+ **MATRIX VARIABLES AND TWO DIMENSIONAL ARRAYS**

Matrices are organized rows and columns of numbers that mathematical operations can be performed on. MATLAB is organized around the rules of matrix operations. We have been working with arrays or row vectors which are actually 1 x N matrices.
It is suggested that the student run the MATLAB demo program on introductory MATLAB operations.

% FROM THE MATLAB DEMO PROGRAM WE EXTRACT THE FOLLOWING MATRIX INFORMATION:
To enter a matrix, spaces are put between the elements and semicolons are used to separate the rows. Brackets are placed around the data. For example, to enter a 3-by-3 matrix A (3 rows and 3 columns), type:

A = [1 2 3;4 5 6;7 8 0]

% which results in:

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

% our matrix A can be transposed with:
B = A'

% which results in:

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

% We note that the **transpose interchanges the rows and columns**.

% Matrix multiplication is indicated with: Can you see what happens ?

C = A * B

% producing:

```
C =  14   32   23
     32   77   68
     23   68  113
```

% The first term of the result, 14, was formed by multiplying the first row of the A matrix with the first column of the B matrix in the manner of
1x1 + 2x2 + 3x3 =14: (Like a dot product of two vectors! Row vector dot column vector!)
The second term of C ,32, is in the **first row and second column**, hence we use the first row of A and the 2 nd column of B to compute 1x4 + 2x5 + 3x6 = 32 and so forth for each term. Another way of defining the matrix product is to think of the corresponding rows and columns in the matrices as vectors. To obtain the terms in the matrix that results from the product, we undertake the scaler product of a row vector with a column vector to get each term in the resulting matrix that corresponds to the row and column used in the other matrices.

As a reminder(assuming the reader has seen matrices and related operations in mathematics) when multiplying matrices, if we have an M row by N column matrix, to form a product, the number of rows of the second matrix must be the same as the number of columns of the first. The dimension of the matrix formed in the product will have the number of rows of the first and the number of columns of the second. So if C is an M by N matrix and D is an N by P matrix the product A*B = a matrix with dimension M by P.
stated differently (M BY N) * (N by P) = also written (MxN)*(NxP) = (M by P) or resulting in a M x P matrix. Inner dimensions must match and the reverse order of multiplication is not possible by this definition unless their is a match in values. Square matrices can be multiplied in any order.

Example: D =[1 2 3; 7 8 9;1 2 5] ; and P=[2 3; 7 7; 4 5];
Or 3x 3 * 3 x 2 = 3 x 2 matrix

```
>> D*P
ans =
   28   32
  106  122
   36   42
```

Note: >> P*D ie 3 x 2 * 3 x 3 not possible by these rules.
Error using *
Inner matrix dimensions must agree.

% A family of functions are available to calculate common matrix properties useful for solving a number of problems that can be done with matrices.
Some examples follow:

% determinant the operator we have talked about for help in solving simultaneous solutions.

```
det(A)
ans =   27
```

```
[m,n] = size(A)
m = 3    % = number of rows
n = 3    % = number of columns
```

% Matrix operations are used to solve a large number of problems such as simultaneous equations, we will return a number of times later to the MATLAB matrix functions.

MATRIX VS. ARRAY OPERATIONS

% MATLAB also permits us to treat the Matrix as an Array. The distinction being that in Array operations (such as .*,./,.^) we perform the **Calculations element by element**. With the above matrices we illustrate some array operations

ARRAY MULTIPLICATION

```
» A.*B
ans =    1    8   21
         8   25   48
        21   48    0
```

Here we note that the first element of A is multiplied by the first element of B. ditto for the second elements, etc.
On the other hand
MATRIX MULTIPLICATION gives quite a different result.

```
» A*B
ans =14   32   23
     32   77   68
     23   68  113
```

% MATRIX MULTIPLICATION A*B is not in general the same as B*A(may not even be possible)
Consider this example

```
» C =[ 3 4 ; 6 7 ;1 2]

C =  3    4
     6    7
     1    2
```

```
» A*C
ans =18   24
       48   63
       69   84
```
% Hence A*C a 3 by 3 * 3 by 2 gives us a 3 by 2 answer.

% What would the result be for C*A??

