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

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

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 weights along the diagonal. If possible, the optimization will produce trim states and inputs such that $f = f_d$.

Implement the following functions in `chap5\trim.py` using the cost and constraints discussed in lecture and described in the function docstrings.

- `velocity_constraint`
- `velocity_constraint_partial`
- `variable_bounds`
- `trim_objective_fun`

In [ ]:
```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
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)
    display_data_view(data_view)
    display_mav_view(mav_view)
```

```
Initializing mav_view
Initializing data_view
```

Problem 2 - Calculate Trim

Do the following:

1. Compute a trim state and input for $V_a = 25\frac{m}{s}$, $\gamma = 0$, and $R = \infty$
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

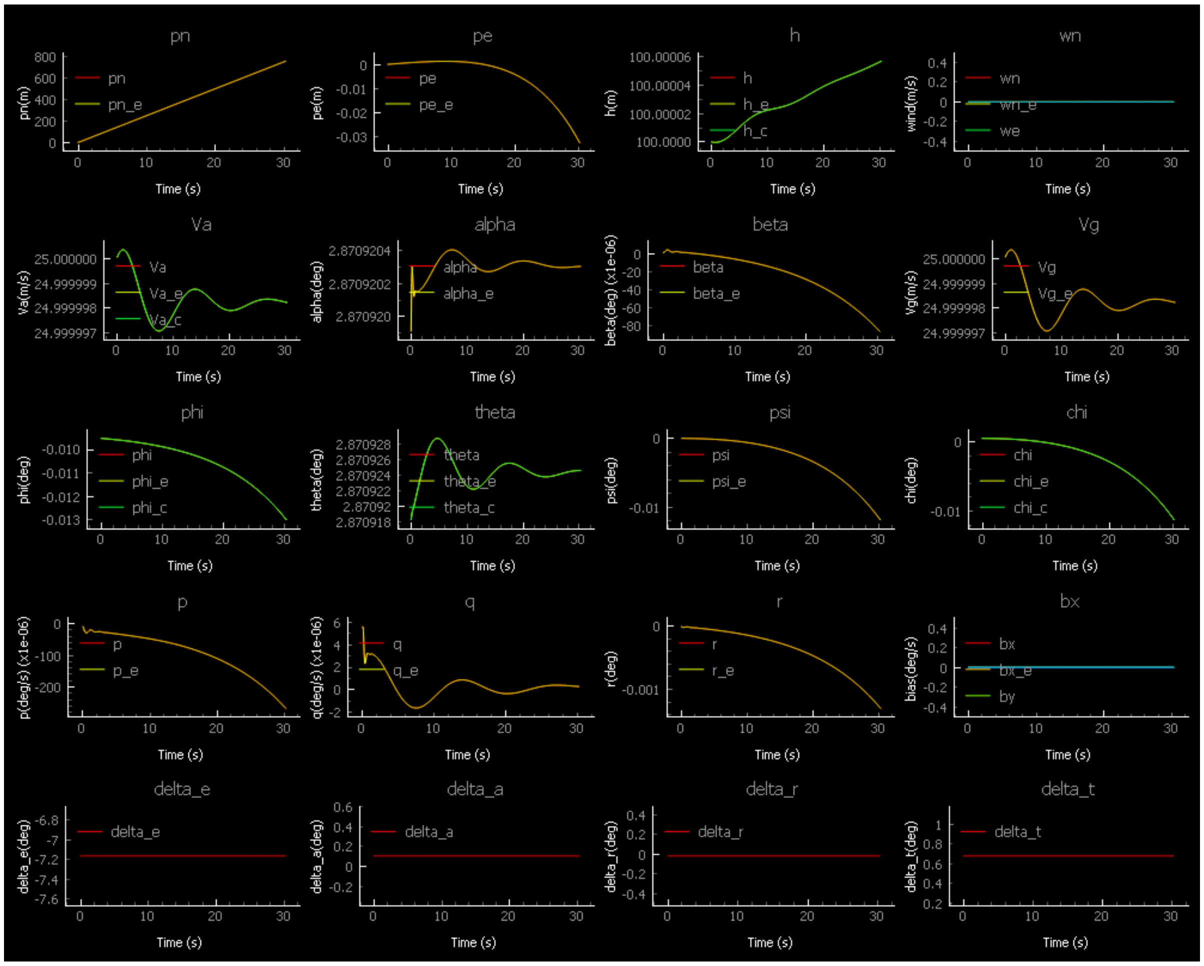
Keep in mind the following when comparing results:

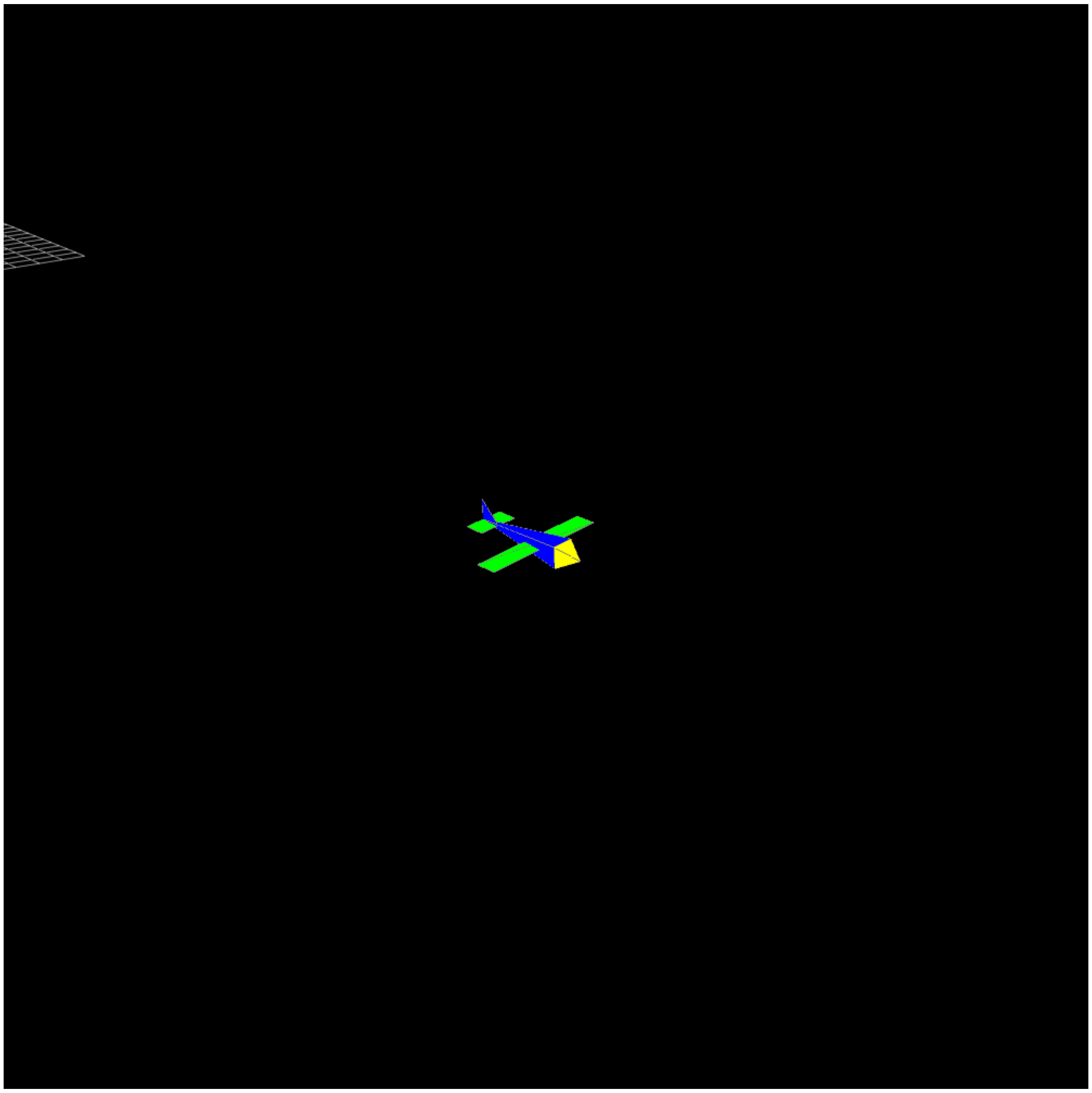
- $\dot{p_n}$ and $\dot{p_e}$ are not important
- Your trim state dynamics will not be exact, but they should be close

