

MATLAB Marina: Arrays 2D

Student Learning Objectives

After completing this module, one should:

1. Be able to create and use MATLAB 2D arrays.
2. Be able to index MATLAB 2D arrays.
3. Be able to perform arithmetic and logic operations and apply built in functions on MATLAB 2D arrays.

Terms

scalar, 1D array, 2D array, row, column, transpose, index, indexing (extracting, slicing), colon operator, colon notation, concatenation

MATLAB Functions, Keywords, and Operators

;, length, size, numel, zeros, ones, min, max, mean, sum, cumsum, find, end, ', (), []

MATLAB 2D Arrays

A 2D array is a two-dimensional collection of data of the same data type. A 2D array with n rows and m columns contains n times m elements.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}$$

Each row of the n by m array contains m elements and each column contains n elements. The element in row p and column q is referred to as the pq element of the array.

Creating 2D Arrays

MATLAB 2D arrays can be created similar to 1D arrays: entering the values directly enclosed by square brackets (row elements separated by commas or spaces and rows are separated by semicolons), using the colon operator to create rows or columns and concatenating the rows/columns, using built in MATLAB functions such as `zeros`, `ones`, and `rand`, and created as the result of operations on arrays.

Figure 1a shows examples of creating 2D arrays by directly entering the values and using the colon operator. Figure 1b shows examples of creating 2D arrays using built in MATLAB functions. MATLAB's colon operator creates row vectors and if a column is desired the transpose operator can be used to convert rows to columns (or columns to rows).

```

A = [0, 1, 2;           % 3 by 3 array
     3, 4, 5;
     6, 7, 8];

B = [0.0:0.5:5.0;      % 2 by 11 array
     0.0:0.25:2.5];

C = [(1:1:5)', (2:2:10)']; % 5 by 2 array

D1 = 1:1:15;           % 1 by 15 array
D2 = 2:2:30;           % 1 by 15 array
D3 = [5; 10];         % 2 by 1 array

D = [D1; D2];         % 2 by 15 array
E = [D, D3];          % 2 by 16 array

```

Figure 1a. Creating 2D Arrays Using Direct Entry, Colon Operator, and Concatenation

```

ZZ = zeros(4,8);
OO = ones(5,5);
RR = 0 + (100 - 0)*rand(4,20);

```

Figure 1b. Creating 2D Arrays Using Built in Functions

MATLAB 2D arrays must be rectangular in shape; all rows of an array must have the same number of elements and all columns of an array must have same number of elements, i.e. all rows must have same number of columns and columns must have same number of rows. This is important for determining when arrays can be concatenated together to create larger arrays.

Array Dimensions

The `size` function should be used with 2D arrays for determining the array dimensions.

There are three common usages of the `size` function for a 2D array:

- `d = size(A)` returns a two-element row vector `d` with the first element being the number of rows and the second element the number of columns
- `[nR,nC] = size(A)` returns the number of rows and columns in separate variables `nR` and `nC`; `nR` holds the number of rows and `nC` holds the number of columns
- `D = size(A, DIM)` returns the length of the array dimension specified by the `DIM` argument. For 2D arrays `size(A,1)` returns the number of rows and `size(A,2)` returns the number of columns

The `length` function is not generally used with 2D arrays. The `length` function when applied to a 2D array returns the largest dimension of the array; i.e. the larger number of the number of rows or columns. The `length` function cannot be used to determine the number rows or columns of a 2D array since `length` only takes one argument and the dimension to take length along cannot be specified.

When working with 1D arrays use the `length` function and with 2D arrays use the `size` function. If you do not know whether the data is 1D versus 2D, the `size` function should be used.

Figure 2 shows examples of using the `size` and `length` functions on 2D arrays.

```
A = [1:1:10;  
     11:1:20;  
     21:1:30];  
[nR, nC] = size(A)  
nR =  
     3  
nC =  
    10  
  
l = length(A)  
l =  
    10  
  
nR = size(A,1)  
nR =  
     3  
  
nC = size(A,2)  
nC =  
    10
```

Figure 2. Using `size` and `length` with 2D Arrays

Indexing 2D Arrays

The elements of 2D arrays are indexed similar to the elements of 1D arrays: using the array name and the index (position). For 2D arrays, both the row and column index (or range of row and column indices) must be specified.

Figure 3 shows examples of indexing a single element, a row, a column, and a rectangular subsection of a 2D array. The colon operator used by itself when indexing is equivalent to `1:1:end` along that dimension and selects an entire row or an entire column. The `end` keyword when used in an indexing expression it is equivalent to the size of the dimension it is being used to index, i.e. the last index along that dimension.

2D Arrays can also be indexed using a 2D array of Booleans of the same dimensions that contains trues (1s) for the elements to be indexed.

```

A = [1:1:6; % 3 by 6 array
     2:2:12;
     3:3:18];

e12_4 = A(2,4); % index element in row 2 column 4
Ar2 = A(2,1:1:end); % index row 2
Ac1 = A(:,1); % index column 1
Ac4 = A(:,4); % index column 4
B = A(1:1:2,3:1:5); % index rows 1-2, columns 3 - 5

```

Figure 3. Indexing 2D Arrays

Modifying and Removing Elements of Arrays

As for 1D arrays, one can modify a portion of a 2D array by specifying the range to modify and providing the appropriate number of new values; i.e. index the places in the array to be modified and assign new values to those places.

