (deg)', {}, delta_f_deg)

simVarPlot('exTOmdot.pdf', 'Mass flow rate (kg/s)', {}, m_dot)

simVarPlot('exTOhdot.pdf', 'Rate of climb (m/s)', {}, h_dot)

simVarPlot('exTOtraction.pdf', 'Forces (N)', ...
    {'Thrust (N)', 'T_{max} (N)', 'Drag (N)', 'Net traction force (N)'}, ...
    T, T_max, D, Traction)

simVarPlot('exTOtsfc.pdf', 'TSFC (kg/N/s)', {}, tsfc)

W = m;
W.Data = 9.81 * W.Data;
simVarPlot('exTOnormal.pdf', 'Forces (N)', ...
    {'Lift (N)', 'Weight (N)', 'Net normal force (N)'}, ...
    L, W, Normal)

simVarPlot('exTOCL.pdf', 'C_{L}', {'C_{L,max}'}, CL_max_new, CL)

simVarPlot('exTOCD.pdf', 'C_{D}', {}, CD)

modelName = 'exTakeOffATR42_PID.slx';

simIn = Simulink.SimulationInput(modelName);

simIn = setModelParameter(simIn, 'Solver', 'ode45');

simOut = sim(modelName);

s = simOut.sim_s;
V = simOut.sim_V;
gamma = simOut.sim_gamma;
h = simOut.sim_h;
m = simOut.sim_m;
alpha = simOut.sim_alpha;
delta_T = simOut.sim_delta_T;
delta_f_deg = simOut.sim_delta_f;
xE = simOut.sim_xE;
m_dot = simOut.sim_m_dot;
h_dot = simOut.sim_h_dot;
T = simOut.sim_T;
tsfc = simOut.sim_tsfc;
Traction = simOut.sim_Traction;
Normal = simOut.sim_Normal;
CL = simOut.sim_CL;
CL_max = simOut.sim_CL_max;
CD = simOut.sim_CD;
L = simOut.sim_L;
D = simOut.sim_D;
gamma_c = simOut.sim_gamma_c;
h_c = simOut.sim_h_c;
V_c = simOut.sim_V_c;
T_max = simOut.sim_T_max;
t_Rot = simOut.sim_t_Rot;
t_LO = simOut.sim_t_LO;
t_V2 = simOut.sim_t_V2;
```

```
t_h_trim = simOut.sim_t_h_trim;
V_Rot = simOut.sim_V_Rot;
V_LO = simOut.sim_V_LO;
V2 = simOut.sim_V2;
s_g = simOut.sim_s_g;
s_TO = simOut.sim_s_TO;
xE_h_trim = simOut.sim_xE_h_trim;

t_end = s.Time(end);
V_c_new.Time = [0, t_end];
V_c_new.Data = [V_c.Data(1), V_c.Data(1)];
h_c_new.Time = [0, t_end];
h_c_new.Data = [h_c.Data(1), h_c.Data(1)];
gamma_c_new.Time = [0, t_end];
gamma_c_new.Data = [gamma_c.Data(1), gamma_c.Data(1)];
V_Rot_new.Time = [0, t_end];
V_Rot_new.Data = [V_Rot.Data(1), V_Rot.Data(1)];
CL_max_new.Time = [0, t_end];
CL_max_new.Data = [CL_max.Data(1), CL_max.Data(1)];

simVarPlot('exTOtimePID.pdf', 'Events time (s)', ...
    {'Rotation', 'Lift-Off', 'V_{2} reached'}, ...
    t_Rot, t_LO, t_V2)

simVarPlot('exTOspacePID.pdf', 'Events space (m)', ...
    {'s_{ground} (m)', 's_{TO} (m)'}, ...
    s_g, s_TO)

simVarPlot('exTOspeedPID.pdf', 'Airspeed (m/s)', ...
    {'TAS (m/s)', 'V_{Rot} (m/s)', 'V_{LO} (m/s)', ...
    'V_{2} (m/s)', 'V_{cmd} (m/s)', 'southeast'}, ...
    V, V_Rot_new, V_LO, V2, V_c_new)

gamma_deg = gamma;
gamma_deg.Data = convang(gamma_deg.Data, 'rad', 'deg');
gamma_c_deg = gamma_c_new;
alpha_deg = alpha;
alpha_deg.Data = convang(alpha_deg.Data, 'rad', 'deg');
simVarPlot('exTOanglesPID.pdf', 'Angles (deg)', ...
    {'\gamma (deg)', '\gamma_{cmd} (deg)', '\alpha (deg)'}, ...
    gamma_deg, gamma_c_deg, alpha_deg)

simVarPlot('exTOaltitudePID.pdf', 'Altitude (m)', ...
    {'h_{cmd} (m)'}, h_c_new, h)

simVarPlot('exTOthrottlePID.pdf', '\delta_{T}', {}, delta_T)

simVarPlot('exTOflapPID.pdf', '\delta_{f} (deg)', {}, delta_f_deg)

simVarPlot('exTOmdotPID.pdf', 'Mass flow rate (kg/s)', {}, m_dot)

simVarPlot('exTOhdotPID.pdf', 'Rate of climb (m/s)', {}, h_dot)

simVarPlot('exTOtractionPID.pdf', 'Forces (N)', ...
    {'Thrust (N)', 'T_{max} (N)', 'Drag (N)', 'Net traction force (N)'}, ...
    T, T_max, D, Traction)

simVarPlot('exTOtsfcPID.pdf', 'TSFC (kg/N/s)', {}, tsfc)

