**Scalar operations**

Scalar operations can be performed on arrays (vectors or matrices). For example, one can perform scalar operations on the following vector *vec*:

>> vec = 1: 0.5: 3

vec =

1.0000 1.5000 2.0000 2.5000 3.0000

Multiplying the vector by 2 performs the multiplication on every element in the vector, resulting in a vector with the same length as the original. Since this is not stored in a variable, the result is stored in the default variable *ans*.

>> vec \* 2

ans =

2 3 4 5 6

All numerical operators can be used in this way. For example, we can add 5 to every element:

>> newv = vec + 5

newv =

6.0000 6.5000 7.0000 7.5000 8.0000

Scalar operations can also be performed on matrices.

>> mat = randi([5, 20], 2, 3)

mat =

17 15 18

20 5 19

>> mat / 2

ans =

8.5000 7.5000 9.0000

10.0000 2.5000 9.5000

**Array operations**

Array operations are operations that are performed on corresponding elements of two arrays, so they must have the same dimensions.

>> vec1 = [2 11 33 5];

>> vec2 = linspace(3, 9, 4)

vec2 =

3 5 7 9

>> vec1 + vec2

ans =

5 16 40 14

For operations that are based on multiplication, the operator must have a dot in front of it. So, the multiplication array operators are:

.^

.\*

./

.\

>> vec2

vec2 =

3 5 7 9

>> vec2 .^ 2

ans =

9 25 49 81

>> mata = [1 3; 2 1]

mata =

1 3

2 1

>> matb = [4 2; 1 5]

matb =

4 2

1 5

>> mata .\* matb

ans =

4 6

2 5

**Matrix multiplication**

So, .\* is array multiplication. The matrix multiplication operator is just \*.

>> matc = mata \* matb

matc =

7 17

9 9

**Logical operators**

Logical operators can be used with arrays, also.

>> vec = randi([1,20], 1,5)

vec =

16 15 8 14 4

>> isless10 = vec < 10

isless10 =

1x5 [logical](matlab:helpPopup('logical')) array

0 0 1 0 1

Logical indexing also works in MATLAB.

>> vec(isless10)

ans =

8 4

>> vec == vec

ans =

1x5 [logical](matlab:helpPopup('logical')) array

1 1 1 1 1

To find out just true/false whether two vectors are equal to each other, use **isequal**.

>> isequal(vec, vec)

ans =

[logical](matlab:helpPopup('logical'))

1

**Arrays as Function Arguments**

Entire arrays can be passed to functions. The function will evaluate each of the elements in the array and return an array with the same dimensions as the original.

>> myvec = [-3: 2]

myvec =

-3 -2 -1 0 1 2

>> abs(myvec)

ans =

3 2 1 0 1 2

>> mymat = randi([0, 5], 2, 4)

mymat =

0 5 3 4

2 2 1 1

>> sqrt(mymat)

ans =

0 2.2361 1.7321 2.0000

1.4142 1.4142 1.0000 1.0000

>> x = linspace(-2\*pi, 2\*pi, 6);

>> y = sin(x)

y =

0.0000 0.5878 -0.9511 0.9511 -0.5878 -0.0000

**Functions that change array dimensions**

The **reshape** function can change the dimensions of any array to any other array that has the same number of elements.

>> ranmat = randi([1,20], 2,4)

ranmat =

9 16 14 17

19 20 1 19

>> reshape(ranmat,4,2)

ans =

9 14

19 1

16 17

20 19

The **rot90** function rotates a matrix 90 degrees counterclockwise.

>> ranmat

ranmat =

9 16 14 17

19 20 1 19

>> rot90(ranmat)

ans =

17 19

14 1

16 20

9 19

There are several functions that flip matrices. The **fliplr** function flips a row vector or the columns of a matrix from left to right.

>> vec = 2:3:14

vec =

2 5 8 11 14

>> fliplr(vec)

ans =

14 11 8 5 2

The **flipud** function flips a column vector or the rows of a matrix up to down.

