M-FILES

M-files

- MATLAB statements can be prepared with any text editor, and stored in a file for later use.
- MATLAB can then execute this sequence of statements. The file is referred to as an "***M-file***".
- Each M-file should have a name that ends in ***.m***. Much of your work with MATLAB will be in creating and refining M-files.
- It is much easier to use M-files than to enter commands line by line at the MATLAB prompt. Writing m-files will make you much more productive.

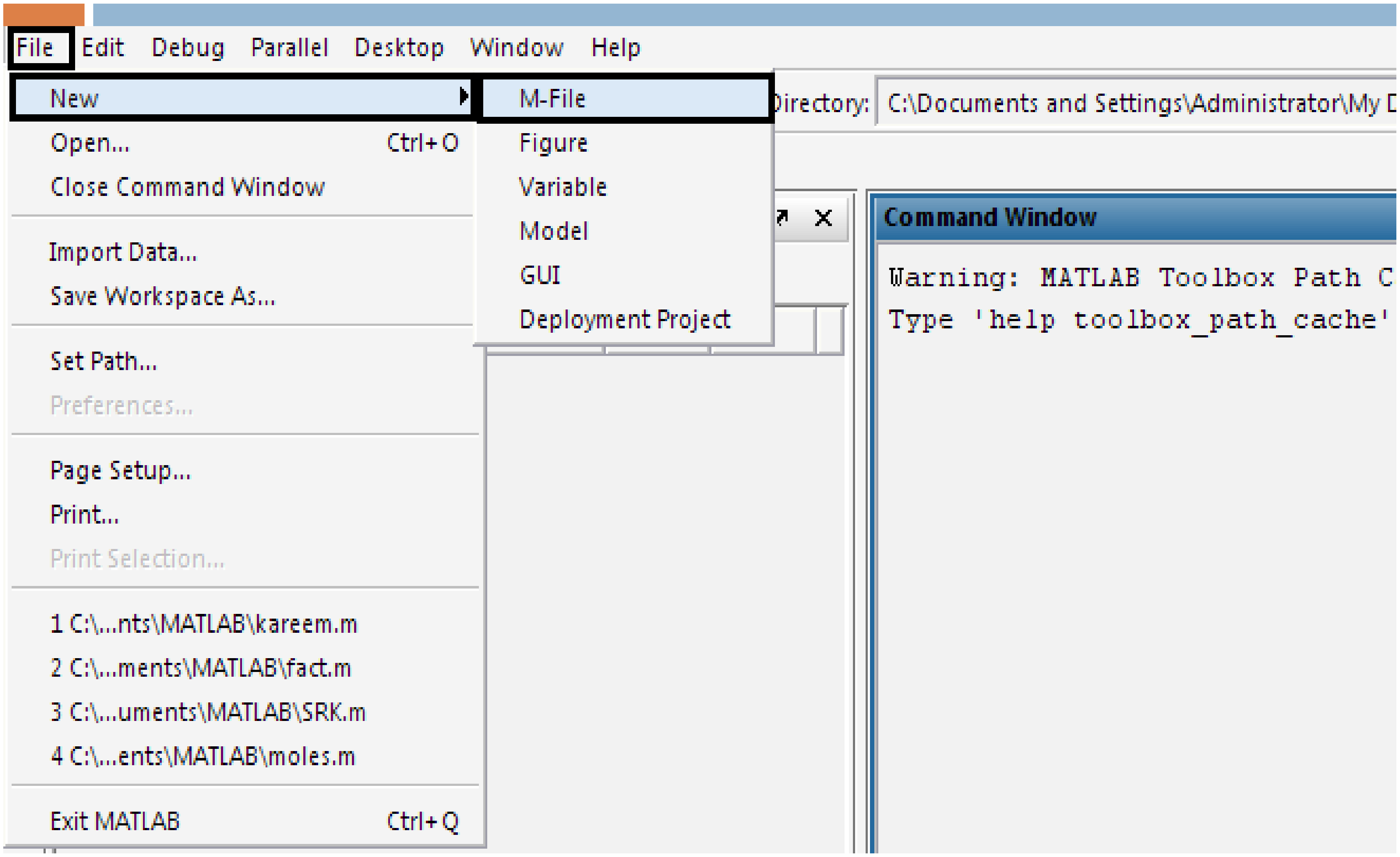
Why to use M-files

- Writing inside the **Command Window** may cause several problems, like:
  - If you wrote an order or an equation and then you wanted to change a value of any variable, you will have to re-write the equation again.
  - If you wrote a long program contains lots of commands and wanted to re-use this program again, you will have to repeat each command with the same order.
  - If there is a mistake in one command in such program you will need to re-write all the others from the beginning.

