Matrix Indexing

Accessing Single Elements
To reference a particular element in a matrix, specify its row and column number using the following syntax, where \( A \) is the matrix variable. Always specify the row first and column second:

\[
A(\text{row, column})
\]

For example, for a 4-by-4 magic square \( A \),

\[
A = \text{magic}(4)
\]

\[
\begin{bmatrix}
16 & 2 & 3 & 13 \\
5 & 11 & 10 & 8 \\
9 & 7 & 6 & 12 \\
4 & 14 & 15 & 1
\end{bmatrix}
\]

you would access the element at row 4, column 2 with

\[
A(4, 2)
\]

\[
\text{ans} = 14
\]

For arrays with more than two dimensions, specify additional indices following the row and column indices. See the section on Multidimensional Arrays.

Linear Indexing
You can refer to the elements of a MATLAB matrix with a single subscript, \( A(k) \). MATLAB stores matrices and arrays not in the shape that they appear when displayed in the MATLAB Command Window, but as a single column of elements. This single column is composed of all of the columns from the matrix, each appended to the last.

So, matrix \( A \)

\[
A = \begin{bmatrix} 2 & 6 & 9; 4 & 2 & 8; 3 & 5 & 1 \end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 6 & 9 \\
4 & 2 & 8 \\
3 & 5 & 1
\end{bmatrix}
\]

is actually stored in memory as the sequence

\[
2, 4, 3, 6, 2, 5, 9, 8, 1
\]

The element at row 3, column 2 of matrix \( A \) (value = 5) can also be identified as element 6 in the actual storage sequence. To access this element, you have a choice of using the standard \( A(3, 2) \) syntax, or you can use \( A(6) \), which is referred to as linear indexing.

If you supply more subscripts, MATLAB calculates an index into the storage column based on the dimensions you assigned to the array. For example, assume a two-dimensional array like \( A \) has size \([d1 \ d2]\), where \( d1 \) is the number of rows in the array and \( d2 \) is the number of columns. If you supply two subscripts \((i, j)\) representing row-column indices, the offset is

\[
(j-1) \times d1 + i
\]

Given the expression \( A(3,2) \), MATLAB calculates the offset into \( A \)'s storage column as \((2-1) \times 3 + 3\), or 6. Counting down six elements in the column accesses the value 5.

Functions That Control Indexing Style
If you have row-column subscripts but want to use linear indexing instead, you can convert to the latter using the `sub2ind` function. In the 3-by-3 matrix $A$ used in the previous section, `sub2ind` changes a standard row-column index of (3,2) to a linear index of 6:

$$A = \begin{bmatrix} 2 & 6 & 9 \\ 4 & 2 & 8 \\ 3 & 5 & 1 \end{bmatrix};$$

```matlab
linearindex = sub2ind(size(A), 3, 2)
linearindex = 6
```

To get the row-column equivalent of a linear index, use the `ind2sub` function:

```matlab
[row col] = ind2sub(size(A), 6)
row = 3
col = 2
```

### Accessing Multiple Elements

For the 4-by-4 matrix $A$ shown below, it is possible to compute the sum of the elements in the fourth column of $A$ by typing

```matlab
A = magic(4);
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

You can reduce the size of this expression using the colon operator. Subscript expressions involving colons refer to portions of a matrix. The expression

```matlab
A(1:m, n)
```

refers to the elements in rows 1 through $m$ of column $n$ of matrix $A$. Using this notation, you can compute the sum of the fourth column of $A$ more succinctly:

```matlab
sum(A(1:4, 4))
```

### Nonconsecutive Elements

To refer to nonconsecutive elements in a matrix, use the colon operator with a step value. The $m:3:n$ in this expression means to make the assignment to every third element in the matrix. Note that this example uses linear indexing:

```matlab
B = A;
B(1:3:16) = -10
```

```matlab
B =
-10  2  3  -10
 5 11 -10  8
 9 -10  6  12
-10 14 15 -10
```

MATLAB supports a type of array indexing that uses one array as the index into another array. You can base this type of indexing on either the values or the positions of elements in the indexing array.

Here is an example of value-based indexing where array $B$ indexes into elements 1, 3, 6, 7, and 10 of array $A$. In this case, the *numeric values of array $B$ designate the intended elements of $A*:

