Input and Output Options

Matlab offers a number of methods of entering or display data, both on the screen or for submission. Effective presentation of your results is an important step in any problem or project. This tutorial will cover input and output options other than the default settings in Matlab.

You can use MATLAB command named **input** for input numbers and texts at any time. This command will display a text string, specified by you, and then wait for input. Be sure to tell the user how to enter the data, especially if it is to be entered as an array.

```
>> time = input('Make an initial guess for the time in seconds time = ')
Make an initial guess for the time in seconds time = 32
time =
32
```

Multiple values may be entered if they are put in square brackets.

```
>> conc = input('Enter as a row array in brackets [ ] ')
Enter as a row array in brackets [ ]
```

Make a guess for the concentrations of A and B.
Enter the values as a row array in square brackets. **[0.5 0.3]**

```
>>conc =
0.5000 0.3000
```

Text Output with disp

Output to the command window is achieved with the **disp** function.
Syntax: disp(*outMatrix*)
where *outMatrix* is *either* a string matrix or a numeric matrix.

Examples: *Numeric output*

```
>> disp(5)
5
>> x = 1:3; disp(x)
1 2 3
>> y = 3-x; disp([x; y])
1 2 3
2 1 0
>> disp([x y])
1 2 3 2 1 0
```

Example: *String output*

```
>> disp('Hello, world!')
Hello, world!
```

M-files

In MATLAB you write programs in **M-files**. M-files are ordinary ASCII text files written in MATLAB language. They are called M-files because they must have a '.m' extension at the end of their name. M-files can be created using any editor or word-processing application. However, MATLAB has a built-in M-file editor which can be accessed through the Main Menu by

Files > New> M-file

There are two types of M-files – **Script** files and **function** files. We now discuss their purpose and syntax.

Script Files

A script file is an M-file with a set of valid MATLAB commands in it. A script file is executed by typing the name of the file on the command line. This is equivalent to typing all the commands in the script file one-by-one. Script files work on global variables, that is, variables currently present in the workspace. Results obtained from executing a script file are left in the workspace.

Example

We will write a script file to solve the following system of equations.

$$\begin{bmatrix} 5 & 2r & r \\ 3 & 6 & 2r-1 \\ 2 & r-1 & 3r \end{bmatrix} \bullet \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 3 \\ 5 \end{Bmatrix}$$

or **Ax = B.**

Clearly, **A** depends on the parameter r, and we want to find the solution of the equations for various values of the parameter r. We will write a script file to do this and call it **solvex.m** which we will save in our workspace.

```
A = [5 2*r r; 3 6 2*r-1; 2 r-1 3*r];   % create the matrix A
b = [2; 3; 5];     % create the vector b
x = A\b            % solve for x
```

If we execute the script by typing **solvex** we get an error,

```
??? Undefined function or variable 'r'.
Error in ==> solvex at 2
A = [5 2*r r; 3 6 2*r-1; 2 r-1 3*r];   % create the matrix A
```

This error arises because we have not defined the variable r. So first we must put r into the workspace and then execute the above script file again.

```
>> r = 1;
>> solvex;
```
The results will be
```
x =
   -0.0312
    0.2344
    1.6875
```

Note that the value of r is specified outside the script file. All variables in the workspace are available for use within script files, and all variables in the script file are left in the work space. Check this. So if you want to do a big set of operations but in the end you only want a few outputs, then a script file is not the correct choice. In this case you should use a **function file**.

Caution using a Script file:

1) Never name a script file the same name as the name of a variable it computes.
2) The name of a script file must begin with a letter; the rest of the name may include digits and the underscore character as well as letters. Names can be up to 19 characters long.
3) Avoid names that clash with built-in functions. It is a good idea to check if a file with the proposed name already exists. You can do this with the command exist('filename') this returns a zero id nothing with the name filename exists. Type help exist for more detailed information.

Function files

An M-file that contains the word **function** at the beginning of the first line is a function file. A function differs from a script in that arguments may be passed, and variables defined and manipulated inside the file are local to the function and don't operate globally on the workspace.

By means of function files we can add new functions to Matlab. A function file contains a sequence of commands, like a script file, but by means of input and output variables it is possible to communicate with other files.

Globally a function file has the following form:

```
function output_variable = function_name(input_variable)
commands
```

The word function in the first line indicates it is not a script file.

Remark: At function_name you need to fill in the file name without the extension .m .