ARRAY MULTIPLICATION between matrices also requires dimensions to be equal since we do the calculation term by term.
% if we try A.*C as an array calculation we get
```
» A.*C
 Error using == .*    Matrix dimensions must agree.
```
% An error results since we need a match for the inner dimensions and with a moments reflection one can see that it is not possible to define the product.
We can further see the distinction between array and matrix operations by **using vectors**, given the following vectors v1 and v2, namely
```
» v1=[1 3 4]
v1 =     1     3     4
» v2 =[ 2 3 5]
v2 =     2     3     5
```
% then array multiplication v1.*v2 gives
```
» v1.*v2
ans =     2     9    20
```
 i.e. an element by element multiplication:

if we try matrix multiplication
```
» v1*v2
Error using == *   Inner matrix dimensions must agree.
```
on the other hand if we take the transpose of the 'row vector' v2 to
we get a column vector as in
```
» v3=v2'
v3 =     2
       3
       5
```
% we can now matrix multiply v3 a 1 x 3 to the column vector v1 a 3 x 1 to get a 3 x 3 matrix (Inner matrix dimensions do agree.), as follows
```
» v3*v1
ans = 2     6     8
        3     9    12
        5    15    20        % by the rules of matrix multiplication.
```
Here the reverse can be done since we multiply v1 a 1 x 3 to the column vector v3 a 3 x 1 to get a 1 x 1 matrix (Inner matrix dimensions do agree.) as
```
» v1*v3
ans =   31
```
%Consider the following ARRAY OPERATIONS:
% these are element by element, study each example with the above vectors)
```
» v1.*v2
ans =     2     9    20

» v2./v1
ans =    2.0000    1.0000    1.2500
```

```
» v2.^v1
ans =    2   27   625

» v1+v2
ans =    3    6    9
```

% Many of the MATLAB basic functions are used as array operations. If the functions receives an array than the answer is an array with matching dimension, as in

```
» sin(v2)
ans =   0.9093    0.1411   -0.9589
```

% or as we saw in graphing

```
» t=1:1:5
t =    1    2    3    4    5
» y=sin(t)
y =    0.8415    0.9093    0.1411   -0.7568   -0.9589
» plot(t,y)
```

MATRICES AS TWO DIMENSIONAL SUBSCRIPTED ARRAYS

% Getting back to treating the MATLAB Matrix as two dimensional Arrays

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

% We can use two subscript variables or numbers to reference the row and column position. The first subscript is the row number and the second the column number. As in:

```
» A(1,1) =  1
» A(2,2) =  5
» A(3,2)+A(2,3) =  14
```

% We can use a **nested pair of for loops to process all members of a MATRIX (a two dimensional array)** as in:

```
» for i=1:3          % OUTPUT BY ROW
    for j=1:3
      A(i,j)
    end;
  end;
ans =    1
ans =    2
ans =    3
ans =    4
ans =    5
ans =    6
ans =    7
ans =    8
ans =    0
```

% CONSIDER THE FOLLOWING NESTED LOOP OUTPUT BY ROW which illustrates us addressing all values of the matrix.

```
» for i=1:3
    for j=1:3
      fprintf( 'Row  %2.0f Column  %2.0f value = %3.0f\n',i,j,A(i,j))
    end;
  end;
Row   1 Column   1 value =   1
Row   1 Column   2 value =   2
Row   1 Column   3 value =   3
Row   2 Column   1 value =   4
Row   2 Column   2 value =   5
Row   2 Column   3 value =   6
Row   3 Column   1 value =   7
Row   3 Column   2 value =   8
Row   3 Column   3 value =   0
```

```
% We can use this technique to initialize a two dimensional array as in
» for i=1:4
   for j=1:3
      SUM(i,j) =i
   end;
  end;
SUM =    1    1    1
         2    2    2
         3    3    3
         4    4    4
```

% Another illustration of the nested for loop processing is to generating a matrix called the identity matrix whose diagonal values are 1 and all other values are 0. A useful Matrix in engineering problem solving.

```
» for i=1:5
   for j=1:5
     if i==j
       idea(i,j) =1
     else
       idea(i,j) =0
     end
   end;
 end
idea =   1    0    0    0    0
         0    1    0    0    0
         0    0    1    0    0
         0    0    0    1    0
         0    0    0    0    1
```

