

Computer Applications Lab Lab 2 Arrays in Matlab

Chapter 2
Sections 1,2,6,7

Dr. Iyad Jafar

Adapted from the publisher slides

Outline

- Introduction
- Arrays in Matlab
- Vectors and arrays
 - Creation
 - Addressing
 - Functions
- Cell and structure arrays

2

Introduction

- Matlab has the capability of handling arrays of numbers and many data types
- Array manipulation in Matlab is much simpler when compared to other programming languages ($C = A + B$ is done without writing loops)
- This makes Matlab the choice for many engineering application that require processing of data sets.

3

Frequently Used Arrays in Matlab

- **Numeric:**
 - Single and double precision
 - Signed integers; int8, int16, int32
 - Unsigned integers; uint8, uint16, uint32
- **Character:** array of strings
- **Logical:** contains '1' or '0' that corresponds to 'True' or 'False'
- **Cell and Structure** arrays
 - Data structures that allows the storage of different data types such as strings and numbers

4

Vectors in Matlab

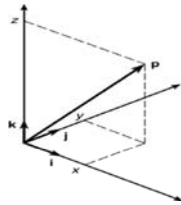
- Vectors are special case of arrays, with one of its dimensions being 1.
- The vector p can be specified by three components: x , y , and z , and is written as

$$p = xi + yj + zk$$

- In Matlab, vector p can be written as

$$p = [x, y, z]$$

- Matlab can use vectors having more than three elements.



NOTE

- We can define vectors in Matlab with more than three elements
- scalar values in Matlab are treated as a vector of size 1×1

Creating Vectors in Matlab

1. To create a row vector, separate the elements by *commas or spaces*. For example,

```
>> p = [3,7,9]
```

```
p =
     3     7     9
```

2. You can create a column vector by using the *transpose notation (')*.

```
>> p = [3,7,9]'
```

```
p =
     3
     7
     9
```

3. You can also create a column vector by separating the elements by *semicolons*. For example

```
>> g = [3;7;9]
```

6

Creating Vectors in Matlab

4. You can create larger vectors by *appending* one vector to another. For example, to create the row vector u whose first three columns contain the values of $r = [2,4,20]$ and whose fourth, fifth, and sixth columns contain the values of $w = [9,-6,3]$, you type $u = [r,w]$. The result is the vector $u = [2,4,20,9,-6,3]$.

NOTE

- In order to *append* vectors they should be either row vectors or column vectors. Otherwise, *transposition is required before appending*.

Example

```
>> u = [1,3,4] ;
>> r = [6;8;9] ;
>> z = [u,r']
z =
     1     3     4     6     8     9
```

7

Creating Vectors in Matlab

5. The colon operator ($:$) easily generates a large vector of *regularly spaced* elements.

Syntax:

- $x = [m;q:n]$ to create a vector x of values with a spacing q .
- The first value is m . The last value is n if $m-n$ is an integer multiple of q . If not, the last value is less than n .
- Number of elements = $((n-m)/q) + 1$

Example:

```
typing x = [0:2:8] creates the vector x = [0,2,4,6,8]
typing x = [0:2:7] creates the vector x = [0,2,4,6].
```

Default increment:

- If the increment q is omitted, it is presumed to be 1.
- Thus typing $y = [-3:2]$ produces the vector $y = [-3,-2,-1,0,1,2]$.

8

Creating Vectors in Matlab

6. The `linspace` command also creates a linearly spaced row vector, but instead you specify the number of elements rather than the increment.

Syntax:

- `linspace(x1,x2,n)` where $x1$ and $x2$ are the lower and upper limits and n is the number of points.
- Increment = $(x2-x1) / (n-1)$

Example:

- `linspace(5,8,3)` is equivalent to `[5:0.1:8]`.

Default spacing: If n is omitted, the spacing is 1.

9

Magnitude and Length of a Vector

- The `length(v)` command gives the *number of elements in the vector*.
- The magnitude of a vector x having elements x_1, x_2, \dots, x_n is a scalar, given by

$$\sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

and is the same as the vector's geometric length.

10

Matrices

- A matrix has multiple rows and columns. For example, the matrix

$$M = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \\ 8 & 4 & 9 \\ 3 & 12 & 15 \end{bmatrix}$$

has four rows and three columns.

- Vectors are special cases of matrices having one row or one column.

