ENGR 1187  |  MATLAB 2: Array Creation
Preparation Material

Learning Objectives
1. Demonstrate proper convention for assigning arrays to a variable (e.g., space, semi-colons, colon operators, linspace function)
2. Apply the transpose operator correctly

Topics
Read Chapter 2.1 – 2.4 of the MATLAB book before coming to class. This preparation material is provided to supplement this reading. Students will learn basic understanding of arrays and how to create them (vectors and matrices) in MATLAB. This material contains the following:

1. Key Definitions & Ideas
2. Vectors
3. Matrices
4. Transpose
5. Examples

1. Key Definitions & Ideas

**Arrays** – An array is a data structure that contains a group of elements of the same data type. An array can be a scalar, vector or matrix (explained below). The size of arrays are commonly described by the number of rows and columns they contain. This is explained in further detail below.

**Scalar** – A scalar variable is a variable with 1 row and 1 column. For example, type the following line in the MATLAB command window:

```
>> x = 5
```

The variable x is a scalar variable (contains a single value) and has only 1 row and 1 column.

**Vector** – A vector is a string of values (or a string of scalars). There are two types of vectors you can create in MATLAB. They are:

- **Row vector** – one row and multiple columns
- **Column vector** – multiple rows and one column

**Matrix** – A matrix is a rectangular array of numbers, otherwise described as having multiple rows and multiple columns.
ENGR 1181 MATLAB 2: Array Creation

Preparation Material

2. Vectors

A vector in MATLAB is an array that contains multiple elements in either a single row or a single column. Vectors are commonly referred to as one-dimensional arrays.

Vectors are useful for many applications (e.g., representing position of an object, scientific data, etc...).

<table>
<thead>
<tr>
<th>Table 1: Population Growth Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population</td>
</tr>
</tbody>
</table>

Creating vectors:

In MATLAB, a row vector can be created by typing the elements (numbers) inside of square brackets [ ], separating them by a comma or space(s). For example, we can create two row vectors of the data presented in Table 1. Type the following line in the MATLAB command window:

```
>> pop  = [127, 130, 136, 145, 158, 178, 211] % Separated by commas
```

You'll notice that both methods of creating a row vector produce the same output (no comma vs. comma).

To create column vectors, semi-colons can be used in place of the comma or if the data is entered into a script file the ‘Enter’ key can be used. For example, type the following in the command window:

```
```

Or in a script file the vector can be created as follows:

```matlab
year = [ 1984
1986
1988
1990
1992
1994
1996]
```
Vectors with equal spacing:
A vector in which the starting value is \( m \), the spacing is \( q \) and the last value is \( n \) can be created by typing \([ m : q : n ]\). This can be described as \([ \text{start} : \text{spacing} : \text{end} ]\).

\[
\begin{align*}
\text{>> } x &= [1 : 2 : 13] \quad \text{or} \quad x &= 1 : 2 : 13 \\
&= 1 \quad 3 \quad 5 \quad 7 \quad 9 \quad 11 \quad 13 \\
\text{>> } x &= [1.5 : 0.1 : 2.1] \\
&= 1.5000 \quad 1.6000 \quad 1.7000 \quad 1.8000 \quad 1.9000 \quad 2.0000 \quad 2.1000 \\
\end{align*}
\]

If spacing \( q \) is omitted, then it is simply \([ m : n ]\) and the default step is 1. This can be described as \([ \text{start} : \text{end} ]\).

\[
\begin{align*}
\text{>> } x &= [-3 : 7] \\
&= -3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \\
\end{align*}
\]

Alternatively, the command \texttt{linspace()} can be used to properly space the variables in a vector. This is useful when the spacing is difficult to identify, but the number of needed data points is known. This is written as \texttt{linspace(m, n, r)}, where the first term is \( m \), the last term is \( n \), and the number of data points is \( r \).

Example: Let’s say someone knows the coordinates on a grid and wants to evenly space 7 items from location 2.5 to 14.5. He/she may not know the exact spacing, especially if design plans change the number of poles is reduced to 6 or 8. To save time and errors, the designer can use \texttt{linspace()} the following way:

\[
\begin{align*}
\text{>> } y &= \text{linspace(2.5,14.5,7)} \\
y &= 2.5 \quad 4.5 \quad 6.5 \quad 8.5 \quad 10.5 \quad 12.5 \quad 14.5 \\
\end{align*}
\]

Note that calculating \((14.5-2.5)/7 = 1.714286\) doesn’t work properly because the end points are included in \texttt{linspace()}. The proper math would actually be \((14.5-2.5)/6\) or referenced as \((n - m) / (r - 1)\).
ENGR 1181 MATLAB 2: Array Creation

Preparation Material

3. Matrices

A \((m \times n)\), or “\(m\) by \(n\)”, matrix has \(m\) rows and \(n\) columns. The number of rows and columns may be the same or different. \((m \times n)\) is called the size of the matrix.

The following matrix \(G\) has two rows and four columns \((2 \times 4)\), or “2 by 4”

\[
\begin{bmatrix}
31 & 26 & 14 & 18 \\
3 & 51 & 20 & 11
\end{bmatrix}
\]

A matrix is created by typing the elements (numbers) row by row inside square brackets \([\]\). Similar to a column vector, the multiple rows in a matrix can be created with a semicolon or the ‘Enter’ key.

\[
\text{>> a = [5 35 43; 4 76 81; 21 32 40]}
\]
\[
a =
\begin{bmatrix}
5 & 35 & 43 \\
4 & 76 & 81 \\
21 & 32 & 40
\end{bmatrix}
\]

\[
\text{>> b = [7 2 76 33 8; 1 98 6 25 6; 5 54 68 9 0]}
\]
\[
b =
\begin{bmatrix}
7 & 2 & 76 & 33 & 8 \\
1 & 98 & 6 & 25 & 6 \\
5 & 54 & 68 & 9 & 0
\end{bmatrix}
\]

4. Transpose

The transpose operation is executed by adding an apostrophe (’) after the vector or matrix:

\[
\text{For a vector: Converts a row vector to a column vector, or converts a column vector to a row vector.}
\]
For a matrix: Interchanges the rows and columns. For example, a 2 x 4 matrix becomes a 4 x 2 matrix.

For a vector: Transposes the vector. For example, a 1 x 4 vector becomes a 4 x 1 vector.

```
>> a = [3 8 1]
   a =
       3 8 1
>> b = a'
   b =
       3
       8
       1
```

```
>> c = 1:2:5
   c =
       1 3 5
>> d = c'
   d =
       1
       3
       5
```

```
>> c = [1 2 3 4; 5 5 5 5; 6 7 8 9]
   c =
       1 2 3 4
       5 5 5 5
       6 7 8 9
>> d = c'
   d =
       1 5 6
       2 5 7
       3 5 8
       4 5 9
```
5. Array Examples

```
>> a = 7
a =
    7

>> E = 3
E =
    3

>> d = [5  a+E  4  E^2]
d =
    5   10    4    9

>> g = [a  a^2  13; a*E  1  a^E]
g =
    7    49   13
         21    1  343

>> V = [a  a^2  13; a*E  1  a^E]’
V =
    7    21
    49    1
   13   343
```