SOME OTHER BASIC MATLAB FUNCTIONS AND MATRICES

The sum() function:

If we need the total of all elements we can use the sum() function

```
% If we use sum(a) and a is a vector as in
» a=[1 2 3 4 5]
a =    1    2    3    4    5
» sum(a)
ans =    15
% i.e. We get a scaler sum of all values BUT if A is a MATRIX
» A=[1 2 3; 4 5 6; 7 8 9]
         A =  1    2    3
              4    5    6
              7    8    9
» s1 = sum(A) =   12    15    18
% s1 is A VECTOR WHOSE ELEMENTS ARE THE
                              SUM OF THE COLUMNS OF THE MATRIX 'A'
% If we sum this vector, s1, we effectively get the sum of all elements of the original Matrix.

» sum(s1) =    45

% or we could of just used sum() twice to get the total of all elements
```

» **sum(sum(A)) =   45**   % Several functions can be used this way!

Review of SOME Matrix operations
Given the following matrices

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

>> B =[2 2 2;3 3 3;1 1 1;]
B =    2    2    2
       3    3    3
       1    1    1
```

^ MATRIX POWER OPERATION ^

```
>> F = B^2
 F =  12   12   12
      18   18   18
       6    6    6
```

% NOTE THIS IS NOT AN ELEMENT BY ELEMENT FUNCTION LIKE IN THE ARRAY OPERATION

```
>> F2 = B.^2
F2 =    4    4    4
        9    9    9
        1    1    1
```

% BUT MATRIX POWER IS EQUIVALENT TO

```
>> F3 = B*B ==B^2 as above
F3 =   12   12   12
       18   18   18
        6    6    6
```

% ' MATRIX TRANSPOSE OPERATION '

```
>> G = B'
G =    2    3    1
       2    3    1
       2    3    1
```

Above we have

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

so

```
>> G2 =A'
G2 =    1    4    7
        2    5    8
        3    6    9
```

% This operation is useful to Transpose a row vector to a column vector
and is used a great deal. as in

```
>> x=[ 1 2 3]'
x =     1
        2
        3
```

```
%                    SCALER MULTIPLICATION
>> A=[1.0 2.2;3.0 4.0;-1.0 0.0]
A =    1.0000    2.2000
       3.0000    4.0000
      -1.0000         0
>> Y = 4*A
Y =    4.0000    8.8000
      12.0000   16.0000
      -4.0000         0
```

% NOTE IN THESE LAST ILLUSTRATIONS THE MATRIX DIMENSIONS OF THE ANSWER MATCH
THE ORIGINAL MATRIX DIMENSION OF 3 x 2 AS IN THE CASE ABOVE OF 3 X 3

% CONSIDER THE FACT THAT IN MULTIPLICATIONS THE INNER DIMENSION MUST MATCH ELSE THE COMPUTATION WILL NOT BE DONE. As noted above. For example. GIVEN

```
A =   1.0000    2.2000
      3.0000    4.0000
     -1.0000         0
% AND

b =[4.0 -3.0;2.0 6.0]
b =     4    -3
        2     6

>> b * A
Error using MM *
Inner matrix dimensions must agree.
```

% NOTE: b IS A 2 X 2 AND A IS A 3 X 2 AND INNER VALUES ARE 2 AND 3

```
>> A * b
ans =    8.4000   10.2000
        20.0000   15.0000
        -4.0000    3.0000
```

% NOTE:FOR A * b WE HAVE A 3 X 2 AND A 2 X 2 AND BY MATRIX RULES WE GET A 3 X 2

% **DOT (&CROSS) PRODUCT AND VECTOR MAGNITUDES**

% THE DOT PRODUCT IS AN OPERATION ON VECTORS (MATRICES WITH 1 COLUMN OR ROW)

```
>> V1 =[3.0 1.5 -0.5]'
V1 =    3.0000
        1.5000
       -0.5000

>> V2=[1.0 2.0 3.0]'
V2 =     1
         2
         3
```

% THE DOT PRODUCT(also called the SCALAR product) IS JUST THE SUM OF THE ARRAY PRODUCT OF THE TWO VECTORS

