Signal Processing using Matlab

Lesson 1

Overview of Matlab Syntax and Programming

1.1 Introduction

In this lesson, we will take an overview of the Matlab language syntax and programming. In the first section, we will take an overview of Matlab syntax. We will start by explaining how we could define arrays in Matlab. Afterwards, we will study Matlab's basic numeric operations (arithmetic, relational, and logical operations). We will then proceed to Matlab's basic array operations (concatenation, indexing, and transposing). We will study an important notation called the colon notation. In the last section of this lesson, we will study some important Matlab functions that will be used in the following lessons. In the second section, we will discuss M-files and programming statements.

Section I: Syntax Basics

1.2 Defining Arrays

The flexibility of Matlab in solving engineering problems comes from its ability to operate with arrays in an easier way than the other programming languages. The first step in learning Matlab is to know how to define arrays. In (1.1), we introduce some new terminology related to Matlab arrays. In (1.2), we learn how to define arrays.

1.2.1 Scalars, Vectors, Arrays and Dimensioning

An array is a data unit that consists of many elements. A 4-by-3 array of numbers is a data unit consisting of 4 rows by 3 columns of numbers. Another name for an array is a matrix. A special case of an array is the vector. A vector is a 1-by-N or an N-by-1 array, i.e. it is an array with only one row or only one column. Another special case of an array is the scalar. A scalar is an array with only one element. Its dimensions are said to be 1-by-1. The following example shows an array A, a row vector VR, a column vector VC and a scalar S.

$$A = \begin{bmatrix} 4 & 2 & 3 & 7 \\ 5 & 8 & 8 & 1 \\ 2 & 1 & 7 & 5 \\ 9 & 3 & 1 & 2 \end{bmatrix} \qquad VR = \begin{bmatrix} 3 & 0 & -1 \end{bmatrix} \qquad VC = \begin{bmatrix} 2 \\ 6 \\ 3 \end{bmatrix} \qquad S = 2.7$$

1.2.2 Defining Arrays, Vectors and Scalars

Defining arrays in Matlab is a general technique out of which we can derive the special cases of defining vectors and scalars. There are four rules for defining arrays in Matlab:

1. Elements on the same row must be separated by white-spaces or commas.
2. A semicolon, wherever it appears, causes a transition to the next row of elements.
3. The definition requires the use of square brackets " [ ] ".
4. The defined array must be rectangular, i.e. the number of elements in all rows must be the same.

The instructions that create the variables in the previous example are as follows:

```
>>A=[4 2 3 7 ; 5 8 8 1 ; 2 1 7 5 ; 9 3 1 2]
>>VR=[3 0 -1]
>>VC=[2 ; 6 ; 3]
>>S=[2.7] %(or >>S=2.7)
```

Note that scalars can be defined without square brackets. This is an exception to the rule to facilitate working with scalars because they occur very frequently.

1.3 Arithmetic, Relational and Logical Operations

1.3.1 Arithmetic Operations

The following table lists the available arithmetic operators, and a brief description of what each of these operators does.

	Example	***Description***
+	A+B	Adds B to A element by element
-	A-B	Subtracts B from A element by element
.*	A.*B	Multiplies A by B element by element
*****	A*B	Matrix multiplies A by B
./	A./B	Divides A by B element by element
/	A/B	Matrix multiplies A by the inverse of B
.^	A.^3	Cubes A element by element, equivalent to A.*A.*A
^	A^3	Equivalent to matrix multiplying A by A by A (A*A*A)

Here are some examples that operate on arrays A, B, C and D shown below.

$$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 5 & 10\end{bmatrix} \qquad B=\begin{bmatrix}3 & 3\\ 3 & 3\\ 3 & 3\end{bmatrix} \qquad C=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad D=\begin{bmatrix}2 & 3\\ 2 & 3\\ 2 & 3\end{bmatrix}$$

```
>> X=A+C

X =

    2    3    4
    5    6    7
    8    6   11
```

```
>> X=B.*D

X =

    6    9
    6    9
    6    9
```

```
>> X=B./D

X =

    1.5000    1.0000
    1.5000    1.0000
    1.5000    1.0000
```

```
>> X=A^2

X =

   30   27   45
   66   63  102
   97   89  151
```

```
>> X=C-A

X =

    0   -1   -2
   -3   -4   -5
   -6   -4   -9
```

```
>> X=A*C

X =

    6    6    6
   15   15   15
   22   22   22
```

```
>> X=C/A

X =

   -0.333    0.333         0
   -0.333    0.333         0
   -0.333    0.333         0
```

```
>> X=A.^2

X =

    1    4    9
   16   25   36
   49   25  100
```

1.3.2 Relational Tests

The following table lists the available relational operators, and a brief description of what each of these operators does.

	Example	***Description***
>	A>B	Tests if the elements of A are greater than the elements of B
>=	A>=B	Tests if the elements of A are greater than or equal to the elements of B
<	A<B	Tests if the elements of A are smaller than the elements of B
<=	A<=B	Tests if the elements of A are smaller than or equal to the elements of B
==	A==B	Tests if the elements of A are equal to the elements of B
~=	A~=B	Tests if the elements of A are not equal to the elements of B

Here are some examples that operate on arrays A, B, C and D shown below.

$$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 5 & 10\end{bmatrix} \qquad B=\begin{bmatrix}3 & 3\\ 3 & 3\\ 3 & 3\end{bmatrix} \qquad C=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad D=\begin{bmatrix}2 & 3\\ 2 & 3\\ 2 & 3\end{bmatrix}$$

```
>> X=A>C

X =

     0     1     1
     1     1     1
     1     1     1

>> X=A>=C

X =

     1     1     1
     1     1     1
     1     1     1
```

```
>> X=B<D

   X =

        0     0
        0     0
        0     0

   >> X=B<=D

   X =

        0     1
        0     1
     0     1
```

```
>> X=B==D

X =

     0     1
     0     1
     0     1

>> X=B~=D

X =

     1     0
     1     0
     1     0
```

Note that relational operations are element-by-element operations.

1.3.3 Logical Operations

The following table lists the available logical operators, and a brief description of what each of these operators does.

	Example	***Description***
&	A&B	Performs logical ANDing of elements of A and B
|	A|B	Performs logical ORing of elements of A and B

~	~A	Complements the logic of elements of A

Here are some examples that operate on arrays A and B shown below.

$$A=\begin{bmatrix}1 & 2 & 3\\ 0 & 5 & 6\\ 7 & 0 & 10\end{bmatrix} \qquad B=\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}$$

```
>> X=A&B

X =

     0     1     0
     0     0     1
     0     0     0
```

```
>> X=A|B

X =

     1     1     1
     1     1     1
     1     1     1
```

```
>> X=~A

X =

     0     0     0
     1     0     0
     0     1     0
```

Note that Matlab treats numbers from the logical point of view as zero and non-zero. A non-zero number is equivalent to logic 1. Also note that logical operations are element-by-element operations.

1.3.4 Size Rules

The following table shows the arithmetic, relational and logical operations and the size rule that must be satisfied by their operands.

Operations	Size Rule
Addition and Subtraction	All operands must have the same size.
Array Multiplication, Division and Power	All operands must have the same size.
Matrix Multiplication	Inner matrix dimensions must agree.
Matrix Division and Power	Matrices must be square.
All Relational Operations	All operands must have the same size.
All Logical Operations	All operands must have the same size.

1.3.5 Scalar Expansion

Matlab has an exception to the size rule of array operations when it comes to operations involving scalars. An operation involving an array and a scalar will cause Matlab to expand the scalar to the size of the array. Here are examples that operate on the array A shown below.

$$A=\begin{bmatrix}1 & 2 & 3\\ 0 & 5 & 6\\ 7 & 0 & 10\end{bmatrix}$$

```
>> X=A-7

X =

    -6    -5    -4
    -7    -2    -1
     0    -7     3
```

```
>> X=2*A

X =

     2     4     6
     0    10    12
    14     0    20
```

```
>> X=A>=5

X =

     0     0     0
     0     1     1
     1     0     1
```

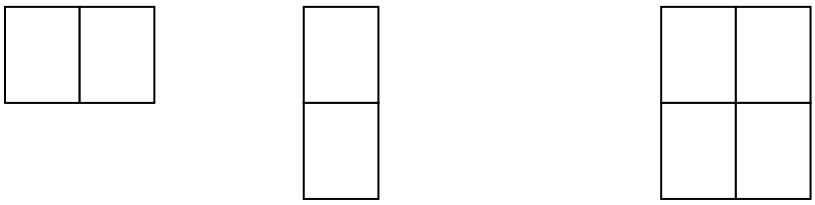
1.3.6 Combining Arithmetic, Relational and Logical Operations

Part of the flexibility of Matlab comes from the ability of combining expressions. The following expression will execute successfully provided that A and B have the same size.

```
>> C=(2*A-B)*~((2*A-B)>=B)
```

1.4 Concatenation

To concatenate two pieces of paper is to join them end-to-end. You can concatenate two pieces of paper horizontally or vertically. You can also concatenate more than two pieces of paper as shown below.



1.4.1 Concatenation Syntax

You can concatenate arrays in Matlab just as you concatenate pieces of paper. Here are the instructions to carry out horizontal and vertical concatenation. Note that the syntax of concatenation is similar to the syntax of array creation. This is because array creation is actually a concatenation of single numbers.

$$A=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \qquad B=\begin{bmatrix}2 & 2\\ 2 & 2\end{bmatrix}$$

```
>> C=[A B]

C =

     1     1     2     2
     1     1     2     2

>> C=[B;A]

C =

     2     2
     2     2
     1     1
     1     1

>> C=[A B;B A]

C =

     1     1     2     2
     1     1     2     2
     2     2     1     1
     2     2     1     1
```

1.4.2 Concatenation Size Rules

Since any array must be rectangular, there are size rules that govern concatenation. Arrays concatenated horizontally must have the same number of rows, and arrays concatenated vertically must have the same number of columns.

1.5 Indexing into Vectors

"Indexing into an array" means dealing with part of the array, either reading from this part or writing to it. The easiest arr ay we can start learning how to index into is the vector. We will first learn how to read single and multiple elements from vectors, then we will learn how to write single or multiple elements into vectors.

1.5.1 Single Element Indexing into Vectors on the Right Hand Side

Suppose we define the vector V as shown below. We then want to read the third element in V. The instruction V(3) obtains the third element in V. The number between round brackets is called the "index". The index may take any integer value from 1 to the length of V. The index may also be end, which corresponds to the last element in the vector.

```
>> V=[7 9 6 2]

V =

     7     9     6     2

>> A=V(3)

A =

     6

>> A=V(end)

A =

     2
```

1.5.2 Multiple Element Indexing into Vectors on the Right Hand Side

If the index is a vector instead of a scalar, Matlab returns multiple elements from the vector. The following instructions provide examples that index into the same vector V used in (4.1).

```
>> A=V([1 3])

A =

     7     6

>> B=V([end-2 end])

B =

     9     2
```

Note that the length of the result is the same as the length of the index.

1.5.3 Single Element Indexing into Vectors on the Left Hand Side

We can use indexing on the left hand side to store a value in a specific element of a vecto r. Look at the following example.

```
>> V=[7 9 6 2]

V =

     7     9     6     2

>> V(end)=4

V =

     7     9     6     4
```

1.5.4 Multiple Element Indexing into Vectors on the Left Hand Side

We can also index on the left hand side using a vector index to store a vector of values in specific locations in a vector. Look at the following example:

```
>> V=[7 9 6 2]


V =


     7     9     6     2


>> V([1 end-1 end])=[4 1 2]


V =


     4     9     1     2
```

1.6 Indexing into 2-D Arrays

We have learnt how to ind ex into vectors. Vectors are 1 -D arrays where only one index per element is needed. In 2 -D arrays, we will need a row index and a column index to specify the position of an element.

1.6.1 Single Element Indexing into 2-D Arrays on the Right Hand Side

The general syntax for indexing into a 2 -D array is A(R,C) where A is the array name, R is the row index and C is the column index. If R and C are scalars, Matlab returns a single element from A.

```
>> A=magic(3)

A =

     8     1     6
     3     5     7
     4     9     2
```

```
>> A(3,1)

ans =

     4
```

1.6.2 Multiple Element Indexing into 2 -D Arrays on the Right Hand Side

If R and C are vectors rather than scalars, Matlab returns multiple elements according to the rows and columns listed in the vector indices.

```
>> A=magic(3)

A =

     8     1     6
     3     5     7
     4     9     2

>> A([1 2],1)

ans =

     8
     3

>> A([1 3],[1 2])

ans =

     8     1
     4     9
```

The colon (:) may be used as a special index meaning "all". If it appears as a row index it means all rows. If it appears as a column index it means all columns.

```
>> A([1 end],:)

ans =

     8     1     6
     4     9     2
```

1.6.3 Single Element Indexing into 2-D Arrays on the Left Hand Side

We can use indexing on the left hand side to write a value into an element in an array specificied by its row and column indices. Here is an example.

```
>> A=magic(3)

A =

     8     1     6
     3     5     7
     4     9     2
```

```
>> A(end,2)=0

A =

     8     1     6
     3     5     7
     4     0     2
```

1.6.4 Multiple Element Indexing into 2-D Arrays on the Left Hand Side

```
>> A=magic(3)

A =

     8     1     6
     3     5     7
     4     9     2

>> A(1,:)=[4 4 4]

A =

     4     4     4
     3     5     7
     4     9     2

>> A([2 3],:)=0

A =

     4     4     4
     0     0     0
     0     0     0
```

The instruction A(1,:)=[4 4 4] demonstrates a size rule. The right hand side must have the same dimensions as the left hand side specification. The instruction A([2 3],:)=0 demonstrates the exception from this rule, which we called scalar expansion. Matlab understands that you want to write zeros in all elements of the second and third rows of A.

1.7 Transposing Arrays

The transpose operation turns the array such that its rows become columns and its columns become rows. Here is an example.

```
>> A=magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> A=A'
```

```
A =

    16     5     9     4
     2    11     7    14
     3    10     6    15
    13     8    12     1
```

1.8 The Colon Notation

The colon (:) has two uses in Matlab. We looked at its first use as an index meaning "all". Its second application is in a special notation called the colon notation. The colon notation is used to generate sequences of numbers given a start number, a step and a stopping limit. Look at the following examples.

```
>> A=4:2:12

A =

     4     6     8    10    12

>> A=8:-3:-5

A =

     8     5     2    -1    -4
```

If the step is equal to 1, it may be omitted from the colon notation, so that the notation appears as follows.

```
>> A=120:130

A =

   120   121   122   123   124   125   126   127   128   129   130
```

The colon notation produces a row vector by default. If you want a column vector, you will have to transpose the result.

1.9 Some Important Matlab Functions

The "zeros" and "ones" Functions

These two functions are responsible for generating arrays of zeros or ones. The syntax of these functions is as follows:

```
>>A=zeros(M,N);
>>B=5*ones(M,N);
```

The first instruction genera tes an array of zeros of dimensions M ×N. The second instruction generates an array of dimensions M×N full of fives based on the "ones" function.

The "length" Function

The "length" function returns the length (number of elements) of a vector. Let V be a vec tor containing 4 elements. The following instruction would return 4.

```
>>L=length(V);
```

The "find" Function

The "find" function returns the indices of non -zero elements in its input array. This function is used mainly to search for elements of an array that sa tisfy a certain condition. To understand, consider the following instruction.

```
>>I=find(A>=2);
```

The input to the "find" function is the array resulting from the relational test (A>=2). This array has the same size of A, but consists entirely of ones and zero s. Ones appear in place of the elements of A that are greater than or equal to 2. Thus, in effect, the find function will return the indices of the elements of A that are greater than or equal to 2. This is a very important function in signal processing.

Section II: Programming

1.11 M-Files

M-files are files in which we can write and save Matlab code to be executed later. We can open a new M -file by writing "edit" in the command prompt, or by clicking on the "New" button in the Matlab button toolbar. You wr ite the instructions in the M -file, and can then run the M -file by pressing the "Run" button in the M -file editor's toolbar, or by writing the M -file's name in the command window.

1.12 Programming Statements

In this section, we will discuss the four differ ent statements used in programming: the "for" statement, the "while" statement, the 'if" statement, and the "switch" statement. The syntax for each statement is demonstrated below.

The "for" statement

```
for i=10:-2:5
    Code …
end
```

The "while" statement

```
while (a>2)&(b<7)
    Code …
end
```

The "if statement

```
if a<0
    Code 1 …
elseif a>2
    Code 2 …
else
    Code 3 …
end

if b<c
    Code …
end
```

```
if d<e
     Code 1 …
else
     Code …
end
```

The "switch" statement

```
switch x
     case 0
          Code 1 …
     case 3
          Code 2 …
end

switch y+z
     case 1
          Code 1 …
     case 2
          Code 2 …
     otherwise
          Code 3 …
end
```

1.13 Exercises

1- The vector S=[150 150 150 160] contains the weekly salaries of 4 employees in June 2004. In June 2005, the salaries were raised by 10 units. In June 2006, the salaries were raised by 10%. Right down a single Matlab instruction to generate the vector of salaries in June 2006.

2- The vector V is given by V=[2 8 7 3 1 0 8 9]. Write down max 2 Matlab instructions to produce a vector that contains 1 in the place of the odd numbers and -1 in the place of the even numbers.

3- The vector V is a column vector. For each of the following operations, write down a single Matlab instruction that carries out the operation:

a) Add 2 to the last 3 elements of V

b)Reverse the order of the last 4 elements of V.

c) Add the elements number 1, 3, 5 …etc to the elements number 2, 4, 6 … etc, and store the results in the place of the later elements.

4- Write down a single Matlab instruction that generates the following sequence: 1 4 9 16 … 64,81 … 9 4 1.

5- An array M is defined by

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -1 & -2 & -3 & -4 \\ 1 & 2 & 3 & 4 \\ -1 & -2 & -3 & -4 \end{bmatrix}$$

Use the appropriate indexing techniques to:

a) Reflect array (M) left-side right,

b) Reflect array (M) upside down,
c) Swap columns 2 and 3 of array (M),
d) Swap rows 1 and 4 of array (M),
e) Shuffle the rows of (M) from [1 2 3 4] to [1 3 4 2] and shuffle the columns of (M) from [1 2 3 4] to [3 2 4 1].

6- Generate the following Matrix: $x = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 2 & 0 & 0 & 0 & -2 \\ 3 & 0 & 0 & 0 & -3 \\ 4 & 0 & 0 & 0 & -4 \\ 5 & 0 & 0 & 0 & -5 \end{bmatrix}$

From Matrix X, generate the following matrices:

$$y = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & -3 & -4 & -5 \end{bmatrix} \qquad z = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & -3 & -2 & -1 \end{bmatrix}$$

$$w = \begin{bmatrix} 2 & 100 & 100 & 100 & 0.1 \\ 4 & 100 & 100 & 100 & 0.2 \\ 6 & 100 & 100 & 100 & 0.3 \\ 8 & 100 & 100 & 100 & 0.4 \\ 10 & 100 & 100 & 100 & 0.5 \end{bmatrix}$$

7-It is required to solve the following system of equations

$$\begin{aligned} 2x_1 + 3x_2 + 5x_3 + 6x_4 + 21x_5 &= 152 \\ 5x_1 + 2x_3 + 2x_4 &= 19 \\ 6x_1 + 7x_2 + 8x_3 + 9x_4 + 11x_5 &= 135 \\ 13x_2 + 17x_3 + 5x_4 + 6x_5 &= 127 \\ x_1 + 4x_2 + 3x_4 + 9x_5 &= 66 \end{aligned}$$

a) Use the 'zeros' function to create an empty 5 -by-5 matrix (A) for the coefficients and an empty 5 -by-1 column vector (B) for the RHSs of the equations.
b) Use the array editor to populate the coefficients into matrix (A).
c) Use the command prompt to populate the RHSs into vector (B).
d) Explain how we can use the 'rank' function to find out whether or not the 5 equations are "independent" enough to a ssign unique solutions for the 5 unknowns.
e) Write down an expression for a variable S that is true if the rank of (A) is equal to the number of variables in the case above and false otherwise.
f) Solve the set of linear equations for x_1 through x_5.

2) Use the Matlab help system to find the names of the following:

a) The exponential function (e^x),
b) A function that returns the natural logarithm (ln) of a number,
c) Functions that return the base-2 and base-10 logarithms of a number,
d) A function that gets the square root of a number.
e) "Sound" & "Image" commands

Matlab Instructions for Lesson 1

1	a=1
2	a=[1 2 3 4]
3	a=[1 2;3 4]
comments	
4	b=[1 4;7 6]
5	c=a+b
6	c=a+3
7	c=a-b
8	c=a-3
9	c=a*b
10	c=a.*b
11	c=a*3
12	c=a./b
13	c=a/3
comments	
14	c=a==b
15	c=a==3
16	c=a>b
17	c=a<b
18	c=a~=b
comments	
19	c=a~=3
20	c=(a==b)&(a==3)
21	c=(a==b)|(a==3)
22	c=~((a==b)|(a==3))

comments	
23	`c=[a b]`
24	`c=[a;b]`
25	`c=[a b;b a]`
comments	
26	`c=a(1,1)`
27	`c=a(2,2)`
28	`c=a([1 2],1)`
29	`c=a(2,[1 2])`
30	`c=a(end,end)`
31	`c=a(end-1,end-1)`
32	`c=a(end,:)`
33	`c=a([end end-1],:)`
comments	
34	`d=1:10`
35	`d=10:-1:1`
36	`d=0:1/100:0.1`
comments	

37	`d=d'`
comments	
38	`a=10*rand(1,16)`
39	`b=a(1:4)`
40	`b=a(4:-1:1)`
41	`b=a(4:2:end-2)'`
comments	
42	`L1=length(a)`
43	`L2=length(b)`
comments	
44	`I1=find(a>3)`
45	`I2=find((a>7)|(a<2))`
comments	
46	`N1=length(I1)`
47	`N2=length(find((a>7)|(a<2)))`
comments	