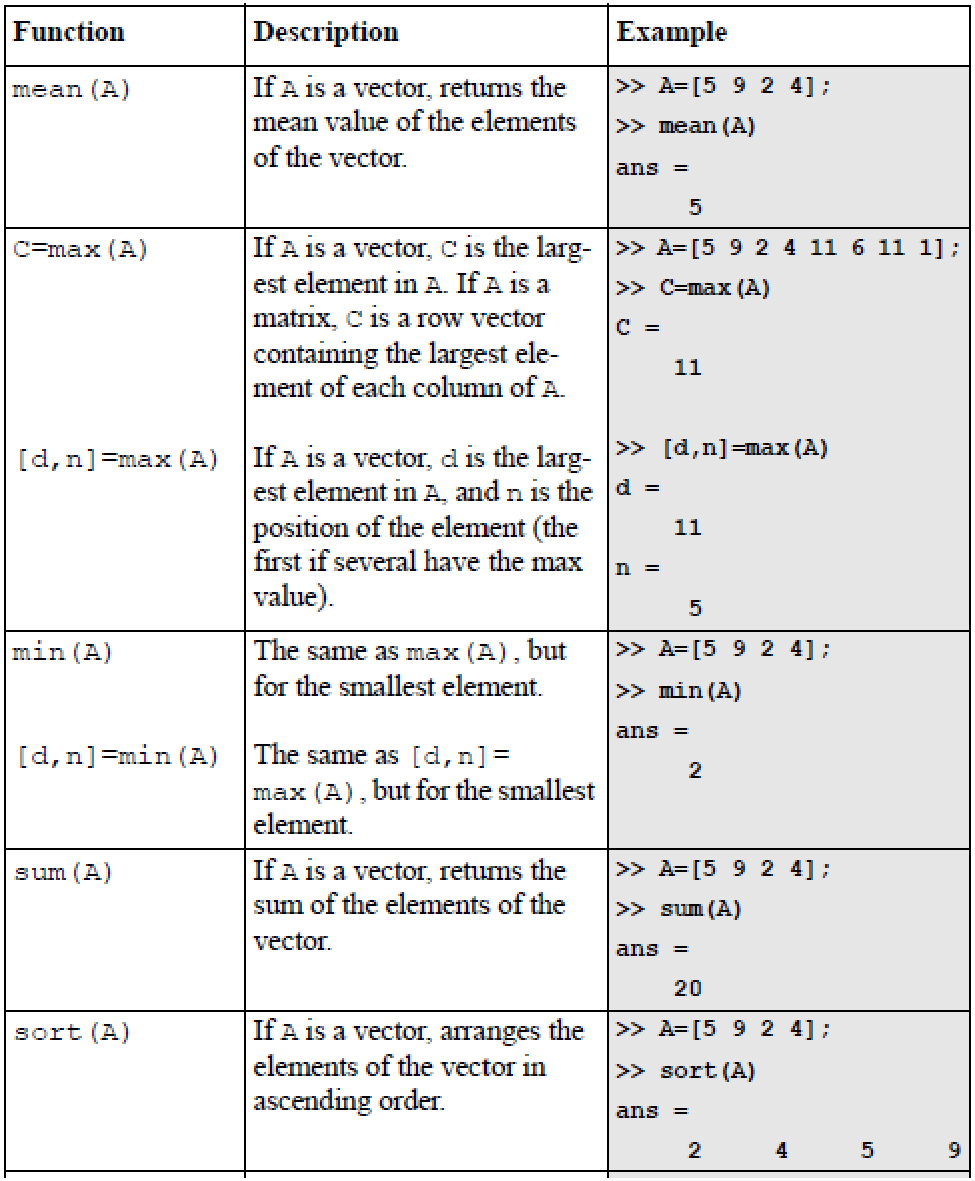
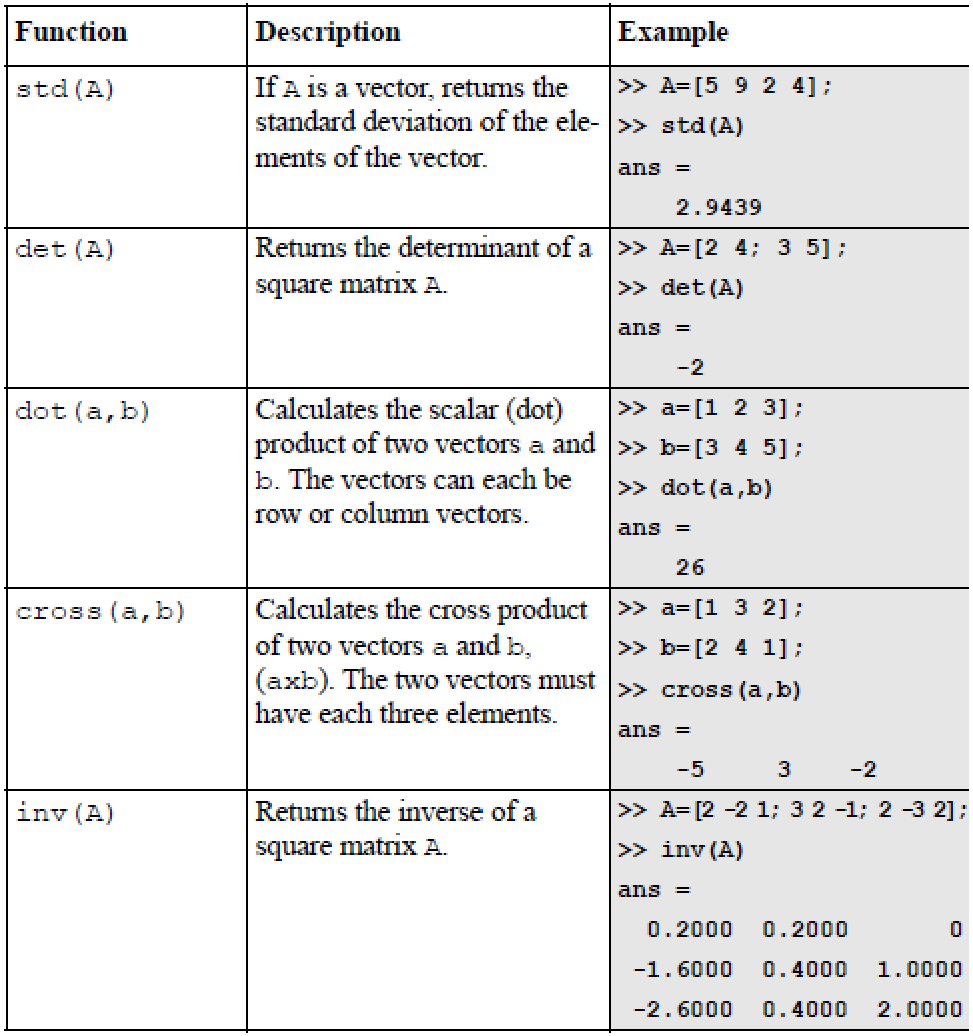


