ENGR 121: Computation Lab I

Handout for Lab 4: User-Defined Functions

Practice Exercises

1. The following is a function called *calcarea* that calculates and returns the area of a circle. Write the function and store it in a file called *calcarea.m*.

```
function area = calcarea(radius)
% This function calculates the area of a circle. It takes one parameter as
% input: the radius of the circle and provides one return value as the
% output: area.

if(radius < 0)
    fprintf('The radius provided to the function is negative. Quitting. \n');
else
    area = pi*radius*radius;
end

end
```

Technically, the above function is called using the name of the file in which the function resides. To avoid confusion, it is best to give the function the same name as the filename. In the above example, the function name is *calcarea* and the name of the file is *calcarea.m*. The result returned from this function can also be stored in a variable using an assignment statement; the name could be the same as the name of the output argument in the function itself, but that is not necessary. For example both of these assignments would be fine:

```
>> area = calcarea(5)
area =
   78.5398
>> myarea = calcarea(6)
myarea =
  113.0973
```

The output can be suppressed using the semicolon operator:

```
>> myarea = calcarea(6);
```

The value returned from the calcarea function cold be printed out to the screen using **disp** or **fprintf**.

```
>> disp(myarea)
  113.0973
>> fprintf('The area of the circle is %.2f \n', myarea)
The area of the circle is 113.10
```

Answer the following questions:

- The function *calcarea* has been written assuming that the argument to the function is a scalar. So, calling the function with a vector argument will result in an error message. For example:

  ```
  >> calcarea(1:5)
  Error using  *
  Inner matrix dimensions must agree.

  Error in calcarea (line 9)
  ```

```
    area = pi*radius*radius;
```

This is because the * was used for multiplication in the function, but .* must be used when multiplying vectors term by term. Rewrite *calcarea* to allow both scalars or vectors to be passed to this function.

- Write a script called *circleCallFn.m* that prompts the user for the radius and calculates the area of the circle by calling the *calcarea* function, and displays the output to the screen using **fprintf**. Running the script should produce the following:

```
>> circleCallFn
Please enter the radius: 5
For a circle with a radius of 5.00, the area is 78.54
```

2. In many cases it is necessary to pass more than one argument to a function. For example the volume of a cone can be calculated using the formula:

$$V = \frac{1}{3}\pi r^2 h$$

where r is the radius of the circular base and h is the height of the cone. Therefore, a function that calculates the volume of a cone needs two input arguments: r and h.

```
function volume = conevol(radius, height)
% conevol caluclates the volume of a cone.
% Format of the call: conevol(radius, height)
% Return value: volume of the cone

volume = 1/3 * pi * radius .^2 .* height; % the .^ and .* operators accommodate vector ...
    inputs as well

end
```

The first value passed to the function is stored in the first input argument (in this case, *radius*) and the second value passed to the function is stored in the second argument (in this case, *height*). So the arguments in the function call must correspond one-to-one with the input arguments in the function header. The result returned by the function is stored in the default variable *ans*. The function can be called in the following ways:

```
>> conevol(4, 6.1)
ans =
  102.2065
>> conevol(1:4, 6.1)
ans =
    6.3879   25.5516   57.4911  102.2065
```

Now, write a script that will prompt the user for the radius and the height, call the function *conevol* to calculate the volume, and print the result in a nice sentence format. So, your MATLAB program will consist of the script and the *conevol* function that it calls.

3. Write a *fives* function that will receive two arguments for the number of rows and columns, and will return a matrix with that size of all fives. Given some user-specified input argument to the function, the output is as follows:

```
fives(2, 2)
ans =
     5     5
     5     5
>> fives(2, 5)
ans =
     5     5     5     5     5
     5     5     5     5     5
```

4. Write a function *isdivby4* that will receive a vector of integer values as the input argument, and will return those elements of the vector that are divisible by four. Given a user-specified vector as input argument, the output is as follows:

```
>> isdivby4(1:25)
ans =
     4     8    12    16    20    24
```

5. A Pythagorean triple is a set of positive integers (a, b, c) such that $a^2 + b^2 = c^2$. Write a function $ispythag$ that will receive three positive integers (a, b, c in that order) and will return logical 1 for true if they form a Pythagorean triple, or 0 for false if not.

6. Many mathematical models in engineering use the exponential function. The general form of the exponential decay function is:

$$y(t) = Ae^{-\tau t}$$

where A is the initial value at time $t = 0$ and τ is the time constant for the function. Write a script to study the effect of the time constant. To simplify the equation, set $A = 1$. Prompt the user for two different values for the time constant, and for beginning and ending values for the range of a t vector. Then, calculate two different y vectors using the above equation and the two time constants, and graph both exponential functions on the same graph within the range the user specified. Use a function to calculate y. Make one plot red and the other blue. Label the graph and both axes. What happens to the decay rate as the time constant gets larger?

7. Variables declared and used within a function have local scope. In other words, functions have their own workspaces. So local variables in functions, input arguments, and output arguments only exist while the function is executing. They cease to exist once the function finishes.

```matlab
1  %% This script implements a simple example to explain the concept of scope and local ...
       variables within a function
2  % Author: Naga Kandasamy
3  % Date: 10/18/2014
4
5  clc;
6  clear all;
7
8  % Variables a and b are stored in the global or base workspace
9  a = 4;
10 b = 10;
11
12 global x; % The variable is declared as having global scope. So, it is accessible from ...
       within all functions.
13 x = 0;
14
15 someFn(); % Function call from the script file. We will declare local variables a and b ...
       within the function
16
17 fprintf('Main script: value of a = %d; value of b = %d. \n', a, b);
18 fprintf('Main script: value of the global variable x = %d. \n', x);
```

```matlab
1  function someFn()
2
3  % Variables declared within a function are local in scope. In other words,
4  % they are not part of the global workspace. These local variables are
5  % removed once the function completes.
6
7  % The variables a and b are local to this function and have no relation to
8  % the a and b variables that were declared previously within the script that
9  % calls this function.
10
11 a = 100;
12 b = 15;
13
14 global x; % Refers to the SAME variable x that was declared previously in the script. ...
       Variable names must match,
15 x = 5;
16
17 fprintf('Function: value of a = %d; value of b = %d. \n', a, b);
18 fprintf('Function: value of global variable x = %d. \n \n', x);
19
20 end
```

In the above example, the variables a and b, declared in lines 9 and 10, respectively, of the script file belong to the base workspace. However, the variables a and b declared within *someFn* are local in scope to that function. If we wish to share variables between *someFn* and the script that calls the function, we must explicitly declare the variable to have global scope using the *global* keyword. Here, x refers to the same variable both in the script and in the function.