Chapter 6 Arrays and Array Operations

6.1 Simple Arrays

Arrays are rectangular entities consisting of rows and columns. Arrays are input a row at a time with individual elements separated by a space or a comma. A new row is begun following a semicolon. The complete set of entries are enclosed between square brackets.

Example 6.1.1

```
x=[1 3 5 7 9]

x =     1     3     5     7     9

Y=[0, 3+4*i; 1-i,2]

Y = 0                   3.0000 + 4.0000i
    1.0000 - 1.0000i    2.0000

a=1; b=2; c=3; d=4; e=5;, f=6;
Z=[a a+b a-c; d e+2*f e^b]

Z =  1     3    -2
     4    17    25
```

The power of MATLAB is its ability to process entire arrays as a basic data type in a single operation.

Example 6.1.2

```
x=[1 3 5 7 9] % Create array x
y=x-5 % Subtract 5 from each element in x and store result in array y

x =     1     3     5     7     9
y =    -4    -2     0     2     4

z=x+y % Add arrays x and y element by element

z =    -3     1     5     9    13

z=5*z + 1 % Multiply all elements in array z by 5 and then add 1 to
          % each element and store result in array z

z =   -14     6    26    46    66
```

Example 6.1.3

```
theta=[0 pi/4 pi/3 pi/2; pi 5*pi/4 4*pi/3 3*pi/2]
r=cos(theta) % Compute cosine of all elements in array theta and store
             % result in array r
```

```
theta =  0         0.7854    1.0472    1.5708
         3.1416    3.9270    4.1888    4.7124

r =  1.0000    0.7071    0.5000    0.0000
    -1.0000   -0.7071   -0.5000   -0.0000
```

6.2 Array Addressing or Indexing

Individual elements in an array are addressed by using subscripts to identify them. A single subscript is used for vectors, i.e. arrays with a single row or column. Double subcripts are required for 2 dimensional arrays.

Example 6.2.1

```
a=[0 1 3 6 10] % Create row vector a
a(1),a(5) % Display 1st and 5th elements of a

a =   0     1     3     6    10

ans =  0
ans = 10

b=[0;1;3;6;10] % Create column vector b
b(2),b(4) % Display 2nd and 4th elements of b

b =  0
     1
     3
     6
    10

ans = 1
ans = 6

x=a(3)+b(3)
y=a(1)-b(5)

x =   6
y = -10

C=[1 2 3 4 5;6 7 8 9 10;11 12 13 14 15] % Create 3 by 5 array C
C(2,3),C(3,5) % Display elements C(2,3),C(3,5)

C =  1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15

ans =  8
ans = 15
```

A block of continuous elements can be addressed using a colon between the first and last elements.

Example 6.2.2

```
x=[1,2,3,4,5; 6,7,8,9,10; 11,12,13,14,15; 16,17,18,19,20] % Create x
y=x(1,2:4) % Selects elements x(1,2),x(1,3),x(1,4)
```

```
x =  1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20

y =  2     3     4

x % Display array x
z=x(2:end,3)  % Selects elements x(2,3),x(3,3),x(4,3)

x =  1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20

z =  8
    13
    18
```

Note the array `y` is 1×3 and the array `z` is 3×1 and therefore `y+z` is an illegal operation. A double colon is used to address non-continuous elements.

Example 6.2.3

```
A=[1 2 3 4 5 6 7 8 9 10;10 9 8 7 6 5 4 3 2 1;0 2 4 6 8 10 12 14 16 18]
B=A(2,1:2:7)  % Selects elements A(2,1),A(2,3),A(2,5),A(2,7)

A =  1     2     3     4     5     6     7     8     9    10
    10     9     8     7     6     5     4     3     2     1
     0     2     4     6     8    10    12    14    16    18

B = 10     8     6     4

A % Display A
C=A(3,1:3:10)  % Selects elements A(3,1),A(3,4),A(3,7),A(3,10)

A =  1     2     3     4     5     6     7     8     9    10
    10     9     8     7     6     5     4     3     2     1
     0     2     4     6     8    10    12    14    16    18

C =  0     6    12    18

A % Display A
D=A(1,10:-2:4)  % Selects elements A(1,10),A(1,8),A(1,6),A(1,4)

A =  1     2     3     4     5     6     7     8     9    10
    10     9     8     7     6     5     4     3     2     1
     0     2     4     6     8    10    12    14    16    18

D = 10     8     6     4
```

