ME40064: System Modelling & Simulation
ME50344: Engineering Systems Simulation
Lecture 2

Dr Andrew Cookson
University of Bath, 2019-20

LECTURE 2
Verification & Validation

- Understand difference between verification & validation
- Understand verification in software development
- Ability to create & manipulate Matlab data structures
- Knowledge of debugging in Matlab

IMPORTANCE OF V & V

The Risks Of Getting It Wrong

IMPORTANCE OF V & V
The Risks Of Getting It Wrong

- reputation - you/your company may struggle to win new work

IMPORTANCE OF V & V
The Risks Of Getting It Wrong

- reputation - you/your company may struggle to win new work
- financial - redoing work costs money

IMPORTANCE OF V & V
The Risks Of Getting It Wrong

- reputation - you/your company may struggle to win new work
- financial - redoing work costs money
- legal - you/your company could be sued

IMPORTANCE OF V & V
The Risks Of Getting It Wrong

- reputation - you/your company may struggle to win new work
- financial - redoing work costs money
- legal - you/your company could be sued
- safety - people may be injured or worse

VERIFICATION & VALIDATION
The Difference Between Them

Verification: making sure you solve the model right

VERIFICATION & VALIDATION
The Difference Between Them

Verification: making sure you solve the model right

- unit testing
- regression testing
- analytical test cases

VERIFICATION & VALIDATION
The Difference Between Them

Verification: making sure you solve the model right

- unit testing
- regression testing
- analytical test cases

Validation: making sure you solve the right model

VERIFICATION & VALIDATION
The Difference Between Them

Verification: making sure you solve the model right

- unit testing
- regression testing
- analytical test cases

Validation: making sure you solve the right model

- comparison with experimental data
- comparison with existing models

SOFTWARE VERIFICATION
Unit Tests

Write the test first

- define inputs
- define expected outputs for each group of inputs

SOFTWARE VERIFICATION
Unit Tests

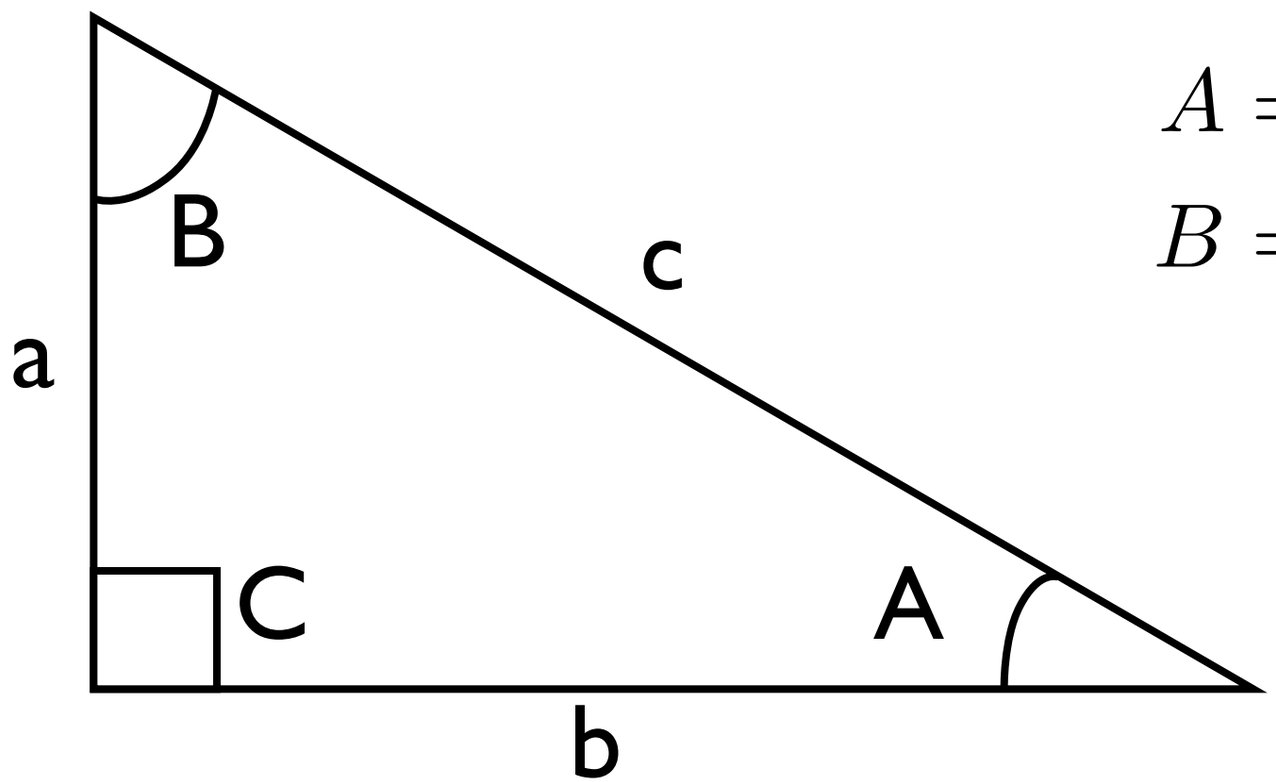
Write the test first

- define inputs
- define expected outputs for each group of inputs

Then write your function

- this is complete when all the test conditions are fulfilled

UNIT TESTS
Consider A Right Angled Triangle



$$A = atan(a/b)$$

$$B = atan(b/a)$$

$$C = 90^\circ$$

UNIT TESTS
An Example

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

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

end
```

UNIT TESTS
Functions - A Refresher

Function name

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

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

end
```

UNIT TESTS
Functions - A Refresher

Input arguments

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

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

end
```

UNIT TESTS
Functions - A Refresher

Function code

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

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

end
```

Output arguments

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

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

end
```

UNIT TESTS
Functions - A Refresher

Indicates function start/end

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

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

end
```

Remember - to define a function in Matlab this code needs to be saved in a file named *rightTri.m*

WRITING UNIT TESTS
An Example

```
%% 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)
