ch05_linear_models

February 15, 2022

0.1 Problem 1: Creating trim trajectory calculator

A trim state and input produce constant dynamic outputs. One way to produce the trim states and inputs is to solve an optimization problem that results in the desired trim state derivative. Consider the objective function defined as

$$\min_{x_{trim},\delta_{trim}} (f(x_{trim},\delta_{trim}) - f_d)^T Q(f(x_{trim},\delta_{trim}) - f_d)$$

where $f(x_{trim},\delta_{trim})$ denotes the dynamics given the trim state and input, f_d denotes the desired dynamics, and $Q \succeq 0$ is a diagonal matrix with zeros and ones along the diagonal (i.e., $Q = diag([0, 0, 1, ..., 1])$. If possible, the optimization will produce trim states and inputs such that $f = f_d$.

Do the following 1. Explain in your own words each of the major lines in the function `compute_trim(...)` located in `chap5\trim.py`. Ensure that you explain each constraint 2. Implement the `trim_objective_fun` located in `chap5\trim.py`

0.1.1 Explanation of `compute_trim(...)`

```
            'fun': lambda x: np.array([
                                # The total airspeed is equal to the sum of its components (No
                                x[IND.U]**2 + x[IND.V]**2 + x[IND.W]**2 - Va**2,
                                # Pitch velocity is 0 (v is always 0)
                                x[IND.V],
                                # Enforce attitude change is at constraint rate
                                x[IND.E0]**2 + x[IND.E1]**2 + x[IND.E2]**2 + x[IND.E3]**2 - 1.
                                # forcing e1=e3=0 ensures zero roll and zero yaw in trim
                                x[IND.E1], # e1=0
                                x[IND.E3], # e3=0
                                # Ensure no change angular velocities in roll, pitch, or yaw
                                x[IND.P],  # p=0
                                x[IND.Q],  # q=0
                                x[IND.R],  # r=0
                                ]),
```

[1]:
```python
# Note that this cell can be run separately to initialize for other cell blocks
import numpy as np
from mav_sim.chap3.mav_dynamics import DynamicState
from mav_sim.chap4.run_sim import run_sim
```

```python
from mav_sim.message_types.msg_sim_params import MsgSimParams
from mav_sim.message_types.msg_delta import MsgDelta
from mav_sim.tools.display_figures import display_data_view, display_mav_view
from mav_sim.chap2.mav_viewer import MavViewer
from mav_sim.chap3.data_viewer import DataViewer
from mav_sim.chap5.trim import compute_trim
from IPython.display import display # Used to display variables nicely in␣
 ↪Jupyter
from mav_sim.chap3.mav_dynamics import DynamicState, derivatives
from mav_sim.chap4.mav_dynamics import forces_moments, update_velocity_data
from mav_sim.tools.signals import Signals

# The viewers need to be initialized once due to restart issues with qtgraph
if 'mav_view' not in globals():
    print("Initializing mav_view")
    global mav_view
    mav_view = MavViewer()  # initialize the mav viewer
if 'data_view' not in globals():
    print("Initializing data_view")
    global data_view
    data_view = DataViewer()  # initialize view of data plots

# Initialize state values
sim_params = MsgSimParams(end_time=40., video_name="chap5.avi")
state = DynamicState()

# Functions used below
def run_sim_and_display(delta_fnc, init_state):
    global mav_view
    global data_view
    data_view.reset(sim_params.start_time)
    (mav_view, data_view) = run_sim(sim_params, delta_fnc,␣
 ↪DynamicState(init_state), mav_view, data_view)
    display_data_view(data_view)
    display_mav_view(mav_view)
```

```
Initializing mav_view
Initializing data_view
```

0.2 Problem 2 - Calculate Trim

Do the following: 1. Compute a trim state and input for $V_a = 25\frac{m}{s}$ and $\gamma = 0$ 2. Show that the resulting dynamics are close to the desired dynamics 3. Simulate the state and show that the trim values are achieved for a period before numerical errors enter the system

Keep in mind the following when comparing results: * $\dot{p_n}$ and $\dot{p_e}$ are not important * Your trim state dynamics will not be exact, but they should be close

[2]:

```
# Create the trim state
Va_trim = 25.
gamma_trim = 0.
trim_state, trim_input = compute_trim(state.convert_to_numpy(), Va_trim,␣
 ↪gamma_trim)

# Display the trim state and input
print('trim_state = ')
display(trim_state)
print('trim input = ')
trim_input.print()

# Calculate h_dot
h_dot = lambda V,g: V*np.sin(g)

# Calculate the desired state dynamics
desired_trim_state_dot = np.array([h_dot(Va_trim,gamma_trim), 0, 0, 0, 0, 0, 0,␣
 ↪0, 0, 0])
print("Need to implement")

# Calculate the actual state dynamics
f = trim_state
print("Need to implement")

# Display the difference
f_diff = f - desired_trim_state_dot
print("Difference between actual and desired (Note that pn and pe are not␣
 ↪important):")
display(f_diff)

# Create a passthrough function for the trim input
pass_delta2 = lambda sim_time: trim_input
run_sim_and_display(pass_delta2, trim_state)
```

```
trim_state =
```

```
/home/alex/spring2022/small_aircraft/Chap05_Linear_Models_start/mav_sim/chap4/ma
v_dynamics.py:254: RuntimeWarning: invalid value encountered in sqrt
  omega_p = (-b + np.sqrt(b**2 - 4*a*c)) / (2.*a)
```

