Mrs. Christine Deeb

Chapter 2: Creating Arrays

Table 1: Definition of Array in MATLAB

An **array** is MATLAB's basic data structure.

- In MATLAB all variables are arrays. Arrays can have any dimensions.
  - ***Scalars*** are arrays with only one element. (All the variables we've worked with are scalars.)
  - ***Vectors*** are arrays that are one dimensional – a single row or single column.
  - ***Matrices*** are arrays with two or more dimensions – multiple rows and/or columns.
- In MATLAB, you do not have to define the variable size before assigning to it. Once you assign an array to a variable, MATLAB will specify the dimensions.
- If you reassign or over-write the variable, its dimensions will change accordingly.

2.1 Creating a One-Dimensional Array (Vector)

There are three major ways to create **row vectors**. MATLAB displays row vectors horizontally.

1. Explicitly typing element by element separated by spaces or commas – when we have **a set of data**:

   `variable_name = [n1, n2, n3]` or `variable_name = [n1 n2 n3]`

2. Using the colon (:) – when we want to create a vector with **specified (constant) spacing**:

   `variable_name = [m:q:n]` or `variable_name = m:q:n`

   - `m` is the first number; `n` is the last number; `q` is the spacing.
   - `v = m:q:n` means `v=[m, m+q, m+2q, m+3q, ⋯ , n]`
   - If `q` is positive, elements are increasing, and `m` should be less than `n`.
   - If `q` is negative, elements are decreasing, and `m` should be larger than `n`.
   - If `q` is omitted, the spacing is positive 1.
   - If the numbers `m`, `q`, and `n` are such that the value of `n` cannot be obtained by adding `q`'s to `m`, then (for positive `n`) the last element in the vector will be the last number that does not exceed `n`.

3. Using `linspace` – when we want to create a vector with **specified number of elements**:

   `variable_name = linspace(xi, xf, n)`

   - `xi` is the first number; `xf` is the last number; `n` is the number of elements.
   - If `n` is omitted, the default is 100 elements.
   - If the numbers `m`, `q`, and `n` are such that the value of `n` cannot be obtained by adding `q`'s to `m`, then (for positive `n`) the last element in the vector will be the last number that does not exceed `n`.

There are two major ways to create **column vectors**. MATLAB displays column vectors vertically.

1. Explicitly type element by element **separated by semicolons**:

```
variable_name = [n1; n2; n3]
```

2. Create a row vector, then put an apostrophe (’) after closing the end bracket:

```
v=[n1 n2 n3]’ or v=[m:q:n]’ or v=[linspace(xi,xf,n)]’
```

Example 1: ***Without using MATLAB***, determine what the following commands will output.

```
>> v1=[3^2, sqrt(16), cos(pi)]
```

```
>> v2=v1’
```

```
>> v3=1:3:15
```

```
>> v4=12:-3:5
```

```
>> v5=12:1:5
```

```
>> v6=linspace(8,8,10)’
```

2.2 Creating a Two-Dimensional Array (Matrix)

There are two ways to **create two-dimensional matrices** using vectors.

1. Build the matrix by **“stacking” row vectors** atop one another using semicolons to separate:

```
M = [row_vector1; row_vector2; row_vector3; ⋯ ; row_vectorN]
```

- It’s important that each row is separated by a semicolon!
- All rows must have the same number of columns.
- Otherwise, MATLAB will output: Error using vertcat … .

2. Build the matrix **by “appending” column vectors** left to right using spaces or commas to separate:

```
M = [col_vector1, col_vector2, col_vector3, ⋯ , col_vectorN]
```

- All columns must have the same number of rows.
- Otherwise, MATLAB will output: Error using horzcat … .

Other Commands to Create Matrices

Command	Description

`zeros(m,n)`
`zeros(n)`
	Makes an m by n matrix of zeros.
	Makes a square n by n matrix of zeros.

`ones(m,n)`
`ones(n)`
	Makes an m by n matrix of ones.
	Makes a square n by n matrix of ones.

`eye(m,n)`
`eye(n)`
	Makes an m by n identity matrix (zeros everywhere except for main diagonal of ones).
	Makes a square n by n identity matrix.

