JSBSim Python Examples

Release 1.0

Agostino De Marco

Apr 10, 2026

DOCUMENTATION

1 Concepts **3**
- 1.1 Frames of reference . . . 3
- 1.2 Simulation . . . 11
- 1.3 Frames of reference . . . 12
- 1.4 Units . . . 12
- 1.5 Properties . . . 12
- 1.6 Math . . . 12
- 1.7 Forces and moments . . . 12
- 1.8 Flight Control and System Modelling . . . 12

2 JSBSim Hello World **13**
- 2.1 What we will do . . . 13
- 2.2 1. Import and version check . . . 13
- 2.3 2. Create the Flight Dynamics Model executor . . . 13
- 2.4 3. Load an aircraft model . . . 14
- 2.5 4. Set initial conditions . . . 14
- 2.6 5. Trim the aircraft . . . 14
- 2.7 6. Step the simulation and read properties . . . 15
- 2.8 7. Explore the property tree . . . 15
- 2.9 Summary . . . 16

3 JSBSim Flight Simulation **17**
- 3.1 Topics covered . . . 17
- 3.2 1. Setup . . . 17
- 3.3 2. Initialise JSBSim and trim . . . 18
- 3.4 3. Run the simulation and record data . . . 18
- 3.5 4. Visualise – altitude and airspeed . . . 19
- 3.6 5. Visualise – attitude angles . . . 20
- 3.7 6. Visualise – ground track . . . 21
- 3.8 7. Reset and re-run: 10-degree turn to the left . . . 22
- 3.9 Summary . . . 24

4 PathSim Introduction **25**
- 4.1 Install . . . 25

5 JSBSim Trim and Elevator Doublet with PathSim **33**
- 5.1 1. Imports . . . 33
- 5.2 2. Initialize the JSBSim FDM . . . 34
- 5.3 3. Aircraft Parameters and Trim Conditions . . . 34
- 5.4 4. Level-Flight Trim . . . 35
- 5.5 5. PathSim Block Diagram . . . 36
- 5.6 6. Simulation Results . . . 38
- 5.7 Summary . . . 39

6 pathsim-flight: JSBSim as a PathSim Block **41**

6.1 Install . . . 41

7 Angle of attack (AoA) vs Calibrated Airspeed (CAS) of a Global 5000 aircraft in wings-level flight 49
7.1 Plotting trim angles of attack vs flight speed . . . 50
7.2 Initialize FDM . . . 53
7.3 Tweak aircraft XML file: remove <output name="..."/> nodes from the officially released files 54
7.4 Different aircraft weights, CoG positions and altitudes . . . 60

8 Rudder Kick 69

9 Thrust Vectoring Analysis 73
9.1 Initialize FDM . . . 73
9.2 Tweak aircraft XML file: remove <input/> nodes from the officially released files . . . 74

10 Trim Envelope 79
10.1 Initialize FDM . . . 79
10.2 Tweak aircraft XML file: remove <input/> nodes from the officially released files . . . 79
10.3 Initialize the JSBSim executable . . . 80
10.4 Work on trim conditions . . . 81

11 Trim Envelope and Climb Analysis 89
11.1 Import Required Libraries . . . 89
11.2 Initialize JSBSim and Set Up Aircraft . . . 89
11.3 Define Envelope Limits and Parameters . . . 89
11.4 Generate Trim Envelope . . . 90
11.5 Plot Trim Envelope Results . . . 90
11.6 Define Flight Simulation Functions . . . 91
11.7 Execute Climb Simulations . . . 93

12 Resources 95
12.1 This documentation . . . 95
12.2 Related projects . . . 95
12.3 Useful references . . . 95

13 Quick start 97

14 Indices and tables 99

LIST OF FIGURES

1 Aircraft structural (or construction) frame of reference with origin O_C. Besides the structural frame axes x_C, y_C, and z_C, the standard body frame axes x_B, y_B, and z_B are also shown with their origin at the center of mass G. The pilot's eye-point is located at P_EP. 4

2 A screenshot taken from the 3D modeling software Blender. The scene shows a model of Cessna 172 with its structural frame $\mathcal{F}_\mathrm{C} = \{O_\mathrm{C}, x_\mathrm{C}, y_\mathrm{C}, z_\mathrm{C}\}$. The origin O_C in this case is located inside the cockpit, near the dashboard. 4

3 Center of gravity (CG) position, point G, determined in a construction frame. 5

4 Definition of ground contact points in terms of construction frame locations. 5

5 Two key point locations P_ARP and $P_\mathrm{CG,EW}$ in the structural frame, respectively, he pole of aerodynamic moments and the Empty Weight CG of the airframe. The shape of the wing root profile and its chord are also sketched. 6

6 Besides point $P_\mathrm{CG,EW}$, are represented two more significant locations, P_Pilot and $P_\mathrm{Right\,Pass}$, where two additional masses are concentrated, respectively, of the pilot and of the right passenger. 6

7 Standard aircraft body axis frame, with origin at the center of gravity G. 7

8 Aerodynamic frame, defining the aerodynamic angles α_B and β. 8

9 Banked lift in a steady coordinated turn at constant altitude. The bank angle ϕ_W is a rotation around the relative wind velocity vector. The motion is freezed in time when the velocity vector is aligned with the North. Coordinated turn means that $\beta = 0$ and constant altitude means that x_A is kept horizontal. 8

10 Earth-Centered Inertial (ECI) frame and Earth-Centered Earth-Fixed (ECEF) frame. 9

11 Earth-Centered Earth-Fixed (ECEF) frame, geografic coordinates, Tangent (T) frame, and local Vertical (V) frame. 9

12 Aircraft body frame and local vertical frame (NED frame). The aircraft Euler angles are also shown: the heading angle ψ (negative in the picture), the elevation angle θ, and the roll angle ϕ. 10

13 Euler angle sequence for an aircraft. The frame $\mathcal{F}_\mathrm{E} = \{O_\mathrm{E}, x_\mathrm{E}, y_\mathrm{E}, z_\mathrm{E}\}$ is an Earth-fixed NED coordinate system, with origin the O_E somewhere on the ground (or at see level) and the plane $x_\mathrm{E}y_\mathrm{E}$ tangent to the Earth surface. If the ground track point G_GT is not too far from O_E, the Earth frame $\mathcal{F}_\mathrm{E}$ axes are parallel to those of the local NED frame $\mathcal{F}_\mathrm{V} = \{G, x_\mathrm{V}, y_\mathrm{V}, z_\mathrm{V}\}$. 10

14 Standard frames of reference and aircraft in climbing flight in calm air. The CG velocity vector $\boldsymbol{V}$ forms the flight path angle γ with the horizontal plane. The standard three aerodynamic resultant force components D, L and Y_A are also shown. 11

JSBSim is a lightweight, data-driven, non-linear, six-degree-of-freedom (6DoF), batch simulation application aimed at modeling flight dynamics and control for aircraft. Since the earliest versions, JSBSim has benefited from the open source development environment it has grown within and from the wide variety of users that have contributed ideas for its continued improvement.

The JSBSim online reference manual is a community effort to keep the users and developers up-to-date with all the functionalities of the software. Yet the manual is not meant to be a tutorial or a user guide, and it does not include examples of how to use the software in practice. This repository acts as a side project to JSBSim and demonstrates real-world usage of the JSBSim Python API through fully executable Jupyter notebooks. Eventually the collection of notebooks will grow to cover a wide range of use cases, from basic API usage to advanced analysis and integration with other tools, and will be integrated into the online reference manual as a companion resource.

The notebooks in this repository are here to serve as an example-based user guide of JSBSim for Python users. For instance, this documentation serves as a reference for how to use the API for common tasks such as loading aircraft models, setting initial conditions, running flight simulations, and analyzing results.

The examples include demos of how JSBSim can be integrated with interesting Python libraries, such as PathSim, and pathsim-flight, and PathView.

The notebooks cover:

- **Basic API usage** – creating an FDM executor, loading aircraft models, setting initial conditions, trimming, and reading properties.
- **Flight simulation** – running trimmed flight, recording time histories, plotting altitude, airspeed, attitude, and ground track.
- **PathSim** – block-diagram simulation with integrators, amplifiers, adders, scopes, and custom function blocks.
- **pathsim-flight** – embedding JSBSim inside a PathSim block diagram, using the International Standard Atmosphere model, and building a simple pitch-hold autopilot.

CHAPTER

ONE

CONCEPTS

This is a work in progress!

The concepts documentation covers the fundamental ideas and principles behind JSBSim, such as the structure of aircraft models, the physics of flight simulation, and the design of the API. It will also include explanations of key concepts such as frames of reference, coordinate systems, and units of measurement.

1.1 Frames of reference

Before moving into a description of the configuration file syntax, one must understand some basic information about the frames of reference used:

- to describe locations of objects on the aircraft,
- to specify conditions related to aircraft position and orientation in space,
- to assign inputs for a given flight condition.

Reference sources: * JSBSim Reference Manual - Frames of reference * Wikipedia about axes conventions

1.1.1 Structural, or "Construction" Frame

This frame is the manufacturer-style reference used to define points on the aircraft: center of gravity, landing gear contact points, pilot eye-point, point masses, thrusters, and more. In JSBSim aircraft configuration files, many locations are specified in this frame.

In the structural frame, the X-axis runs along fuselage length and points toward the tail, the Y-axis points toward the right wing, and Z is positive upward. A common origin O_C is near the front of the aircraft. This frame is often denoted as $\mathcal{F}_\mathrm{C} = \{O_\mathrm{C}, x_\mathrm{C}, y_\mathrm{C}, z_\mathrm{C}\}$.

The x_C axis is often aligned with the fuselage centerline and frequently with thrust axis. Positions along the axes are commonly called stations (x_C), buttlines (y_C), and waterlines (z_C).

JSBSim mainly uses relative distances between points, so the absolute origin location is not critical if geometry is consistent.

1.1.2 Body Frame

In JSBSim, the body frame is equivalent to a 180-degree rotation of the construction frame around y_C, with origin at the center of mass G. The body frame is often written as $\mathcal{F}_\mathrm{B} = \{G, x_\mathrm{B}, y_\mathrm{B}, z_\mathrm{B}\}$.

- x_B points forward (roll axis),
- y_B points to the right wing (pitch axis),
- z_B points downward (yaw axis direction convention in body coordinates).

Forces and moments are summed in body axes and integrated to obtain translational and rotational states.

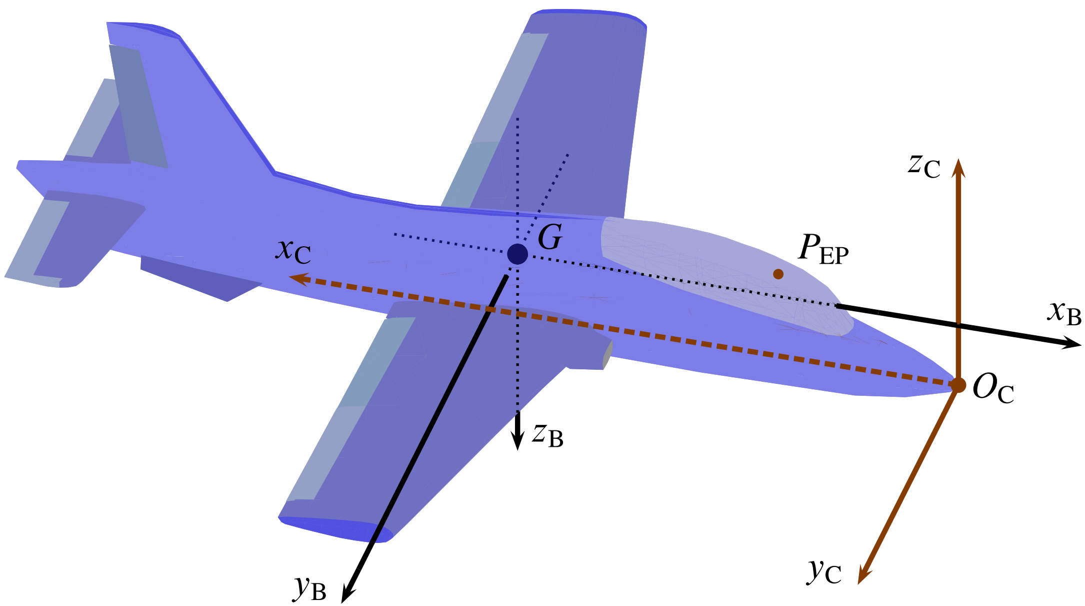


Fig. 1: Aircraft structural (or construction) frame of reference with origin O_C. Besides the structural frame axes x_C, y_C, and z_C, the standard body frame axes x_B, y_B, and z_B are also shown with their origin at the center of mass G. The pilot's eye-point is located at P_{EP}.

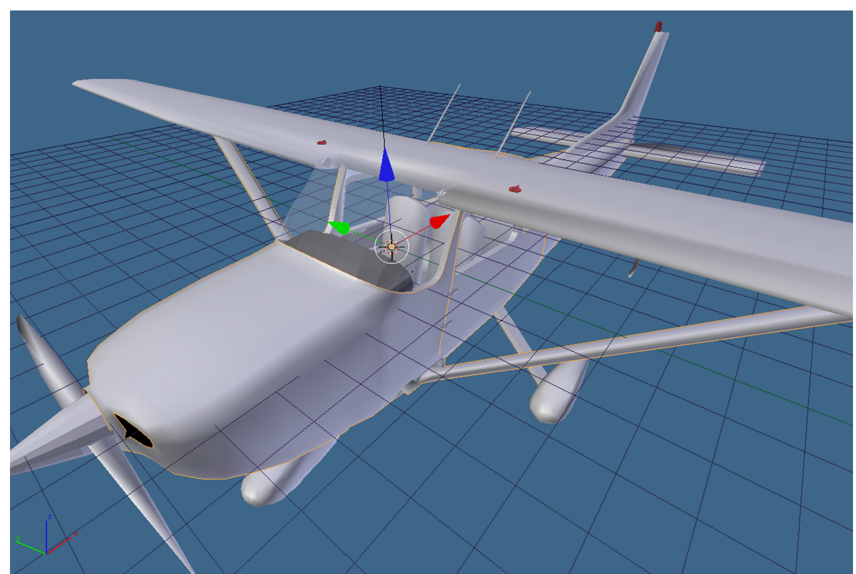


Fig. 2: A screenshot taken from the 3D modeling software Blender. The scene shows a model of Cessna 172 with its structural frame $\mathcal{F}_C = \{O_C, x_C, y_C, z_C\}$. The origin O_C in this case is located inside the cockpit, near the dashboard.

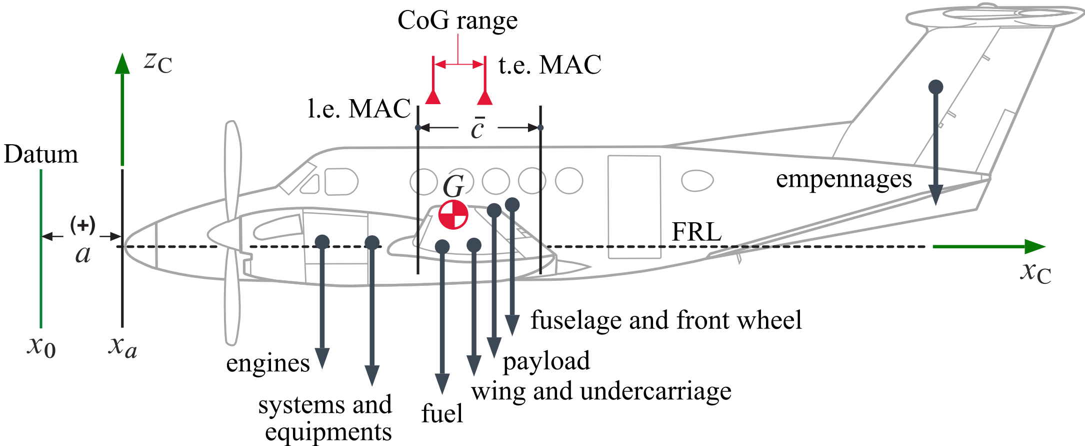


Fig. 3: Center of gravity (CG) position, point G, determined in a construction frame.

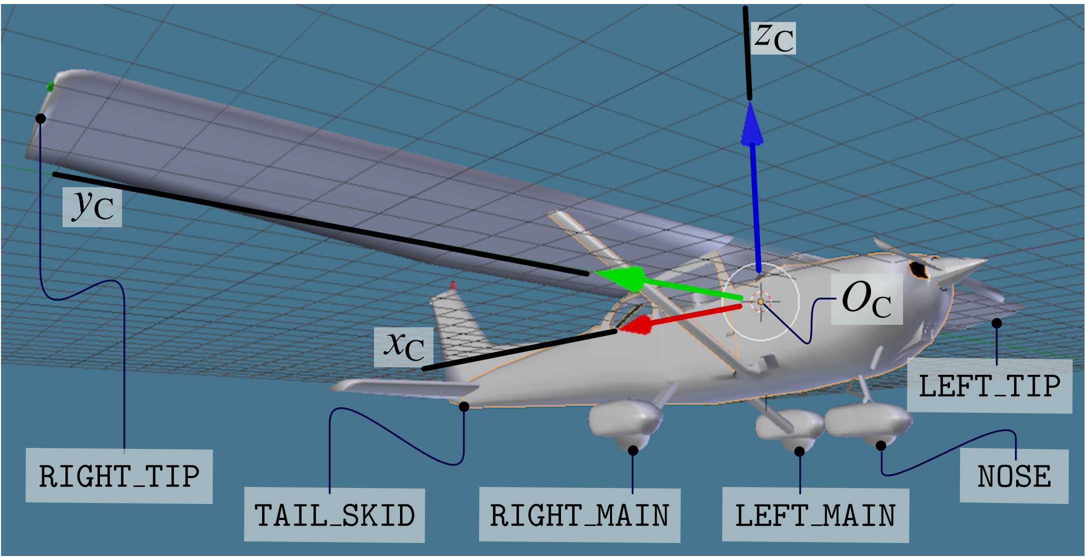


Fig. 4: Definition of ground contact points in terms of construction frame locations.

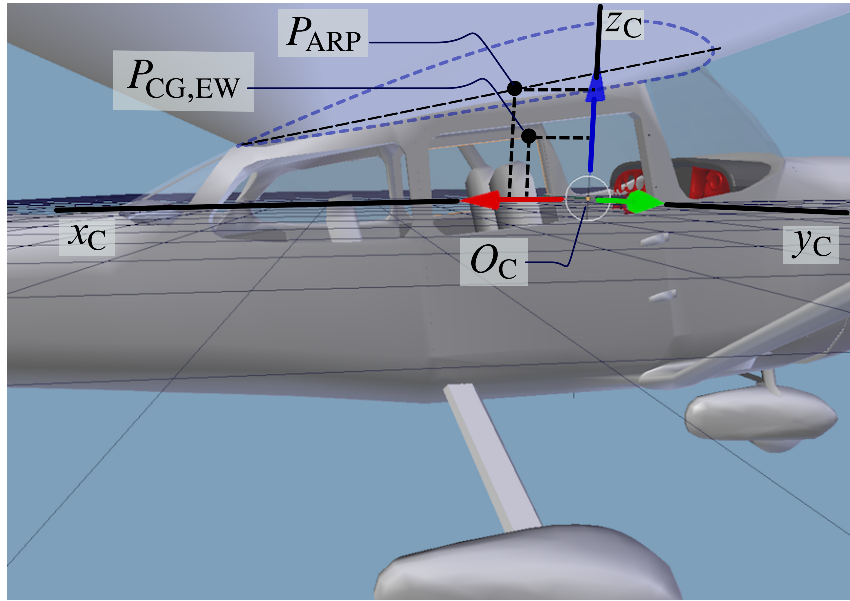


Fig. 5: Two key point locations P_{ARP} and $P_{\mathrm{CG,EW}}$ in the structural frame, respectively, he pole of aerodynamic moments and the Empty Weight CG of the airframe. The shape of the wing root profile and its chord are also sketched.

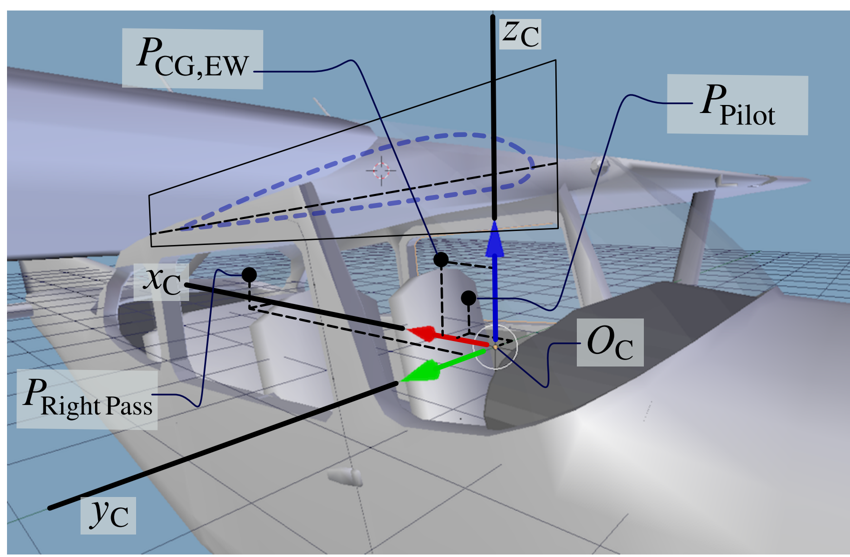


Fig. 6: Besides point $P_{\mathrm{CG,EW}}$, are represented two more significant locations, P_{Pilot} and $P_{\mathrm{Right\,Pass}}$, where two additional masses are concentrated, respectively, of the pilot and of the right passenger.

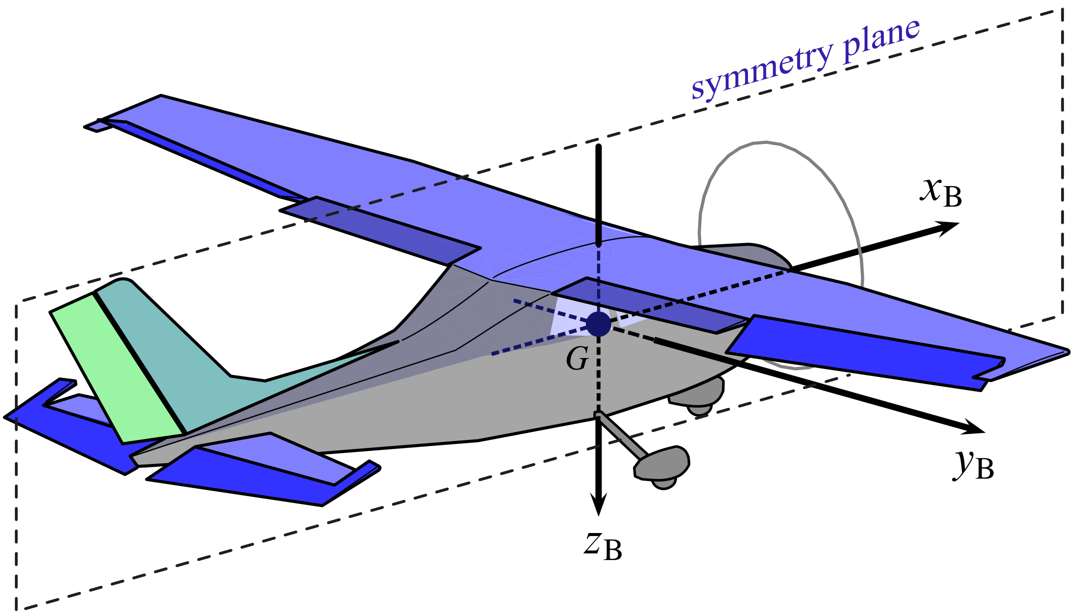


Fig. 7: Standard aircraft body axis frame, with origin at the center of gravity G.

1.1.3 Stability, or "Aerodynamic" Frame

This frame is defined by the relative wind orientation with respect to the aircraft. Denote it as $\mathcal{F}_A = \{G, x_A, y_A, z_A\}$.

- x_A points into the relative wind projected onto the symmetry plane,
- y_A coincides with y_B,
- z_A completes the right-handed triad.

The aerodynamic angles are angle of attack α_B and sideslip β.

In JSBSim usage, the term stability frame commonly refers to this aerodynamic frame. Lift is aligned with $-z_A$ and drag with the opposite wind direction.

1.1.4 Earth-Centered Frames (ECI and ECEF)

The Earth-Centered Inertial frame is $\mathcal{F}_{ECI} = \{O_{ECI}, x_{ECI}, y_{ECI}, z_{ECI}\}$. Its axes are fixed relative to inertial space.

The Earth-Centered Earth-Fixed frame is $\mathcal{F}_{ECEF} = \{O_{ECEF}, x_{ECEF}, y_{ECEF}, z_{ECEF}\}$. Its axes rotate with Earth, with angular rate ω_E.

