04-Arrays Part 2

text: Chapter 2.5-2.7

ECEGR 101
Engineering Problem Solving with Matlab
Professor Henry Louie

Overview

- Array Size
- Array Addressing (single elements)
- Array Addressing (multiple elements)
- Adding Elements

Finding Out Array Size

- size(A) returns vector [m,n] where mxn is the size of the array A.
  - M: number of rows
  - N: number of columns

memorize this convention

```
>> x = [1 3 5; 6 9 7]
x =
    1    3    5
    6    9    7
>> size(x)
ans =
    2    3
```

```
>> x = [1 6 4 3 9 0 7 8 6]
x =
    1    6    4    3    9    0    7    8    6
>> size(x)
ans =
    1    9
```


Finding Out Array Size

- length(x) returns the number of elements in the vector x.
- length(A) returns the length of the longest dimension of the matrix A.

```
>> x = 1 : 5
x =
    1    2    3    4    5
>> length(x)
ans =
    5
```

```
>> x = 1 : 5; x = x'
x =
    1
    2
    3
    4
    5
>> length(x)
ans =
    5
```

```
>> x = [1 3 0 5; 6 9 7 0]
x =
    1    3    0    5
    6    9    7    0
>> length(x)
ans =
    4
```

Finding Out Array Size

- numel(x) counts the number of elements in x, regardless of dimension

```
>> x=[0 9 pi; 4 8 7; 100 25 40; 80 -8 40]

x =

         0    9.0000    3.1416
    4.0000    8.0000    7.0000
  100.0000   25.0000   40.0000
   80.0000   -8.0000   40.0000

>> numel(x)

ans =

    12
```

Numel is usually preferred over the use of length

Finding Out Array Size

- whos shows all variables, size, and other attributes

```
>> x=[0 9 pi; 4 8 7; 100 25 40; 80 -8 40];
>> y=[4 5];
>> whos
  Name        Size              Bytes  Class     Attributes

  ans         1x1                   8  double
  x           4x3                  96  double
  y           1x2                  16  double
```

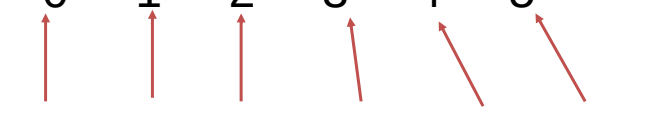
Array Addressing

Array elements can be accessed individually or in groups.

Array Addressing: Vectors

Individual addressing:

```
>> x = 0:5
x =
     0     1     2     3     4     5
```



first index is always 1, not 0

Array Addressing: Vectors

Examples:

```
>> x = [1 6 4 3 9 0 7 8 6]
x =
    1    6    4    3    9    0    7    8    6
>> x(2)
ans =
    6
>> x(6)
ans =
    0
>> x(1)
ans =
    1
>> x(end)
ans =
    6
```

```
>> y = x(4) + x(8)
y =
   11
>>
>> x(5) = 0
x =
    1    6    4    3    0    0    7    8    6
```


Exercise

If x = 0:5 then what is the result of the following commands?

1. y = x(1) + x(3)
2. z = x(2) + x(end)
3. x(1) = -8
4. x(end) = 12
5. w = x(1) + x(end)

Do not use Matlab

Exercise

```
>> x = 0:5
x =
    0    1    2    3    4    5
```

y = x(1) + x(3)

Exercise

```
>> x = 0:5
x =
    0    1    2    3    4    5
```

```
>> y = x(1) + x(3)

y =

    2
```

Exercise

```
>> x = 0:5
x =
    0    1    2    3    4    5
```

```
>> z = x(2) + x(end)

z =

    6
```


Exercise

```
>> x = 0:5
x =
    0    1    2    3    4    5
```

```
>> x(1) = -8

x =

   -8    1    2    3    4    5
```

Exercise

```
>> x = 0:5
x =
    0    1    2    3    4    5
```

Note: we redefined x

```
>> x(end) = 12

x =

    0    1    2    3    4   12
```


Exercise

```
>> x = 0:5
x =
    0    1    2    3    4    5
```

```
>> w = x(1) + x(end)

w =

    5
```

Exercise

If x = 3:10 then:

1. What is x(0)?
2. What is x(end)?
3. What is x(1)?

Exercise

If x = 3:10 then:

1. What is x(0)?
2. What is x(end)?
3. What is x(1)?

```
>> x = 3:10

x =

   3   4   5   6   7   8   9   10
```

Exercise

If x = 3:10 then:

1. What is x(0)?
2. What is x(end)?
3. What is x(1)?

```
>> x(0)
??? Subscript indices must
      either be real positive
      integers or logicals.
```

```
>> x = 3:10

x =

   3   4   5   6   7   8   9   10
```


Exercise

If x = 3:10 then:

1. What is x(0)?
2. What is x(end)?
3. What is x(1)?

```
>> x(end)

ans =

   10
```

```
>> x = 3:10

x =

   3   4   5   6   7   8   9   10
```

Exercise

If x = 3:10 then:

1. What is x(0)?
2. What is x(end)?
3. What is x(1)?

```
>> x(1)

ans =

    3
```

```
>> x = 3:10

x =

    3    4    5    6    7    8    9   10
```


Exercise

What is the difference between the following commands?