6.3 Array Construction

For large arrays with linearly spaced values, construction is simplified by the use of colon notation or the MATLAB function `'linspace'` . The syntax for colon notation is:

```
x = Initial Value:Increment:Final Value
```

Example 6.3.1

```
x=0:5:100

x = Columns 1 through 12
     0     5    10    15    20    25    30    35    40    45    50    55
    Columns 13 through 21
    60    65    70    75    80    85    90    95   100

y=10:-2:-10

y = 10     8     6     4     2     0    -2    -4    -6    -8   -10

z=25:0.5:30

z = Columns 1 through 7
    25.0000   25.5000   26.0000   26.5000   27.0000   27.5000   28.0000
    Columns 8 through 11
    28.5000   29.0000   29.5000   30.0000

a=15;
t=a:a+5

t = 15    16    17    18    19    20
```

Note, in the last example, the increment defaults to one when only two values are present. The closing brackets are not required unless the array is part of a larger array.

Example 6.3.2

```
x=[ 1 2 10:2:20 30]

x = 1     2    10    12    14    16    18    20    30

y=[1:10; 0 0 50:2:64]

y = 1     2     3     4     5     6     7     8     9    10
    0     0    50    52    54    56    58    60    62    64
```

The `'linspace'` operator is used to create arrays consisting of a specified number of elements ranging from an initial value to a final value. The syntax is

```
x=linspace(Initial Value, Final Value, Number of points)
```

Example 6.3.3

```
x=linspace(0,2*pi,20)

x = Columns 1 through 7
    0         0.3307    0.6614    0.9921    1.3228    1.6535    1.9842
    Columns 8 through 14
    2.3149    2.6456    2.9762    3.3069    3.6376    3.9683    4.2990
    Columns 15 through 20
    4.6297    4.9604    5.2911    5.6218    5.9525    6.2832

A=[linspace(0,5,7);linspace(-1,1,7)]

A =  0         0.8333    1.6667    2.5000    3.3333    4.1667    5.0000
    -1.0000   -0.6667   -0.3333         0    0.3333    0.6667    1.0000
```

Arrays with logarithmically spaced values are created with the `'logspace'` function. In this case the exponents of 10 are equally spaced.

Example 6.3.4

```
y=logspace(2,6,5) % Create 5 element array of values from 10^2 to 10^6

y = 100        1000       10000      100000     1000000

z=logspace(0,3,6) % Create 1´6 array of values from 10^0 to 10^3

z = 1.0e+003 *
    0.0010    0.0040    0.0158    0.0631    0.2512    1.0000
```

6.4 Array Orientation

Row vectors are arrays with a single row. Column vectors are arrays with a single column. Column vectors are created directly using the semicolon to assign values to consecutive rows or alternatively by transposing a single-row array. The transpose operator is (').

Example 6.4.1

```
y=[1;2;3;4;5;6;7] % Create a 7´1 array or column vector

y =  1
     2
     3
     4
     5
     6
     7

x=1:7 % Create row vector x
x=x' % Transpose row vector x to column vector x

x =  1     2     3     4     5     6     7

x =  1
     2
     3
     4
     5
     6
     7
```

Rectangular arrays are transposed with the same operator.

Example 6.4.2

```
A=[1:4;5:8;9:12] % Create 3´4 array A
B=A' % Transpose A into 4´3 array B by swapping rows and columns of A

A =  1     2     3     4
     5     6     7     8
     9    10    11    12

B =  1     5     9
     2     6    10
     3     7    11
     4     8    12
```

For complex arrays, i.e. arrays with complex numbers, the transpose operator produces the complex conjugates in a transposed array.

Example 6.4.3

```
x=[1+2*i, 3*i, 5, 4-9*i]
y=x'

x =  1.0000 + 2.0000i     0 + 3.0000i     5.0000     4.0000 - 9.0000i

y =  1.0000 - 2.0000i
     0      - 3.0000i
     5.0000
     4.0000 + 9.0000i
```

The dot-transpose operator (**.'**) transposes the array but does not conjugate the complex numbers.

