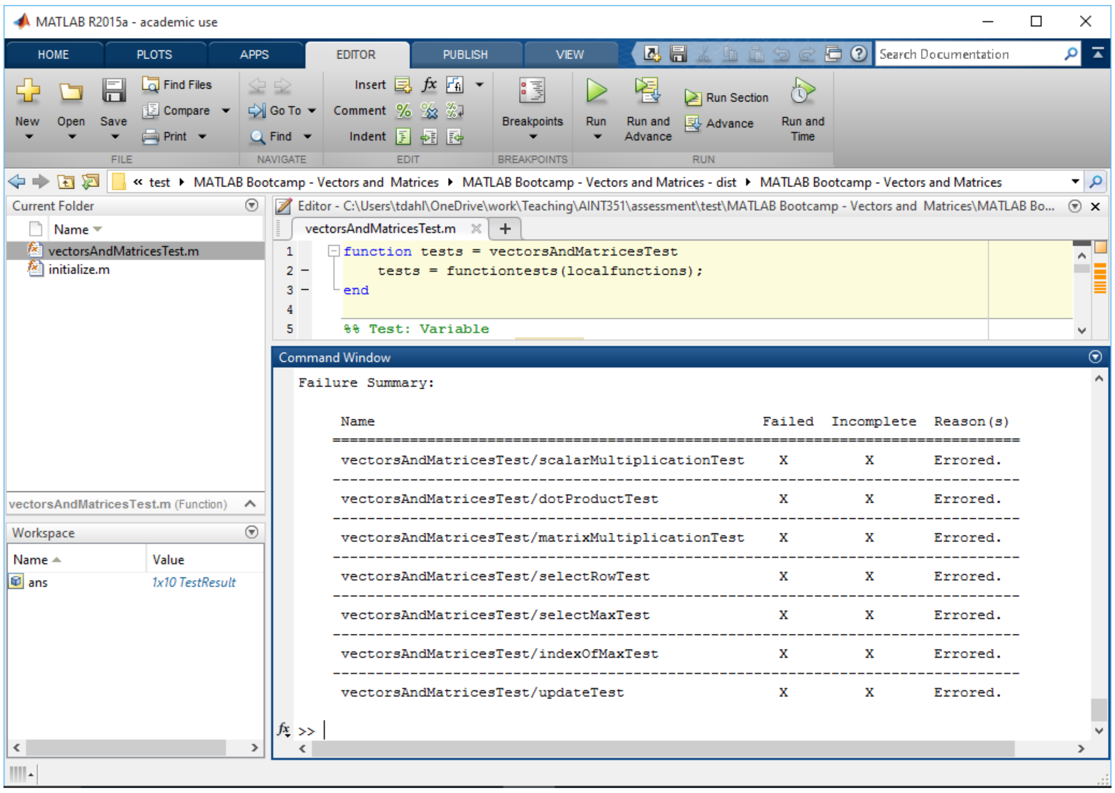
Vectors and Matrices

This practical requires you to use the MATLAB programming language to implement a set of specific features described below. The practical is based loosely on the requirements of a reinforcement learning (RL) algorithm which learns to associate observations and actions with potential future rewards.

Test-Driven Development

Start by download and extracting the zip archive provided. The archive contains a set of MATLAB files, including some that you need to edit. The archive also contains a unit test file *vectorAndMatricesTest.m* that enables automated marking of your MATLAB code.