Why to use M-files

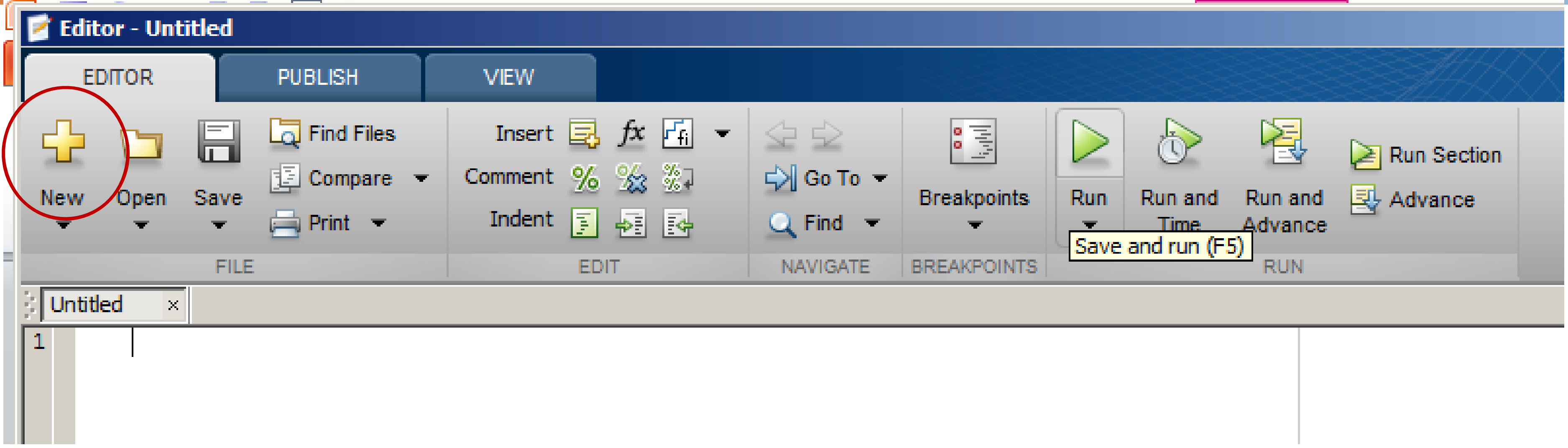
- Writing a program inside the M-file overcomes these problems since it works as a text editor where you can write the whole program without run. Then you can run the program after finishing. This gives you the ability to change the variables values without re-writing the whole program again.

To open new M-file



```
Warning: MATLAB Toolbox Path C
Type 'help toolbox_path_cache'
```

To open new M-file



OR

- use the edit command, e.g.

```
>> edit plotrand
```

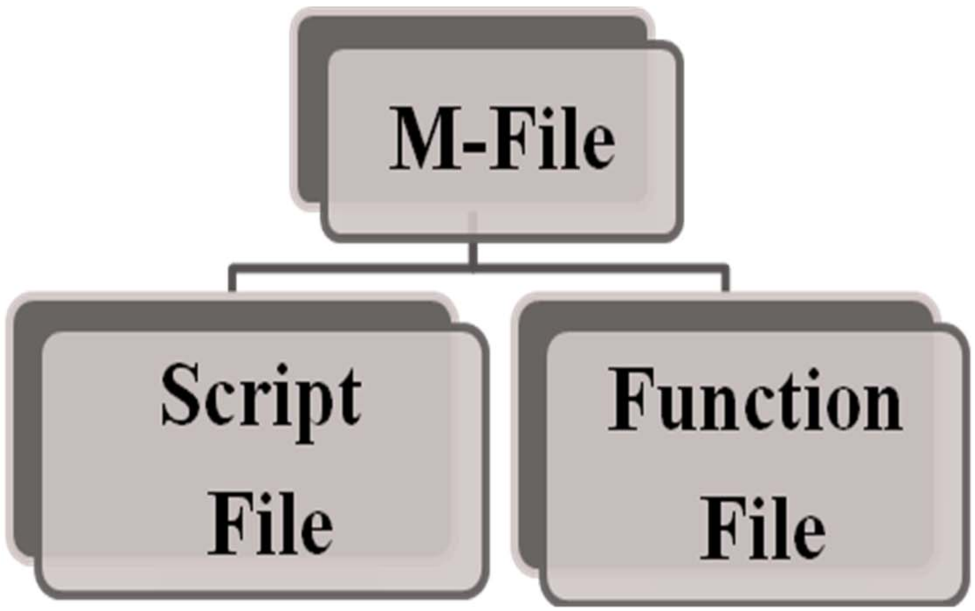
- This opens a blank file named plotrand.m

M-file Naming

- M-file names as variable names:
  - The first character in a M-File name must be a letter.
  - Contains no signs, e.g.: * % / **?** **+** , ………etc.
  - Contains no spaces like: ***cubic root***. You can type it ***cubic_root***
  - Not an order or a function name, e.g.: ***help, sin, ode45,*** ………… etc.
  - It is a good idea to use appropriate and memorable names for functions.
  - To avoid confusion, make sure your M-files don't have the same name as previously defined variables, M-files or MATLAB functions.

