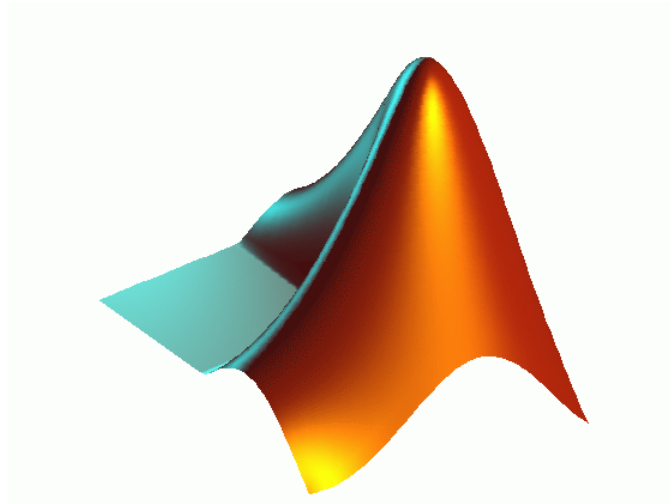




# Lab Notes

# MATLAB

## Session (3)



*Eng. Sara Hassan Kamel*

## Using “input” and “disp” functions

### ***Input Function (input):***

```
user_entry = input('prompt')  
user_entry = input('prompt', 's')
```

Use “input” function when you want to ask the user to enter a certain value then you can save it in a certain variable.

Example:

```
>> x = input('Enter the value of x: ')
```

The previous command displays the following in the command window:

```
Enter the value of x:
```

Now the user should type in the value of x, say 10. So type 10 and click enter. This way the value 10 is assigned to the variable x and it appears in the workspace.

Example:

```
>> word = input('Enter password: ', 's')
```

The previous command has an extra input ‘s’ which means that whatever the user types in will be treated as a string not a numerical value. So if you type in lab3 then lab3 will be treated as a string and stored in the variable word.

---

### ***Display Function (disp):***

Use this function to display a certain string or array to the user in the command window. Here whatever you type in as an input to the function disp will not be stored anywhere, it will just be displayed.

Example:

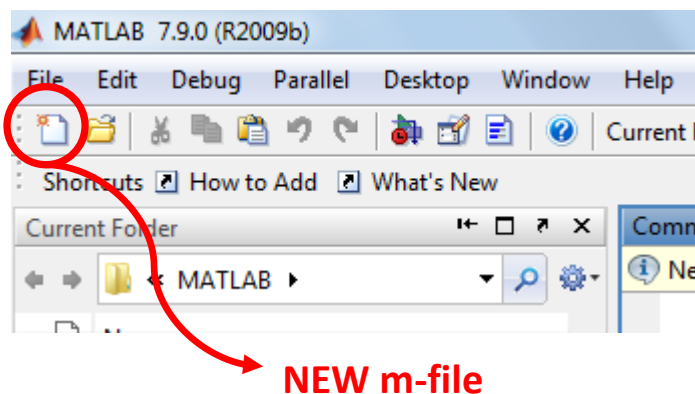
```
>> disp('          Corn          Oats          Hay')  
>> disp(rand(5,3))
```

Results in:

Corn	Oats	Hay
0.2113	0.8474	0.2749
0.0820	0.4524	0.8807
0.7599	0.8075	0.6538
0.0087	0.4832	0.4899
0.8096	0.6135	0.7741

---

## **MATLAB m-files:**



Instead of writing commands and executing them one by one in the command window we can create a file, write all commands and then run the file. There are two types of m-files:

### **SCRIPTS:**

- No inputs or outputs defined
- Consist of a sequence of instructions
- You can choose any filename to save the script

### **FUNCTIONS:**

- Can have inputs and outputs
- Must start with the following line:  

```
function [op1,op2,...]=function_name(ip1,ip2,...)
```
- The filename must be the same as the name specified in the previous (first) line i.e. in this example it should be **function\_name.m**

Note that both scripts and functions have the extension “.m”, but the content of each type is different as mentioned earlier.

### Example:

Write a function and a script that count the number of elements of a given matrix. Test your program for the matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 0 & 0 \\ 2 & 0 & 4 \end{bmatrix}$$

Now to do that you can use the function `numel ( )` which directly gives the number of elements in the given matrix. Another solution would be to use the function `size` which had 2 outputs: the number of rows and the number of columns, then multiply the two results.

