Saghar Gorjiduz | 95243096
Tara Barghian | 97243009
Mohammad Hashemi | 97243073
Instructor: Dr. Yaser Shekofte
May 9, 2020

# Matlab #2 Report

## Matlab Programming Workshop | Spring 99

### 2-1 : Functions

```
function volume = cylinder(height, radius)
% function to compute the volume of a cylinder
% volume = cylinder(height, radius)
base = pi * radius^2;
volume = base * height;
```

This function has two argument : `height` & `radius`. It calculates the volume of and 3D-shape and returns the result within `volume`.

```
function [area, volume] = cylinder2(height, radius)
% function to compute the area and volume
% of a cylinder
% usage: [area, volume]=cylinder2(height, radius)
base = pi .* radius.^2;
volume = base .* height;
area = 2 * pi * radius .* height + 2 * base;
```

This function has two argument `height` & `radius` and two output `area` & `volume`. It calculates the area and volume of an 3D-shape.

Note: We can't name an function with a number as starting letter. Valid function names begin with an alphabet character and can contain letters, numbers, or underscores.

Here is a list of other ways we can define a function in MATLAB:

| Name | Syntax | Description |
|---|---|---|
| Anonymous Function | `F = @(arglist)`<br>`  expression` | An anonymous function is like an inline function in traditional programming languages, defined within a single MATLAB statement. It consists of a single MATLAB expression and any number of input and output arguments.<br><br>You can define an anonymous function right at the MATLAB command line or within a function or script. This way you can create simple functions without having to create a file for them. |
| Primary & Sub-Functions | - | Any function other than an anonymous function must be defined within a file. Each function file contains a required primary function that appears first and any number of optional sub-functions that comes after the primary function and used by it.<br><br>Primary functions can be called from outside of the file that defines them, either from command line or from other functions, but sub-functions cannot be called from command line or other functions, outside the function file.<br><br>Sub-functions are visible only to the primary function and other sub-functions within the function file that defines them. |
| Nested Functions | `Function x = A(p1, p2)`<br>`...`<br>`B(p2)`<br>`  Function y = B(p3)`<br>`  ...`<br>`  End`<br>`...`<br>`End` | You can define functions within the body of another function. These are called nested functions. A nested function contains any or all of the components of any other function.<br><br>Nested functions are defined within the scope of another function and they share access to the containing function's workspace. |

In the next page you can see some examples of these types of functions:

```m
clc; clear;

%Example of Anonymous Function

power = @(x, n) x .^ n;
result1 = power(7, 3)
result2 = power(49, 0.5)
```

COMMAND WINDOW

```
result1 =

   343


result2 =

     7
```

```m
%Example of Nested Function

function [ans1, ans2] = quadraticEquation(a, b, c)
    function [delta] = delta(a, b, c)
        delta = b ^ 2 - 4 * a * c;
    end
    if delta(a, b, c) < 0
       disp('NO ANSWER!');
    else
        ans1 = ((-1) * b + sqrt(delta(a, b, c))) / (2 * a);
        ans2 = ((-1) * b - sqrt(delta(a, b, c))) / (2 * a);
    end
end
```

COMMAND WINDOW

```
>> quadraticEquation(8, 2, 3)
NO ANSWER!
>> [root1, root2] = quadraticEquation(1, 8, 3)

root1 =

   -0.3944


root2 =

   -7.6056
```

```m
%this is primary function

function [ans1, ans2] = quadraticEquation(a, b, c)
    delta = calculateDelta(a, b, c);    %uses sub-function defined below
    if delta < 0
       disp('NO ANSWER!');
    else
        ans1 = ((-1) * b + sqrt(delta)) / (2 * a);
        ans2 = ((-1) * b - sqrt(delta)) / (2 * a);
    end
end

%this is sub-function:
function delta = calculateDelta(a, b, c)
    delta = b ^ 2 - 4 * a * c;
end
```

