SVKM'S NMIMS Deemed to be UNIVERSITY | NAVI MUMBAI

MATLAB
Unit 5-Lecture 16

BTech (CSBS) -Semester VII

13 September 2022, 09:35AM

*few figures/content have been prepared/referred from internet source/books

Introduction to programming

1) Introduction,
2) M-File Scripts,
3) script side-effects,
4) M-File functions,
5) anatomy of a M- File function,
6) input and output arguments,
7) input to a script file,
8) output commands.

Introduction to programming

1) What is an m-file?

An m-file, or script file, is a simple text file where you can place MATLAB commands. When the file is run, MATLAB reads the commands and executes them exactly as it would if you had typed each command sequentially at the MATLAB prompt. All m-file names must end with the extension '.m' (e.g. test.m). If you create a new m-file with the same name as an existing m-file, MATLAB will choose the one which appears first in the path order (type help path in the command window for more information). To make life easier, choose a name for your m-file which doesn't already exist. To see if a filename.m already exists, type help filename at the MATLAB prompt.

2) Why use m-files?

For simple problems, entering your requests at the MATLAB prompt is fast and efficient. However, as the number of commands increases or trial and error is done by changing certain variables or values, typing the commands over and over at the MATLAB prompt becomes tedious. M-files will be helpful and almost necessary in these cases.

Introduction to programming

3) How to create, save or open an m-file?

If you are using PC or Mac:

- To create an m-file, choose New from the File menu and select Script. This procedure brings up a text editor window in which you can enter MATLAB commands.
- To save the m-file, simply go to the File menu and choose Save (remember to save it with the '.m' extension). To open an existing m-file, go to the File menu and choose Open.

If you are using Unix:

- To create an m-file, use your favorite text editor (pico, nedit, vi, emacs, etc.) to create a file with .m extension (e.g. filename.m).

4) How to run the m-file?

- After the m-file is saved with the name filename.m in the current MATLAB folder or directory, you can execute the commands in the m-file by simply typing filename at the MATLAB command window prompt.
- If you don't want to run the whole m-file, you can just copy the part of the m-file that you want to run and paste it at the MATLAB prompt.

Script M-file: Creating M-file

Quite often we need to be able to calculate the value of a function y=f(x) for any value of x. Obviously, it is not practical to change value of x each time. For this purpose MATLAB has a special type of M-file, called M-function.

> The following script M-file finds the value of the function at $y= \sin x + x^3$ at x = 3.

```
% exercise1script.m

x = 3

y = sin(x) + x^3
```

> Run this M-file by typing the following in the *Command Window*:

```
exercise1script
```

> Then update the M-file to find the value of f(x) for x = 4, 5, 6.

Properties of Function M-Files

A MATLAB file of a special format that contains code with optional inputs and outputs is called function M-file.

Some advantages:

- Functions can be called from inside of other script and function M-files.
- Inputs allow variable values to be modified when calling the function (eg from the Command Window).
- Outputs can be assigned to other variables at the Command Window or within a separate M-file.

Disadvantages:

- A slight disadvantage with a function M-file is that you must follow the prescribed format, or else it won't run correctly. Once you get the hang of that, you will see they are often very useful.

Properties of Function M-Files

> The following function M-file finds the value of the function $f(x) = \sin x + x^3$ for any value of x. Type it in and save as **exercise1func.m** in your *Current Directory*

```
% <insert your name and the date here>

% exercise1func.m

% input: x

% output: p, solved in terms of x

function p = exercise1func(x) %Note special format!

p = sin(x) + x^3
```

> Call this M-file from the *Command Window* using the following command and then update it to find the value of for p(x) for x=3, 4, 5, 6.

```
exercise1func(3) % returns value for x = 3
```

> Consider the case, when the variable x is an array [3 4 5 6]. Modify your function M-file so, that it will be able to work with arrays.

Constructing Function M-files

Function M-files allow you to define, construct and store your own functions.

First, let's recall some of the in-built MATLAB functions you have already used. If you type command help <name of the function> , such as help abs, in the Command Window, MATLAB will print description and correct syntax of this function. These functions have a few things in common and a few differences.

Function: y = abs(x)

Description: Assigns the value x to y if x is non-negative, or –x if x is negative.

Input: 1 number: x

Output: 1 number: either x or –x

Constructing Function M-files

Function: y=rem(a,b)

Description: Assigns the remainder of a/b to y

Input: 2 numbers: a is the numerator, b is the denominator

Output: 1 number: remainder of a/b

Function: plot(xArray,yArray)

Description: Plots a graph, given an array of x-coordinates, xArray, and an array of y-coordinates, yArray.

Input: 2 arrays of equal length: xArray, yArray.

Note: there can be many additional optional inputs.

Output: A graph.

Constructing Function M-files

Exercise: Determining the Inputs & Outputs

> For the following function (which you may recall from the first practical), use MATLAB to help you write the description, inputs and outputs.

Function: `round(a)`

Description: _____

Inputs: _____

Outputs: _____

Constructing Function M-files

> Exercise: Creating a Customised Random Number Generator

> Create a function M-file called `myRand.m` that outputs a random number between the inputted values of `minRand` and `maxRand` by adding code in the starred lines.

The MATLAB `rand` function returns a random value between 0 and 1. You will need to use this function as well as calculating the scale and offset values.

