Armstrong State University
Engineering Studies
MATLAB Marina – 1D Arrays and Vectors Primer

Prerequisites

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

Learning Objectives

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

Terms

scalar, 1D array, vector, index, indexing (extracting, slicing), colon operator, colon notation, concatenation, array operation, element by element operation

MATLAB Functions, Keywords, and Operators

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

MATLAB 1D Arrays

A scalar is a single element. A 1D array or vector is a one-dimensional collection of data of the same data type. For example a 1D array $v = \begin{bmatrix} v_1 & v_2 & \cdots & v_{10} \end{bmatrix}$ of 10 integers could either be a row (1 by 10) or column (10 by 1) of ten integer values.

MATLAB Colon Operator

The MATLAB colon operator allows one to create a range of values without using a loop structure; K:L is the same as [K, K+1, K+2, …, L] and K:D:L is the same as [K, K+D, K+2D,…, L]. Figure 1 shows two examples of using the colon operator to create ranges of numbers.

```
>> 1:1:8
ans = 1  2  3  4  5  6  7  8
>> 0.0:0.25:2.5
ans =  0 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50
```

Figure 1, Creating Ranges of Numbers using the Colon Operator

Creating 1D Arrays

There are several ways to create 1D arrays (vectors) in MATLAB: entering the values directly enclosed by square brackets (row elements separated by commas or spaces and elements in columns separated by semicolons), using the colon operator, using built in MATLAB functions such as `linspace`, `zeros`, `ones`, and `rand`, and created as the result of operations on 1D arrays. Figure 2a shows examples of creating 1D arrays by directly entering the values and using the colon operator. Figure 2b shows examples of creating 1D arrays using built in MATLAB functions.

```
>> emptyVector = []
emptyVector = []
>> vecDirect = [3, 7, -1, 2]
vecDirect = 3  7  -1  2
>> vecColon = 0:2:20
vecColon = 0  2  4  6  8  10  12  14  16  18  20
```

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

```
>> vecLinspace = linspace(0,5,10);
vecLinspace = 0  0.5556  1.1111  1.6667  2.2222  2.7778
3.3333    3.8889  4.4444  5.0000
>> rowvecZeros = zeros(1,5);
rowvecZeros = 0  0  0  0  0
>> colvecOnes = ones(7,1);
>> vecRand = rand(1,100);
```

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

Multiple 1D arrays can be concatenated to create larger 1D arrays. MATLAB's 1D array concatenation is similar to directly entering 1D arrays. Row arrays are concatenated by providing the list of 1D row arrays enclosed in square brackets separated by spaces or commas. Column arrays are concatenated by providing the list of 1D column arrays enclosed in square brackets separated by semicolons.

```
>> row1 = [2, 4, -6];
>> row2 = [1, 3, 5];
>> row = [row1 , row2];
>> col1 = [2; 4; -6];
>> col2 = [1; 3; 5];
>> col = [col1 ; col2];
```

Figure 2c, 1D Array Concatenation

Indexing 1D Arrays

The individual items in a 1D array are called elements and the position in the 1D array is called the index. For example, the 1 by 6 array `vec = [13, 7, -5, 2, 63, 8]` contains the element 13 at index 1, 7 and index 2, and 8 at index 6. Individual elements of a 1D array can be indexed (accessed) using the array name and the index (position) enclosed in parentheses. Multiple elements that are successive can be indexed by a providing range of indexes. Figure 3 illustrates how to index the 4th element and the third through sixth elements of the 1D array.

```
>> vec = [13, 7, -5, 2, 63, 8];
>> vec(4)
ans = 2
>> vec(3:1:6)
ans = -5  2  63  8
```

Figure 3, Indexing 1D Arrays

Indexing arrays is also called slicing, accessing, and extracting.

Modifying and Removing Elements of Arrays

One can modify a portion of an 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. Elements can be added to the beginning or end of a 1D array using concatenation. Elements can be removed from an 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.

```
>> vec = [13, 7, -5, 2, 63, 8];
>> vec(2) = 0
vec = 13  0  -5  2  63  8
>> vec(4:end) = zeros(1,3)
vec = 13  0  -5  0  0  0
>> vec(3) = []
vec = 13  0  0  0  0
```

Figure 4, Modifying and Removing Elements of Arrays

Useful Reserved Words and Built in Functions for Arrays

The `size` and `length` functions provide information about the dimensions of a variable. The `length` function returns the length of a 1D array (vector). If the argument is more than one-dimensional it returns the value of the largest dimension. The `size` function returns the dimensions of the variable. For 1D arrays it is generally better to use the `length` function.