```

RUNNING UNIT TESTS
An Example

Save the test function *rightTri* in a script file *rightTriTest.m*, then

Run the test with the following command:

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

Do not run the script rightTriTest.m directly - this won't provide the same diagnostic information!

SOFTWARE VERIFICATION
Analytic Solutions

Compare output of the software with an analytically derived solution of the model

- tests that software is working as a whole
- reveals limits of numerical method used to solve the model

We'll return to this in the assessments when you've written your models & code

MODELING IN COMPUTER CODE
Data Structures

- Another form of modelling
- Identify objects, their components, and their functions
- Represent this hierarchy in code

DATA STRUCTURES
The Motivation

- Allow storage of multiple related attributes/variables within a single variable
- Consider the variable ‘car’, this has, for example:
  - `colour`
  - `wheels`
  - `bhp`
  - `fuel type`

Note: Matlab is a dynamic, weakly typed language

LIFE WITHOUT DATA STRUCTURES
The Motivation

Multiple instances of the same variables

- `car1, car2, car3, ...`
- `wheels1, wheels2, wheels3, ...`

Passing into a function:

- `comparePerformance(car1, car2, car3, ..., bhp1, bhp2, bhp3, ...)`

This is very unwieldy

LIFE WITH DATA STRUCTURES
The Benefits

Define attributes of a car as members of the car variable - dot notation

- `car.colour = ‘red’`
- `car.wheels = ‘4’`
- `car.bhp = ‘120’`
- `car.fuel_type = ‘diesel’`

Define multiple cars using an array

- `car(1), car(2), car(3), ...`

LIFE WITH DATA STRUCTURES
The Benefits

Define multiple cars using an array

- `car(1), car(2), car(3), ...`

For example:

- `car(1).colour`
- `car(1).wheels`
- `car(2).colour`
- `car(2).wheels`

LIFE WITH DATA STRUCTURES
The Benefits

Other arrays & structures can be members of this structure:

- `car(1).wheels(1).position = 'front left'`
- `car(1).wheels(2).position = 'front right'`

LIFE WITH DATA STRUCTURES
The Benefits

To define several member variables at once, use `struct` command

- `car(1) = struct('bhp',130,'wheels',4);`
- `car(2) = struct('bhp',60,'wheels',3);`

UNIT TESTS REVISITED
The Data Structure

The results data structure:

```
>> results

results =

  1x4 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration

Totals:
   4 Passed, 0 Failed, 0 Incomplete.
   1.0045 seconds testing time.
```

UNIT TESTS REVISITED
The Data Structure

First results array entry:

```
>> results(1)

ans =

  TestResult with properties:

          Name: 'rightTriTest/Test1_SumOfAngles'
        Passed: 1
        Failed: 0
    Incomplete: 0
      Duration: 0.9782

Totals:
   1 Passed, 0 Failed, 0 Incomplete.
   0.97815 seconds testing time.
```

UNIT TESTS REVISITED
The Data Structure

First results array entry:

```
>> results(1)

ans =

  TestResult with properties:

          Name: 'rightTriTest/Test1_SumOfAngles'
        Passed: 1
        Failed: 0
    Incomplete: 0
      Duration: 0.9782

Totals:
   1 Passed, 0 Failed, 0 Incomplete.
   0.97815 seconds testing time.
```

Extract name of first test result:

```
>> results(1).Name

ans =

rightTriTest/Test1_SumOfAngles
```

WHAT IF THE TEST FAILS?
Debugging In Matlab

Matlab provides a powerful and easy to use debugging environment:

https://uk.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html

DEBUGGING IN MATLAB
The Key Steps

Set a breakpoint - when the code is run, it will pause here. Set it where you suspect the bug might be.