```
>> ARRPROD = V1.*V2
ARRPOD =    3.0000
            3.0000
           -1.5000
```

DOT (SCALAR) PRODUCT IS THE THE SUM OF THE ARRPOD
>> sum(ARRPOD)
ans =   4.5000

% I.E. WE ARE SUMMING THE PRODUCTS OF THE VECTOR COMPONENTS BY THE USUAL DOT PRODUCT DEFINITION
MATLAB has the dot() function that does the same thing ie.
>> dot(V1,V2)
ans =   4.5000

MATLAB also has a simple way to do the cross product whose answer is a vector perpendicular to the two original vectors.
>> C=cross(V1,V2)
C =   5.5000   -9.5000   4.5000
If you recall from math classes the dot product of vectors perpendicular to each other is zero! This is easily shown with the two vectors V1 and V2 with C which is perpendicular to both. Taking the dot product of C with V1 and V2 verifies this point
>> dot(C,V1)
ans =   0

>> dot (C,V2)
ans =   0

% THE MAGNITUDE OF A VECTOR IS JUST THE SQUARE ROOT OF THE DOT PRODUCT OF ITSELF as we have already pointed out. For examples:
>>MAG_V1 = sqrt(sum(V1.*V1))
MAG_V1 =   3.3912
>> MAG_V2 = sqrt(sum(V2.*V2))
MAG_V2 =   3.7417

Of course the magnitude can be found with the MATLAB function ‘norm”
As illustrated as follows.
>>norm(V1)
ans =   3.3912
>> norm(V2)
ans =   3.7417
Both agree with MAG_V1 and MAG_V2 above.

%FOR MORE ACCURATE ANSWER WE CAN ALWAYS..
>> format long

MAG_V1 = sqrt(sum(V1.*V1))
MAG_V1 =   3.39116499156263

% AS A FURTHER APPLICATION

% We again review the ANGLE BETWEEN TWO VECTORS
% IT IS NOT DIFFICULT TO SEE THAT IF phi IS THE ANGLE BETWEEN V1 AND V2 THEN
>> cos_phi = sum(V1.*V2)/(sqrt(sum(V1.*V1)).*sqrt(sum(V2.*V2)))
cos_phi =   0.3546

% The function acos() gives us the angle
>> phi = acos(.3546)
phi =   1.2083
% in radians

% NOTE: WE CAN THEN FIND THE ANGLE WITH A FORMULA LIKE
% angle = acos(sum(V1.*V2)/(norm(V1).*norm(V2))

>>acos(dot(V1, V2) / (norm(V1) * norm(V2)))
ans =   1.2083   which agrees with above

% SIMULTANEOUS EQUATIONS:

% MATLAB provides a large variety of matrix and function operations to solve simultaneous equations with n variables and n equations
The problem for three unknowns x,y,z can be stated as given;

$$a_{11}x + a_{12}y + a_{13}z = b_1$$
$$a_{21}x + a_{22}y + a_{23}z = b_2$$
$$a_{31}x + a_{32}y + a_{33}z = b_3$$

% It is usually assumed we know all of the coefficients, **a**, as well as the values of the **b**'s. We have three equations to solve three unknowns in this illustrative case.
In other words, what are the values of x,y and z that simultaneously solve all three of the above equations.

We can interpret each of the above equations as planes in space. If the three planes intersect at a common (x,y,z) point then the common point values of x,y and z are the solution to the problem.
It could happen that two of the planes are the same.If two planes intersect they form a line (of values) which results in an infinite number of solutions
The is no solution if all three planes are parallel.

THE MATLAB APPROACH TO SIMULTANEOUS EQUATION

%CONSIDER THE FOLLOWING LINEAR EQUATIONS

$$2x + 3y - z = 1$$
$$3x + 5y + 2z = 8$$
$$x - 2y - 3z = -1$$

% THIS PROBLEM IS CONVERTED TO MATRIX FORM BY IDENTIFYING THE COEFFICIENTS OF THE UNKNOWN AS A MATRIX CALLED, **A,** AND THE VALUES OF x, y AND z AS A COLUMN VECTOR CALLED, **U**, AND THE CONSTANT TERMS ON THE RIGHT A COLUMN VECTOR CALLED **B** AS FOLLOWS