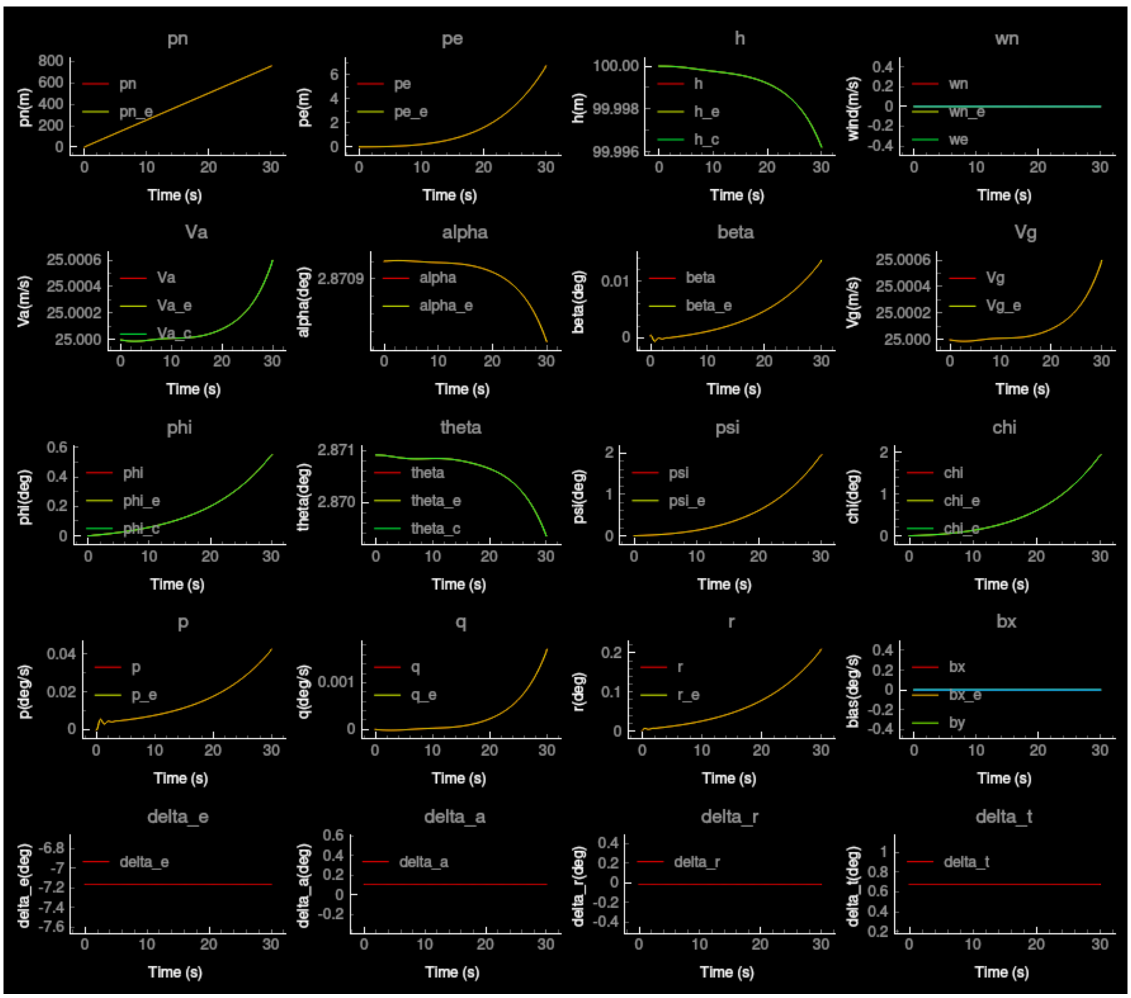
```
array([[-2.23382207e-11],
       [-3.26585996e-12],
       [-1.00000000e+02],
       [ 2.49686227e+01],
       [ 0.00000000e+00],
       [ 1.25215104e+00],
       [ 9.99686179e-01],
       [ 0.00000000e+00],
```

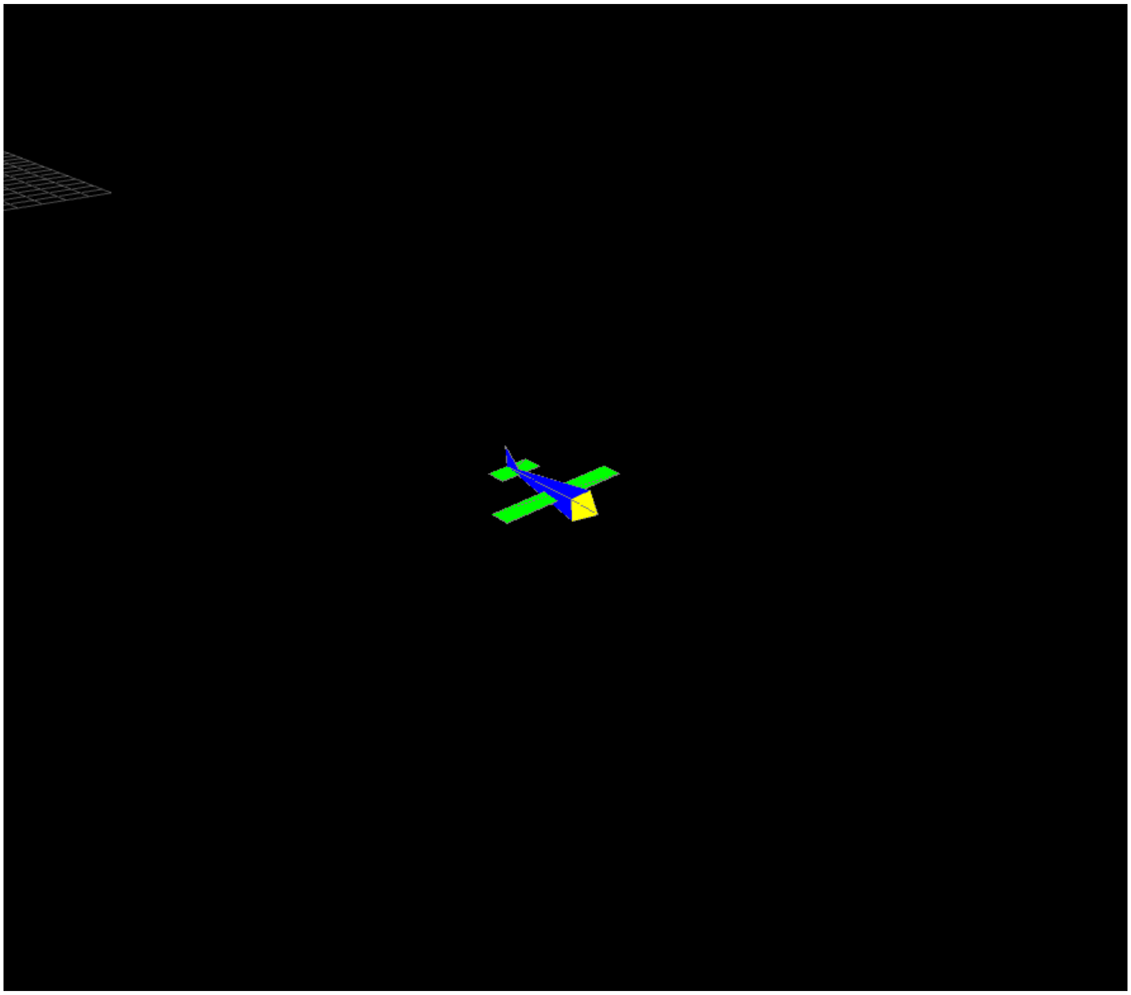
```
       [ 2.50508167e-02],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00]])
```

```
trim input =
elevator= -0.12504364911744936 aileron= 0.001837087616601899 rudder=
-0.00030276453798068806 throttle= 0.6767747501906854
Need to implement
Need to implement
Difference between actual and desired (Note that pn and pe are not important):
```

```
array([[-2.23382207e-11, -2.23382207e-11, -2.23382207e-11,
        -2.23382207e-11, -2.23382207e-11, -2.23382207e-11,
        -2.23382207e-11, -2.23382207e-11, -2.23382207e-11,
        -2.23382207e-11],
       [-3.26585996e-12, -3.26585996e-12, -3.26585996e-12,
        -3.26585996e-12, -3.26585996e-12, -3.26585996e-12,
        -3.26585996e-12, -3.26585996e-12, -3.26585996e-12,
        -3.26585996e-12],
       [-1.00000000e+02, -1.00000000e+02, -1.00000000e+02,
        -1.00000000e+02, -1.00000000e+02, -1.00000000e+02,
        -1.00000000e+02, -1.00000000e+02, -1.00000000e+02,
        -1.00000000e+02],
       [ 2.49686227e+01,  2.49686227e+01,  2.49686227e+01,
         2.49686227e+01,  2.49686227e+01,  2.49686227e+01,
         2.49686227e+01,  2.49686227e+01,  2.49686227e+01,
         2.49686227e+01],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [ 1.25215104e+00,  1.25215104e+00,  1.25215104e+00,
         1.25215104e+00,  1.25215104e+00,  1.25215104e+00,
         1.25215104e+00,  1.25215104e+00,  1.25215104e+00,
         1.25215104e+00],
       [ 9.99686179e-01,  9.99686179e-01,  9.99686179e-01,
         9.99686179e-01,  9.99686179e-01,  9.99686179e-01,
         9.99686179e-01,  9.99686179e-01,  9.99686179e-01,
         9.99686179e-01],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [ 2.50508167e-02,  2.50508167e-02,  2.50508167e-02,
         2.50508167e-02,  2.50508167e-02,  2.50508167e-02,
         2.50508167e-02,  2.50508167e-02,  2.50508167e-02,
         2.50508167e-02],
```

```
[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00]])
```





