EXPERIMENT-THREE

VECTORS, MATRICES and ARRAYS

1- VECTORS

In MATLAB we can make a row vector (X) by separating the elements of a row with blanks or commas and surrounding the entire list of elements with square brackets, [ ].

```
>> X= [3 6 4]
```

or

```
>> X= [3,6,4]
```

MATLAB displays the vector:

```
X =
    3    6    4
```

Also we can use the colon operator, :, to perform a row vector . The expression 1:10 is an increased row vector containing the integers from 1 to 10

```
>> X= 1:10

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

and

```
>> C= 0:pi/4:pi

C=
   0    0.7854    1.5708    2.3562    3.1416
```

or a decreased row vector:

```
>> S= 100:-7:50
S=
  100  93  86  79  72  65  58  51
```

If you want to make X as a column vector, separate the elements of a row with semicolons, ; , and surrounding the entire list of elements with square brackets, [ ].

```
>> X= [4; 6; 7]

X =
    4
    6
    7
```

2- Matrices

The best way for you to get started with MATLAB is to learn how to handle matrices. So you have only to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon, ; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].

As an example, if we enter in the command window the matrix A:

```
>> A=[2 3 4; 7 5 8;14 23 55]
```

MATLAB displays the matrix you just entered.

```
A =
    2    3    4
    7    5    8
   14   23   55
```

Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A. Now you have A in the workspace.

2.1- Arithmetic operations on matrices:

The function sum enabled you to take the sum along any row or column, or along either of the two main diagonals. Let's verify that using MATLAB, If we want the sum of each column in matrix A above:

```
>> sum(A)
```

MATLAB replies with

```
ans =

   23    31    67
```

So You have computed a row vector containing the sums of the columns of A. How about the row sums? MATLAB has a preference for working with the columns of a matrix, so the easiest way to get the row sums is to transpose the matrix, compute the column sums of the transpose, and then transpose the result. The transpose operation is denoted by an apostrophe or single quote, '.
It flips a matrix about its main diagonal and it turns a row vector into a column vector. So

```
>> A'
```

produces

```
ans =
    2    7   14
    3    5   23
    4    8   55
```

and

```
>> sum(A')'
```

produces a column vector containing the row sums

```
ans =

    9
   20
   92
```

The sum of the elements on the main diagonal is easily obtained with the help of the diag function, which picks off that diagonal.

```
>> diag(A)
```

produces

```
ans =

     2
     5
    55
```

and

```
>> sum(diag(A))
```

produces

```
ans =
    62
```

The other diagonal, the so-called *anti diagonal*, is not so important mathematically, so MATLAB does not have a ready-made function for it. But a function originally intended for use in graphics, fliplr, flips a matrix from left to right. So the sum of the anti diagonal is:

```
>> sum(diag(fliplr(A)))

ans =

    23
```

2.2- **Subscripts:**

The element in row i and column j of A is denoted by A(i,j). For example, A(3,2) is the number in the third row and second column. For our matrix (A), A(3,2) is 23. So it is possible to compute the sum of the elements in the third column of A by typing

```
>> A(1,3)+A(2,3)+A(3,3)
```

This produces

```
ans =

    67
```

But this is not an efficient way. The following subsection will produce the best way for this case.

2.3- The Colon Operator:

The colon operator is extremely powerful, and provide for very efficient ways of handling matrices. Subscript expressions involving colons refer to portions of a matrix. A(1:k,j) is the first k elements of the jth column of A. So:

```
>> sum(A(1:3,3))
```

computes the sum of the third column.

```
ans =
   67
```

But there is a better way. The colon by itself refers to *all* the elements in a row or column of a matrix and the keyword end refers to the *last* row or column. So

```
>> sum(A(:,end))
```

computes the sum of the elements in the last column of A.

```
ans =
   67
```

Other advantages of the colon operator will be listed in the following examples:

```
>> A(2:3,1:2)
```

this mean: return second and third rows, first and second columns of A.

```
ans =

    7     5
   14    23
```

```
>> A(3,:)
```

this mean: return the third row of A.

```
ans =
   14    23   55
```

```
>> A(1:2,1:2) = ones(2)
```

this mean: replace first and second rows and columns of A by a square matrix of 1's.

```
ans =
    1     1     4
    1     1     8
   14    23    55
```

2.4- Deleting Rows and Columns:

You can delete rows and columns from a matrix using just a pair of square brackets. Start with

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

```
D =

    4    6    8    7
    3    4    5    1
    2    3    8    9
    5    3    1    9
>>X = D;
```

Then, to delete the second column of X, use

```
>> X(:,2) = [ ]
```

This changes X to

```
X =
    4    8    7
    3    5    1
    2    8    9
    5    1    9
```

If you delete a single element from a matrix, the result isn't a matrix anymore. So, expressions like

```
>> X(1,2) = []
```

result in an error. However, using a single subscript deletes a single element, or sequence of elements, and reshapes the remaining elements into a row vector. So

```
>> X(2:2:10) = []
```

results in

```
X =
     4  2  8  8  7  9  9
```

2.5- Concatenation:

Concatenation is the process of joining small matrices to make bigger ones. In fact, you made your first matrix by concatenating its individual elements. The pair of square brackets, [], is the concatenation operator. For an example, start with the 2- by-2 square matrix, C,

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

C =

    1    2
    3    4
```

and form the matrix B:

```
>> B = [C C+2; C+4 C+6]
```

The result is an 4-by-4 matrix, obtained by joining the four sub matrices.

```
B =

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