```
>> A = [2 3 -1;3 5 2; 1 -2 -3]
A =     2    3   -1
        3    5    2
        1   -2   -3

>> B = [1 8 -1]'
B =     1
        8
       -1

U =     x
        y
        z
```

% **THE LINEAR EQUATIONS REDUCE TO THE MATRIX EQUATION**

% AU = B

% ONE CAN CHECK THIS LAST EQUATION BY DOING THE MATRIX MULTIPLICATION AND YOU WILL GET THE MATRIX EQUIVALENT TO THE ABOVE LINEAR EQUATIONS

THE SIMPLEST AND THE BEST SOLUTION OF THE MATRIX EQUATION AU = B IN MATLAB IS TO USE THE **LEFT DIVISION OPERATION** DEFINED AS

% U = A\B

% IF WE CARRY THIS OUT WE GET THE U VECTOR WHICH IS THE x,y AND z SOLUTION

```
>> U = A\B
U =    3
      -1
       2
```

% NAMELY x = 3 y = -1 AND z =2 : extremely easy way to solve such a problem!
% To solve linear equations of more variables is still done by the simple application of the left division operation AFTER ASSIGNING THE APPROPRIATE VECTORS AND MATRICES.

```
>> help \

\    Left division.
     A\B is the matrix division of A into B, which is roughly the
     same as INV(A)*B , except it is computed in a different way.
     If A is an N-by-N matrix and B is a column vector with N
     components, or a matrix with several such columns, then
     X = A\B is the solution to the equation A*X = B computed by
     Gaussian elimination. A warning message is printed if A is
     badly scaled or nearly singular....
```

% As you see from the MATLAB definition left division combines a number of techniques to get the solutions. In general basic solutions to simultaneous equations are obtain by Cramer's rule or Gauss Elimination or Gauss-Seidel methods. Here in MATLAB the left division operator is basically a Gaussian elimination technique that is very simple to set up. A great deal of basic coding is not needed here.

% Note that in the definition above left division is equivalent to inv(A)*B
inv(A) is the inverse Matrix of the original Matrix. This is **not** a good way to solve the system IN MATLAB and should be avoided.

% Additional illustration

We set up as AU =B problem as follows

```
>> A =[7 -4 0;-4 15 -6;0 -6 8]
A =    7    -4     0
      -4    15    -6
       0    -6     8

>>B = [30 0 40]'
B =   30
       0
      40
```

 and the solution is U = A\B

```
>> U = A\D
U =   7.5652
      5.7391
      9.3043
```

More precision is seen with

```
>> format long
U =   7.56521739130435
      5.73913043478261
      9.30434782608696
```

% **CHECKING THE SOLUTION**

 How good is the solution : we can Check U since AU should = B

```
>> A*U
ans =   30.0000
```

```
    -0.0000
    40.0000
```

% WE CONCLUDE LEFT DIVISION GIVES US VERY GOOD SOLUTIONS
% The CHECK tells us how good our solution is! Since MATLAB used Double Precision in all calculations our answers were very good because "roundoff" errors are reduced.

% PROBLEMS IN GETTING A SOLUTION
If two equations are in the same plane as in this example (one line is a simple multiple of another)
CONSIDER

```
>> A = [2 3 -1;4 6 -2;1 -2 -3]
A =     2     3    -1
        4     6    -2
        1    -2    -3

B =[1 8 -1]'
```

% Attempting to get a solution results in

```
>> U=A\B
Warning: Matrix is singular to working precision.
U =     ∞
        ∞
        ∞
```

That is there are an infinite number of solutions along the line of intersection formed by the two non-parallel planes

EIGENVECTORS AND EIGENVALUES (very useful engineering concepts)

% Once we have a linear equation system and define a matrix of the coefficients of our unknowns, A below, it also occurs in many scientific and engineering applications to FIND VECTORS X AND SCALARS E THAT SATISFY

% AX=XE

THIS IS KNOWN AS AN EIGENVALUE PROBLEM.