### SCRIPT:

```
A=input('Enter the input matrix: ');
x=numel(A);
disp(['The number of elements in the given matrix is ', num2str(x)])
```

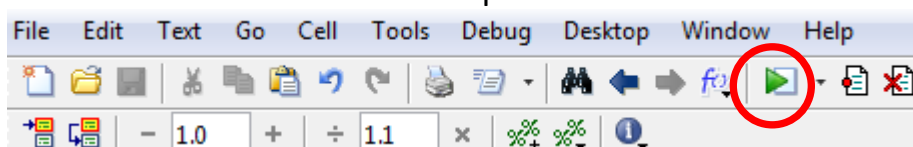
Or use this:

```
disp('The number of elements in the given matrix is:')
disp(x)
```

Where the function “**num2str**” converts `x` from a numerical value to a string. We use this function because the input to **disp** must be only one input; either a string or a numerical value/array. So we construct a vector whose elements are all strings, the first part is the string ‘The of elements in the given matrix is ’ and the other is the number in `x` converted to string.

To run the script and execute the commands in the script, there are two ways:

1. Type the filename in the command window and click enter  
`>> lab3_script1`
2. Click the “Run” button at the top of the m-file window:



**FUNCTION:**

Here we shall try using **size** instead of **numel**

```
function [x]=lab3_fn1(A)
[r c]=size(A);
x=r*c;
```

Now the function **lab3\_fn1** can be used the exact same way you use **numel** or any other built-in function, as long as the function is saved in your current directory.

```
>> B=[1 2 3;4 5 6;1 0 0;2 0 4]
```

```
B =
```

```
     1     2     3
     4     5     6
     1     0     0
     2     0     4
```

```
>> numel(B)
```

```
ans =
```

```
    12
```

```
>> lab3_fn1(B)
```

```
ans =
```

```
    12
```

Write a function that replaces a specific column of a given matrix by some other column vector. Test your program on the matrix in the previous problem. Try replacing the second column by the vector  $v = [1 \ 1 \ 0 \ 0]^t$ .

```
function [A_new]=lab3_fn2(A,x,n_col)
A_new=A;
A_new(:,n_col)=x;
```

The last line means that we replace all rows in the column number `n_col` in the matrix `A` by the vector `x` (which must be a column vector whose size is suitable for this replacement to occur).

```
>> lab3_fn2(B,[1;1;0;0],2)
```

```
ans =
```

1	1	3
4	1	6
1	0	0
2	0	4

Write a script file that takes as input a matrix and displays a menu of elementary row operations (interchanging rows, addition of a multiple of a row to another, multiplying a row by a non-zero constant) together with an exit option. Display the resulting matrix after performing the desired row operation.

We will use the **switch** function:

Syntax:

```
switch variable
    case value1
        ....
    case value2
        ....
    :
    :
    otherwise
        ....
end
```

**ANSWER:**

```

A=input('Enter the matrix: ');
disp('Which of these row operations do you want to execute?')
disp('(1)Interchanging Rows')
disp('(2)Multiplying a row by a Scalar')
disp('(3)Adding Two Rows')
x1=input('Enter 1,2 or 3 for the required row operation: ');
switch x1
    case 1
        disp('Which rows do you want to interchange?')
        r1=input('Number of 1st row: ');
        r2=input('Number of 2nd row: ');
        temp=A(r1,:);
        A(r1,:)=A(r2,:);
        A(r2,:)=temp;
        disp(A)
    case 2
        c=input('Scalar: ');
        r=input('Number of row: ');
        x=c*A(r,:);
        A(r,:)=x;
        disp(A)
    case 3
        r1=input('First row: ');
        r2=input('Second row (in which the addition operation
is executed: ');
        x=A(r1,:)+A(r2,:);
        A(r2,:)=x;
        disp(A)
end

```

---

Write a function to plot a circle. (Hint: use the parametric representation of the circle).

A circle of radius “a” is written in parametric form as:

$$X=a*\cos(t)$$

$$Y=a*\sin(t)$$

NOTE:

A function may not have numerical outputs, such as this one, so we omit the output part from the first line and we just write

```
function fn_name(ip1,...)
```

So the function will be written as follows:

```
function plotcircle(r)
t=0:0.01:2*pi;
x=r*cos(t);
y=r*sin(t);
plot(x,y)
axis square
```

**This makes the y-axis appear the same length as the x-axis so the circle doesn't appear oval.**

---

**Write a script and a function according to the description:**

**1) Write a program to plot the function  $\text{Acos}(wt)$ . The user should be able to specify the amplitude  $A$ , the frequency  $w$ , the range of  $t$  (starting point, end point and step size separately) as well as the line color, type and marker type.**