Function	Description	Example
mean(A)	If A is a vector, returns the mean value of the elements of the vector.	>> A=[5 9 2 4]; >> mean(A) ans = 5
C=max(A)	If A is a vector, C is the largest element in A. If A is a matrix, C is a row vector containing the largest element of each column of A.	>> A=[5 9 2 4 11 6 11 1]; >> C=max(A) C = 11
[d,n]=max(A)	If A is a vector, d is the largest element in A, and n is the position of the element (the first if several have the max value).	>> [d,n]=max(A) d = 11 n = 5
min(A)	The same as max(A), but for the smallest element.	>> A=[5 9 2 4]; >> min(A) ans = 2
[d,n]=min(A)	The same as [d,n]= max(A), but for the smallest element.	
sum(A)	If A is a vector, returns the sum of the elements of the vector.	>> A=[5 9 2 4]; >> sum(A) ans = 20
sort(A)	If A is a vector, arranges the elements of the vector in ascending order.	>> A=[5 9 2 4]; >> sort(A) ans = 2 4 5 9



Function	Description	Example

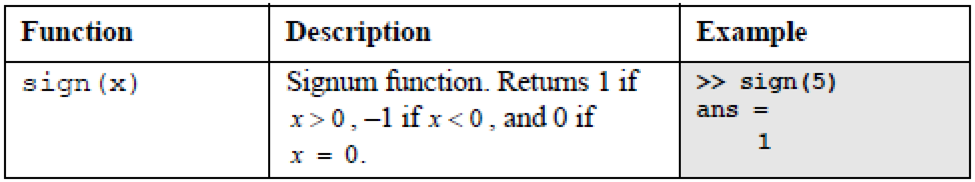
`std(A)`	If `A` is a vector, returns the standard deviation of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> std(A)`
		`ans =`
		`    2.9439`

