AYDIN
ADNAN MENDERES UNIVERSITY
FACULTY OF ENGINEERING
Department of Electrical and Electronics Engineering

EE214 – Computer Aided Engineering Applications

Prof. Dr. Yılmaz KALKAN

EE214 – Computer Aided Engineering Applications

Lecture 2

VECTOR AND MATRIX REPRESENTATION

Objective

Introduce MATLAB syntax in the context of vectors and matrices and their manipulation.

Topics

- Definitions of Matrices and Vectors
- Creation of Vectors
- Creation of Matrices
- Dot Operations

Definitions of Matrices and Vectors

An array *A* of *m* rows and *n* columns is called a matrix of order (*m*×*n*) and is written as

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & \\ \dots & & \dots & \\ a_{m1} & & & a_{mn} \end{bmatrix} \rightarrow (m \times n)$$

The elements of the matrix are denoted a_{ij}, where *i* indicates the row number and *j* the column number.

Square Matrix: $n = m$

Diagonal Matrix: When $n = m$ and $a_{ij} = 0$, $i \neq j$; thus

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & & \\ \dots & & \dots & \\ 0 & & & a_{nn} \end{bmatrix} \rightarrow (n \times n)$$

```
>> diag([1 2 3])

ans =

     1     0     0
     0     2     0
     0     0     3

>>
```

```
>> diag([1 2 3],1)

ans =

     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
```

```
>> diag([1 2 3],2)

ans =

     0     0     1     0     0
     0     0     0     2     0
     0     0     0     0     3
     0     0     0     0     0
     0     0     0     0     0
```

Identity Matrix: Diagonal matrix with $a_{ii} = 1$; thus

$$I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & & \\ \dots & & \dots & \\ 0 & & & 1 \end{bmatrix}$$

```
>> x =[1 1 1]

x =

     1     1     1

>> diag(x)

ans =

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

```
>> ones(2)

ans =

     1     1
     1     1
```

```
>> ones(3)

ans =

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

```
>> zeros(2,3)

ans =

     0     0     0
     0     0     0
```

Column Matrix - Vector:

When $a_{ij} = a_{i1}$ (there is only one column), then

$$a = \begin{bmatrix} a_{11} \\ a_{21} \\ \dots \\ a_{m1} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_m \end{bmatrix} \rightarrow (m \times 1)$$

Row Matrix - Vector:

When $a_{ij} = a_{1j}$, (there is only one row), then

$$a = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \rightarrow (1 \times n)$$

Default definition of a vector in MATLAB

Row Matrix - Vector:

```
>> a = [1 2 3 4]
a =
     1     2     3     4

>> a = [1,2,3,4]
a =
     1     2     3     4
```

Column Matrix - Vector:

```
>> a = [1; 2; 3; 4]

a =

     1
     2
     3
     4
```

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

b =

     1     2
     3     4
     5     6
```

```
>> b = [-1 2 -3; 4 -5 6; -7 8 -9]

b =

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

Transpose of a Matrix and a Vector

The transpose of a matrix is denoted by an apostrophe (′). If *A* is defined as before and *W* = *A*′, then

$$W = A' = \begin{bmatrix} w_{11} = a_{11} & w_{12} = a_{21} & \dots & w_{1m} = a_{m1} \\ w_{21} = a_{12} & w_{22} = a_{22} & & \\ \dots & & \dots & \\ w_{n1} = a_{1n} & & & w_{nm} = a_{mn} \end{bmatrix} \rightarrow (n \times m)$$

Recall that

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & \\ \dots & & \dots & \\ a_{m1} & & & a_{mn} \end{bmatrix} \rightarrow (m \times n)$$

For a column vector -

$$\text{If} \quad a = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_m \end{bmatrix} \rightarrow (m \times 1) \qquad \text{then} \qquad a' = \begin{bmatrix} a_1 \; a_2 \; \dots \; a_m \end{bmatrix} \rightarrow (1 \times m)$$

For a row vector -

$$\text{If} \quad a = \begin{bmatrix} a_1 \; a_2 \; \dots \; a_m \end{bmatrix} \rightarrow (1 \times m) \quad \text{then} \quad a' = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \rightarrow (m \times 1)$$

```
>> A = [1 2 3 4]

A =

     1     2     3     4
```

```
>> A'

ans =

     1
     2
     3
     4
```

```
>> A = [1:4]

A =
     1     2     3     4

>> transpose(A)
ans =

     1
     2
     3
     4
```

```
>> B = [1 0 -1 2; 3 4 10 -4;0.5 -5 -1.6 1;]

B =

    1.0000         0   -1.0000    2.0000
    3.0000    4.0000   10.0000   -4.0000
    0.5000   -5.0000   -1.6000    1.0000
```

```
>> B'

ans =

    1.0000    3.0000    0.5000
         0    4.0000   -5.0000
   -1.0000   10.0000   -1.6000
    2.0000   -4.0000    1.0000
```

Creation of Vectors

If *a*, *x*, *b*, … are either variable names, numbers, expressions, or strings, then vectors are expressed as either

***f* = [*a x b* …]** **(Blanks required)**

or

***f* = [*a*, *x*, *b*, …]** **(Blanks optional)**

Caution about blanks: If $a = h + d^s$, then *f* is written as either

***f* = [*h*+*d*^*s x b* ...]**

or

***f* = [*h*+*d*^*s*, *x*, *b*, ...]**

No blanks permitted within expression

Ways to Assign Numerical Values to the Elements of a Vector -

- **Specify the range of the values and the increment between adjacent values – called colon notation.**

  Used when the increment is either important or has been specified.

