Chapter 6

Function file in MATLAB

1. Introduction

The functions are an essential part of MATLAB programming, which can be defined as a block containing a set of instructions for generating some computations. In addition, the functions in MATLAB provide a well organization for code and ensure its reusability, and maintainability. In MATLAB, to execute a function it is necessary to store it in a separate .m file where **the file name should be the same name of the function**.

2. Main parts of a function

The general syntax of a function in MATLAB is expressed as follows:

```
function output = functionName(input)

      instructions

end
```

Functions can support several input arguments and may generate more than one output argument.

```
function [output1, output2, ...] = functionName(input1, input2, ...)

      instructions

end
```

The detailed form of a function can be given as follows:

```
function [output1, output2, ...] = functionName(input1, input2, ...)

   % Help text: Comments introduced to denote a description of the
   % function instructions and to be displayed with help.
   ...
   Output1 = ...
   Output2 = ...
   ...
end
```

Based on the presented syntax, the main elements of a MATLAB function are:

- **Function declaration:** The declaration of a function is presented by two parts separated by the equal sign. The first or the left part began with the keyword **function** followed by the output arguments. The second or the right part starts with the function name followed by the input arguments.
- **The function body:** This part is located between the function declaration part and the end of the function. It contains the set of computation instructions including different MATLAB commands.
- **End of the function**: Any function ends with the keyword **end**.

The first comments that appear between the function declaration and the first executable or blank line are reserved to denote a description of the function task (Function help). When we type **`help functionName`** MATLAB command displays the function help.

Local variables: Any variables declared inside the function are called local variables which are not accessible outside it where the function has its own workspace. So, a variable named **`'var'`** inside a function will not conflict with a variable named **`'var'`** outside the function.

Global variables: By default, the variables defined in the base workspace are not available inside the function. To make any variable accessible inside any function we declare it using the keyword '`global`' in the command window and in the function. If a variable is declared as global in several functions and in the base workspace, then any assignment to that variable, inside any function, is available to all functions declaring it global.

Some functions are useful when using global variables:

`isglobal(var)`: introduced to check if the variable is global or not, which returns 1 if '`var`' is global, and 0 otherwise.

`who global`: gives a list of global variables.

`clear global`: makes all variables nonglobal.

`clear VAR`: makes VAR nonglobal if it is declared as global.

The following example presents a function that computes the area of a rectangle

rectangle_area.m

```
function area = rectangle_area(length, width)
      % This function computes the area of a rectangle
  if length < 0 || width < 0              % Check for valid input
       error('Both length and width must be positive!');
  else
        area = length * width;      % Calculate the area
  end
end
```

To use this function, it is necessary to save it in an .m file that takes the same name as the function name (`rectangle_area.m`).
After saving the function file. Now, we run it in the MATLAB Command Window or in a script to calculate the area of a rectangle with a value 10 for length and 5 for width.

```
>> myarea = rectangle_area(10, 5)

myarea =

    50
```

3. Anonymous function

The anonymous function is the simple way to create a simple function without saving it in a script file. In general, it is used in case of simple calculation in one-line code. The syntax of the anonymous function is given by:

`fvar  = @(arguments) expression`

where **`fvar`** is the function handle variable name. The word **`arguments`** represents the arguments passed to the function and **`expression`** is the corresponding code of the function. An example of an anonymous function is given in this example.

```
>> F = @(x) 2*x^2 +3*x -2;

>> F (3)

ans =

    25
```

In the above example, we defined an anonymous function with an input x where the body or the instruction of the function is **`2*x^2 +3*x -2`**. **`F`** is called the function handle and is used to call the function as indicated in the above command window: **`F(3)`**.

The function handle can be also used for referring to other defined user or standard MATLAB functions. As an example, we can use a function handle to built-in the key MATLAB function `sqrt`.

```
>> Fs = @sqrt

Fs =

function_handle with value:

    @sqrt

>> Fs(4)

ans =

     2
```

4. Practical examples

In this section we elucidate some MATLAB functions that tackle different problems.

