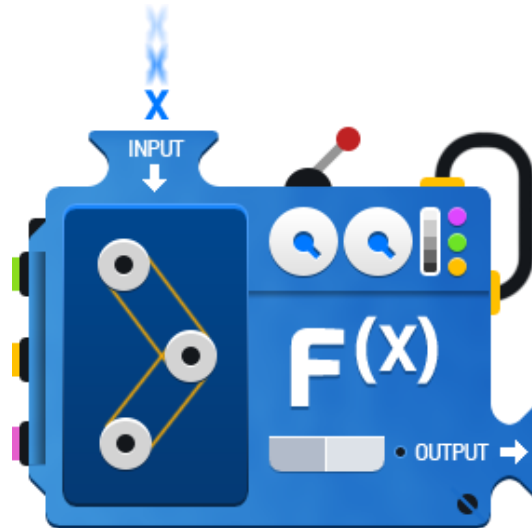


## MATLAB Lab 5 – User-defined Functions and Debugging

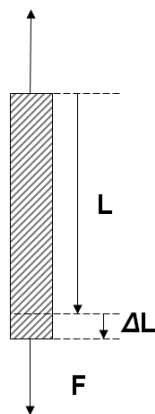


Download the resources from the MATLAB Labs section of the VLE, saving them to your working directory.

### Exercise 5.1 – Writing your own user-defined functions.

In this first exercise you'll create a simple user defined function that will then be called by an m-file just to get you used to the concept of function calls and input arguments.

1. Ensure the `determine_youngs.m` is in your working directory and remember to set the current directory in the MATLAB window to the same directory. Open `determine_youngs.m`. We want to determine the Young's Modulus of a particular material. The Young's Modulus gives us a measure of stiffness of an elastic material. It can be determined experimentally by placing a uniform material under compressive or tensile loading and recording the change in length of the material. The equation for doing so is displayed below.



**E** Young's Modulus (Pa)  
**F** Force (N)  
**A** Cross sectional area (m<sup>2</sup>)  
**ΔL** Increase in length (m)  
**L** Original Length of specimen (m)

$$E = \frac{F / A}{\Delta L / L}$$

- a. `determine_youngs.m` requests the user to input values, it will then use the function `young()` to determine the Young's Modulus. Create a new m-file and save it as `young.m`.
- b. Define the function in the first line: `function y = young(F,A,L,deltaL)` and add some comments explaining what the function will do. Note that the function name (in this case `young`, must match the filename, `young.m` otherwise MATLAB will not find it when called.)
- c. Implement the formula above into the `young.m` file and then save.
- d. Run `determine_youngs.m` by either typing `determine_youngs` in the Command Window or pressing the run button in the Editor window. Enter the following values describing a 1cm square bar that is 1m long with a tensile force of 20kN applied to it, increasing its length by 1mm:

Applied force (N) = 20000

Cross sectional area (m<sup>2</sup>) = 0.0001  
 Original length (m) = 1  
 Change in length (m) = 0.001

This will give you the Young's Modulus of the material in Pa or N/m<sup>2</sup>, a search for Young's Modulus on the internet should give you an idea of the material used.

Note that when calling a function, the input argument and output argument variable names do not need to match with those given in the function definition. The importance is in the order.

2. Ensure quadratic\_solver.m is in your working directory and remember to set the current directory in the MATLAB window to the same directory. The program will determine the type of roots produced by a quadratic equation of the form:

$$ax^2 + bx + c = 0$$

The roots of the above equation represent the values of  $x$  that will satisfy the above equation. It is possible for the roots of the equation to be real, repeated or complex. The roots can be calculated using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The result of discriminant,  $d = b^2 - 4ac$  informs us if the roots are real or complex:

if  $d < 0$  roots are complex

if  $d = 0$  roots are repeated

if  $d > 0$  roots are real and distinct

We want to create a function that determines the type of roots the equation might have.

