Armstrong State University
Engineering Studies
MATLAB Marina – 2D Arrays and Matrices Primer

Prerequisites

The 2D Arrays and Matrices Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, and 1D arrays and vectors. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module, MATLAB Marina Variables module, and MATLAB Marina 1D Arrays and Vectors Module.

Learning Objectives

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, vector, matrix, row, column, index, indexing (extracting, slicing), colon operator, colon notation, concatenation, array operation, element by element operation, transpose

MATLAB Functions, Keywords, and Operators

:, length, size, zeros, ones, min, max, mean, sum, cumsum, find, end, ( ), [ ], ‘

MATLAB 2D Arrays

A 2D array or matrix 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.

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

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

2D arrays can be created in MATLAB 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 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 and 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.

```
>> A = [0, 1, 2; 3, 4, 5; 6, 7, 8];      % 3 by 3 array
>> B = [0.0:0.5:5.0; 0.0:0.25:2.5];      % 2 by 11 array
>> 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

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.

Size of Arrays

The `size` command returns the dimensions of the array. The `length` command returns the largest dimension of the array, i.e. the larger number of the number of rows or columns. Generally, the `length` command is used with 1D arrays and the `size` command with 2D arrays.

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

- `D = size(XX)` returns a two-element row vector D with the first element being the number of rows and the second element the number of columns
- `[M,N] = size(XX)` returns the number of rows and columns in separate varaibles M and N, M is the number of rows and N is the number of columns
- `M = size(XX,DIM)` returns the length of the array dimension specified by the `DIM` argument, for 2D arrays `size(XX,1)` returns the number of rows and `size(XX,2)` returns the number of columns

Figure 2 shows examples of using the `size` and `length` commands on a 2D array.

```
>> AA = [1:1:10; 11:1:20 ; 21:1:30];
>> length(AA)
ans = 10
>> D = size(AA)
D = 3  10
>> [M, N] = size(AA)
M = 3
N = 10
>> nrow = size(AA,1)
nrow = 3
>> ncol = size(AA,2)
ncol = 10
```

Figure 2, Using `size` and `length` Commands with 2D Arrays

Indexing 2D Arrays

Elements of 2D arrays are indexed similar to elements of 1D arrays; using the array name and the 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.

```
>> AA = [1:1:6; 2:2:12; 3:3:18];   % 3 by 6 array
>> AA(2,4)                         % element in row 2 column 4
ans = 8
>> AA(2,1:1:end)                   % row 2
ans = 2  4  6  8  10  12
>> AA(:,3)                         % column 3
ans = 3
      6
      9
>> BB = AA(1:1:2,3:1:5)            % rows 1-2, columns 3-5
ans = 3  4  5
      6  8  10
```

Figure 3, Indexing 2D Arrays

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. The colon operator used by itself when indexing is equivalent to 1:1:end along that dimension and selects and entire row or an entire column. Arrays can also be indexed using an array of Booleans of the same dimensions that contains trues (1s) for the elements to be indexed.

Modifying and Removing Elements of 2D 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 to ensure that all the rows of the resulting array 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.

```
>> DD = [1:1:6; 2:2:12; 3:3:18];   % 3 by 6 array
>> DD(2,4) = 99;                   % modify element 2,4
>> DD(:,3) = [79; 89; 99];         % modify column 3
>> DD(:,5) = [];                   % remove column 5
>> disp(DD)
DD = 1   2  79   4    6
     2   4  89   99   12
     3   6  99   12   18
```

Figure 4, Modifying and Removing Elements of 2D Arrays

Array Transpose

The transpose (and conjugate transpose) operator changes the rows of an array to columns and the columns of an array to rows.

$$\begin{pmatrix} a_1 & a_2 & \cdots & a_m \end{pmatrix}^T = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}^T = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \end{pmatrix}$$

The transpose of an m by n array is an n by m array. The transpose operator in MATLAB is a period followed by a single quote and is a post operator, i.e. it is given to the right of the variable. If the matrix is composed of complex values, the single quote operator can be used to take the complex conjugate transpose (transpose rows and columns and tale complex conjugate of each value). When the array consists of all real numbers it does not matter which transpose operator you use. Figure 5 shows examples of the MATLAB transpose operation.

