03-Arrays Part 1

text: Chapter 2.1-2.4

ECEGR 101
Engineering Problem Solving with Matlab
Professor Henry Louie

Overview

- Matrices and Vectors
- Creating Vectors
- Creating Matrices
- Using Built-in Functions to Create Arrays

Arrays: Matrices and Vectors

MATLAB can store several numerical values in a single variable.

- Matrix – rectangular object consisting of rows and columns
- Vector – special type of matrix, having only one row, or one column
- Array – any matrix, vector, or scalar

Arrays: Matrices and Vectors

Vectors

3 x 1

$$\begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$$

rows ↓

1 x 3

$$\begin{bmatrix} 1 & 2 & 4 \end{bmatrix}$$

→ column

Matrices

2 x 3

$$\begin{bmatrix} 0 & -1 & 2 \\ 1 & 3 & 4 \end{bmatrix}$$

3 x 2

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

3 x 3

$$\begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$$

By Matlab convention, rows x columns

Arrays: Matrices and Vectors

- All variables in MATLAB are called arrays (scalar is an array with one element)
- Arrays are created by assigning values to them
- No need to define the size of the array before creating it

Creating Vectors From a List of Elements

1. Type the elements:
   variable_name = [vector elements]

```
>> x = [1 3 0 -1 5]

x =

     1     3     0    -1     5

>> x = [1,3,0,-1,5]

x =

     1     3     0    -1     5
```

row vectors

```
>> x = [1; 3; 0; -1; 5]

x =

     1
     3
     0
    -1
     5
```

column vector

Creating Vectors with Constant Spacing

2. Specify the first term, the spacing, and the last term:

   variable_name = m : q : n

```
>> x = 1:5

x =

     1     2     3     4     5

>> x = 1:0.5:3

x =

    1.0000    1.5000    2.0000    2.5000    3.0000
```

Default spacing is 1.

Exercise

What do you get with the following command?

```
>> y = [10: -5: -20]
```

Exercise

What do you get with the following command?

```
>> y = [10: -5: -20]

y =

 10     5     0    -5   -10   -15   -20
```


Exercise

What will be the last element of a vector created with the following command?

```
>> x = 0:2:11
```

Exercise

What will be the last element of a vector created with the following command?

```
>> x = 0:2:11

x =

 0     2     4     6     8    10
```


Creating Vectors with Constant Spacing

3. Specify the first (xi) and last (xf) terms and the number of terms (n):

   variable_name = linspace(xi, xf, n)

```
>> y = linspace(-0.5, 0.5, 11)
y =
  Columns 1 through 5
      -0.5       -0.4       -0.3       -0.2       -0.1
  Columns 6 through 10
         0        0.1        0.2        0.3        0.4
  Column 11
       0.5
```

See also "logspace"

Exercise

Create a row vector a1 starting with the element 1, ending with the element 10, with five equally spaced points in the vector.

Exercise

Create a row vector a1 starting with the element 1, ending with the element 10, with five equally spaced points in the vector.

```
>> a1 = linspace(1, 10, 5)
a1 =
        1       3.25        5.5       7.75         10
```

or

```
>> a2 = 1:2.25:10
a2 =
        1       3.25        5.5       7.75         10
```

Creating Matrices

All rows/columns must have the same number of elements.

variable_name = [1st row elements; 2nd row elements; 3rd row elements; … ; last row elements]

```
>> a = [1 2 3; 4 5 6]

a =

     1     2     3
     4     5     6
```

rectangular matrix (2x3)

```
>> x = [ 1 : 4; 5 : 8; 9 : 12; 13 : 16]
x =
    1    2    3    4
    5    6    7    8
    9   10   11   12
   13   14   15   16
```

square matrix (4x4)

Exercise

Using the constant spacing commands create a matrix (with a variable name) with the following rows:

- First row: from 1 to 9 with 5 elements
- Second row: from 50 to 35 with 5 elements