Example 2: ***Without using MATLAB***, determine what the following commands will output.

```
>> v1=[3^2, sqrt(16), cos(pi)];
>> cd=6; e=3; h=4;
>> v2=[e, cd*h, h^2];
>> M=[v1; v2]
```

```
>> M2=[v1 v2]
```

```
>> M3=[v1', v2']
```

```
>> B=[1:4; linspace(1,4,5)]
```

```
>> z=100*ones(3,4)
```

```
>> Q=eye(2,3)/2
```

2.4 The Transpose Operator

Table 2: Transposing an Array in MATLAB

In linear algebra, we use a superscript “T” to denote the transpose. **In MATLAB, we use an apostrophe after the variable to denote the transpose.**

- If `v` is a scalar, `v'` is a scalar.
- If `v` is a row vector, `v'` is a column vector.
- If `v` is a column vector, `v'` is a row vector.
- If `M` is a matrix with m rows and n columns, `M'` is a matrix with n rows and m columns; where the first row becomes the first column, second row becomes the second column, etc.
- Just like in linear algebra, ***applying the transpose twice takes you back to the original array.***

Example 3: ***Without using MATLAB***, determine what the following commands will output.

```
>> aa=[5^2; 8; 1.2]
```

```
>> bb=aa'
```

```
>> M=[1 2; 3 4; 5 6];
>> MT=M'
```

```
>> MTT=MT'
```

2.5 Array Addressing

Table 3: Addressing Elements in an Array

The elements in an array can be addressed individually or in groups. The **address** or **index** is the element’s position in the variable. In MATLAB, addresses always start at 1 (not 0)!

- In rows, the address 1 refers to the leftmost position.
  - Similarly, you can use `end` for the rightmost position.
- In columns, the address 1 refers to the topmost position.
  - Similarly, you can use `end` for the bottom position.
- One-dimensional variables require one number to address an element.
- Two-dimensional variables require two numbers to address an element: the row address followed by the column address separated by a comma.

Table 4: How to Write an Addressing Command

```
variable_name(index) or variable_name(index1, index2)
```

Example 4: Let's try these commands in MATLAB!

```
>> VCT=[35 46 78 23 5 14 81 3 55];
>> VCT(4)
>> VCT(6)=273 %this will change the 6th element of VCT to 273
>> VCT(2)+VCT(8)
>> VCT(5)^VCT(8)+sqrt(VCT(7))
```

Question: Would these answers change if `VCT=[35 46 78 23 5 14 81 3 55]'` ?

Answer:

Example 5: Let's try another one with a matrix!

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
>> MAT(4) %What can you conclude after seeing this output?
>> MAT(3,1)
>> MAT(3,1)=20 %What did we change by executing this command?
>> MAT(2,4)-MAT(1,2)
```

Question: Would these answers change if `MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]'` ?

Answer:

On your own time, try the same commands with `MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]'`
Pay special attention to what happens when you execute `MAT(2,4)-MAT(1,2)`!

2.6 Using a Colon (:) in Addressing Arrays

Table 5: Addressing a Range of Elements in an Array

If `va` is a vector:

- `va(:)` address all elements in the vector
- `va(m:n)` address all elements m through n

If `A` is a matrix:

- `A(:)` will create a column vector (column by column) of all the elements of `A`
- `A(:,n)` address all elements the nth column
- `A(m,:)` address all elements of the mth row
- `A(:,m:n)` address all elements of the mth column ***through*** the nth column
- `A(m:n,:)` address all elements of the mth row ***through*** the nth row
- `A(m:n,p:q)` address all elements from rows m ***through*** n and columns p ***through*** q

Table 6: Addressing Specific Elements in an Array

If we don't want a full range, we can specify elements to pick out:

- `va([a b c:d])` address elements a, b, and c through d of the vector `va`
- `A([a b], [c:d e])` address elements in columns c through d and e of rows a and b in `A`

Notice the addition of brackets for this kind of addressing!

We can use vector/matrix indexing (or addressing) to change multiple elements at once, or to create new vectors/matrices from existing elements.

Example 6: Let's try the following commands in the command window!