**Solution:**

**Script:**

```
ts=input('Enter the starting point of the range: ');
tf=input('Enter the end point of the range: ');
tstep=input('Enter the step: ');
A=input('Enter the Amplitude of the Sine Wave: ');
w=input('Enter the value of w in Acos(wt): ');
spec=input('Specify the line color, type and marker: ','s');
t=ts:tstep:tf;
x=A*sin(w*t);
plot(t,x,spec)
title('Sine Function')
xlabel('x')
ylabel('y')
grid on
```



**Function: (the filename is plot2Dsin.m)**

```
function plot2Dsin(ts,tf,tstep,A,w,spec)
t=ts:tstep:tf;
x=A*sin(w*t);
plot(t,x,spec)
title('Sine Function')
xlabel('x')
ylabel('y')
grid on
```

---

**Exercises:**

- 1- Write a function that solves a  $3 \times 3$  system of linear equations using Cramer's rule. Your program should test whether the input matrix is singular in which case displays an error message. Make use of the function you created in problem 2. Test your program using the following system of linear equations:

$$x - y + z = 3, \quad x + 3y + z = -1, \quad x - 3z = -2.$$

- 2- Write a function that computes the mean, variance and standard deviation of a given set of data points. Test your program using the following data set:  $x_1 = 90, x_2 = 75, x_3 = 93, x_4 = 69$ . Display helpful messages. Compare your results with those obtained using the corresponding MATLAB built-in functions. (Hint: the estimates for the

mean, variance and standard deviation are given by  $m = \frac{1}{n} \sum_{i=1}^n x_i$ ,

$$v = \sum_{i=1}^n (x_i - m)^2 / (n - 1) \text{ and } s = \sqrt{v}, \text{ respectively})$$

**HINT: Cramer's Rule:**

Consider a system of linear equations represented in matrix multiplication form as follows:

$$Ax=b \quad \text{Where } A \text{ is invertible.}$$

Then the theorem states that:

$$x_i = \frac{\det(A_i)}{\det(A)} \quad i = 1, \dots, n$$

where  $A_i$  is the matrix formed by replacing the  $i$ th column of  $A$  by the column vector  $b$ .

Example:

$$2x + y + z = 3$$

$$x - y - z = 0$$

$$x + 2y + z = 0$$

Coefficient Matrix is  $D$ :

$$D = \begin{vmatrix} 2 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & 2 & 1 \end{vmatrix}$$

$$D_x = \begin{vmatrix} 3 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 2 & 1 \end{vmatrix} \quad D_y = \begin{vmatrix} 2 & 3 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{vmatrix} \quad D_z = \begin{vmatrix} 2 & 1 & 3 \\ 1 & -1 & 0 \\ 1 & 2 & 0 \end{vmatrix}$$

Evaluating each determinant, we get:

$$D = \begin{vmatrix} 2 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & 2 & 1 \end{vmatrix} = (-2) + (-1) + (2) - (-1) - (-4) - (1) = 3$$

$$D_x = \begin{vmatrix} 3 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 2 & 1 \end{vmatrix} = (-3) + (0) + (0) - (0) - (-6) - (0) = -3 + 6 = 3$$

$$D_y = \begin{vmatrix} 2 & 3 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{vmatrix} = (0) + (-3) + (0) - (0) - (0) - (3) = -3 - 3 = -6$$

$$D_z = \begin{vmatrix} 2 & 1 & 3 \\ 1 & -1 & 0 \\ 1 & 2 & 0 \end{vmatrix} = (0) + (0) + (6) - (-3) - (0) - (0) = 6 + 3 = 9$$

Cramer's Rule says that  $x = D_x \div D$ ,  $y = D_y \div D$ , and  $z = D_z \div D$ . That is:

$$x = 3/3 = 1, y = -6/3 = -2, \text{ and } z = 9/3 = 3$$

## **PART (II)**

### **Introduction:**

In this session, you will learn how to use some programming functions in MATLAB. These functions include “for”, “if”, “switch” and others. Such functions are best explained through examples. This session focuses on “for”.

### **for:**

Use “for” to create a loop, i.e. when you want a certain command or a set of commands to be executed several times.

```
for variable = initval:step:endval
    statement
    ...
    statement
end
```

Where the “variable” is the counter, you can give it any name, it’s just a variable. For-loops are best explained through examples.

### **Example (1):**

Use a for-loop to sum the numbers from 1 to 5.

### **Answer:**

```
s=0;
for n=1:5
    s=s+n;
end
disp(s)
```

Now we shall explain how this works:

The idea is to create a variable, give it an initial value and use the loop to add the numbers inside this variable one by one:

$$S_{\text{new}} = S_{\text{old}} + n$$

n	Old value of s	New value of s (s=s+n)
1	0	1+0=1
2	1	2+1=3
3	3	3+3=6
4	6	4+6=10
5	10	5+10= <b>15</b>

Notice that n is a vector (1:5 = [1 2 3 4 5]) so when we enter the loop for the first time, n takes the value of the first element, and so n=1.

