University of Cyprus
Department of Electrical and Computer Engineering

ECE 327: Introduction to Matlab

Table of Contents

MATLAB

- Short for MATrix LABoratory
- Premier software package for electrical engineers
- Control system toolbox, signal processing toolbox, system identification toolbox, robust-control toolbox, and many others
- Matlab access for everyone at University of Cyprus!

GETTING HELP

Online help is available from the Matlab prompt by typing *help* followed by the name of a specific command:

```
help cos
```

```
  cos    Cosine of argument in radians.
     cos(X) is the cosine of the elements of X.

     See also acos, cosd, cospi.

     Documentation for cos
     Other functions named cos
```

The second way to get help is the *lookfor* command. The *lookfor* command searches the quick summary information in each function for a match.

(Exercise: Using the Command Window, look for a function to take the inverse of a matrix)

VARIABLES

In order to define a new variable in Matlab, the following formula is used

```
variable_name=value;
```

Expressions typed without a variable name are evaluated and the result is stored and displayed by a variable name *ans. For example,*

```
250/sin(pi/6)
```

```
ans = 500.0000
```

The result of an expression can be assigned to a variable name for further use. For example,

```
x=exp(-0.2696*.2)*sin(2*pi*.2)/(.01*sqrt(3)*log(18))
```

```
x = 18.0001
```

and the result is assigned to x.

Output can be suppressed by appending a "semicolon" to the command lines

```
x = 3;
y = x^2;
y
```

```
y = 9
```

For example, if we define a 1000 column array representing a time interval, there is no point in displaying the result on screen.

Active variables: At any time you want to know the active variables you can use the *who* command.

Removing a variable: To remove a variable, type *clear* followed by the variable name. (What does *clear all* command does?)

Variable arithmetic: Matlab uses some fairly standard notation. Powers are performed before division and multiplication, which are done before subtraction and addition. For example,

$$8 \div 2(2+2)?$$

VECTOR OPERATIONS

An n vector is a row or a column array of n numbers.

In Matlab elements enclosed by brackets and separated by semicolons generate a column vector. For example,

```
X = [2; -4; 8]
```

```
X = 3×1
     2
    -4
     8
```

If elements are separated by blanks or commas, a row vector is produced. Elements may be any expression. For example,

```
R = [tan(pi/4) sqrt(9) -5]
```

```
R = 1×3
    1.0000    3.0000   -5.0000
```

The tranpose of a column vector results in a row vector, and vice versa. For example,

```
Y = R'
```

```
Y = 3×1
    1.0000
    3.0000
   -5.0000
```

Vectors of the same size can be added or subtracted componentwise. However, for multiplication, specific rules must be followed in order to obtain the correct resulting values. The operation of multiplying a vector X with a scalar k (*scalar multiplication*) is performed componentwise. For example,

```
P = 5*R
```

```
P = 1×3
    5.0000   15.0000  -25.0000
```

The operator $.*$ performs element-by-element operation. For example, for the previously defined vectors, X and Y,

```
E = X.*Y
```

```
E = 3×1
    2.0000
  -12.0000
  -40.0000
```

The inner product or the *dot product* of two vectors X and Y denoted by $< X, Y >$ is a scalar quantity defined by $\sum_{i=1}^{n} x_i y_i$. If X and Y are both column vectors defined above, the inner product is given by

```
S = X'*Y
```

```
S = -50
```

To build a zero row vector of size 4, the following command can be used

```
Z = zeros(1,4)
```

```
Z = 1×4
     0     0     0     0
```

To build an all ones row vector of size 4, the following command can be used

```
Z = ones(1,4)
```

```
Z = 1×4
     1     1     1     1
```

In Matlab the colon `(:)` can be used to generate a row vector of equally spaced numbers. This is particularly useful when creating time intervals, but is also used inside `for` loops. The command is

```
t=a:step:b
```

where `a,b` represent numbers and `step` is the step value used (if `step` is omitted the default value is 1). For example,

```
t = 1:8
```

```
t = 1×8
     1     2     3     4     5     6     7     8
```

```
%% or, for negative increments
t1 = 8:-1:1
```

```
t1 = 1×8
     8     7     6     5     4     3     2     1
```

A second way uses the following command

```
t=linspace(a,b,n)
```

that creates an array of `n` equally spaces numbers between `a` and `b`. As an example,

```
t2=linspace(0,5,4)
```

```
t2 = 1×4
         0    1.6667    3.3333    5.0000
```

MATRIX OPERATIONS

In Matlab a matrix is created with a rectangular array of numbers surrounded by brackets. The elements in each row are separated by blanks or commas. A semicolon must be used to indicate the end of a row. For example,

```
A = [1 2 3; 4 5 6; 7 8 9]
```

```
A = 3×3
     1     2     3
     4     5     6
     7     8     9
```