W = m;
W.Data = 9.81 * W.Data;
simVarPlot('exTOnormalPID.pdf', 'Forces (N)', ...
    {'Lift (N)', 'Weight (N)', 'Net normal force (N)'}, ...
    L, W, Normal)

simVarPlot('exTOCLPID.pdf', 'C_{L}', {'C_{L,max}'}, CL_max_new, CL)

simVarPlot('exTOCDPID.pdf', 'C_{D}', {}, CD)
```

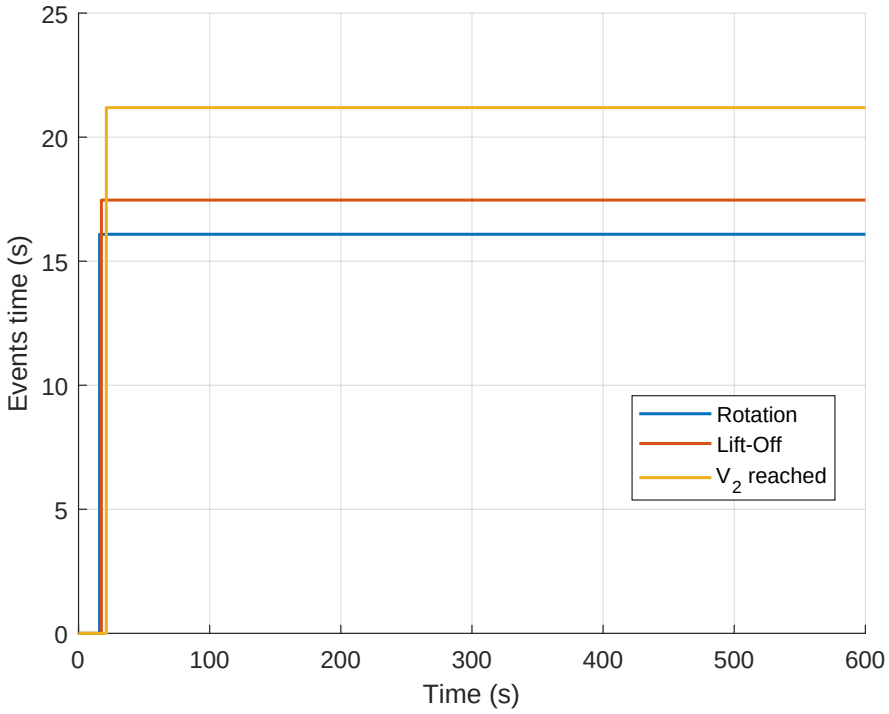


Figure 4.20 Time at which the rotation phase, the lift-off and the passage over the obstacle are reached.

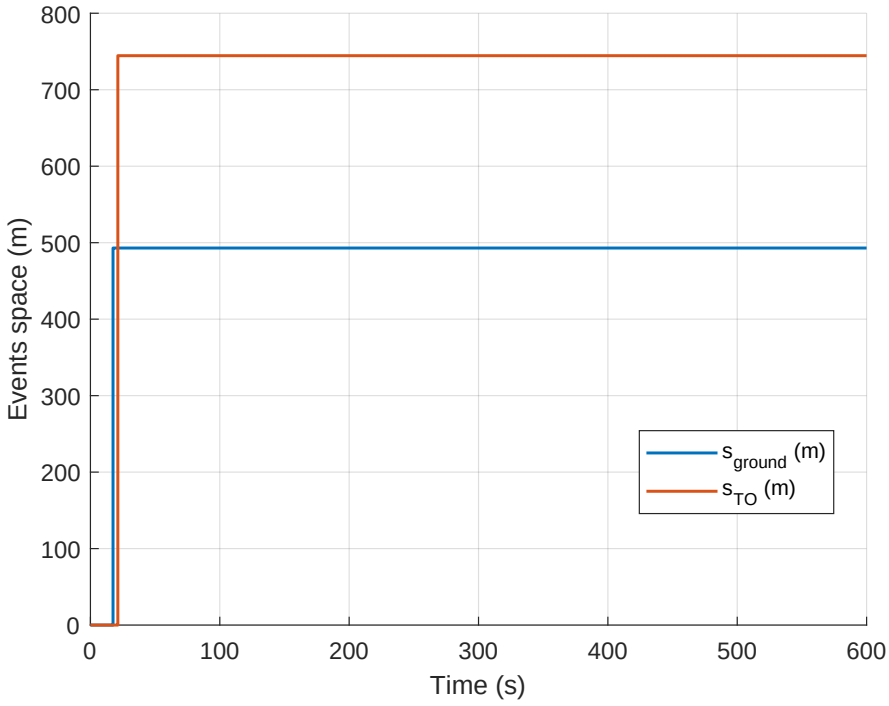


Figure 4.21 Space required to take-off.

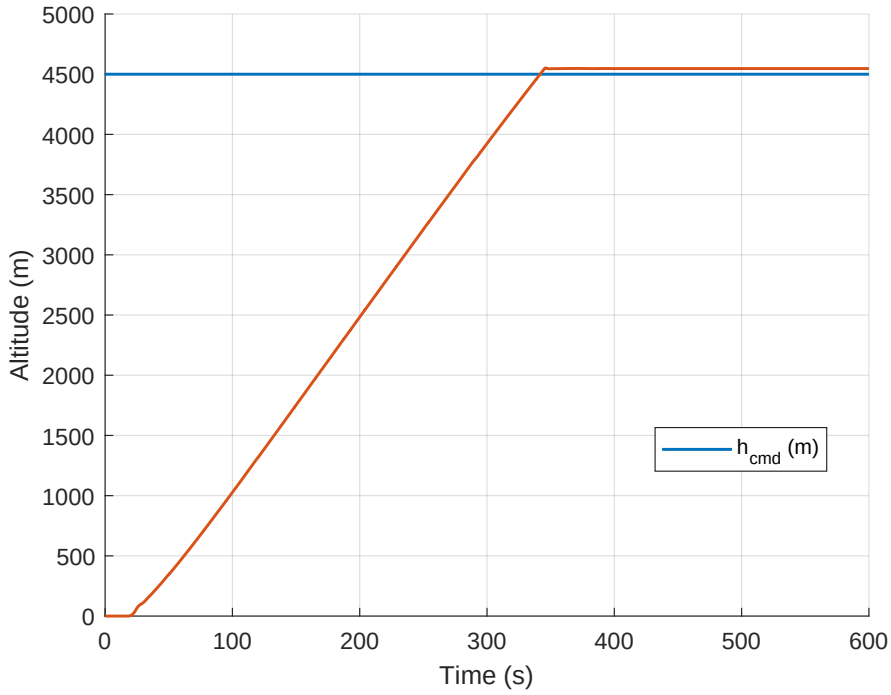


Figure 4.22 Simulated altitude. Observe how the actual altitude is almost equal to the commanded altitude h_{cmd}.

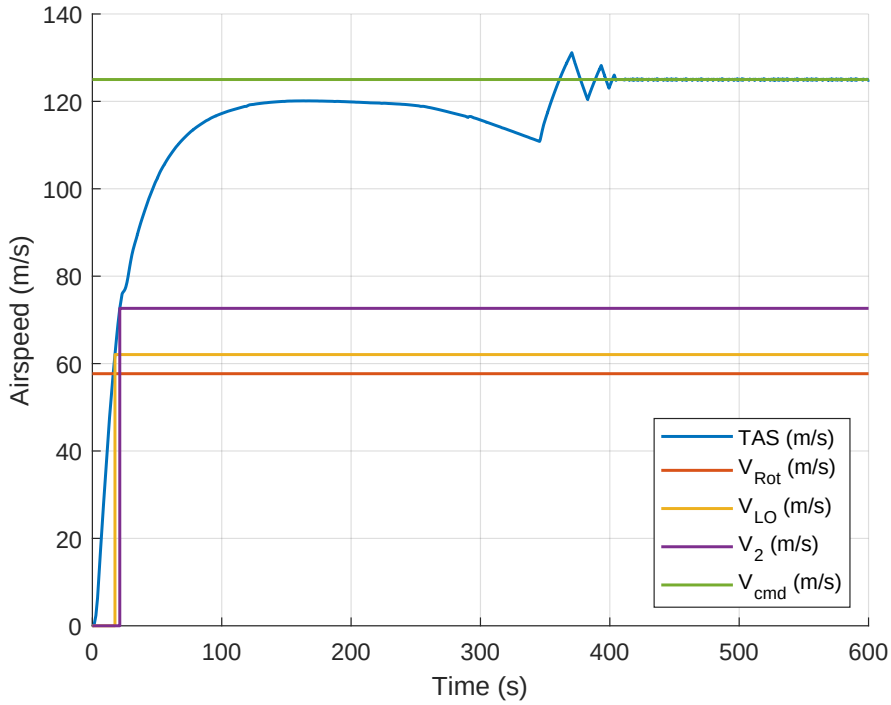


Figure 4.23 Simulated airspeed. Observe how the commanded airspeed V_{cmd} is reached successfully.

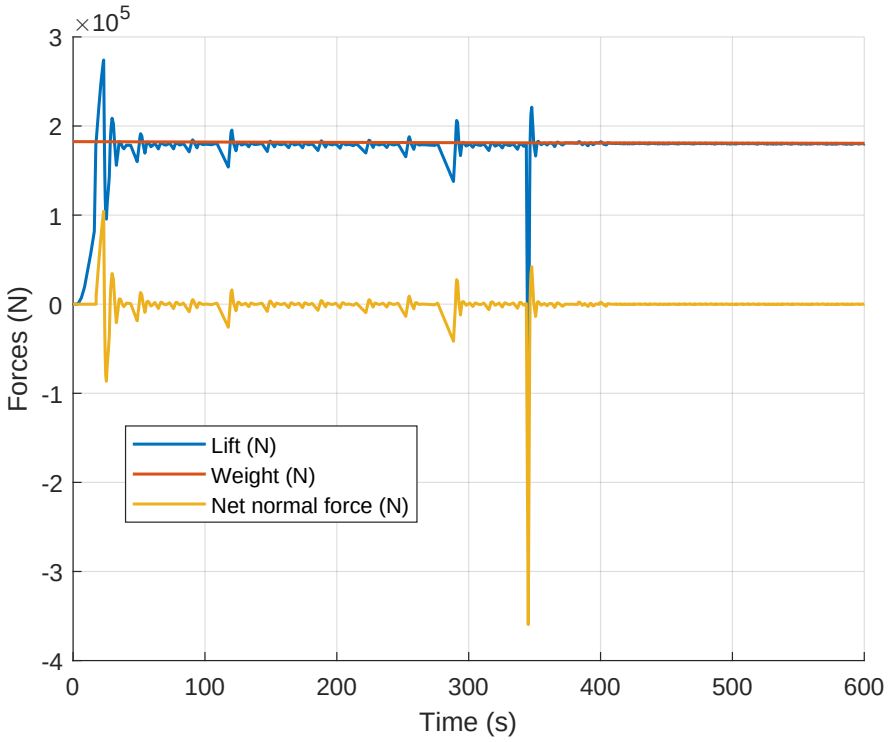


Figure 4.24 Simulated forces in the normal direction with respect to the aircraft wing plane.

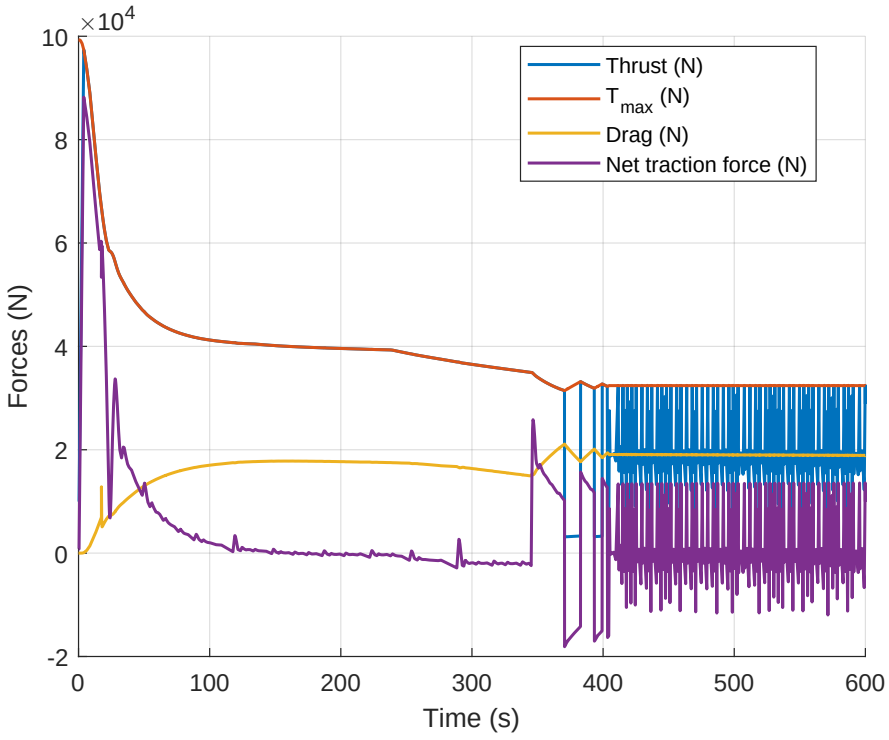


Figure 4.25 Simulated forces in the longitudinal direction with respect to the aircraft fuselage reference line. Observe the oscillation in the forces caused by the controlled throttle.