The `end` reserved word 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. Using the colon operator by itself when indexing is equivalent to using `1:1:end` along that dimension.

```
>> vec = [13, 7, -5, 2, 63, 8];
>> length(vec)
ans = 6
>> vec(4:end)
ans = 2  63  8
>> vec(:)
ans = 13  7  -5  2  63  8
```

Figure 5, Examples of Using `length` function, `end` keyword, and : Operator

Arithmetic Operations on 1D Arrays
MATLAB supports two types of arithmetic operations: matrix (regular) operations and array (element by element or dot) operations. Array operations (addition, subtraction, multiplication, division, power) can be performed on the elements of 1D arrays as long as the operation involves two 1D arrays of the same size or a 1D array and a scalar. Array operations are performed element by element. MATLAB does not have separate operators for array addition and subtraction as array addition and subtraction are defined the same as matrix addition and subtraction. Figures 6a and 6b show some examples of array arithmetic operations. Note that when one of the operands is a scalar and for addition and subtraction the dot operation is not needed.

```
>> vec1 = [1 3 5 9 13];
>> vec2 = [2 1 2 4 3];
>> vec1 + 1
ans = 2  4  6  10  14
>> vec3 = vec1 + vec2
vec3 = 3  4  7  13  16
>> vec4 = vec1 - vec2
vec4 = -1  2  3  5  10
```

Figure 6a, 1D Array Addition and Subtraction

```
>> vec1 = [1 3 5 9 13];
>> vec2 = [2 1 2 4 3];
>> 2*vec1
ans = 2  6  10  18  26
>> vec3 = vec1.*vec2
vec3 = 2  3  10  36  39
>> vec4 = vec1./vec2
vec4 = 0.5  3  2.5  2.25  4.33
>> vec5 = vec1.^2
vec5 = 1  9  25  81  169
```

Figure 6b, 1D Array Multiplication, Division, and Power

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.

Applying Built in Functions to 1D Arrays
Built in MATLAB functions will generally accept either scalars or arrays as arguments. Generally MATLAB functions perform their operation on each element in the argument and return a result the same size as the argument. Some of the more commonly used MATLAB functions are: `cos` (cosine), `sin` (sine), `sqrt` (square root), `pow` (power), `exp` (exponential), `log` (natural log), and `log10` (log base 10). Remember that MATLAB's help can be used to determine the arguments needed and the different variations of the built in functions.

The built in MATLAB functions `sum`, `mean`, `min`, and `max` are commonly used to operate on arrays but they do not return a result of the same size. The `sum` and `mean` functions return the sum of the elements in the array and the mean of the elements in the array. The `min` and `max` functions return the value of the minimum and maximum element in the vector as well as the location of the element in the vector.

```
>> data = [8, 4, 6, 9, 10, 8, 7, 2, 5];
>> meanOfData = mean(data)
meanOfData = 6.5556
>> minOfData = min(data)
minOfData = 2
>> [maxOfData, loc] = max(data)
maxOfData = 10
loc = 5
>> sumOfData = sum(data)
sumOfData = 59
```

Figure 7, MATLAB sum, mean, min, and max Functions

Logic and Relational Operations on 1D Arrays
MATLAB supports the element by element relational operators: less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equality (==), not equal (~=) and the element by element logical operators not (~), and (&), or (|). There are additional logic operations such as exclusive or (`xor`) supported though built in functions. Element by element logical and relational operations can be performed on all the elements of an array as long as the operation involves two arrays of the same size or an array and a scalar.

There are short circuit versions of logic and (&&) and logic or (||). The short circuit versions of the logic operators must produce a scalar result and only use the second operand if the result cannot be determined from the first operand. In most cases, the element by element logic operators should be used.

Figure 8 shows some example of logic and relational operations on 1D arrays.

```
>> vec1 = [1 3 5 9 13];
>> vec2 = [2 1 2 4 3];
>> vec1 > vec2
ans =  0   1   1   1   1
>> vec1 < 5
ans =  1     1     0     0     0
>> res = (vec1 > 4) & (vec1 < 10)
res = 0  0  1  1  0
```

Figure 8, MATLAB Logic and Relational Operations

MATLAB treats the value of 0 as false and the value of 1 as true (actually any nonzero value is treated as true).