If a semicolon is not used, each row must be entered in a separate line as shown below.

```
A = [ 1  2  3
      4  5  6
      7  8  9 ]
```

```
A = 3×3
     1     2     3
     4     5     6
     7     8     9
```

To choose among certain elements of a matrix, one can simply use parenthesis `(:)` to define which element or which parts of the matrix are to be chosen. For example,

```
r1 = A(1,:)
```

```
r1 = 1×3
     1     2     3
```

```
c3 = A(:,3)
```

```
c3 = 3×1
     3
     6
     9
```

Modifying any element of the matrix can be done by referring to that element, for example,

```
A(2,3) = 100
```

```
A = 3×3
     1     2     3
     4     5   100
     7     8     9
```

```
A1=A;
A1(3,:)=A(3,end:-1:1)
```

```
A1 = 3×3
     1     2     3
     4     5   100
     9     8     7
```

The commands $zeros(m,n)$ and $ones(m,n)$ produce an m-by-n matrix of zeros and ones, respectively.

Matrix addition/subtraction: The standard addition and subtraction requires that the two matrices have the same number of columns and rows. For example,

```
E = [ 2  3; 4 5; 6 7];
F = [ 1 -2; 3 6.5; 10 -45];
E+F
```

```
ans = 3×2
    3.0000    1.0000
    7.0000   11.5000
   16.0000  -38.0000
```

Matrix multiplication: Matrix multiplication requires that the sizes of the matrices match. If they don't, an error message is generated. For example,

```
 A= [ 1  2  3
      4  5  6
      7  8  9 ];
 B= [ 1  2  3
      4  5  6
      7  8  9 ];
  C = A*B
```

```
C = 3×3
    30    36    42
    66    81    96
   102   126   150
```

Example: For the matrix equation below. $AX = B$, determine the vector X.

$$\begin{bmatrix} 4 & -2 & -10 \\ 2 & 10 & -12 \\ -4 & -6 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -10 \\ 32 \\ -16 \end{bmatrix}$$

```
A = [4 -2 -10; 2 10 -12; -4 -6 16];
B = [-10; 32; -16];
X=inv(A)*B
```

```
X = 3×1
    2.0000
    4.0000
    1.0000
```

BRANCHING STRUCTURES

Matlab provides a number of language structures for branching a program's control of flow.

If-end structure:

```
a = 2; b = 2;
if a <= b
     c = 3
end
```

```
c = 3
```

Here the condition is a logical expression that will evaluate to either true or false (i.e., with values 1 or 0). When the logical expression evaluates to 0, the program control moves on to the next program construction.

If-elseif-end structure:

```
temp=2;
if temp > 100
 disp('Too hot - equipment mulfuctioning')
elseif temp > 90
 disp('Normal operating temperature')
elseif temp > 50
 disp('Temperature below desired operating range')
else
 disp('Too cold - turn off equipment')
end
```

```
Too cold - turn off equipment
```

(Exercise: Run the above code for *temp*=0, 35, 60, 120.)

LOOPING STRUCTURES

For loops:

```
for i = 1 : 4
```

```
      c = 2*i
end
```

```
c = 2
c = 4
c = 6
c = 8
```

The execution of the calculations will repeat for each index value i = 1, 2 .... 5. The following example shows how looping structures can be nested. The contents of the matrix are created inside a nested for loop:

```
for i=1:3
        for j=1:3
             A(i,j) = i/j;
        end
end
A
```

```
A = 3×3
    1.0000    0.5000    0.3333
    2.0000    1.0000    0.6667
    3.0000    1.5000    1.0000
```

There are actually two loops here, with one nested inside the other; they define

`A(1,1), A(1,2), A(1,3) ... A(1,10), A(2,1), ... A(10,10)`

in that order.

While loop:

Is a structure used for repeating a set of statements as long as a specified condition is true. For example,

```
a=0;
while(a<10)
   a=a+3
end
```

```
a = 3
a = 6
a = 9
a = 12
```

GRAPHICS

Two- and three-dimensional Matlab graphs can be given titles, have their axes labeled, and have text placed within the graph.

The *`plot`* command creates linear x-y plots; if `x` and `y` are vectors of the same length, the command *`plot(x,y)`* opens a graphics window and draws an x-y plot of the elements of *`x`* versus the elements of *`y`*.

Example: Draw the graph of the sine function over the interval -4 to 4 with the following commands:

```
x1 = -4:0.1:4;
y5 = sin(x1);
plot(x1,y5,'-o'); grid;
xlabel('x');
```

```
ylabel('sin(x)');
title('Sine function')
```

The vector x is a partition of the domain with increment 0.01 while y is a vector giving the values of sine at the nodes of this partition.