Exercise

```
>> x = [linspace(1,9,5); linspace(50, 35, 5)]

x =

    1.00    3.00    5.00    7.00    9.00
   50.00   46.25   42.50   38.75   35.00
```


Creating Arrays Using Built-In Functions

- zeros(m,n) and ones(m,n) create an mxn matrix with all the elements equal to 0 and 1.

```
>> x = zeros(3, 4)
x =
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
>> y = ones(4, 3)
y =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
```

Exercise

Create two arrays:
- a row vector of length 4 with all zeros
- a column vector of length 6 with all ones

Exercise

Create two arrays:
- a row vector of length 4 with all zeros
- a column vector of length 6 with all ones

```
>> x = zeros(1,4)
x =
     0     0     0     0
```

Exercise

Create two arrays:
- a row vector of length 4 with all zeros
- a column vector of length 6 with all ones

```
>> x = ones(6,1)
x =
    1.00
    1.00
    1.00
    1.00
    1.00
    1.00
```


Creating Arrays Using Built-In Functions

- eye(n) creates an nxn identity matrix.

```
>> a = eye(5)
a =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Creating Arrays Using Built-In Functions

diag(v) creates a square matrix with the elements of v in the diagonal (v must be a vector)

```
>> x = 0.5 : 0.1 : 1.0
x =
       0.5       0.6       0.7       0.8       0.9         1
>> y = diag(x)
y =
       0.5         0         0         0         0         0
         0       0.6         0         0         0         0
         0         0       0.7         0         0         0
         0         0         0       0.8         0         0
         0         0         0         0       0.9         0
         0         0         0         0         0         1
```


Creating Arrays Using Built-In Functions

diag(A) creates a vector from the diagonal elements of A (A must be a matrix)

```
>> x = [1 2 3 4; 4 3 2 1; 5 6 7 8; 8 7 6 5]
x =
    1    2    3    4
    4    3    2    1
    5    6    7    8
    8    7    6    5
>> z = diag(x)
z =
    1
    3
    7
    5
```

Transpose Operator

The transpose operator (') switches the rows (columns) to columns (rows) in a matrix.

```
>> x = [1 2 3 4; 10 20 30 40]
x =
    1    2    3    4
   10   20   30   40
>>
>> y = x'
y =
    1   10
    2   20
    3   30
    4   40
```

```
>> x = [1 1 1 1; 2 2 2 2; 3 3 3 3; 4 4 4 4]
x =
    1    1    1    1
    2    2    2    2
    3    3    3    3
    4    4    4    4
>>
>> z = x'
z =
    1    2    3    4
    1    2    3    4
    1    2    3    4
    1    2    3    4
```


Reshaping Arrays

- reshape(A,m,n) rearrange matrix A to have m rows and n columns.
- The number of elements in the input and output matrices is the same.

```
>> x = 1:12
x =
    1    2    3    4    5    6    7    8    9   10   11   12
>> y = reshape(x, 3, 4)
y =
    1    4    7   10
    2    5    8   11
    3    6    9   12
```

Exercise

- Given the following vector x, use the reshape command to generate all possible arrays all with different dimensions. x = [4 5 6 7 8 9]

Exercise

- Given the following vector x, use the reshape command to generate all possible arrays all with different dimensions. x = [4 5 6 7 8 9]

```
>> x = [4 5 6 7 8 9]
x =
     4     5     6     7     8     9
```

```
>> reshape(x, 2,3)
ans =
     4     6     8
     5     7     9
>>
>> reshape(x, 3,2)
ans =
     4     7
     5     8
     6     9
```

```
>> reshape(x, 6,1)
ans =
     4
     5
     6
     7
     8
     9
```

See Also

- circshift: circularly shifts values in an array by a given number of elements
- permute: general permutation of matrix dimensions
- transpose: same as using the ' command
- fliplr: flips columns in the left/right direction
- flipud: flips rows in the up/down direction
- rot90: 90 degree counterclockwise rotation of a matrix
- repmat: repeats a matrix in a tiling manner