COMMAND WINDOW

```
NO ANSWER!
>> [root1, root2] = quadraticEquation(1, 10, 5)

root1 =

   -0.5279


root2 =

   -9.4721
```

## 2-2 : mean3

```
1   function [arithmetic, geometric, harmonic] = mean3(a, b)
2   arithmetic = (a + b) / 2;
3   geometric = sqrt(a * b);
4   harmonic = ((a .^ (-1) + b .^ (-1) ) / 2) .^ (-1);
5   end
```

COMMAND WINDOW

```
>> [a, g, h] = mean3(2, 5)

a =

    3.5000


g =

    3.1623


h =

    2.8571
```

## 2-3 : A program to use mean3

```
1 -  clear; clc; close all
2
3 -  x = input("Enter 2 numbers in the form of a vector \n [a,b]:  ");
4
5 -  if(length(x) ~= 2 )%check problem condition
6 -      disp("invalid number of args");
7
8 -  else
9 -  y = input("which kind of mean do you want? \n arithmetic 0; \n geometric  1; \n harmonic   2: ");
10 -     [a, g, h] = mean3(x(1), x(2));
11
12       %print required mean
13 -     if(y == 0)
14 -         disp(a);
15 -     elseif (y == 1)
16 -         disp(g);
17 -     elseif(y == 2)
18 -         disp(h);
19 -     else
20 -         disp("invalid input")
21 -     end
22 - end
```

COMMAND WINDOW

```
Enter 2 numbers in the form of a vector
 [a,b]:
[2, 5]
which kind of mean do you want?
 arithmetic 0;
 geometric  1;
 harmonic   2:
2
    2.8571
```

## 2-6: nargin & nargout

| Name | Syntax | Description |
|---|---|---|
| nargin | `Args = nargin` | Returns the number of function input arguments given in the call to the currently executing function. Use this syntax in the body of a function only. |
| | `Args = nargin(funct)` | Returns the number of input arguments that appear in the fun function definition. We use it outside of function's body. The `funct` is a string variable that stores the name of purpose function. |

```
1  function res = myGeometryCalculator(length, area)
2      if nargin == 1
3          res = length * length;  %to calculate the area
4      end
5      if nargin == 2
6          res = length * area;    %to calculate the volume
7      end
8      if nargin == 0
9          error('enter at least one argument :/')   %error message shown
10     end
11 end
```

COMMAND WINDOW

```
>> myGeometryCalculator(2)

ans =

     4

>> myGeometryCalculator(2, 5)

ans =

    10

>> myGeometryCalculator()
Error using myGeometryCalculator (line 9)
enter at least one argument :/
```

```
function res = myGeometryCalculator(length, area)
    if nargin == 1
        res = length * length;  %to calculate the area
    end
    if nargin == 2
        res = length * area;    %to calculate the volume
    end
    if nargin == 0
        error('enter at least one argument :/')   %error message shown
    end
end
```

COMMAND WINDOW

```
>> funct = 'myGeometryCalculator';
>> nargin(funct)

ans =

     2
```

| Name | Syntax | Description |
|---|---|---|
| Nargout | `Args = nargout` | Returns the number of function output arguments given in the call to the currently executing function. Use this syntax in the body of a function only. |
| | `Args = nargout(funct)` | Returns the number of output arguments that appear in the fun function definition. We use it outside of function's body. The `funct` is a string variable that stores the name of purpose function. |

```
function [add, subtract] = simpleCalculator(a, b)
    if (nargout > 1)
       add = a + b;
       subtract = a - b;
    end
end
```