>> vec = 2:3:14

vec =

2 5 8 11 14

>> fliplr(vec)

ans =

14 11 8 5 2

The **flip** function flips a row vector left to right or a column vector or matrix up to down.

There are functions that replicate a matrix or element from a matrix. The **repmat** function replicates an entire matrix. The following replicates a matrix variable *mymat* 6 times, as a 2 x 3 matrix of mymats.

>> mymat = [1 2; 3 4]

mymat =

1 2

3 4

>> repmat(mymat,2,3)

ans =

1 2 1 2 1 2

3 4 3 4 3 4

1 2 1 2 1 2

3 4 3 4 3 4

The **repelem** function replicates each element, in this case as a 2 x 3 matrix of each element.

>> mymat = [1 2; 3 4]

mymat =

1 2

3 4

>> repelem(mymat, 2,3)

ans =

1 1 1 2 2 2

1 1 1 2 2 2

3 3 3 4 4 4

3 3 3 4 4 4

**Array Functions and Statistical Functions**

There are functions that perform statistical analyses on vectors, including:

* **min**: minimum value
* **max**: maximum value
* **mean**: average
* **mode**: number that appears most frequently
* **median**: number in the middle of a sorted vector
* **std**: standard deviation
* **var**: variance
* **sum**: sum

For example,

>> vec = [5 33 11 2 7 9 4]

vec =

5 33 11 2 7 9 4

>> min(vec)

ans =

2

>> mean(vec)

ans =

10.1429

For matrices, all of these functions work column-wise:

>> mat = randi([1, 10], 3, 5)

mat =

4 4 7 7 1

2 6 3 8 3

8 2 7 5 10

>> max(mat)

ans =

8 6 7 8 10

>> sum(mat)

ans =

14 12 17 20 14

So, notice that the result is a row vector that contains the result of the function for each column. To get an overall, for example overall maximum, get the max of this row vector:

>> max(max(mat))

ans =

10

>> sum(sum(mat))

ans =

77

Other similar functions include:

* **prod**: product
* **cumsum**: cumulative sum
* **cumprod**: cumulative product
* **cummin**: cumulative minimum
* **cummax**: cumulative maximum

These can similarly work on a vector or the columns of a matrix.

>> rvec = 2:5

rvec =

2 3 4 5

>> prod(rvec)

ans =

120

>> rvec = [rvec 11]

rvec =

2 3 4 5 11

>> cumsum(rvec)

ans =

2 5 9 14 25

>> mat = randi([0,5], 3, 4)

mat =

0 4 3 1

1 3 1 4

3 2 4 1

>> prod(mat)

ans =

0 24 12 4

>> cummin(mat)

ans =

0 4 3 1

0 3 1 1

0 2 1 1

The **diff** function returns differences between consecutive elements in a vector.

>> vec = [5 33 11 2 7]

vec =

5 33 11 2 7

>> diff(vec)

ans =

28 -22 -9 5

Note that the result has one fewer element than the original vector.

The **sort** function sorts a vector, or columns of a matrix.

>> mat

mat =

0 4 3 1

1 3 1 4

3 2 4 1

>> sort(mat)

ans =

0 2 1 1

1 3 3 1

3 4 4 4

**Square Matrices**

There are functions that operate on square matrices. The **diag** function returns the diagonal of a matrix, or creates a diagonal matrix by putting a vector on the diagonal.

>> mydm = diag([2:4])

mydm =

2 0 0

0 3 0

0 0 4

>> diag(mydm)

ans =

2

3

4

The trace of a square matrix is the sum of the diagonal; the **trace** function will return this.

>> trace(mydm)

ans =

9

The **eye** function creates an Identity matrix.

>> eye(4)

ans =

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

There are “is” functions that ask True/False questions about square matrices. The **isdiag** function returns 1 for true if a matrix is a diagonal matrix.

>> isdiag(mydm)

ans =

logical

1

The **issymmetric** function returns logical 1 if the matrix is a symmetric matrix.

>> smat = [1 2 3; 2 7 4; 3 4 5]

smat =

1 2 3

2 7 4

3 4 5

>> issymmetric(smat)

ans =

logical

1