0.3 Problem 3 - Trim Calculations

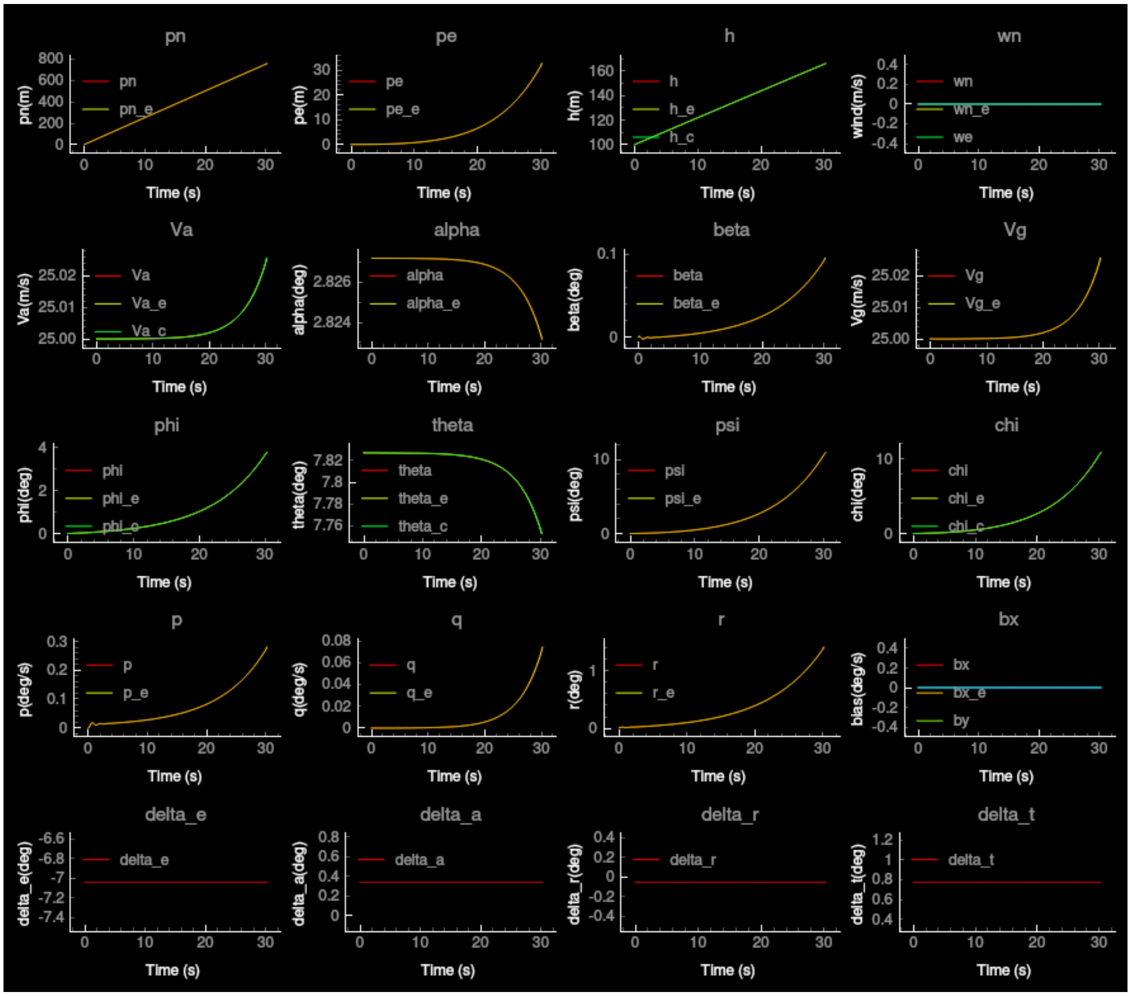
Repeate problem 2 with $V_a = 35\frac{m}{s}$ and $\gamma = 5$ degrees