`det(A)`	Returns the determinant of a square matrix `A`.	`>> A=[2 4; 3 5];`
		`>> det(A)`
		`ans =`
		`    -2`

`dot(a,b)`	Calculates the scalar (dot) product of two vectors `a` and `b`. The vectors can each be row or column vectors.	`>> a=[1 2 3];`
		`>> b=[3 4 5];`
		`>> dot(a,b)`
		`ans =`
		`    26`

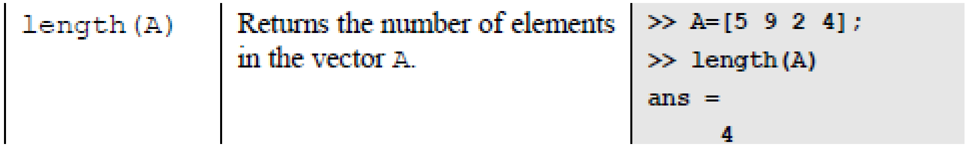
`cross(a,b)`	Calculates the cross product of two vectors `a` and `b`, (`axb`). The two vectors must have each three elements.	`>> a=[1 3 2];`
		`>> b=[2 4 1];`
		`>> cross(a,b)`
		`ans =`
		`    -5     3    -2`

`inv(A)`	Returns the inverse of a square matrix `A`.	`>> A=[2 -2 1; 3 2 -1; 2 -3 2];`
		`>> inv(A)`
		`ans =`
		`   0.2000   0.2000        0`
		`  -1.6000   0.4000   1.0000`
		`  -2.6000   0.4000   2.0000`



Function	Description	Example

`sign(x)`	Signum function. Returns 1 if $x > 0$, –1 if $x < 0$, and 0 if $x = 0$.	`>> sign(5)`
		`ans =`
		`     1`



`length(A)`	Returns the number of elements in the vector `A`.	`>> A=[5 9 2 4];`
		`>> length(A)`
		`ans =`
		`     4`

