COGS 119/219
MATLAB for Experimental Research

Fall 2014 – Week 1

Vectors, Matrices and Matrix Operations

In Matlab

Matlab is more than a calculator!

- The array is a fundamental form that MATLAB uses to store and manipulate data.
- An array is a list of numbers arranged in rows or columns.

Vector

The simplest array (one-dimensional) is a row, or column of numbers, aka vector.

```
>>  x = [1 2 3 4]
   x =
        1 2 3 4
```

Row vector

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

Column vector

Vector

```
>> a = 1:4
a =
     1   2   3   4

>> b = [1 : 2 : 9]
b =
     1   3   5   7   9
>> c = [21 : -3 : 6]
c =
    21   18   15   12   9   6
```

Matrix

- A more complex array (two-dimensional) is a collection of numbers arranged in rows and columns.

```
>> x = [1 2 3 ; 5 1 4 ; 3 2 -1]
x =
     1   2   3
     5   1   4
     3   2  -1
```

Matrix

- A more complex array (two-dimensional) is a collection of numbers arranged in rows and columns.

```
>> x = [1 2 3 ; 5 1 4 ; 3 2 -1]
x =
     1   2   3
     5   1   4
     3   2  -1
```

Matrix

- A more complex array (two-dimensional) is a collection of numbers arranged in rows and columns.

```
>> x = [1 2 3 ; 5 1 4 ; 3 2 -1]
x =
```

columns

rows

1	2	3
5	1	4
3	2	-1

3 rows and 3 columns,
3 x 3 matrix

Matrix

```
>> y = [ 10 4 6
5 2 8
3 7 12
9 4 15]
y =
      10   4   6
       5   2   8
       3   7  12
       9   4  15
```

ENTER key is pressed before a new line is entered

4 x 3 matrix

NOTE: All the rows in a matrix must have the same number of elements.

Special Matrices

zeros(m, n)

Creates an m x n matrix with elements 0.

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

Special Matrices

ones(m, n)

Creates an m x n matrix with elements 1.

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

Special Matrices

eye(n) ⟶ Identity matrix

Creates an n x n matrix with diagonal elements 1, and other elements 0.

```
>> e = eye (4)
e =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Transpose operator (')

- When applied to a vector, it switches a row vector to a column vector, and a column vector to a row vector.

```
>> aa = [3  8  1]
aa =
     3  8  1
>> bb = aa'
bb =
     3
     8
     1
```

bb is the transpose vector of aa

Transpose operator (')

- When applied to a matrix, it switches the rows and columns.

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

>> n = m'
n =
        1 4
        2 5
        3 6
```

n is the tranpose of m

Addressing array elements

- Elements in an array (either vector or matrix) can be addressed individually or in subgroups.

VECTORS

- The address of an element in a vector is its position in the row (or column).

- For a vector v, v(k) refers to the element in position k.

Addressing vector elements

```
>> v = [35  46  78  23  5  14];

>> v(1)
ans =
        35

>> v(4)
ans =
        23
```

Addressing vector elements

- It is possible to change the value of vector elements by reassigning a new value to a specific address.

```
>> v = [ 12  14  16  18  20];
>> v (3) = 30;
>> v
        12   14   30   18   20
```

Addressing vector elements

- It is possible to use vector elements in mathematical expressions.

```
>> v

    12   14   30   18   20

>> x = v(2) + v(5)
x =

    34
```

Addressing matrix elements

- The address of an element in a matrix is its position, defined by the row number and the column number where it is located.

- For matrix m, m(i,j) refers to the element in row i and column j.

Addressing matrix elements

```
>> m = [3  11  6  5 ; 4  7  10  2 ; 13  9  0  8]
m =
     3   11   6   5
     4    7   10   2
    13    9    0   8
```

m(1,1) = ?

m(2, 3) = ?

m(3, 2) = ?

Addressing matrix elements

```
>> m = [3  11  6  5 ; 4  7  10  2 ; 13  9  0  8]
m =
     3  11  6   5
     4   7  10  2
    13   9  0   8
```

m(1,1) = 3

m(2, 3) = 10

m(3, 2) = 9

Using a colon (:) in addressing arrays

- A colon can be used to address a range of elements in a vector or matrix.