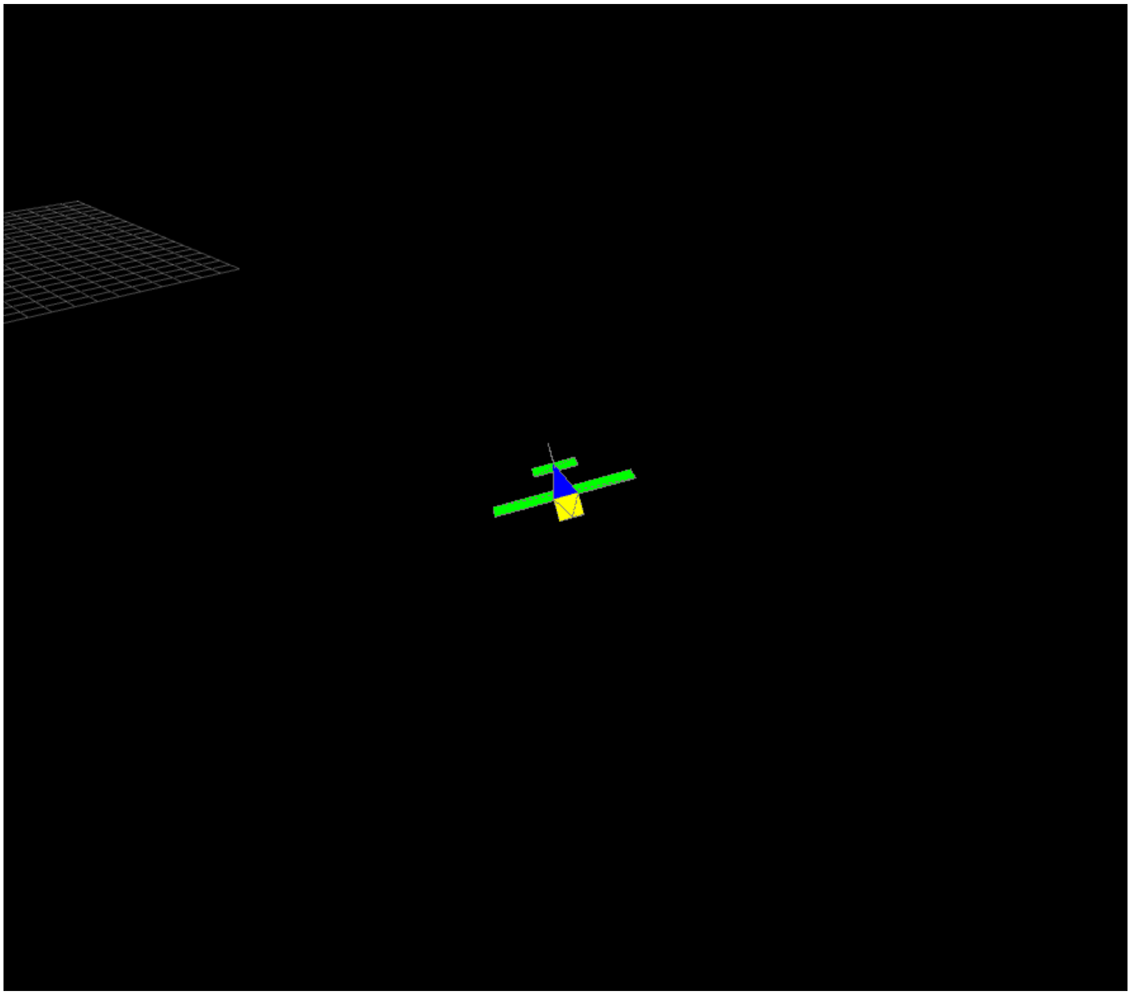
[3]:
```
# Create the trim state
Va_trim = 25.
gamma_trim = np.deg2rad(5.)
trim_state, trim_input = compute_trim(state.convert_to_numpy(), Va_trim,␣
↪gamma_trim)

# Display the trim state and input
print('trim_state = ')
display(trim_state)
print('trim input = ')
trim_input.print()

# Calculate h_dot
```

```python
h_dot = lambda V,g: V*np.sin(g)

# Calculate the desired state dynamics
desired_trim_state_dot = np.array([h_dot(Va_trim,gamma_trim), 0, 0, 0, 0, 0, 0,␣
    ↪0, 0, 0])

# Calculate the actual state dynamics
f = trim_state

# Display the difference
f_diff = f - desired_trim_state_dot
print("Difference between actual and desired (Note that pn and pe are not␣
    ↪important):")
display(f_diff)

# Create a passthrough function for the trim input
pass_delta2 = lambda sim_time: trim_input
run_sim_and_display(pass_delta2, trim_state)
```

```
trim_state =

array([[-1.81037848e-11],
       [ 5.98728640e-13],
       [-1.00000000e+02],
       [ 2.49695712e+01],
       [ 0.00000000e+00],
       [ 1.23309109e+00],
       [ 9.97668128e-01],
       [ 0.00000000e+00],
       [ 6.82517842e-02],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00]])

trim input =
elevator= -0.12293096105580768 aileron= 0.005931436776133369 rudder=
-0.0009774962373447855 throttle= 0.7737412925592396
Difference between actual and desired (Note that pn and pe are not important):

array([[-2.17889357e+00, -1.81037848e-11, -1.81037848e-11,
        -1.81037848e-11, -1.81037848e-11, -1.81037848e-11,
        -1.81037848e-11, -1.81037848e-11, -1.81037848e-11,
        -1.81037848e-11],
       [-2.17889357e+00,  5.98728640e-13,  5.98728640e-13,
         5.98728640e-13,  5.98728640e-13,  5.98728640e-13,
         5.98728640e-13,  5.98728640e-13,  5.98728640e-13,
         5.98728640e-13],
       [-1.02178894e+02, -1.00000000e+02, -1.00000000e+02,
```

```
 -1.00000000e+02, -1.00000000e+02, -1.00000000e+02,
 -1.00000000e+02, -1.00000000e+02, -1.00000000e+02,
 -1.00000000e+02],
[ 2.27906776e+01,  2.49695712e+01,  2.49695712e+01,
  2.49695712e+01,  2.49695712e+01,  2.49695712e+01,
  2.49695712e+01,  2.49695712e+01,  2.49695712e+01,
  2.49695712e+01],
[-2.17889357e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[-9.45802479e-01,  1.23309109e+00,  1.23309109e+00,
  1.23309109e+00,  1.23309109e+00,  1.23309109e+00,
  1.23309109e+00,  1.23309109e+00,  1.23309109e+00,
  1.23309109e+00],
[-1.18122544e+00,  9.97668128e-01,  9.97668128e-01,
  9.97668128e-01,  9.97668128e-01,  9.97668128e-01,
  9.97668128e-01,  9.97668128e-01,  9.97668128e-01,
  9.97668128e-01],
[-2.17889357e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[-2.11064178e+00,  6.82517842e-02,  6.82517842e-02,
  6.82517842e-02,  6.82517842e-02,  6.82517842e-02,
  6.82517842e-02,  6.82517842e-02,  6.82517842e-02,
  6.82517842e-02],
[-2.17889357e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[-2.17889357e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[-2.17889357e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[-2.17889357e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00]])
```


pn
pn(m)
pn
pn_e
pe
pe(m)
pe
pe_e
h
h(m)
h
h_e
h_c
wn
wind(m/s)
wn
wn_e
we
Va
Va(m/s)
Va
Va_e
Va_c
alpha
alpha(deg)
alpha
alpha_e
beta
beta(deg)
beta
beta_e
Vg
Vg(m/s)
Vg
Vg_e
phi
phi(deg)
phi
phi_e
phi_c
theta
theta(deg)
theta
theta_e
theta_c
psi
psi(deg)
psi
psi_e
chi
chi(deg)
chi
chi_e
chi_c
p
p(deg/s)
p
p_e
q
q(deg/s)
q
q_e
r
r(deg)
r
r_e
bx
bias(deg/s)
bx
bx_e
by
delta_e
delta_e(deg)
delta_e
delta_a
delta_a(deg)
delta_a
delta_r
delta_r(deg)
delta_r
delta_t
delta_t(deg)
delta_t
Time (s)



0.4 Problem 4 - Evaluate Eigenvalues of Longitudinal System

The `compute_ss_model(...)` function inside `chap5\compute_models.py` provides a numerical approximation for the models described in (5.44) and (5.51).