- **Specify the range of the values and the number of values desired.**

  Used when the number of values is important.

If *s*, *d*, and *f* are any combination of numerical values, variable names, and expressions, then the colon notation that creates a vector x is either

x = s:d:f or x = (s:d:f) or x = [s:d:f]

where

***s* = start or initial value**
***d* = increment or decrement**
***f* = end or final value**

Thus, the following row vector *x* is created

***x* = [*s*, *s*+*d*, *s*+2*d*, …, *s*+*nd*] (Note: *s*+*nd* ≤ *f*)**

```
>> A = [1:10]

A =

    1    2    3    4    5    6    7    8    9   10
```

```
>> A = [1:2:10]

A =

    1    3    5    7    9
```

Also, when *d* is omitted MATLAB assumes that *d* = 1. Then

x = s:f

creates the vector

***x* = [*s*, *s*+1, *s*+2, … , *s*+*n*]**

where *s*+*n* ≤ *f*

The number of terms, called the length of the vector, that this expression has created is determined from

`length`(x)

```
>> A = [1:2:10]
A =
     1     3     5     7     9

>> length(A)
ans =
     5
```

Example –

Create a row vector that goes from 0.2 to 1.0 in increments of 0.1.

```
x = 0.2:0.1:1
n = length(x)
```

When executed, we obtain

```
x =
    0.20  0.30  0.40  0.50  0.60  0.70  0.80  0.90  1.00
n =
     9
```

To create a column vector, we use

```
x = (0.2:0.1:1)'
n = length(x)
```

→

```
x =
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
    0.8000
    0.9000
    1.0000
n =
     9
```

Generation of *n* Equally Spaced Values

x = `linspace`(s, f, n)

where the increment (decrement) is computed by MATLAB from

$$d = \frac{f - s}{n - 1}$$

The values of *s* and *f* can be either positive or negative and either *s* > *f* or *s* < *f*. When *n* is not specified, it is assigned a value of 100. Thus, `linspace` creates the vector

***x* = [*s*, *s*+*d*, *s*+2*d*, …, *f* = *s*+(*n*–1)*d*]**

```
>> x=linspace(1,10,10)

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

```
>> x=linspace(1,10,4)

x =
    1    4    7   10
```

If equal spacing on a logarithmic scale is desired, then

x = `logspace`(s, f, n)

where the initial value is $x_{start} = 10^s$, the final value is $x_{final} = 10^f$ and *d* is defined above.

This expression creates the row vector

$$x = [10^s \; 10^{s+d} \; 10^{s+2d} \; \ldots \; 10^f]$$

$$d = \frac{f - s}{n - 1}$$

Examples -

```
x = linspace(0, 2, 5)
```

gives

```
x =

    0    0.5000    1.0000    1.5000    2.0000
```

and

```
x = logspace(0, 2, 5)   % [10^0, 10^0.5, 10^1, 10^1.5, 10^2]
```

gives

```
x =

    1.0000    3.1623   10.0000   31.6228  100.0000
```

Transpose of a Vector with Complex Values

If a vector has elements that are complex numbers, then taking its transpose also converts the complex quantities to their complex conjugate values.

Consider the following script

```
z ≡ [1, 7+4j, -15.6, 3.5-0.12j];
w ≡ z′
```

which, upon execution, gives

```
w =
   1.0000
   7.0000 - 4.0000i
 -15.6000
   3.5000 + 0.1200i
```

If the transpose is desired, but not the conjugate, then the script is written as

```
z = [1, 7+4j, -15.6, 3.5-0.12j];
w = conj(z')    (or w=z.')
```

which, upon execution, gives

```
w =
   1.0000
   7.0000 + 4.0000i
 -15.6000
   3.5000 - 0.1200i
```

Accessing Elements of Vectors – Subscript Notation

Consider the row vector

$$b = [b_1\ b_2\ b_3\ \ldots\ b_n]$$

Access to one or more elements of this vector is accomplished using subscript notation in which the subscript refers to the location in the vector as follows:

b(1) $\rightarrow b_1$

…

b(k) $\rightarrow b_k$

...

b(n) $\rightarrow b_n$ **% or b(end)**

Example –

```
x =  [-2, 1, 3, 5, 7, 9, 10];
b = x(4)
c = x(6)
xlast = x(end)
```

When executed, we obtain