FILE NAVIGATE BREAKPOINTS RUN

```
1      % Calculate hypotenuse (c) of a right angled triangle with sides a, b
2 -    a = 3;
3 -    b = 4;
4
5 -    c = sqrt(a^2 + b^2);
```

Click here on the dash to set breakpoint

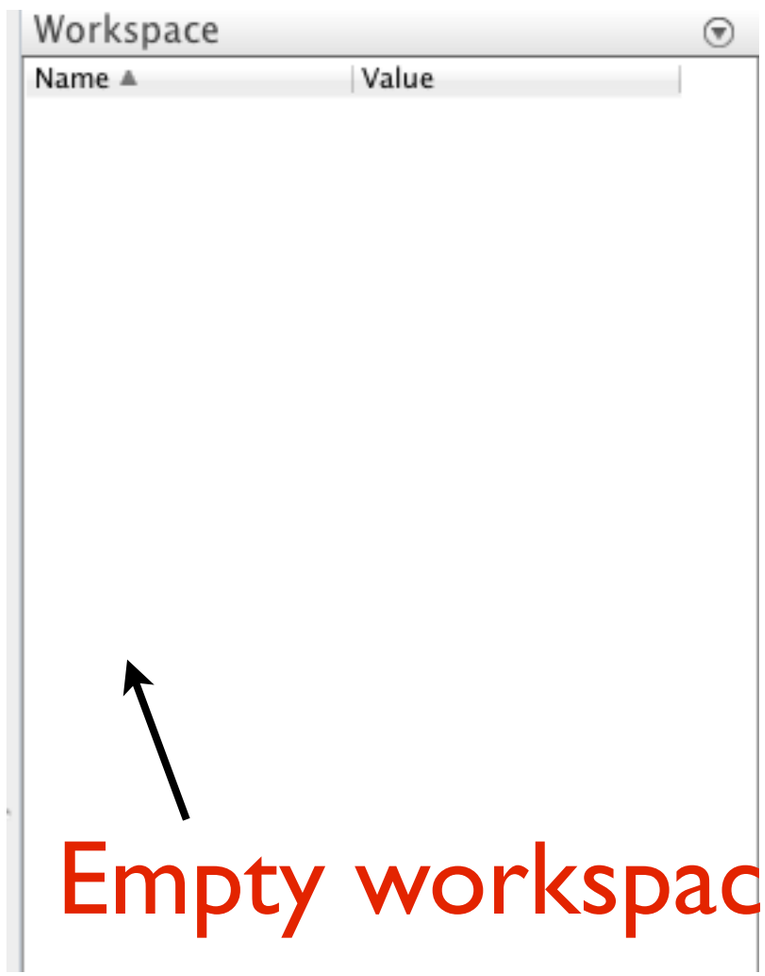
FILE NAVIGATE BREAKPOINTS RUN

```
1      % Calculate hypotenuse (c) of a right angled triangle with sides a, b
2 -    a = 3;
3 -    b = 4;
4
5 ●    c = sqrt(a^2 + b^2);
```

Red dot indicates position of breakpoint

DEBUGGING IN MATLAB
The Key Steps

Now run the code - it will execute until it reaches the breakpoint (assuming it doesn't produce an error message/crash before this point)



Empty workspace before running the code

DEBUGGING IN MATLAB
The Key Steps

Once the code has run, inspect the values of the variables to see if they are correct - this will help to locate the bug

BREAKPOINTS DEBUG

```
1     % Calculate hypotenuse (c) of a right angled triangle with sides a, b
2 -   a = 3;
3 -   b = 4;
4
5 ●➡  c = sqrt(a^2 + b^2);
```