1.1.5 North-Oriented Tangent Frame

A local tangent frame can be defined at a point O_E on Earth's surface: $\mathcal{F}_E = \{O_E, x_E, y_E, z_E\}$. Here x_E points North, y_E points East, and z_E points Down (NED convention).

1.1.6 Local-Vertical Local-Level Frame (Local NED)

The local vertical frame is $\mathcal{F}_V = \{G, x_V, y_V, z_V\}$. It depends on aircraft position over Earth, not on body attitude.

In this frame, weight has components $(0, 0, mg)$. The aircraft Euler angles ψ, θ, ϕ (3-2-1 sequence) define body orientation with respect to local NED.

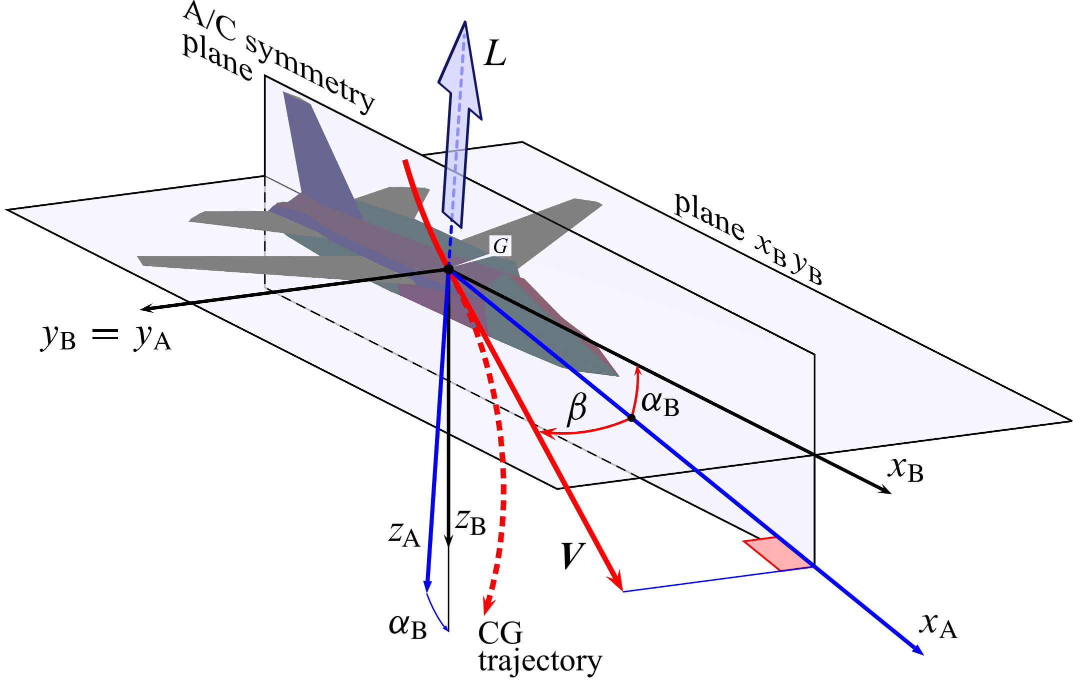


Fig. 8: Aerodynamic frame, defining the aerodynamic angles α_B and β.

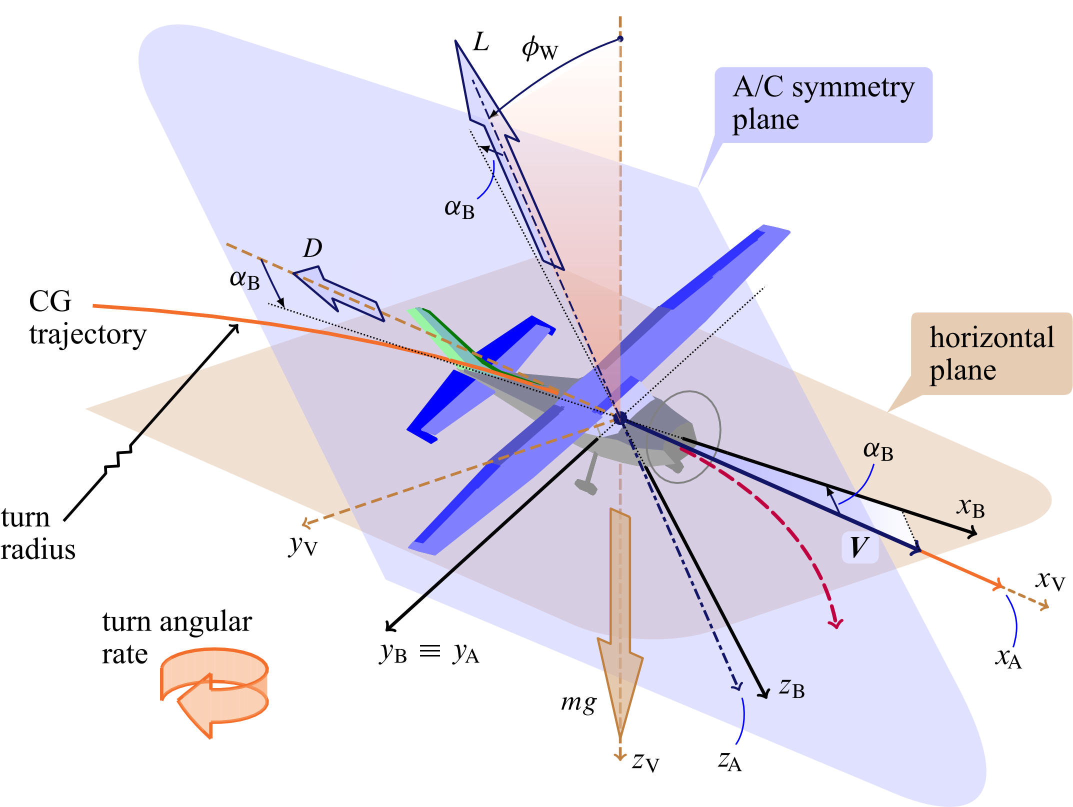


Fig. 9: Banked lift in a steady coordinated turn at constant altitude. The bank angle ϕ_W is a rotation around the relative wind velocity vector. The motion is freezed in time when the velocity vector is aligned with the North. Coordinated turn means that $\beta = 0$ and constant altitude means that x_A is kept horizontal.

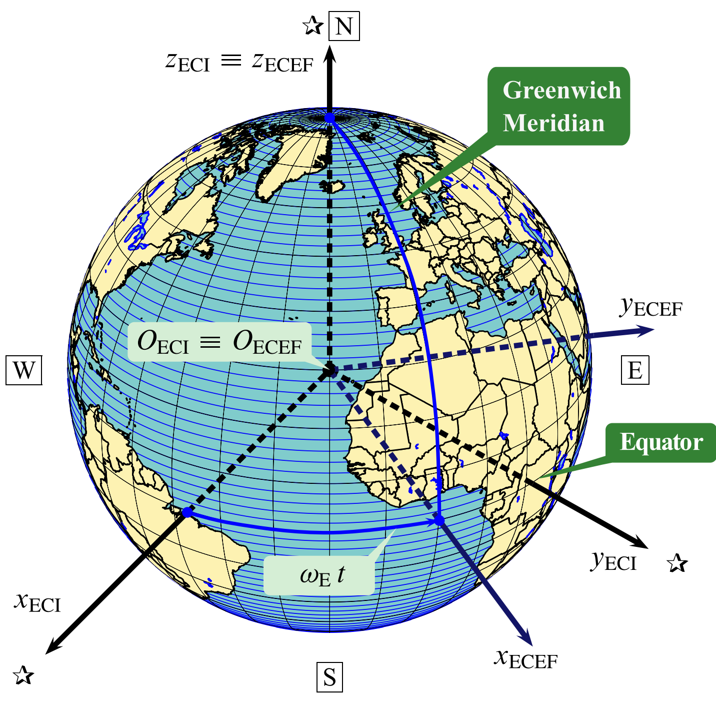


Fig. 10: Earth-Centered Inertial (ECI) frame and Earth-Centered Earth-Fixed (ECEF) frame.

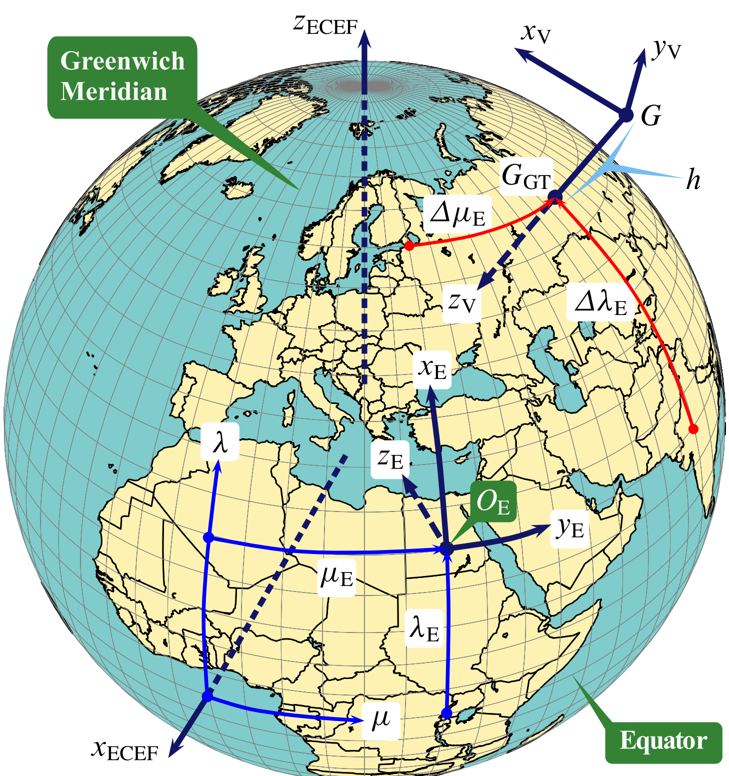


Fig. 11: Earth-Centered Earth-Fixed (ECEF) frame, geografic coordinates, Tangent (T) frame, and local Vertical (V) frame.

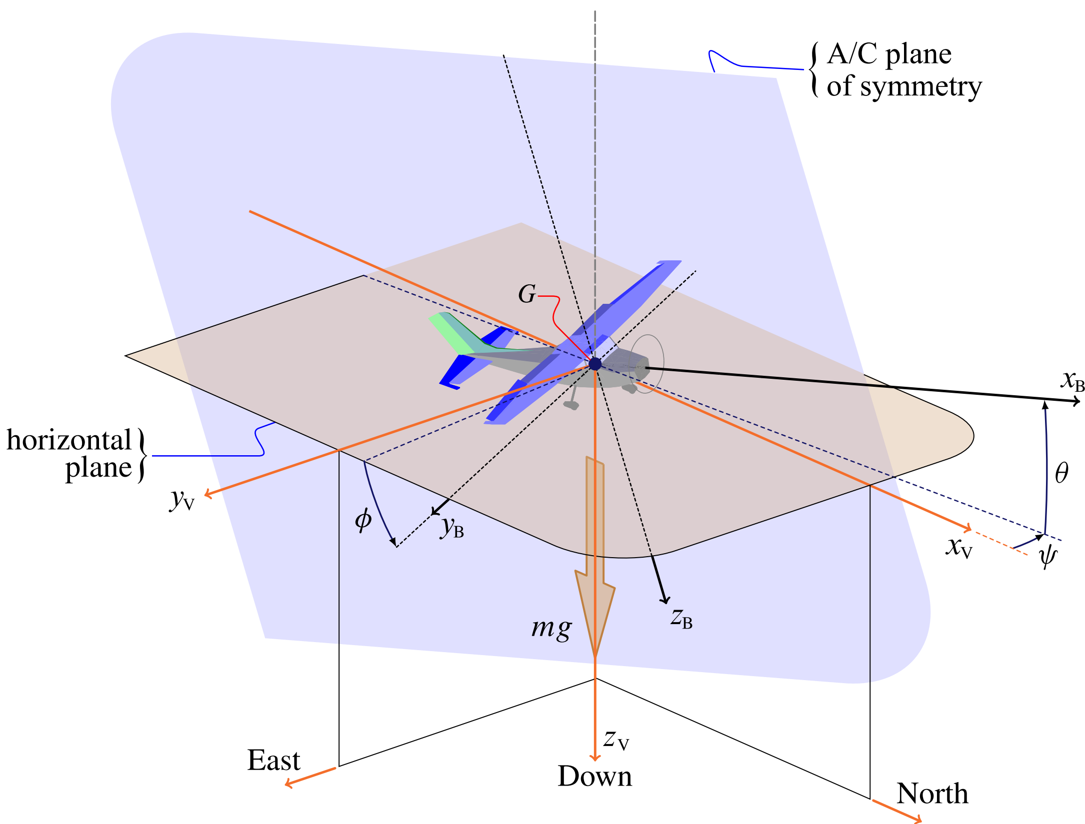


Fig. 12: Aircraft body frame and local vertical frame (NED frame). The aircraft Euler angles are also shown: the heading angle ψ (negative in the picture), the elevation angle θ, and the roll angle ϕ.

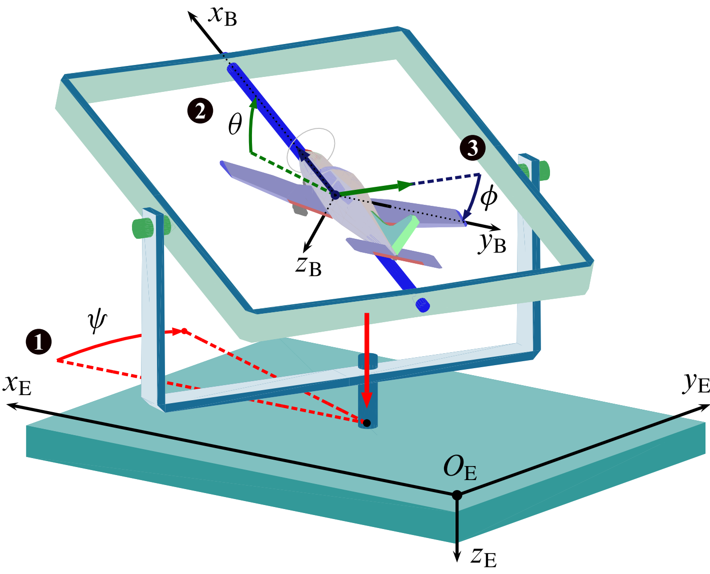


Fig. 13: Euler angle sequence for an aircraft. The frame $\mathcal{F}_E = \{O_E, x_E, y_E, z_E\}$ is an Earth-fixed NED coordinate system, with origin the O_E somewhere on the ground (or at see level) and the plane $x_E y_E$ tangent to the Earth surface. If the ground track point G_{GT} is not too far from O_E, the Earth frame $\mathcal{F}_E$ axes are parallel to those of the local NED frame $\mathcal{F}_V = \{G, x_V, y_V, z_V\}$.

1.1.7 Wind Frame

The wind frame $\mathcal{F}_\mathrm{W} = \{G, x_\mathrm{W}, y_\mathrm{W}, z_\mathrm{W}\}$ uses:

- x_W along velocity direction,
- z_W along lift line (equal to z_A),
- y_W completing the right-handed frame.

Drag and lift in wind axes satisfy $X_\mathrm{W} = -D$ and $Z_\mathrm{W} = -L$. When $\beta = 0$, wind and aerodynamic frames coincide.

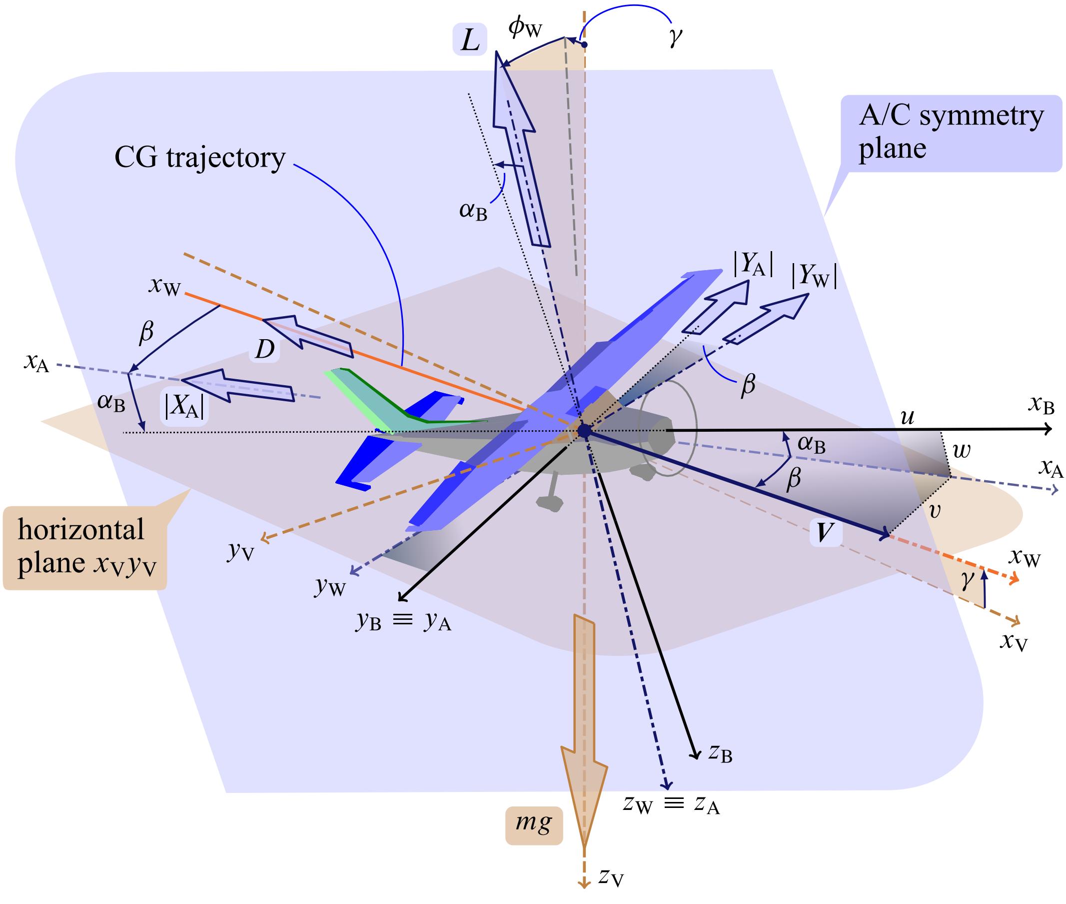


Fig. 14: Standard frames of reference and aircraft in climbing flight in calm air. The CG velocity vector $\boldsymbol{V}$ forms the flight path angle γ with the horizontal plane. The standard three aerodynamic resultant force components D, L and Y_A are also shown.

The relation among wind, aerodynamic, and body frames is:

$$\mathcal{F}_\mathrm{W} \xrightarrow{-\beta \text{ around } z_\mathrm{W}} \mathcal{F}_\mathrm{A} \xrightarrow{\alpha_\mathrm{B} \text{ around } y_\mathrm{A}} \mathcal{F}_\mathrm{B}$$

Using drag, side force and lift, body-axis components are written as:

$$\begin{bmatrix} X_\mathrm{B} \\ Y_\mathrm{B} \\ Z_\mathrm{B} \end{bmatrix} = \begin{bmatrix} \cos\alpha_\mathrm{B} & 0 & -\sin\alpha_\mathrm{B} \\ 0 & 1 & 0 \\ \sin\alpha_\mathrm{B} & 0 & \cos\alpha_\mathrm{B} \end{bmatrix} \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -D \\ Y_\mathrm{W} \\ -L \end{bmatrix}$$

1.2 Simulation

TBD.

1.3 Frames of reference

TBD.

1.4 Units

TBD.

1.5 Properties

TBD.

1.6 Math

TBD.

1.7 Forces and moments

TBD.

1.8 Flight Control and System Modelling

TBD.

Note

This page was generated from a Jupyter notebook.

CHAPTER

TWO

JSBSIM HELLO WORLD

This notebook provides a minimal introduction to the JSBSim Python API.

JSBSim is a fully featured, open-source, multi-platform, flight dynamics model (FDM) written in C++ and exposed to Python via a Cython wrapper. The same FDM is used in FlightGear, OpenPilot, Paparazzi UAV, and many other simulators.

2.1 What we will do

1. Import the `jsbsim` Python module.
2. Create an `FGFDMExec` instance – the top-level JSBSim object.
3. Load the bundled **Cessna 172P** aircraft model.
4. Set initial conditions and trim the aircraft.
5. Step the simulation and read back properties.
6. Print a summary of the simulation state.

2.1.1 Install

```
pip install jsbsim
```

2.2 1. Import and version check

[1]:
```
# If running on Google Colab, install the required packages.

import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab \u2013 installing jsbsim \u2026')
    !pip install jsbsim
```

[2]:
```
import jsbsim

print(f"JSBSim version: {jsbsim.__version__}")
```

```
JSBSim version: 1.3.0
```

2.3 2. Create the Flight Dynamics Model executor

`FGFDMExec` is the main JSBSim class. Passing `None` as the `root_dir` argument makes it use the aircraft and engine data bundled with the Python wheel.

[3]:
```
fdm = jsbsim.FGFDMExec(None)    # None → use data bundled with the pip package
fdm.set_debug_level(0)          # Suppress verbose JSBSim console output

if fdm is not None:
    print("FDM created successfully")
else:
    print("Failed to create FDM")
```

```


     JSBSim Flight Dynamics Model v1.3.0 Apr  9 2026 10:00:08
            [JSBSim-ML v2.0]

JSBSim startup beginning . . .

FDM created successfully
```

2.4 3. Load an aircraft model

The JSBSim wheel ships with many aircraft models. Here we use the **c172p** (Cessna 172P).

[4]:
```
aircraft = 'c172p'
result = fdm.load_model(aircraft)

print(f"load_model('{aircraft}') → {result}")
```

```
load_model('c172p') → True
```

2.5 4. Set initial conditions

JSBSim exposes all model state through a *property tree*. Initial-condition properties are prefixed with `ic/`.

[5]:
```
# Initial altitude above sea level [ft]
fdm['ic/h-sl-ft'] = 3000.0

# Initial calibrated airspeed [knots]
fdm['ic/vc-kts'] = 80.0

# Flight path angle [deg]  (0 = level flight)
fdm['ic/gamma-deg'] = 0.0

# Heading [deg]
fdm['ic/psi-true-deg'] = 45.0

# Apply the initial conditions
fdm.run_ic()

print("Initial conditions applied")
```

```
Initial conditions applied
```

2.6 5. Trim the aircraft

Trimming finds a steady-state flight condition where the net forces and moments acting on the aircraft are (approximately) zero. JSBSim provides a built-in full trim algorithm.

[6]:
```
# Start engines
fdm['propulsion/set-running'] = -1

# Request a full trim (blocks until convergence)
fdm['simulation/do_simple_trim'] = 1

print("Trim complete")
print(f"  Throttle        : {100.*fdm['fcs/throttle-cmd-norm[0]']:.3f} (perc.)")
print(f"  Elevator trim   : {100.*fdm['fcs/pitch-trim-cmd-norm']:.4f} (perc.)")
print(f"  Angle of attack : {fdm['aero/alpha-deg']:.2f} deg")
```

```
Trim complete
  Throttle        : 62.157 (perc.)
  Elevator trim   : 5.2799 (perc.)
  Angle of attack : 2.53 deg
```

2.7 6. Step the simulation and read properties

`fdm.run()` advances the simulation by one time step (`fdm.get_delta_t()` seconds, typically 1/120 s). Properties are read with dictionary-style access.

[7]:
```
dt = fdm.get_delta_t()
print(f"Time step Delta t = {dt:.6f} s  ({1/dt:.1f} Hz)")

# Run for 1 second
n_steps = int(1.0 / dt)
for _ in range(n_steps):
    fdm.run()

print("\nSimulation state after 1 second:")
print(f"  Simulation time : {fdm['simulation/sim-time-sec']:.3f} s")
print(f"  Altitude (MSL)  : {fdm['position/h-sl-ft']:.1f} ft")
print(f"  True airspeed   : {fdm['velocities/vt-fps']:.1f} fps")
print(f"  Pitch angle     : {fdm['attitude/theta-deg']:.2f} deg")
print(f"  Roll angle      : {fdm['attitude/phi-deg']:.2f} deg")
print(f"  Heading         : {fdm['attitude/psi-deg']:.1f} deg")
```

```
Time step Delta t = 0.008333 s  (120.0 Hz)

Simulation state after 1 second:
  Simulation time : 1.000 s
  Altitude (MSL)  : 3000.0 ft
  True airspeed   : 141.1 fps
  Pitch angle     : 2.53 deg
  Roll angle      : 0.06 deg
  Heading         : 45.0 deg
```

2.8 7. Explore the property tree

JSBSim exposes hundreds of properties. The `query_property_catalog` method searches by substring.