```
b =
   5
c =
   9
xlast =
  10
```

Accessing Elements of Vectors Using Subscript Colon Notation

The subscript colon notation is a shorthand method of accessing a group of elements.

For a vector *b* with *n* elements, one can access a group of them using the notation

b(k:d:m)

where $1 \leq k < m \leq n$, *k* and *m* are positive integers and, in this case, *d* is a positive integer.

Thus, if one has 10 elements in a vector *b*, the third through seventh elements are selected by using

b(3:7)

Example –

Consider the following script

```
z = [-2, 1, 3, 5, 7, 9, 10];
z(2) = z(2)/2;
z(3:4) = z(3:4)*3-1;
z
```

When executed, we obtain

```
z =
   -2.00   0.50   8.00   14.00   7.00   9.00   10.00
```

Example –

Consider the script

```
y = [-1, 6, 15, -7, 31, 2, -4, -5];
x = y(3:5)
```

whose execution creates the three-element vector

```
x =
    15    -7    31
```

Example –

We shall show that there are several ways to create a vector *x* that is composed of the first two and the last two elements of a vector *y*. Thus,

```
y = [-1, 6, 15, -7, 31, 2, -4, -5];
x = [y(1), y(2), y(7), y(8)]
```

or

```
y = [-1, 6, 15, -7, 31, 2, -4, -5];
index = [1, 2, 7, 8];   % or [1:2, length(y)-1, length(y)]
x = y(index)
```

or more compactly

```
y = [-1, 6, 15, -7, 31, 2, -4, -5];
x = y([1, 2, 7, 8])  % or y([1:2, length(y)-1, length(y)])
```

Two useful functions: `sort` and `find`

`sort`: Sorts a vector *y* in ascending order (most negative to most positive) or descending order (most positive to most negative)

[ynew, indx] = `sort`(y, mode) % mode = 'ascend‘ or 'descend'

where *ynew* is the vector with the rearranged (sorted) elements of *y* and *indx* is a vector containing the original locations of the elements in *y*.

```
>> x = [1 2 -4 0 8 3 5]
x =
     1     2    -4     0     8     3     5
```

```
>> [x_sorted, indx] = sort(x, 'ascend')

x_sorted =
    -4     0     1     2     3     5     8

indx =
     3     4     1     2     6     7     5
```

```
>> [x_sorted, indx] = sort(x, 'descend')

x_sorted =
     8     5     3     2     1     0    -4

indx =
     5     7     6     2     1     4     3
```

Two useful functions: `sort` and `find`

`find`: Determines the locations (not the values) of all the elements in a vector (or matrix) that satisfy a user-specified relational and/or logical condition or expression *Expr*

indxx = `find`(Expr)

```
>> x = [1 2 -4 0 8 3 5]
x =
     1     2    -4     0     8     3     5
```

```
>> find(x==1)

ans =

     1
```

```
>> find(x==0)

ans =

     4
```

Example –

```
y = [-1, 6, 15, -7, 31, 2, -4, -5];
z = [10, 20, 30, 40, 50, 60, 70, 80];
[ynew, indx] = sort(y, 'ascend')
znew = z(indx)
```

when executed, gives

```
ynew =
   -7   -5   -4   -1   2   6   15   31
indx =
    4   8   7   1   6   2   3   5
znew =
   40   80   70   10   60   20   30   50
```

Example –

```
y = [-1, 6, 15, -7, 31, 2, -4, -5];
indxx = find(y<=0)
s = y(indxx)
```

Upon execution, we obtain

```
indxx =
     1    4    7    8
s =
    -1   -7   -4   -5
```

One of the great advantages of MATLAB's implicit vector and matrix notation is that it provides the user with a compact way of performing a series of operations on an array of values.

Example –

The script

```
x = linspace(-pi, pi, 10);
y = sin(x)
```

yields the following vector

```
y =
   -0.0000   -0.6428   -0.9848   -0.8660   -0.3420    0.3420
    0.8660    0.9848    0.6428    0.0000
```

Minimum and Maximum Values of a Vector: `min` and `max`

To find the magnitude of the smallest element *xmin* and its location *locmin* in a vector, we use

[xmin, locmin] = `min`(x)

To find the magnitude of the largest element *xmax* and its location *locmax* in a vector, we use

[xmax, locmax] = `max`(x)

Example –

```
x = linspace(-pi, pi, 10);
y = sin(x);
[ymax, kmax] = max(y)
[ymin, kmin] = min(y)
```

Upon execution, we find that

```
ymax =
    0.9848
kmax =
     8
ymin =
   -0.9848
kmin =
     3
```

```
y =
   -0.0000   -0.6428   -0.9848   -0.8660   -0.3420
    0.3420    0.8660    0.9848    0.6428    0.0000
```

Example - Analysis of the elements of a vector

Consider 50 equally-spaced points of the following function

$$f(t) = \sin(t) \qquad 0 \le t \le 2\pi$$

We shall

(a) determine the time at which the minimum positive value of *f*(*t*) occurs

(b) the average value of its negative values

The script is:

```
t = linspace(0, 2*pi, 50);
f = sin(t);
```

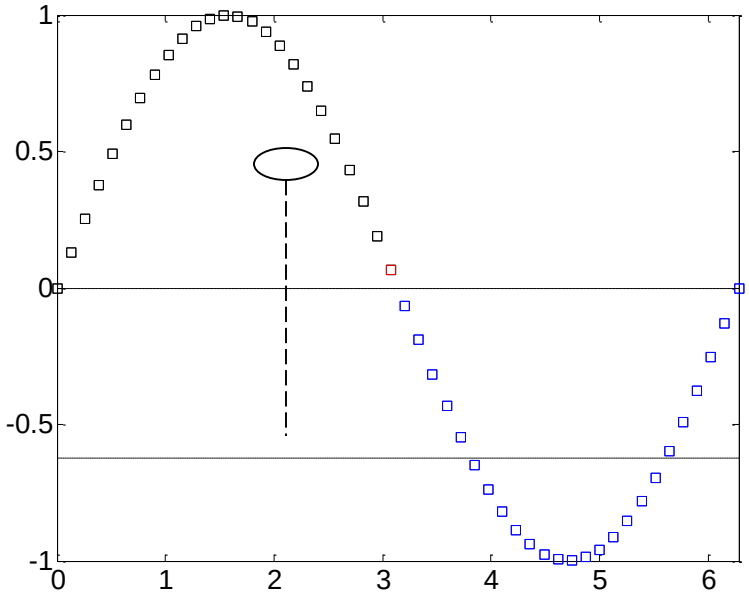
```
f_Avg_Neg = mean(f(find(f<0)))
```

```
Min_Value_f = min(f(find(f>0)));
```

```
t_Min_Value_f = t(find(f==Min_Value_f))
```

Upon execution, we find that

```
f_Avg_Neg =  -0.6237
t_Min_Value_ f = 3.0775
```



Creation of Matrices

The basic syntax to create a matrix is

$$A = [a_{11}\ a_{12}\ \ a_{13};\ a_{21}\ a_{22}\ \ a_{23};\ a_{31}\ a_{32}\ \ a_{33};\ a_{41}\ a_{42}\ \ a_{43}]$$

where the semicolons are used to indicate the end of a row and the a_{ij} can be numbers, variable names, expressions, or strings.

Alternate ways are:

```
A = [a11 a12  a13; ...
     a21 a22  a23; ...
     a31 a32  a33; ...
     a41 a42  a43]
```

where the ellipses (...) are required to indicate that the expression continues on the next line.

For the second alternate way,

One can omit the ellipsis (…) and instead to use the *Enter* key to indicate the end of a row. In this case, the expression will look like

```
A = [a11 a12  a13  %<Enter>
     a21 a22  a23  %<Enter>
     a31 a32  a33  %<Enter>
     a41 a42  a43]
```

For a third alternate way,

Create four separate row vectors, each with the same number of columns, and then combine these vectors to form the matrix as follows

```
v1 = [ a11 a12  a13];
v2 = [ a21 a22  a23];
v3 = [ a31 a32  a33];
v4 = [ a41 a42  a43];
 A = [ v1; v2;  v3; v4]
```

where the semicolons in the first four lines are used to suppress display to the command window.

The order of the matrix is determined by

```
[r, c] = size(A)
```

Transpose of a matrix with complex elements

If a matrix has elements that are complex numbers, then taking its transpose also converts the complex quantities to their complex conjugate values.

Consider the following script

```
Z = [1+2j, 3+4j; 5+6j, 7+9j]
W = Z'
```

the execution of which gives

```
Z =
   1.0000 + 2.0000i   3.0000 + 4.0000i
   5.0000 + 6.0000i   7.0000 + 9.0000i
W =
   1.0000 - 2.0000i   5.0000 - 6.0000i
   3.0000 - 4.0000i   7.0000 - 9.0000i
```

Special Matrices

A matrix of all 1's

`ones`(r, c) → on(1:r,1:c) = 1

```
on = ones(2, 5)
on =
     1     1     1     1     1
     1     1     1     1     1
```

A null matrix

`zeros`(r, c) → zer(1:r,1:c) = 0

```
zer = zeros(3, 2)
zer =
     0     0
     0     0
     0     0
```

Diagonal matrix –

Create an ($n \times n$) diagonal matrix whose diagonal elements are a vector *a* of length *n*

```
diag(a)
```

```
a = [4, 9, 1];
A = diag(a)
A =
     4     0     0
     0     9     0
     0     0     1
```

Extract the diagonal elements of a square matrix *A*