Code runs and pauses before executing line 5

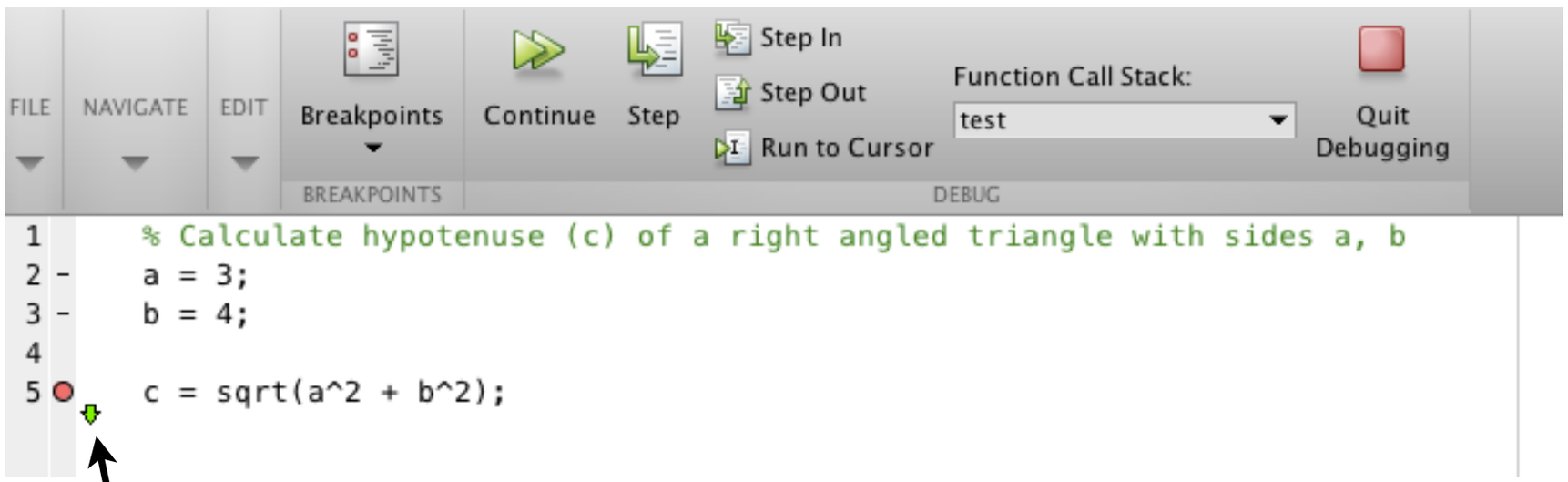
Workspace

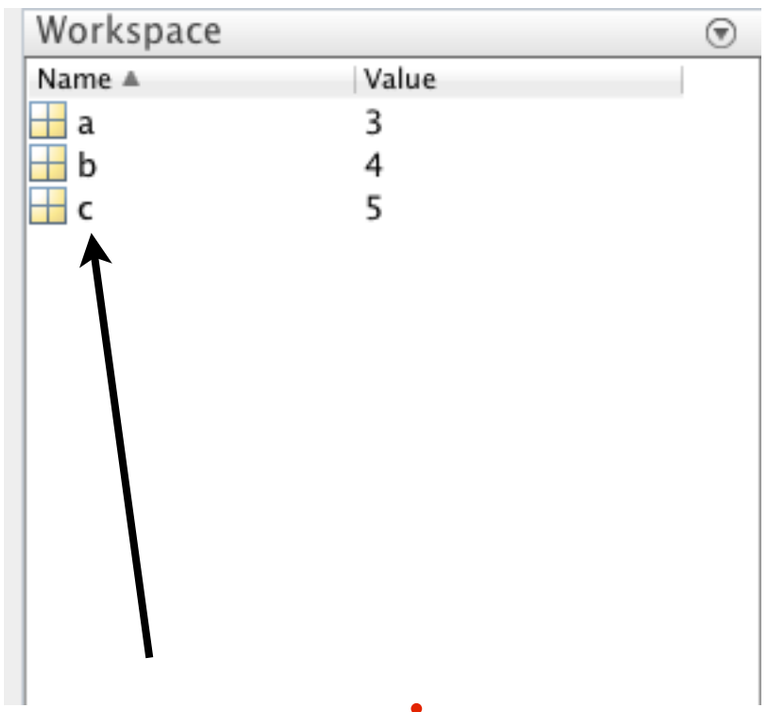
Name ▲	Value
a	3
b	4

Workspace now contains values for variables a and b

DEBUGGING IN MATLAB
The Key Steps

Either press continue to run to the end/next breakpoint or press step to execute the next line





Code continued to runs and executes line 5

Workspace now contains values for variables `a`, `b` and `c`. In this case, are all correct.

PUTTING THIS INTO PRACTICE

The Reality

The best way to understand unit tests, data structures, and debugging is to try them...