In [ ]:
```
# Create the trim state
Va_trim = 25.
gamma_trim = 0.
R=np.inf
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 the desired state dynamics
desired_trim_state_dot = np.array(([0.],
                        [0.],
                        [-Va_trim*np.sin(gamma_trim)],
                        [0.],
                        [0.],
                        [0.],
                        [0.],
                        [0.],
                        [0.],
                        [0.],
                        [0.],
                        [0.]))
```

```
# Calculate the actual state dynamics
[VaActual,alpha,beta,_]=update_velocity_data(trim_state)
fm = forces_moments(trim_state,trim_input,Va_trim,0,0)
f=derivatives(trim_state,fm)

# 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([[ 2.25131491e-15],
       [ 1.06848215e-13],
       [-1.00000000e+02],
       [ 2.49686227e+01],
       [ 0.00000000e+00],
       [ 1.25215085e+00],
       [ 9.99686174e-01],
       [-8.30798876e-05],
       [ 2.50508635e-02],
       [ 2.08187627e-06],
       [ 3.15099560e-21],
       [ 9.99999996e-13],
       [-7.27617752e-09]])
trim input =
elevator= -0.1250436498057945 aileron= 0.0018374628776380315 rudder= -0.0002928845
8282197925 throttle= 0.6767758538402535
Difference between actual and desired (Note that pn and pe are not important):
array([[ 2.50000000e+01],
       [ 2.08122417e-04],
       [ 7.34150048e-07],
       [-3.95462340e-01],
       [-2.26587989e-07],
       [ 5.56221002e+00],
       [-4.95138104e-15],
       [-9.11372661e-11],
       [ 1.97591080e-13],
       [-3.63694708e-09],
       [-2.79901039e-06],
       [ 5.00806667e+00],
       [-1.27356685e-06]])
```


pn
pe
h
wn
Va
alpha
beta
Vg
phi
theta
psi
chi
p
q
r
bx
delta_e
delta_a
delta_r
delta_t
Time (s)