To begin programming, start MATLAB and change the *MATLAB current folder* to the folder you just extracted from the zip archive. Next, enter the command *runtests('vectorsAndMatricesTest');* in the *MATLAB Command Window.* This should produce the output shown in the in Figure 1, indicating that all the given tests have failed. The error messages also includes explicit feedback on why your tests failed and can be a very helpful development tool.



```
function tests = vectorsAndMatricesTest
    tests = functiontests(localfunctions);
end

%% Test: Variable
```

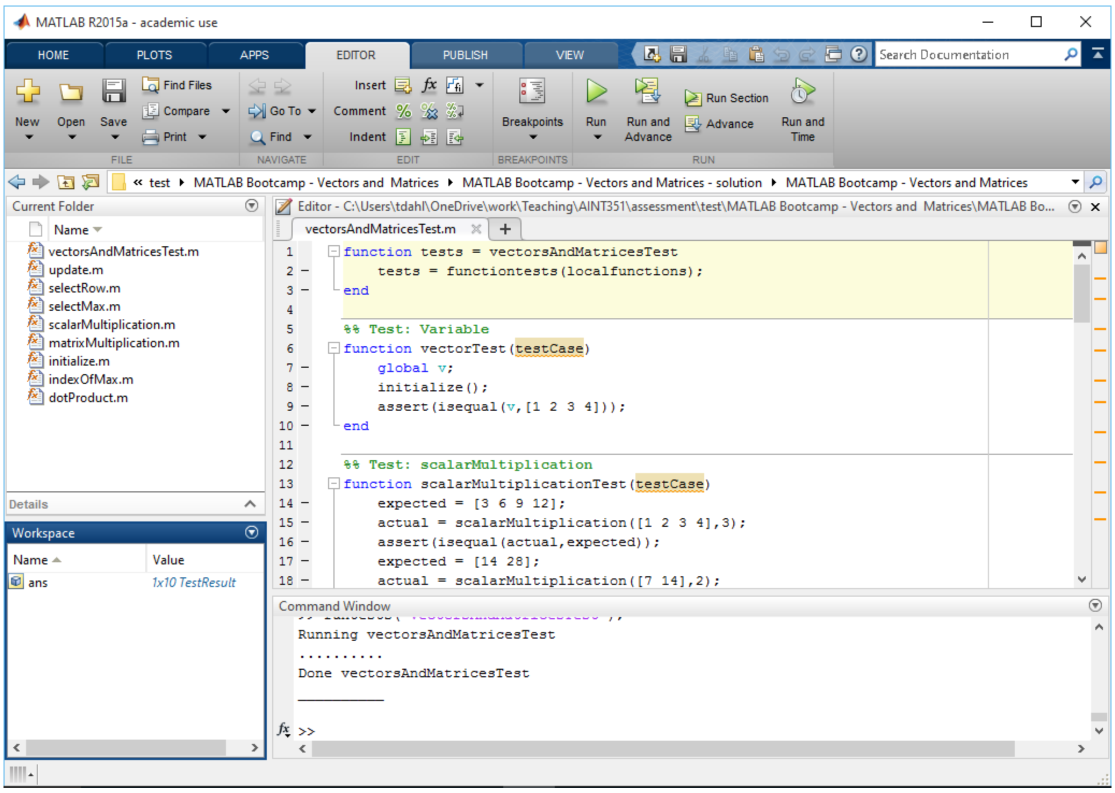
```
Failure Summary:

     Name                                              Failed  Incomplete  Reason(s)
    ================================================================================
     vectorsAndMatricesTest/scalarMultiplicationTest     X          X       Errored.
    --------------------------------------------------------------------------------
     vectorsAndMatricesTest/dotProductTest               X          X       Errored.
    --------------------------------------------------------------------------------
     vectorsAndMatricesTest/matrixMultiplicationTest     X          X       Errored.
    --------------------------------------------------------------------------------
     vectorsAndMatricesTest/selectRowTest                X          X       Errored.
    --------------------------------------------------------------------------------
     vectorsAndMatricesTest/selectMaxTest                X          X       Errored.
    --------------------------------------------------------------------------------
     vectorsAndMatricesTest/indexOfMaxTest               X          X       Errored.
    --------------------------------------------------------------------------------
     vectorsAndMatricesTest/updateTest                   X          X       Errored.

>> 
```

Figure 1: The output from running the provided tests on the given files. Five failed tests are indicated.