[8]:
```
# Find all velocity-related properties
catalog_str = fdm.query_property_catalog('velocities')
velocity_props = [p for p in catalog_str.split('\n') if p.strip()]

print(f"Found {len(velocity_props)} velocity properties (first 15):")
```

(continues on next page)

(continued from previous page)

```
for p in velocity_props[:15]:
    name = p.strip().split(' ')[0]   # strip access flag suffix
    try:
        value = fdm[name]
        print(f"  {name:<45s} = {value:.4f}")
    except Exception:
        print(f"  {name:<45s} (unreadable)")
```

```
Found 40 velocity properties (first 15):
  velocities/h-dot-fps                          = 0.0013
  velocities/v-north-fps                        = 99.7897
  velocities/v-east-fps                         = 99.7803
  velocities/v-down-fps                         = -0.0013
  velocities/u-fps                              = 140.9793
  velocities/v-fps                              = -0.0000
  velocities/w-fps                              = 6.2395
  velocities/p-rad_sec                          = 0.0000
  velocities/q-rad_sec                          = 0.0000
  velocities/r-rad_sec                          = -0.0000
  velocities/pi-rad_sec                         = 0.0001
  velocities/qi-rad_sec                         = -0.0000
  velocities/ri-rad_sec                         = 0.0000
  velocities/eci-x-fps                          = -0.1255
  velocities/eci-y-fps                          = 1625.9213
```

2.9 Summary

In this notebook you learned how to:

- Create an `FGFDMExec` instance.
- Load an aircraft model bundled with the JSBSim Python wheel.
- Set initial conditions and perform a trim.
- Advance the simulation with `fdm.run()` and read properties.
- Query the property tree.

Next: *02_jsbsim_flight_simulation.ipynb* – a longer simulation with time-history plots.

Note

This page was generated from a Jupyter notebook.

CHAPTER

THREE

JSBSIM FLIGHT SIMULATION

This notebook demonstrates a trimmed flight simulation with the **Cessna 172P** using the JSBSim Python API. We record a set of key flight parameters over a 60-second time window and visualise them as time-history plots.

3.1 Topics covered

- Running a longer simulation loop with data collection.
- Working with JSBSim's property tree (positions, velocities, attitudes, forces).
- Unit conversions (feet → metres, knots → m/s, radians → degrees).
- Producing publication-quality plots with `matplotlib`.

3.2 1. Setup

[1]:
```
# If running on Google Colab, install the required packages.

import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab \u2013 installing jsbsim \u2026')
    !pip install jsbsim
```

[2]:
```
import math
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import jsbsim

matplotlib.rcParams.update({
    'figure.dpi': 120,
    'axes.grid': True,
    'grid.alpha': 0.4,
})

print(f"JSBSim version : {jsbsim.__version__}")
print(f"NumPy version  : {np.__version__}")
print(f"Matplotlib ver : {matplotlib.__version__}")
```

```
JSBSim version : 1.3.0
NumPy version  : 2.4.4
Matplotlib ver : 3.10.8
```

3.3 2. Initialise JSBSim and trim

[3]:
```
# ---------- configuration ----------
AIRCRAFT      = 'c172p'
ALT_FT        = 5000.0   # Initial altitude [ft MSL]
AIRSPEED_KTS  = 90.0     # Calibrated airspeed [kts]
HEADING_DEG   = 90.0     # True heading [deg]
SIM_DURATION  = 60.0     # Simulation duration [s]
RECORD_HZ     = 10       # Data recording frequency [Hz]
# -----------------------------------

fdm = jsbsim.FGFDMExec(None)
fdm.set_debug_level(0)
fdm.load_model(AIRCRAFT)

fdm['ic/h-sl-ft']      = ALT_FT
fdm['ic/vc-kts']       = AIRSPEED_KTS
fdm['ic/gamma-deg']    = 0.0
fdm['ic/psi-true-deg'] = HEADING_DEG
fdm.run_ic()

fdm['propulsion/set-running'] = -1
fdm['simulation/do_simple_trim'] = 1

dt          = fdm.get_delta_t()
record_step = max(1, int(1.0 / (RECORD_HZ * dt)))

print(f"Aircraft       : {AIRCRAFT}")
print(f"Alt (MSL)      : {ALT_FT:.0f} ft")
print(f"Airspeed       : {AIRSPEED_KTS:.0f} kts KCAS")
print(f"Time step Δt    : {dt:.6f} s  ({1/dt:.0f} Hz)")
print(f"Record step    : every {record_step} steps ({RECORD_HZ} Hz)")
print(f"Trim AoA       : {fdm['aero/alpha-deg']:.2f} deg")
print(f"Trim throttle  : {fdm['fcs/throttle-cmd-norm[0]']:.3f}")
```

```


     JSBSim Flight Dynamics Model v1.3.0 Apr  9 2026 10:00:08
            [JSBSim-ML v2.0]

JSBSim startup beginning . . .

Aircraft       : c172p
Alt (MSL)      : 5000 ft
Airspeed       : 90 kts KCAS
Time step Δt    : 0.008333 s  (120 Hz)
Record step    : every 12 steps (10 Hz)
Trim AoA       : 1.28 deg
Trim throttle  : 0.694
```

3.4 3. Run the simulation and record data

[4]:
```
# Containers for recorded data
rec = {
    't_s':        [],
    'alt_ft':     [],
```

(continues on next page)

(continued from previous page)

```
    'alt_m':      [],
    'vt_fps':     [],
    'vc_kts':     [],
    'alpha_deg':  [],
    'theta_deg':  [],
    'phi_deg':    [],
    'psi_deg':    [],
    'nx':         [],    # load factor (longitudinal)
    'nz':         [],    # load factor (normal)
    'lat_deg':    [],
    'lon_deg':    [],
}

total_steps = int(SIM_DURATION / dt)
step = 0

while step < total_steps:
    fdm.run()
    step += 1

    if step % record_step == 0:
        rec['t_s'].append(fdm['simulation/sim-time-sec'])
        rec['alt_ft'].append(fdm['position/h-sl-ft'])
        rec['alt_m'].append(fdm['position/h-sl-ft'] * 0.3048)
        rec['vt_fps'].append(fdm['velocities/vt-fps'])
        rec['vc_kts'].append(fdm['velocities/vc-kts'])
        rec['alpha_deg'].append(fdm['aero/alpha-deg'])
        rec['theta_deg'].append(fdm['attitude/theta-deg'])
        rec['phi_deg'].append(fdm['attitude/phi-deg'])
        rec['psi_deg'].append(fdm['attitude/psi-deg'])
        rec['nx'].append(fdm['accelerations/Nx'])
        rec['nz'].append(fdm['accelerations/Nz'])
        rec['lat_deg'].append(math.degrees(fdm['position/lat-geod-rad']))
        rec['lon_deg'].append(math.degrees(fdm['position/long-gc-rad']))

# Convert to NumPy arrays for convenient indexing
for key in rec:
    rec[key] = np.array(rec[key])

print(f"Recorded {len(rec['t_s'])} samples over {SIM_DURATION:.0f} s")
print(f"Final altitude : {rec['alt_ft'][-1]:.1f} ft")
print(f"Final airspeed : {rec['vc_kts'][-1]:.1f} kts KCAS")
```

```
Recorded 600 samples over 60 s
Final altitude : 5000.2 ft
Final airspeed : 90.0 kts KCAS
```

3.5 4. Visualise – altitude and airspeed

[5]:
```
fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

axes[0].plot(rec['t_s'], rec['alt_ft'], color='steelblue', linewidth=1.5)
axes[0].set_ylabel('Altitude (ft MSL)')
axes[0].set_title(f'{AIRCRAFT.upper()} – Trimmed Level Flight  '
                  f'({ALT_FT:.0f} ft, {AIRSPEED_KTS:.0f} kts)')
```

(continues on next page)

(continued from previous page)

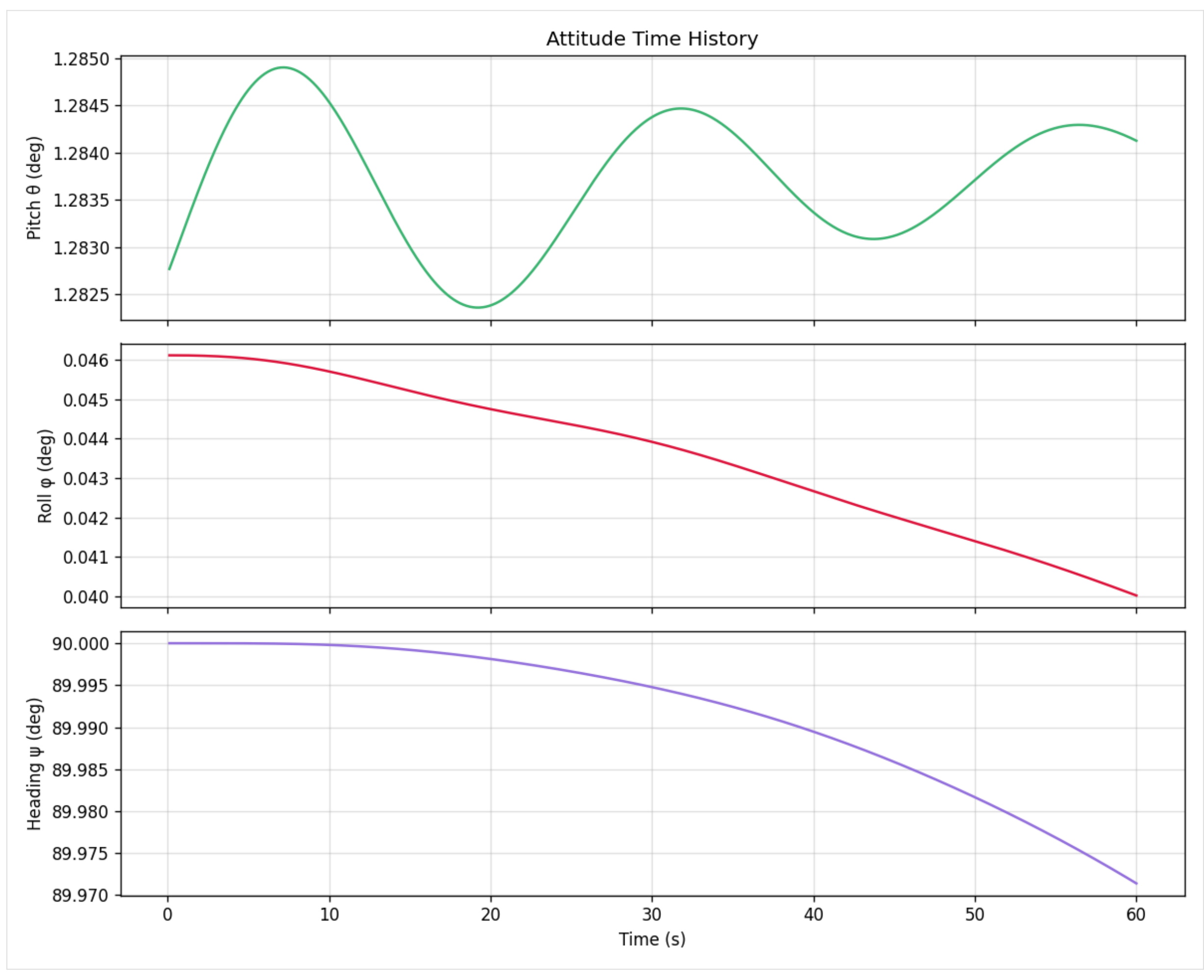
```
axes[1].plot(rec['t_s'], rec['vc_kts'], color='darkorange', linewidth=1.5)
axes[1].set_ylabel('Calibrated Airspeed (kts)')
axes[1].set_xlabel('Time (s)')

plt.tight_layout()
plt.savefig('altitude_airspeed.png', bbox_inches='tight')
plt.show()
print("Figure saved as altitude_airspeed.png")
```



```
Figure saved as altitude_airspeed.png
```

3.6 5. Visualise – attitude angles

```
[6]: fig, axes = plt.subplots(3, 1, figsize=(10, 8), sharex=True)

axes[0].plot(rec['t_s'], rec['theta_deg'], color='mediumseagreen', linewidth=1.5)
axes[0].set_ylabel('Pitch θ (deg)')
axes[0].set_title('Attitude Time History')

axes[1].plot(rec['t_s'], rec['phi_deg'], color='crimson', linewidth=1.5)
axes[1].set_ylabel('Roll ϕ (deg)')

axes[2].plot(rec['t_s'], rec['psi_deg'], color='mediumpurple', linewidth=1.5)
axes[2].set_ylabel('Heading ψ (deg)')
axes[2].set_xlabel('Time (s)')

plt.tight_layout()
plt.savefig('attitude.png', bbox_inches='tight')
plt.show()
```



3.7 6. Visualise – ground track

[7]:
```
fig, ax = plt.subplots(figsize=(7, 7))

sc = ax.scatter(
    rec['lon_deg'], rec['lat_deg'],
    c=rec['t_s'], cmap='plasma', s=6, zorder=3
)
cbar = fig.colorbar(sc, ax=ax, label='Time (s)')

ax.plot(rec['lon_deg'][0], rec['lat_deg'][0],
        'go', ms=10, label='Start', zorder=4)
ax.plot(rec['lon_deg'][-1], rec['lat_deg'][-1],
        'rs', ms=10, label='End', zorder=4)

ax.set_xlabel('Longitude (deg)')
ax.set_ylabel('Latitude (deg)')
ax.set_title('Ground Track')
ax.legend()

x_min, x_max = ax.get_xlim()
y_min, y_max = ax.get_ylim()
span = max(x_max - x_min, y_max - y_min)
x_center = 0.5 * (x_min + x_max)
```

(continues on next page)

(continued from previous page)

```
y_center = 0.5 * (y_min + y_max)
ax.set_xlim(x_center - 0.5 * span, x_center + 0.5 * span)
ax.set_ylim(y_center - 0.5 * span, y_center + 0.5 * span)
ax.set_aspect('equal', adjustable='box')

plt.tight_layout()
plt.savefig('ground_track.png', bbox_inches='tight')
plt.show()
```


3.8 7. Reset and re-run: 10-degree turn to the left

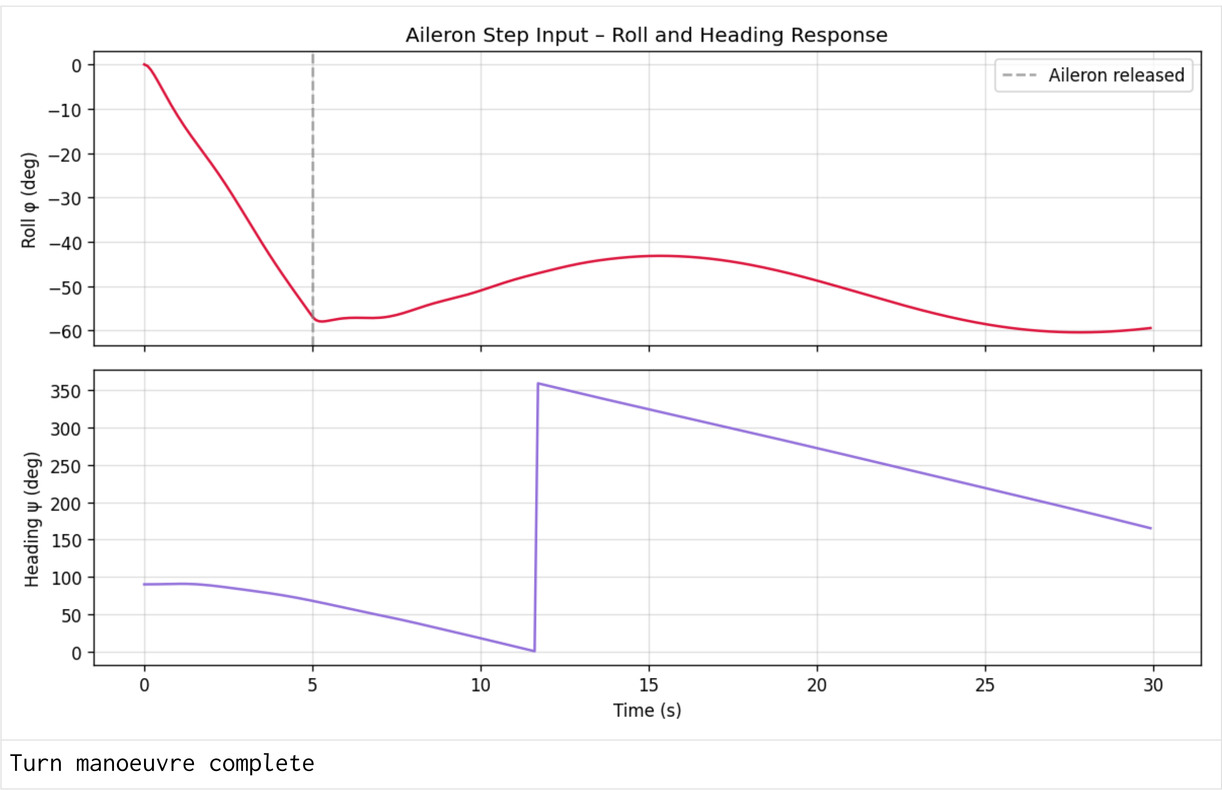
JSBSim can be reset and re-trimmed to explore different manoeuvres. Here we apply a step aileron input to initiate a gentle left turn.

```
[8]: # Reset
fdm.reset_to_initial_conditions(0)
```

(continues on next page)

(continued from previous page)

```
fdm['propulsion/set-running'] = -1
fdm['simulation/do_simple_trim'] = 1

turn_rec = {'t_s': [], 'phi_deg': [], 'psi_deg': [], 'alt_ft': []}

total_steps = int(30.0 / dt)
for step in range(total_steps):
    # Apply a gentle left aileron deflection for the first 5 s
    if fdm['simulation/sim-time-sec'] < 5.0:
        fdm['fcs/aileron-cmd-norm'] = -0.15
    else:
        fdm['fcs/aileron-cmd-norm'] = 0.0

    fdm.run()

    if step % record_step == 0:
        turn_rec['t_s'].append(fdm['simulation/sim-time-sec'])
        turn_rec['phi_deg'].append(fdm['attitude/phi-deg'])
        turn_rec['psi_deg'].append(fdm['attitude/psi-deg'])
        turn_rec['alt_ft'].append(fdm['position/h-sl-ft'])

for key in turn_rec:
    turn_rec[key] = np.array(turn_rec[key])

fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

axes[0].plot(turn_rec['t_s'], turn_rec['phi_deg'], color='crimson', linewidth=1.5)
axes[0].axvline(5.0, linestyle='--', color='grey', alpha=0.7, label='Aileron released')
axes[0].set_ylabel('Roll φ (deg)')
axes[0].set_title('Aileron Step Input – Roll and Heading Response')
axes[0].legend()

axes[1].plot(turn_rec['t_s'], turn_rec['psi_deg'], color='mediumpurple', linewidth=1.5)
axes[1].set_ylabel('Heading ψ (deg)')
axes[1].set_xlabel('Time (s)')

plt.tight_layout()
plt.savefig('turn_manoeuvre.png', bbox_inches='tight')
plt.show()
print("Turn manoeuvre complete")
```




```
Turn manoeuvre complete
```

3.9 Summary

In this notebook you:

- Ran a 60-second trimmed level-flight simulation.
- Recorded and plotted altitude, airspeed, attitude, and ground track.
- Demonstrated how to apply control inputs (aileron step) and observe the aircraft's response.

Next: *03_pathsim_intro.ipynb* – introduction to PathSim.

Note

This page was generated from a Jupyter notebook.

CHAPTER

FOUR

PATHSIM INTRODUCTION

PathSim is a block-based, time-domain system simulation framework for Python. It lets you model dynamical systems graphically using *blocks* (sources, integrators, functions, scopes, …) and *connections*, then simulate them with a choice of ODE solvers.

PathSim is particularly useful for:

- Rapid prototyping of control systems.
- Running hardware-in-the-loop (HIL) scenarios.
- Wrapping external simulation engines (e.g. JSBSim – see `04_pathsim_jsbsim_trim_elevator_doublet.ipynb`).

Documentation: https://docs.pathsim.org

4.1 Install

```
pip install pathsim
# or
conda install conda-forge::pathsim
```

4.1.1 1. Imports

[1]:
```
# If running on Google Colab, install the required packages.

import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab \u2013 installing pathsim \u2026')
    !pip install pathsim
```

[2]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

from pathsim import Simulation, Connection
from pathsim.blocks import (
    Integrator,
    Amplifier,
    Adder,
    Scope,
    Source,
    Constant,
    Function,
```

(continues on next page)

(continued from previous page)

```
)
import pathsim

matplotlib.rcParams.update({'figure.dpi': 120, 'axes.grid': True, 'grid.alpha': 0.4})

print(f"PathSim version : {pathsim.__version__}")
```

```
PathSim version : 0.20.0
```

4.1.2 2. Example 1 – Damped harmonic oscillator

We model the second-order ODE

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0$$

with $\omega_n = 2$ rad/s and $\zeta = 0.25$ (under-damped).

The block diagram is:

```
Adder  ⟶ Integrator(v0)  ⟶ Integrator(x0) -⟶ Scope
 ↑                ↓                ↓
 — Amp(-2ζωn) —    Amp(-ωn²) —
```

[3]:
```
omega_n = 2.0   # natural frequency [rad/s]
zeta    = 0.25  # damping ratio

# Initial conditions: x0 = 1, v0 = 0
int_v = Integrator(0.0)                # integrates acceleration ⟶ velocity
int_x = Integrator(1.0)                # integrates velocity   ⟶ position
amp_d = Amplifier(-2 * zeta * omega_n) # damping term
amp_k = Amplifier(-omega_n**2)         # stiffness term
adder = Adder()
scope = Scope(labels=['position x', 'velocity v'])

sim1 = Simulation(
    blocks=[int_v, int_x, amp_d, amp_k, adder, scope],
    connections=[
        Connection(int_v, int_x, amp_d),        # v  ⟶ x-dot and damping amp
        Connection(int_x, amp_k, scope[1]),      # x  ⟶ stiffness amp and scope ch-1
        Connection(amp_d, adder),                # damping ⟶ adder ch-0
        Connection(amp_k, adder[1]),             # stiffness ⟶ adder ch-1
        Connection(adder, int_v),                # sum ⟶ x-ddot
        Connection(int_v, scope[0]),             # v   ⟶ scope ch-0  (reorder display)
    ],
    dt=0.01
)

sim1.run(20)

t, scope_data = scope.read()
x_data, v_data = scope_data[0], scope_data[1]

fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(t, x_data, label='position $x$', linewidth=1.5)
ax.plot(t, v_data, label='velocity $\\dot{x}$', linewidth=1.5, linestyle='--')
ax.axhline(0, color='k', linewidth=0.7)
ax.set_xlabel('Time [s]')
```

(continues on next page)

(continued from previous page)

```
ax.set_title(
    f'Damped Harmonic Oscillator  '
    f'($\\omega_n={omega_n}$ rad/s, $\\zeta={zeta}$)'
)
ax.legend()
plt.tight_layout()
plt.savefig('harmonic_oscillator.png', bbox_inches='tight')
plt.show()
```