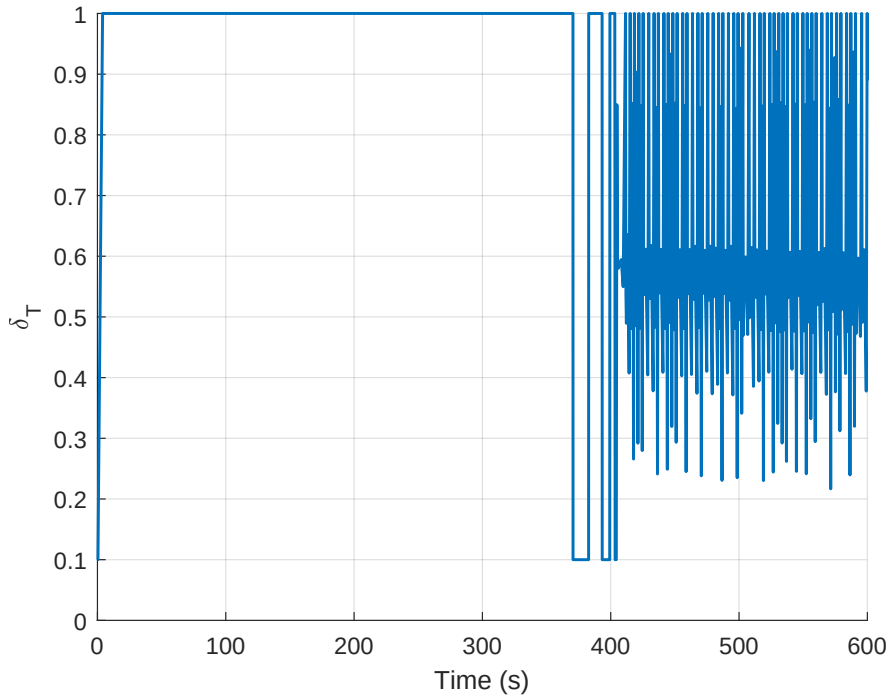


Figure 4.26 Throttle δ_T time law. Observe how the PI controller used in this simulation causes very quick changes in the input variable.

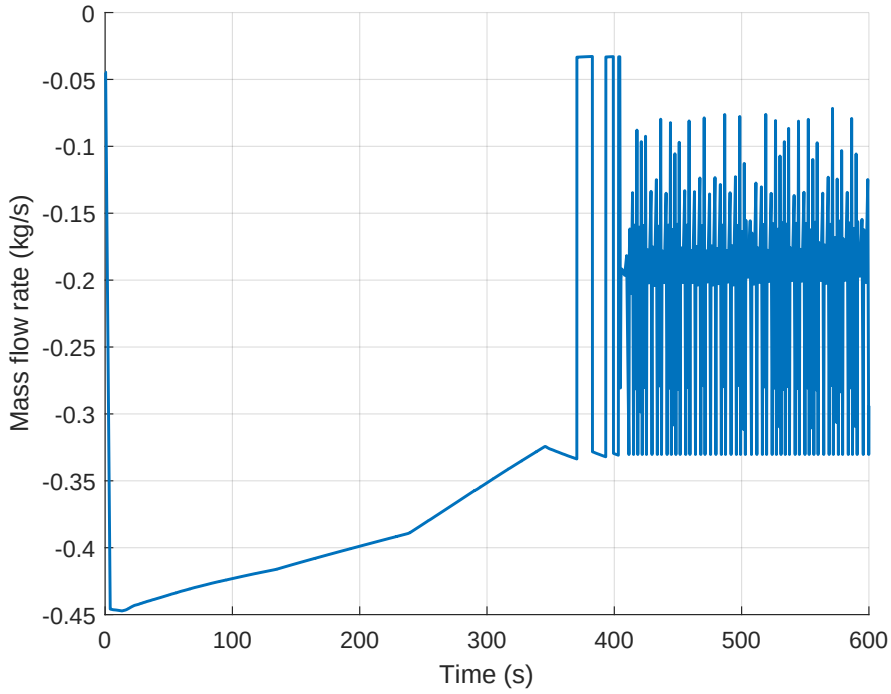


Figure 4.27 The same oscillation seen in the δ_T time law is of course visible also in the simulated mass flow rate $\dot{m}$.

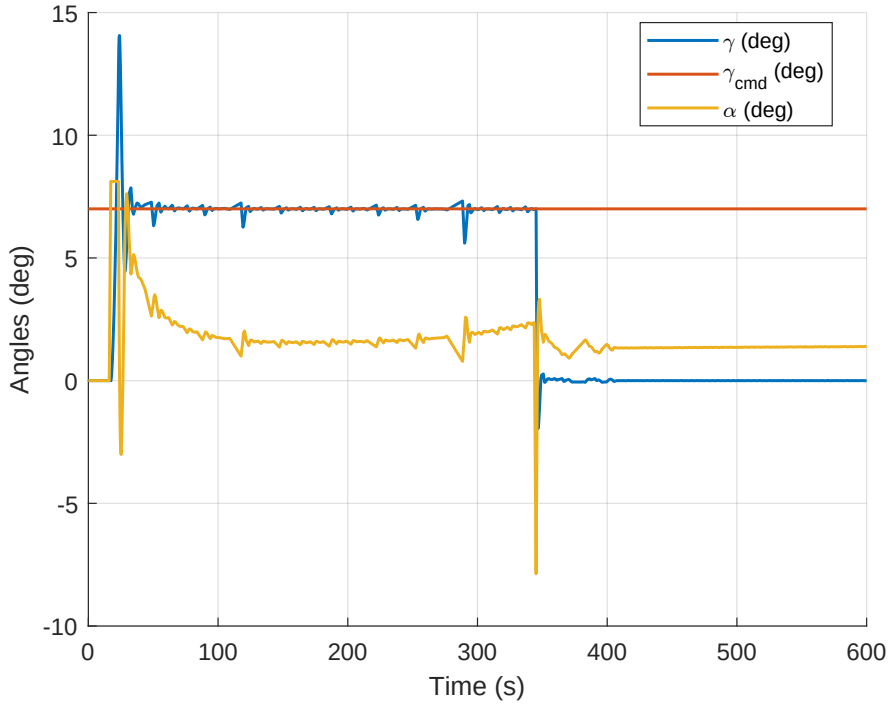


Figure 4.28 Simulated flight path angle γ and input variable α controlled time law. The commanded flight path angle γ_{cmd} is reached successfully.

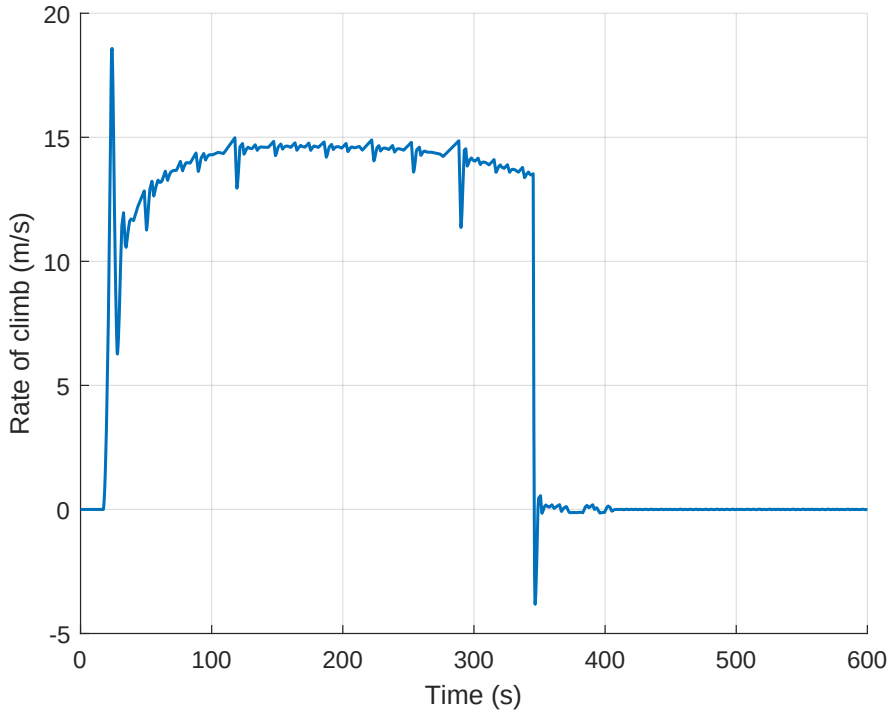


Figure 4.29 Simulated rate of climb $\dot{h}$.

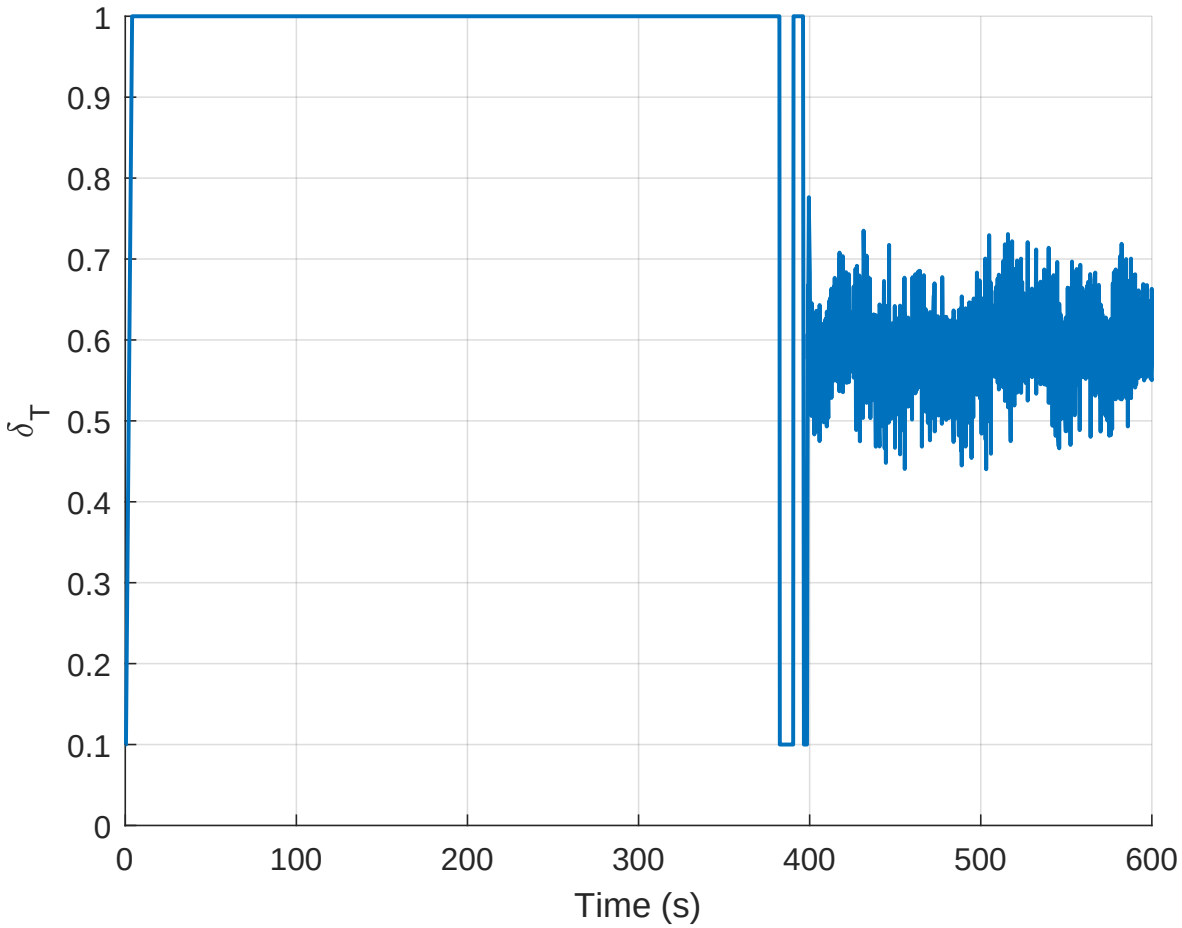


Figure 4.30 Input variable throttle δ_T PID-controlled time law.

How already pointed out, the PI controller used for the throttle control law causes those very quick oscillations. They can be damped by changing it with a PID controller, so changing its derivative gain K_D from 0 to a new value.

Another simulation has been carried out with PID gains $K_P = 1.8$, $K_I = 2.5$ and $K_D = 0.25$. This change mitigates the oscillations in the input variable δ_T as shown in figure 4.30 without changing the adherence to the requirements, i.e. commanded variables V_{cmd}, h_{cmd} and γ_{cmd}.