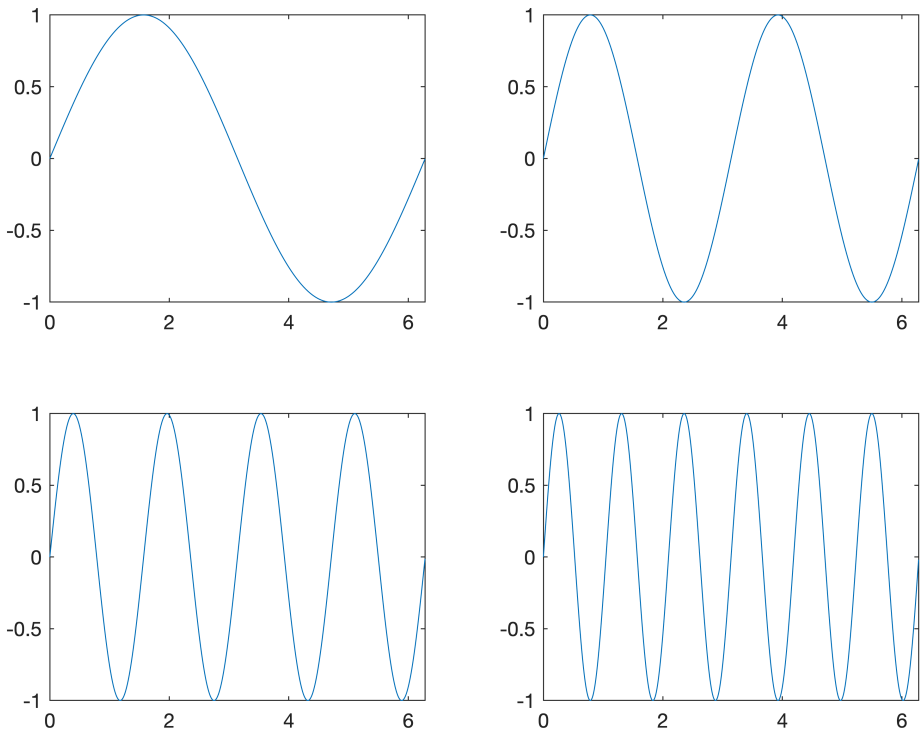
Example: Draw multiple plots on a single graph:

```
x = 0:.01:2*pi;
y1 = sin(x);
y2 = sin(2*x);
y3 = sin(4*x);
y4 = sin(6*x);
plot(x,y1,x,y2,x,y3);grid;hold on;
plot(x,y4);hold off
```

subplot splits the graph window into multiple portions, in order to show several plots at the same time. The statement *subplot(mnp)* breaks the graph window into an *m-by-n* box and uses the *pth* box for the subsequent plot.

Example: Divide the graph window into four partitions and plot separately y_1, y_2, y_3, y_4 over x.

```
figure(3),subplot(221),plot(x,y1),
subplot(222),plot(x,y2),
subplot(223),plot(x,y3),
subplot(224),plot(x,y4);
```



FUNCTIONS

A function is a piece of code that accepts an input argument from the user and provides output to the program. Functions allow us to program efficiently because we do not need to repeat code for calculations that are performed frequently. User defined functions usually are written in M-files.

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

For example:

```
output(3)
```

```
ans = 6
```

(Exercise: Type *output(3)*. What is the output of the function? Try it with different inputs. Try *output(3)+5*)

The first line declares the function name, input arguments, and output arguments. Notice that the file name corresponds to the function name.

A function may have multiple input arguments or multiple output arguments or both. For example:

```
time=7;
[distance, velocity, acceleration] = motion(time)
```

```
distance = 171.5000
velocity = 24.5000
acceleration = 3.5000
```

Exercise 1:

Create a linear $X - Y$ plot for the following variables:

$t = 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 12$

$y = 0,\ 0.5,\ 1,\ 2,\ 4,\ 7,\ 11,\ 14,\ 15.5,\ 16,\ 16,\ 16,\ 16$

Exercise 2:

Find a polynomial of degree 3 to fit the following data:

$x = 0,\ 1,\ 2,\ 4,\ 6,\ 10$

$y = 1,\ 7,\ 23,\ 109,\ 307,\ 1231$

```
function f = output(x)
% This function adds 3 to the input
f=x+3;
end

function [dist,vel,accel]=motion(t)
% Distance, velocity and acceleration of a vehicle
accel=0.5*t;
vel=accel*t;
```

```
dist=vel*t;
end
```