M-files

- M-Files can be classified into 2 categories depending on their use:



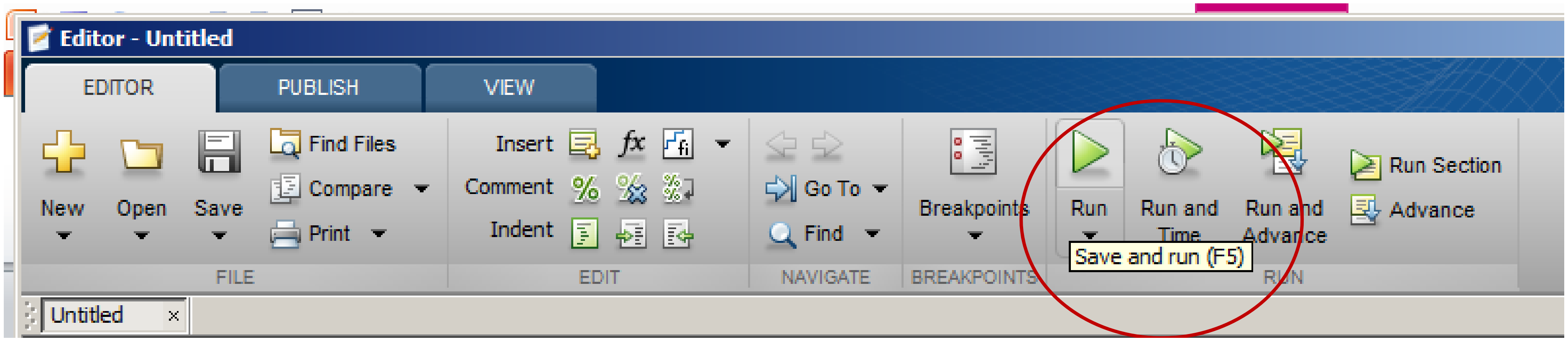
Script Files

- A script file is a regular text file that contains a series of MATLAB commands written just as you would enter them in the MATLAB Command Window.

- Statements that begin with % are considered to be **comments** and are ignored by MATLAB.

Script Files

- To perform commands in a script file in MATLAB Command Window:
  - Simply enter the name of the script file **without the ".m", or**
  - click ***F5*** after saving the script with a suitable name, or
  - click ***Save and run*** in the toolbar



Script Files

- Example:

```
>> edit areacir
```

- Type the following statements:

```
% areacir.m: example m-file to
% compute the area and circumference of
  a circle
r = 2.5;
area = pi*r^2
circum=2*pi*r
```

Script Files

- Typing **`areacir`** at the MATLAB prompt will yield the following MATLAB response:

```
>> areacir
  area =
              19.6349
  circum =
              15.7079
```

- which are the area and circumference of a circle of a radius 2.5.

Script Files

- **Note:** Entering

```
>> help areacir
```

gives you back the text in the comment lines at the beginning of the file:

```
% areacir.m: example m-file to
% compute the area and circumference of
  a circle
```

it is very desirable to include at least some brief comment as a header to each m-file you create.

Script Files

- MATLAB treats script files exactly as if they are command sequences
- All variables currently in the MATLAB workspace can be used by the script file commands
- All variables created by the script file are available for use after the script file has been run.
- Example - examine **area:**

```
>> area
area  =
          19.6349
```

Function-files

- Function files provide extensibility to MATLAB.
- You can create new functions specific to your problem, which will then have the same status as other MATLAB functions.
- Function definitions are stored in files with the name ***function_name.m.***

Function-files

- For a function returning one variable, the first line of a function definition must start with the command ***function*** and can be of the form:

```
function Dependant_variable =
function_name(Inependent_variable1,...)
```

This specifies **the name of the function** and its **input arguments**.