Control Flow Statements

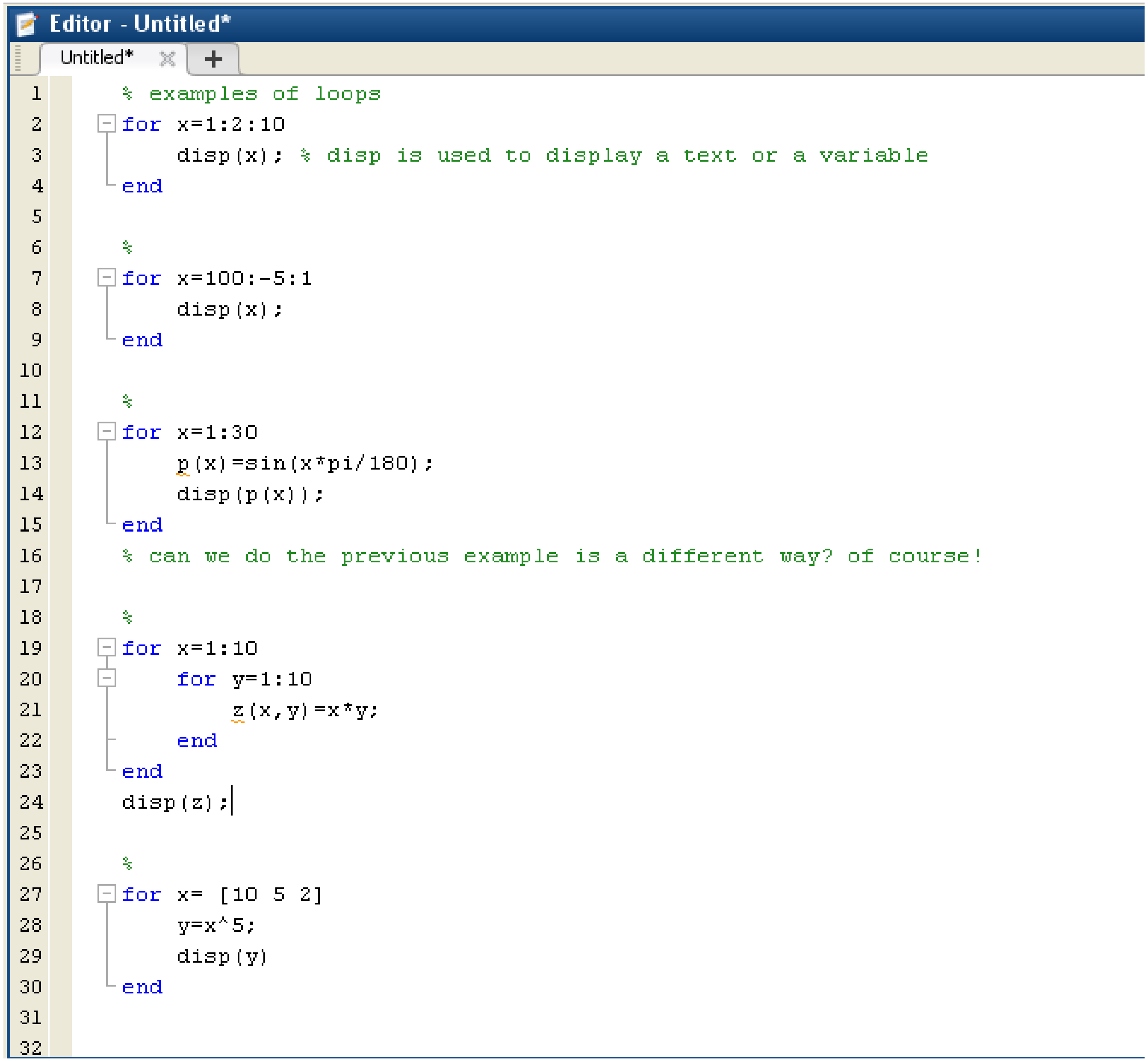
- *For-Loops:*

Allow a group of commands (statements) to be repeated a fixed number of times. The general form of a **for** is:

```
For x= initial_value: increment : End_value

   statement 1
   statement 2
   statement 3
   …………
   statement n

end
```



```
% examples of loops
for x=1:2:10
    disp(x); % disp is used to display a text or a variable
end

%
for x=100:-5:1
    disp(x);
end

%
for x=1:30
    p(x)=sin(x*pi/180);
    disp(p(x));
end
% can we do the previous example is a different way? of course!

%
for x=1:10
    for y=1:10
        z(x,y)=x*y;
    end
end
disp(z);

%
for x= [10 5 2]
    y=x^5;
    disp(y)
end


```

- *while Loops:* evaluate a group of commands (statements) an indefinite number of times when while's condition is true. The general form is:

```
while expression
    statements……
end
```

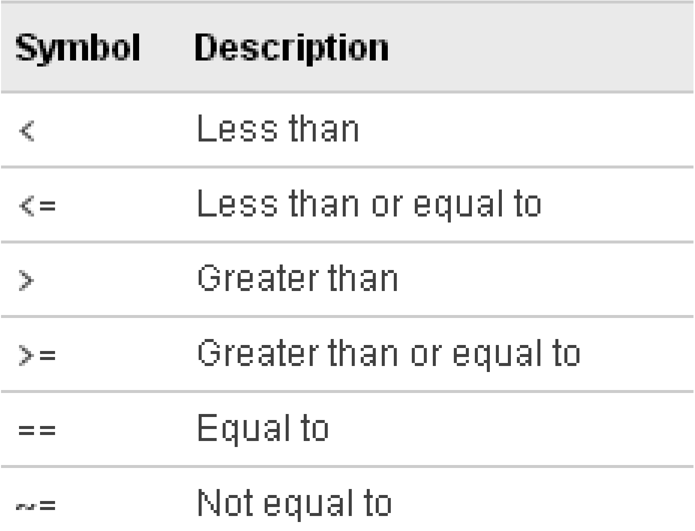
*% the statements between the **while** and **end** are executed as long as ALL elements in expression are true!*

```
% Example !
x=0;
while x<10
     x=x+1;
    disp(x);
 end
```

- *if-else-end:* evaluates an expression, and executes a group of statements when the expression is true. The simplest if-eles-end construction is:

```
If expression
  statements
end
```

- Relational Operators: we use them to compare two arrays of the same size or to compare an array to a scalar.



Symbol	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

How to use them? Easy!! Let’s take an example:

Assume that you have two arrays with the same size **a** and **b**. And we want to compare both arrays if the corresponding elements have a same value or not:

```
  a=[1 2 3 4 5];
  b=[3 4 5 2 3];
  k=a==b
  c=a>b
% Interesting !
```