Both the input variable as well as the output variable may be vectors.

For example:

```
function y = ratio(v);
% This function computes the ratio of the elements of vector v.
y = v/sum(v);
```

Variables used in a function file are local (i.e. only accessible inside the function), and therefore they are not stored in the Matlab memory.

```
function y = comp(x);
a = 2; % the variable a has the value 2, but only within this function
y = a * x;
```

Often it is necessary to use a variable in a function that gets its value outside the function. You can show Matlab you want to use such a variable by adding it to the list of parameters of that function. In the following example we add the variable **a** to the list of parameters of the function comp:

```
function y = comp(x,a);
y = a * x; % the variable a gets its value outside the function and is
% given to the function
```

This is not always possible. It may sometimes be necessary to define a variable globally, by using **global**. If you do so the variable is defined in all program parts that use the same declaration.

```
function y = comp (x);
global a;
y = a * x; % the variable a is also defined globally at another
% location where a gets its value
```

Remark: The declaration with **global** should only be used when there is no other possibility.

Example

We will now write a function to solve the system of equations defined above. We give the function a value of **r** as input and **x** will be output. We will call this function **solvexf.m**

```
function [x]=solvexf(r);
A = [5 2*r r; 3 6 2*r-1; 2 r-1 3*r];   % create the matrix A
b = [2; 3; 5];                         % create the vector b
x = A\b;                               % solve for x
```

Now r and x are all local variables. Any other variable names can be used in their place in the function call statement. Sample output would be as follows:

[y] = solvexf(1)

The result will be as below

```
 y =
      -0.0312
       0.2344
       1.6875
```

The only variables left in the workspace after execution will be the variables in the workspace.

Let's create a m-function which when passed an x value, returns the value of $f(x) = x^3 + x^2 - 3x - 3 = 0$. In the MatLab editor, open a new page and type the following:

```
function [y] = f1(x)
y = x.^3 + x.^2 - 3*x - 3;
```

Now save this as f1. Matlab will automatically add the .m extension

To call this m-function at the command prompt, for example to find the value of f(9), and assign to a variable z, we type :-

```
>> z=f1(9)
z =
  780
```

Note, that f1.m will also work for vector:-

```
>> z=f1(-1:.5:1)
z =
    0   -1.3750   -3.0000   -4.1250   -4.0000
```

Of course, Matlab m-functions can have multiple arguments and multiple return values, e.g.

Notes

a) **function[u,v,w] = yourfun (x,y)**;% For 3 outputs u, v, w and 2 inputs x, y

b) **function[W]= myfun(x,y,z);**%For 1 output W and three inputs x, y, z

OR equivalently

function W = myfun(x,y,z)

c) **[A,B,C] = yourfun(-5,4)**

This means that input values are x=-5, y=4, and output values for A, B, and C, are calculated according to the function prescription

d) [M] = myfun(2,4,-3)

OR equivalently

M = myfun(2,4,-3)

Example

(a) Write a function that performs an ideal gas calculation, where the function is called as follows:

IdealGas(P,V,T,R), and returns the value of n, the number of moles.

```
function n = IdealGas(P,V,T,R)
n = P*V/R/T;
```

(b) Apply the function in the command window to calculate the number of moles that exist in a volume of 22.4 L, at a pressure of 1 atm, and at a temperature of 273 K.

```
>> IdealGas(1,22.4,273,0.082)
ans =
1.0006
```

(c) Write a script file IG_main.m that performs the ideal gas calculation to determine number of moles. The script should ask the user for each of the other four quantities.

```
R = input('Enter the universal gas constant: ');
P = input('Enter the gas pressure: ');
T = input('Enter the gas temperature :');
V = input('Enter the gas volume: ');
n = IdealGas(P,V,T,R);
disp('Moles of ideal gas =')
disp(n)
```

Example: Ideal gas law

Pressure calculation using the ideal gas law

Input: V in [L], n in [moles], and T in [K]

Output: P in [atm]

```
function [P]=pressure(V,n,T)
R=0.082 ;% [L atm/K mol]
P=n*R*T/V;
```

Example

The quadratic equation provides the roots to the polynomial $ax^2+bx+c=0$ according to: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Write a function called qe(a,b,c), that solves the quadratic equation and returns solutions. Use your function to solve for the roots of the equation $x^2 + 6x - 91 = 0$.

```
function [x1, x2] = qe(a,b,c)
x1 = (-b + sqrt(b.^2 -4.*a.*c))/(2.*a);
x2 = (-b - sqrt(b.^2 -4.*a.*c))/(2.*a);

>> [x1,x2]=qe(1,6,-91)
  x1 = 7
  x2 = -13
```