11

Creating Matrices in Matlab

1. For small matrices, type the elements such that

- Elements in the same row are separated by commas or spaces
- Rows are separated by semicolons

Example

```
>>A = [2,4,10;16,3,7];
```

Creates the matrix

$$A = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \end{bmatrix}$$

12

Creating Matrices in Matlab

Suppose $a = [1, 3, 5]$ and $b = [7, 9, 11]$ (row vectors). Note the difference between the results given by $[a \ b]$ and $[a;b]$ in the following session:

```
>>c = [a b];
c =
     1     3     5     7     9    11
>>D = [a;b]
D =
     1     3     5
     7     9    11
```

13

Addressing Array and Vector Elements

Row = 1 , column = 1

$$A = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \end{bmatrix}$$

Row = 2 , column = 3

- To access a single element use
 - Matrix $\rightarrow A(\text{row number}, \text{column number})$
 - Vectors $\rightarrow v(\text{element position})$
 - or
 - Row vectors $\rightarrow v(1, \text{column number})$
 - Column vectors $\rightarrow v(\text{row number}, 1)$

14

Addressing Array and Vector Elements

- To access a group of elements, rows, columns, or subarrays of arrays use the **colon symbol**

- **Examples**

- $v(:)$ represents all the row or column elements of the vector v .
- $v(2:5)$ represents the second through fifth elements; that is $v(2)$, $v(3)$, $v(4)$, $v(5)$.
- $A(:,3)$ denotes all the elements in the third column of the matrix A .
- $A(:,2:5)$ denotes all the elements in the second through fifth columns of A .
- $A(2:3,1:3)$ denotes all the elements in the second and third rows that are also in the first through third columns.

15

Array Addressing : Example

You can use array indices to extract a smaller array from another array. For example, if you first create the array B

$$B = \begin{bmatrix} 2 & 4 & 10 & 13 \\ 16 & 3 & 7 & 18 \\ 8 & 4 & 9 & 25 \\ 3 & 12 & 15 & 17 \end{bmatrix}$$

then type $C = B(2:3,1:3)$, you can produce the following array:

$$C = \begin{bmatrix} 16 & 3 & 7 \\ 8 & 4 & 9 \end{bmatrix}$$

16

Additional Array Functions

Function	Description
size(A)	Returns a row vector [m n] containing the size of the mxn array A
sort(A)	Sorts each column of the array A in ascending order and returns an array of same size as A
sum(A)	Sums the elements in each column of the array A and returns a row vector containing the sums
inv(A)	Computes the inverse of array A
diag(A)	Returns the elements along the main diagonal of A
fliplr(A)	Flips array A about its central column
flipud(A)	Flips array A about its central row

Exercise: open the Matlab help browser and read the documentation of the following array-related functions: find(A), max(A), min(A), cat(n,A,B,C)

Examples

```
>> A=[3 0 6 1; 0 2 4 5; 7 2 0 4]
A =
     3     0     6     1
     0     2     4     5
     7     2     0     4

>> size(A)
ans =
     3     4

>> sort(A)
ans =
     0     0     0     1
     3     2     4     4
     7     2     6     5

>> sum(A)
ans =
    10     4    10    10
```

18

Multidimensional Arrays

- Matlab supports multidimensional arrays.
- Examples
 - Three dimensional arrays have the dimension $m \times n \times q$
 - four dimensional arrays have the dimension $m \times n \times q \times r$
- *The first two dimensions are the row and the column.*
- *Higher dimensions are called pages.*

19

Multidimensional Arrays

- *Create multidimensional arrays by defining the pages then appending*
- **Example**

```
>> A = [1 , 2; 3 , 4] ;
>> A(:, :, 2) = [5 , 6; 7 , 8];
```

Creates a three dimensional array with two pages.

20

Cell Arrays

- A **cell array** is an array in which each element is a bin, or cell, which can contain an array.
- Information of different data types and dimensions can be stored in cell arrays.

Creation

- You can create a cell array by using the **cell function**

```
myCell = cell(m,n)
creates empty cell array C with mxn cells
```

- Using the assignment operator and the curly brackets to define its elements.

```
myCell(1,1) = {'CA LAB'} % creates the cell array my cell and initializes element (1,1)
myCell(1,2) = {0907311} % assign value to element (1,2)
```

21

Cell Arrays Creation

```
>> A(1,1)={'Walden Pond'};
>> A(1,2)={June 13, 1997};
>> A(2,1)={{60,72,65}};
>> A(2,2)={{[55,57,56;54,56,55;52,55,53]}};
>> A
```

```
A =
'Walden Pond' 'June 13, 1997'
[1x3 double] [3x3 double]
```

```
A{1,1} =
Walden Pond
```

```
A{2,1} =
60 72 65
```

```
A{1,2} =
June 13, 1997
```

```
A{2,2} =
55 57 56
54 56 55
52 55 53
```

22

Creating a 2x2 cell array

```
>> B={ [2,4], [6,-9;3,5]; [7;2],
10};
>> disp(B)
[1x2 double] [2x2 double]
[2x1 double] [ ] 10]
>> B
B =
[1x2 double] [2x2 double]
[2x1 double] [ ] 10]
```

Do not name a cell array with the same name as a previously used numeric array without first using the `clear` command! Use the `iscell` function to determine if an array is a cell array. And, use the `num2cell` function to convert a numeric array to a cell array.