Example 6.4.3

```
x=[12-i; 3; 2+5*i;]
y=x.'

x = 12.0000 - 1.0000i
     3.0000
     2.0000 + 5.0000i

y = 12.0000 - 1.0000i     3.0000      2.0000 + 5.0000i

x=[1+i,1-i; 2+3*i,5-2*i; 4*i,8+2*i]
y=x' % Transpose and conjugate x
z=x.' % Transpose x

x = 1.0000 + 1.0000i   1.0000 - 1.0000i
    2.0000 + 3.0000i   5.0000 - 2.0000i
    0      + 4.0000i   8.0000 + 2.0000i

y = 1.0000 - 1.0000i   2.0000 - 3.0000i        0 - 4.0000i
    1.0000 + 1.0000i   5.0000 + 2.0000i   8.0000 - 2.0000i

z = 1.0000 + 1.0000i   2.0000 + 3.0000i        0 + 4.0000i
    1.0000 - 1.0000i   5.0000 - 2.0000i   8.0000 + 2.0000i
```

6.5 Scalar-Array Mathematics

Mathematical operations involving scalars and arrays apply to every element of the array. For example, scalar multiplication of an array results in every element of the array multplied by the scalar. The same holds for addition, subtraction and division.

Example 6.5.1

```
x=0:5
y=(pi/4)*x
z=y/(pi/4)
u=[y;z]
w=2*u+pi

x =  0    1     2     3     4     5
y =  0    0.7854    1.5708    2.3562    3.1416    3.9270
z =  0    1     2     3     4     5

u =  0    0.7854    1.5708    2.3562    3.1416    3.9270
     0    1.0000    2.0000    3.0000    4.0000    5.0000

w =  3.1416    4.7124    6.2832    7.8540    9.4248   10.9956
     3.1416    5.1416    7.1416    9.1416   11.1416   13.1416
```

Be careful not to attempt scalar-array operations similar to `'a/x'` where `a` is a scalar and `x` is an array. If you do, expect to see an error message.

Example 6.5.2

```
x=0:pi/2:2*pi
y=1/x % Invalid - Cannot divide scalar by an array

x =    0    1.5708    3.1416    4.7124    6.2832
 ??? Error using ==> /
Matrix dimensions must agree.
```

It is possible to construct a new array where each element is numerically equal to the scalar `a` divided by the corresponding element in array `x`. This will be demonstrated in the following section. Similarly, expressions such as `'a^x'` and `'x^a'` are incorrect, unless `x'` is a square array, because MATLAB interprets them as invalid matrix operations.

Example 6.5.3

```
A=[0:2:8;1:2:9] % A is 2´3

A =  0     2     4     6     8
     1     3     5     7     9
```

```
B=A^2

 ??? Error using ==> ^
Matrix must be square.

C=2^A

 ??? Error using ==> ^
Matrix must be square.
```

The correct syntax to achieve the desired results in Example 6.5.3 are likewise deferred until the next section.

6.6 Array-Array Mathematics

Mathematical operations involving several arrays or matrices of like size can be performed on an element by element basis. For example if A and B are $m \times n$ arrays (m and n different) with typical element a_{ij} and b_{ij}, then 'A+B' and 'A-B' produce what would normally be expected, i.e. $m \times n$ matrices equal to the sum and difference of the given matrices. Suppose an $m \times n$ matrix C is required where the typical element c_{ij} is equal to $a_{ij} \cdot b_{ij}$. The MATLAB statement 'C = A*B' is incorrect and in fact will not even execute. The reason is quite simple. MATLAB interprets the expression 'A*B' as a matrix multiplication and in this case the matrix product of A and B is not defined. If the indices m and n are equal, i.e. A and B are square matrices, the expression 'A*B' will result in the correct product matrix, still not the desired result. The dilemma is solved by inserting a period (.) before the multiplication operator, instructing MATLAB to perform the element by element operation (multiplication in this example).

Example 6.5.1

```
A=[1 2 3; 4 5 6; 7 8 9; 10 11 12] % A is 4´3
B=[12 11 10; 9 8 7; 6 5 4; 3 2 1] % B is 4´3

A =  1     2     3
     4     5     6
     7     8     9
    10    11    12

B = 12    11    10
     9     8     7
     6     5     4
     3     2     1

C=A*B

 ??? Error using ==> *
Inner matrix dimensions must agree.

C=A.*B

C = 12    22    30
    36    40    42
    42    40    36
    30    22    12
```

Constructing the matrix C where $c_{ij} = a_{ij}/b_{ij}$ is achieved in a similar way, namely by using the './' operator to specify the element by element operations required to achieve the desired result.