```
       A1  A2  A3
A =    A4  A5  A6
       A7  A8  A9
```

IN the three equation CASE THERE WOULD BE THREE SCALER EIGENVALUES FOR THREE EIGENVECTORS(here A is a 3 x 3 square matrix) with the original problem and E is a 3 x 3 diagonal matrix with the eigenvalues in the diagonal or all other members are zero. And X is a 3 x 3 matrix containing three columns which represent 3 eigenvectors. That is we would need to get three separate vectors FOR THREE CORRESPONDING SCALARS E1,E2 AND E3 (THE EIGENVALUES) THAT WOULD SATISFY THE ABOVE EQUATION. WE WOULD HAVE THREE SEPARATE EQUATIONS FOR EACH Eigenvalue and corresponding Eigenvector as follows

% MATLAB HAS THE FUNCTION eig TO SATISFY THESE REQUIREMENT
>> help eig
EIG(A) is a **vector** containing the eigenvalues (SCALARS) of a square matrix A.

[XV,D] = EIG(A) produces a diagonal matrix D of eigenvalues and a full matrix XV whose columns are the corresponding eigenvectors so that A*XV = XV*D....

Example 1

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

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

>> eig(A)   gives just the 3 eigenvalues

```
ans =   16.1168
        -1.1168
        -0.0000
```

>> [XV,E]=eig(A)

```
XV =   -0.2320   -0.7858    0.4082     each column represents a vector the eigenvectors
       -0.5253   -0.0868   -0.8165
       -0.8187    0.6123    0.4082


E =   16.1168         0         0     E is a diagonal matrix containing the eigenvalues.
            0   -1.1168         0
            0         0   -0.0000
```

Since A*XV=XV*E we can check this as follows and see that they match

>> A*XV=

```
   -3.7386    0.8776   -0.0000
   -8.4665    0.0969   -0.0000
  -13.1944   -0.6839         0
```

>> XV*E=

```
   -3.7386    0.8776   -0.0000
   -8.4665    0.0969    0.0000
  -13.1944   -0.6839   -0.0000
```

%EIGENVECTOR - EIGENVALUE EXAMPLE 2
GIVEN MATRIX A AS

```
>> A=[2 3 -1;3 5 2;1 -2 -3]
A =     2     3    -1
        3     5     2
        1    -2    -3
```

% WE CAN SEE THE EIGENVALUES WITH eig(A) AS

```
>> eig(A)
ans =    6.7195
        -1.3598 + 1.1938i
        -1.3598 - 1.1938i
```

Each value of the resulting vector is one of the scalar numbers
that satisfy AX=EX

%IT IS BETTER TO USE THE [XV,E] =eig(A) WHICH GIVES US BOTH THE VECTORS AND SCALAR VALUES

```
>> [XV,E]=eig(A)
XV =  -0.5500  -0.5341 - 0.1335i  -0.5341 + 0.1335i
      -0.8274   0.4415 + 0.0445i   0.4415 - 0.0445i
       0.1137  -0.6293 + 0.3224i  -0.6293 - 0.3224i
```

THE THREE COLUMNS OF XV CORRESPOND TO THE THREE EIGENVECTORS XEV1,XEV2 AND XEV3 WE NOTE HERE THAT TWO OF THEM ARE IMAGINARY

THE VALUE OF E IS

```
E =    6.7195                0                  0
       0            -1.3598 + 1.1938i           0
       0                     0           -1.3598 - 1.1938i
```

Here the scalar EIGENVALUES are the diagonal elements of D

% **CHECKING THE RESULT**

The MATLAB definition suggest a check A*XV = XV*E

```
>>A*XV
ans =  -3.6958            0.8857 - 0.4560i   0.8857 + 0.4560i
       -5.5596           -0.6534 + 0.4666i  -0.6534 - 0.4666i
        0.7638            0.4709 - 1.1896i   0.4709 + 1.1896i

>> XV*E
ans =  -3.6958            0.8857 - 0.4560i   0.8857 + 0.4560i
       -5.5596           -0.6534 + 0.4666i  -0.6534 - 0.4666i
        0.7638            0.4709 - 1.1896i   0.4709 + 1.1896i
```

Which we see gives us equivalent answers.

% We can see the problem from the original definition AX = EX by focusing on the real solution.

FROM ABOVE IT SHOULD BE TRUE THAT

A*XEV1 =E1*XEV1

THE REAL EIGENVECTOR IS THE FIRST COLUMN OF XV