Example: If you want to find a number between 3 and 10, your scale is 7 and your offset is 3.

```
% <insert your name and the date here>

% myRand.m

% inputs: minRand, maxRand

% outputs: y, a random number with value between

% minRand & maxRand
```

Constructing Function M-files

```
function y = myRand(minRand, maxRand)

% **calculate the scale**

% **calculate the offset**

% **calculate y, using your scale, offset and %MATLAB's rand function**
```

Note: Save your function in the *Current Directory*, otherwise you will need to switch to the directory you saved your function in or type in a full path.

Constructing Function M-files

> Type in the following lines to call this function from your Command Window and verify that it works.

```
myRand(1,10)

myRand(100,100+1)

myRand(3,pi)

myRand(20)

myRand(20,1)
```

Steps to Create

Steps for creating function M-files

1. The function name and its M-file name must be identical (except that the M-file must end with .m).

2. The first executable statement must be a function declaration of the form

```
function <outputVariables> = <functionName>(<inputVariables>)
```

> One of the following function declarations has been taken from the MATLAB `rem.m` function M-file. Determine which one must be correct declaration:

```
function rem = remainder(x,y)

function out = rem(x,y)

out = function rem(x,y)

function out(x,y) = rem(x,y)
```

Excercise

Exercise: Writing Function Declarations

Write down the `<outputVariables>`, `<functionName>` and `<InputVariables>` for the two function M-files from the previous exercises and verify they match the function declaration statement.

Function M-file 1: $p(x) = \sin x + x^3$

Function Name: _____

Output Variables: _____

Input Variables: _____

Function M-file 2: Creating a Customised Random Number Generator

Function Name: _____

Output Variables: _____

Input Variables: _____

Exercise: Creating a Function M-file

Follow steps below to create a simple function M-file that computes and outputs the n^{th} power of 2, 2^n, where n is a number specified each time the function M-file is run.

Q1. Create a blank M-file and save it as twoN.m
Q2. What should go at the top of every M-file? Add in header comments. This time, make sure you include the function description, inputs and outputs as well as your name and the date.
Q3. Type the function declaration into your file called twoN.m
a) replace <function Name> with twoN
b) decide upon an appropriate input variable name. In this case you may call it simply n.
c) decide upon an output variable name. The name y will be used in this case.
(In other examples you could use fn, f_n or another name of your choice as an output variable name.)
Important: Every output variable must be assigned a value within your code.

Exercise: Creating a Function M-file

Now, you need to write your code. This function is pretty simple, so the code should only contain the following line: y=2^n

Note: if you have used different input/output variable names, you must change y to match your output variable name and n to match your input variable name.

Q5. OK, now you're ready to save and test your function M-file. After saving, make sure that your Current Directory matches the one you saved your M-file in.

Q6. Run the following lines in the Command Window to verify your function M-file works.

```
twoTo8 = twoN(8)
newNumber = twoN(5)
squareOfTwo = twoN(2)
twoN(9)
rootOfPower = twoN(5)^(1/2)
twoN %Why this does not work?
```

Exercise: Creating a Function M-file

Exercise: Writing your Own Function M-file

> Create a function M-file called `quadRoots` to find the roots of quadratic polynomials of the form $y=ax^2+bx+c$

Its inputs will be the coefficients a, b and c.

Its outputs will be the two roots, r_1 and r_2 and calculated by the formula:

$$r_1, r_2 = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

Test quadRoots in the Command Window with the following three polynomials:

$y = x^2 + 3x + 2$ (ans r_1=-1, r_2=-2)

$y = x^2 + 6x + 10$ (ans r_1=-3+i, r_2=-3-i)

$y = x^2 + 6x + 13$ (ans r_1=-3+2i, r_2=-3-2i)

Functions on Functions

Suppose a function y = demoFun(x) has been defined in a function M-file demoFun.m

`fplot(@demoFun,[a b])`	Plots the function for `a ≤ x ≤ b` without setting up arrays
`fzero(@demoFun,[a b])`	Finds one value of x for `demoFun(x) = 0` provided that the signs of `demoFun(a)` and `demoFun(b)` are opposite.
`fzero(@demoFun,c)`	Finds one value of x for `demoFun(x) = 0` by starting a search at `x = c`
`fminbnd(@demoFun,a,b)`	Finds the coordinates of a minimum point of `demoFun(x)` at the interval `a≤x≤b`
`quadl(@demoFun,a,b)`	Finds an accurate value for $\int_a^b y(x)\,dx$ Note: `quadl` uses arrays, so therefore you must set your function up treating x as an array and so using the dot notation for operations.

Exercise: Using Functions

1. Create a function M-file called `myCubic.m` whose output is the value of the function $y = x^3 + 2x^2 - 5x - 8$

    Input: `x`

    Output: `y`

> Verify that this function works by checking that `myCubic(-5)=-58` and `myCubic(5)=142`

2. Create a script M-file called cubicExercise.m that contains the following five cells with code that:

    a) plots `myCubic(x)` between the values of [-5, 5],

    b) finds a local minimum of `myCubic(x)`, located between O and 5,

    c) finds the all three roots of `myCubic(x)` using appropriate intervals

    `[a,b]`

    d) finds the value of the definite integral of `myCubic(x)` between -5 and 5.

Hint: You will need to use the array dot notation so you can use the `quadl` function to calculate the integral of y.

3. Make sure `cubicExercise.m` is marked up using cell formatting and publish it.