commands relate to variables

- **who** shows variables that have been defined in this Command Window

  (this just shows the names of the variables)

```
>> a=1;
>> b=2;
>> n=a+b
n =
    3
>> who
Your variables are:
a  b  n
```

- **whos** shows variables that have been defined in this Command Window

  (this shows more information on the variables, similar to what is in the Workspace Window)

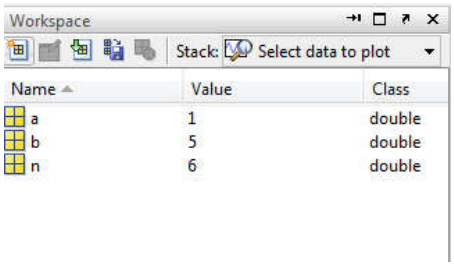
```
>> whos
```

Name	Size	Bytes	Class	Attributes
a	1x1	8	double	
b	1x1	8	double	
n	1x1	8	double	

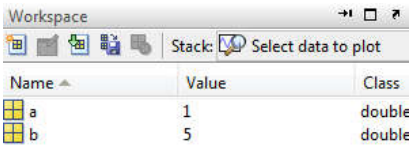
- **Clear** clears out all variables so they no longer exist.

- **clear** *variablename* clears out a particular variable.

```
>> a=1;
>> b=5;
>> n=a+b

n =

     6

>> clear n
```



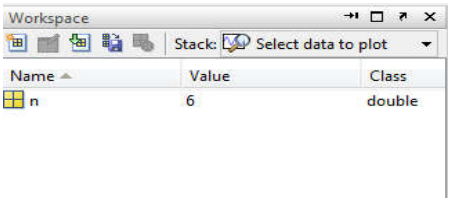
Workspace before using command clear n



Workspace after using command clear n

- **clear** *variablename1 variablename2* . . . clears out a list of variables (note: separate the names with spaces)

>> clear a b



Workspace after using command

clear a b

If nothing appears when **who** or **whos** is entered, that means there aren't any variables! For example, in the beginning of a MATLAB session, variables could be created and then selectively cleared (remember that the semicolon suppresses output).

Constants

Variables are used to store values that might change, or for which the values are not known ahead of time. Most languages also have the capacity to store Constants, which are values that are known ahead of time, and cannot possibly Change. An example of a constant value would be pi , or p, which is 3.14159. . . . In MATLAB, there are functions that return some of these constant values, some of which include

pi 3.14159. . . .

i $\sqrt{-1}$

j $\sqrt{-1}$

inf infinity ∞

NaN stands for "not a number," such as the result of 0/0

Practice:

```
>> pi
ans =
   3.1416
```

```
>> i
ans =
   0 + 1.0000i
```

```
>> j
ans =
   0 + 1.0000i
```

```
>> inf
ans =
   Inf
```

```
>> NaN
ans =
   NaN
```

Introduction to Arrays

Array and matrices form the fundamental mathematical entities that MATLAB handles effectively and easily. An Array is a list of numbers or characters arranged in rows or columns while a matrix is tow- dimensional array. MATLAB can handle higher-dimensional arrays as well.

MATLAB supports two types of operations, known as *matrix operations* and array *operations* Matrix operations will be discussed first.

1- Matrix generation

Matrices are fundamental to MATLAB. Therefore, we need to become familiar with matrix generation and manipulation. Matrices can be generated in several ways.

1.1 Entering a vector

A vector is a special case of a matrix. As discussed earlier, an array of dimension 1xn is called a row vector, whereas an array of dimension mx1 is called a column vector. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, v, type

```
>> v = [1 4 7 10 13]
v =
 1    4    7   10   13
```

Column vectors are created in a similar way, however, semicolon (;) must separate the components of a column vector,

```
>> w = [1;4;7;10;13]
w =
    1
    4
    7
   10
   13
```

1.2 Colon operator

The colon operator will prove very useful and understanding how it works is the key to efficient and convenient usage of MATLAB. It occurs in several different forms.

Often we must deal with matrices or vectors that are too large to enter one element at a time. For example, suppose we want to enter a vector x consisting of points
(0, 0.1, 0.2, 0.3,…. , 5).We can use the command

```
>> x = 0:0.1:5;
```

The row vector has 51 elements.

1.3 Creating vector by Colon operator

Example1:

```
>> x=1:4
x =
   1    2    3    4
```

Example2:

```
>> y=4:2:10
y =
    4       6    8    10
```

1.4Linear spacing

On the other hand, there is a command to generate linearly spaced vectors: linspace. It is similar to the colon operator (:), but gives direct control over the number of points. For example,

y = linspace(a,b)
generates a row vector y of 100 points linearly spaced between and including a and b.

y = linspace(a,b,n)
generates a row vector y of n points linearly spaced between and including a and b. This is useful when we want to divide an interval into a number of subintervals of the same length.