Example 1: BMI calculator

This function computes the Body Mass Index (BMI) helping to assess health risk. The weight and the height are given in kilograms and meters respectively.

$$BMI = \frac{weight}{height^2} \quad \text{(Kg/m}^2\text{)}$$

Normal : 18.5 – 24.9 **Overweight : 25.0 – 29.9** **Obese : 30.0 – 34.9**

BMIcalculator.m

```
function bmi = BMIcalculator (weight, height)
      % This function computes the Body Mass Index (BMI).
  bmi = weight / height^2;
end
```

Example 2: Population growth

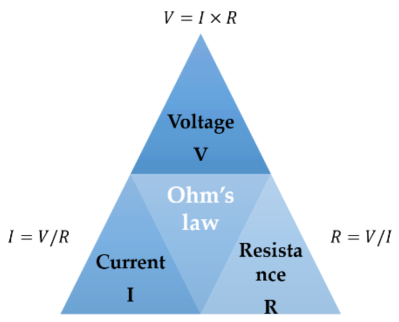
This function estimates the population growth using an exponential model given by $population = initialpop \times e^{(growthrate \times time)}$.

popGrowth.m

```
function pop = popGrowth (initialPop, growthRate, time)
      % This function computes the population growth.
  pop = initialPop * exp(growthRate * time);
end
```

Example 3: Ohm's law calculation in electrical circuit

This function calculates the current, resistance or the voltage in an electrical circuit using Ohm's law. It calculates the non-given value if the two other values are given.



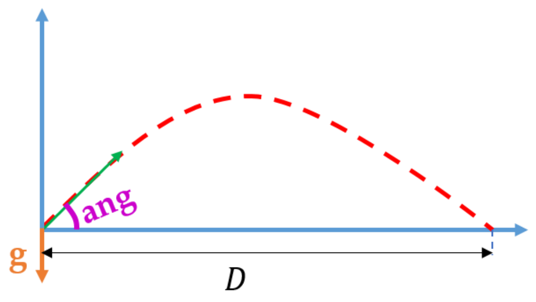
ohmsLaw.m

```
function result = ohmsLaw (current, resistance, voltage)
      % This function computes the current, resistance or the voltage in
      % an electrical circuit.
      % Provide only two values and leave one empty!
  if isempty(current)
      result = voltage / resistance;
  elseif isempty(resistance)
      result = voltage / current;
  elseif isempty(voltage)
      result = current * resistance;
  else
      error('Provide only two values and leave one empty []');
  end
end
```

Example 4: Projectile motion calculation

In this example we write a function to calculate the horizontal distance traveled by a projectile given its initial speed and angle. Gravity estimated to be 9.81.

$$D = (speed^2 \times \sin(2 \times angle))/gravity$$



projectileD.m

```
function distance = projectileD (speed, ang)
% This function computes the horizontal distance traveled by a projectile
  gravity = 9.81;
  angRad = deg2rad(angle);
  distance = (speed^2 * sin(2*anglRad))/gravity;
end
```

Example 5: Factorial of a number

In this example we introduce a recursive function that calculates the factorial of a number.

facto.m

```
function f = facto(n)
      % This function computes the factorial of a number.
  if n<=1
       f = 1;
  else
        f = n * facto(n-1);
  end
end
```

Example 6: Guessing number Game

This function represents a simple game to let the user guess a randomly generated number.

guessingNumber.m

```
function guessingNumber()
% A simple game to guess a randomly generated number between 1 and 100.
number = randi(100);
guess = -1;
attempt = 1;
 while guess ~= number
    guess = input('Enter your guess (1-100): ');
    attempt = attempt +1;
    if guess < number
        disp('Too low!');
    elseif guess > number
        disp('Too heigh!');
    else
    fprintf('Congratulations! you have guessed the number after %d attempts\n', attempt);
    end
 end
end
```

5. Practical work 6

1. Write a function that determines the area of a circle.
2. Write a function that calculates the area and the volume of a cylinder.
3. Write a function to find the roots of a quadratic equation $ax^2 + bx + c = 0$.

4. Write a function that returns the mean grade of five introduced points of a student with the corresponding observation and the appropriate decision (Pass or Fail). The observations are indicated below:
   15-20: “Very good”, 10-14: “Good”, 5-9: “Week”, 0-4: “Very Week”.

6. Exercises

1. Create a function *isEven(n)* that returns:

   1 if the number is even.

   0 if the number is odd.

2. Write a function converting Celsius to Fahrenheit.

   $$F = C.\frac{9}{5} + 32$$

3. Write a function *[p, n] = PosNegCount(V)* where:

   P = number of positive values, n = number of negative values.

4. Write a recursive function that returns a factorial of a number.

5. Write a function that returns the nth Fibonacci number without loops.