Department of Mechanical Engineering
University of Bath

ME40064: System Modelling & Simulation
ME50344: Engineering Systems Simulation

Tutorial 1: Unit Testing & Data Structures

Part A: Writing Unit Tests

1. The tutorial will begin by following Matlab's example provided in the documentation (the specific page can be found by searching for "Write Script-Based Unit Tests" in the Matlab Help).

In it you will first write a function that generates information about a right-angled triangle. First, create the function, `rightTri.m`, to contain the following code:

```
function [A,B,C] = rightTri(a,b)

     C = 90;
     A = atand(a/b);
     B = atand(b/a);

end
```

Next you will create the script file that will contain the unit tests for this function. Create a file with the filename `rightTriTest.m` and type the following code into it:

```
%% Test 1: sum of angles
% Test that all angles add up to 180 degrees.
[A,B,C] = rightTri(7,9);
assert(A+B+C == 180)

%% Test 2: isosceles triangles
% Test that if sides a and b are equal, then
% angles A and B are equal.
[A,B,C] = rightTri(4,4);
assert(A == B)

%% Test 3: 3-4-5 triangle
% Test that if side a is 3 and side b is 4, then
% side c (hypotenuse) is 5.
[A,B,C] = rightTri(3,4);
c = 3/sind(A);
assert(isequal(c,5))

%% Test 4: 30-60-90 triangle
% Test that if side a is 1 and side b is sqrt(3),
% the angles A and B are 30
% and 60 degrees, respectively.
[A,B,C] = rightTri(1,sqrt(3));
assert(A == 30)
```

```
assert(B == 60)
```

To run the unit test, enter the following into the command window:

```
result = runtests('rightTriTest');
```

Note that it is the `assert()` command within the unit test script that is checking the condition provided within the parentheses, and returning the error message is the condition is not met.

You should see the following output:

```
Running rightTriTest
...
================================================================================
Error occurred in rightTriTest/Test4_30_60_90Triangle and it did not run to completion.

    --------------
    Error Details:
    --------------
    Assertion failed.

================================================================================
.
Done rightTriTest
__________

Failure Summary:

     Name                                  Failed  Incomplete  Reason(s)
    =====================================================================
     rightTriTest/Test4_30_60_90Triangle    X         X       Errored.
```

Test 4 is poorly written as it is comparing floating-point values directly, and the test fails for this reason, and not because the function `rightTri` is wrong. Therefore, modify test 4 to be the following:

```
%% Test 4: 30-60-90 triangle
% Test that if side a is 1 and side b is sqrt(3),
% the angles A and B are 30
% and 60 degrees, respectively.
tol = 1e-14;
[A,B,C] = rightTri(1,sqrt(3));
assert(abs(A-30) <= tol)
assert(abs(B-60) <= tol)
```

This now tests whether the difference between the answer and the known correct result is smaller than the specified tolerance, `tol`. This is a more robust way to test a floating-point value, as it won't fail due to the inherent finite precision of floating-point arithmetic. When run it should produce the following output:

```
Running rightTriTest
....
Done rightTriTest
_________
```

Explore the data structure for the test results and access different elements of the results as shown in the lecture, for example, the elapsed times for each test.

2. Now that you know how to run a set of unit tests within Matlab, you will write your own to help you write code to perform the following mathematical operation. The function must take an input argument that is a vector of values, **x**, which is of length N, i.e.

$$\mathbf{x} = (x_1, x_2, \ldots, x_N)$$

You must compute the normalised sum of squares of this vector:

$$1/N \sum_{i=1}^{N} x_i^2$$

Consider what would happen (or could happen if the code was incorrectly written) in the cases:

- six different positive, real data points, x, are provided
- that some of the input values, x, are negative
- the answer you are writing the test for is a floating point number

You are trying to demonstrate that the function returns the correct answer when provided with input of all validities, and also must ensure that your unit tests will correctly and robustly check the expected result.

Part B: Matlab Data Structures

1. Following the example given in the lecture create the variable, `car`, and populate it with the following members variables, using the 'dot' notation:

```
- bhp
- colour
- weight
- wheels
- fuel_type
- model_name
```

Now turn this into an array i.e. `car(1)`, `car(2)`, `car(3)`, etc. such that each entry of the `car` array represents a different type of car (real or fictional), possible examples could be Reliant Robin, Tesla Model 3, Ferrari 250 GTO, and Ford Focus.

2. Write a function to compute the power-to-weight ratio of your car and store the answer in a new variable: `car.pwr` .

3. Write a function to compare the power-to-weight ratio performance of these cars stored in this array, ranking them in order from most to least powerful. You might wish to investigate Matlab's sorting algorithm.