

MAV Simulation Unit Tests

Dan Richards

18 April 2022

1 Introduction

The aim of this project was to develop unit tests for Chapters 3, 4, and 5 of the MAV Simulation Lab in order to facilitate easier grading and debugging. The library, *pytest*, was used to enable the tests. The format of these tests were intended to match the formatting of the existing *mavsim_python* repository. To accomplish this, pairs of files for each chapter exist in a unit tests folder. One file contains each of the test functions for the chapter (see Figure 1). The other file contains the true values for each chapter. To align with requirements of *pytest*, each of the test files must have the word *test* in the file name, as well as in the names of each test function within that file (see Figure 1).

```
194 def test_update_Mz():
195     # initialize elements of the architecture
196     mav = MavDynamics(SIM.ts_simulation)
197     # set state
198     mav._state = np.zeros((13,1),dtype=float)
199     mav._state[3,0] = 25.0 # set u0
200     mav._state[6,0] = 1.0 # set e0
201     # set forces and moments
202     forces_moments = np.zeros((6,1),dtype=float) # fx, fy, fz, Mx, My, Mz
203     forces_moments[5,0] = 0.1 #set Mz
204     # get update results
205     mav.update(forces_moments) # propagate the MAV dynamics
206     #compare against true values
207     assert np.allclose(mav._state, trueValues.state_Mz)
```

Figure 1

```
121 state_Mz = np.array( [[ 2.50000000e-01],
122                      [-6.69747058e-20],
123                      [-1.25415774e-14],
124                      [ 2.50000000e+01],
125                      [-7.17806614e-05],
126                      [ 3.04437995e-11],
127                      [ 1.00000000e+00],
128                      [ 2.09665008e-07],
129                      [ 3.07878132e-13],
130                      [ 1.43561323e-06],
131                      [ 8.38660032e-05],
132                      [ 2.46302506e-10],
133                      [ 5.74245291e-04]])
```

Figure 2

1.1 Chapter 3 Tests

The unit tests for Chapter 3 test both the *_derivatives* and *update* functions from the MAV dynamics file. Separate tests are performed for each force and moment: fx, fy, fz, Mx, My, and Mz.

1.2 Chapter 4 Tests

The unit tests for Chapter 4 test the *_motor_thrust_torque*, *_forces_moments*, and *_update_velocity_data* functions from the MAV dynamics file. The *_motor_thrust_torque* function is test with a change in the throttle command. To test the *_forces_moments* function, separate tests are performed for aileron, elevator, and rudder commands. To test the *_update_velocity_data* function, separate tests are performed for wind coming from each of the cardinal directions.

1.2 Chapter 5 Tests

For Chapter 5, unit tests were written for the *compute_trim* function in the *trim.py* file, and the *f_euler*, *df_dx*, *df_du*, *compute_ss_model*, and *compute_tf_model* functions in the *compute_model.py* file.

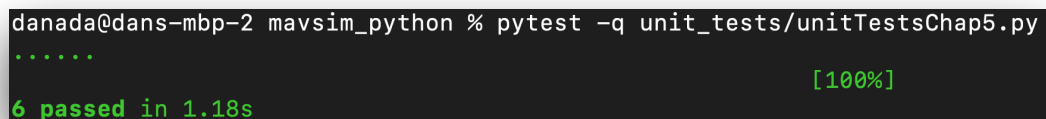
Method

- How is it tested
- What do the results show and what do they mean

To run these tests, the following command can be run in the terminal:

```
cd /<mavsim_python directory path>/mavsim_python  
pytest -q unit_tests/unitTestsChap4.py
```

After this command has been executed, either all tests passed (see Figure 3) or at least one of the tests failed (see Figure 4).



```
danada@dans-mbp-2 mavsim_python % pytest -q unit_tests/unitTestsChap5.py  
.....  
[100%]  
6 passed in 1.18s
```

Figure 3

For each of the tests that fail, details of that test function and its results are shown (see Figure 4). First, the test function is shown. Next is shown the failed logical assertion statement within the function and the values of the two variables that were compared. Third, a section displaying any statements printed through *stdout*. Last, the test summary showing the names of each test function that failed and the count of passed and failed tests.

Conclusion

These test should allow students to have an easier time debugging their code in the early chapters when much of the assignment includes coding equations.

If there are any changes to the structure of the *mav_sim* files, or any other changes or reasons, the true values can easily be changed using the debugging tool of an IDE to obtain the necessary intermediate values, so the students have accurate values against which they can test their own functions. The method I used to do this is shown in Figure 5. Using Visual Studio Code's debugging tool, breakpoints were set so I could get a variable's value at a certain point