Function-files

- We would like to create a function called ***circlarea***, which calculates the area of a circle having any given radius ***r***.

```
% This function calculates the area of
% a circle. The arguments r and y
% represent the radius and the area of
% the circle, respectively.

function y = circlarea(r)
y=pi*r^2;
```

Function-files

- We can now invoke the function called ***circlarea*** in the same way as we do for commonly used MATLAB functions.
- In order to find the area of a circle with a radius 6.5, we can type the following:

```
>> A = circlarea (6.5)
```

and this is what we get:

```
A =
     132.7323
```

Function-files

- If you want the function to return more than one value, let's say two values, the first line of the function definition must start with the word function and can be of the form:

-
  ```
  function [variable1, variable2] =
  function_name(Independent_variable1,...)
  ```

Function-files

- For example suppose we want to create the function called ***circle***, which returns the area and the perimeter of a circle with any given radius. The M-file for this function will be:

```
% This function calculates the area of a
  circle and its perimeter
% The arguments r, y and p represent the
  radius, the area and
% perimeter of the circle, respectively
function [y, p] = circle(r)
y = pi*r^2;
p = 2*pi*r;
```

Function-files

- We can now invoke the function called ***circle*** in the same way as we do for commonly used MATLAB functions. Therefore, in order to find the area of a circle with a radius 6.5, and its perimeter we can type the following:

```
>>[Area Perim]=circle (6.5)
```

In this case we get

```
Area =
  132.7323
Perim =
   40.8407
```

Function-files

- A script file name can only be entered by itself at the MATLAB prompt (or as a line in another script file)
- A function can be called when needed
- Example
  - if ***p*** is a variable that has been specified, you could use ***circlarea*** in an expression such as:

```
>> p=2
>> differ = 100 - circlarea(p)
```

Summary

Script File	Function File
A list of commands to be performed with same order, easier to modified than in Command Window.	To create a new function in MATLAB like those already exist as: ***sin***, ***cos***, ***sqrt***, ...etc
Doesn't contain the command ***function*** in the first line	Must contain the command ***function*** in the first line
Variables are global: so you can use the variables defined in a script file after running into the Command Window	Variables are local: so the variables defined in a function file are used only inside it and can't be used inside the Command Window
To run: by entering the file name in the Command Window. Or by clicking Run.	To substitute in: as any other function in MATLAB by writing the function name & the arguments in Parentheses

Example of a Script File

Problem:

The speed v of a falling object dropped with no initial velocity is given as a function of time t by $v = gt$.

Plot v as a function of t for $0 \leq t \leq t_f$, where t_f is the final time entered by the user.

Example of a Script File

```
% Program falling_speed.m:
% Plots speed of a falling object.
% Created on March 1, 2004 by W. Palm
%
% Input Variable:
% tf = final time (in seconds)
%
% Output Variables:
% t = array of times at which speed is
% computed (in seconds)
% v = array of speeds (meters/second)
```

Example of a Script File

```
% Parameter Value:
g = 9.81; % Acceleration in SI units
% Input section:
tf = input('Enter final time in seconds:');
```

Example of a Script File

```
% Calculation section:
% Create an array of 500 time values.
t = linspace(0,tf,500);
% Compute speed values.
v = g*t;
%
% Output section:
Plot(t,v),xlabel(’t (s)’),ylabel(’v
  m/s)’)
```

Notes on Function-files

A function may have no input arguments and no output list.

For example, the function `show_date` computes and stores the date in the variable `today`, and displays the value of `today`.

```
function show_date
today = date
```

Examples of Function Definition Lines

1. One input, one output:

```
function [area_square] = square(side)
```

2. Brackets are optional for one input, one output:

```
function area_square = square(side)
```

3. Two inputs, one output:

```
function [volume_box] = box(height,width,length)
```

4. One input, two outputs:

```
function [area_circle,circumf] = circle(radius)
```

5. No named output: `function sqplot(side)`

Function Example

```
function [dist,vel] = drop(g,vO,t)
% Computes the distance travelled and
% the velocity of a dropped object,
% as functions of g,
% the initial velocity vO, and
% the time t.
vel = g*t + vO;
dist = 0.5*g*t.^2 + vO*t;
```

Function Example (continued)

1. The variable names used in the function definition may, but need not, be used when the function is called:

```
>>a = 32.2;
>>initial_speed = 10;
>>time = 5;
>>[feet_dropped,speed] = ...
 drop(a,initial_speed,time)
```

Function Example (continued)

2. The input variables need not be assigned values outside the function prior to the function call:

```
[feet_dropped,speed] = drop(32.2,10,5)
```

3. The inputs and outputs may be arrays:

```
[feet_dropped,speed]=drop(32.2,10,[0:1:5])
```

This function call produces the arrays `feet_dropped` and `speed`, each with six values corresponding to the six values of time in the array `time`.