For the trim trajectory corresponding to $V_a = 25\frac{m}{s}$ and $\gamma = 0$, do the following: * Calculate the eigenvalues of A_{lon} and A_{lat} * Answer the questions below

0.4.1 Question: Which eigenvalue(s) correspond to the short-period mode?

The pair with the larger n correspond to the short-period mode. (-0.10413332)

0.4.2 Question: Which eigenvalue(s) correspond to the phugoid mode?

The pair with the smaller n correspond to the phugoid mode (-4.87859822)

0.4.3 Question: Which eigenvalue(s) corresponds to the spiral-divergence mode?

The real eigenvalue in the right half plane is the spiral-divergence mode (8.93941805e-02)

0.4.4 Question: Which eigenvalue(s) corresponds to the roll mode?

The real eigenvalue in the left half plane is the roll mode (-2.24411598e+01)

0.4.5 Question: Which eigenvalue(s) corresponds to the dutch-roll mode?

The complex eigenvalues are the dutch-roll mode (-1.14075619e+00+4)

[4]:
```python
from mav_sim.chap5.compute_models import compute_ss_model

# Compute the trim state and input
Va_trim                = 25.
gamma_trim             = 0.
trim_state, trim_input = compute_trim(state.convert_to_numpy(), Va_trim,␣
 ↪gamma_trim)

# Compute A_lon and A_lat
A_lon, B_lon, A_lat, B_lat = compute_ss_model(trim_state, trim_input)

# Compute the eigenvalues of A_lon
print("eig(A_lon): \n", np.linalg.eigvals(A_lon))

# Compute the eigenvalues of A_lat
print("eig(A_lat): \n", np.linalg.eigvals(A_lat))
```

```
eig(A_lon):
 [ 0.        +0.j          -4.87859822+9.86957041j -4.87859822-9.86957041j
 -0.10413332+0.48908574j -0.10413332-0.48908574j]
eig(A_lat):
 [-2.24411598e+01+0.j          -1.14075619e+00+4.65506181j
 -1.14075619e+00-4.65506181j  8.93941805e-02+0.j
 -4.36439390e-33+0.j        ]
```

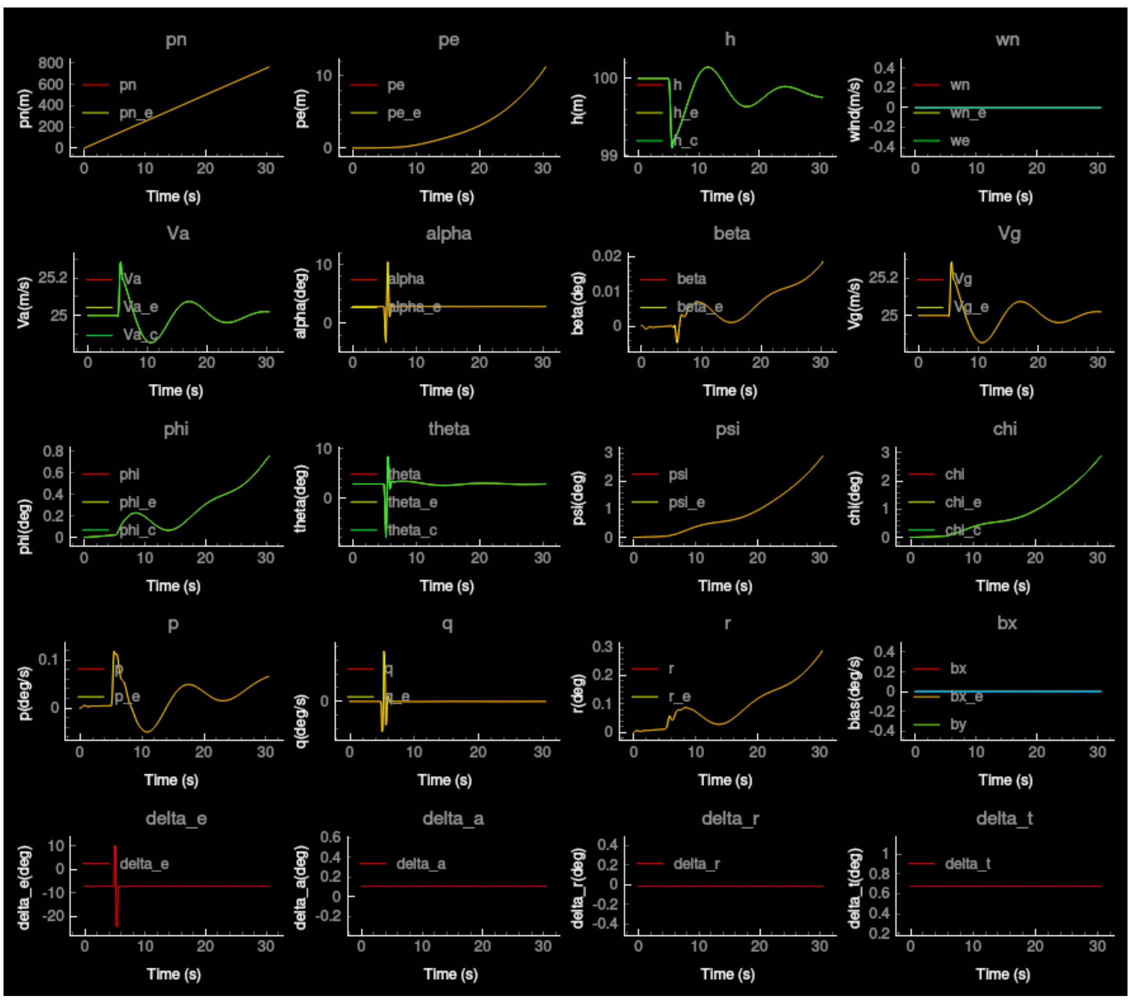
```
/home/alex/spring2022/small_aircraft/Chap05_Linear_Models_start/mav_sim/chap4/ma
v_dynamics.py:254: RuntimeWarning: invalid value encountered in sqrt
  omega_p = (-b + np.sqrt(b**2 - 4*a*c)) / (2.*a)
```