Example

Function for calculating enthalpy depending on entering temperature, reference temperature and vector of specific heat equation constants.

```
function dH = deltaH_IG(Ti,Tf,Cp)
 dH = Cp(1)*(Tf-Ti)+Cp(2)*(Tf^2-Ti^2)/2+Cp(3)*(Tf^3-Ti^3)/3+Cp(4)*(Tf^4-Ti^4)/4;
```

Cp must be a vector of four values

Example

Vapor Pressure for a given Temperature ln P = A - B/(T+C)
Input: Antoine Constants (A,B,C), T
Output: Vapor Pressure
A, B, C constants of Antoine equation.

```
function vp = sat_pr_antoine(A, B, C, T)
vp = exp(A - B./(T+C));
```

Example

Heat capacity of air in J/(mol K), where T is in K. The equation is valid only in the range 273 K < T < 1800 K the file is called as Cp = CpAir(T)

```
function Cp = CpAir(T)
if T < 273
   disp('Temperature too low')
elseif T > 1800
   disp('Temperature too high')
else
   Cp = 28.09 + 0.1965e-2*T + 0.4799e-5*T^2 - 1.965e-9*T^3;
end
```

Example

MATLAB M-function developed for the pressure-explicit van der Waals equation. This M-function calculates pressure according to the van der Waals equation of state

$$P = \frac{RT}{\underline{V} - b} - \frac{a}{\underline{V}^2}$$

The code shown below must be saved in a file with the following name: **VdWEOS_P.m**

```
function P = VdWEOS_P(V, T, VdWPars)
T=T + 273.15        ; % [K]
R= 8.314e-2         ; % [m^3-bar/K-kmol]
a=VdWPars(1)        ;
b=VdWPars(2)        ;
P=R*T/(V-b)-a/V^2 ;% [bar]
```

Note: functions can be saved as separate files and called from any program or embedded into other functions and used from the main function only. In the latter case specify the main program as a function.

Global Variables

If you want more than one function to share a single copy of a variable, simply declare the variable as global in all the functions. Do the same thing at the command line if you want the base workspace to access the variable. The global declaration must occur before the variable is actually used in a function. Although it is not required, using capital letters for the names of global variables helps distinguish them from other variables. For example, create a new function in a file called falling.m:

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

Then interactively enter the statements

```
global GRAVITY
GRAVITY = 2;
y = fall((0:.1:5)')
```

The two global statements make the value assigned to GRAVITY at the command prompt available inside the function. You can then modify GRAVITY interactively and obtain new solutions without editing any files.

Saving Your Work

When using MATLAB, you will frequently wish to end your session and return to it later. Using the save command, you can save all of the variables in the MATLAB workspace to a file. The variables are stored efficiently in a binary format to a file with a ".mat" extension. The next time you start MATLAB, you can load the data using the load command. See "help save" and "help load" for details.

If you want to save all your variables and their current values, type:

>> **save filename**

Before you exit MATLAB, and your work will be saved in a file named *filename*.mat (default filename matlab.mat). If you only want to save some of the variables, you can give the **save** command the names of the variables to save. If you type:

>> save filename x y

MATLAB will save just the variables **x** and **y** in the file filename.mat.

When you start up MATLAB at some future time, you can restore all your variables from the file filename.mat by typing:

>> **load filename**

The command **load**, by itself, loads all the variables saved in matlab.mat.

A file with the extension .mat is assumed to be binary MATLAB format. To retrieve the variables from *filename*.mat, the command is:

>>load filename

To retrieve the one variable from *filename*.mat, the command is:

>>load filename variable_name

Example

```
>> x=1:1:10
x =
   1   2   3   4   5   6   7   8   9   10

>> y=2:2:20
y =
   2   4   6   8   10   12   14   16   18   20
```

```
>> z=x.*y
z =
    2    8   18   32   50   72   98  128  162  200
>> save data
```

All existed data in the workspace will saved in the data name

```
>> clear all
>> load data x
>> x
x =
    1    2    3    4    5    6    7    8    9   10
>> y
??? Undefined function or variable 'y'.
>> z
??? Undefined function or variable 'z'
```