1. x = 3:10
2. x = [3:10]
3. x = (3:10)

Exercise

```
>> x = 3:10

x =
    3    4    5    6    7    8    9   10

>> x = [3:10]

x =
    3    4    5    6    7    8    9   10

>> x = (3:10)

x =
    3    4    5    6    7    8    9   10
```

They are the same!

Array Addressing: Vectors

- The colon operator (:) is used to access groups of consecutive elements.

```
>> x = [1 6 4 3 9 0 7 8 6]
x =
    1    6    4    3    9    0    7    8    6
```

```
>> x(:)
ans =
    1
    6
    4
    3
    9
    0
    7
    8
    6
```

```
>> x(1:3)
ans =
    1    6    4
>> x(6:end)
ans =
    0    7    8    6
```

```
>> x(3:5) = -1
x =
    1    6   -1   -1   -1    0    7    8    6
```

Array Addressing: Matrices

- If x is a matrix, x(i,j) refers to the element in row i and column j.

```
>> a = [1 4 7 8 2; 2 5 4 7 0]
a =
    1    4    7    8    2
    2    5    4    7    0
>> a(2, 3)
ans =
    4
>> a(1,4)
ans =
    8
>>
>>
>> a(3,2)
??? Index exceeds matrix dimensions.
```

```
>> a(2,3) = 0
a =
    1    4    7    8    2
    2    5    0    7    0
>> a(2,1) = a(2,2) + a(2,4)
a =
    1    4    7    8    2
   12    5    0    7    0
```


Array Addressing: Matrices

The colon operator (:) can be used to access a range of elements in a matrix:

A(:,n) elements in all the rows of column n
A(n,:) elements in all the columns of row n

A(:, m:n) elements in all the rows between columns m and n
A(m:n, :) elements in all the columns between rows m and n

A(m:n, p:q) elements in rows m through n and columns p through q

Array Addressing: Matrices

Examples:

```
>> x = [1 4 5 3 2; 4 6 7 4 4; 8 8 4 3 0]
x =
    1    4    5    3    2
    4    6    7    4    4
    8    8    4    3    0
```

```
>> x(:,2)
ans =
    4
    6
    8
```

```
>> x(:,2:3)
ans =
    4    5
    6    7
    8    4
```

```
>> x(1:2,:)
ans =
    1    4    5    3    2
    4    6    7    4    4
```

```
>> x(3,:)
ans =
    8    8    4    3    0
```

```
>> x(1:2,3:5)
ans =
    5    3    2
    7    4    4
```


Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1   -3.2    3.4    0.6
    0.6    1.1   -0.6    3.1
    1.3    0.6    5.5    0
```

What is: c(2,:)

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1   -3.2    3.4    0.6
    0.6    1.1   -0.6    3.1
    1.3    0.6    5.5      0
```

c(2,:)

Second row, all columns

```
>> c(2,:)
ans =
    0.6    1.1   -0.6    3.1
```


Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1   -3.2    3.4    0.6
    0.6    1.1   -0.6    3.1
    1.3    0.6    5.5      0
```

What is c(:,end)

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1    -3.2     3.4     0.6
    0.6     1.1    -0.6     3.1
    1.3     0.6     5.5       0
```

```
>> c(:,end)
ans =
    0.6
    3.1
      0
```

All rows, last column

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1    -3.2     3.4     0.6
    0.6     1.1    -0.6     3.1
    1.3     0.6     5.5       0
```

What is c(1:2,2:end)

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1000   -3.2000    3.4000    0.6000
    0.6000    1.1000   -0.6000    3.1000
    1.3000    0.6000    5.5000         0
```

```
>> c(1:2,2:end)
ans =
   -3.2000    3.4000    0.6000
    1.1000   -0.6000    3.1000
```

First and second row, second through last column

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1000   -3.2000    3.4000    0.6000
    0.6000    1.1000   -0.6000    3.1000
    1.3000    0.6000    5.5000         0
```

What is c(6)

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1    -3.2     3.4     0.6
    0.6     1.1    -0.6     3.1
    1.3     0.6     5.5       0
```

```
>> c(6)
ans =
    0.6
```

Sixth element, counting down rows, then columns

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1    -3.2     3.4     0.6
    0.6     1.1    -0.6     3.1
    1.3     0.6     5.5       0
```

What is c(4:end)

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
       1.1      -3.2       3.4       0.6
       0.6       1.1      -0.6       3.1
       1.3       0.6       5.5         0
```

Fourth through last elements

```
>> c(4:end)
ans =
  Columns 1 through 7
      -3.2       1.1       0.6       3.4      -0.6       5.5       0.6
  Columns 8 through 9
       3.1         0
```


Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
       1.1      -3.2       3.4       0.6
       0.6       1.1      -0.6       3.1
       1.3       0.6       5.5         0
```

What is c([1 3],2)

Hint: You can list the rows you want inside brackets

Exercise

Assume that array c is defined as shown, and determine the contents of the following sub-arrays:

```
>> c = [1.1 -3.2 3.4 0.6; 0.6 1.1 -0.6 3.1; 1.3 0.6 5.5 0.0]
c =
    1.1000   -3.2000    3.4000    0.6000
    0.6000    1.1000   -0.6000    3.1000
    1.3000    0.6000    5.5000         0
```

```
>> c([1 3],2)
ans =
   -3.2000
    0.6000
```

Rows 1 and 3, second column