Example 6.5.2

```
A=[0 1 2; 3 4 5; 6 7 8] % A is 3´3
B=[1 1 1;  2 2 2; 3 3 3] % B is 3´3
C=A./B % Perform array-array division

A =  0     1     2
     3     4     5
     6     7     8

B =  1     1     1
     2     2     2
     3     3     3

C = 0         1.0000    2.0000
    1.5000    2.0000    2.5000
    2.0000    2.3333    2.6667
```

Array, i.e. element by element operations involving exponentiation are possible as well using the `'.^'` operator.

Example 6.5.3

```
x=[1:5] % x=[1 2 3 4 5]
u=3.^x % u=[3^1 3^2 3^3 3^4 3^5]
w=x.^3 % w=[1^3 2^3 3^3 4^3 5^3]

x =  1     2     3     4     5
u =  3     9    27    81   243
w =  1     8    27    64   125

x % x=[1 2 3 4 5]
y=[5:-1:1] % y=[5 4 3 2 1]
z=x.^y % z=[1^5 2^4 3^3 4^2 5^1

x =  1     2     3     4     5
y =  5     4     3     2     1
z =  1    16    27    16     5
```

6.7 Standard Arrays

Certain arrays occur often enough to justify a simpler process for creating them. For example, arrays with all ones or all zeros are created using the `'ones'` and `'zeros'` functions. The following example illustrates their use.

Example 6.7.1

```
A=ones(2,3) % Create 2´3 array of all 1's

A =  1     1     1
     1     1     1

y=zeros(1,5) % Create 1´5 array of all 0's

y =  0     0     0     0     0

B=ones(4) % Create 4´4 array of all 1's

B =  1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

C=zeros(3) % Create 3´3 array of all 0's

C =  0     0     0
     0     0     0
     0     0     0

D=[ones(3) zeros(3); zeros(3) ones(3)]% Create 6´6 array with sub-arrays

D =  1     1     1     0     0     0
     1     1     1     0     0     0
     1     1     1     0     0     0
     0     0     0     1     1     1
     0     0     0     1     1     1
     0     0     0     1     1     1
```

The identity matrix and others similar to it, but not square, are easily formed.

Example 6.7.2

```
I=eye(3) % Create 3´3 Identity matrix
I24=eye(2,4) % Create 2´4 array with embedded 2´2 Identity matrix
I42=eye(4,2) % Create 4´2 array with embedded 2´2 Identity matrix

I =  1     0     0
     0     1     0
     0     0     1
```

```
I24 = 1     0     0     0
      0     1     0     0

I42 = 1     0
      0     1
      0     0
      0     0
```

Arrays of random numbers uniformly distributed from 0 to 1 and standard normal numbers are constructed with MATLAB functions `'rand'` and `'randn'`.

Example 6.7.3

```
x=rand(1,7) % Create 1´7 row vector of 7 U(0,1) random numbers
y=10+(15-10)*x % Transform x to array of 7 U(10,15) random numbers

x =  0.1341    0.0653    0.3751    0.3735    0.4840    0.9695    0.3421
y = 10.6704   10.3266   11.8757   11.8676   12.4201   14.8473   11.7103

A=rand(4) % Create 4´4 array of U(0,1) random numbers

A = 0.2527    0.4864    0.8578    0.1030
    0.5849    0.4961    0.6098    0.1583
    0.5237    0.8432    0.5657    0.4136
    0.1634    0.8062    0.6119    0.5604

y=randn(10,1) % Create 10´1 column vector of N(0,1) Normally
              % distributed random numbers
z=50+3*y % Transform y to 10´1 column vector of N(50,3) Normally
         % distributed random numbers

y = -0.4326
    -1.6656
     0.1253
     0.2877
    -1.1465
     1.1909
     1.1892
    -0.0376
     0.3273
     0.1746

z = 48.7023
    45.0032
    50.3760
    50.8630
    46.5606
    53.5727
    53.5675
    49.8871
    50.9819
   50.5239
```

```
B=randn(4) % Create 4´4 array of N(0,1) Normal deviates

B = -0.1867   -0.1364   -0.0956    0.7143
     0.7258    0.1139   -0.8323    1.6236
    -0.5883    1.0668    0.2944   -0.6918
     2.1832    0.0593   -1.3362    0.8580
```

Diagonal arrays are created from row arrays using the elements of the row array to populate the principal diagonal as shown below.