```
>> XEV1 =XV(1:3)'    %Remember MATLAB references matrix by column here.
XEV1 =   -0.5500      i.e. when we use one subscript variable for index
         -0.8274
          0.1137
```

% THE REAL EIGENVALUE IS THE FIRST DIAGONAL ELEMENT OF E

```
>> E1 = E(1)
   E1 =    6.7195
```

% NOW WE CHECK IF A*XEV1 = E1*XEV1

```
>> A*XEV1
ans =   -3.6958
        -5.5596
         0.7638

>> E1*XEV1
ans =   -3.6958
        -5.5596
         0.7638
```

And we see that the eigenvector XVE1 and eigenvalue E1 satisfy the original DEFINITION OF THE concept.

% By the way the MATLAB function 'eig' solves for vectors that have a magnitude of 1. In other words the eigenvectors are unit vectors we can prove this for our case by using the dot product definition above

of the eigenvector XV and OR THE NORM FUNCTION TO get the magnitude OF THE
% VECTORS. NAMELY
```
>> magXEV1 =sqrt(sum(XEV1.*XEV1))
magE =     1
```
or
```
>> norm(XEV1)
ans =    1.0000
```

% WHICH SHOWS THE **EIGENVECTORS ARE UNIT VECTORS**.

LABORATORY TASKS (print m files and outputs and any data files created. Number all tasks)

46: Given A = [1 -2 9;3 7 -6; 5 -8 0]
find

1. **B=A' ?**
2. **A*B**
3. **B*A does 2 and 3 equal each other? Most math 2*3=3*2 but not here!**
4. The determinant of B=?

47: write a short m file that will load up a matrix G below with the following values one at a time and uses a double for loop for each member. All members get summed. Once loaded the program then prints out each value using a double for loop which matches the following ,also the final sum of all members is printed(done in the loops).

```
G=  6 4 9 5
    8 2 3 6
    9 3 2 1
```

The program also produces the sum(sum(G)) to check it's the same as the above sum of all members

48:Given the vectors V1 =[9 -2 4] and V2=[6 5 3] Show formulas used for the following

1. Calculate the magnitude of each vector two ways.
2. Calculate the dot product of V1 and V2 two ways
3. Calculate the angle between the vectors with two formulas

49. See section on importing large amounts of data… for this NEXT problem.

The following laboratory task will drill you on the above MATRIX principles and introduce you to sending a programs output to a file.

1.Create a matrix called rain that is a 3 x 4 matrix with the following data. Your program should ask for one of the values and assign it to the proper position. Use a double for loop to create the matrix rain.

```
2.0 3.1 2.8 0.7 1.1 0.0 1.1 0.4 0.0 0.0 3.2 2.1
```

DO NOT TRANSFORM THE MATRIX INTO A ONE DIMENSIONAL VECTOR!
The program (m file) should also do the following after loading the matrix.

2. use **sum()** to get the total of all values

3. Use **nested for loops** to obtain the sum and compare with the function answer!

4. use the **size()** to get the total count of values ie. m x n

5. express the average value 'ave' using the sum() and total count

6. Use **mean(**) to get the average value and compare with the above average value.

7. Use **min()** and **max()** to get the minimum and maximum values of the rainfall data.

8. use **nested loops** to determine the number of days with greater than average rainfall as well as the day number this took place. Output this information together.

9. THE ENTIRE OUTPUT SHOULD BE SENT TO A FILE WITH THE "fprintf()' COMMAND

SEE THE DEFINITION IN THE **help fprintf**. Use the Editor to add text to make a report out of the file and print the file on paper from the Editor.

50. Consider the linear equations

$$
\begin{aligned}
4x - y + z - 2w &= 4 \\
x + 6y + 2z - w &= 9 \\
-x - 2y + 5z - 3w &= 2 \\
2x + 3y + z + w &= -12
\end{aligned}
$$

1. Find the values of x,y,z,w that satisfy these
2. Check the answer
3. Find the eigenvectors and eigenvalues
4. Check answers FOR ALL CASES (REAL EIGENVALUES ONLY)