```
diag(A)
```

```
Ad = diag([11, 12, 13, 14; …
           21, 22, 23, 24; …
           31, 32, 33, 34; …
           41, 42, 43, 44])
Ad =
    11
    22
    33
    44
```

Identity matrix whose size is ($n \times n$)

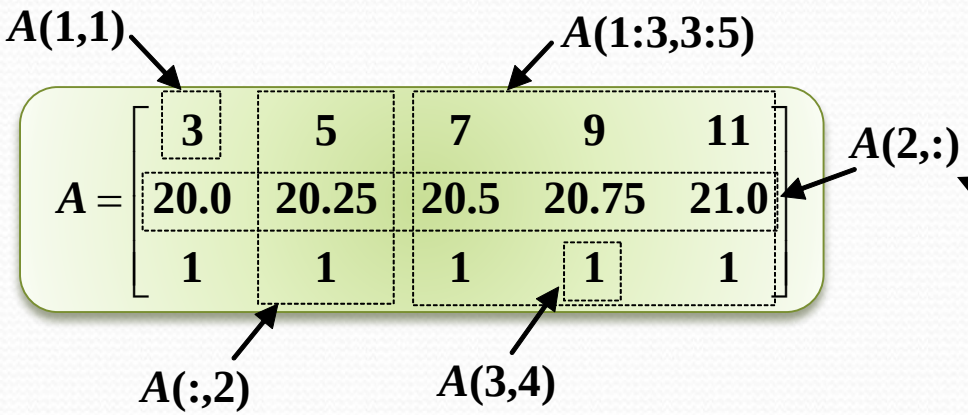
`eye(n)`

```
A = eye(3)
A =
     1     0     0
     0     1     0
     0     0     1
```

Accessing Matrix Elements

Matrix created with –

```
A = [3:2:11; …
     linspace(20, 21, 5); …
     ones(1, 5)];
```



: means all elements of row 2

: means all elements of column 2

```
B = A(1:3,3:5)
B =
    7.0000    9.0000   11.0000
   20.5000   20.7500   21.0000
    1.0000    1.0000    1.0000
```

In several of the following examples, we shall use

`magic(n)`

which creates an ($n \times n$) matrix in which the sum of the elements in each column and the sum of the elements of each row and the sum of the elements in each diagonal are equal.

For example,

`magic(4)`

creates

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

Example –

We shall set all the diagonal elements of `magic`(4) to zero; thus

```
Z = magic(4);
```

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

```
Z = Z-diag(diag(Z))
```

which results in

```
Z=
     0    2    3   13
     5    0   10    8
     9    7    0   12
     4   14   15    0
```

```
diag(Z) =
   16
   11
    6
    1
```

```
diag(diag(Z)) =
    16    0    0    0
     0   11    0    0
     0    0    6    0
     0    0    0    1
```

Example –

We shall replace all the diagonal elements of `magic(4)` with the value 5; thus,

```
Z = magic(4);
```

```
Z = Z-diag(diag(Z))+5*eye(4)
```

which results in

```
Z =
     5     2     3    13
     5     5    10     8
     9     7     5    12
     4    14    15     5
```

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

Use of `find` with Matrices

For a matrix,

[row, col] = `find`(Expr)

where *row* and *col* are column vectors of the locations in the matrix that satisfied the condition represented by *Expr*.

Let us find all the elements of `magic`(3) that are greater than 5. The script is;

Let us find all the elements of `magic`(3) that are greater than 5. The script is;

```
m = magic(3)
[r, c] = find(m > 5);
subscr = [r c]          % Column augmentation
```

Upon execution, we obtain

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

subscr =
     1     1
     3     2
     1     3
     2     3
```

Minimum and Maximum Values in a Matrix

`min` and `max` operate on a column by column basis.

Example –

```
M = magic(4)
minM = min(M)
maxM = max(M)
```

which, upon execution, gives

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

```
minM =
     4     2     3     1
```

```
maxM =
    16    14    15    13
```

To find the maximum of all elements we use `max` twice:

```
M = magic(4);
maxM = max(max(M))
```

which, upon execution, gives

```
maxM =
    16
```

Example - Creation of a special matrix

We shall create the following (9×9) array

$$A = \left[\begin{array}{ccc:ccc:ccc} 0 & 1 & 1 & 0 & 2 & 2 & 0 & 3 & 3 \\ 1 & 0 & 1 & 2 & 0 & 2 & 3 & 0 & 3 \\ 1 & 1 & 0 & 2 & 2 & 0 & 3 & 3 & 0 \\ \hdashline 0 & 4 & 4 & 0 & 5 & 5 & 0 & 6 & 6 \\ 4 & 0 & 4 & 5 & 0 & 5 & 6 & 0 & 6 \\ 4 & 4 & 0 & 5 & 5 & 0 & 6 & 6 & 0 \\ \hdashline 0 & 7 & 7 & 0 & 8 & 8 & 0 & 9 & 9 \\ 7 & 0 & 7 & 8 & 0 & 8 & 9 & 0 & 9 \\ 7 & 7 & 0 & 8 & 8 & 0 & 9 & 9 & 0 \end{array}\right]$$

(Dashed lines have been added to enhance visual clarity.)