Bibliography

[1] G. H. Bryan. *Stability in Aviation: An Introduction to Dynamical Stability as Applied to the Motions of Airplanes*. Ed. by Macmillan and Co. Limited, London, 1911.

[2] Agostino De Marco et al. "A Simulation-Based Performance Analysis Tool for Aircraft Design Workflows". In: *Aerospace* 7 (Oct. 2020).

[3] William Rowan Hamilton. *Lectures on Quaternions: Containing a Systematic Statement of a New Mathematical Method; of which the Principles Were Communicated in 1843 to the Royal Irish Academy; and which Has Since Formed the Subject of Successive Courses of Lectures, Delivered in 1848 and Subsequent Years, in the Halls of Trinity College, Dublin: with Numerous Illustrative Diagrams, and with Some Geometrical and Physical Applications*. Hodges and Smith, 1853.

[4] Iliff Kenneth W. and Wang Kon-Sheng Charles. *Flight-Determined Subsonic Longitudinal Stability and Control Derivatives of the F-18 High Angle of Attack Research Vehicle (HARV) with Thrust Vectoring*. Tech. rep. National Aeronautics and Space Administration, 1997.

[5] Rodrigues. "Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire." fre. In: *Journal de Mathématiques Pures et Appliquées* (1840), pp. 380–440.

[6] Robert F. Stengel. *Fligth Dynamics*. Ed. by Princeton University Press. Princeton, 2004.

List of symbols

$O_{m\times n}$, $[O]_{m\times n}$ m by n zero matrix.

$\mathbb{1}_n$, $[1]_n$ n by n unit matrix.

$\boldsymbol{a}\times\boldsymbol{b}$ vector product between $\boldsymbol{a}$ and $\boldsymbol{b}$.

$\dot{\square}$ time derivative of a variable.

$\mathscr{F}_{\square}$ Reference Frame.

$\{\square\}_{\square}$ components of a vector in a specified frame of reference.

$[\square]_{\square}$ matrix representation of a linear operator in a specified frame of reference.

$[\tilde{\square}]_{\square}$ matrix representation of the vector product operator $\square\times$ in a specified frame of reference.

$\square_{\mathrm{B}}$ relative to the aircraft Body reference frame.

$\square_{\mathrm{E}}$ relative to the Earth reference frame.

$\square_{\mathrm{A}}$ relative to the aircraft Aerodynamic reference frame.

g acceleration of gravity.

G aircraft center of gravity.

$\boldsymbol{W}$ aircraft weight vector.