Example 6.7.4

```
x=[1 5 2 0 8];
A=diag(x) % Create diagonal array with elements of x on diagonal

A =  1     0     0     0     0
     0     5     0     0     0
     0     0     2     0     0
     0     0     0     0     0
     0     0     0     0     8
```

6.8 Array Manipulation

Arrays are modified and manipulated by addressing the appropriate cells and inserting the desired values. The follwing examples demonstrate array manipulation in MATLAB.

Example 6.8.1

```
clear
A(3,3)=0 % Create 3´3 array A and fill with 0's
A(2,2)=5 % Change element A(2,2) to 5
A(1,3)=-1 % Change element A(1,3) to -1
A(4,5)=10 % Expand A to a 4×5 array and set value of A(4,5) to 10
```

```
A =  0     0     0
     0     0     0
     0     0     0

A =  0     0     0
     0     5     0
     0     0     0

A =  0     0    -1
     0     5     0
     0     0     0

A =  0     0    -1     0     0
     0     5     0     0     0
     0     0     0     0     0
     0     0     0     0    10
```

Note, if array 'A' does not currently exist in the MATLAB Workspace, the assignment `'A(3,3) = 0'` creates a 3×3 array with all zeros. The last statement `'A(4,5)=10'` expands the original 3×3 array in size to a 4×5 array with extra zeros inserted and sets the A(4,5) element to 10.

The colon is an effective way to create or modify entire rows and columns (or parts thereof). The line `'A(3,:)=0'` in the example below is equivalent to `'A(3,1:5)=0'`.

Example 6.8.2

```
A=diag(0:4) % Create a 5 ´ 5 diagonal array with elements 0,1,2,3,4
A(3,:)=0 % Set the 3rd row of A equal to all zeros
A(2:4,5)=-8 % Set A(2,5), A(3,5) and A(4,5) equal to -8
```

```
A =  0     0     0     0     0
     0     1     0     0     0
     0     0     2     0     0
     0     0     0     3     0
     0     0     0     0     4
```

```
A =  0     0     0     0     0
     0     1     0     0     0
     0     0     0     0     0
     0     0     0     3     0
     0     0     0     0     4

A =  0     0     0     0     0
     0     1     0     0    -8
     0     0     0     0    -8
     0     0     0     3    -8
     0     0     0     0     4

B=[1:5;6:10;11:15;16:20;21:25] % Create 5´5 array B
B(2:3,4:5)=0 % Set elements B(2,4), B(2,5), B(3,4), B(3,5) to zero

B =  1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20
    21    22    23    24    25

B =  1     2     3     4     5
     6     7     8     0     0
    11    12    13     0     0
    16    17    18    19    20
    21    22    23    24    25

C=B(:,[1 3 5]) % Set C equal to 5´3 array made up of the 1st, 3rd
               % and 5th columns of array B

C =  1     3     5
     6     8     0
    11    13     0
    16    18    20
    21    23    25

D=C(3:4,2:3) % Set D equal to 2´2 array using elements in 3rd and
             % 4th rows, and 2nd and 3rd columns of C

D = 13     0
    18    20

E=C(:) % Create a 15´1 array E made up of the 3 columns of array C

E =  1
     6
    11
    16
    21
     3
     8
    13
    18
    23
     5
     0
```

```
     0
    20
    25
```

Entire rows and columns from an array are easily deleted by assigning them equal to an empty array, i.e. `[]`.

Example 6.8.3

```
A=[1 2 3 4; 5 6 7 8; 9 10 11 12;13 14 15 16] % Create 4´4 array A
A(2,:)=[] % Delete the 2nd row of array A making it 3´4

A =  1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

A =  1     2     3     4
     9    10    11    12
    13    14    15    16

B=A % Save 3´4 array A in array B
C=B' % Create 4´3 array C as the transpose of 3×4 array B
A(:,3)=[] % Delete the 3rd column of A making it 3×3
x=-5:5:5 % Create 1×3 array x
A(2,:)=x % Replace 2nd row of array A with array x

B =  1     2     3     4
     9    10    11    12
    13    14    15    16

C =  1     9    13
     2    10    14
     3    11    15
     4    12    16

A =  1     2     4
     9    10    12
    13    14    16

x = -5     0     5

A =  1     2     4
    -5     0     5
    13    14    16
```

Elements of a two dimensional array can be addressed using a single index. MATLAB counts the elements starting in the first column and continues down subsequent columns until the index value is reached.