The script is ????

```
a = ones(3, 3)-eye(3);
```

```
A = [a, 2*a, 3*a; 4*a, 5*a, 6*a; 7*a, 8*a, 9*a;]
```

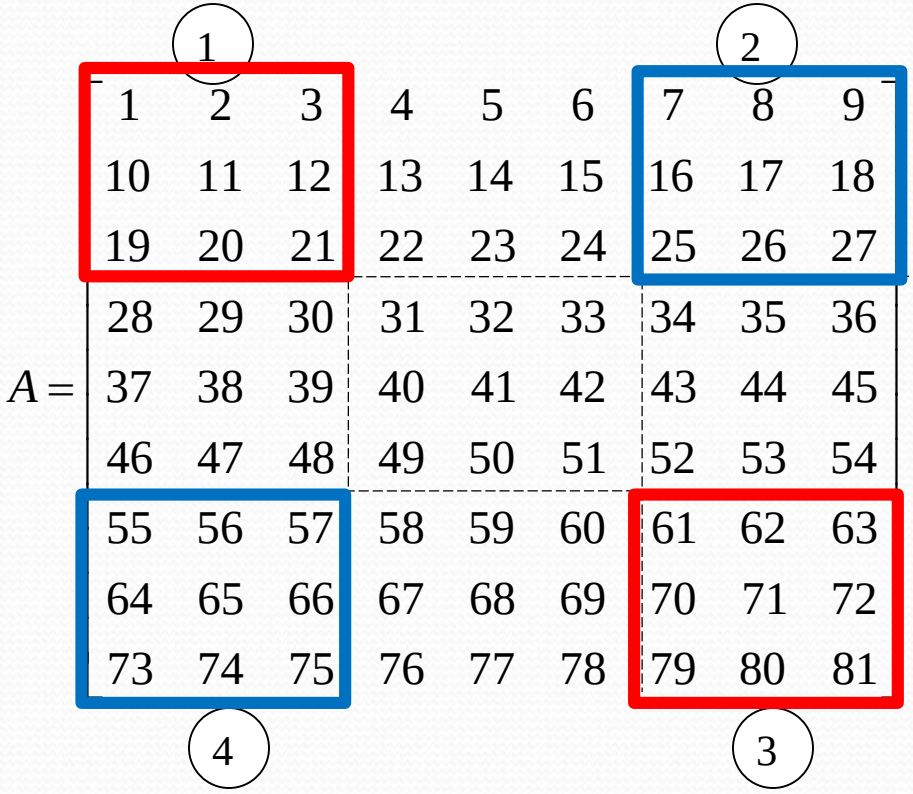
Upon execution, we obtain

A =

0	1	1	0	2	2	0	3	3
1	0	1	2	0	2	3	0	3
1	1	0	2	2	0	3	3	0
0	4	4	0	5	5	0	6	6
4	0	4	5	0	5	6	0	6
4	4	0	5	5	0	6	6	0
0	7	7	0	8	8	0	9	9
7	0	7	8	0	8	9	0	9
7	7	0	8	8	0	9	9	0

Example - Rearrangement of Sub Matrices of a Matrix

Consider the following (9×9) array



(Dashed lines have been added to enhance visual clarity.)

For the four (3×3) sub matrices identified by the circled numbers 1 to 4, we shall perform a series of swaps of these sub matrices to produce the following array

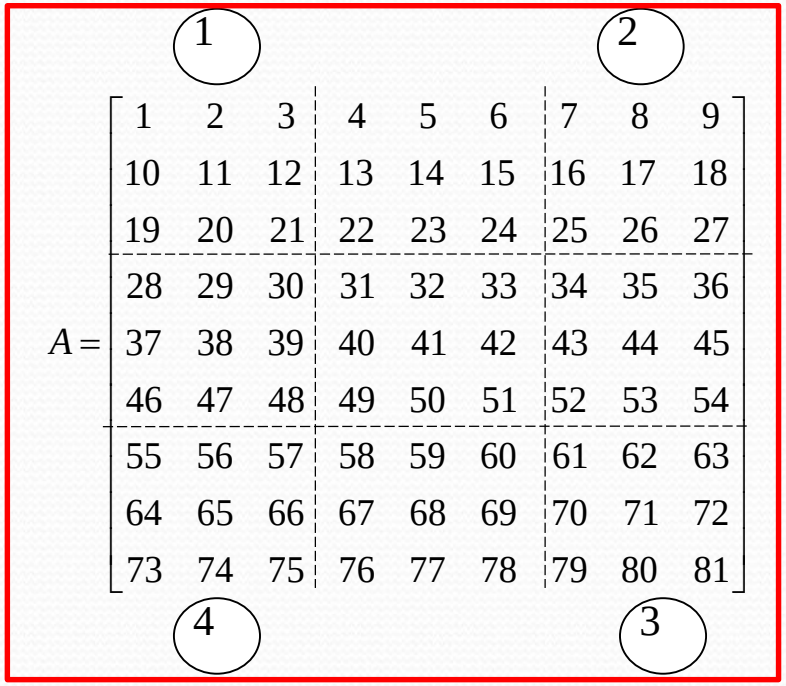
$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} & 3 & & & & & & 4 & \end{array} \\
A = & \left[\begin{array}{ccc:ccc:ccc}
61 & 62 & 63 & 4 & 5 & 6 & 55 & 56 & 57 \\
70 & 71 & 72 & 13 & 14 & 15 & 64 & 65 & 66 \\
79 & 80 & 81 & 22 & 23 & 24 & 73 & 74 & 75 \\
\hdashline
28 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 \\
37 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 \\
46 & 47 & 48 & 49 & 50 & 51 & 52 & 53 & 54 \\
\hdashline
7 & 8 & 9 & 58 & 59 & 60 & 1 & 2 & 3 \\
16 & 17 & 18 & 67 & 68 & 69 & 10 & 11 & 12 \\
25 & 26 & 27 & 76 & 77 & 78 & 19 & 20 & 21
\end{array}\right] \\
 & \begin{array}{ccccccccc} & 2 & & & & & & 1 & \end{array}
\end{array}
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} & 1 & & & & & & 2 & \end{array} \\
A = & \left[\begin{array}{ccc:ccc:ccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\
19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 \\
\hdashline
28 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 \\
37 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 \\
46 & 47 & 48 & 49 & 50 & 51 & 52 & 53 & 54 \\
\hdashline
55 & 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \\
64 & 65 & 66 & 67 & 68 & 69 & 70 & 71 & 72 \\
73 & 74 & 75 & 76 & 77 & 78 & 79 & 80 & 81
\end{array}\right] \\
 & \begin{array}{ccccccccc} & 4 & & & & & & 3 & \end{array}
\end{array}
$$

