ENME202 Matlab

CUSTOM FUNCTIONS

Syntax for a custom function with single input & output values:

```
function return_variable = function_name(arguments)
  % return_variable -->  local variable value returned by the function
  % function_name   -->  name of the function
  % arguments  -->  one of more values passed to the function
  %
  % Function code goes here...
  %
  % Note that code in the function block is indented
  %
end
```

The function can be saved as a separate m-file with the same name as the function_name itself. For example, the following function must be saved as "timestwo.m":

```
function z = timestwo(y)
  z = y*2;
end
```

Alternately, functions may be defined **at the end of an m-file**, with all other code located above the function definition(s). Such functions are considered "local", meaning that they can only be accessed by code within the same m-file where the function is defined.

Function overview:

A function is a construct that takes one or more inputs (arguments), and returns either nothing or a single value. Note that any type of value is allowed, including arrays containing multiple values internal to the array data structure.

Let's look at the above function declaration line in detail:

```
function z = timestwo(y)
```

This is the function declaration. This particular function takes a single argument, i.e. a value that is passed to the function and assigned to the variable name y within the function. A single value is returned by the function. This value is whatever is stored in the local variable name z (see below for discussion of local vs. global variables). Thus whatever value variable z holds at the end of the function block is the value that will be returned to the calling code.

Variable scope:

The workspace holds variables declared in our main code (as well as variables declared directly in the command window). Variables in the workspace are **global**, meaning that any normal code we run will have access to those variable names and their corresponding values.

Variables defined in a function (like y and z above) reside in their own separate workspaces. These variables are **local** to each function, i.e. the variable **scope** is limited to the function.

In fact, Matlab functions DO NOT HAVE ACCESS TO THE GLOBAL WORKSPACE AT ALL. Functions cannot normally access global variables, and the workspace cannot access variables defined inside a function. As a result, the names used for the input and output variables in a function are completely arbitrary. The workspace never needs to know the names of these variables. We could rewrite the above function code as below, and it will work identically to the initial version above:

```
function bar = timestwo(foo)
  bar = foo*2;
end
```

However, if we **explicitly declare a variable to be global**, functions can access the variable from the main workspace and visa vera. Variables can be declared to be global inside or outside of the function. Here is an example with a variable declared global in the main code:

```
global a   % declare a to be global
a = 5;

function x = fn1(z)
  global a      % access global a
  x = 2*z*a;
  a = 10;       % modifying a will change its value globally
end

function y = fn2(z)
  global a      % access global a
  y = z*a/2;
end
```

Below are some additional examples of functions that implement the quadratic formula. To use the functions, either save each code block to a separate m-file with a name matching the function, i.e. quadform1.m, quadform2.m, quadform3.m etc, or add the blocks to the end of a normal m-file containing code that calls the functions.

quadform1.m :

```
function r = quadform1(a,b,c)
  % Re-implementation of quadform.m as a function.
  % Three coefficients of quadratic polynomial as inputs (a,b,c).
  % One output (r), which will be a vertical array of the two roots.

  x1 = (-b+sqrt(b^2-4*a*c))/(2*a);
  x2 = (-b-sqrt(b^2-4*a*c))/(2*a);

  r = [x1; x2];

end
```

quadform2.m :

```
function r = quadform2(p)
```

```
  % Another implementation, here designed to take only
  % one input (a 3-element array) to more closely match
  % the operation of the built-in "roots" function

  a = p(1);
  b = p(2);
  c = p(3);

  x1 = (-b+sqrt(b^2-4*a*c))/(2*a);
  x2 = (-b-sqrt(b^2-4*a*c))/(2*a);

  r = [x1; x2];

end
```

quadform3.m :

```
function [x1,x2] = quadform3(a,b,c)
  % Yet another implementation designed to output
  % the two roots separately, instead of together
  % in a single array as in the previous two examples
  %
  % This function has two separate outputs, similar to
  % the max/min commands we have used.  The first will be
  % x1, the second will be x2

  x1 = (-b+sqrt(b^2-4*a*c))/(2*a);
  x2 = (-b-sqrt(b^2-4*a*c))/(2*a);

end
```

Function handles:

At the start of this topic, we said that functions must either be defined in a separate m-file or placed at the end of the m-file where the function is called. This is not quite true. Matlab supports the concept of "function handles", allowing simple in-line functions to be defined within any m-file (this is similar to the concept of "lambda functions" in Python, for those familiar with that language).

Here is an example:

```
f = @(x,y) (x.^2 - y.^2);
```

The @(x,y) syntax indicates the local variables defined within the function to hold values passed to the function (two in the above case, but there could be none, or many). The expression in parentheses at the end of the line defines the value that will be returned by the function. So:

```
f(-1,9)          % returns (-1)^2 - (9)^2 = -80

f([1 2],[3 4])   % returns [1 2].^2 - [3 4].^2 = [1 4] - [9 16] = [-8 -12]
```

You are NOT responsible for knowing how to use function handles in ENME202.