```
15:13:41 - INFO - LOGGING (log: True)
15:13:41 - INFO - BLOCKS (total: 6, dynamic: 2, static: 4, eventful: 0)
15:13:41 - INFO - GRAPH (nodes: 6, edges: 8, alg. depth: 3, loop depth: 0, runtime: 0.
↪073ms)
15:13:41 - INFO - STARTING -> TRANSIENT (Duration: 20.00s)
15:13:41 - INFO - --------------------   1% | 0.0s<0.4s | 5479.5 it/s
15:13:41 - INFO - ####----------------  20% | 0.1s<0.2s | 6912.9 it/s
15:13:41 - INFO - ########------------  40% | 0.1s<0.2s | 6455.0 it/s
15:13:41 - INFO - ############--------  60% | 0.2s<0.1s | 13251.3 it/s
15:13:41 - INFO - ################----  80% | 0.2s<0.0s | 13101.9 it/s
15:13:41 - INFO - #################### 100% | 0.3s<--:-- | 10958.3 it/s
15:13:41 - INFO - FINISHED -> TRANSIENT (total steps: 2000, successful: 2000, runtime:␣
↪277.51 ms)
```


4.1.3 3. Example 2 – First-order system step response

A first-order linear system driven by a unit step:

$$\tau\dot{y} + y = u(t), \quad u(t) = 1 \text{ for } t \geq 0$$

The analytical solution is $y(t) = 1 - e^{-t/\tau}$.

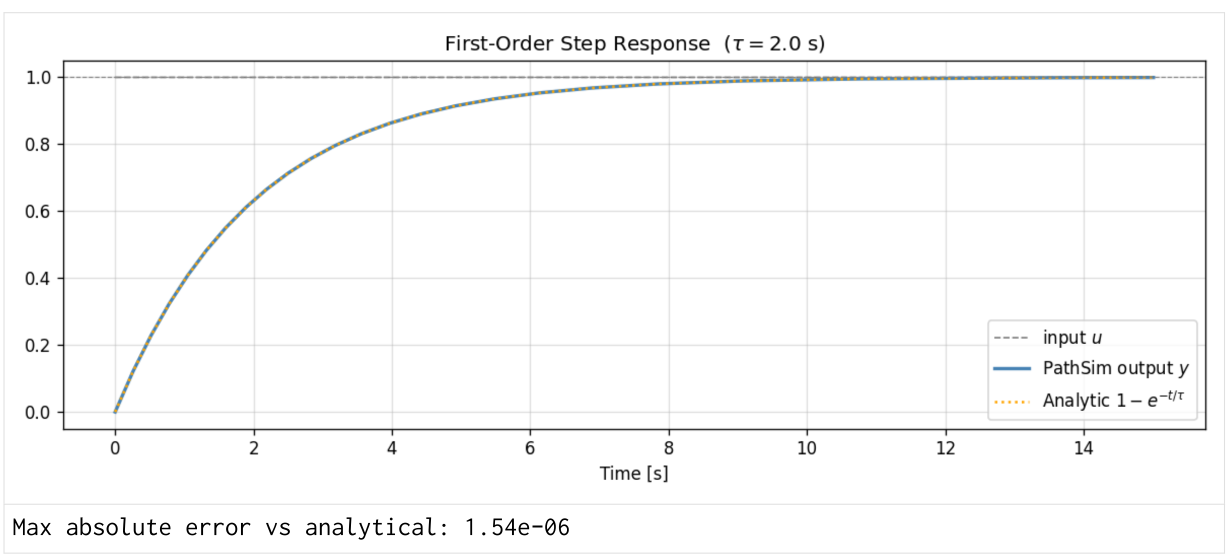
[4]:
```
tau = 2.0  # time constant [s]

step_src  = Constant(1.0)               # unit step input
adder2    = Adder('+-') # error = u - y
amp_tau   = Amplifier(1.0 / tau)        # 1/τ
integr    = Integrator(0.0)             # state y
scope2    = Scope(labels=['y', 'u'])

sim2 = Simulation(
    blocks=[step_src, adder2, amp_tau, integr, scope2],
```

(continues on next page)

(continued from previous page)

```
    connections=[
        Connection(step_src, adder2, scope2[1]),  # u → error adder and scope
        Connection(integr, adder2[1], scope2[0]), # y → error adder (neg) and scope
        Connection(adder2, amp_tau),               # error → 1/τ
        Connection(amp_tau, integr),               # ẏ = (u-y)/τ
    ],
    dt=0.01
)

sim2.run(15)

t2, scope2_data = scope2.read()
y2, u2 = scope2_data[0], scope2_data[1]
y_analytic = 1 - np.exp(-np.array(t2) / tau)

fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(t2, u2, 'k--', linewidth=1.0, label='input $u$', alpha=0.5)
ax.plot(t2, y2, color='steelblue', linewidth=2.0, label='PathSim output $y$')
ax.plot(t2, y_analytic, color='orange', linewidth=1.5,
        linestyle=':', label='Analytic $1-e^{-t/\\tau}$')
ax.axhline(1.0, color='grey', linewidth=0.7, linestyle='--')
ax.set_xlabel('Time [s]')
ax.set_title(f'First-Order Step Response  ($\\tau = {tau}$ s)')
ax.legend()
plt.tight_layout()
plt.savefig('first_order_step.png', bbox_inches='tight')
plt.show()

max_err = np.max(np.abs(np.array(y2) - y_analytic))
print(f"Max absolute error vs analytical: {max_err:.2e}")
```

```
15:13:41 - INFO - LOGGING (log: True)
15:13:41 - INFO - BLOCKS (total: 5, dynamic: 1, static: 4, eventful: 0)
15:13:41 - INFO - GRAPH (nodes: 5, edges: 6, alg. depth: 3, loop depth: 0, runtime: 0.
↪069ms)
15:13:41 - INFO - STARTING -> TRANSIENT (Duration: 15.00s)
15:13:41 - INFO - --------------------   1% | 0.0s<0.2s | 7242.4 it/s
15:13:41 - INFO - ####----------------  20% | 0.0s<0.1s | 15352.0 it/s
15:13:41 - INFO - ########------------  40% | 0.0s<0.1s | 16822.7 it/s
15:13:41 - INFO - ############--------  60% | 0.1s<0.1s | 8802.3 it/s
15:13:42 - INFO - ################----  80% | 0.1s<0.0s | 16808.7 it/s
15:13:42 - INFO - #################### 100% | 0.1s<--:-- | 16611.6 it/s
15:13:42 - INFO - FINISHED -> TRANSIENT (total steps: 1501, successful: 1501, runtime:␣
↪140.00 ms)
```




```
Max absolute error vs analytical: 1.54e-06
```

4.1.4 4. Example 3 – Sinusoidal source and phase portrait

PathSim provides a Source block that wraps an arbitrary time-varying function.

```
[5]: import math

# Van der Pol oscillator:  x'' - μ(1-x²)x' + x = 0
mu = 1.5

int_vdp_v = Integrator(0.5)   # ẋ,  initial velocity
int_vdp_x = Integrator(2.0)   # x,  initial position

# The nonlinear damping term μ(1-x²)ẋ
def vdp_nonlinear(x, v):
    return mu * (1 - x**2) * v

nl_block  = Function(vdp_nonlinear)
adder_vdp = Adder('+-')
scope_vdp = Scope(labels=['x', 'v'])

sim3 = Simulation(
    blocks=[int_vdp_v, int_vdp_x, nl_block, adder_vdp, scope_vdp],
    connections=[
        Connection(int_vdp_v, int_vdp_x, nl_block[1], scope_vdp[1]),  # v
        Connection(int_vdp_x, nl_block[0], scope_vdp[0]),              # x
        Connection(nl_block, adder_vdp),                                # μ(1-x²)v
        Connection(int_vdp_x, adder_vdp[1]),                           # x (→ -x)
        Connection(adder_vdp, int_vdp_v),                              # ẍ
    ],
    dt=0.01
)

sim3.run(40)

t3, vdp_data = scope_vdp.read()
x3, v3 = vdp_data[0], vdp_data[1]
x3, v3 = np.array(x3), np.array(v3)

fig, axes = plt.subplots(1, 2, figsize=(12, 4))
```

(continues on next page)

(continued from previous page)

```
axes[0].plot(t3, x3, linewidth=1.2, label='$x$', color='steelblue')
axes[0].plot(t3, v3, linewidth=1.2, label='$\\dot{x}$',
             color='darkorange', linestyle='--')
axes[0].set_xlabel('Time [s]')
axes[0].set_title(f'Van der Pol Oscillator  ($\\mu={mu}$)')
axes[0].legend()

axes[1].plot(x3, v3, linewidth=1.0, color='mediumpurple')
axes[1].plot(x3[0], v3[0], 'go', ms=8, label='start')
axes[1].set_xlabel('$x$')
axes[1].set_ylabel('$\\dot{x}$')
axes[1].set_title('Phase Portrait')
axes[1].legend()

plt.tight_layout()
plt.savefig('van_der_pol.png', bbox_inches='tight')
plt.show()
```

```
15:13:42 - INFO - LOGGING (log: True)
15:13:42 - INFO - BLOCKS (total: 5, dynamic: 2, static: 3, eventful: 0)
15:13:42 - INFO - GRAPH (nodes: 5, edges: 8, alg. depth: 3, loop depth: 0, runtime: 0.
↪054ms)
15:13:42 - INFO - STARTING -> TRANSIENT (Duration: 40.00s)
15:13:42 - INFO - --------------------   1% | 0.0s<0.7s | 5449.2 it/s
15:13:42 - INFO - ####----------------  20% | 0.1s<0.5s | 6963.1 it/s
15:13:42 - INFO - ########------------  40% | 0.2s<0.2s | 13459.5 it/s
15:13:42 - INFO - ############--------  60% | 0.3s<0.1s | 13599.2 it/s
15:13:42 - INFO - ################----  80% | 0.4s<0.1s | 13061.9 it/s
15:13:42 - INFO - #################### 100% | 0.5s<--:-- | 6963.2 it/s
15:13:42 - INFO - FINISHED -> TRANSIENT (total steps: 4000, successful: 4000, runtime:␣
↪458.85 ms)
```

4.1.5 5. PathView

PathView is the graphical block-diagram editor for PathSim. You can:

1. **Design** your block diagram visually in the browser.
2. **Export** the diagram as a Python script.
3. **Run** the exported script in this environment.

The exported Python file uses the same `Simulation`, `Connection`, and `Block` API shown above.

```
[6]: from IPython.display import IFrame

     # Display PathView in an iframe (requires internet access)
     IFrame(src='https://view.pathsim.org', width='100%', height=500)
```

```
[6]: <IPython.lib.display.IFrame at 0x74f1aeedfa10>
```

4.1.6 Summary

You have seen three examples of PathSim block-diagram simulation:

1. **Damped harmonic oscillator** – second-order ODE via two cascaded integrators.
2. **First-order step response** – verified against an analytical solution.
3. **Van der Pol oscillator** – nonlinear system with a custom `Function` block and phase portrait.

Next: *04_pathsim_jsbsim_trim_elevator_doublet.ipynb* – embedding JSBSim directly in a PathSim block diagram (trim + elevator doublet).

Note

This page was generated from a Jupyter notebook.

CHAPTER

FIVE

JSBSIM TRIM AND ELEVATOR DOUBLET WITH PATHSIM

This notebook demonstrates how to embed a JSBSim flight dynamics model (FDM) directly inside a PathSim block diagram using a `DynamicalFunction` block — without any additional wrapper library.

The example is adapted from test_pathsim_01_Trim_Elevator_Doublet.ipynb and covers:

1. Instantiating and trimming a JSBSim FDM (Global 5000 business jet).
2. Wrapping the FDM step inside a PathSim `DynamicalFunction` block.
3. Assembling a block diagram that applies a **doublet elevator input** and records the angle-of-attack response.
4. Plotting the results.

Prerequisites: `jsbsim` and `pathsim` must be installed.

```
pip install jsbsim
pip install pathsim
```

Previous notebook: *03_pathsim_intro.ipynb* – PathSim block-diagram basics.
Next notebook: *05_pathsim_flight.ipynb* – higher-level `pathsim-flight` integration.

5.1 1. Imports

[1]:
```python
# If running on Google Colab, install the required packages.

import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab \u2013 installing jsbsim, pathsim \u2026')
    !pip install jsbsim pathsim
```

[2]:
```python
import os
import xml.etree.ElementTree as ET

import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import jsbsim
import pathsim

matplotlib.rcParams.update({'figure.dpi': 120, 'axes.grid': True, 'grid.alpha': 0.4})
```

(continues on next page)

(continued from previous page)

```
# Suppress JSBSim console output
# jsbsim.FGJSBBase().debug_lvl = 0

print(f"JSBSim version  : {jsbsim.__version__}")
print(f"PathSim version : {pathsim.__version__}")
```

```
JSBSim version  : 1.3.0
PathSim version : 0.20.0
```

5.2 2. Initialize the JSBSim FDM

Instantiate an `FGFDMExec` object and load the **Global 5000** business-jet model (`global5000`) from the JSBSim Python-package data directory.

[3]:
```
AIRCRAFT_NAME = "global5000"

# Use the aircraft data bundled with the jsbsim Python package
fdm = jsbsim.FGFDMExec(jsbsim.get_default_root_dir())

fdm.set_debug_level(0) # Suppress verbose JSBSim console output

load_status = fdm.load_model(AIRCRAFT_NAME)

# Property manager gives direct node access (used later to read alpha)
pm = fdm.get_property_manager()

print(f"Aircraft loaded  : {AIRCRAFT_NAME}")
print(f"JSBSim root dir  : {fdm.get_root_dir()}")
print(f"Load status      : {load_status}")
```

```


     JSBSim Flight Dynamics Model v1.3.0 Apr  9 2026 10:00:08
            [JSBSim-ML v2.0]

JSBSim startup beginning . . .

Aircraft loaded   : global5000
JSBSim root dir   : /home/vscode/.local/lib/python3.11/site-packages/jsbsim
Load status       : True
```

5.3 3. Aircraft Parameters and Trim Conditions

Parse the aircraft XML to extract the empty weight and centre-of-gravity location, then define payload, fuel, and initial flight conditions for the trim.

[4]:
```
# Parse aircraft XML to get empty weight and CG
ac_xml_path = os.path.join(
    fdm.get_root_dir(), f"aircraft/{AIRCRAFT_NAME}/{AIRCRAFT_NAME}.xml"
)
ac_xml_root = ET.parse(ac_xml_path).getroot()

# Empty weight [lbs]
empty_weight = float(
```

(continues on next page)

(continued from previous page)

```
    ac_xml_root.find("mass_balance/emptywt").text
)

# Original CG x-location [inches from construction-axes origin]
x_cg_0 = float(
    ac_xml_root.find("mass_balance/location/x").text
)

# --- Payload and fuel ---
payload_0     = 15172 / 2      # half payload [lb]
fuelmax       = 8097.63        # Global 5000 max fuel [lb]
fuel_per_tank = fuelmax / 2    # each of the three tanks loaded to half its capacity

# --- Initial flight conditions ---
speed_cas = 250.0    # calibrated airspeed [kts]
h_ft_0    = 15000.0 # altitude [ft]
gamma_0   = 0.0      # flight-path angle [deg]

weight_0 = empty_weight + payload_0 + fuel_per_tank * 3

print(f"Empty weight        : {empty_weight:.0f} lb")
print(f"Total weight (est.) : {weight_0:.0f} lb")
print(f"CG x (original)     : {x_cg_0:.2f} in")
print(f"Trim altitude       : {h_ft_0:.0f} ft")
print(f"Trim CAS            : {speed_cas:.0f} kts")
```

```
Empty weight        : 48235 lb
Total weight (est.) : 67967 lb
CG x (original)     : 790.82 in
Trim altitude       : 15000 ft
Trim CAS            : 250 kts
```

5.4 4. Level-Flight Trim

Apply the initial conditions, start the engines, and ask JSBSim's built-in trimmer (`simulation/do_simple_trim`) to find the steady level-flight equilibrium. The resulting elevator position and throttle are saved as the trim reference.

[5]:
```
# Set engines running
fdm['propulsion/set-running'] = -1

# Initial conditions
fdm['ic/h-sl-ft']    = h_ft_0
fdm['ic/vc-kts']     = speed_cas
fdm['ic/gamma-deg'] = gamma_0

# Fuel load (three tanks)
fdm['propulsion/tank[0]/contents-lbs'] = fuel_per_tank
fdm['propulsion/tank[1]/contents-lbs'] = fuel_per_tank
fdm['propulsion/tank[2]/contents-lbs'] = fuel_per_tank

# Payload pointmass
fdm['inertia/pointmass-weight-lbs[0]'] = payload_0

# Initialise and trim
fdm.run_ic()
```

(continues on next page)

(continued from previous page)

```
fdm.run()
fdm['simulation/do_simple_trim'] = 1
fdm.run()

# Trim results
trim_vc_kts       = fdm['velocities/vc-kts']
trim_alpha_deg    = fdm['aero/alpha-deg']
trim_elev_rad     = fdm['fcs/elevator-pos-rad']
trim_elev_deg     = fdm['fcs/elevator-pos-deg']
trim_elev_norm    = fdm['fcs/elevator-pos-norm']
throttle_cmd_0    = fdm['fcs/throttle-cmd-norm']

print("Trim results:")
print(f"  CAS                : {trim_vc_kts:.2f} kts")
print(f"  Angle of attack    : {trim_alpha_deg:.4f} deg")
print(f"=========")
print(f"  Elevator (rad)     : {trim_elev_rad:.6f} rad")
print(f"  Elevator (deg)     : {trim_elev_deg:.4f} deg")
print(f"  Elevator (norm)    : {trim_elev_norm:.6f}")
print(f"=========")
print(f"  Elevator Autopilot Cmd (norm) : {fdm['ap/elevator_cmd']:.6f}")
print(f"  Elevator Trim Tab Cmd (norm) : {fdm['fcs/elevator-cmd-norm']:.6f}")
print(f"=========")
print(f"  Throttle cmd norm : {throttle_cmd_0:.6f}")
```

```
Trim results:
  CAS                : 250.00 kts
  Angle of attack    : 4.3176 deg
=========
  Elevator (rad)     : -0.051721 rad
  Elevator (deg)     : -2.9634 deg
  Elevator (norm)    : -0.147775
=========
  Elevator Autopilot Cmd (norm) : -0.000000
  Elevator Trim Tab Cmd (norm) : 0.000000
=========
  Throttle cmd norm : 0.772610
```

5.5 5. PathSim Block Diagram

The flight-dynamics model is embedded as a PathSim DynamicalFunction block whose **input** is the normalised elevator command and whose **output** is the angle of attack in degrees.

The total elevator command is the sum of two sources:

Block	Role
`srcElevatorAtTrim`	Constant source — elevator position at trim
`srcStepElevator`	`StepSource` — doublet perturbation (±10 % of full deflection)
`addedElevatorSignals`	`Adder` — sums the two signals
`dynFunAircraft`	`DynamicalFunction` — calls `fdm.run()` and returns α
`ampElevatorNormalized`	`DynamicalFunction` — scales command × 100 for display
`sco`	`Scope` — records both channels for plotting

The doublet shape: see code below

[6]:
```
from pathsim import Simulation, Connection
from pathsim.blocks import DynamicalFunction, Source, StepSource, Adder, Scope

# ------------------------------------------------------------------------
# Aircraft block: receives elevator command, advances JSBSim by one step,
# and returns the angle of attack.
# ------------------------------------------------------------------------
def f_aircraft(u, t):
    # Hold throttle at trim power to isolate the elevator response
    fdm['fcs/throttle-cmd-norm'] = throttle_cmd_0
    # PathSim passes u as a 1-D array; recent numpy rejects float() on non-0-d arrays,
    # so .flat[0] reliably extracts the scalar regardless of array shape.
    fdm['fcs/elevator-cmd-norm'] = float(np.asarray(u).flat[0])
    fdm.run()
    return pm.get_node('aero/alpha-deg').get_double_value()

dynFunAircraft = DynamicalFunction(f_aircraft)

# ------------------------------------------------------------------------
# Signal sources
# ------------------------------------------------------------------------

# Constant: normalised elevator position at trim
def f_elevator_command_at_trim(t):
    return 0 # already set to: trim_elev_norm ... Why?

srcElevatorAtTrim = Source(f_elevator_command_at_trim)

# Doublet: −0.10 at t=10 s, 0 at t=11 s, +0.10 at t=12 s, 0 at t=13 s
srcStepElevator = StepSource(
    amplitude=[-0.1, 0, 0.1, 0],
    tau=[10, 11, 12, 13]
)

# ------------------------------------------------------------------------
# Signal processing blocks
# ------------------------------------------------------------------------

# Sum trim command and doublet perturbation
addedElevatorSignals = Adder('++')

# Scale elevator command × 100 so both traces fit on a similar y-scale
def f_gain_100x(u, t):
    return 100.0 * u

ampElevatorNormalized = DynamicalFunction(f_gain_100x)

# Scope records two channels: scaled elevator command and alpha
sco = Scope(labels=["Elevator Command (% of full deflection)", "Angle of Attack (deg)"])

# ------------------------------------------------------------------------
# Assemble the block diagram
# ------------------------------------------------------------------------
blocks = [
    srcElevatorAtTrim,
    srcStepElevator,
    addedElevatorSignals,
```
(continues on next page)

(continued from previous page)

```
    dynFunAircraft,
    ampElevatorNormalized,
    sco,
]

connections = [
    Connection(srcElevatorAtTrim,      addedElevatorSignals[0]),
    Connection(srcStepElevator,        addedElevatorSignals[1]),
    Connection(addedElevatorSignals,   dynFunAircraft),
    Connection(addedElevatorSignals,   ampElevatorNormalized),
    Connection(ampElevatorNormalized,  sco[0]),
    Connection(dynFunAircraft,         sco[1]),
]

# -----------------------------------------------------------------------
# Run the simulation for 40 s with a JSBSim standard dt (60 Hz)
# -----------------------------------------------------------------------
sim = Simulation(blocks, connections,
                 dt=fdm.get_delta_t(),
                 log=True)

print("Running 40-second simulation . . . ")
sim.run(40.0)
print("Done.")
```

```
15:15:34 - INFO - LOGGING (log: True)
15:15:34 - INFO - BLOCKS (total: 6, dynamic: 0, static: 6, eventful: 1)
15:15:34 - INFO - GRAPH (nodes: 6, edges: 6, alg. depth: 3, loop depth: 0, runtime: 0.
↪083ms)
Running 40-second simulation . . .
15:15:34 - INFO - STARTING -> TRANSIENT (Duration: 40.00s)
15:15:34 - INFO - --------------------   1% | 0.0s<0.7s | 7178.7 it/s
15:15:35 - INFO - ####----------------  20% | 0.1s<0.2s | 18680.5 it/s
15:15:35 - INFO - ########------------  40% | 0.2s<0.1s | 19257.6 it/s
15:15:35 - INFO - ############--------  60% | 0.2s<0.1s | 15004.6 it/s
15:15:35 - INFO - ################----  80% | 0.3s<0.1s | 18916.7 it/s
15:15:35 - INFO - #################### 100% | 0.3s<--:-- | 10187.1 it/s
15:15:35 - INFO - FINISHED -> TRANSIENT (total steps: 4801, successful: 4801, runtime:␣
↪334.44 ms)
Done.
```

5.6 6. Simulation Results

The upper panel shows the normalised elevator command (scaled to percent of full deflection) and the lower panel shows the resulting angle-of-attack response.

[7]:
```
%matplotlib inline

t_rec   = sco.recording_time
data    = np.array(sco.recording_data[:])

label = (
    f"Weight {weight_0:.0f} lb, "
    f"Alt {h_ft_0/1000:.0f} kft, "
    f"CoG-x {x_cg_0:.2f} in"
)
```

(continues on next page)

(continued from previous page)

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(11, 6), sharex=True)

ax1.plot(t_rec, data[:, 0], color='steelblue', linewidth=1.5, label=label)
ax1.set_title(
    f"Normalised Elevator Command — trim altitude {h_ft_0/1000:.0f} kft"
)
ax1.set_ylabel('Elevator Command (% full deflection)')
ax1.set_ylim(-30, 30)
ax1.legend(loc='upper right', fontsize=8)

ax2.plot(t_rec, data[:, 1], color='tomato', linewidth=1.5, label=label)
ax2.set_title(
    f"Angle of Attack — trim altitude {h_ft_0/1000:.0f} kft"
)
ax2.set_xlabel('Time (s)')
ax2.set_ylabel('AoA (deg)')
ax2.set_ylim(-2, 10)
ax2.legend(loc='lower right', fontsize=8)

plt.tight_layout()
plt.show()
```

5.7 Summary

In this notebook you:

1. **Initialised** a JSBSim FDM for the Global 5000 business jet using the data bundled with the `jsbsim` Python package.
2. **Trimmed** the aircraft to steady level flight at 15 000 ft / 250 KCAS.
3. **Wrapped** the JSBSim step function inside a PathSim `DynamicalFunction` block — no additional library required.
4. **Built a block diagram** that superimposes a doublet elevator perturbation on the trim elevator command and records the angle-of-attack response.

5. **Plotted** the two-channel scope output.

Next: *05_pathsim_flight.ipynb* – using the higher-level `pathsim-flight` library (`JSBSimWrapper`, `ISAtmosphere`) to build a closed-loop altitude-hold autopilot.

Note

This page was generated from a Jupyter notebook.