I_{B} aircraft inertia tensor.

I_{ii} moment of inertia with respect to the i-th axis.

I_{ij} product of inertia.

A measure of the imbalance in the mass distribution.

ψ azimuth angle of body axis x_{B}.

First angle of Euler angles 321 tern (ψ, θ, φ) of the vehicle orientation with respect to fixed reference frame.

θ inclination angle on the horizontal direction of body axis x_{B}.

Second angle of Euler angles 321 tern (ψ, θ, φ) of the vehicle orientation with respect to fixed reference frame.

φ lateral inclination angle of the wings.

Third angle of Euler angles 321 tern (ψ, θ, φ) of the vehicle orientation with respect to fixed reference frame.

$\mathbb{q}$ aircraft orientation quaternion.

Adopting the *scalar-first* convention, with components $\left[q_z,\ q_x,\ q_y,\ q_z\right]^{\mathrm{T}}$.

$\boldsymbol{V}_G$ aircraft center of gravity velocity vector.

u velocity vector $\boldsymbol{V}_G$ component along x_{B} axis.

v velocity vector $\boldsymbol{V}_G$ component along y_{B} axis.

w velocity vector $\boldsymbol{V}_G$ component along z_{B} axis.

$\boldsymbol{\Omega}_{\mathrm{B}}$ aircraft body angular velocity vector.

p roll rate.