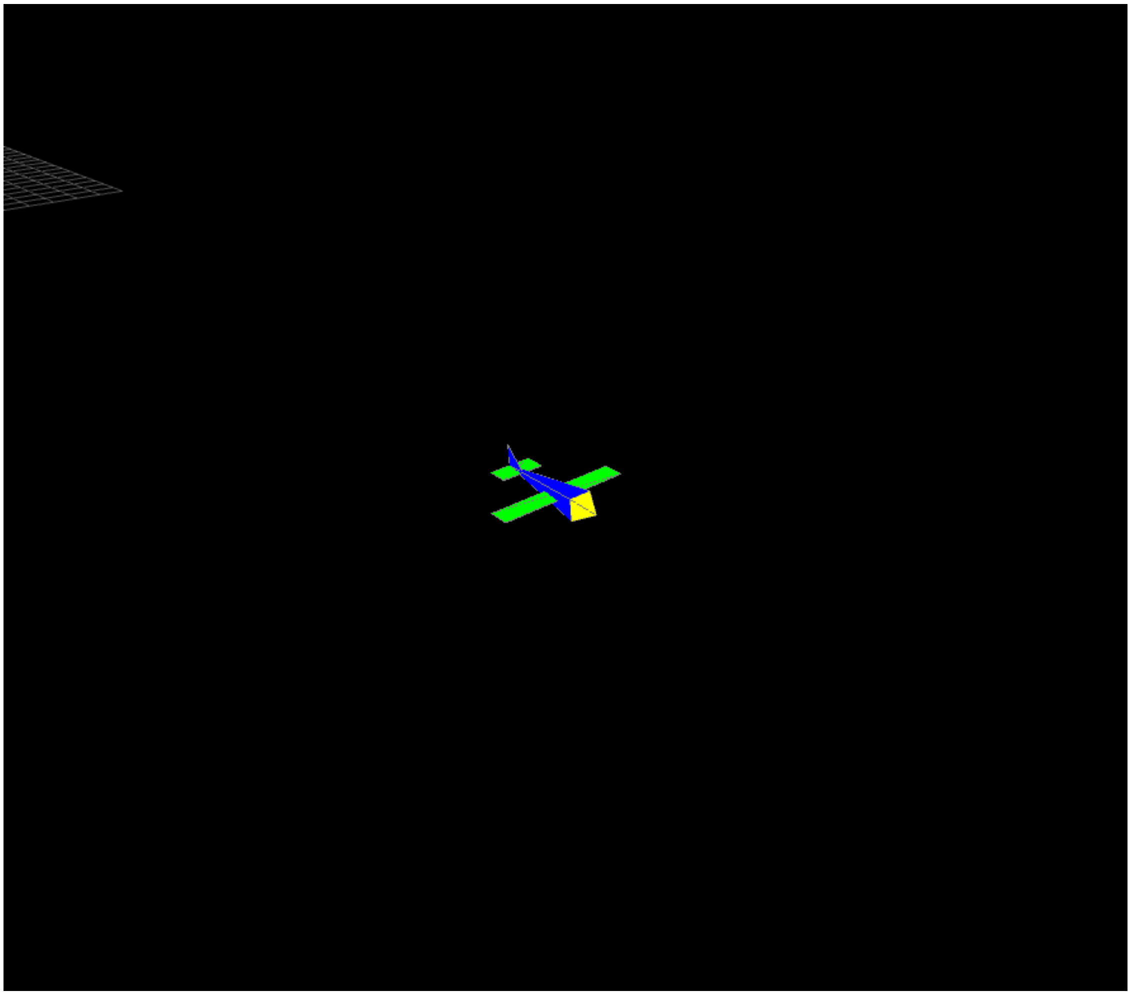
0.5 Problem 5 - Phugoid mode

For the trim trajectory corresponding to $V_a = 25\frac{m}{s}$ and $\gamma = 0$, use a doublet to excite the phugoid mode. Simulate the response. (Note that this problem is provided for you)

[5]:
```python
# Create the trim state
Va_trim = 25.
gamma_trim = 0.
trim_state, trim_input = compute_trim(state.convert_to_numpy(), Va_trim,␣
 ↪gamma_trim)


# Create an input signal
```

```python
input_signal = Signals(amplitude=0.3,
                       duration=0.3,
                       start_time=5.0)

# Create a function for exciting the phugoid mode
def excite_phugoid(sim_time: float):
    # copy the trim command
    delta_cmd = MsgDelta()
    delta_cmd.copy(trim_input)

    # Excite the phugoid mode
    delta_cmd.elevator += input_signal.doublet(sim_time)
    return delta_cmd

# Run the command
run_sim_and_display(excite_phugoid, trim_state)
```





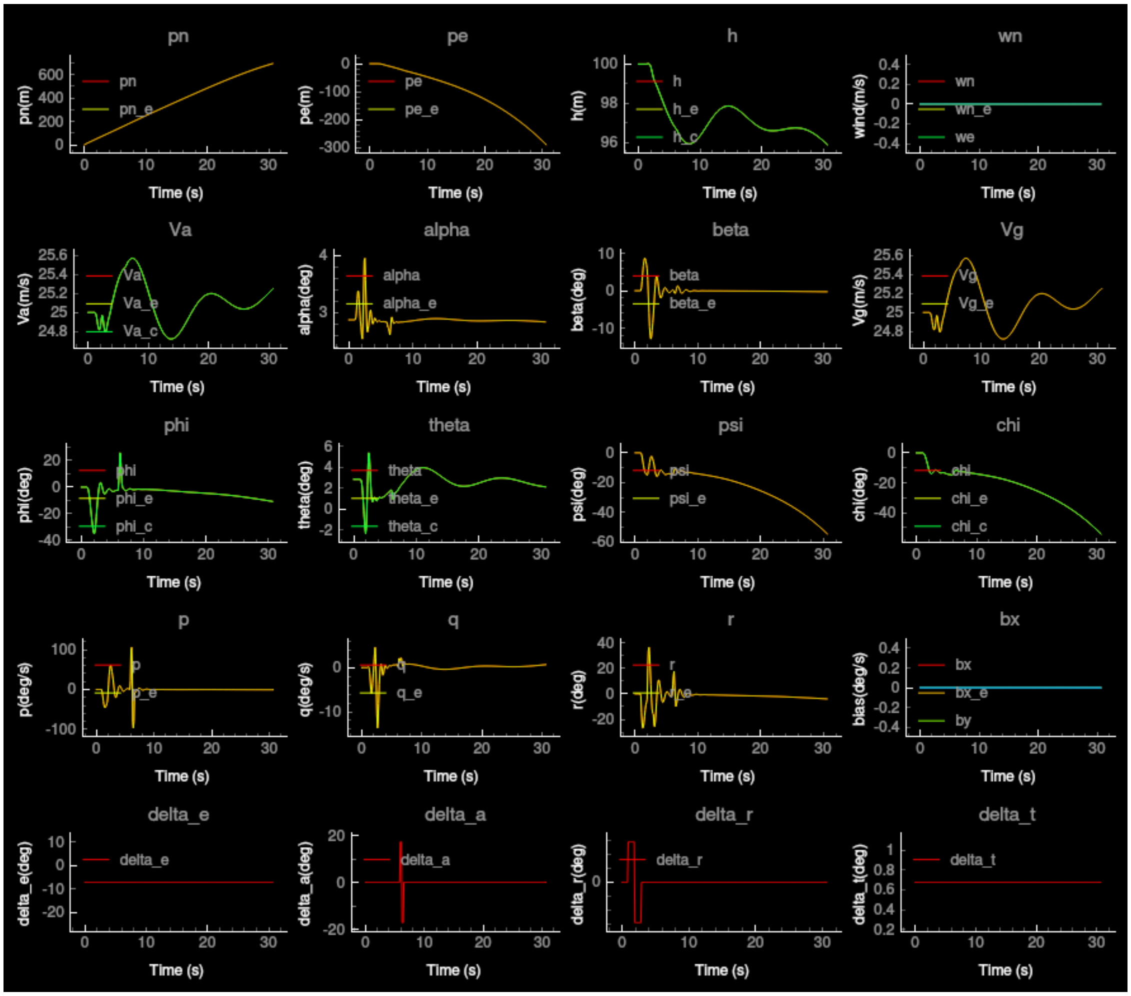
0.6 Problem 6 - Roll and spiral divergence modes