The script is;

```
Matr = [1:9; 10:18; 19:27; 28:36; 37:45; ...
        46:54; 55:63; 64:72; 73:81];

Temp1 = Matr(1:3, 1:3);

Matr(1:3, 1:3) = Matr(7:9, 7:9);

Matr(7:9, 7:9) = Temp1;
```

```
Temp1 = Matr(1:3, 7:9);

Matr(1:3, 7:9) = Matr(7:9, 1:3);

Matr(7:9, 1:3) = Temp1;

Matr
```

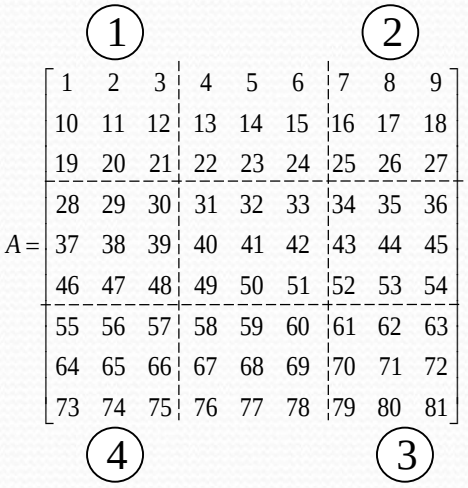


1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81

Upon execution, we obtain

Matr =

61	62	63	4	5	6	55	56	57
70	71	72	13	14	15	64	65	66
79	80	81	22	23	24	73	74	75
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
7	8	9	58	59	60	1	2	3
16	17	18	67	68	69	10	11	12
25	26	27	76	77	78	19	20	21



Clarification of Notation –

Consider the following two (2×5) matrices *A* and *B*

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \end{bmatrix}$$

Addition or subtraction: $C = A \pm B$

$$C = \begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & a_{13} \pm b_{13} & a_{14} \pm b_{14} & a_{15} \pm b_{15} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & a_{23} \pm b_{23} & a_{24} \pm b_{24} & a_{25} \pm b_{25} \end{bmatrix} \rightarrow (2 \times 5)$$

Clarification of Notation –

Consider the following two (2×5) matrices *A* and *B*

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \end{bmatrix}$$

Column augmentation: *C* = [*A*, *B*]

$$C = \left[\begin{array}{|ccccc|ccccc} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \end{array}\right] \rightarrow (2 \times 10)$$

Row augmentation: *C* = [*A*; *B*]

$$C = \left[\begin{array}{|ccccc|} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \end{array}\right] \rightarrow (4 \times 5)$$

Furthermore, if

$x = [x1\ x2\ x3]$
$y = [y1\ y2\ y3]$

then either

Z = [x', y'] or Z = [x; y]' gives $\rightarrow$ $Z = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} \rightarrow (2 \times 3)$

whereas

Z = [x'; y'] yields $\rightarrow$ $Z = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \rightarrow (6 \times 1)$

We illustrate these results with the following two vectors: *a* = [1, 2, 3] and *b* = [4, 5, 6]. The script is

```
x = [1, 2, 3];
y = [4, 5, 6];

Z1= [x', y']

Z2= [x; y]‘

Z3= [x'; y']
```

```
Z1 =
     1     4
     2     5
     3     6
```

```
Z2 =
     1     4
     2     5
     3     6
```

```
Z3 =
     1
     2
     3
     4
     5
     6
```

Dot Operations

A means of performing, on matrices of the *same* order, arithmetic operations on an element-by-element basis.