CHAPTER

SIX

PATHSIM-FLIGHT: JSBSIM AS A PATHSIM BLOCK

pathsim-flight bridges JSBSim and PathSim. It provides:

Class	Description
JSBSimWrap	Wraps a JSBSim FDM as a discrete-time PathSim Wrapper block. Inputs are control surface commands; outputs are state properties.
ISAtmosphe	International Standard Atmosphere model: given geometric altitude and temperature offset, returns pressure, density, temperature, and speed of sound.
Airspeed utilities	CAStoTAS, TAStoEAS, MachtoCAS, …

6.1 Install

```
pip install pathsim
pip install git+https://github.com/pathsim/pathsim-flight.git
```

6.1.1 1. Imports

[1]:
```
# If running on Google Colab, install the required packages.

import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab \u2013 installing jsbsim, pathsim, pathsim-flight \
↪u2026')
    !pip install jsbsim pathsim pathsim-flight
```

[2]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import jsbsim
import pathsim
import pathsim_flight

from pathsim import Simulation, Connection
from pathsim.blocks import Scope, Constant
from pathsim_flight import JSBSimWrapper, ISAtmosphere
from pathsim_flight.utils.airspeed_conversions import CAStoTAS, TAStoCAS

matplotlib.rcParams.update({'figure.dpi': 120, 'axes.grid': True, 'grid.alpha': 0.4})
```

(continues on next page)

(continued from previous page)

```
# Suppress JSBSim console output globally
# jsbsim.FGJSBBase().debug_lvl = 0

print(f"JSBSim version        : {jsbsim.__version__}")
print(f"PathSim version       : {pathsim.__version__}")
print(f"pathsim-flight version: {pathsim_flight.__version__}")
```

```
JSBSim version        : 1.3.0
PathSim version       : 0.20.0
pathsim-flight version: 0.1.2.dev1+g904cf68e8
```

6.1.2 2. International Standard Atmosphere

`ISAtmosphere` is a PathSim `Function` block. Its two inputs are:

- **Port 0** – geometric altitude [m]
- **Port 1** – temperature offset from ISA standard day [K] (0 = standard day)

Its four outputs are pressure [Pa], density [kg/m³], temperature [K], and speed of sound [m/s].

[3]:
```
from pathsim.blocks import Source

# Sweep altitude from 0 to 11 000 m (tropopause)
alt_max_m = 11_000.0
altitudes_m = np.linspace(0, alt_max_m, 200)

isa = ISAtmosphere()
temp_offset = Constant(0.0)   # standard day

# We will evaluate standalone (not in a Simulation) for a quick sweep
pressures    = []
densities    = []
temperatures = []
speeds_of_sound = []

for alt in altitudes_m:
    result = isa._eval(alt, 0.0)
    pressures.append(result[0])
    densities.append(result[1])
    temperatures.append(result[2])
    speeds_of_sound.append(result[3])

fig, axes = plt.subplots(2, 2, figsize=(12, 8))
fig.suptitle('International Standard Atmosphere', fontsize=14)

axes[0, 0].plot(pressures, altitudes_m / 1000, color='steelblue')
axes[0, 0].set_xlabel('Pressure [Pa]')
axes[0, 0].set_ylabel('Altitude [km]')
axes[0, 0].set_title('Pressure')

axes[0, 1].plot(densities, altitudes_m / 1000, color='darkorange')
axes[0, 1].set_xlabel('Density [kg/m³]')
axes[0, 1].set_ylabel('Altitude [km]')
axes[0, 1].set_title('Density')

axes[1, 0].plot(np.array(temperatures) - 273.15, altitudes_m / 1000, color='mediumseagreen
```

(continues on next page)

(continued from previous page)

```
↪')
axes[1, 0].set_xlabel('Temperature [°C]')
axes[1, 0].set_ylabel('Altitude [km]')
axes[1, 0].set_title('Temperature')

axes[1, 1].plot(speeds_of_sound, altitudes_m / 1000, color='mediumpurple')
axes[1, 1].set_xlabel('Speed of Sound [m/s]')
axes[1, 1].set_ylabel('Altitude [km]')
axes[1, 1].set_title('Speed of Sound')

plt.tight_layout()
plt.savefig('isa_atmosphere.png', bbox_inches='tight')
plt.show()

print(f"Sea-level pressure     : {pressures[0]:.1f} Pa")
print(f"Sea-level temperature  : {temperatures[0] - 273.15:.2f} °C")
print(f"Sea-level speed of snd : {speeds_of_sound[0]:.3f} m/s")
```

```
Sea-level pressure     : 101325.0 Pa
Sea-level temperature  : 15.00 °C
Sea-level speed of snd : 340.294 m/s
```

6.1.3 3. Airspeed conversions

`pathsim_flight` provides utility functions to convert between CAS, TAS, EAS, and Mach.

[4]:
```
cas_kts = 120.0  # Calibrated Airspeed [kts]

altitudes_ft = [0, 5000, 10000, 20000, 30000, 40000]

print(f"{'Altitude [ft]':>16} {'CAS [kts]':>10} {'TAS [kts]':>10} {'TAS/CAS':>8}")
```

(continues on next page)

(continued from previous page)

```
print('-' * 50)
for alt_ft in altitudes_ft:
    alt_m = alt_ft * 0.3048
    p, rho, T, a = isa._eval(alt_m, 0.0)
    # CAS → TAS conversion (using sea-level standard density)
    rho_sl  = 1.225  # kg/m³ sea-level standard
    tas_kts = cas_kts * (rho_sl / rho) ** 0.5
    print(f"{alt_ft:>16} {cas_kts:>10.1f} {tas_kts:>10.1f} {tas_kts/cas_kts:>8.3f}")
```

```
   Altitude [ft]  CAS [kts]  TAS [kts]  TAS/CAS
--------------------------------------------------
               0      120.0      120.0    1.000
            5000      120.0      129.3    1.077
           10000      120.0      139.6    1.164
           20000      120.0      164.3    1.370
           30000      120.0      196.0    1.634
           40000      120.0      241.4    2.012
```

6.1.4 4. JSBSimWrapper – embedding JSBSim in a PathSim diagram

`JSBSimWrapper` wraps a JSBSim FDM as a discrete-time `Wrapper` block.

Key parameters:

Parameter	Description
T	Discrete-time step for JSBSim [s]
`input_properties`	List of JSBSim property names for block inputs
`output_properties`	List of JSBSim property names for block outputs
`aircraft_model`	Aircraft directory name (e.g. `'c172p'`)
`trim_airspeed`	KCAS for trim
`trim_altitude`	Altitude [ft] for trim

Below we build a **closed-loop pitch-hold autopilot** using:

- A `JSBSimWrapper` for the Cessna 172P dynamics.
- A `Constant` block to provide a pitch reference.
- A simple proportional controller (`Function` block).

[5]:
```
from pathsim.blocks import Function, Amplifier, Adder

# ---------- Parameters ----------
AIRCRAFT     = 'c172p'
ALT_FT       = 5000.0    # trim altitude [ft]
AIRSPEED_KTS = 90.0      # trim KCAS
SIM_DUR      = 30.0      # simulation duration [s]
DT           = 1 / 60    # JSBSim time step [s]
Kp           = 0.5       # proportional gain [elevator/rad]
PITCH_REF    = 3.0       # pitch reference [deg]
# --------------------------------

# Flight dynamics model (JSBSim inside a PathSim block)
aircraft_block = JSBSimWrapper(
    T=DT,
    input_properties=['fcs/elevator-cmd-norm'],        # elevator input
    output_properties=[
```

(continues on next page)

(continued from previous page)

```
        'attitude/theta-deg',     # pitch angle
        'position/h-sl-ft',       # altitude
        'velocities/vc-kts',      # calibrated airspeed
        'aero/alpha-deg',         # angle of attack
    ],
    aircraft_model=AIRCRAFT,
    trim_airspeed=AIRSPEED_KTS,
    trim_altitude=ALT_FT,
    trim_gamma=0.0,
)

# Reference pitch angle
pitch_ref = Constant(PITCH_REF)                        # deg

# Error = reference − actual pitch
pitch_error = Adder('+-')

# Proportional controller: elevator = Kp * error (deg → normalised)
controller = Amplifier(Kp / 90.0)    # /90 to stay roughly in [-1, 1]

# Record pitch, altitude, and airspeed
scope_flight = Scope(labels=['pitch_deg', 'alt_ft', 'vc_kts', 'alpha_deg'])
scope_ctrl   = Scope(labels=['elevator_cmd', 'pitch_error'])

sim_ap = Simulation(
    blocks=[pitch_ref, aircraft_block, pitch_error, controller, scope_flight, scope_ctrl],
    connections=[
        # Autopilot loop
        Connection(pitch_ref, pitch_error, scope_ctrl[1]),      # reference → error adder
        Connection(aircraft_block[0], pitch_error[1]),           # actual pitch → error␣
↪adder
        Connection(pitch_error, controller, scope_ctrl[0]),      # error → gain
        Connection(controller, aircraft_block),                  # elevator command → FDM
        # Scopes
        Connection(aircraft_block[0], scope_flight[0]),          # pitch
        Connection(aircraft_block[1], scope_flight[1]),          # altitude
        Connection(aircraft_block[2], scope_flight[2]),          # airspeed
        Connection(aircraft_block[3], scope_flight[3]),          # AoA
    ],
    dt=DT
)

sim_ap.run(SIM_DUR)

print("Autopilot simulation complete")
```

```
15:15:51 - INFO - LOGGING (log: True)
15:15:51 - INFO - BLOCKS (total: 6, dynamic: 0, static: 6, eventful: 1)
15:15:51 - INFO - GRAPH (nodes: 6, edges: 10, alg. depth: 1, loop depth: 3, runtime: 0.
↪102ms)
15:15:51 - INFO - STARTING -> TRANSIENT (Duration: 30.00s)
15:15:51 - INFO - --------------------   1% | 0.0s<0.2s | 11112.0 it/s
15:15:51 - INFO - ####----------------  20% | 0.1s<0.3s | 5338.6 it/s
15:15:51 - INFO - ########------------  40% | 0.1s<0.1s | 8345.0 it/s
15:15:52 - INFO - ############--------  60% | 0.2s<0.2s | 4453.2 it/s
15:15:52 - INFO - ################----  80% | 0.5s<0.1s | 5842.9 it/s
```

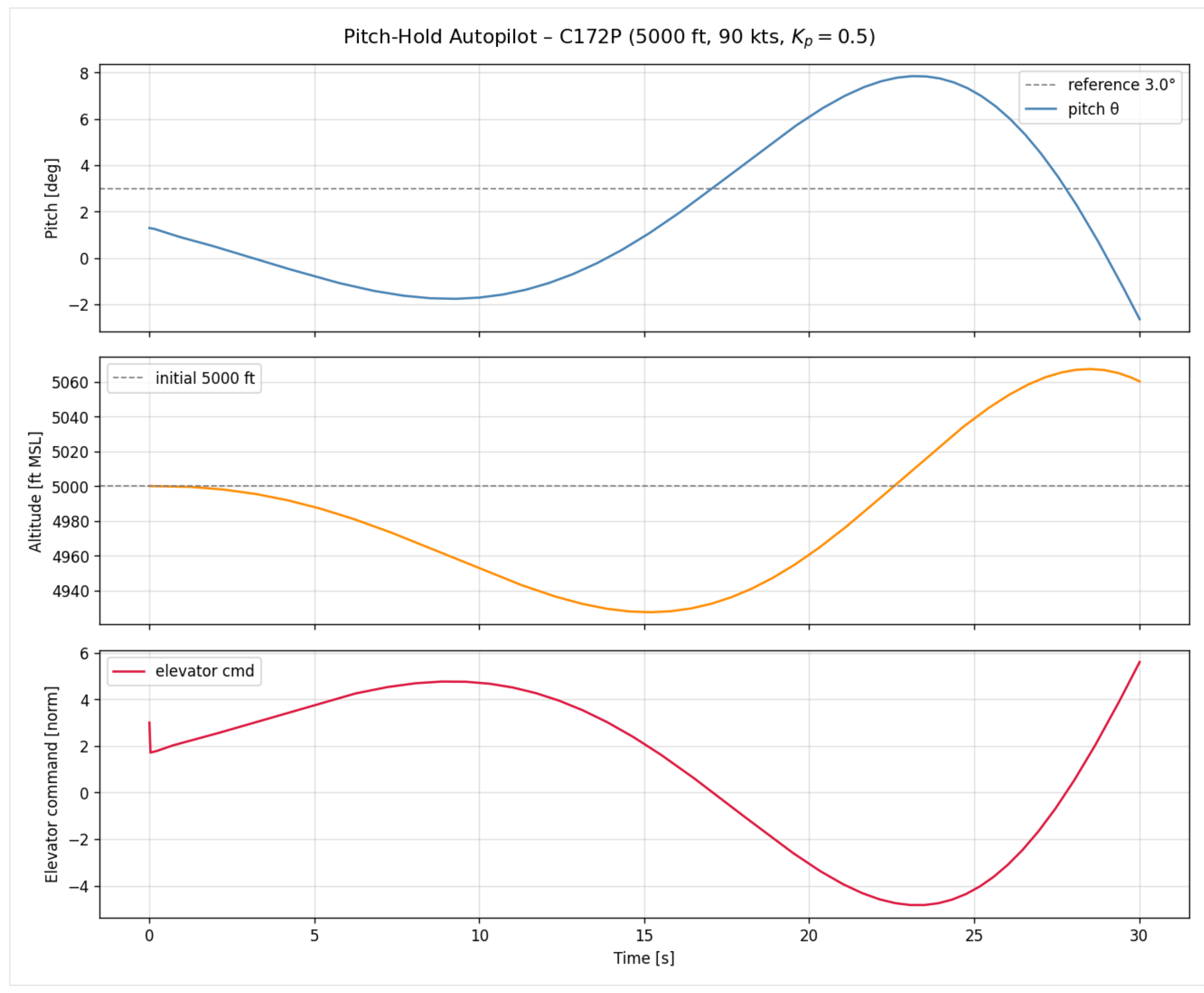
(continues on next page)

(continued from previous page)

```
15:15:52 - INFO - #################### 100% | 0.6s<--:-- | 11092.7 it/s
15:15:52 - INFO - FINISHED -> TRANSIENT (total steps: 1801, successful: 1801, runtime:␣
↪570.84 ms)
Autopilot simulation complete
```

6.1.5 5. Plot the results

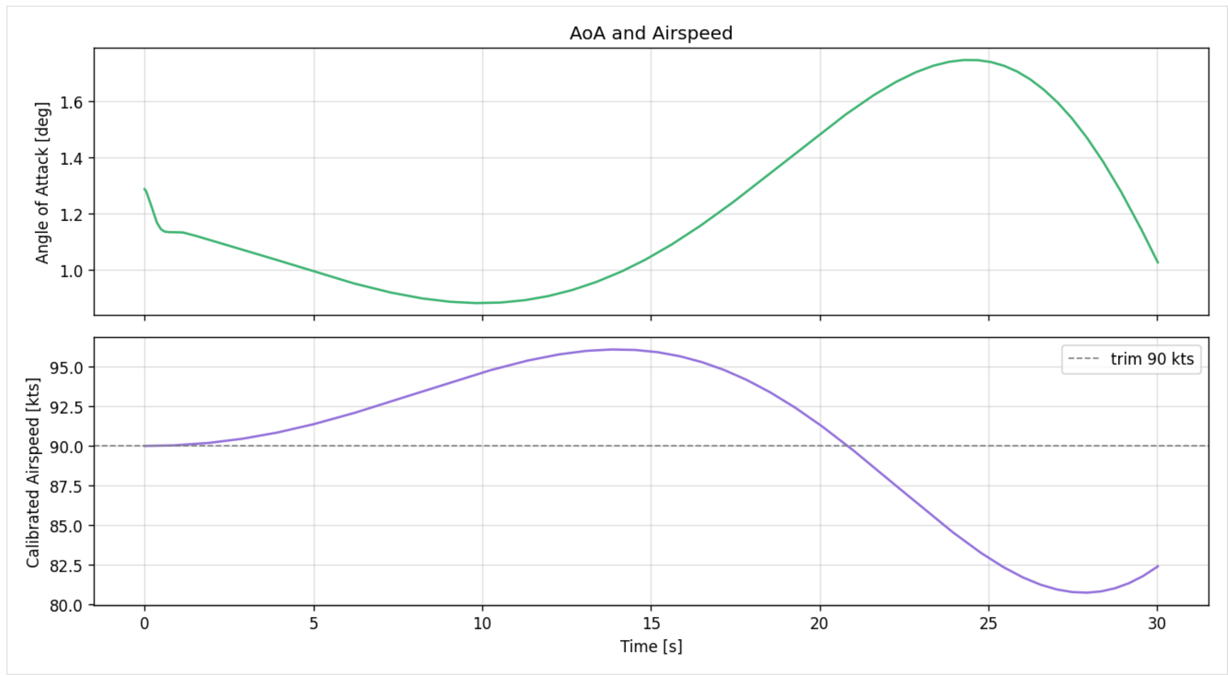
[6]:
```
t_ap, flight_data = scope_flight.read()
pitch_ap, alt_ap, vc_ap, aoa_ap = flight_data[0], flight_data[1], flight_data[2], flight_
↪data[3]
t_ctrl, ctrl_data = scope_ctrl.read()
elev_ap, err_ap = ctrl_data[0], ctrl_data[1]

fig, axes = plt.subplots(3, 1, figsize=(11, 9), sharex=True)
fig.suptitle(
    f'Pitch-Hold Autopilot - {AIRCRAFT.upper()} '
    f'({ALT_FT:.0f} ft, {AIRSPEED_KTS:.0f} kts, $K_p={Kp}$)',
    fontsize=13
)

axes[0].axhline(PITCH_REF, color='grey', linestyle='--', linewidth=1.0,
                label=f'reference {PITCH_REF}°')
axes[0].plot(t_ap, pitch_ap, color='steelblue', linewidth=1.5, label='pitch θ')
axes[0].set_ylabel('Pitch [deg]')
axes[0].legend()

axes[1].plot(t_ap, alt_ap, color='darkorange', linewidth=1.5)
axes[1].axhline(ALT_FT, color='grey', linestyle='--', linewidth=1.0, label=f'initial {ALT_
↪FT:.0f} ft')
axes[1].set_ylabel('Altitude [ft MSL]')
axes[1].legend()

axes[2].plot(t_ctrl, elev_ap, color='crimson', linewidth=1.5, label='elevator cmd')
axes[2].set_ylabel('Elevator command [norm]')
axes[2].set_xlabel('Time [s]')
axes[2].legend()

plt.tight_layout()
plt.savefig('autopilot_pitch_hold.png', bbox_inches='tight')
plt.show()
```



6.1.6 6. Angle-of-attack and airspeed

[7]:

```
fig, axes = plt.subplots(2, 1, figsize=(11, 6), sharex=True)

axes[0].plot(t_ap, aoa_ap, color='mediumseagreen', linewidth=1.5)
axes[0].set_ylabel('Angle of Attack [deg]')
axes[0].set_title('AoA and Airspeed')

axes[1].plot(t_ap, vc_ap, color='mediumpurple', linewidth=1.5)
axes[1].axhline(AIRSPEED_KTS, color='grey', linestyle='--', linewidth=1.0,
                label=f'trim {AIRSPEED_KTS:.0f} kts')
axes[1].set_ylabel('Calibrated Airspeed [kts]')
axes[1].set_xlabel('Time [s]')
axes[1].legend()