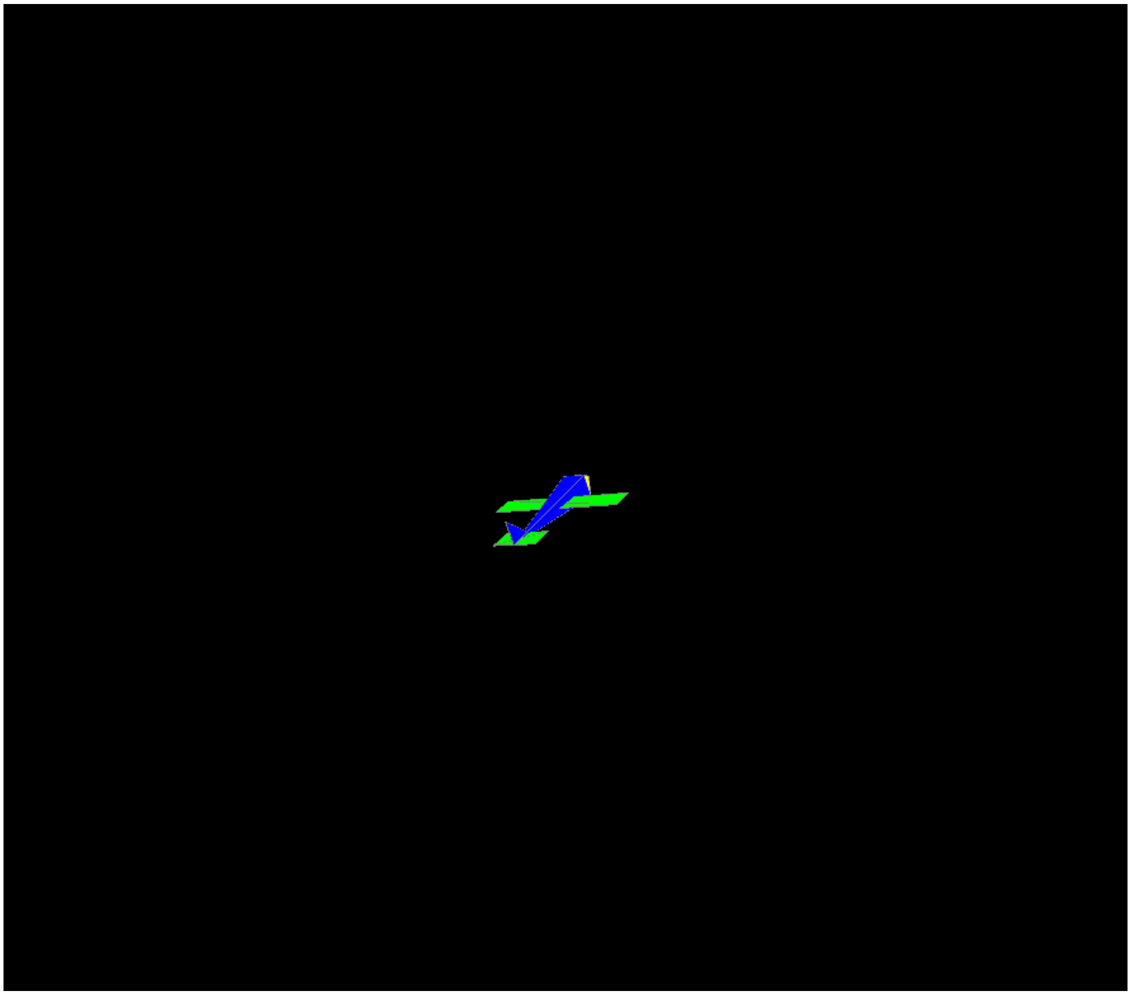
For the trim trajectory corresponding to $V_a = 25\frac{m}{s}$ and $\gamma = 0$, use a doublet to excite the roll and spiral divergence modes.

[6]:
```
# Create the trim state
Va_trim = 25.
gamma_trim = 0.
trim_state, trim_input = compute_trim(state.convert_to_numpy(), Va_trim,␣
 ↪gamma_trim)

# Create an input signal
input_signal = Signals(amplitude=0.3,
                       duration=0.3,
                       start_time=6.0)

input_signal_slow = Signals(amplitude=0.1,
```

```
                    duration=1,
                    start_time=1.0)

# Create a function for exciting the phugoid mode
def excite_roll(sim_time: float):
    # copy the trim command
    delta_cmd = MsgDelta()
    delta_cmd.copy(trim_input)

    # Excite the phugoid mode
    delta_cmd.aileron += input_signal.doublet(sim_time)
    delta_cmd.rudder  += input_signal_slow.doublet(sim_time)
    return delta_cmd

# Create a function for exciting the phugoid mode
def excite_sprial(sim_time: float):
    # copy the trim command
    delta_cmd = MsgDelta()
    delta_cmd.copy(trim_input)

    # Excite the phugoid mode

    return delta_cmd

# Run the command
run_sim_and_display(excite_roll, trim_state)
#run_sim_and_display(excite_sprial, trim_state)
```

```
/home/alex/spring2022/small_aircraft/Chap05_Linear_Models_start/mav_sim/chap4/ma
v_dynamics.py:254: RuntimeWarning: invalid value encountered in sqrt
  omega_p = (-b + np.sqrt(b**2 - 4*a*c)) / (2.*a)
```


pn
pn(m)
pn
pn_e
Time (s)
pe
pe(m)
pe
pe_e
Time (s)
h
h(m)
h
h_e
h_c
Time (s)
wn
wind(m/s)
wn
wn_e
we
Time (s)
Va
Va(m/s)
Va
Va_e
Va_c
Time (s)
alpha
alpha(deg)
alpha
alpha_e
Time (s)
beta
beta(deg)
beta
beta_e
Time (s)
Vg
Vg(m/s)
Vg
Vg_e
Time (s)
phi
phi(deg)
phi
phi_e
phi_c
Time (s)
theta
theta(deg)
theta
theta_e
theta_c
Time (s)
psi
psi(deg)
psi
psi_e
Time (s)
chi
chi(deg)
chi
chi_e
chi_c
Time (s)
p
p(deg/s)
p
p_e
Time (s)
q
q(deg/s)
q
q_e
Time (s)
r
r(deg)
r
r_e
Time (s)
bx
bias(deg/s)
bx
bx_e
by
Time (s)
delta_e
delta_e(deg)
delta_e
Time (s)
delta_a
delta_a(deg)
delta_a
Time (s)
delta_r
delta_r(deg)
delta_r
Time (s)
delta_t
delta_t(deg)
delta_t
Time (s)



