Dr. A. Cookson
2019/2020

Department of Mechanical Engineering
University of Bath

ME40064: System Modelling & Simulation
ME50344: Engineering Systems Simulation

Tutorial 1: Unit Testing & Data Structures
Example Solutions

In order to gain the maximum amount of insight from these example solutions, it is recommended that in addition to reading them, you also implement and run them within Matlab.

Part A: Writing Unit Tests

Presented here are two possible implementations of the function to compute:

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

Note that there are many ways to do this, but also note that the vector & matrix-based nature of Matlab means many operations can be performed quickly and efficiently by exploiting these features, as you will see by comparing the two solutions below.

Function

```
function [ sumNorm ] = SumSquareNormExample1( x, N )
%SumSquareNorm Calculates average of squared values
%   This function takes an input vector x and computes the sum of the
%   squared values, normalising by the number of values in the vector.
%   This implementation uses a for loop to update the sum with each value
%   in turn, before normalising using the value of N provided by the user
%   square and sum the values of x
   sumNorm=0;
   if(N > 0)
     if(length(x) == N)
       sum = 0.0;
       for i =1:N
         sum = sum + x(i)*x(i);
       end
       sumNorm = sum/N;
     else
       error('SumSquare:Nincorrect','N must be equal to the number of elements of x');

     end
   else
       error('N must be greater than 0');
   end

end
```

Unit tests

```
%% Test 1:six positive numbers [1 2 3 4 5 6]
% Test that the solution is 15.1667
tol = 1e-14;
sumvar = SumSquareNormExample1([1 2 3 4 5 6],6);
assert(abs(sumvar - 91/6) <= tol)

%% Test 2: a mixture of positive and negative numbers [-1 -2 -3 4 5 6]
% Test that the solution is also 15.1667 in this case
tol = 1e-14;
sumvar = SumSquareNormExample1([-1 -2 -3 4 5 6],6);
assert(abs(sumvar - 91/6) <= tol)
```

A more compact implementation that uses the vector dot product is as follows:

Function

```
function [ sumNorm ] = SumSquareNorm( x )
%SumSquareNorm Calculates average of squared values
%   This function takes an input vector x and computes the sum of the
%   squared values, normalising by the number of values in the vector.
%   This implementation uses the dot product function to simultaneously
%   square and sum the values of x

   sum = dot(x,x);
   sumNorm = sum/(length(x));

end
```

Unit test

```
%% Test 1:six positive numbers [1 2 3 4 5 6]
% Test that the solution is 91/6 (i.e. 15.16666666666667)
tol = 1e-14;
sumvar = SumSquareNorm([1 2 3 4 5 6]);
assert(abs(sumvar - 91/6) <= tol)

%% Test 2: a mixture of positive and negative numbers [-1 -2 -3 4 5 6]
% Test that the solution is also 91/6 (ie. 15.16666666666667) in this case
tol = 1e-14;
sumvar = SumSquareNorm([-1 -2 -3 4 5 6]);
assert(abs(sumvar - 91/6) <= tol)
```

Notice that with the second implementation there is no need to check whether `N` is the same as the number of entries in `x`, as it is determined directly from the array `x`, itself. Similarly, for the case where `N` could be equal to zero.

Part B: Matlab Data structures

1. There are several ways to do this. The first way is to do it variable by variable.

```
car(1).name = 'Ferrari';
car(1).bhp = 500;
car(1).colour = 'red';
car(1).weight = 1500;
car(1).fuel_type = 'petrol';
```

The second way is to use the struct command, and do this for each entry in the array, `car`:

```
car(2)=struct('name','BMW','bhp',300,'colour','grey','weight',1300,'fuel_type','petrol'
);
```

Similarly, fill out the variables `car(3)` and `car(4)`, with data of your own choosing.

2. To calculate the power-to-weight ratio of a particular car, define the following Matlab function:

```
function [pwr] = pwrCalc(bhp,weight)

  pwr = bhp/weight;

end
```

To use the function for the variable `car(1)` would be the following:

```
car(1).pwr = pwrCalc(car(1).bhp,car(1).weight)
```

Alternatively, the function can be written in the following way:

```
function vehicle = pwr(vehicle)
  vehicle.pwr = vehicle.bhp/vehicle.weight;
end
```

To use this version of the function for the variable `car(1)`:

```
pwr(car(1));
```

3. One possible way to rank the performance of all the cars stored in the car array is the following function:

```
function comparePerf(car)

  vec=[car.pwr];
  [B,I] = sort(vec,'descend');
  car(I).name

end
```

To run this function, type the following:

```
comparePerf(car)
```