Angular velocity vector $(\boldsymbol{\Omega}_{\mathrm{B}})$ component along x_{B} axis.

q pitch rate.

Angular velocity vector $(\boldsymbol{\Omega}_{\mathrm{B}})$ component along y_{B} axis.

r yaw rate.

Angular velocity vector $(\boldsymbol{\Omega}_{\mathrm{B}})$ component along z_{B} axis.

α_{B} aircraft body angle of attack.

Angle between the wind longitudinal plane and the aircraft longitudinal axis x_{B}.

α aircraft body absolute angle of attack.

Angle between the wind longitudinal plane and the zero-lift line.

μ_x aircraft zero-lift angle of attack.

Angle between the zero-lift line and the aircraft longitudinal axis x_{B}.

β sideslip angle.

Angle between the aircraft velocity vector $\boldsymbol{V}_G$ and the projection of the aircraft longitudinal axis onto the wind longitudinal plane, which describes whether there is a lateral component of the aircraft velocity.

h aircraft height with respect to sea level (altitude).

M Mach number.

It is a dimensionless quantity in fluid dynamics representing the ratio of flow velocity past a boundary to the local speed of the sound. By definition, at Mach 1, the local flow velocity is equal to the speed of the sound.

S aerodynamic surface area.

$\bar{c}$ mean aerodynamic chord.

b aerodynamic surface span.

$\mathcal{AR}$ wing aspect ratio b^2/S.

e Oswald's factor.

T thrust force magnitude.

L lift.

D drag.

Y_{A} crossforce.

C_L aircraft lift coefficient.

C_D aircraft drag coefficient.

$C_{Y_{\mathrm{A}}}$ aircraft side force coefficient.

X resultant force component along x_{B} axis.

Y resultant force component along y_{B} axis.

Z resultant force component along z_{B} axis.

$\mathscr{L}$ aircraft roll moment.
Resultant moment component about the x_{B} axis.

$\mathscr{M}$ aircraft pitch moment.
Resultant moment component about the y_{B} axis.

$\mathscr{N}$ aircraft yaw moment.
Resultant moment component about the z_{B} axis.

$C_{\mathscr{L}}$ aircraft aerodynamic roll moment coefficient.

$C_{\mathscr{M}}$ aircraft aerodynamic pitch moment coefficient.

$C_{\mathscr{N}}$ aircraft aerodynamic yaw moment coefficient.

δ_{T} engine throttle.

δ_{e} elevator deflection angle.

δ_{s} stabilator deflection angle.

δ_{a} ailerons deflection angle.

δ_{r} rudder deflection angle.

δ_{t} elevator trim tab deflection angle.

δ_{f} flaps deflection angle.