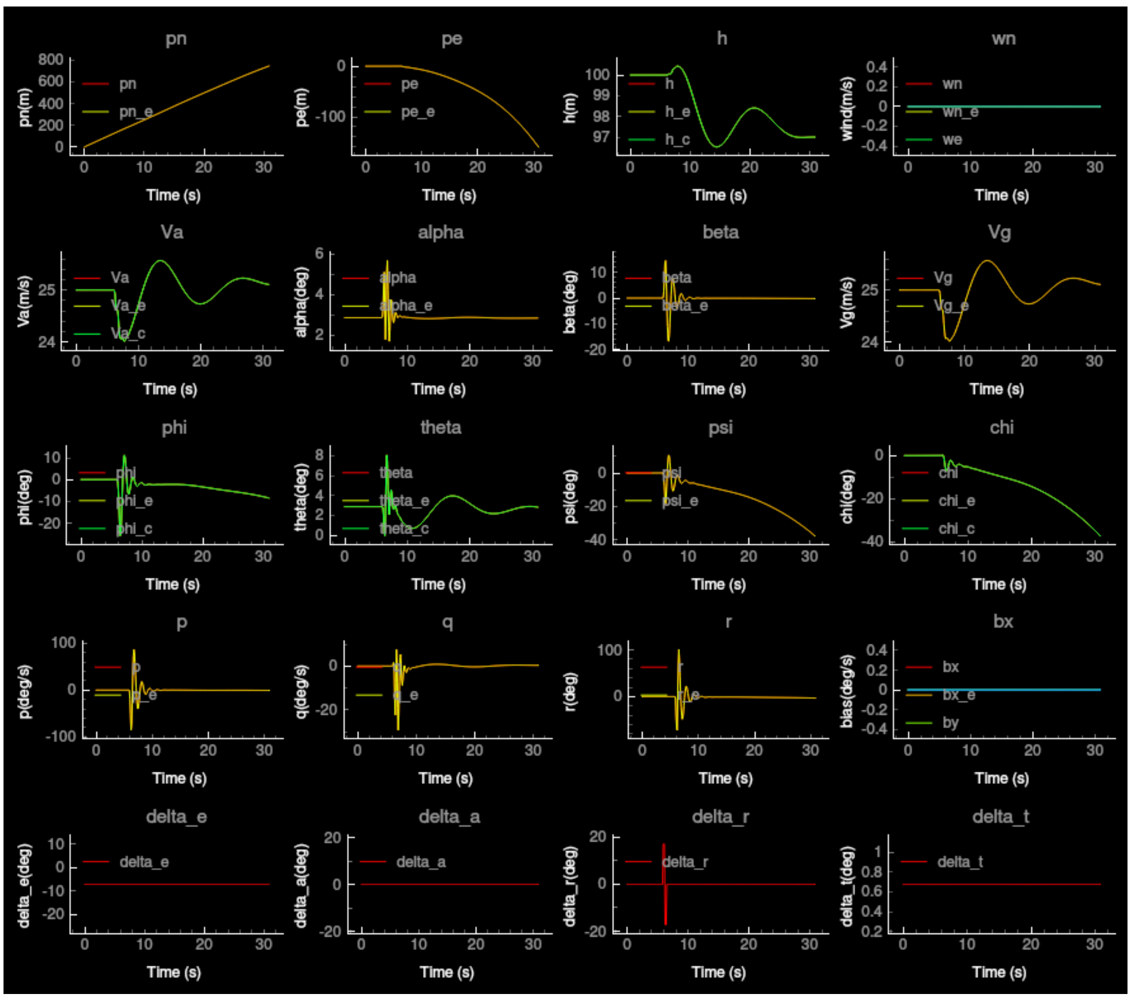
0.7 Problem 6 - Dutch roll mode

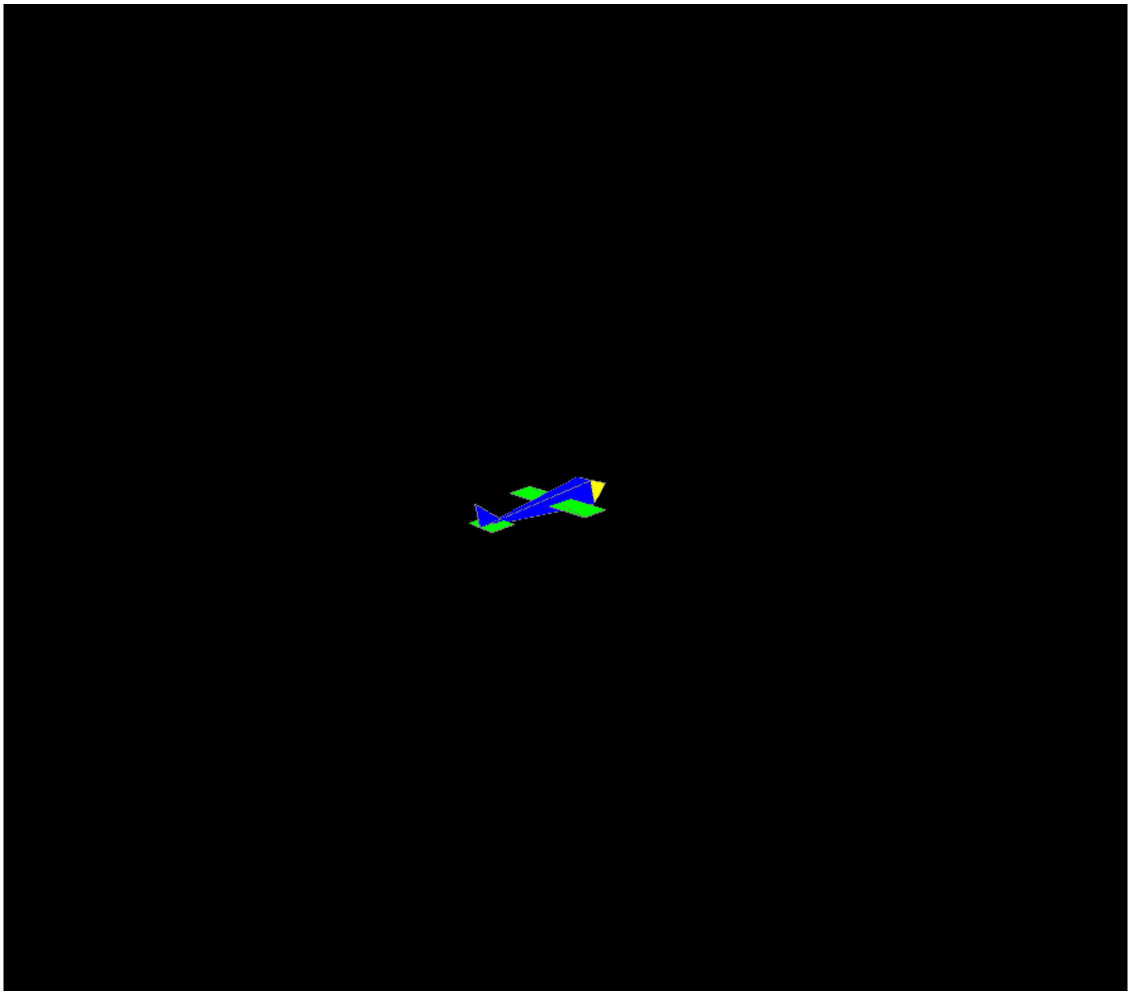
For the trim trajectory corresponding to $V_a = 25\frac{m}{s}$ and $\gamma = 0$, use a doublet to excite the dutch roll mode.

[7]:
```
# Create the trim state
Va_trim = 25.
gamma_trim = 0.
trim_state, trim_input = compute_trim(state.convert_to_numpy(), Va_trim,␣
 ↪gamma_trim)

# Create an input signal
input_signal = Signals(amplitude=0.3,
                       duration=0.3,
                       start_time=6.0)

# Create a function for exciting the phugoid mode
```

```python
def excite_dutch_roll(sim_time: float):
    # copy the trim command
    delta_cmd = MsgDelta()
    delta_cmd.copy(trim_input)

    # Excite the phugoid mode
    delta_cmd.rudder  += input_signal.doublet(sim_time)
    return delta_cmd

# Run the command
run_sim_and_display(excite_dutch_roll, trim_state)
```

```
/home/alex/spring2022/small_aircraft/Chap05_Linear_Models_start/mav_sim/chap4/ma
v_dynamics.py:254: RuntimeWarning: invalid value encountered in sqrt
  omega_p = (-b + np.sqrt(b**2 - 4*a*c)) / (2.*a)
```





[ ]: