Digital Signal Processing using Matlab

session 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 discuss M-files and programming statements.

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.

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.

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

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
```

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.

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

```
>> X=A+C

X =

    2    3    4
    5    6    7
    8    6   11

>> X=C-A

X =

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

```
>> X=B.*D

X =

    6    9
    6    9
    6    9

>> X=A*C

X =

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

```
>> X=B./D

X =

    1.5000    1.0000
    1.5000    1.0000
    1.5000    1.0000

>> X=C/A

X =

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

```
>> X=A^2

X =

   30   27   45
   66   63  102
   97   89  151

>> 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.

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

```
>> 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.

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

```
>> 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.

1.3.5 Scalar Expansion

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

```
>> 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
```

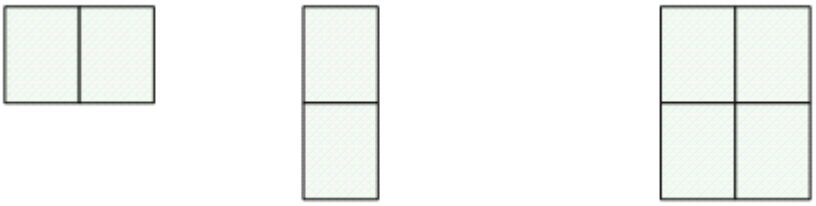
```
>> X=A>=5

X =

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

1.4 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



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

```
>> 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
```

1.5 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 array 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
```

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

A =

     7     6

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

B =

     9     2
```

1.5.2 Single Element Indexing into Vectors on the Left Hand Side

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

V =

     7     9     6     2

>> V(end)=4

V =

     7     9     6     4
```

```
>> 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 index 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
```

```
>> 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
```

```
>> 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
```

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=A'
```

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
```

```
>> A=120:130

A =

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

1.9 Some Important Matlab Functions

>>A=zeros(M,N);

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

>>L=length(V);

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

II: Programming

1.10 Programming Statements

In this section, we will discuss the four different 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
```

1.11 Exercises

1- Generate the following sequence: V=1 4 9 16 25 …… 16 9 4 1

Where V is a sequence of length of 49 elements.

2- For the previous generated sequence V:

a) Add 2 to the last 3 elements.

b) Reverse the order of the last 10 elements.

c) For the first 48 elements, add the elements in the even places to that in the odd places and store the output in the odd places.

3-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]
f) shuffle the columns of (M) from [1 2 3 4] to [3 2 4 1].