plt.tight_layout()
plt.savefig('autopilot_aoa_airspeed.png', bbox_inches='tight')
plt.show()
```



6.1.7 Summary

In this notebook you:

- Used `ISAtmosphere` to plot ISA profiles from sea level to the tropopause.
- Demonstrated airspeed-conversion utilities from `pathsim_flight`.
- Built a **closed-loop pitch-hold autopilot** using `JSBSimWrapper` embedded in a PathSim block diagram alongside a proportional controller.

The same pattern can be extended to:

- Multi-axis controllers (pitch + roll + yaw).
- Altitude-hold and airspeed-hold outer loops.
- Hardware-in-the-loop (HIL) testing by replacing the `JSBSimWrapper` with real sensor data.

Note

This page was generated from a Jupyter notebook.

CHAPTER

SEVEN

ANGLE OF ATTACK (AOA) VS CALIBRATED AIRSPEED (CAS) OF A GLOBAL 5000 AIRCRAFT IN WINGS-LEVEL FLIGHT

Credits:**Sean McLeod** (Computer Software Consultant and Professional)**Guilherme Araujo Lima da Silva** (CEO @ ATS4i Aerothermal Solutions)**Agostino De Marco** (Prof. Flight Mechanics, Univ. Naples Federico II, Italy)

The Bombardier Global Express is a large, long-range business jet produced by Bombardier Aerospace. Introduced in the late 1990s, it is known for its exceptional range, speed, and luxurious cabin, making it popular among corporate and VIP travelers. The Global Express can fly intercontinental distances without refueling, connecting cities such as New York and Tokyo or Paris and Singapore nonstop. Its advanced avionics, high cruise speed, and spacious, comfortable interior set a benchmark in the business aviation market. The aircraft has also served as the basis for several variants, including the Global 5000 and Global 6000.

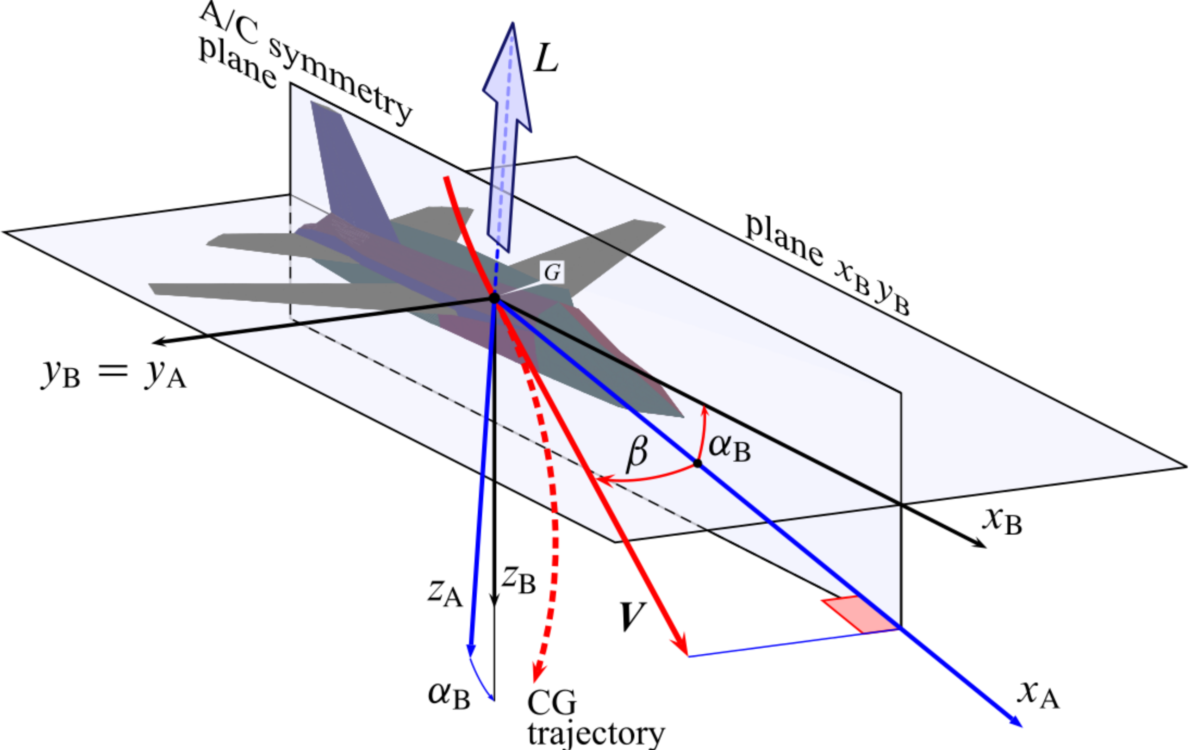
See this Wikipedia page https://en.wikipedia.org/wiki/Bombardier_Global_Express to learn more.

Bombardier Global Express in flight (Photo: Wikipedia).

Bombardier Global 5000 (Photo: Wikipedia).

7.1 Plotting trim angles of attack vs flight speed

Consider the following generic flight conditions:



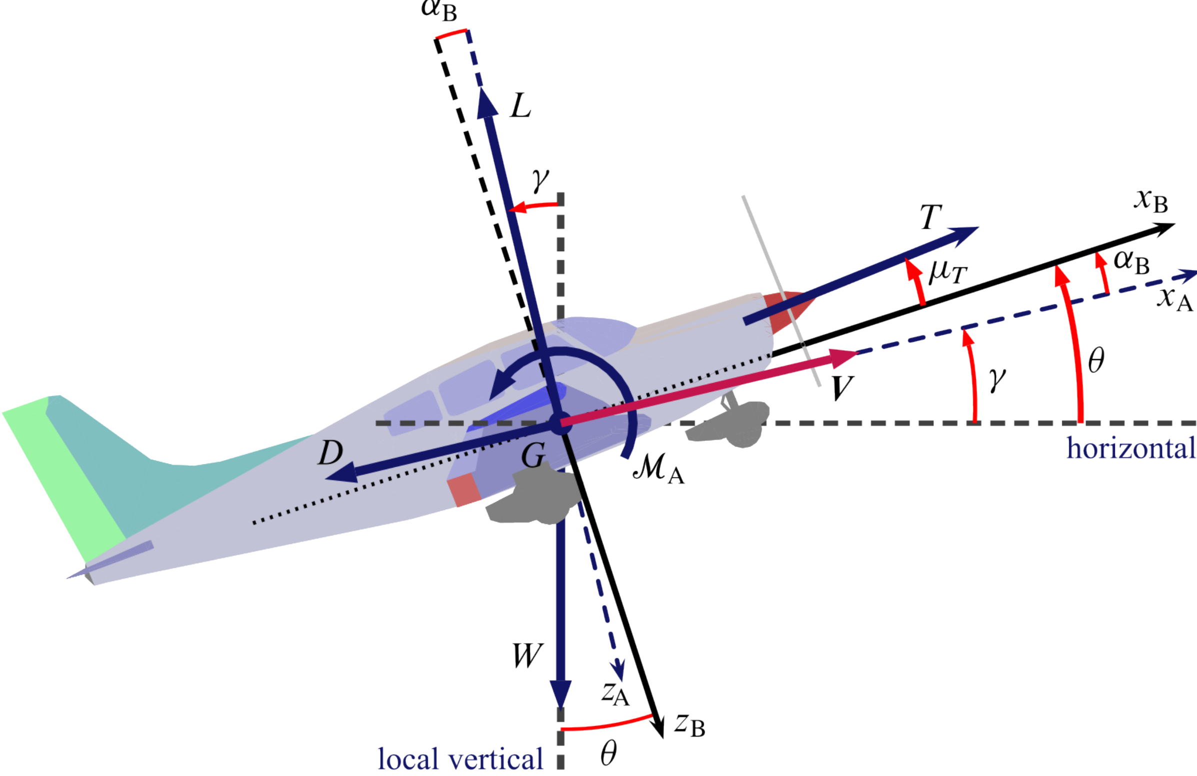
Aerodynamic axes (From JSBSim reference manual).

We are interested in finding particular cases of the above condition, where:

- wings are not banked, i.e. roll angle $\phi = 0$ deg,
- the flight trajectory is a straight line,
- the altitude is constant, i.e. the trajectory is a straight, horizontal line (FPA, Flight Path Angle $\gamma_0 = 0$ deg),
- the aircraft translates, i.e. the angular rates (p, q, r) are constant and zero,
- the True Air Speed V (TAS) is constant, i.e. the motion is unaccelerated (that is, equilibrated),
- the sideslip angle β is zero (that is, symmetrical aerodynamics).

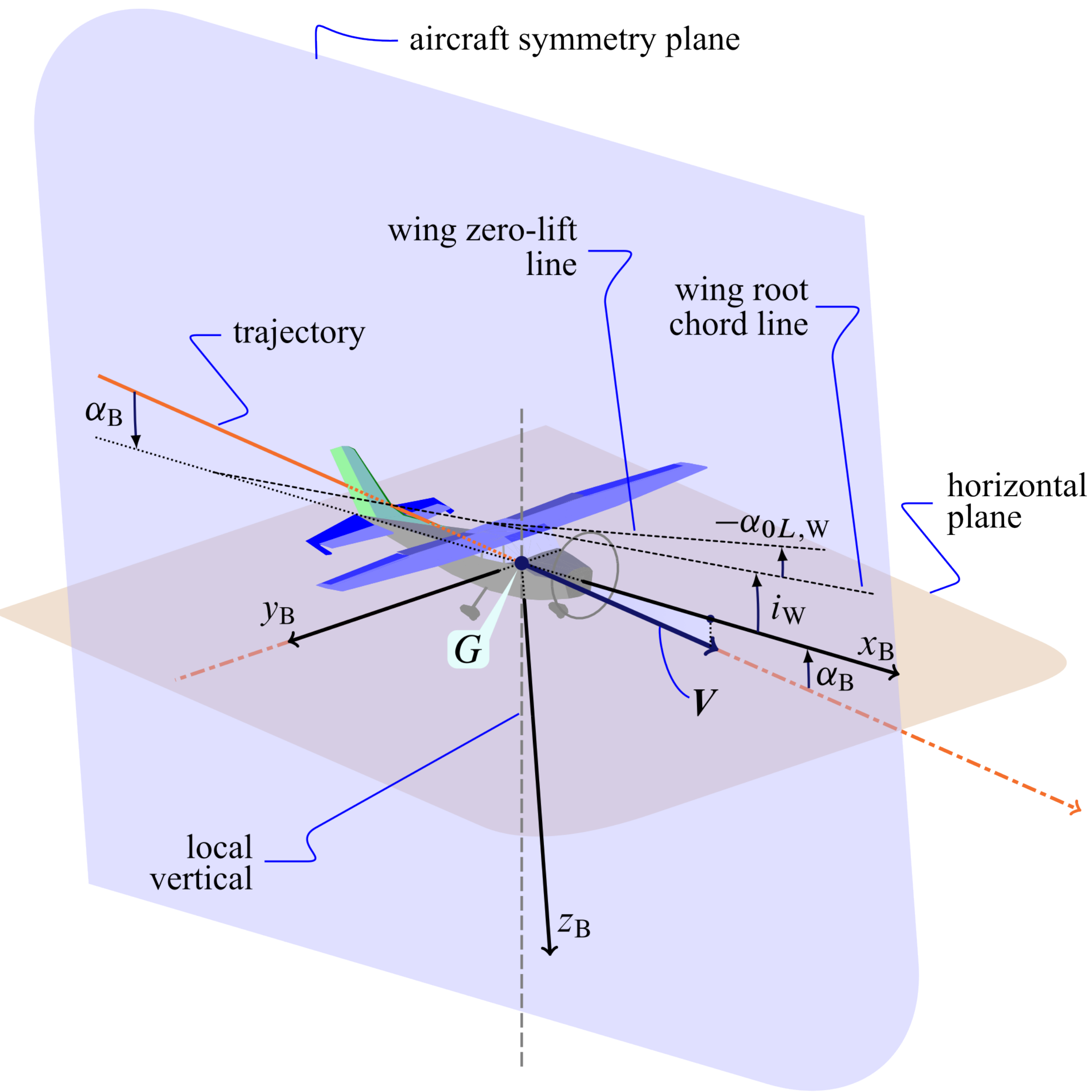
These are the *trimmed flight* conditions in level flight.

An aircraft in trimmed flight with a non-zero flight path angle γ is shown below:



Aircraft in straight, wings-level, climbing flight.

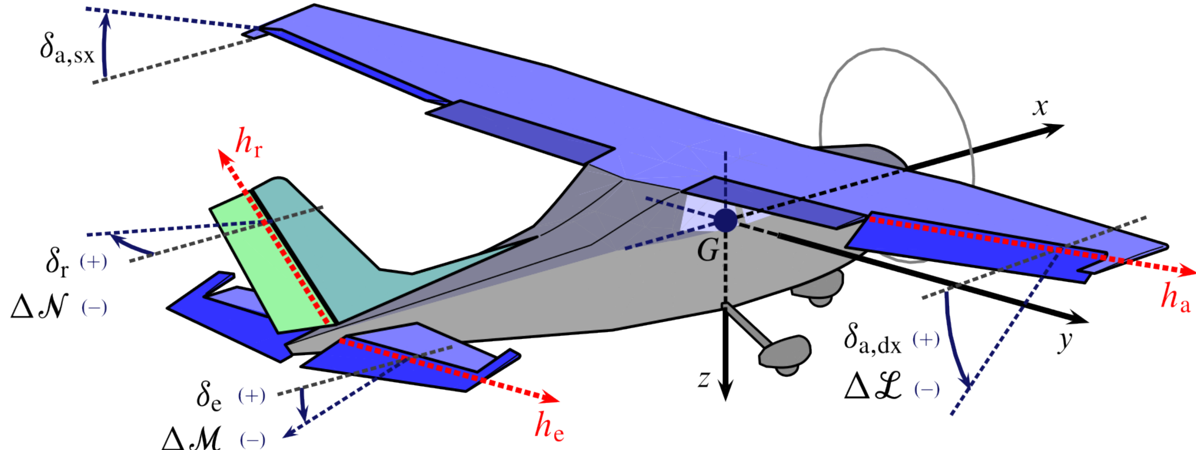
An aircraft in wings-level, constant altitude (zero flight path angle γ), trimmed flight is shown below:



Aircraft in straight, wings-level, constant altitude flight.

For a given flight altitude h_0, for a given aircraft weight W_0, there are certainly a minimum speed V_{min} and a maximum V_{max} the aircraft can fly in the above trimmed conditions. Now, for all V_0 within a speed interval $[V_1, V_2] \subset [V_{min}, V_{max}]$, we want to plot the required flight angle of attack α_B (AoA) and the required (normalized) elevator deflection.

See next image introducing aerosurface deflection angles and their standard symbols.



Aircraft standard aerosurface deflections.

7.1.1 Setting up a simulation script

[1]:
```
import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab - installing jsbsim . . . ')
    !pip install jsbsim

PATH_TO_JSBSIM_FILES = None
```

7.2 Initialize FDM

[2]:
```
import jsbsim

# --- JSBSim Initialization ---
# These lines initialize the flight dynamics model.

# Create a flight dynamics model (FDM) instance.
fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)

fdm.set_debug_level(0) # Suppress verbose JSBSim console output

if fdm is not None:
    print("FDM created successfully")
    fdm.set_debug_level(0) # Suppress verbose JSBSim console output
else:
    print("Failed to create FDM")
```

```


     JSBSim Flight Dynamics Model v1.3.0 Apr  9 2026 10:00:08
            [JSBSim-ML v2.0]

JSBSim startup beginning . . .

FDM created successfully
```

7.3 Tweak aircraft XML file: remove `<output name="..."/>` nodes from the officially released files

[3]:
```
import os
import xml.etree.ElementTree as ET
import matplotlib.pyplot as plt

# Global variables that must be modified to match your particular need
# The aircraft name
# Note - It should match the exact spelling of the model file
AIRCRAFT_NAME="global5000"

ac_xml_file_path = os.path.join(fdm.get_root_dir(), f'aircraft/{AIRCRAFT_NAME}/{AIRCRAFT_
↪NAME}.xml')
print(f"Aircraft original XML file: {ac_xml_file_path}")

print("Parsing XML...")
ac_xml_tree = ET.parse(ac_xml_file_path)

# Save a copy of the original XML file for backup
backup_xml_file_path = ac_xml_file_path.replace('.xml', '_backup.xml')
print(f"Saving backup XML file: {backup_xml_file_path}")
ac_xml_tree.write(backup_xml_file_path)

# Traverse the XML tree and remove all <output ... /> elements with name attribute
ac_xml_root = ac_xml_tree.getroot()
for x in ac_xml_root.findall('output'):
    has_port = x.get('name') is not None
    if has_port:
        print(f"\tRemoving <output ... /> with name: {x.get('name')}")
        ac_xml_root.remove(x)

print("Saving modified XML...")
ac_xml_tree.write(ac_xml_file_path)

print("No CSV file will be generated since all <output ... /> elements with name␣
↪attribute have been removed from the XML file.")
```

```
Aircraft original XML file: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/
↪aircraft/global5000/global5000.xml
Parsing XML. . .
Saving backup XML file: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/aircraft/
↪global5000/global5000_backup.xml
Saving modified XML. . .
No CSV file will be generated since all <output . . .  /> elements with name attribute have␣
↪been removed from the XML file.
```

7.3.1 Run a simulation for a single altitude, single weight, single CoG position

[4]:
```
%matplotlib inline
# Fuel max for Global5000, lbm
fuelmax = 8097.63

# Instantiate the FDMExec object and load the aircraft
fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)
fdm.set_debug_level(0) # Suppress verbose JSBSim console output
```

(continues on next page)

(continued from previous page)

```
fdm.load_model(AIRCRAFT_NAME)
fdm.disable_output()  # Prevent JSBSim from trying to open/write model CSV output files

ac_xml_tree = ET.parse(os.path.join(fdm.get_root_dir(), f'aircraft/{AIRCRAFT_NAME}/
↪{AIRCRAFT_NAME}.xml'))
ac_xml_root = ac_xml_tree.getroot()

# Get the empty weight from aircraft xml [assume lbs]
for x in ac_xml_root.findall('mass_balance'):
        w = x.find('emptywt').text

empty_weight = float(w)

# Get the original CG from aircraft xml, assum inches from Construction axes origin
for loc in ac_xml_root.findall('mass_balance/location'):
        x_cg_ = loc.find('x').text

x_cg_0 = float(x_cg_)

# Assume a payload, midweight, lb
payload_0 = 15172/2

# Assume the mass of fuel, half tanks, lb
fuel_per_tank = fuelmax/2

# Assume a flight altitude
h_ft_0 = 15000

weight_0 = empty_weight + payload_0 + fuel_per_tank*3

# Assume a zero flight path angle, gamma_0
gamma_0 = 0

print("-----------------------------------------")
print("Altitude {} ft, Weight {} lb, CoG-x {} in".format(h_ft_0, weight_0, x_cg_0))
print("-----------------------------------------")

# Run the simulation varying speed
speed_cas_1 = 90
speed_cas_2 = 550

print("Running simulation in the CAS range {} to {} kts".format(speed_cas_1, speed_cas_2))

# Set engines running
fdm['propulsion/set-running'] = -1

results = []
results_fcs = []

# Prepare two stacked subplots (AoA and elevator deflection)
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 6), sharex=True)

# Run for different speeds
for speed in range(speed_cas_1, speed_cas_2, 10):
    fdm['ic/h-sl-ft'] = h_ft_0
    fdm['ic/vc-kts'] = speed
```

(continues on next page)

(continued from previous page)

```
    fdm['ic/gamma-deg'] = gamma_0
    fdm['propulsion/tank[0]/contents-lbs'] = fuel_per_tank
    fdm['propulsion/tank[1]/contents-lbs'] = fuel_per_tank
    fdm['propulsion/tank[2]/contents-lbs'] = fuel_per_tank
    fdm['inertia/pointmass-weight-lbs[0]'] = payload_0

    # Initialize the aircraft with initial conditions
    fdm.run_ic()
    # Run fdm model
    fdm.run()

    # Trim
    try:
        fdm['simulation/do_simple_trim'] = 1
        results.append((fdm['velocities/vc-kts'], fdm['aero/alpha-deg']))
        results_fcs.append((fdm['velocities/vc-kts'], fdm['fcs/elevator-pos-rad'], fdm[
↪'fcs/elevator-pos-norm']))
    except jsbsim.TrimFailureError:
        print("\n\nTrim failed, continuing with other configurations...\n\n")
        pass  # Ignore trim failure exceptions

# Plot results
speed, alpha = zip(*results)
speed, dElev_rad, dElev_norm = zip(*results_fcs)
dElev_deg = [d * 180 / 3.14159 for d in dElev_rad]

# Plot AoA vs KCAS
ax1.plot(speed, alpha,
         label="Weight {0:.0f} lb, Altitude {1:.0f} kft, CoG-x {2:.2f} in".format(weight_
↪0, h_ft_0/1000, x_cg_0),
         marker='o', linestyle='-', color='blue')

ax1.set_title("Trimmed flight conditions (FPA = {0:.0f} deg)".format(gamma_0))
ax1.set_ylabel('Angle of Attack (deg)')
ax1.set_ylim(-5, 15)
ax1.legend(loc='upper right')

# Plot elevator deflection vs KCAS
ax2.plot(speed, dElev_deg,
         label="Weight {0:.0f} lb, Altitude {1:.0f} kft, CoG-x {2:.2f} in".format(weight_
↪0, h_ft_0/1000, x_cg_0),
         marker='o', linestyle='-', color='red')

#ax2.set_title("Trimmed flight conditions (FPA = {0:.0f} deg) - Elevator Deflection vs␣
↪KCAS".format(gamma_0))
ax2.set_ylabel("Elevator Deflection (deg)")
ax2.set_ylim(-15, 10)
ax2.legend(loc="upper right")

# Plot normalized elevator deflection vs KCAS
ax3.plot(speed, dElev_norm,
         label="Weight {0:.0f} lb, Altitude {1:.0f} kft, CoG-x {2:.2f} in".format(weight_
↪0, h_ft_0/1000, x_cg_0),
         marker='o', linestyle='-', color='green')

#ax3.set_title("Trimmed flight conditions (FPA = {0:.0f} deg) - Normalized elevator␣
```

(continues on next page)

(continued from previous page)

```
↪deflection vs KCAS".format(gamma_0))
ax3.legend(frameon=False)
ax3.set_ylabel("Normalized elevator\n deflection (-)")
ax3.set_xlabel("KCAS (kt)")

plt.tight_layout()
plt.show()
```

```
----------------------------------------
Altitude 15000 ft, Weight 67967.445 lb, CoG-x 790.82 in
----------------------------------------
Running simulation in the CAS range 90 to 550 kts
  Sorry, wdot doesn't appear to be trimmable


Trim failed, continuing with other configurations. . .


  Sorry, wdot doesn't appear to be trimmable


Trim failed, continuing with other configurations. . .


  Sorry, wdot doesn't appear to be trimmable


Trim failed, continuing with other configurations. . .


  Sorry, wdot doesn't appear to be trimmable


Trim failed, continuing with other configurations. . .


  Sorry, wdot doesn't appear to be trimmable


Trim failed, continuing with other configurations. . .


  Sorry, wdot doesn't appear to be trimmable


Trim failed, continuing with other configurations. . .


  Sorry, udot doesn't appear to be trimmable


Trim failed, continuing with other configurations. . .


  Sorry, udot doesn't appear to be trimmable
```

(continues on next page)

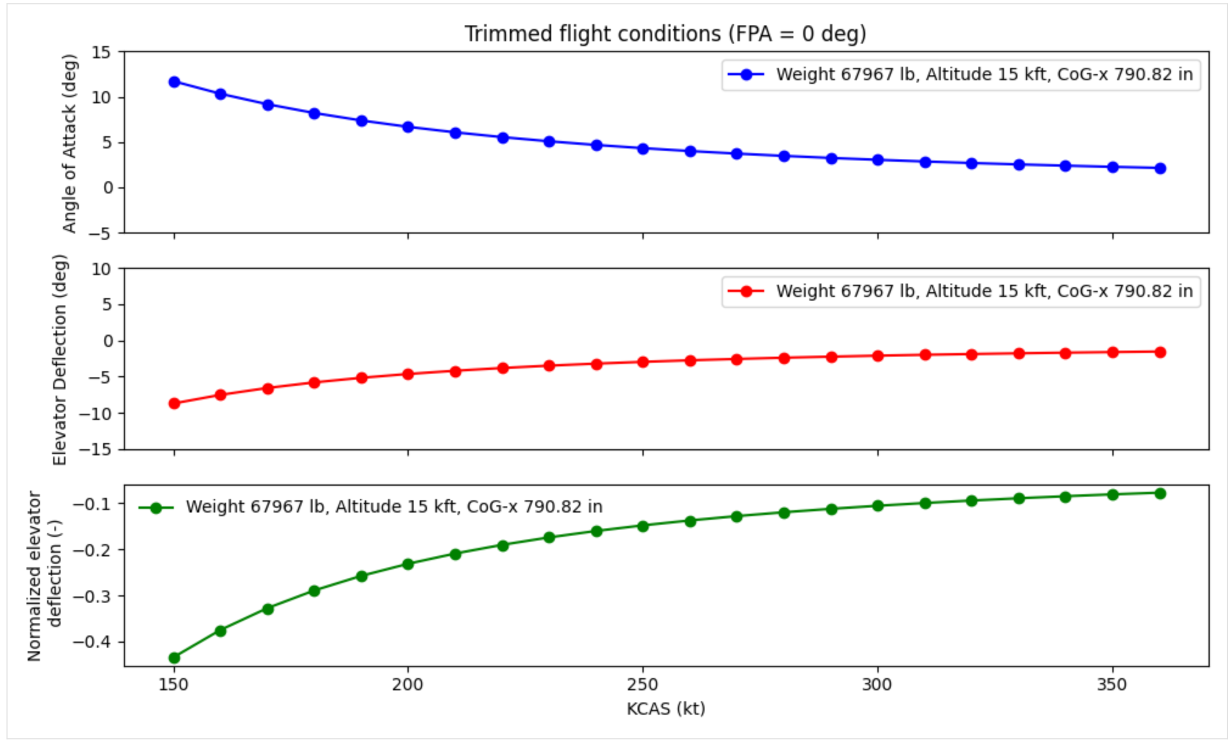
(continued from previous page)

```
Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations...
```

(continues on next page)

(continued from previous page)

```
  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations. . .

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations. . .

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations. . .

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations. . .

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations. . .

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations. . .

  Sorry, udot doesn't appear to be trimmable

Trim failed, continuing with other configurations. . .
```



7.4 Different aircraft weights, CoG positions and altitudes

Plots the variation in AoA versus CAS for level trim for different aircraft weights, cg and altitude.

Calculation required for aircraft icing engineering.

```
[5]: # Control whether to print results to console in addition to generating a plot
print_output = False

# Function to change CG in aircraft xml. Change the directory to the aircraft to be␣
↪studied.
# Note - Moments of inertia are not updated.
# Input x_cg is a float. Returns a float.
def changeCG(fdm, x_cg, readOnly):
    tree = ET.parse(os.path.join(fdm.get_root_dir(), f'aircraft/{AIRCRAFT_NAME}/{AIRCRAFT_
↪NAME}.xml'))
    root = tree.getroot()

    for loc in root.findall('mass_balance/location'):
        x = loc.find('x').text
        if not readOnly:
             loc.find('x').text = str(x_cg) # replace with the new value
             # Save the modified XML back to the file
             tree.write(os.path.join(fdm.get_root_dir(), f'aircraft/{AIRCRAFT_NAME}/
↪{AIRCRAFT_NAME}.xml'))
             # return the same value
             x = str(x_cg)
    return float(x)

# Fuel max for Global5000
fuelmax = 8097.63

# Assume a zero flight path angle, gamma_0
```

(continues on next page)

(continued from previous page)

```
gamma_0 = 0

# Prepare subplots to overlay plots
# Prepare two stacked subplots (AoA and elevator deflection)
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 12), sharex=True)

# Define here the payloads to be studied
payload = [1500, 15172/2, 15172]

# Define here the mass of fuel (per tank)
fuel = [1000, fuelmax/2, fuelmax]

# Three cases for weight
weight = ["light", "mid", "heavy"]

# Marker settings for each weight case: (marker, fillstyle)
# i=0 (light): hollow square, i=1 (mid): solid circle, i=2 (heavy): triangle up
marker_settings = (
    ('s', 'none'),   # light: hollow square
    ('o', 'full'),   # mid: solid circle
    ('^', 'full'),   # heavy: triangle up
)

fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)
fdm.load_model(f'{AIRCRAFT_NAME}')
fdm.disable_output()  # Prevent JSBSim from trying to open/write model CSV output files

# Get the original CG from aircraft xml
x_cg_Orig = changeCG(fdm, 0, True)

# Vary CG in the study
xs_cg = [x_cg_Orig*0.95, x_cg_Orig*1.05]

# Vary altitude
h_ft = [8000, 30000]

# Linestyle settings for each altitude case:
# j=0 (8000 ft): densely dashed (5 pts on, 1 pt off), j=1 (30000 ft): solid
linestyle_settings = ((0, (5, 1)), 'solid')

# Run the simulation varying CG, altitude, speed and total weight
fdm = []
# Run for different CG's
for j in range(2):
    fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)
    fdm.load_model(f'{AIRCRAFT_NAME}')
    fdm.disable_output()  # Prevent JSBSim from trying to open/write model CSV output␣
↪files

    # Set engines running
    fdm['propulsion/set-running'] = -1

    # Change CoG-x in the aircraft xml
    xcg_ = changeCG(fdm, xs_cg[j], False)

    # Run for different weights
```

(continues on next page)

(continued from previous page)

```
    for i in range(3):
        results = []
        results_fcs = []

        # Run for different speeds
        for speed in range(90, 550, 10):
            fdm['ic/h-sl-ft'] = h_ft[j]
            fdm['ic/vc-kts'] = speed
            fdm['ic/gamma-deg'] = gamma_0
            fdm['propulsion/tank[0]/contents-lbs'] = fuel[i]
            fdm['propulsion/tank[1]/contents-lbs'] = fuel[i]
            fdm['propulsion/tank[2]/contents-lbs'] = fuel[i]
            fdm['inertia/pointmass-weight-lbs[0]'] = payload[i]

            # Initialize the aircraft with initial conditions
            fdm.run_ic()
            # Run fdm model
            fdm.run()

            # Trim
            try:
                fdm['simulation/do_simple_trim'] = 1
                results.append((fdm['velocities/vc-kts'], fdm['aero/alpha-deg']))
                results_fcs.append((fdm['velocities/vc-kts'], fdm['fcs/elevator-pos-rad'],
↪ fdm['fcs/elevator-pos-norm']))
            except jsbsim.TrimFailureError:
                pass  # Ignore trim failure exceptions

        if print_output:
            print("------------------------------------------")
            print("Altitude {} - Weight {} - CG {}".format(h_ft[j], weight[i], xs_cg[j]))
            print("------------------------------------------")
            print("KCAS, AoA (deg)")
            for result in results:
                print(result[0], result[1])

            print("---------")
            print("KCAS, Elevator deflection (rad), Normalized elevator deflection (-1 to␣
↪1)")
            for result_fcs in results_fcs:
                print(result_fcs[0], result_fcs[1], result_fcs[2])

        speed, alpha = zip(*results)
        ax1.plot(speed, alpha,
                 label="{0} weight, {1:.0f} kft, {2:.2f} % CoG".format(weight[i], h_ft[j]/
↪1000, (float(xs_cg[j])/float(x_cg_Orig)-1)*100),
                 marker=marker_settings[i][0], fillstyle=marker_settings[i][1],␣
↪linestyle=linestyle_settings[j])

        speed, dElev_rad, dElev_norm = zip(*results_fcs)
        # Convert elevator deflection from radians to degrees
        dElev_deg = [d * 180 / 3.14159 for d in dElev_rad]

        ax2.plot(speed, dElev_deg,
                 label="{0} weight, {1:.0f} kft, {2:.2f} % cg".format(weight[i], h_ft[j]/
↪1000, (float(xs_cg[j])/float(x_cg_Orig)-1)*100),
```

(continues on next page)

(continued from previous page)