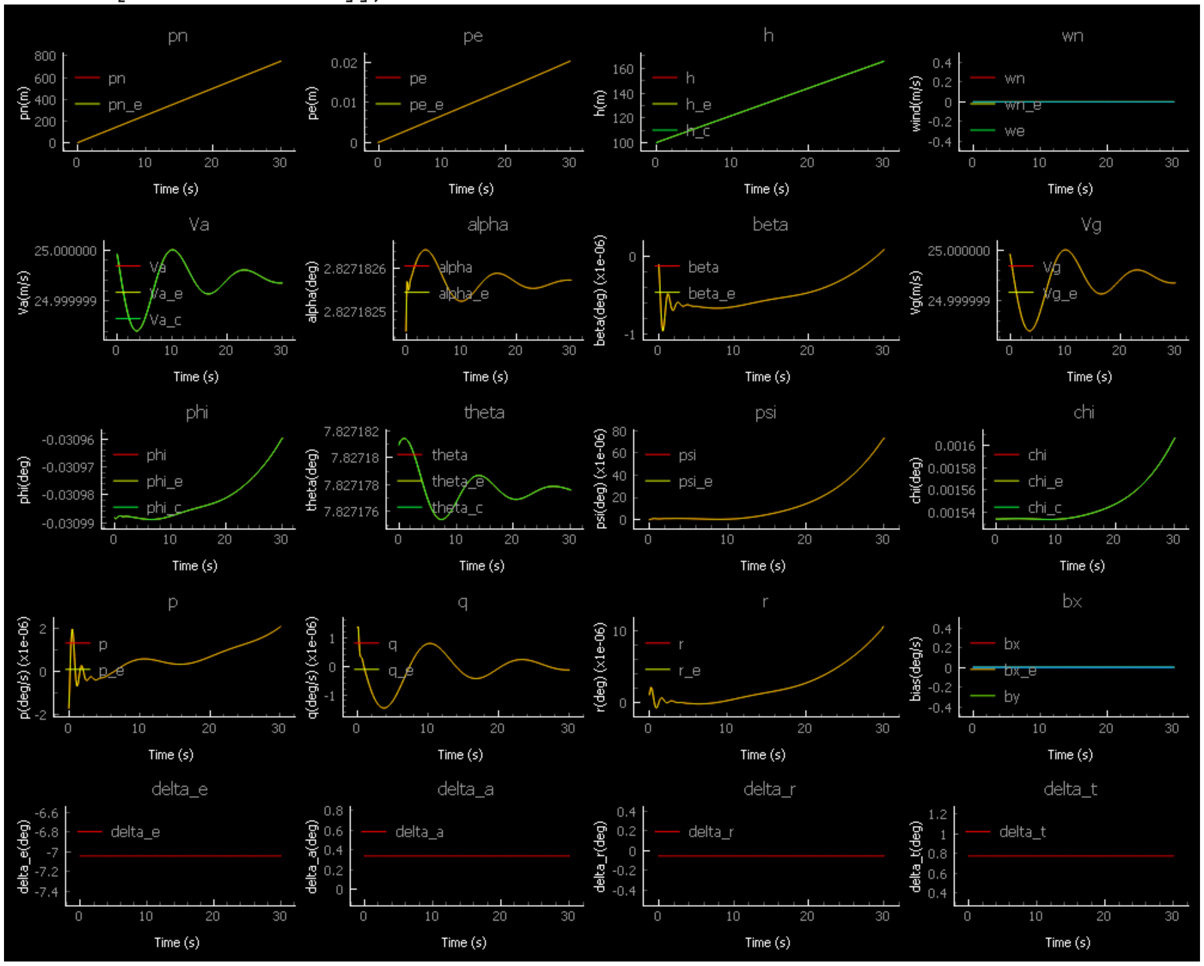
Problem 3 - Trim Calculations

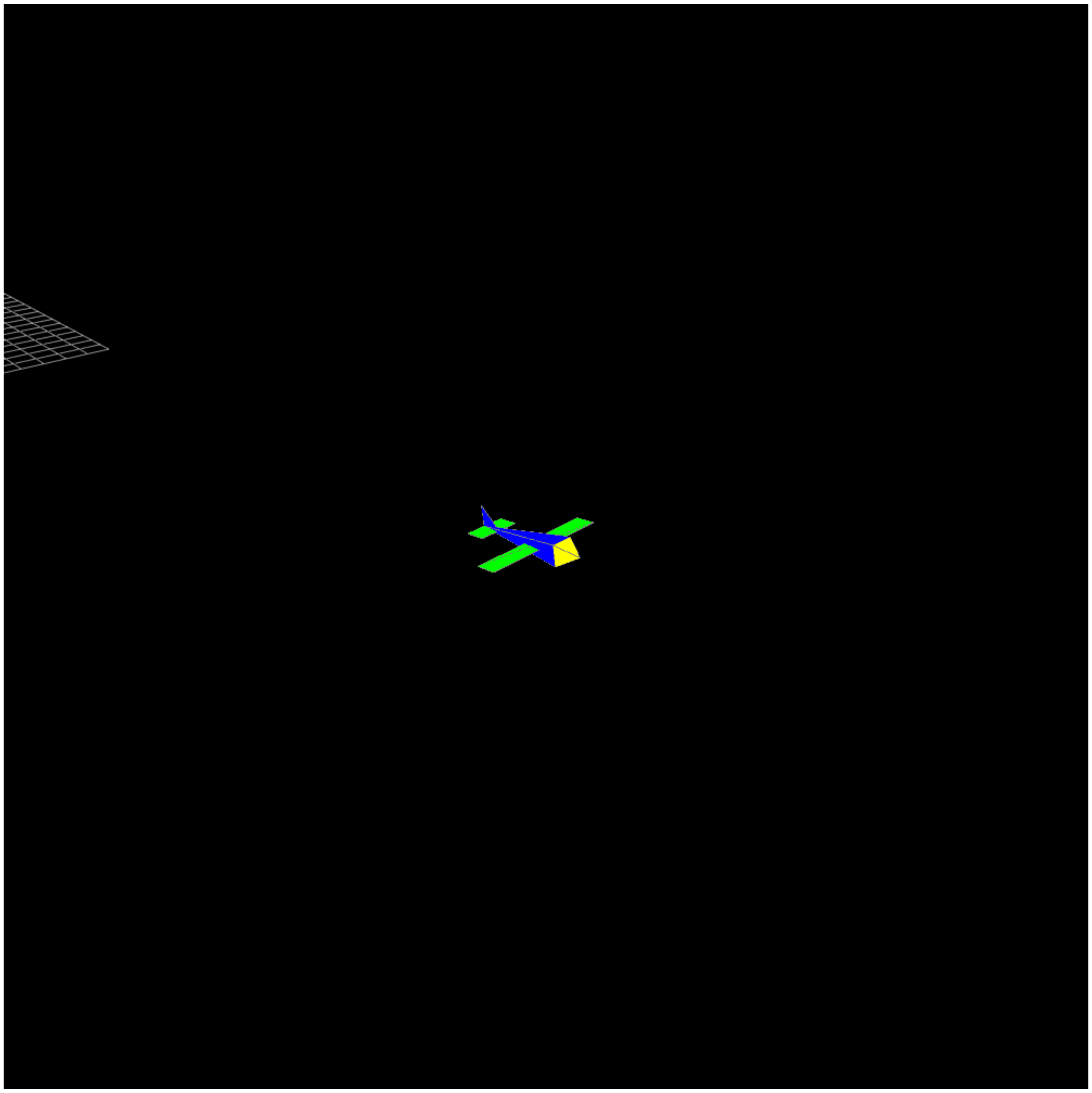
Repeat problem 2 with $V_a = 25\frac{m}{s}$, $\gamma = 5$ degrees, and $R = \infty$

```
In [ ]: # Create the trim state
        Va_trim = 25.
        gamma_trim = np.pi*5./180.
        R=np.inf
        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 the desired state dynamics
        desired_trim_state_dot = np.array(([0.],
                       [0.],
                       [-Va_trim*np.sin(gamma_trim)],
                       [0.],
                       [0.],
                       [0.],
                       [0.],
                       [0.],
                       [0.],
                       [0.],
                       [0.],
                       [0.],
                       [0.]))

# Calculate the actual state dynamics
[VaActual,alpha,beta,_]=update_velocity_data(trim_state)
fm = forces_moments(trim_state,trim_input,Va_trim,0,0)
f=derivatives(trim_state,fm)

# 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([[ 1.49483048e-15],
       [ 3.08395231e-14],
       [-1.00000000e+02],
       [ 2.49695712e+01],
       [ 6.19775862e-21],
       [ 1.23309051e+00],
       [ 9.97668081e-01],
       [-2.69793085e-04],
       [ 6.82519348e-02],
       [ 1.84569407e-05],
       [ 0.00000000e+00],
       [ 9.99999990e-13],
       [-7.43615311e-09]])
trim input =
elevator= -0.12293089441183659 aileron= 0.00593266119506005 rudder= -0.00094579852
36477303 throttle= 0.7737438667662282
Difference between actual and desired (Note that pn and pe are not important):
```

```
array([[ 2.49048675e+01],
       [ 6.66913750e-04],
       [ 4.84384234e-07],
       [-3.86069589e-01],
       [-4.29495160e-08],
       [ 5.47768228e+00],
       [ 3.44983513e-14],
       [-2.53765928e-10],
       [-5.04277310e-13],
       [-3.70940644e-09],
       [-9.99785556e-07],
       [ 4.93177002e+00],
       [ 2.42265413e-07]])
```