Example1:

```
>> f=linspace(1,5,4)
f =
   1.0000    2.3333    3.6667    5.0000
```

Example2:

```
>> g=linspace(3,9,3)
g =
    3     6     9
```

1.5 Vector Transpose

a row vector is converted to a column vector using the transpose operator. The transpose operation is denoted by an apostrophe or a single quote (').

Example1:

```
>> w = [1;4;7;10;13]
w =
     1
     4
     7
    10
    13

>> v=w'
v =
     1     4     7    10    13
```

Example 2:

```
>> a=[3 4 1 8 ]
a =
     3     4     1     8

>> b=a'
b =
     3
     4
     1
     8
```

1.6 access blocks of vector elements

Thus, v(1) is the first element of vector v, v(2) its second element, and so forth.

Example1:

```
>> v=[5 1 9 6 3]
v =
   5    1    9    6    3


>> v(2)
ans =
      1

>> v(4)
ans =
   6
```

Furthermore, to access blocks of elements, we use MATLAB's colon notation (:). For example to access the first three elements of v, we write

Example2:

```
>> v=[5 1 9 6 3]
v =
   5    1    9    6    3


>> v(1:3)
ans =
   5    1    9


>> v(3:end)
ans =
   9    6    3
```

```
>> v(:)

ans =

    5

    1

    9

    6

    3
```

1.7 deleting elements from vector

example1:

```
>> a=[6 10 4 1 8 9]

a =

    6    10     4     1     8     9

>> a(2)=[ ]

a =

    6     4     1     8     9

>> a(4)=[]

a =

    6     4     1     9
```

Example2:

```
>> b=[8;1;22;6;9;7]

b =

    8

    1

   22

    6

    9

    7
```

```
>> b(3:5)=[ ]

b =

   8

   1

   7
```

1.8 adding element to the vector

example1:

```
>> v=[1 2 3 4 5]

v =

   1    2    3    4    5


>> v(6)=[6]

v =

   1    2    3    4    5    6
```

1.9 Replacing vector elements

Example1:

```
 >> v=[1 2 3 4 5 ]
v =
   1    2    3    4    5

>> v(1:3)=[8 8 8]
v =
   8    8    8    4    5

>> v(4:end)=[8 8]
v =
   8    8    8    8    8
```

2.1Entering matrix

X=[first row elements; second row element ;third row element]

Example1:

```
X=[1 2 3 4 ;5 6 7 8;9 10 11 12]
X =
   1    2    3    4
   5    6    7    8
   9   10   11   12
```

By using colon operator

X=[first value : final value; first value:final value; first value : final value]

Example 2:

```
>> x=[1:3;3:2:7;10:-2:6]
x =
    1    2    3
    3    5    7
   10    8    6
```

By using <u>zeros</u> function

Zeros (number of rows , number of column)

Example3 :

```
>> zeros(3,2)

ans =

    0    0
    0    0
    0    0
```

Example 4:

```
>> zeros(4,5)
ans =
   0    0    0    0    0
   0    0    0    0    0
   0    0    0    0    0
   0    0    0    0    0
```

By using ones function

ones(number of rows, number of column)

example5:

```
>> ones(3,4)
ans =
   1   1   1   1
   1   1   1   1
   1   1   1   1
```

Example 6:

```
>> ones(5,5)
ans =
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
      1       1   1   1   1
```

2.2Matrix indexing

We select elements in a matrix just as we did for vectors, but now we need two indices. The element of row i and column j of the matrix A is denoted by A(i,j).

Thus, A (i,j) In MATLAB refers to the element A_{ij} of matrix A. The first index is the row number and the second index is the column number. For example, A(1,3) is an element of first row and third column. Here, A(1,3)=3.

Correcting any entry is easy through indexing. Here we substitute A(3,3)=9 by A(3,3)=0. The result is

```
>> A(3,3) = 0
A =
   1   2   3
   4   5   6
   7   8   0
```

2.3 access blocks of matrix elements

example 1:

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

>> y(2,1)
ans =
   1

>> y(2,:)
ans =
   1   4   9
>> y(:,3)
ans =
   8
   9
   2
>> y(:,2:end)
ans =
   6   8
   4   9
   3   2

>> y(1,2:end)
ans =
   6   8
```

Example 2:

```
>> x=[1 2 3 4;5 6 7 8;9 10 11 12]
x =
   1   2   3   4
   5   6   7   8
   9  10  11  12

>> x(1,3:4)
ans =
   3   4
```

```
>> x(3,2:4)
ans =
 10   11   12

>> x(2:3,1)
ans =
    5
    9

>> x(1:2,2:3)
ans =
    2    3
    6    7

>> x(1:end,2:end)
ans =
    2    3    4
    6    7    8
   10   11   12
```

2.4 deleting elements from vector

Example1:

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

   >> m(:,2)=[]
   m =
      3    9    1
      1    4    5
      8    1    2
   >> m(2,:)=[]
   m =
      3    7    9    1
      8    9    1    2
```