Rows or columns can be added to 2D arrays using concatenation and rows or columns can be removed from array by specifying the range to remove and assigning the empty vector to the specified elements; i.e. index the places to be removed and assign the empty vector to those places. One must be careful when adding or removing elements to a 2D array as the resulting array must have the same number of columns and all the columns have the same number of rows. Entire rows or columns must be removed and resulting array must be rectangular.

```

A = [1:1:6; % 3 by 6 array
     2:2:12;
     3:3:18];

A(2,4) = 99; % modify the value of element 2,4 to 99
A(:,3) = [79; 89; 99]; % modify column 3
A(:,5) = []; % remove column 5

```

Figure 4. Modifying and Removing Elements of 2D Arrays

Element by Element Operations and Functions on 2D Arrays

Element by element operations can be performed on 2D arrays of the same dimensions (same size) or on a 2D array and a scalar.

MATLAB functions will generally accept 2D arrays as arguments. MATLAB functions perform their operation on each element in the argument and return a result the same size as the argument. Some commonly used functions behave differently than one might expect when applied to 2D arrays.

The `sum` and `mean` functions applied to 2D arrays return a 1D row array containing the sum or mean of the elements in each column of the 2D array (sum and mean operate along the columns of a 2D array). The `min` and `max` functions applied to 2D arrays return the minimum

or maximum element in each column of the array as well as the row location of the minimum or maximum element in the column. Applying the `min` or `max` functions twice to a 2D array returns the minimum or maximum element in the 2D array. Applying the `sum` function twice to a 2D array returns the sum of all the elements in the 2D array.

The `sum`, `mean`, `min`, and `max` functions have versions with an argument that can be used to specify what dimension (along rows or along columns) to apply the function along. Remember that MATLAB's help can be used to determine the arguments needed and the different variations of the built-in functions.

```
A = [8, 6, 10, 9, 2;
     7, 8, 9, 9, 4;
     8, 8, 9, 8, 0];
Amin = min(A)
Amin =
     7     6     9     8     0

Amax = max(A)
Amax =
     8     8    10     9     4

Asum = sum(A)
Asum =
    23    22    28    26     6

Amean = mean(A)
Amean =
    7.67    7.33    9.33    8.67    2.00
```

Figure 5. Applying min, max, sum, and mean Functions to 2D Array

Comparisons on 2D Arrays

Comparisons using logical and relational operators can be performed on the elements of a 2D array for 2D arrays of same dimensions or a 2D array and a scalar). The result of the comparison will be a 2D array of logicals.

```
A = [8, 6, 10, 9, 2;
     7, 8, 9, 9, 4;
     8, 8, 9, 8, 0];
ALT7_log = A < 7
ALT7_log =
    3x5 logical array
     0     1     0     0     1
     0     0     0     0     1
     0     0     0     0     1
```

Figure 6. Comparison on a 2D Array

The resulting array of logicals can be used to index the 2D array to get the values for which the comparison is true. Note that after the indexing operation the dimensions of the result is not longer a 2D array of the same size as A, rather one has a 1D column array of the values.

```
ALT7 = A(ALT7_log)
ALT7 =
     6
     2
     4
     0
```

Figure 7. Indexing 2D Array using Array of Logicals from a Comparison

Indexing 2D arrays after a comparison is more straightforward using the array of logicals rather than corresponding indices. For more information see the optional linearized array section.

Linearized Arrays (Optional)

Multidimensional arrays (2D arrays, 3D arrays, etc.) are stored in memory as a linearized array. One can think of a multidimensional array as being a vector with first the values from column 1, then the values from column 2 and so on.

Consider the 3 by 4 array A:

$$A = \begin{pmatrix} 1 & 7 & 2 & 4 \\ 8 & 0 & 0 & 3 \\ 0 & 1 & 5 & 5 \end{pmatrix}$$

The array A is stored in memory as an 1D array of 12 elements:

$$(1, 8, 0, 7, 0, 1, 2, 0, 5, 4, 3, 5)$$

Row 1 column 1 is linearized index 1, row 2 column 1 is linearized index 2, row 3 column 1 is linearized index 3, row 1 column 2 is linearized index 4, and so on.

The `find` function when used for converting logicals to indices returns a linearized array of indices. The `find` function can also return the corresponding row and column indices. The example of Figure 8 shows the 2D array of logicals, 1D array of linearized indices, and two 1D arrays of row and column indices resulting from a comparison on a 2D array.

Indexing after a comparison can be accomplished using either the array of logicals or array of linearized indices. However, the row and column indices resulting from a comparison on a 2D array whiel useful for manually identifying the result of a comparison are less useful for indexing; see example of Figure 9.

```

A = [1, 7, 2, 4;
     8, 0, 0, 3;
     0, 1, 5, 5];

AGT5_log = A > 5
AGT5_log =
    3x4 logical array
     0     1     0     0
     1     0     0     0
     0     0     0     0

AGT5_ind = find(A>5)
AGT5_ind =
     2
     4

[r_ind,c_ind] = find(A >5)
r_ind =
     2
     1
c_ind =
     1
     2

```

Figure 8. Linearized Array Indices and Corresponding Row and Column Indices

```

AGT5_a = A(AGT5_log)
AGT5_a =
     8
     7

AGT5_b = A(AGT5_ind)
AGT5_b =
     8
     7

AGT5_c = A(r_ind,c_ind)
AGT5_c =
     8     0
     1     7

```

Figure 9. Indexing 2D Array using Result of a Comparison

In the example of Figure 9, indexing A using the 2D array of logicals and the linearized indices both produce the correct result. Indexing A using the pairs of row and column indices produces an incorrect result and more values than one might expect.

When indexing a 2D array using two 1D arrays or row and column indices, all permutations of the row and column numbers are used for the indexing operation, for the example of Figure 9, A is indexed for row column pairs (2,1), (1,1), (1,2), and (2,2).

Generally when working with multidimensional arrays, indexing should be performed using arrays of logicals or linearized indices rather than pairs of row and column indices.

Last modified Friday, September 18, 2020

 [MATLAB Marina](#) is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.