You can run the tests at any time, and as you add code, the list of failed tests will get shorter until your code passes all the tests. The output when your code passes all tests is shown in Figure 2.



MATLAB R2015a - academic use

Editor - C:\Users\tdahl\OneDrive\work\Teaching\AINT351\assessment\test\MATLAB Bootcamp - Vectors and Matrices\MATLAB Bo...

Current Folder:
vectorsAndMatricesTest.m
update.m
selectRow.m
selectMax.m
scalarMultiplication.m
matrixMultiplication.m
initialize.m
indexOfMax.m
dotProduct.m

Workspace:

Name	Value
ans	1x10 TestResult

vectorsAndMatricesTest.m

```
function tests = vectorsAndMatricesTest
    tests = functiontests(localfunctions);
end

%% Test: Variable
function vectorTest(testCase)
    global v;
    initialize();
    assert(isequal(v,[1 2 3 4]));
end

%% Test: scalarMultiplication
function scalarMultiplicationTest(testCase)
    expected = [3 6 9 12];
    actual = scalarMultiplication([1 2 3 4],3);
    assert(isequal(actual,expected));
    expected = [14 28];
    actual = scalarMultiplication([7 14],2);
```

Command Window

```
Running vectorsAndMatricesTest
..........
Done vectorsAndMatricesTest
__________

>>
```

Figure 2: The test output when your code satisfies all tests

Required Features

1. **Vector: Add code to the *initialize.m* file so that it declares a global variable named *v* and initialises it to:**

$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

2. **Scalar multiplication: Add a file containing a function called *scalarMultiplication(s, v)*. The function should take two arguments, a scalar value, s, and a vector, v. The function should return the result of the scalar multiplication of the scalar and the vector.**

3. **Column vector: : Add code to the *initialize.m* file so that it declares a global variable named *u* and initialises it to:**

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

4. **Dot product: Add a file containing a function called *dotProduct(v, w)*. The function should take two arguments, both vectors, and return the dot product (scalar product or inner product) of the two vectors.**

5. **Matrix: Add code to the *initialize.m* file so that it declares a global variable named *M* and initialises it to:**

$$\begin{bmatrix} 2 & 4 & 6 & 8 \\ 10 & 12 & 14 & 16 \\ 18 & 20 & 22 & 24 \end{bmatrix}$$

6. Matrix multiplication: Add a file containing a function called *matrixMultiplication(M, N)*. The function should take two arguments, both matrices, and return the product of the two matrices.
7. Row selection: In an RL scenario we are typically interested in finding the Q-values of a specific state (row). Add a file containing a function called *selectRow(M, r)*. The function should take two arguments, a matrix and a scalar value, and return the row that has the given scalar as its index (starting from 1).
8. Maximum value selection: Out of a row of Q-values, we typically want to identify the maximum value. Add a file containing a function called *selectMax(M)*. The function should take a row vector as an argument and return the maximum value of the vector.
9. Maximum value index: In RL we want our algorithms to learn by updating the maximum value, so we are more interested in its index than the value itself. Add a file containing a function called *indexOfMax(M)*. The function should take a row vector as an argument and return the index of the maximum value.
10. Maximum value update: So finally, given a matrix and a row index, we want to update the maximum value of that row, e.g., by multiplying it by 1.2. Add a file containing a function called *update(M,s)*. The function should take two arguments, a matrix and a row index. The function should return a copy of M where the maximum value in the given row has been multiplied by 1.2.

    Example: For the matrix M1 below, say we want to update the maximum value of row 2. The matrix M2 should be returned by the update function, where 2.4 = 2.0*1.2.

$$M1 = \begin{bmatrix} 0.3 & 2.2 \\ 2.0 & 1.6 \end{bmatrix}$$

$$M2 = \begin{bmatrix} 0.3 & 2.2 \\ 2.4 & 1.6 \end{bmatrix}$$