Preallocating Empty Cell Arrays

You can preallocate cell arrays of a specified size by using the `cell` function.

```
>> C=cell(3,5)
C =
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

Now you can use assignment statements to enter the contents of the cells:

```
C(2,4)=[6,-3,7] puts a 1x3 array in cell (2,4)
C(1,5)=[1:10] puts the numbers from 1 to 10 in cell (1,5)
C(3,4)={'30 mph'} puts the string in cell (3,4)
```

23

Accessing Cell Arrays

Cell indexing (Using parenthesis)

- Extracts cell(s) and places them in a **new cell** array
- `Speed = C(3,4)` places the contents of cell (3,4) of the array C in the new variable Speed
- `D = C(1:3,2:5)` this places the contents of the cells in rows 1 to 3, columns 2 to 5 in the new cell array D. The new cell array will have three rows, four columns, and 12 arrays.
- Arithmetic operations are not allowed on cells data.** They have to be converted to numeric type first (use the double function), or use Content indexing to extract the content.

Content indexing (Using curly brackets)

- Extracts the content of the cell and places it in a variable of the same type
- `speed = C {3,4}` assigns '30 mph' to variable speed of type string

24

Structure Arrays

- Structure arrays are a class of arrays that allows you to store dissimilar arrays together. The elements in the structure are accessed using **named fields**.
- **Example**
 - In a student database (e.g. name, social security number, email address, test scores). There are four fields (4 field names): 3 string and 1 vector containing numerical elements.
 - A **structure** consists of all this information for a single student and a **structure array** is an array of such structures for different students.
- **Creation**
 - Using assignment statements
 - Using the **struct function**
 - Structure arrays use the dot notation (.) to specify and to access the fields.

25

Example

```
>> student.name='John Smith';
>> student.ssn='123-45-6789';
>> student.email='smithj@myschool.edu';
>> student.tests=[67,75,84];
>> student
student =
name: 'John Smith'
ssn: '123-45-6789'
email: 'smithj@myschool.edu'
tests: [67 75 84]
>> size(student)
ans=1 1
```

26

Example (cont'd)

```
>> student(2).name='Mary Jones';
>> student(2).ssn='987-65-4321';
>> student(2).email='jonesm@myschool.edu';
>> student(2).tests=[84,78,93];
>> student
student = 1x2 struct array with fields:
    name
    ssn
    email
    tests
```

27

Example (cont'd)

- The previous structure array can be constructed using the **struct function**

```
>> student(1) =struct('name',
'John Smith', 'SSN', '123-45-6789',
'email',
'smithj@myschool.edu', 'tests', [67,75,84])
>> student(2) =struct('name', Mary
Jones, 'SSN', '987-65-4321',
'email',
'jonesm@myschool.edu', 'tests', [84,78,93])
```

28

Structure Functions

Function	Description
<code>names = fieldnames(S)</code>	Returns the field names associated with structure array S as names; a cell array of strings
<code>F = getfield(S,'field')</code>	Returns the contents of the field 'field' in the structure array S.
<code>isfield(S,'field')</code>	Returns 1 if 'field' is the name of a field in the structure array S, and 0 otherwise
<code>S = rmfield(S,'field')</code>	Removes the field 'field' from the structure array S
<code>S = setfield(S,'field',V)</code>	Sets the contents of the field 'field' to the value V in the structure array S
<code>S = struct('f1',v1,'f2',v2,...)</code>	Creates a structure array with the fields 'f1','f2',... having the values 'v1','v2',...

29

Accessing Structure Arrays

- Type a period after the structure array name followed by the field name to access the contents of the field
`student(2).name` returns the value 'Mary Jones'
- You can assign the result to a variable in the usual way
`name2=student(2).name`
assigns the value 'Mary Jones' to the variable name2
- To add more information to an existing database
 - `student(1).phone='555=1653'`
 - This adds a phone number field to the first student's
 - All other structures in the array will also have the phone number field, but will contain an empty array until you give them values.
- To delete a field from every structure in the array, use the `rmfield` function.

`new_struc = rmfield(array,'field')`

30