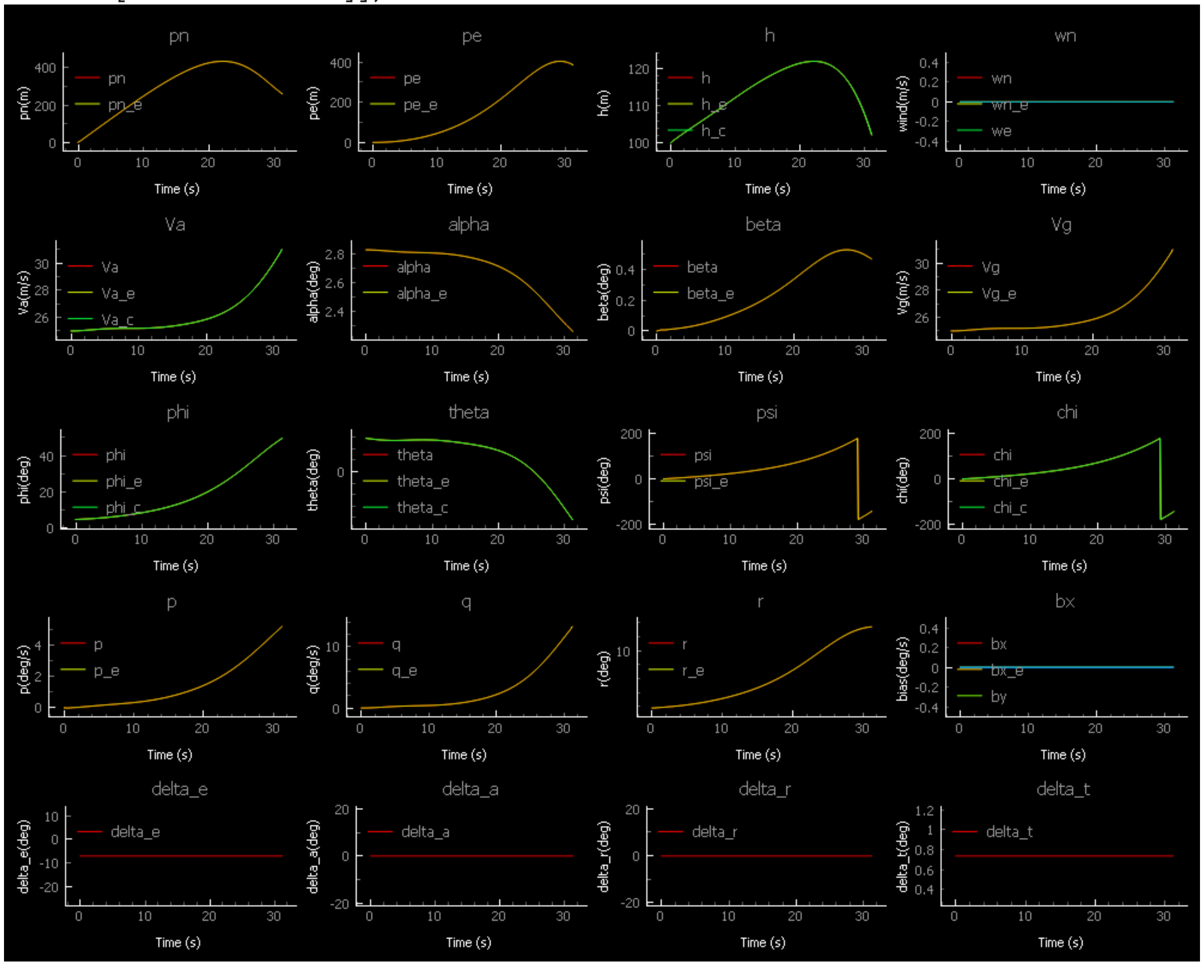
Problem 4 - Trim Calculations

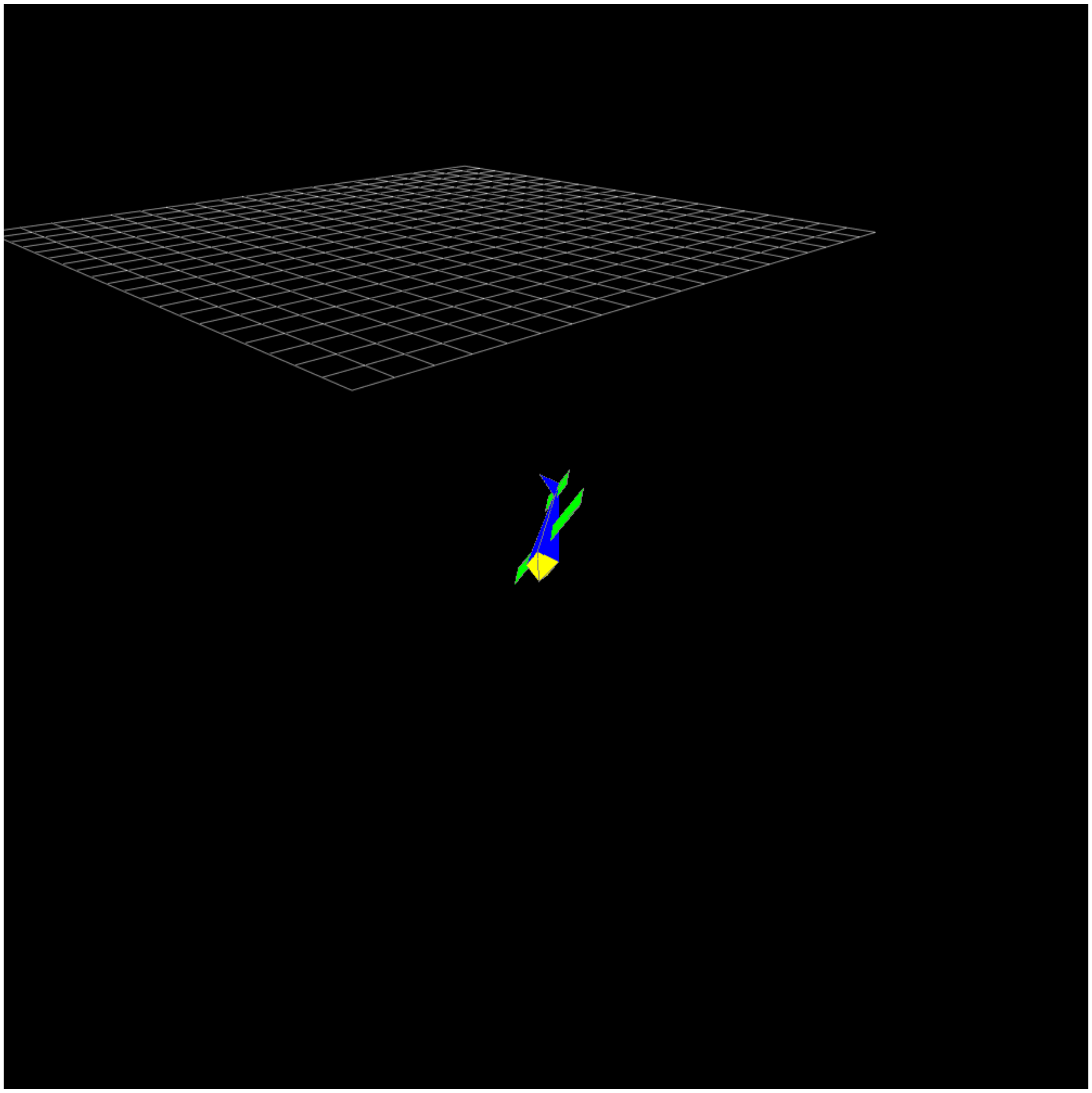
Repeat problem 2 with $V_a = 30\frac{m}{s}$, $\gamma = 3$ degrees, and $R = 800m$