```
>> A=[1:2:11; 2:2:12; 3:3:18; 4:4:24; 5:5:30]
>> q=A(:)
>> B=A(:,3)
>> C=A(2,:)
>> E=A(2:4,:)
>> F=A(1:3,2:4)
>> v=4:3:34
>> u=v([3,5,7:10])

>> clear,clc
>> A=[10:-1:4; ones(1,7); 2:2:14; zeros(1,7)]
>> B=A([1 3],[1 3 5:7])     %Same as B=A([1, 3],[1, 3, 5:7])
>> r=A(end, 2)
>> s=A(2, end)
```

2.7 Adding Elements to Existing Variables

Table 7: Adding Elements to Variables

1. Assign values to indices that don't exist. Then, MATLAB will expand the array filling new space with 0's.
2. Add values to ends of variables. This is what we call ***concatenating*** or ***appending***.
   - This becomes a tricky process because we have to be super careful with dimensions.
   - If we want to append variables from top to bottom, we must ensure all the variables have the same number of columns.
   - If we want to append variables from left to right, we must ensure all the variables have the same number of rows.

Example 7: Let’s try these in the command window!

```
>> DF=1:4  %this creates a row vector with four elements
>> DF(5:10)=10:5:35 %this adds the elements 10,15,…,35 to end of DF

>> AD=[5 7 2]
>> AD(8)=4 %what are elements four through seven equal to?
>> AD(9:12)=2:2:10 %what went wrong?

>> AW=[3 6 9; 8 5 11]
>> AW(4,5)=17

>> BG(3,4)=15

>> A2=[1 2 3; 4 5 6] %dimensions:
>> B2=[7 8; 9 10] %dimensions:
>> C2=eye(3) %dimensions:
>> Z=[A2 B2]
>> Z=[A2;C2]
>> Z=[A2;B2]
```

2.8 Deleting Elements

To delete elements in an existing variable, assign elements to empty brackets:

```
>> mtr=[5 78 4 24 9; 4 0 36 60 12; 56 13 5 89 3] %dimensions of mtr:
>> mtr(:,2:4)=[] %dimensions of mtr:
>> mtr(3,2)=[]
```

2.9 Built-In Function for Handling Arrays

Command	Description
length(A)	If A is a vector, outputs the number of elements. If A is a matrix, outputs the maximum dimension between rows and columns.
size(A)	Outputs a 1 by 2 vector with the number of rows and columns of the variable A.
reshape(A,m,n)	Changes the number of rows and columns of a matrix or vector while keeping number of elements the same; works column by column going downwards.
diag(v)	If v is a vector, outputs a square matrix of zeros with v as the main diagonal.
diag(A)	If A is a matrix, outputs a vector whose elements are the main diagonal of A.

Example 8: Let’s try these in the command window!

```
>> A=[1 2 3; 4 5 6]
>> length(A)
>> size(A)
>> diag(A)
>> B=reshape(A,3,2)
>> B=reshape(A,3,3)

>> v=[2:3:14]
>> length(v)
>> u=v’
>> length(u)
>> diag(v)
>> diag(u)
```

2.10 Strings and Strings as Variables

Table 8: A String

A string is an array of characters.

- You can create a string by typing the characters within **single quotes**: `'This is a string'`
- Strings can include letters, digits, other symbols, and spaces!
    - If a string contains a number, MATLAB does not recognize it as a number.
- To include a single quote within a string, type two single quotes in its place: `'I''m cool'`
- Strings are used in MATLAB to display text output, specify formatting for plots, input arguments for some functions, text input from user or data files (later chapters!).
- Characters of a string are stored in a row vector. Each character, including a space, is an element.

Note: Please DO NOT use double quotes!!!! Double quotes and single quotes create different types of objects!

Example 9:

```
>> S1='I''m cool'
>> S1(6)
>> length(S1), size(S1)
>> S2="I''m cool"
>> S2(6)
>> length(S2), size(S2)
>> whos
```

Example 10:

```
>> x='312'
>> 2*x     %Look into ASCII encoding for an explanation!
>> x(1), x(2), x(3)

>> word='dale'
>> word(1)='v'
>> word(end)=[]
>> word(end+1:end+3)='ley'
```

Table 9: Concatenating Strings

MATLAB stores strings with multiple lines as an array.

- Each line must have the same number of columns (i.e., characters).
- `char` function ***pads*** each line on the right with enough spaces so that all lines have the same number of characters:

  ```
  char('string 1', 'string 2', 'string 3')
  ```
- MATLAB will automatically make all rows as long as the longest row.

Example 11:

```
>> table=['john'; 'mark'; 'zayn']
>> size(table)

>> table2=['harry'; 'ron'; 'draco']
>> table2=['harry'; 'ron  '; 'draco'] %fix manually

>> table2=char('harry', 'ron', 'draco')
>> size(table2)

>> BigTable= char('Christine Deeb', 'Math 320', 'February 4')
```