In the second iteration (step), n takes the following element's value - which is 2 - and so on... (see the table above).

### **Example (2):**

Write a program (script) to sum up the first 10 even numbers (excluding zero).

### **Answer:**

```
s=0;
for n=0:2:20
    s=s+n;
end
disp(s)
```

OR

```
s=0;
for n=1:10
    s=s+2*n;
end
disp(s)
```

Both solutions are correct. However, if you don't know that the first 20 even numbers are from 2 to 20 (like for example if the question asked you to sum up

the first 57 or 396 even numbers!), then the second solution is the more appropriate one.

For the second solution:

$$S_{\text{new}} = S_{\text{old}} + 2n$$

n	Old value of s	New value of s (s=s+2n)
1	0	2*1+0=2
2	2	2*2+2=6
3	6	2*3+6=12
4	12	2*4+12=20
5	20	2*5+20=30
6	30	2*6+30=42
7	42	2*7+42=56
8	56	2*8+56=72
9	72	2*9+72=90
10	90	2*10+90= <b>110</b>

Note that each time we add an even number; because when n takes the values 1, 2, 3, 4 ... 10, 2n takes the values 2, 4, 6, 8 ...20.

### **Example (3):**

Write a function m-file to calculate the factorial of a given integer. Use the function name "fact".

### **Answer:**

```
function f=fact(n)
f=1;
for i=1:n
    f=f*i;
end
```

e.g.: The factorial of 5 → 5! = 5 x 4 x 3 x 2 x 1 = 1 x 2 x 3 x 4 x 5

In general n!= n(n-1)(n-2) ... 1

Here we initialize  $f$  to 1, because any number multiplied by 1 stays as it is.

The loop here keeps the result in  $f$ , each iteration it multiplies the previous value of  $f$  to the following integer in the sequence from 1 to  $n$ :

To try out the function, do not click "Run". Instead you go to the command window and try the function for any value of  $n$  – say 5:

```
>> fact(5)
```

n	f (Old Value)	f (New Value) → $f=f*n$
1	1	$1*1=1$
2	1	$1*2=2$
3	2	$2*3=6$
4	6	$6*4=24$
5	24	$24*5=120$

The result:

```
ans = 120
```

```
>> fact(0)
```

In this case we have  $f=1$ , then in the loop  $n=1:0$  (so this means a vector whose starting point is 1 and end point is 0 with a step equal to 1, and  $1+1=2 > 0$ .

Hence  $n$  is an empty matrix) and the for-loop is not executed at all, therefore  $f$  remains 1.

```
ans = 1
```

Indeed:  $0!=1$

**NOTE:**

MATLAB actually has built-in functions to do some of the examples introduced here, but still the examples show you how to create a for-loop and trace the code.

**sum(x)** → This function sums up the elements of x

**factorial(x)** → This function computes the factorial

**Example (3):**

Create the vector [1 2 4 8 16 .... 128]

**Answer:**

```
A=[];  
for i=0:7  
    A=[A 2^i];  
end  
disp(A)
```

This is what we do:

1. First we create an empty matrix A (or else when we reach the step inside the loop A will be undefined).
2. A=[A 2^i] will concatenate the new A and the old A to create the vector step by step.

To see how it works, we'll remove the semicolon and display the value of i:

```
A=[];  
for i=0:7  
    disp('i=')  
    disp(i)  
    A=[A 2^i]  
end
```

Output:

```
i=
    0
A =
    1
i=
    1
A =
    1     2
i=
    2
A =
    1     2     4
i=
    3
A =
    1     2     4     8
i=
    4
A =
    1     2     4     8    16
i=
    5
A =
    1     2     4     8    16    32
i=
    6
A =
    1     2     4     8    16    32    64
i=
    7
A =
    1     2     4     8    16    32    64    128
```



**Example (4):**

Write a script to create a 9x8 matrix whose rows are the same as the vector A in example (3). You'll need to regenerate A, don't use a previously created vector.