In [ ]:
```
from mav_sim.chap3.mav_dynamics_euler import derivatives_euler,quat_state_to_euler_
# Create the trim state
Va_trim = 25.
gamma_trim = np.pi*3./180.
R=800.
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 the desired state dynamics
desired_trim_state_dot=np.array(([0.],
                        [0.],
                        [-Va_trim*np.sin(gamma_trim)],
                        [0.],
                        [0.],
                        [0.],
                        [0.],
                        [0.],
                        [Va_trim/R*np.cos(gamma_trim)],
                        [0.],
                        [0.],
                        [0.],))


# Calculate the actual state dynamics
[VaActual,alpha,beta,_]=update_velocity_data(trim_state)
fm = forces_moments(trim_state,trim_input,Va_trim,0,0)
f=derivatives_euler(quat_state_to_euler_state(trim_state),fm)

# 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([[ 2.22283600e-15],
       [ 2.96626917e-13],
       [-1.00000000e+02],
       [ 2.49694628e+01],
       [ 0.00000000e+00],
       [ 1.23528419e+00],
       [ 9.97898666e-01],
       [ 4.02272812e-02],
       [ 5.07526568e-02],
       [-2.04594060e-03],
       [ 1.52629078e-18],
       [ 9.99999992e-13],
       [ 3.11471963e-02]])
trim input =
elevator= -0.12317119995717175 aileron= 0.0017011983845397118 rudder= -0.002754663
1757894365 throttle= 0.7367055262745735
Difference between actual and desired (Note that pn and pe are not important):
```

```
array([[ 2.49655406e+01],
       [-9.94319460e-02],
       [ 4.01738281e-06],
       [-3.87146078e-01],
       [-8.22825646e-06],
       [ 5.48741153e+00],
       [ 3.16617291e-03],
       [-2.50713670e-03],
       [-1.42451559e-08],
       [-1.14672231e-06],
       [ 4.94055094e+00],
       [-8.26361681e-07]])
```





Problem 5 - 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

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

-4.87859823+9.86957041j, -4.87859823-9.86957041j

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

-0.10413335+0.48908573j, -0.10413335-0.48908573j

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

8.93943127e-02+0.j,

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

-2.24411599e+01+0.j,

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

-1.14075620e+00+4.65506199j, -1.14075620e+00-4.65506199j

In [ ]:
```
from mav_sim.chap5.compute_models import compute_ss_model
Va_trim = 25.
gamma_trim = np.pi*0./180.
R=np.inf
trim_state, trim_input = compute_trim(state.convert_to_numpy(), Va_trim, gamma_trim
# Compute the trim state and input
[A_lon,_,A_lat,_]=compute_ss_model(trim_state,trim_input)

# Compute A_lon and A_lat

# Compute the eigenvalues of A_lon
[eiganvaluesAlong,_]=np.linalg.eig(A_lon)
display(eiganvaluesAlong)
# Compute the eigenvalues of A_lat
[eiganvaluesAlat,_]=np.linalg.eig(A_lat)
display(eiganvaluesAlat)
```

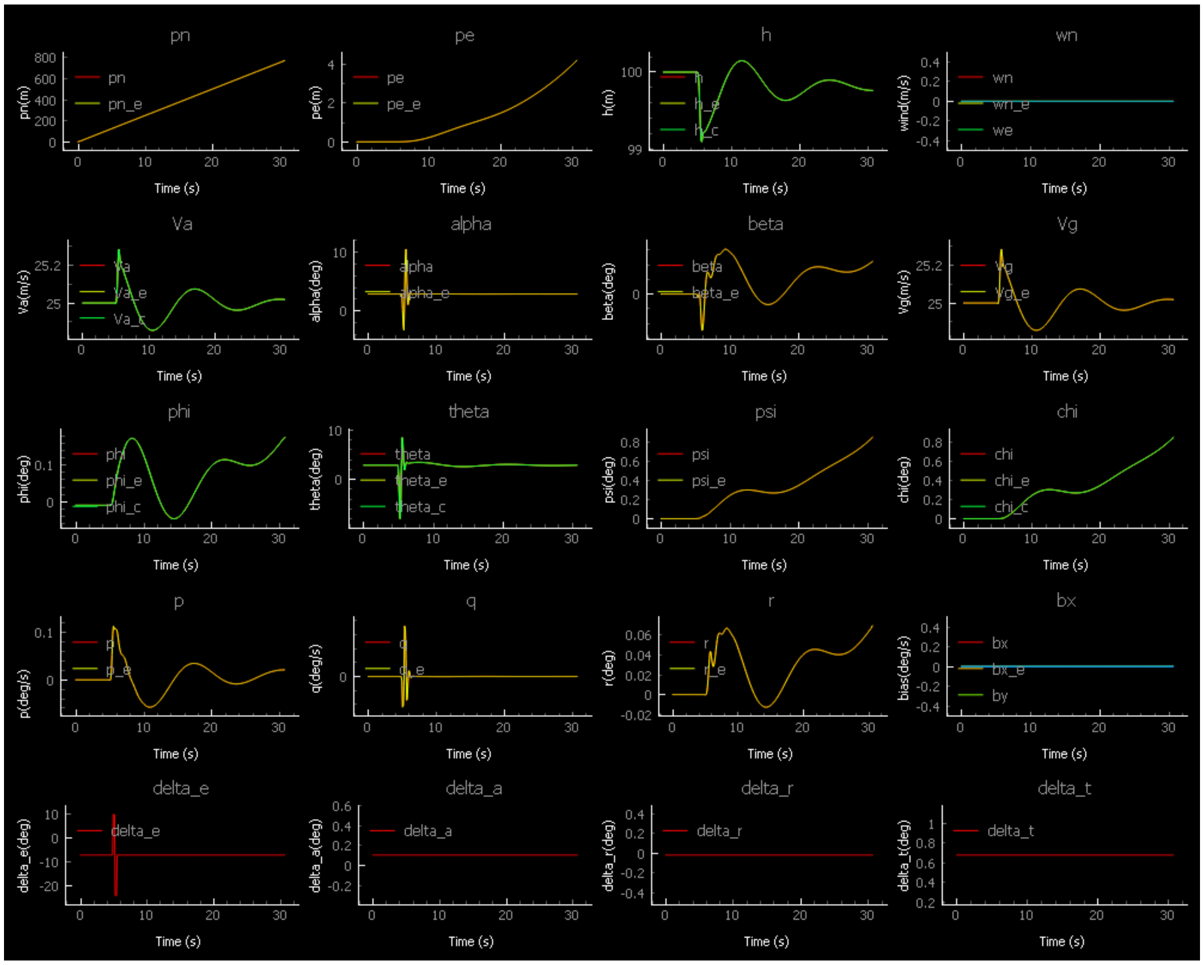
```
array([ 0.        +0.j        , -4.87859823+9.86957041j,
       -4.87859823-9.86957041j, -0.10413335+0.48908573j,
       -0.10413335-0.48908573j])
array([-2.24411599e+01+0.j        , -1.14075620e+00+4.65506199j,
       -1.14075620e+00-4.65506199j,  8.93943127e-02+0.j        ,
        3.02906480e-12+0.j        ])
```