```matlab
A = 5:5:50
A =
5  10  15  20  25  30  35  40  45  50
```
B = [1 3 6 7 10];

A(B)
ans =
  5   15   30   35   50

If you index into a vector with another vector, the orientation of the indexed vector is honored for the output:

A(B')
ans =
  5   15   30   35   50

A1 = A'; A1(B)
ans =
  5
  15
  30
  35
  50

If you index into a vector with a nonvector, the shape of the indices is honored:

C = [1 3 6; 7 9 10];
A(C)
ans =
  5   15   30
  35   45   50

The end Keyword

MATLAB provides the keyword `end` to designate the last element in a particular dimension of an array. This keyword can be useful in instances where your program does not know how many rows or columns there are in a matrix. You can replace the expression in the previous example with

B(1:3:end) = -10

---

**Note** The keyword `end` has several meanings in MATLAB. It can be used as explained above, or to terminate a conditional block of code such as `if` and `for` blocks, or to terminate a nested function.

Specifying All Elements of a Row or Column

The colon by itself refers to all the elements in a row or column of a matrix. Using the following syntax, you can compute the sum of all elements in the second column of a 4-by-4 magic square A:

`sum(A(:, 2))`

ans =
  34

By using the colon with linear indexing, you can refer to all elements in the entire matrix. This example displays all the elements of matrix A, returning them in a column-wise order:

A(:)
ans =
Using Logicals in Array Indexing

A logical array index designates the elements of an array $A$ based on their position in the indexing array, $B$, not their value. In this masking type of operation, every true element in the indexing array is treated as a positional index into the array being accessed.

In the following example, $B$ is a matrix of logical ones and zeros. The position of these elements in $B$ determines which elements of $A$ are designated by the expression $A(B)$:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$B = \text{logical}([0 \ 1 \ 0; \ 1 \ 0 \ 1; \ 0 \ 0 \ 1])$$

$$A(B)$$

$$\text{ans} = \begin{bmatrix} 4 \\ 2 \\ 6 \\ 9 \end{bmatrix}$$

The `find` function can be useful with logical arrays as it returns the linear indices of nonzero elements in $B$, and thus helps to interpret $A(B)$:

$$\text{find}(B)$$

$$\text{ans} = \begin{bmatrix} 2 \\ 4 \\ 8 \\ 9 \end{bmatrix}$$

Logical Indexing – Example 1
This example creates logical array B that satisfies the condition A > 0.5, and uses the positions of ones in B to index into A:

```matlab
rand('twister', 5489); % Initialize the state of the random number generator.
A = rand(5);
B = A > 0.5;
A(B) = 0
A =
0 0.0975 0.1576 0.1419 0
0 0.2785 0 0.4218 0.0357
0.1270 0 0 0 0
0 0 0.4854 0 0
0 0 0 0 0
```

A simpler way to express this is

```
A(A > 0.5) = 0
```

### Logical Indexing – Example 2

The next example highlights the location of the prime numbers in a magic square using logical indexing to set the nonprimes to 0:

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

B = isprime(A)
B =
0 1 1 1
1 1 0 0
0 1 0 0
0 0 0 0

A(~B) = 0; % Logical indexing
A
A =
0  2  3 13
 5 11  0  0
 0  7  0  0
 0  0  0  0
find(B)
ans =
```

```matlab
B =
0 1 1 1
1 1 0 0
0 1 0 0
0 0 0 0
```
Logical Indexing with a Smaller Array

In most cases, the logical indexing array should have the same number of elements as the array being indexed into, but this is not a requirement. The indexing array may have smaller (but not larger) dimensions:

\[ A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \]

\[ B = \text{logical}([0 \ 1 \ 0; \ 1 \ 0 \ 1]) \]

\[ B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \]

isequal(numel(A), numel(B))
ans =
0

A(B)
an =
4
7
8

MATLAB treats the missing elements of the indexing array as if they were present and set to zero, as in array \( C \) below:

% Add zeros to indexing array \( C \) to give it the same number of elements as \( A \).
C = logical([B(:);0;0;0]);

isequal(numel(A), numel(C))
an =
1

A(C)
an =
4
7
Single-Colon Indexing with Different Array Types

When you index into a standard MATLAB array using a single colon, MATLAB returns a column vector (see variable \( n \), below). When you index into a structure or cell array using a single colon, you get a comma-separated list (see Access Data in a Structure Array and Access Data in a Cell Array for more information.)

Create three types of arrays:

\[
\begin{align*}
\text{n} &= [1 \ 2 \ 3; \ 4 \ 5 \ 6]; \\
\text{c} &= \{1 \ 2; \ 3 \ 4\}; \\
\text{s} &= \text{cell2struct}(\text{c}, \{'a', 'b'\}, 1); \quad \text{s(:,2)}=\text{s(:,1)};
\end{align*}
\]

Use single-colon indexing on each:

\[
\begin{align*}
\text{n(\_)} & \quad \text{c(\_)} & \quad \text{s(:,a)} \\
\text{ans} &= 1 & \quad \text{ans} = 1 & \quad \text{ans} = 1 \\
\text{ans} &= 4 & \quad \text{ans} = & \quad \text{ans} = \\
\text{ans} &= 2 & \quad \text{ans} = 3 & \quad \text{ans} = 2 \\
\text{ans} &= 5 & \quad \text{ans} = \quad \text{ans} = \\
\text{ans} &= 3 & \quad \text{ans} = 2 & \quad \text{ans} = 1 \\
\text{ans} &= 6 & \quad \text{ans} = 4 & \quad \text{ans} = 2
\end{align*}
\]

Indexing on Assignment

When assigning values from one matrix to another matrix, you can use any of the styles of indexing covered in this section. Matrix assignment statements also have the following requirement.

In the assignment \( A(J, K, \ldots) = B(M, N, \ldots) \), subscripts \( J, K, M, N, \) etc. may be scalar, vector, or array, provided that all of the following are true:

- The number of subscripts specified for \( B \), not including trailing subscripts equal to 1, does not exceed \( \text{ndims}(B) \).

- The number of nonscalar subscripts specified for \( A \) equals the number of nonscalar subscripts specified for \( B \). For example, \( A(5, 1:4, 1, 2) = B(5:8) \) is valid because both sides of the equation use one nonscalar subscript.

- The order and length of all nonscalar subscripts specified for \( A \) matches the order and length of nonscalar subscripts specified for \( B \). For example, \( A(1:4, 3, 3:9) = B(5:8, 1:7) \) is valid because both sides of the equation (ignoring the one scalar subscript 3) use a 4-element subscript followed by a 7-element subscript.