- a. Open quadratic\_solver.m in Editor and have a look at the code. You'll notice it calls a function called quadratic. In this exercise you will create the function quadratic.m. Notice the function will have 3 input arguments for  $a$ ,  $b$  and  $c$  and will have 2 outputs,  $d$  is the result of the discriminant of the equation, i.e.  $b^2 - 4ac$  and  $i$  will inform us of the type of root:
  - $i = 0$  means roots are imaginary
  - $i = 1$  means roots are repeated
  - $i = 2$  means roots are real and distinct.
 Open a blank m-file and save it as quadratic.m
- b. Define the function in the first line of code, `function [d,i] = quadratic(a,b,c)` and add some comments that will describe what the function does, which is finding the type of roots the quadratic equation has.
- c. The function must then find the discriminant  $d$  using the above formula.
- d. Once  $d$  has been found, use an if/elseif/else structure to determine if  $i$  is equal to 0, 1 or 2. The function will return the two values,  $d$  and  $i$ .
- e. Save the function quadratic.m. Now run quadratic\_roots.m entering these different values for  $a$ ,  $b$  and  $c$ :
  - i.  $a = 1, b = 2, c = -3$
  - ii.  $a = 1, b = 4, c = 4$
  - iii.  $a = 2, b = 3, c = 5$

Depending on the values, you will get the answer for  $d$  and some text explaining the type of roots.

3. If you finish, look at adapting the quadratic.m function such that it also returns the calculated roots of the quadratic. To do this you can use the inbuilt roots function `rts = roots([a b c])` which returns the roots whether real or complex. To do this you'll need to add another output to the function and also adapt quadratic\_roots.m to allow for the extra output `rts` when calling the function. Display the roots at the end of quadratic\_roots.m by just typing `rts`.

## End of exercise 5.1

## Exercise 5.2 – Debugging an m-file.

In this exercise you will use the debugging tools in MATLAB to correct a simple guessing game similar to those covered in Lab 3. While there are obvious errors, there are also sections of code that will run fine without errors or warnings but will not give you the correct answer. To correct the code you'll have to step through the code using the debugging tools and look at the variable values as they change to determine the error.

1. Open `guessing_game.m` in the editor, correct errors in the program. Use the Error Bar/Message Indicator to pinpoint any errors and warnings. Hovering over the error will bring up an explanation of the error or warning. Note that some of the warnings will not stop the system from running such as recommendations for the use of semicolons to suppress the output.



- a. Correct any errors and warnings found in the Error Bar until the green box shows at the top of it. Once they have been removed, save and run the program. It is running correctly?


Note: a clean Error Bar doesn't not necessarily mean that a program will run, but it does mean that your syntax is correct.

- b. By looking in the Command Window at any errors occurring, you can pinpoint the line in which an error still presides. Unless you've already corrected it, you should see:



```
Undefined function or variable 'rend'.

Error in guessing_game_int (line 9)
    randnum = rend;
```

In this case it is the fact that `rend` is not a function or defined variable. Correct the function name to `rand` and save and run again to see the next cause of error.

- c. Once the code runs, attempt to see if it is behaving as expected. Note that to stop a program running in MATLAB you can use `ctrl+c`. You'll be aware that the program, while able to actually run, is not doing what it should. Using a breakpoint and then stepping through the code using the debugging tools will allow you to view how variables are changing as the code is running. Place a breakpoint at the start of the while loop by left clicking on the black line next to the line number or by clicking on the text on the line of code and pressing the set breakpoint button .

Note: A breakpoint will appear grey if there are still syntax errors in your code or if you have not saved your code. You will not be able to run until the breakpoint is red.

- d. Run the program, it will run until the breakpoint. Step through the code using  checking how the variables change and which if statements are skipped. Once you've found an error, Exit debug mode using  and then correct the code before running again.
  - e. Keep debugging the code until it runs correctly.

### End of exercise 5.2

## Sorting function challenge

You are going to create a user-defined function that will sort a list of scientists by their year of birth. A .csv file contains a list of famous scientists and their respective years of birth. The code to read the file is provided in `scientist_sorter.m`. Note the use of the `textscan` function, search documentation for `textscan` for how this function works.

From the .m file, a user-defined function is called that takes a cell array called `scientist_list` as an input argument. This contains the two further cell arrays, each containing a single column from the csv file. You can access the data in each using the braces `{}`. The UDF should return two vectors, one a cell array containing the sorted names and a second containing an array of double values of the sorted years of birth. Look at `scientist_sorter.m` for the format of the function call, then write the `sort_list.m` function. Hint: type `help sort` into the command window, note it can return 2 output arguments.