```
                  marker=marker_settings[i][0], fillstyle=marker_settings[i][1],␣
↪linestyle=linestyle_settings[j])

         ax3.plot(speed, dElev_norm,
                  label="{0} weight, {1:.0f} kft, {2:.2f} % cg".format(weight[i], h_ft[j]/
↪1000, (float(xs_cg[j])/float(x_cg_Orig)-1)*100),
                  marker=marker_settings[i][0], fillstyle=marker_settings[i][1],␣
↪linestyle=linestyle_settings[j])

# Restore original CG for the aircraft xml
x_cg__ = changeCG(fdm," {:.2f} ".format(x_cg_Orig), False)

print("-----------------------------------------")
print("Restored CoG-x {} in".format(x_cg__))
print("-----------------------------------------")

# Plot final results

ax1.set_title("Trimmed flight conditions (FPA = {0:.0f} deg)".format(gamma_0))
ax1.set_ylabel("Angle of Attack (deg)")
ax1.legend(frameon=False)
ax1.legend(loc='upper right')
ax1.set_ylim(0, 15)

ax2.set_ylabel('Elevator Deflection (deg)')
ax2.legend(frameon=False)
ax2.legend(loc='lower right')
ax2.set_ylim(-20, 0)

ax3.set_ylabel("Normalized elevator deflection (-)")
ax3.set_xlabel("KCAS (kt)")
ax3.legend(frameon=False)
ax3.legend(loc='lower right')
ax3.set_ylim(-1.0, 0)

plt.show()
```

```
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
```

(continues on next page)

(continued from previous page)

```
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, qdot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
```

(continues on next page)

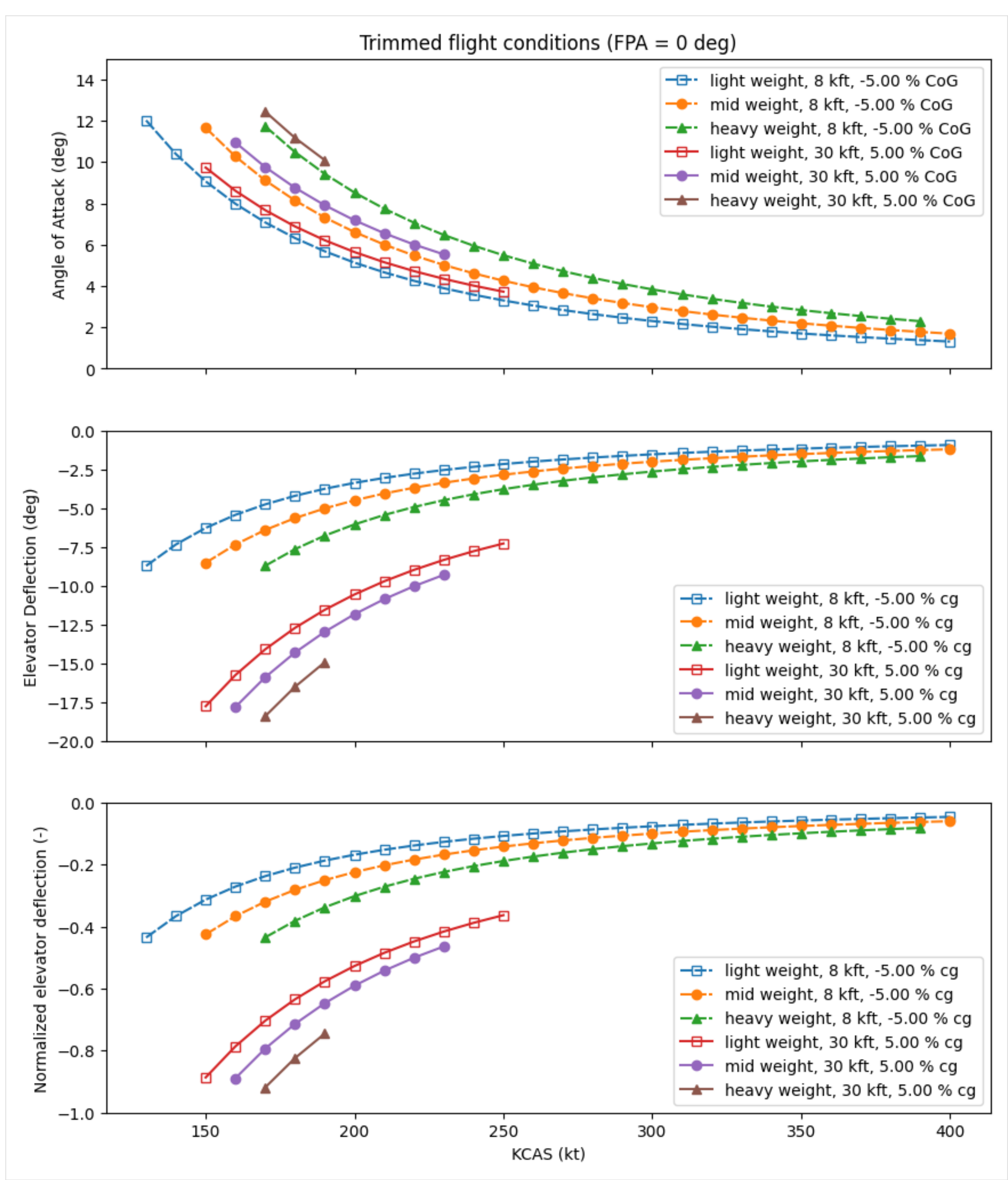
(continued from previous page)

```
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, qdot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
```

(continues on next page)

(continued from previous page)

```
  Sorry, udot doesn't appear to be trimmable
  Sorry, qdot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
  Sorry, udot doesn't appear to be trimmable
----------------------------------------
Restored CoG-x 790.82 in
----------------------------------------
```



Note

This page was generated from a Jupyter notebook.

CHAPTER

EIGHT

RUDDER KICK

Simulate a pilot performing a rudder kick test by inputing a rudder input based on a ramp input. Aileron input is also included to maintain a steady heading sideslip (SHSS). The time histories of the control inputs and beta (sideslip angle) are plotted.

[1]:
```
import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab - installing jsbsim . . . ')
    !pip install jsbsim

PATH_TO_JSBSIM_FILES = None
```

[2]:
```
import jsbsim
import matplotlib.pyplot as plt
import math

# --- Configuration Section ---
# Global variables that must be modified to match your particular need
# The aircraft name
# Note - It should match the exact spelling of the model file
AIRCRAFT_NAME="737"

# --- JSBSim Initialization ---
# These lines initialize the flight dynamics model.

# Avoid flooding the console with log messages
jsbsim.FGJSBBase().debug_lvl = 0

# Create a flight dynamics model (FDM) instance.
fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)

# Load the aircraft model
fdm.load_model(AIRCRAFT_NAME)

# Set engines running
fdm['propulsion/set-running'] = -1


# --- Simulation Parameters ---
# These lines set the simulation parameters.

# Set alpha range for trim solutions
fdm['aero/alpha-max-rad'] = math.radians(12)   # Maximum angle of attack in radians.
```

(continues on next page)

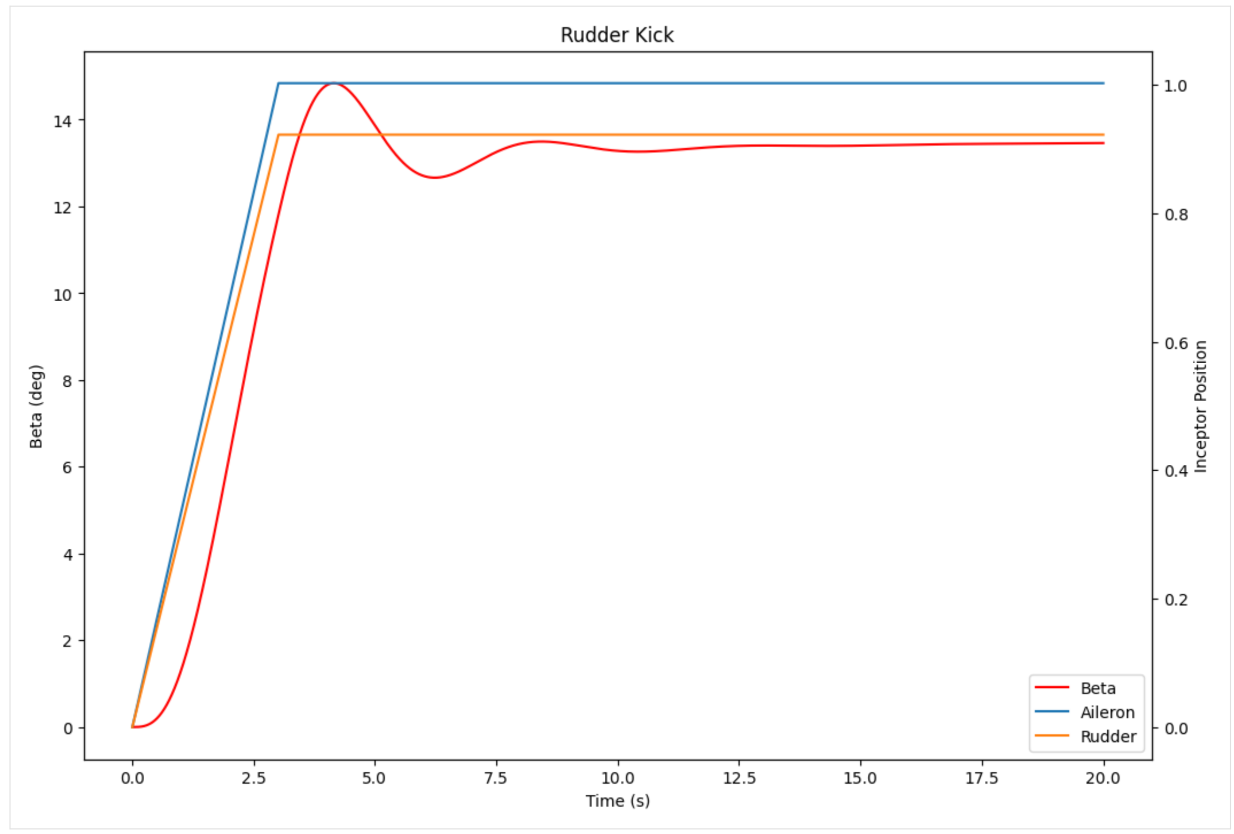
(continued from previous page)

```
fdm['aero/alpha-min-rad'] = math.radians(-4.0) # Minimum angle of attack in radians.

# Get the simulation time step (delta time).
dt = fdm.get_delta_t()

# Max control deflection
aileronMax = 1   # Maximum normarized aileron command (-1~1)
rudderMax = 0.92 # Maximum normarized rudder command (-1~1)

# Number of seconds for control surface to reach max deflection
risetime = 3

# Calculate the increment (change) in control surface deflection command per time step.
diAileron = aileronMax / (risetime/dt)
diRudder = rudderMax / (risetime/dt)


# --- Data Storage ---
# This section initializes lists to record simulation data.

times = []      # List to record the simulation time at each step.
betas = []      # List to record the beta angle at each step.
bankAngle = []  # List to record the bank angle at each step.
ailerons = []   # List to record the aileron control surface deflection.
rudder = []     # List to record the rudder control surface deflection.


# --- Simulation Initialization ---
# This line initializes the flight dynamics model.

# Initial conditions
fdm['ic/h-sl-ft'] = 1000  #altitude above sea level (ft)
fdm['ic/vc-kts'] = 200    #calibrated airspeed (kts)
fdm['ic/gamma-deg'] = 0   #flight path angle (deg)
fdm['ic/beta-deg'] = 0    #sideslip angle (deg)

# Initialize the aircraft with initial conditions
fdm.run_ic()

# Attempt to trim the aircraft.
try:
    # 1 means straight flight by using all changeable control variables.
    fdm['simulation/do_simple_trim'] = 1

except jsbsim.TrimFailureError:
    print("Trim failed, continuing rudder kick in an untrimmed state.")
    pass  # Ignore trim failure


# --- Simulation Loop ---
# This is the main simulation loop that runs the simulation for a specified duration.

# Time to run for in seconds.
run_period = 20

for i in range(int(run_period/dt)):
```

(continues on next page)

(continued from previous page)

```
    # Advance the simulation by one time step.
    fdm.run()

    # Record the simulation data.
    times.append(fdm.get_sim_time())
    betas.append(fdm['aero/beta-deg'])
    bankAngle.append(fdm['attitude/phi-deg'])
    ailerons.append(fdm['fcs/aileron-cmd-norm'])
    rudder.append(fdm['fcs/rudder-cmd-norm'])

    # Control Surface Update
    aileronCmd = fdm['fcs/aileron-cmd-norm']
    rudderCmd = fdm['fcs/rudder-cmd-norm']

    # Increment the aileron command if it's less than the maximum.
    if aileronCmd < aileronMax:
        aileronCmd += diAileron
        fdm['fcs/aileron-cmd-norm'] = aileronCmd

    # Increment the rudder command if it's less than the maximum.
    if rudderCmd < rudderMax:
        rudderCmd += diRudder
        fdm['fcs/rudder-cmd-norm'] = rudderCmd


# --- Plot Results ---
# This section plots the simulation results.

plt.rcParams["figure.figsize"] = (12, 8) # Set the figure size.
fig, ax1 = plt.subplots()

# Plot the beta data on the primary y-axis.
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Beta (deg)')
line1 = ax1.plot(times, betas, label='Beta', color='red')

# Create a secondary y-axis for the control surface positions.
ax2 = ax1.twinx()

# Plot the aileron and rudder commands on the secondary y-axis.
ax2.set_ylabel('Inceptor Position')
line2 = ax2.plot(times, ailerons, label='Aileron')
line3 = ax2.plot(times, rudder, label='Rudder')

# Add a legend to the plot.
ax1.legend(handles=line1+line2+line3, loc=4)

# Set the title of the plot.
plt.title('Rudder Kick')

# Display the plot.
plt.show()
```



Note

This page was generated from a Jupyter notebook.

CHAPTER

NINE

THRUST VECTORING ANALYSIS

Based on a NASA report - Optimal Pitch Thrust-Vector Angle and Benefits for all Flight Regimes

Use JSBSim to compare how varying the thrust vector angle can minimize fuel burn for a given flight condition and compare the results to the NASA report.

Tests performed for a cruise condition and for a climb condition.

[1]:
```
import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab - installing jsbsim . . . ')
    !pip install jsbsim

PATH_TO_JSBSIM_FILES = None
```

9.1 Initialize FDM

[2]:
```
import jsbsim

# --- JSBSim Initialization ---
# These lines initialize the flight dynamics model.

# Create a flight dynamics model (FDM) instance.
fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)

fdm.set_debug_level(0) # Suppress verbose JSBSim console output

if fdm is not None:
    print("FDM created successfully")
    fdm.set_debug_level(0) # Suppress verbose JSBSim console output
else:
    print("Failed to create FDM")
```

```


     JSBSim Flight Dynamics Model v1.3.0 Apr  9 2026 10:00:08
            [JSBSim-ML v2.0]

JSBSim startup beginning . . .

FDM created successfully
```

9.2 Tweak aircraft XML file: remove <input/> nodes from the officially released files

[3]:
```
import os
import xml.etree.ElementTree as ET

AIRCRAFT_NAME="737"

ac_xml_file_path = os.path.join(fdm.get_root_dir(), f'aircraft/{AIRCRAFT_NAME}/{AIRCRAFT_
↪NAME}.xml')
print(f"Aircraft original XML file: {ac_xml_file_path}")

print("Parsing XML ...")
ac_xml_tree = ET.parse(ac_xml_file_path)
ac_xml_root = ac_xml_tree.getroot()

# Save a copy of the original XML file for backup
backup_xml_file_path = ac_xml_file_path.replace('.xml', '_backup.xml')
print(f"Saving backup XML file: {backup_xml_file_path}")
ac_xml_tree.write(backup_xml_file_path)

# Traverse the XML tree and remove <input ... /> occurrences with a 'port' attribute
for x in ac_xml_root.findall('input'):
    has_port = x.get('port') is not None
    if has_port:
        print(f"\tRemoving <input ... /> with port: {x.get('port')}")
        ac_xml_root.remove(x)

print(f"Saving modified XML: {ac_xml_file_path}")
ac_xml_tree.write(ac_xml_file_path)

#check that the input occurrences are removed
ac_xml_tree2 = ET.parse(ac_xml_file_path)
ac_xml_root2 = ac_xml_tree2.getroot()
inputs = ac_xml_root2.findall('input')
if not inputs:
    print("All <input ... /> occurrences successfully removed.")
else:
    print(f"Warning: Found {len(inputs)} <input/> occurrences remaining.")
```

```
Aircraft original XML file: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/
↪aircraft/737/737.xml
Parsing XML . . .
Saving backup XML file: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/aircraft/
↪737/737_backup.xml
Saving modified XML: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/aircraft/737/
↪737.xml
All <input . . . /> occurrences successfully removed.
```

[4]:
```
import math
import numpy as np
import matplotlib.pyplot as plt

# --- Configuration Section ---
# Global variables that must be modified to match your particular need
# The aircraft name
# Note - It should match the exact spelling of the model file
```

(continues on next page)

(continued from previous page)

```
AIRCRAFT_NAME="737"

# --- JSBSim Initialization ---
# These lines initialize the flight dynamics model.

# Avoid flooding the console with log messages
jsbsim.FGJSBBase().debug_lvl = 0

# Create a flight dynamics model (FDM) instance.
fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)

# Load the aircraft model
fdm.load_model(AIRCRAFT_NAME)

# Set engines running
fdm['propulsion/set-running'] = -1


def thrust_vector_range_test(altitude, speed, flight_path_angle, title):
    # altitude: altitude above sea level (ft)
    # speed: mach number of speed (<1)
    #        calibrated airspeed (kts) (>=1)
    # flight_path_angle: flight path angle (deg)
    # title: title for plot

    # Thrust vectoring angles in pitch (deg) to test
    tv_angles = np.linspace(0, 10, 100)

    # Thrust and AoA trim results storage
    thrusts = []
    alphas = []

    # Initialize the minimum thrust and thrust vectoring angles to a very large number
    # to track/record the minimum thrust and the angle at which the minimum occurs.
    min_thrust = 1000000  # thrust (lbf)
    min_angle = 100       # Thrust Vector Angle in pitch (deg)

    # Iterate each thrust vector angles in pitch.
    for tv_angle in tv_angles:

        # --- Simulation Initialization ---
        # This line initializes the flight dynamics model.

        # Initial conditions
        fdm['ic/h-sl-ft'] = altitude  # altitude above sea level (ft)
        # Check the speed and set the value according to if the speed is mach or kts
        if speed < 1.0:
            fdm['ic/mach'] = speed  # mach number of speed
        else:
            fdm['ic/vc-kts'] = speed # calibrated airspeed (kts)
        fdm['ic/gamma-deg'] = flight_path_angle  # flight path angle (deg)

        # Initialize the aircraft with initial conditions
        fdm.run_ic()
```

(continues on next page)

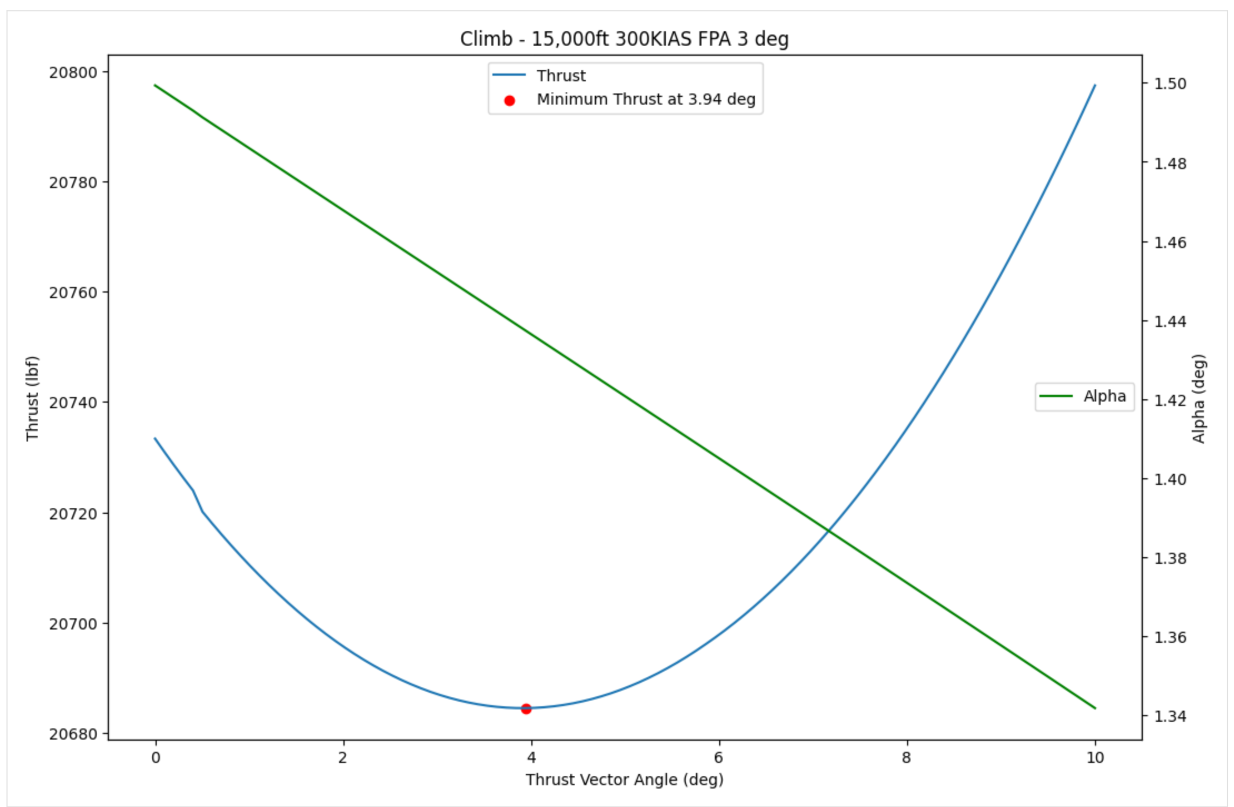
(continued from previous page)

```
        # --- Simulation running ---
        # These lines run the simulation.

        # Trim the aircraft.
        try:
            # Set thrust vector angle in pitch (deg) for both engines
            fdm["propulsion/engine[0]/pitch-angle-rad"] = math.radians(tv_angle)
            fdm["propulsion/engine[1]/pitch-angle-rad"] = math.radians(tv_angle)

            # Trim the aircraft.
            # 1 means straight flight by using all changeable control variables.
            fdm['simulation/do_simple_trim'] = 1

            # Record the simulation data.
            # Append the angle of attack to the result storage.
            alphas.append(fdm["aero/alpha-deg"])
            # Append the thrust to the result storage.
            thrust = fdm["propulsion/engine[0]/thrust-lbs"]*2  # because there are two␣
↪engines
            thrusts.append(thrust)

            # Update the minimum thrust and thrust vectoring angles.
            if thrust < min_thrust:
                min_thrust = thrust
                min_angle = tv_angle

        except jsbsim.TrimFailureError:
            print("Trim failed....")
            pass  # Ignore trim failure


    # --- Plot Results ---
    # This section plots the simulation results.

    plt.rcParams["figure.figsize"] = (12, 8)  # Set the figure size for matplotlib plots.
    fig, ax1 = plt.subplots()
    plt.title(title)

    # Plot the thrust values against the thrust vector angles.
    ax1.plot(tv_angles, thrusts, label='Thrust')
    # Plot the minimum thrust as a red scatter point.
    ax1.scatter(min_angle, min_thrust, color='red', label=f'Minimum Thrust at {min_angle:.
↪2f} deg')
    ax1.set_xlabel('Thrust Vector Angle (deg)')
    ax1.set_ylabel('Thrust (lbf)')

    # Create the second y-axis for AoA
    ax2 = ax1.twinx()
    ax2.set_ylabel('Alpha (deg)')
    # Plot the alpha values against the thrust vector angles.
    ax2.plot(tv_angles, alphas, color='green', label='Alpha')

    ax1.legend(loc='upper center')
    ax2.legend(loc='center right')

    # Save the figure as an SVG file.
```

(continues on next page)

(continued from previous page)

```
    plt.savefig(f"{title}.svg", format="svg")

    # Show the plot.
    plt.show()


# Cruise conditions - 30,000ft Mach 0.8
thrust_vector_range_test(30000, 0.8, 0, 'Cruise - 30,000ft Mach 0.8')

# Climb conditions - 15,000ft 300KIAS flight path angle of 3 degrees
thrust_vector_range_test(15000, 300, 3, 'Climb - 15,000ft 300KIAS FPA 3 deg')
```



Note

This page was generated from a Jupyter notebook.

CHAPTER

TEN

TRIM ENVELOPE

Calculate the set of trim points for an aircraft over a range of airspeeds and range of flight path angles γ. The required thrust and AoA is indicated via a colormap for each trim point.

[1]:
```
import sys

if 'google.colab' in sys.modules:
    print('Running on Google Colab - installing jsbsim . . . ')
    !pip install jsbsim

PATH_TO_JSBSIM_FILES = None
```

10.1 Initialize FDM

[2]:
```
import jsbsim

# --- JSBSim Initialization ---
# These lines initialize the flight dynamics model.

# Create a flight dynamics model (FDM) instance.
fdm = jsbsim.FGFDMExec(PATH_TO_JSBSIM_FILES)

fdm.set_debug_level(0) # Suppress verbose JSBSim console output

if fdm is not None:
    print("FDM created successfully")
    fdm.set_debug_level(0) # Suppress verbose JSBSim console output
else:
    print("Failed to create FDM")
```

```


     JSBSim Flight Dynamics Model v1.3.0 Apr  9 2026 10:00:08
            [JSBSim-ML v2.0]

JSBSim startup beginning . . .