Example 6.8.4

```
A=[1:4;8:-1:5; 0 -3 9 12] % Create 3×4 array A
x=A(3) % Finds the A(3,1) element of array A
y=A(7) % Finds the A(1,3) element of A
z=A(end) % Finds the A(3,4) element of array A
u=A(5:10) % Finds elements A(2,2),A(3,2),A(1,3),A(2,3),A(3,3),A(1,4)

A =  1     2     3     4
     8     7     6     5
     0    -3     9    12

x =  0

y =  3

z = 12

u =  7    -3     3     6     9     4
```

A new data type in MATLAB called a logical array can be an effective way to adddress elements of an array that have a common property.

Example 6.8.5

```
x=100+10*randn(1,7) % Generate array of random numbers ~ N(100,10)
xgt98=x>98 % Create logical array xgt98 same size as x with 1's (True)
           % in positions corresponding to wherever element in x >98
           % and 0's (False) where element in x <=98
y=x(xgt98) % Find the elements in array x which are > 98


x = 112.5400   84.0627   85.5904  105.7115   96.0011  106.9000  108.1562
xgt98 =  1     0     0     1     0     1     1
y =  112.5400  105.7115  106.9000  108.1562
```

Logical arrays differ from numeric arrays and can be used to address other arrays of the same size while numeric arrays consisting of 1's and 0's cannot.

Example 6.8.6

```
x=1:2:15 % Create 1´8 array x
a=[2 5 7] % Create index vector 'a'
w=x(a) % Address x using index vector 'a' and store in 1 by 3 array w

x =  1     3     5     7     9    11    13    15
a =  2     5     7
w =  3     9    13


x36=(x>=3 & x<=6)% Create 1´8 logical array 'x36' with 1's where
                 % elements in x are betwen 3 and 6 and 0's elsewhere
y=x(x36) % Find elements (numerical values) in x between 3 and 6
```

```
x36 =  0     1     1     0     0     0     0     0
y =    3     5
```

```
b=[1 0 1 0 1 0 0 0] % Create 1´8 numerical array b
x=0:5:35
%z=x(b) % Invalid MATLAB statement because b is not a logical array
Lb=logical(b) % Convert numerical array b to a logical array Lb
z=x(Lb) % Extracts the 1st, 3rd and 5th elements of array x
```

```
b =  1     0     1     0     1     0     0     0
x =  0     5    10    15    20    25    30    35
Lb = 1     0     1     0     1     0     0     0
z =  0    10    20
```

6.9 Subarray Searching

The `'find'` function returns the indices or subscripts of elements in an array that satisfy a relational expression.

Example 6.9.1

```
x=100*rand(1,7) % Create array x of 7 random numbers beteen 0 and 100
n=find(x>50) % Find the indices of elements in x which are > 50

x = 26.8677   78.4254   38.7871    3.0984   58.5502   55.8559   20.0696
n =  2     5     6

A=[1:4;2:5;3:6;4:7] % Create 4´4 array A
[i,j]=find(rem(A,3)==0) % Find subcripts of elements in A which are
                        % divisible by 3

A =  1     2     3     4
     2     3     4     5
     3     4     5     6
     4     5     6     7

i =  3    j =  1
     2         2
     1         3
     4         3
     3         4

A=[-5:2:5;-2:3] % Create 2´6 array A
LB=abs(A)>2 % Create logical array LB with 1's at locations where
            % absolute value of elements of A are > 2 and 0's elsewhere
[i,j]=find(LB) %Find row and column indices pointing to 1's in array LB

A = -5    -3    -1     1     3     5
    -2    -1     0     1     2     3

LB = 1     1     0     0     1     1
     0     0     0     0     0     1

i =  1    j =  1
     1         2
     1         5
     1         6
     2         6
```

6.10 Array Manipulation Functions

MATLAB provides several utility functions for manipulation and creation of new arrays. Some of them rearrange the elements keeping the original array dimensions in tact while others convert arrays from one size to another with the same elements. Several examples are shown.