**Answer:**

Here we will use two for loops, one inside the other:

```
A=[];
for i=1:9
    B=[];
    for j=0:7
        B=[B 2^j];
    end
    A=[A;B];
end
disp(A)
```

1. First we initialize A as an empty matrix.
2. The first loop repeats 9 times, once for each row.
3. The code within the outer loop is the same as the one in example (3), including initializing the matrix B as an empty matrix to make it ready for concatenation.
4. After the inner loop is finished, matrix B contains the vector [ 1 2 4 ...128] and is vertically concatenated with A (which starts out as an empty matrix, then one row, then two and so on...)
5. The last line displays the matrix A

Note: This example is for you to learn how to trace two loops one embedded inside the other. However, you can create the vector [1 2 4 ... 128] and then use it in one loop to create the matrix. This way you'll use two separate loops.

```
B=[];
for j=0:7
    B=[B 2^j];
end
A=[];
for i=1:9
    A=[A;B];
end
disp(A)
```

**Example (5):**

Suppose you want to create a script that enables a user who does not know how to write a matrix using MATLAB to do so by prompting the user to enter the elements one-by-one. Also, the user should be able to decide the size of the matrix.

**Answer:**

```
m=input('Enter the number of rows: ');
n=input('Enter the number of columns: ');
A=[];
for i=1:m
    for j=1:n
        x=input('Enter the element: ');
        A(i,j)=x;
    end
end
disp(A)
```

First, we ask the user for the number of rows and columns, and we use these in the loops:

- ➔ The outer loops is for the rows and the inner one for the columns
- ➔ The total number of iterations (number of times the commands inside the inner loop is repeated) is the product of m and n:

For example: if m=3 and n=5: Total number of iterations =  $3 \times 5 = 15$

```
i=1
    j=1
    j=2
    j=3
    j=4
    j=5
i=2
    j=1
    j=2
    j=3
    j=4
    j=5
i=3
    j=1
    j=2
    j=3
    j=4
    j=5
```

This fills up the matrix A row by row:

$A(1,1) = \dots$

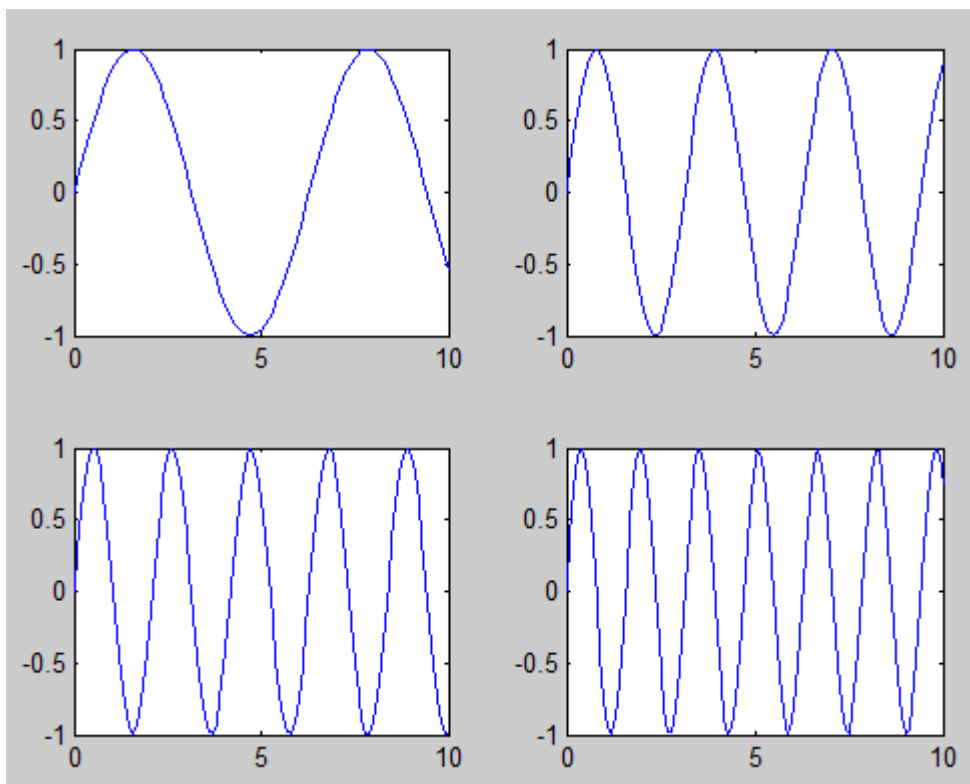
$A(1,2) = \dots$

$A(1,3) = \dots$

and so on...

### **Example (6):**

Write a script using a for-loop to produce the following plot where the functions drawn are  $\sin(x)$ ,  $\sin(2x)$ ,  $\sin(3x)$  and  $\sin(4x)$ :



```
x=0:0.1:10;  
for n=1:4  
    subplot(2,2,n)  
    plot(x,sin(n*x))  
end
```

⇒ Do not forget to define x before the loop (if it's inside, it doesn't matter but there's no use repeating it every iteration if it's the same in all plots).