Array indexing can be performed using arrays of logical values (true, 1 and false , 0) as long as the logic array is the same size as the array being indexed. When indexing with logic arrays, the elements corresponding to the true indices are returned.

```
>> vec1 = [1 3 5 9 13];
>> res = (vec1 > 4) & (vec1 < 10)
res = 0  0  1  1  0
>> vec1(res)
ans = 5  9
```

Figure 9, Indexing with a Logic Array

MATLAB find Function

The `find` function returns the linear indices corresponding to the locations where a condition evaluates to true. If the condition is the vector, then find returns the indices of the nonzero values. In effect, the `find` function converts a logic array of 0s and 1s to an array of indices corresponding to the places where the logic array is true.

```
>> vec1 = [1 3 5 9 13];
>> ind = find((vec1 > 4) & (vec1 < 10))
ind = 3  4
>> vec1(ind)
ans = 5  9
```

Figure 10, Using find to Convert Logic Array to Indices

Either a logic array or array of indices can be used to index an array to extract the elements in the array matching the condition.

Using Arrays and Array Operations to Evaluate Formulas
MATLAB's arrays and array operations are useful for many engineering applications such as: evaluating a function for a range of values, determining the min and max values of a function over some range, and evaluating summations or approximating infinite summations.

The MATLAB program of Figure 11 evaluates and plots the function $f(t) = 4t^2 - 7t + 6$ over the range $-5.0 \le t \le 5.0$.

```
clear;
clc;
close all;
% vector of independent variable values
t = -5.0 : 0.1 : 5.0;
% evaluate and plot f(t) = 4t^2 – 7t + 6
f = 4*t.^2 - 7*t + 6;
figure(1)
plot(t,f);
xlabel('t (sec)');
ylabel('f(t)');
```

Figure 11, MATLAB Program to Evaluate Function f(t) = 4t^2 – 7t + 6

When evaluating a function over an independent variable range: first generate a vector of the independent variable values being careful to use an appropriate interval between points and then evaluate the function for the vector being careful to use array operations when appropriate. Notice that when referring to the variable for the function, that `f` rather than `f(t)` is used.

MATLAB interprets parentheses as either array indexing or enclosing the arguments of a function call. If we tried to use the statement `f(t) = 4*t.^2 - 7*t + 6;` the program would have a syntax error as MATLAB would evaluate the formula and then try to place the result in the variable `f` in the range of indices in the variable `t`. Since array indices have to be integers a syntax error results.

The following two statements could be used in place of the one statement to evaluate the function:

```
ind = 1:1:length(t);
f(ind) = 4*t.^2 - 7*t + 6
```

In this case, an array of integer indices named `ind` of the same size as the result of evaluating the function for the vector `t` is created. It is syntactically and logically correct to then place the result of the formula evaluation into the places of the variable `f` indicated by the indices `ind`. However this extra step is not necessary when a variable is being used for the first time or being redefined. MATLAB will allocate enough space for the result of the operation and place the elements of the result in the variable in the same order as the evaluation results in.

The MATLAB program of Figure 12 evaluates and plots the piecewise function

$$f(t)=\begin{cases} 2t & 0\le t<1 \\ 4t-1 & 1\le t<2 \\ 4-3t & 2\le t\le 4 \end{cases}$$

```
clear;
clc;
close all;
% generate each of the three time regions
t1 = 0.0 : 0.1 : 0.9;
t2 = 1.0 : 0.1 : 1.9;
t3 = 2.0 : 0.1 : 4.0;
% evaluate the piecewise function for each region
f1 = 2*t1;
f2 = 4*t2 – 1;
f2 = 4 – 3*t3;
% concatenate the results of the three regions for
% the entire piecewise function
t = [t1, t2, t3];
f = [f1, f2, f3];
```

Figure 12, MATLAB Program to Evaluate Piecewise Function

The MATLAB program of Figure 13a determines an approximation of an infinite summation $S=\sum_{n=1}^{\infty}\frac{1}{n}$ for N terms. The approximation for N terms is $S_N=\sum_{n=1}^{N}\frac{1}{n}$.

```
clear;
clc;
% number of terms in approximation
N = 20;
% generate array of term numbers
n = 1:1:N;
% evaluate the formula for each term
terms = 1./n;
% sum the terms
SN = sum(terms)
```

Figure 13a, MATLAB Program to Evaluate Approximation of Infinite Series

Figure 13b shows a version with several of the steps combined.

```
clear;
clc;
% number of terms in approximation
N = 20;
% generate array of term numbers and evaluate sum of terms
n = 1:1:N;
SN = sum(1./n)
```

Figure 13b , MATLAB Program to Evaluate Approximation of Infinite Series

Last modified Tuesday, September 09, 2014