Example 6.10.1

```
A=[1 2 3 ;4 5 6; 7 8 9; 10 11 12]
B=flipud(A) % Flips array A upside down
C=fliplr(A) % Flips array A in left right direction
D=rot90(A) % Roates elements in array A 90 degrees counterclockwise
E=reshape(A,2,6) % Reshapes array into a 2´6 array
F=reshape(A,1,12) % Reshapes array A into a 1´12 array

A =  1     2     3
     4     5     6
     7     8     9
    10    11    12

B = 10    11    12
     7     8     9
     4     5     6
     1     2     3

C =  3     2     1
     6     5     4
     9     8     7
    12    11    10

D =  3     6     9    12
     2     5     8    11
     1     4     7    10

E =  1     7     2     8     3     9
     4    10     5    11     6    12

F =  1     4     7    10     2     5     8    11     3     6     9    12
```

Its possible to extract the principal diagonal from a square array as well as create a diagonal array from a vector.

Example 6.10.2

```
A=[1 5 8; 2 7 3; 4 9 5] % Create 3´3 array A
d=diag(A) % Store the principal diagonal of matrix A in vector d
B=diag(d) % Create a diagonal matrix with principal diagonal taken
          % from the elements of vector d

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

```
d =  1
     7
     5

B =  1     0     0
     0     7     0
     0     0     5
```

```
X=[1 0 -1; 0 2 -2; 3 -4 5] % Create 3´3 array X
C=diag(diag(X)) % Find the principal diagonal of X and use it to
                % create a diagonal matrix C
```

```
X =  1     0    -1
     0     2    -2
     3    -4     5

C =  1     0     0
     0     2     0
     0     0     5
```

Note in the above example that the function `'diag'` accepts either a vector or square array and produces the appropriate result. In fact, the array need not be square and the result will be a minor diagonal.

Triangular arrays are formed using `'triu'` and `'tril'`.

Example 6.10.3

```
A=ones(4) % Create 4´4 Identity matrix
B=triu(A) % Create upper triangular matrix from A
C=tril(A) % Create lower triangular matrix from A
D=B+C
```

```
A =  1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

B =  1     1     1     1
     0     1     1     1
     0     0     1     1
     0     0     0     1

C =  1     0     0     0
     1     1     0     0
     1     1     1     0
     1     1     1     1

D =  2     1     1     1
     1     2     1     1
     1     1     2     1
     1     1     1     2
```

An array can be replicated to construct a larger array by using the `'repmat'` function.

```
A=[1:3;4:6] % Create 2´3 array A
B=repmat(A,1,3) % Replicate A 1 ´ 3 times
C=repmat(A,2,2) % replicate A 2 ´ 2 times
```

```
A =  1     2     3
     4     5     6

B =  1     2     3     1     2     3     1     2     3
     4     5     6     4     5     6     4     5     6

C =  1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6
```

6.11 Array Size

There are two very useful MATLAB functions which allow one to determine the size of an array or vector. The first is `'size'` which returns the dimensions of a given array.

Example 6.11.1

```
m=ceil(10*rand) % Round up random number from 1 to 10
n=ceil(10*rand) % Round up random number from 1 to 10
A=rand(m,n) % Create m ´ n array A of random numbers from 0 to 1
x=size(A) % Store number of rows and columns of A in 1´2 numeric
          % array x
[r,c]=size(A) % Store number of rows of A in 1st output r and
              % number of columns of A in 2nd output c
rr=size(A,1) % Find number of rows of A and save in scalar rr
cc=size(A,2) % Find number of columns of A and save in scalar cc

m =  1

n = 10

A = Columns 1 through 7
    0.2594    0.2042    0.0492    0.6062    0.5463    0.0958    0.6370
    Columns 8 through 10
    0.4429    0.0664    0.3743

x =   1    10

r =   1

c =  10

rr =  1

cc = 10
```

Note that the '`size`' function accepts a second argument when either the number of rows or the number of columns of an array are needed.

The second MATLAB function is `'length'` which returns the larger of the two array indices when the argument is a two dimensional array or the number of elements in a row or column vector.

Example 6.11.2

```
A=rand(2,3) % Create 2´3 array of random numbers
s1=size(A) % Store dimensions of array A in 1´2 row vector s1
l1=length(A) % Store the larger of array A's two dimensions in l1
b=A(:) % Create column vector b from the columns of array A
s2=size(b) % Store dimensions of column vector b in row vector s2
l2=length(b) % Store number of components in column vector b in l2
```

```
A = 0.2491    0.6295    0.6417
    0.9249    0.8783    0.7984

s1 = 2     3

l1 = 3

b =  0.2491
     0.9249
     0.6295
     0.8783
     0.6417
     0.7984

s2 = 6     1

l2 = 6
```