COMMAND WINDOW

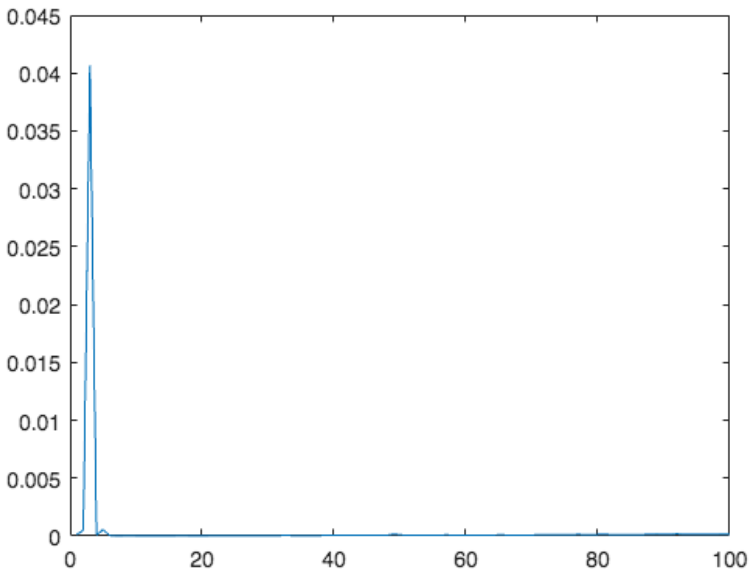
```
>> [add, subtract] = simpleCalculator(2, 5)

add =

     7


subtract =

    -3
```

## 2-8: tic toc

| Name | Syntax | Description |
|---|---|---|
| tic & toc | `tic`<br>`...`<br>`toc` | Actually `tic` starts a stopwatch timer and `toc` returns the elapsed time since `tic` was used.<br>`tic` Works with the `toc` function to measure elapsed time. The `tic` function records the current time, and the `toc` function uses the recorded value to calculate the elapsed time.<br><br>`timerVal = tic` stores the current time in `timerVal` so that you can pass it explicitly to the `toc` function. Passing this value is useful when there are multiple calls to `tic` to time different parts of the same code. `timerVal` is an integer that has meaning only for the `toc` function. |
| | `timerVal = tic`<br>`...`<br>`toc(timerVal)` | |
| | `tic`<br>`...`<br>`elapsedTime = toc` | |
| | `timerVal = tic`<br>`...`<br>`elapsedTime = toc(timerVal)` | |

Note: `tic` & `toc` functions both have one output argument. `tic` has always has one input argument but `toc` sometimes has one and sometimes has two input arguments.

```
1 -  clc; clear;
2    %This example measures how the time required
3    % to solve a linear system varies with
4    % the order of a matrix.
5
6 -  for n = 1:100
7 -      A = rand(n,n);
8 -      b = rand(n,1);
9 -      tic
10 -     x = A\b;
11 -     t(n) = toc;
12 - end
13 - plot(t)
```



## 2-9: varargin & varargout

| Name | Syntax | Description |
|---|---|---|
| `varargin` | `varargin` | It is an input variable in a function definition statement that enables the function to accept any number of input arguments. |

```
1       %Example for varargin
2
3       function strangeFunciton(varargin)
4           disp('Number of input arguments is :\n')
5           nargin
6           celldisp(varargin);
7       end
```

COMMAND WINDOW

```
>> strangeFunction('hi matlab', 2, pi)
Number of input arguments is :\n

ans =

     3


varargin{1} =

hi matlab


varargin{2} =

     2


varargin{3} =

    3.1416
```

| Name | Syntax | Description |
|---|---|---|
| `varargout` | `varargout` | is an output variable in a function definition statement that enables the function to return any number of output arguments. When the function executes, `varargout` is a 1-by-*N* cell array, where *N* is the number of outputs requested after the explicitly declared outputs. |

```
1      %Example for varargout
2
3      function  varargout = strangeFunciton(varargin)
4          disp('Number of output arguments is :\n')
5          n = nargout
6
7          for i = 1:n
8              varargout{i} = i;
9          end
10     end
11
```

COMMAND WINDOW

```
>> [a, b, c] = strangeFunction('hey', 2, pi)
Number of output arguments is :\n

n =

     3


a =

     1


b =

     2


c =

     3
```