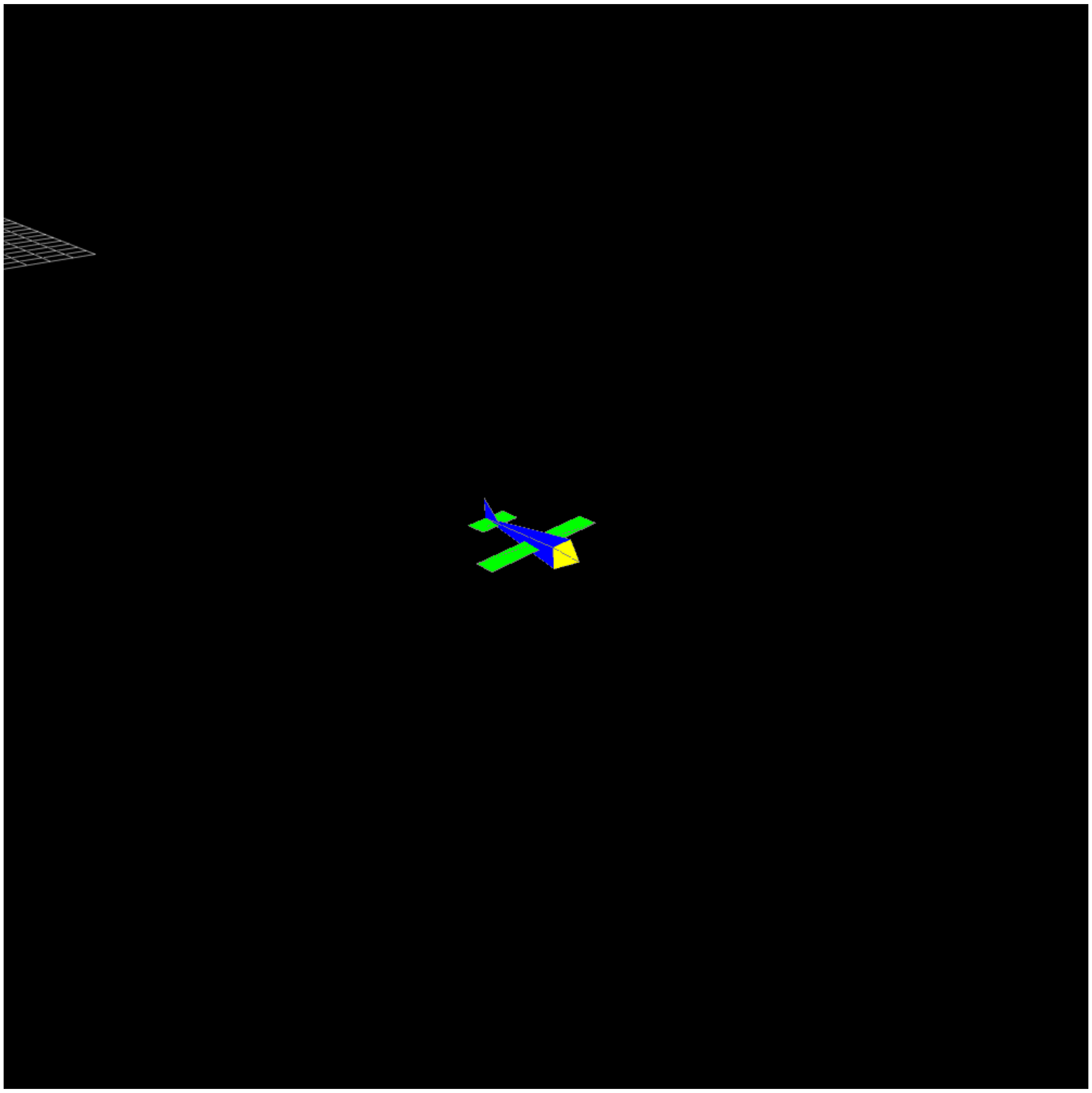
Problem 6 - Phugoid mode

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)

In [ ]:
```
# 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=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)
```