Consider the following (3×4) matrices

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}$$

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}$$

Then, the **dot multiplication** of *X* and *M* is

$$Z_m = X.*M = \begin{bmatrix} x_{11}*m_{11} & x_{12}*m_{12} & x_{13}*m_{13} & x_{14}*m_{14} \\ x_{21}*m_{21} & x_{22}*m_{22} & x_{23}*m_{23} & x_{24}*m_{24} \\ x_{31}*m_{31} & x_{32}*m_{32} & x_{33}*m_{33} & x_{34}*m_{34} \end{bmatrix}$$

The **dot division** of *X* by *M* is

$$Z_d = X./M = \begin{bmatrix} x_{11}/m_{11} & x_{12}/m_{12} & x_{13}/m_{13} & x_{14}/m_{14} \\ x_{21}/m_{21} & x_{22}/m_{22} & x_{23}/m_{23} & x_{24}/m_{24} \\ x_{31}/m_{31} & x_{32}/m_{32} & x_{33}/m_{33} & x_{34}/m_{34} \end{bmatrix}$$

The **dot exponentiation** of X and M is

$$Z_e = X.\hat{}\,M = \begin{bmatrix} x_{11} \hat{} m_{11} & x_{12} \hat{} m_{12} & x_{13} \hat{} m_{13} & x_{14} \hat{} m_{14} \\ x_{21} \hat{} m_{21} & x_{22} \hat{} m_{22} & x_{23} \hat{} m_{23} & x_{24} \hat{} m_{24} \\ x_{31} \hat{} m_{31} & x_{32} \hat{} m_{32} & x_{33} \hat{} m_{33} & x_{34} \hat{} m_{34} \end{bmatrix}$$

Use of Dot Operations

Let *a*, *b*, *c*, *d*, and *g* each be a (3×2) matrix, and consider the expression

$$Z = \left[\tan a - g\left(\frac{b}{c}\right)^{d} \right]^{2}$$

Then the MATLAB expression is

```
Z = (tan(a)-g.*(b./c).^d).^2;
```

which is equivalent to

$$Z = \begin{bmatrix} (\tan(a_{11}) - g_{11} * (b_{11}/c_{11}) \wedge d_{11}) \wedge 2 & (\tan(a_{12}) - g_{12} * (b_{12}/c_{12}) \wedge d_{12}) \wedge 2 \\ (\tan(a_{21}) - g_{21} * (b_{21}/c_{21}) \wedge d_{21}) \wedge 2 & (\tan(a_{22}) - g_{22} * (b_{22}/c_{22}) \wedge d_{22}) \wedge 2 \\ (\tan(a_{31}) - g_{31} * (b_{31}/c_{31}) \wedge d_{31}) \wedge 2 & (\tan(a_{32}) - g_{32} * (b_{32}/c_{32}) \wedge d_{32}) \wedge 2 \end{bmatrix}$$

Example –

Evaluate

$$v = e^{-a_1 t}\,\frac{\sin(b_1 t + c_1)}{t + c_1}$$

for 6 equally-spaced values of *t* in the interval $0 \le t \le 1$ when

$a_1 = 0.2$, $b_1 = 0.9$, and $c_1 = \pi/6$.

The script is

```
a1 = 0.2;  b1 = 0.9;  c1 = pi/6;
t = linspace(0, 1, 6);
v = exp(-a1*t).*sin(b1*t+c1)./(t+c1)
```

Upon execution, we obtain

```
v =
    0.9549    0.8590    0.7726    0.6900    0.6097    0.5316
```

Summation and Cumulative Summation Functions

sum: When $v = [v_1, v_2, \ldots, v_n]$, then

$$S = \mathrm{sum}(v) \rightarrow \sum_{k=1}^{\mathrm{length}(v)} v_k$$

where S a scalar.

If Z is a (3×4) matrix with elements z_{ij}, then

$$S = \mathrm{sum}(Z) \rightarrow \left[\sum_{n=1}^{3} z_{n1}, \sum_{n=1}^{3} z_{n2}, \sum_{n=1}^{3} z_{n3}, \sum_{n=1}^{3} z_{n4} \right] \rightarrow (1 \times 4)$$

To sum all the elements in an array, we use

$$S = \mathrm{sum}(\mathrm{sum}(Z)) \rightarrow \sum_{n=1}^{3} z_{n1} + \sum_{n=1}^{3} z_{n2} + \sum_{n=1}^{3} z_{n3} + \sum_{n=1}^{3} z_{n4} \rightarrow \sum_{i=1}^{4} \sum_{n=1}^{3} z_{ni} \rightarrow (1 \times 1)$$

Example –

Evaluate $$z = \sum_{m=1}^{4} m^m$$

Thus,

```
m = 1:4;

z = sum(m.^m)
```

which, when executed, gives

```
z =
   288
```

EE214 – Computer Aided Engineering Applications

End of Lecture 2