```
>> AA = [0 1 2; 3 4 5];       % 2 by 3 array
>> BB = AA.'
BB = 0  3
     1  4
     2  5
>> CC = [2+3j, 1-5j]
CC = 2.0 + 3.0i   1.0 - 5.0i
>> CC'
ans = 2.0 - 3.0i
      1.0 + 5.0i
>> CC.'
ans = 2.0 + 3.0i
      1.0 - 5.0i
```

Figure 5, Array Transpose and Conjugate Array Transpose Examples

Arithmetic Operations on 2D Arrays

MATLAB supports matrix (regular) and array (element by element) arithmetic operations on 2D arrays just as it does for 1D arrays. MATLAB's array (element by element or dot) operations can be performed on the elements of 2D arrays as long as the operation involves two 2D arrays of the same size or a 2D array and a scalar.

As with 1D arrays, one must be careful to use the matrix operations and the array operations correctly. Generally when operations are to be performed on corresponding elements of two arrays, the array operations should be used. When one of the operands is a scalar and for addition and subtraction the dot operation is not needed. Figure 6 shows some examples of 2D array arithmetic operations.

```
>> AA = [3, 2, 1; 8, 7, 6];        % 2 by 3 array
>> BB = [1, 2, 3; 4, 5, 6];        % 2 by 3 array
>> AA + 2
ans = 5   4   3
      10  9   8
>> AA/2
ans = 1.5   1    0.5
      4     3.5  3
>> AA.*BB
ans = 3    4    3
      32   35   36
```

Figure 6, 2D Array Arithmetic Operations

Applying Built in Functions to 2D Arrays

MATLAB functions will generally accept 2D arrays as arguments just as they accept scalars and 1D arrays. Generally MATLAB functions perform their operation on each element in the argument and return a result the same size as the argument. Remember that MATLAB's help can be used to determine the arguments needed and the different variations of the built in functions.

When the MATLAB functions `sum` and `mean` are applied to 2D arrays, they return a 1D row array containing the sum or mean of the elements in each column of the 2D array. If one wanted the sum or mean of all the elements in a 2D array, one could apply the function twice. The `sum` and `mean` functions also have versions that allow one to specify what dimension to apply the function along.

When the MATLAB functions `min` and `max` are applied to 2D arrays, they 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.

```
>> data = [8, 6, 8, 7; 5, 6, 6, 4; 7, 7, 10, 9];
>> sumOfData = sum(data)
sumOfData = 20  19  24  20
>> sumOfAllData = sum(sum(data))
sumOfAllData = 83
>> meanOfDataRows = mean(data,2)
meanODataRows = 7.2500
                5.2500
                8.2500
>> [minOfData, loc] = min(data)
minOfData = 5   6   6   4
loc = 2   1   2   2
>> maxOfAllData = max(max(data))
maxOfAllData = 10
```

Figure 7, MATLAB sum, mean, min, and max Functions Applied to 2D Arrays

Logic and Relational Operations on 2D Arrays

MATLAB's logical and relational operations can be performed on all the elements of a 2D array as long as the operation involves two arrays of the same size or an array and a scalar. The same guidelines provided for 1D arrays should be followed for logic and relational operations using 2D arrays.

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 etc. Consider the 3 by 4 array AA

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

The array AA is stored in memory as a vector $\begin{pmatrix} 1 & 8 & 0 & 7 & 0 & 1 & 2 & 0 & 5 & 4 & 3 & 5 \end{pmatrix}$.

Understanding linearized arrays is necessary for understanding how MATLAB's `find` function operates. The `find` function returns a linearized array of indices that can be used to access the array.

```
>> AA = [1 7 2 4; 8 0 0 3; 0 1 5 5];
>> ind = find(AA == 8)
ind = 2
>> inds = find(AA > 4)
inds = 2
       4
       9
      12
```

Figure 8a, Linearized Arrays and Result of `find` Function

Notice in the example of Figure 8a that the MATLAB `find` function does not return row and column indices where a relation is true but rather returns the linearized indices. For linearized indices, one counts down columns starting from the element in row one column one. In the array AA above, linearized index 6 corresponds to element 3,2 (element in row 3 column 2) and linearized index 8 corresponds to element 2,3. One can index 2D arrays using either the row and column indices or linearized indices as shown in Figure 8b.

```
>> AA = [1 7 2 4; 8 0 0 3; 0 1 5 5];
>> AA(6)                      % index element 3,2
ans = 1
>> AA(3,2)                    % index element 3,2
ans = 1
>> AA(1:6)                         % index columns 1 and 2
ans = 1     8     0     7     0     1
```

Figure 8b, Indexing 2D Arrays using Linearized Indices

Last modified Tuesday, September 09, 2014