FDM created successfully
```

10.2 Tweak aircraft XML file: remove <input/> nodes from the officially released files

[3]:
```
import os
import xml.etree.ElementTree as ET

AIRCRAFT_NAME="737"

ac_xml_file_path = os.path.join(fdm.get_root_dir(), f'aircraft/{AIRCRAFT_NAME}/{AIRCRAFT_
↪NAME}.xml')
print(f"Aircraft original XML file: {ac_xml_file_path}")

print("Parsing XML ...")
ac_xml_tree = ET.parse(ac_xml_file_path)

# Save a copy of the original XML file for backup
backup_xml_file_path = ac_xml_file_path.replace('.xml', '_backup.xml')
print(f"Saving backup XML file: {backup_xml_file_path}")
ac_xml_tree.write(backup_xml_file_path)

# Parse the XML and get the root element
ac_xml_root = ac_xml_tree.getroot()
# Remove <input ... /> occurrences with a 'port' attribute
for x in ac_xml_root.findall('input'):
    has_port = x.get('port') is not None
    if has_port:
        print(f"\tRemoving <input ... /> with port: {x.get('port')}")
        ac_xml_root.remove(x)

print(f"Saving modified XML: {ac_xml_file_path}")
ac_xml_tree.write(ac_xml_file_path)

#check that the input occurrences are removed
ac_xml_tree2 = ET.parse(ac_xml_file_path)
ac_xml_root2 = ac_xml_tree2.getroot()
inputs = ac_xml_root2.findall('input')
if not inputs:
    print("All <input ... /> occurrences successfully removed.")
else:
    print(f"Warning: Found {len(inputs)} <input/> occurrences remaining.")
```

```
Aircraft original XML file: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/
↪aircraft/737/737.xml
Parsing XML ...
Saving backup XML file: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/aircraft/
↪737/737_backup.xml
Saving modified XML: /home/vscode/.local/lib/python3.11/site-packages/jsbsim/aircraft/737/
↪737.xml
All <input ... /> occurrences successfully removed.
```

10.3 Initialize the JSBSim executable

[4]:
```
import matplotlib.pyplot as plt

if fdm is not None:
    print("FDM instance found, proceeding with model loading...")
    # Load the aircraft model
    if fdm.load_model(AIRCRAFT_NAME):
        print(f"Aircraft model '{AIRCRAFT_NAME}' loaded successfully")
```

(continues on next page)

(continued from previous page)

```
    else:
        print(f"Failed to load aircraft model '{AIRCRAFT_NAME}'")
else:
    print("FDM instance not found, cannot load aircraft model")
```

```
FDM instance found, proceeding with model loading. . .
Aircraft model '737' loaded successfully
```

10.4 Work on trim conditions

[5]:
```
import math

# Set engines running
fdm['propulsion/set-running'] = -1

# --- Simulation Parameters ---
# These lines set the simulation parameters.

# Set alpha range for trim solutions
fdm['aero/alpha-max-rad'] = math.radians(12)   # Maximum angle of attack in radians.
fdm['aero/alpha-min-rad'] = math.radians(-4.0) # Minimum angle of attack in radians.

# Set envelope limits
min_speed = 120   # Set the minimum calibrated airspeed (kts).
max_speed = 460   # Set the maximum calibrated airspeed (kts).
altitude = 15000  # Set the altitude (ft).
min_gamma = -10   # Set the minimum flight path angle (deg).
max_gamma = 10    # Set the maximum flight path angle (deg).

# Trim results
results = []  # Initialize an empty list to store the trim results.


# --- Simulation running ---
# These lines run the simulation for each conditions.

# Iterate over a range of speeds and for each speed a range of flight path angles (gamma)
# and check whether a trim point is possible
for speed in range(min_speed, max_speed, 10):
    for gamma in range(min_gamma, max_gamma, 1):
        # set the initial conditions
        fdm['ic/h-sl-ft'] = altitude  # altitude above sea level (ft)
        fdm['ic/vc-kts'] = speed      # calibrated airspeed (kts)
        fdm['ic/gamma-deg'] = gamma   # flight path angle (deg)

        # Initialize the aircraft with initial conditions
        fdm.run_ic()

        # Trim the aircraft.
        try:
            # 1 means straight flight by using all changeable control variables.
            fdm['simulation/do_simple_trim'] = 1

            # Append the airspeed, AoA, gamma, and throttle to the results list.
            results.append((fdm['velocities/vc-kts'], fdm['aero/alpha-deg'], gamma, fdm[
```

(continues on next page)

(continued from previous page)

```
↪'fcs/throttle-cmd-norm[0]']))

        except jsbsim.TrimFailureError:
            pass  # Ignore trim failures


# --- Plot Results ---
# This section plots the simulation results.

# Extract the trim results
speed, alpha, gamma, throttle = zip(*results)

plt.rcParams["figure.figsize"] = (16, 8)  # Set the figure size for matplotlib plots.

# Plot the trim envelope results, with required thrust and AoA indicated via a colormap
fig, (axThrust, axAoA) = plt.subplots(1, 2)  # Create a figure with two subplots (thrust␣
↪and AoA)

# Graph data for each of the sub plots (title, ax, data)
graph_data = [ ('Thrust', axThrust, throttle), ('AoA', axAoA, alpha) ]

for title, ax, data in graph_data:
    # Scatter plot with airspeed on x-axis, gamma on y-axis and either thrust setting or
    # AoA indicated via color map
    scatter = ax.scatter(speed, gamma, c=data, cmap='viridis')
    cb = fig.colorbar(scatter, ax=ax)  # Add a colorbar to the plot.
    cb.set_label(title)

    # Graph axis range for speed and gamma
    ax.set_xlim(min_speed - 20, max_speed + 20)
    ax.set_ylim(min_gamma * 2, max_gamma * 2)

    ax.grid(True, linestyle='-.')   # Add a grid to the plot.

    ax.set_xlabel('IAS (kt)')
    ax.set_ylabel('Flight Path Angle $\gamma$ (deg)')
    ax.set_title(f'Trim Envelope - {title}')

plt.show()  # Display the plot.
```

```
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
```

(continues on next page)

(continued from previous page)

```
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, wdot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
```

(continues on next page)

(continued from previous page)

```
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
```

(continues on next page)

(continued from previous page)

```
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
```

(continues on next page)

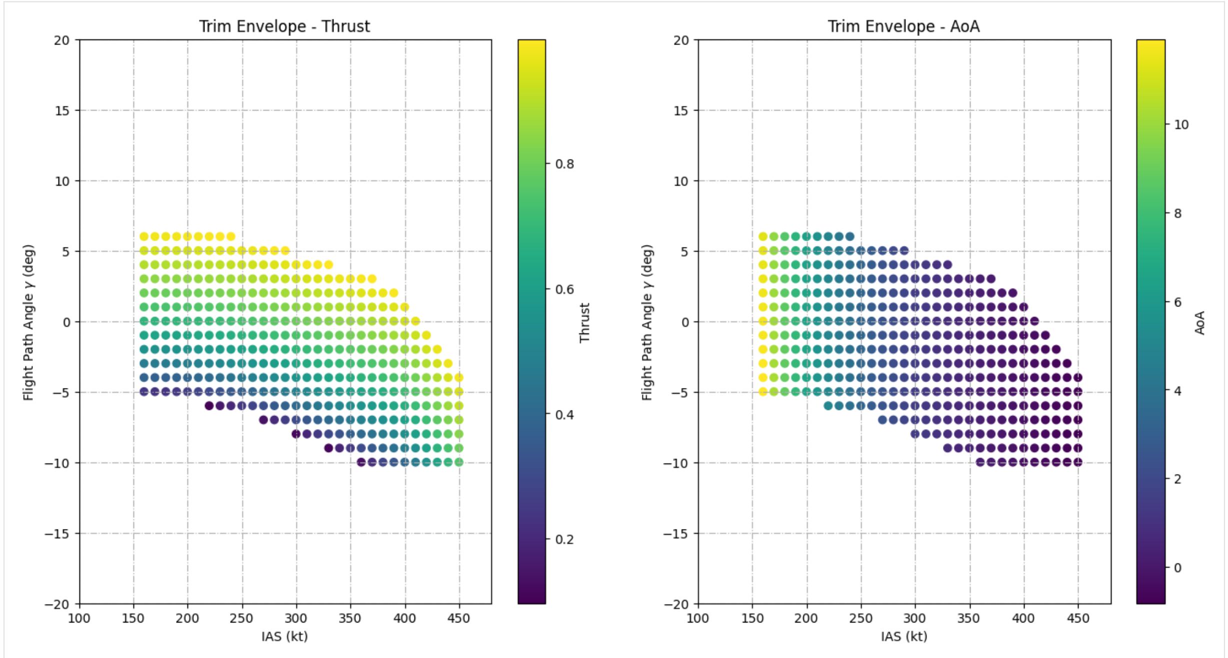
(continued from previous page)

```
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
```

(continues on next page)

(continued from previous page)

```
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
Sorry, udot doesn't appear to be trimmable
```



Note

This page was generated from a Jupyter notebook.

CHAPTER

ELEVEN

TRIM ENVELOPE AND CLIMB ANALYSIS

This notebook demonstrates how to:

1. Generate a trim envelope across a range of altitudes
2. Analyze the required throttle and elevator settings
3. Simulate climbing from 1,000 ft to 20,000 ft using initial trim settings
4. Simulate climbing using interpolated trim settings from the envelope

11.1 Import Required Libraries

[1]:
```
import jsbsim
import matplotlib.pyplot as plt
import numpy as np
import math
```

11.2 Initialize JSBSim and Set Up Aircraft

[2]:
```
AIRCRAFT_NAME="A4"

fdm = jsbsim.FGFDMExec(None)
fdm.set_debug_level(0) # Suppress verbose JSBSim console output

fdm.load_model(AIRCRAFT_NAME)


# Set engines running
fdm['propulsion/set-running'] = -1
```

```


     JSBSim Flight Dynamics Model v1.3.0 Apr  9 2026 10:00:08
            [JSBSim-ML v2.0]

JSBSim startup beginning . . .
```

11.3 Define Envelope Limits and Parameters

[3]:
```
# Set alpha range for trim solutions
fdm['aero/alpha-max-rad'] = math.radians(12)   # Maximum angle of attack in radians.
fdm['aero/alpha-min-rad'] = math.radians(-4.0) # Minimum angle of attack in radians.
```

(continues on next page)

(continued from previous page)

```
# Set envelope limits
speed = 300          # KCAS
min_alt = 1000       # Set the minimum alt (ft).
max_alt = 20000      # Set the maximum alt (ft).
alt_step = 1000      # Set the altitude step (ft).
gamma = 5            # Set the flight path angle range (deg).
```

11.4 Generate Trim Envelope

Compute trim solutions across the altitude range for a constant speed and flight path angle.

```
[4]: # Trim results
results = []  # Initialize an empty list to store the trim results.

# Iterate over a range of altitudes and for each speed a range of flight path angles␣
↪(gamma)
for alt in range(min_alt, max_alt+1, alt_step):
    # Set the initial conditions
    fdm['ic/h-sl-ft'] = alt          # altitude above sea level (ft)
    fdm['ic/vc-kts'] = speed         # calibrated airspeed (kts)
    fdm['ic/gamma-deg'] = gamma      # flight path angle (deg)

    # Initialize the aircraft with initial conditions
    fdm.run_ic()

    # Trim the aircraft.
    try:
        # 1 means straight flight by using all changeable control variables.
        fdm['simulation/do_simple_trim'] = 1

        results.append((alt, fdm['fcs/throttle-cmd-norm[0]'], fdm['fcs/pitch-trim-cmd-norm
↪']))

    except jsbsim.TrimFailureError:
        pass  # Ignore trim failures
```

11.5 Plot Trim Envelope Results

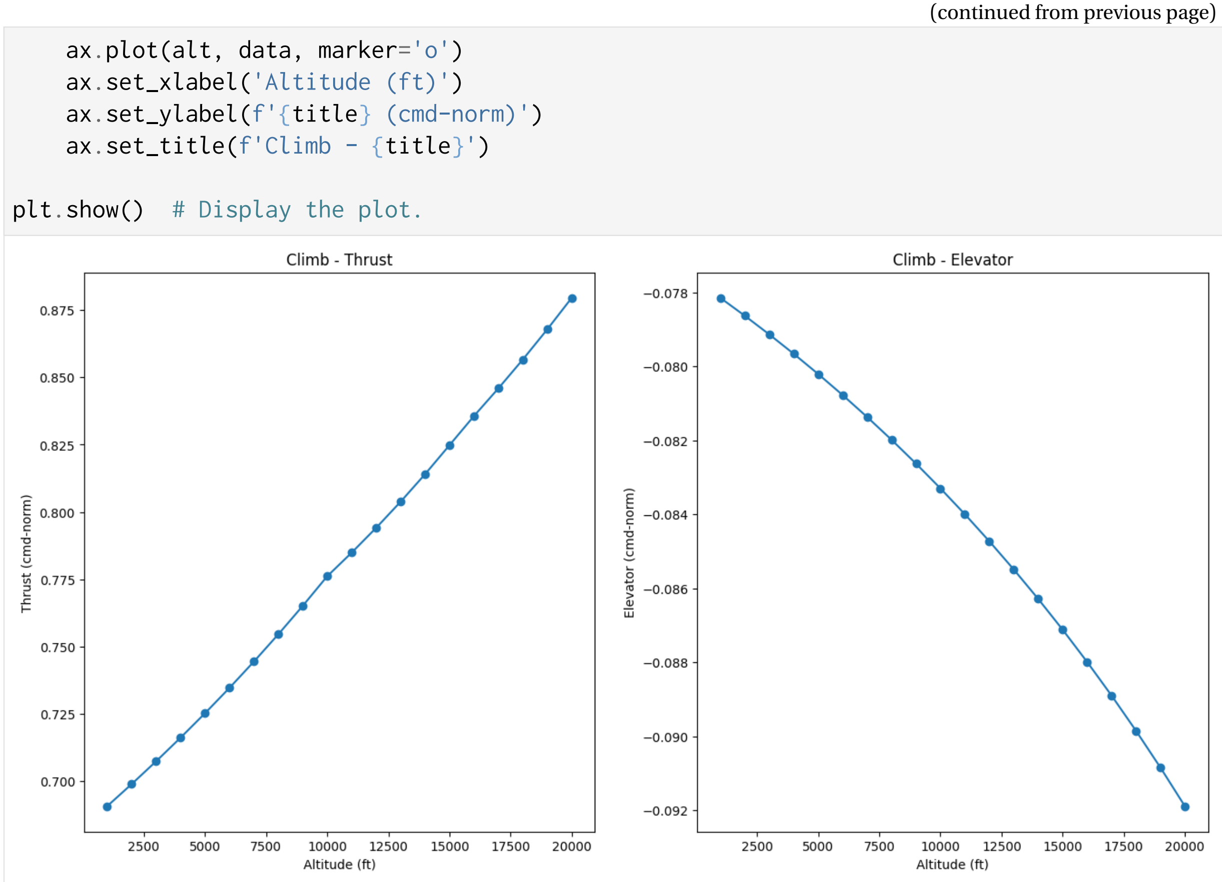
Display the required throttle and elevator settings as functions of altitude.

```
[5]: # Extract the trim results
alt, throttle, elevator = zip(*results)

plt.rcParams["figure.figsize"] = (16, 8)  # Set the figure size for matplotlib plots.

# Plot the trim envelope results, with required thrust and AoA indicated via a colormap
fig, (axThrust, axElevator) = plt.subplots(1, 2)  # Create a figure with two subplots␣
↪(thrust and AoA)

# Graph data for each of the sub plots (title, ax, data)
graph_data = [ ('Thrust', axThrust, throttle), ('Elevator', axElevator, elevator) ]

for title, ax, data in graph_data:
```

(continues on next page)

(continued from previous page)

```
    ax.plot(alt, data, marker='o')
    ax.set_xlabel('Altitude (ft)')
    ax.set_ylabel(f'{title} (cmd-norm)')
    ax.set_title(f'Climb - {title}')

plt.show()  # Display the plot.
```



11.6 Define Flight Simulation Functions

11.6.1 Function 1: Climb with Initial Trim

This function climbs from 1,000 ft to 20,000 ft using only the initial trim solution at 1,000 ft.

```
[6]: def fly_initial_trim(speed, gamma):

    # Set the initial conditions
    fdm['ic/h-sl-ft'] = 1000        # altitude above sea level (ft)
    fdm['ic/vc-kts'] = speed        # calibrated airspeed (kts)
    fdm['ic/gamma-deg'] = gamma     # flight path angle (deg)

    # Initialize the aircraft with initial conditions
    fdm.run_ic()

    # Trim the aircraft.
    fdm['simulation/do_simple_trim'] = 1

    climb_results = []

    while fdm['position/h-sl-ft'] < 20000:
        fdm.run()
        climb_results.append((fdm['position/h-sl-ft'], fdm['velocities/vc-kts'], fdm[
↪'flight-path/gamma-deg']))

    alts, speeds, gammas = zip(*climb_results)
```

(continues on next page)

(continued from previous page)

```
    fig, (axSpeed, axGamma) = plt.subplots(1, 2)

    axSpeed.plot(alts, speeds)
    axGamma.plot(alts, gammas)

    axSpeed.set_ylabel('KCAS (kt)')
    axSpeed.set_xlabel('Altitude (ft)')

    axGamma.set_ylabel('Gamma (deg)')
    axGamma.set_xlabel('Altitude (ft)')

    plt.show()
```

11.6.2 Function 2: Climb with Interpolated Trim

This function climbs from 1,000 ft to 20,000 ft using interpolated throttle and elevator commands from the trim envelope.

```
[7]: def fly_interpolated_trim(speed, gamma, alt, throttle, elevator):

    # Set the initial conditions
    fdm['ic/h-sl-ft'] = 1000        # altitude above sea level (ft)
    fdm['ic/vc-kts'] = speed        # calibrated airspeed (kts)
    fdm['ic/gamma-deg'] = gamma     # flight path angle (deg)

    # Initialize the aircraft with initial conditions
    fdm.run_ic()

    # Trim the aircraft.
    fdm['simulation/do_simple_trim'] = 1

    interp_climb_results = []

    while fdm['position/h-sl-ft'] < 20000:
        fdm.run()

        # Interpolate
        throttle_cmd = np.interp(fdm['position/h-sl-ft'], alt, throttle)
        elevator_cmd = np.interp(fdm['position/h-sl-ft'], alt, elevator)

        fdm['fcs/throttle-cmd-norm'] = throttle_cmd
        fdm['fcs/pitch-trim-cmd-norm'] = elevator_cmd

        interp_climb_results.append((fdm['position/h-sl-ft'], fdm['velocities/vc-kts'],␣
↪fdm['flight-path/gamma-deg']))

    alt, speeds, gammas = zip(*interp_climb_results)

    fig, (axSpeed, axGamma) = plt.subplots(1, 2)

    axSpeed.plot(alt, speeds)
    axGamma.plot(alt, gammas)

    axSpeed.set_ylabel('KCAS (kt)')
    axSpeed.set_xlabel('Altitude (ft)')
```

(continues on next page)

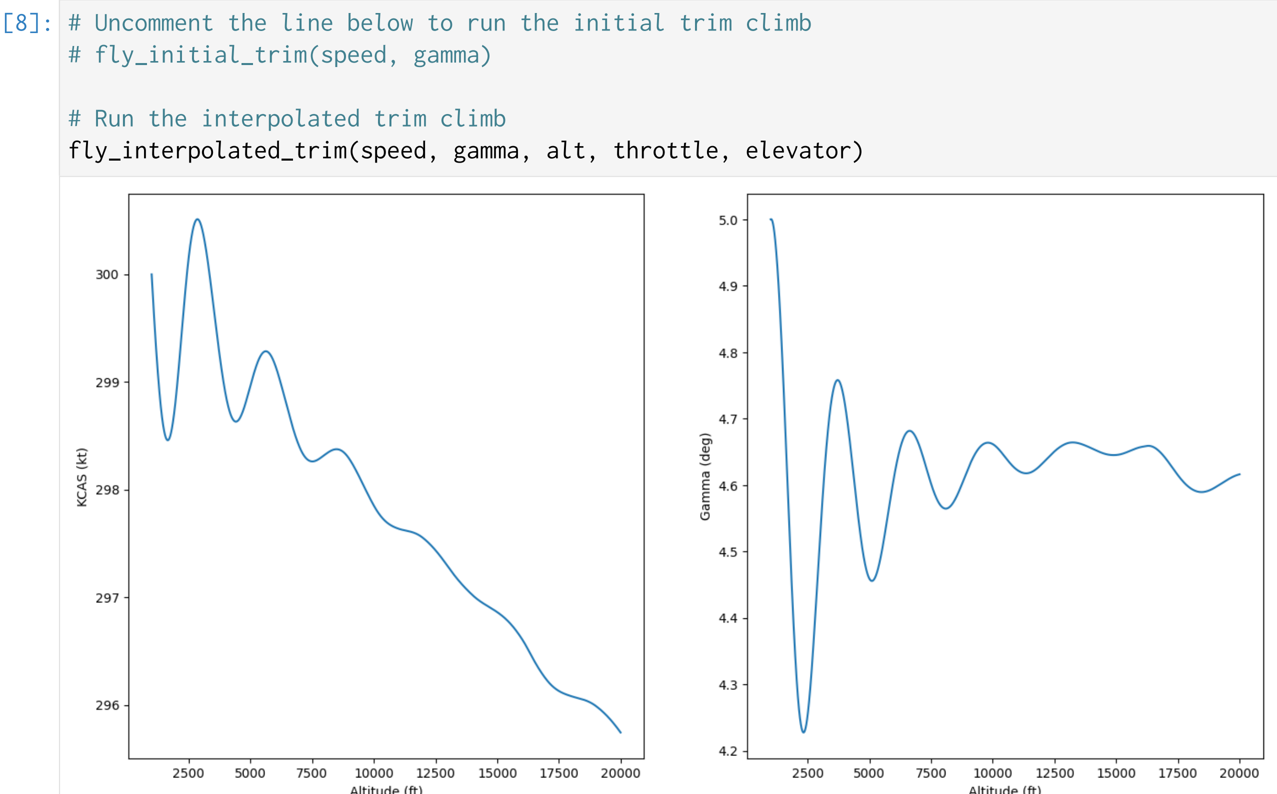
(continued from previous page)

```
    axGamma.set_ylabel('Gamma (deg)')
    axGamma.set_xlabel('Altitude (ft)')

    plt.show()
```

11.7 Execute Climb Simulations

Run the interpolated trim climb simulation. You can uncomment the initial trim function to compare results.

[8]:
```
# Uncomment the line below to run the initial trim climb
# fly_initial_trim(speed, gamma)

# Run the interpolated trim climb
fly_interpolated_trim(speed, gamma, alt, throttle, elevator)
```



CHAPTER

TWELVE

RESOURCES

12.1 This documentation

- Download this documentation as PDF

12.2 Related projects

- JSBSim – Open-source, multi-platform flight dynamics model (FDM) written in C++ with a Python wrapper.
- JSBSim documentation – Reference manual including the Python API.
- PathSim – Block-based, time-domain system simulation framework in Python.
- PathSim documentation – Tutorials, API reference, and examples.
- PathView – Browser-based graphical editor for PathSim block diagrams. Export diagrams as Python scripts.
- pathsim-flight – Flight-dynamics blocks for PathSim, including a JSBSim wrapper and the International Standard Atmosphere.

12.3 Useful references

- JSBSim Property Reference
- Sphinx
- nbsphinx – Sphinx extension for Jupyter notebooks

CHAPTER

THIRTEEN

QUICK START

```
# 1. Create and activate the conda environment
conda env create -f environment.yml
conda activate jsbsim-python-examples

# 2. Launch JupyterLab
jupyter lab
```

CHAPTER FOURTEEN

INDICES AND TABLES

- genindex
- search