L05: More on Functions

Sub- and nested Functions; Function Handles; Anonymous Functions

Lucas A. J. Bastien

E7 Spring 2017, University of California at Berkeley

January 27, 2017

Version: release

Announcements

Lab 02 was posted yesterday. It is **due on February 3**

- Step up in length and difficulty compared to Lab 01
- Start working on Lab 02 early!
- Come to lab section with questions!

Today:

- Sub-functions and nested functions
- Getting help in Matlab
- Function handles and anonymous functions

Next week:

- Branching (`if` statements)
- Introduction to matrix multiplication
  - As a linear algebra concept
  - In Matlab

Using your own functions inside functions

Your functions can call:

- Matlab built-in functions
- Your own user-defined functions

Work on examples in class (see diary):

- Calling Matlab built-in functions: `my_quadrilateral`
- Your own user-defined functions:
  - Defined as separate functions: `my_quadrilateral_separate` and `my_distance_separate`
  - Sub-functions: `my_quadrilateral_sub`
  - Nested functions: `my_quadrilateral_nested`

Using your own functions inside functions

Every user-defined function, whether main, sub-, or nested, uses the **same syntax for its header and must end with the keyword** `end`

Sub-functions:

- Defined below the main function in an m-file
- There can be more than one sub-function in an m-file
- Can only be called by functions defined in the same m-file*
- Does not have access to its caller's workspace

Nested functions

- Defined within a function in an m-file. The function within which it is defined is called the "parent" function.
- Has access to its parent function's workspace.
- Can only be called by its parent function*

*: There are workarounds to these limitations, but I will not talk about them

Nested functions

Nested functions have access to their parent function's workspace.
For example, consider the function:

```
function [y] = my_function(x)
a = 2;
  function [] = my_nested_function()
  y = x + a;
  end
my_nested_function()
end
```

Now, consider executing the following command:

```
>> my_function(5)
ans =
   7
```

In this example, if `my_nested_function` were defined as a sub-function instead of a nested function, executing `my_function(5)` would result in Matlab throwing the error: Undefined function or variable 'x'.

When to write functions?

Each function should perform a specific task.

If a part of your code is repeated, you should probably write this part of your code as its own function

- You implement each part of your code only once
- If you have to improve your code or fix mistakes, you have to do it in one location only

My advice for this semester: do not use nested functions, use sub-functions instead.

Matlab documentation and help

Take some time to browse the official Matlab documentation:

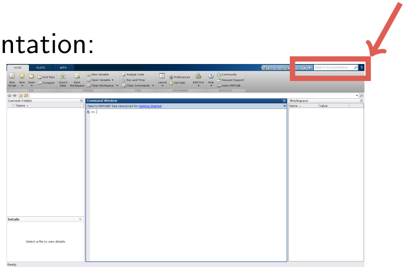
- https://www.mathworks.com/help/matlab
  - Not all sections are relevant to this class, but the first few are

List of Matlab built-in functions:

- https://www.mathworks.com/help/matlab/functionlist.html

From within Matlab:

- Search the documentation:



New to MATLAB? See resources for Getting Started.

```
>> 
```

- Use the `help` command

Function handles

A function handle is an association to a function that can be stored in variables. One can use `@` to obtain a function handle to an existing function given its name. One can store the function handle in a variable and then use that variable to call the function. For example:

```
>> handle_to_cos = @cos;
>> cos(pi/3)
    0.5000
>> handle_to_cos(pi/3)
    0.5000

>> % handle_to_cos is not a function. Rather, it is a
   % variable that contains a function handle
>> class(handle_to_cos)
    function_handle
>> another_handle = handle_to_cos; % No @ here
>> another_handle(pi/3)
    0.5000
```

Function handles can be passed as input arguments to other functions (see diary for examples)

Anonymous functions (1)

An anonymous function is a function that does not have a name, but that can be used with the help of a function handle associated with it. Anonymous functions are defined in-line following the syntax:

```
my_handle = @(input_1, input_2, ...) expression;
```

where `expression` must be a single statement. For example:

```
>> circle_area = @(r) pi * r.^2;

>> class(circle_area)

    function_handle

>> circle_area(5)

    78.5398
```

Anonymous functions (2)

One can use the values of existing variables (as is the case with `x0` and `y0` in the example below) when defining anonymous functions . Changing the values of these variables after the anonymous function has been defined has no effect on the anonymous function

```
>> x0 = 2;
>> y0 = 5;

>> % Function handle to calculate the distance from the
   % point of coordinates (x0,y0)
>> distance = @(x, y) sqrt((x-x0).^2 + (y-y0).^2);
>> distance(4, 5)
    2

>> x0 = 1000;
>> y0 = 2000;

>> distance(4, 5)
    2
```

I want your feedback!

On a piece of paper, write your feedback about E7 lectures so far:

- Things that are going well
- Aspects of the class that could be improved

Suggestions of topics to discuss:

- Are the lectures engaging?
- Am I well prepared for lectures?
- Too easy? Too difficult? Too fast? Too slow?
- Suggestions for improving lectures?
- Are my explanations clear?
- Do the examples that we discuss in class help you understand the course material?