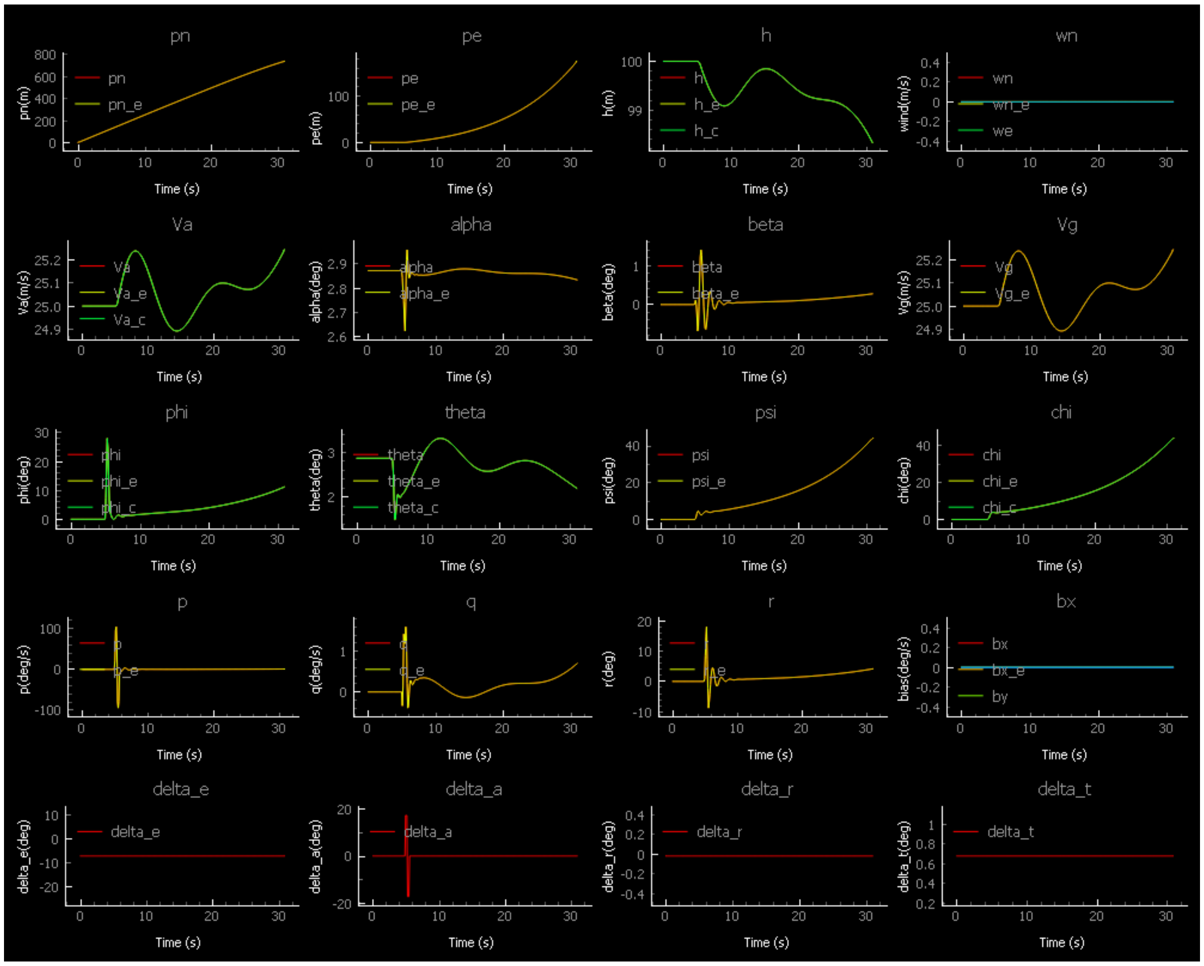


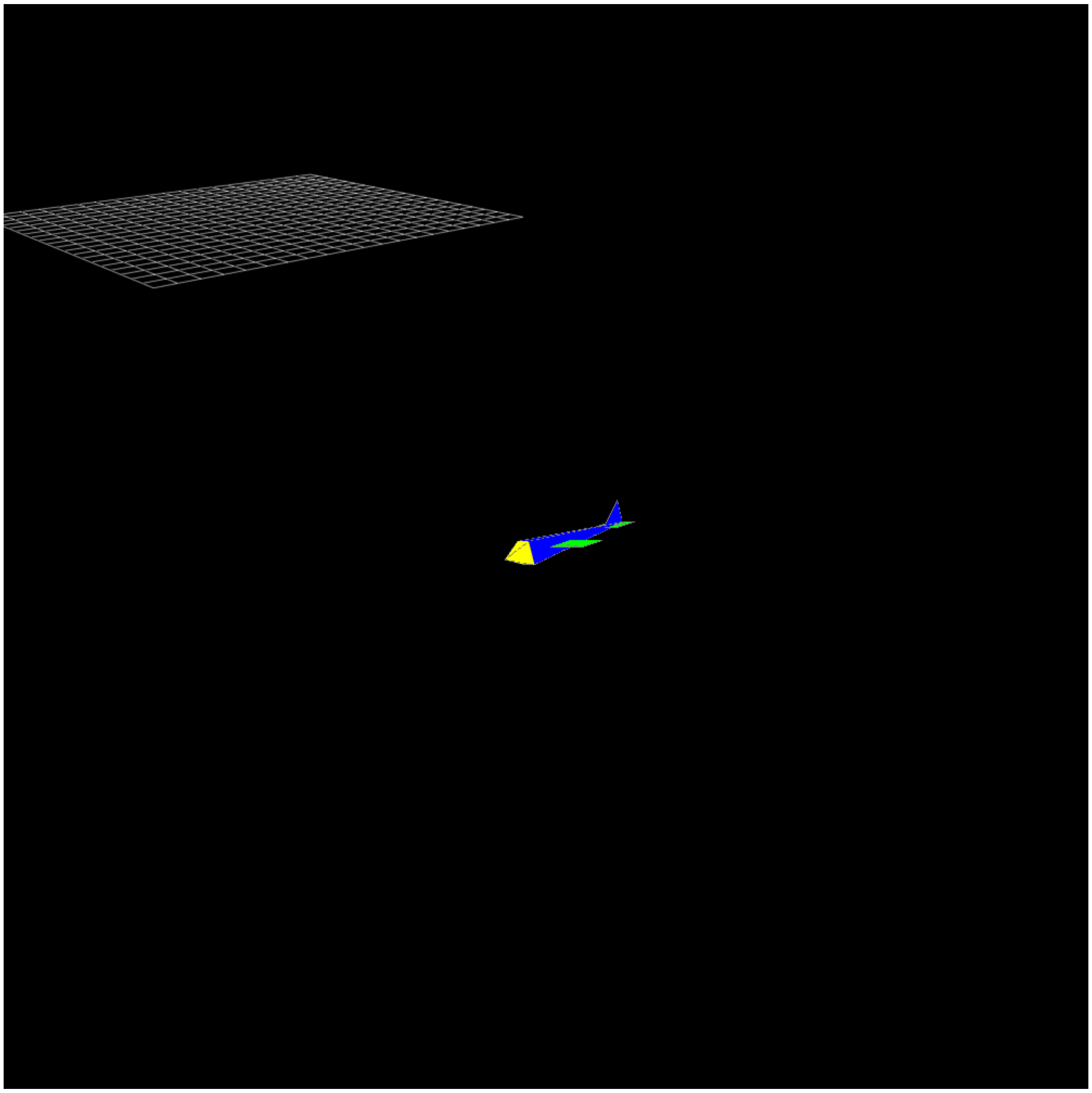


Problem 7 - Roll and spiral divergence modes

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.

```
In [ ]: # 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=5.0)

# Create a function for exciting the phugoid mode
def excite_Roll_Spiral(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)
    return delta_cmd

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

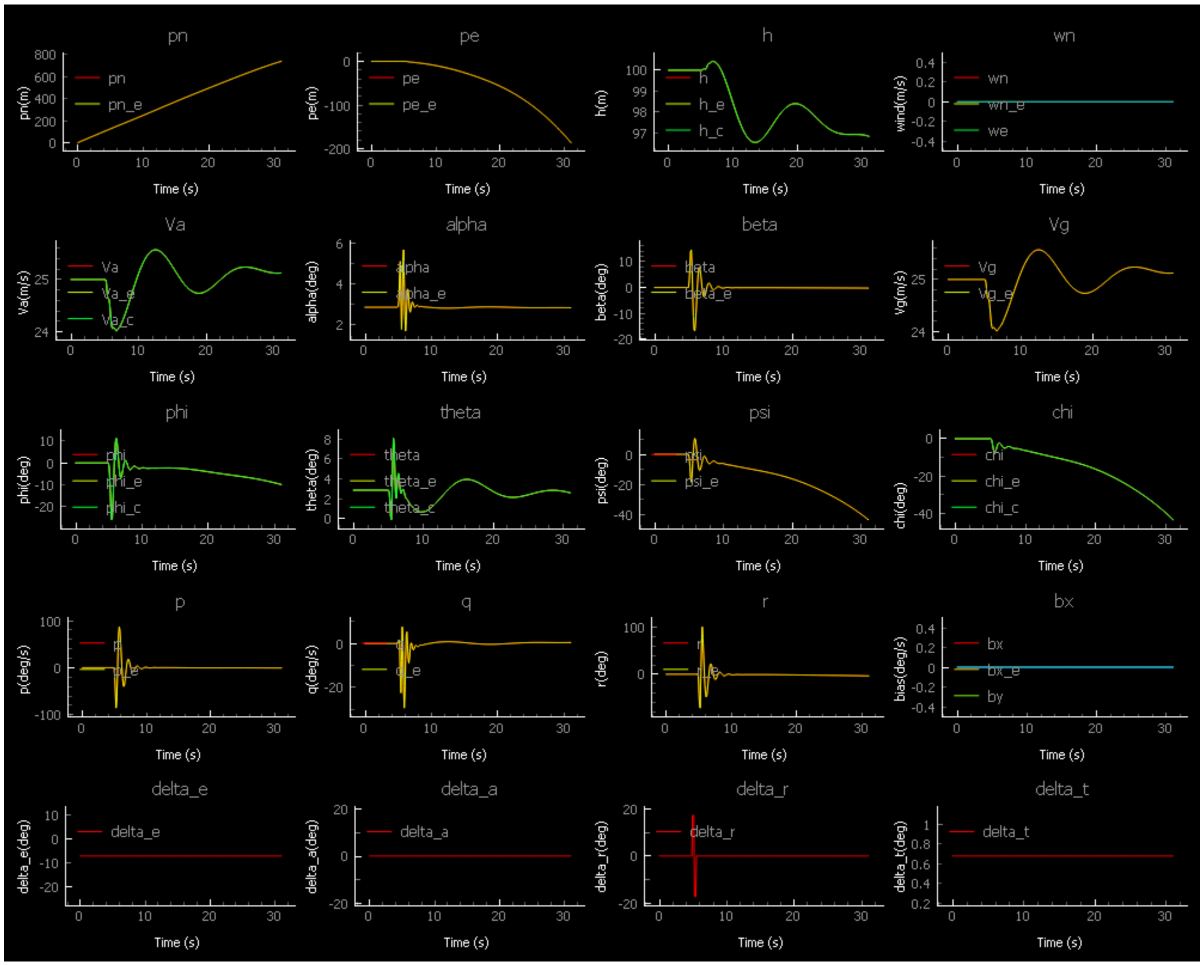


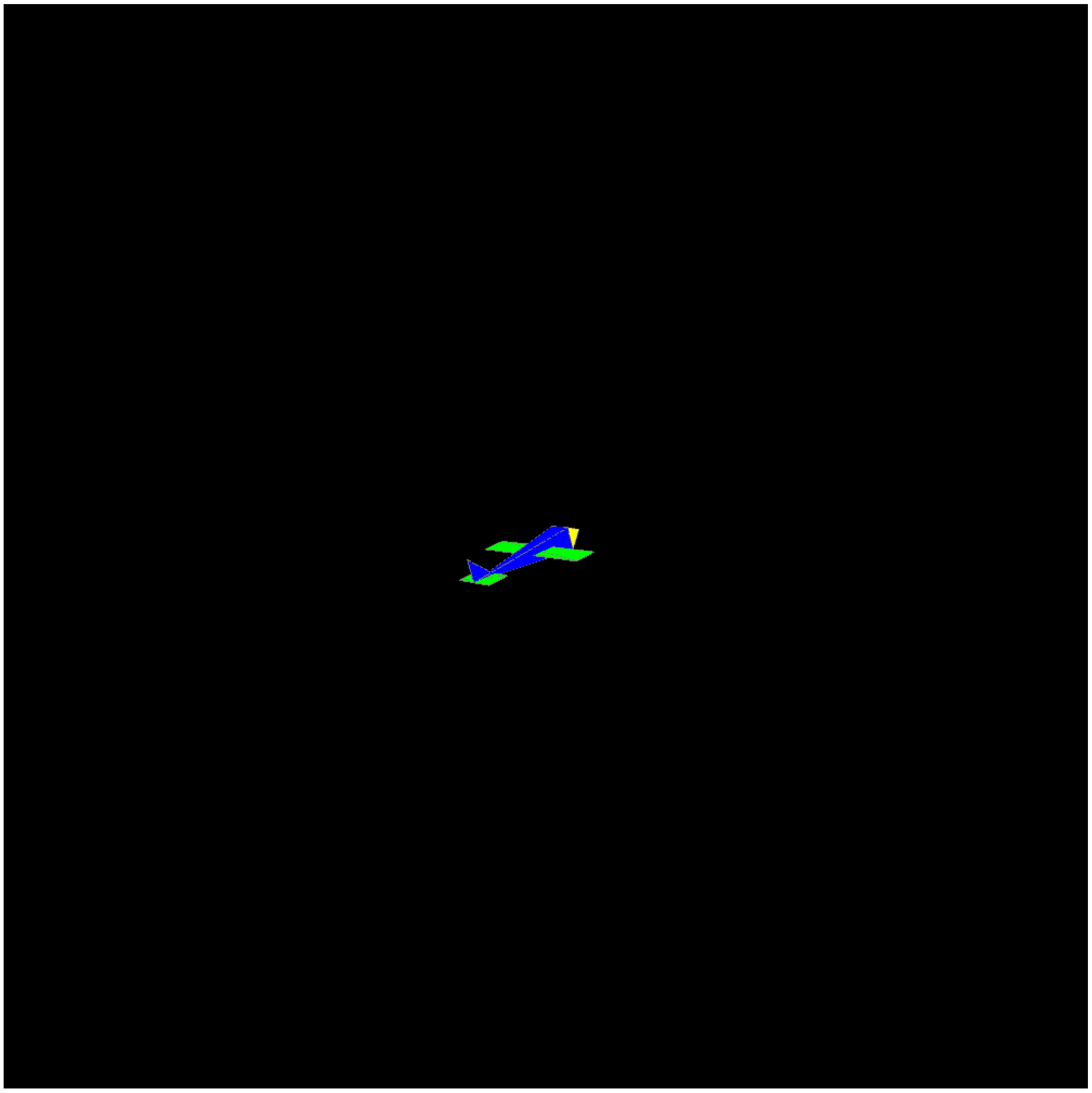


Problem 8 - Dutch roll mode

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

In [ ]:
```
# 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=5.0)

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





Static analysis

Run the static code analysis (you must have zero static code analysis errors to get credit). You may not modify the static code analysis configuration files.

ISORT

Run Isort:

```
python -m isort mav_sim book_assignments
```

Terminal output (should be nothing):

MyPy

Run MyPy

```
python -m mypy mav_sim/chap2/ mav_sim/chap3/ mav_sim/chap4/
mav_sim/chap5/ book_assignments
```

Terminal output (should indicate no error):

```
Success: no issues found in 31 source files
```

Pylint

Run Pylint

```
python -m pylint --jobs 0 --rcfile .pylintrc mav_sim/chap2/
mav_sim/chap3/ mav_sim/chap4/ mav_sim/chap5/ book_assignments/
```

Terminal output (should indicate `10/10`)

```
Your code has been rated at 10.00/10 (previous run: 9.96/10, +0.04)
```

Simple code checking

The following code does not need to change. It should just be used as a sanity check so that you know the code is implemented properly. The output should not have any lines reading `Failed test!`

Note that due to the random nature of the test, you may get a warning stating

```
RuntimeWarning: invalid value encountered in sqrt
  omega_p = (-b + np.sqrt(b**2 - 4*a*c)) / (2.*a)
```

Just ignore that warning.

In [ ]:
```
from mav_sim.unit_tests.ch5_dynamics_test import run_auto_tests, VelocityConstraint
run_auto_tests()
```

```
Starting velocity_constraint test
Passed test on velocity_constraint

Starting velocity_constraint_partial test
Passed test on velocity_constraint_partial

Starting variable_bounds test
Passed test on variable_bounds

Starting trim_objective_fun test
```

```
c:\users\a02303304\mae5330\mavsim_python\mav_sim\chap4\mav_dynamics.py:267: Runtim
eWarning: invalid value encountered in sqrt
  Omega_p=(-b+np.sqrt(b**2-4*a*c))/(2*a)
```

```
Passed test on trim_objective_fun
```