FOR A VECTOR:

- v(:) -> refers to all the elements of the vector v
- v(m:n) -> refers to elements m through n of the vector v

```
>> v = [4  15  8  12  34  2  50  23  11];
>> u = v(3:7)
u =
        8   12   34   2   50
```

Using a colon (:) in addressing arrays

FOR A MATRIX:

A(:,n) -> refers to the elements in all the rows of column n of the matrix A.

```
>> A = [9  8  7  6 ; 5  4  3  2; 1  0  9  8]
A =
        9  8  7  6
        5  4  3  2
        1  0  9  8
```

```
>> B = A (:, 3)
B =
        7
        3
        9
```

Using a colon (:) in addressing arrays

A(n,:) -> refers to the elements in all the columns of row n of the matrix A.

```
>> A = [9  8  7  6 ; 5  4  3  2; 1  0  9  8]
A =
        9  8  7  6
        5  4  3  2
        1  0  9  8

>> C = A (2, :)
C =
        5  4  3  2
```

Using a colon (:) in addressing arrays

A(: , m : n) -> refers to the elements in all the rows between columns m and n of the matrix A.

```
>> A = [9  8  7  6 ; 5  4  3  2; 1  0  9  8]
A =
     9  8  7  6
     5  4  3  2
     1  0  9  8
```

```
>> D = A(: , 3:4)
D =
     7  6
     3  2
     9  8
```

Using a colon (:) in addressing arrays

A(m : n , :) -> refers to the elements in all the columns between rows m and n of the matrix A.

```
>> A = [9  8  7  6 ; 5  4  3  2;  1  0  9  8]
A =
     9  8  7  6
     5  4  3  2
     1  0  9  8

>> E = A(2:3 , :)
E =
     5  4  3  2
     1  0  9  8
```

Using a colon (:) in addressing arrays

A(m : n , p : q) -> refers to the elements in rows m through n and columns p through q of the matrix A.

```
>> A = [9  8  7  6 ; 5  4  3  2; 1  0  9  8]
A =
        9  8  7  6
        5  4  3  2
        1  0  9  8

>> F = A(1:2 , 2:3)
```

Using a colon (:) in addressing arrays

A(m : n , p : q) -> refers to the elements in rows m through n and columns p through q of the matrix A.

```
>> A = [9  8  7  6 ; 5  4  3  2; 1  0  9  8]
A =
     9  8  7  6
     5  4  3  2
     1  0  9  8
```

```
>> F = A(1:2 , 2:3)
F =
     8  7
     4  3
```

Appending two vectors

```
>> v1 = [1  2  3  4]
v1 =
        1  2  3  4

>> v2 = [5  6  7  8]
v2 =
        5  6  7  8

>> v3 = [v1  v2]
```

Appending two vectors

```
>> v1 = [1  2  3  4]
v1 =
        1  2  3  4

>> v2 = [5  6  7  8]
v2 =
        5  6  7  8

>> v3 = [v1  v2]
v3 =
        1  2  3  4  5  6  7  8
```

Appending two vectors

```
>> v4 = [v1 ; v2]
v4 =
     1  2  3  4
     5  6  7  8
```

Appending two matrices

```
>> m1 = [1  2  3 ; 4  5  6 ]
m1 =
        1  2  3
        4  5  6
>> m2 = eye(2)
m2 =
        1  0
        0  1

>> m3 = [m1  m2]
m3 =
        1  2  3  1  0
        4  5  6  0  1
```

Try m3 = [m1; m2] – you will get an error about matrix sizes

Appending two matrices

```
>> m4 = [ 7 8 9 ; 6 5 4]
m4 =
        7 8 9
        6 5 4

>> m5 = [m1 ; m4]
m5 =
        1  2  3
        4  5  6
        7  8  9
        6  5  4
```

Built-in functions for handling arrays

Function	Description	Example
length(A) Try >> help length	Returns the number of elements in the vector A	>> A = [5 9 2 4]; >> length(A) ans = 4
size(A) Try >> help size	Returns a row vector [m , n], where m and n are the size of m x n of the array A	>> A = 6 1 4 ; 5 19 6 A = 6 1 4 5 19 6 >> size(A) ans = 2 3

Built-in functions for handling arrays

Function	Description	Example
diag(v) Try >> help diag	When v is a vector, creates a square matrix with the elements of v in the diagonal	>> v = [7 4 2]; >> A = diag(v) A = 7 0 0 0 4 0 0 0 2
diag(A)	When A is a matrix, creates a vector from the diagonal elements of A	>> A =[1 2 3 ; 4 5 6 ; 7 8 9] A = 1 2 3 4 5 6 7 8 9 >> vec = diag(A) vec = 1 5 9

Examples

```
>> diagmat = diag(4)
diagmat =
        4

>> diagmat = diag ([4 3 1])
diagmat =
        4  0  0
        0  3  0
        0  0  1
```

Examples

```
>> diagmat3 = diag([ 4 3  1; 5 6 7; 2 3 4]);
diagmat3 =
          4
          6
          4

>> diagmat2 = diag([4 3  1])
diagmat2 =
          4  0  0
          0  3  0
          0  0  1
```

Examples

```
>> diagmat3 = diag([ 4 3 1; 5 6 7; 2 3 4]);
diagmat3 =
     4
     6
     4
```

```
>> diagmat2 = diag([4 3 1])
diagmat2 =
     4  0  0
     0  3  0
     0  0  1
```

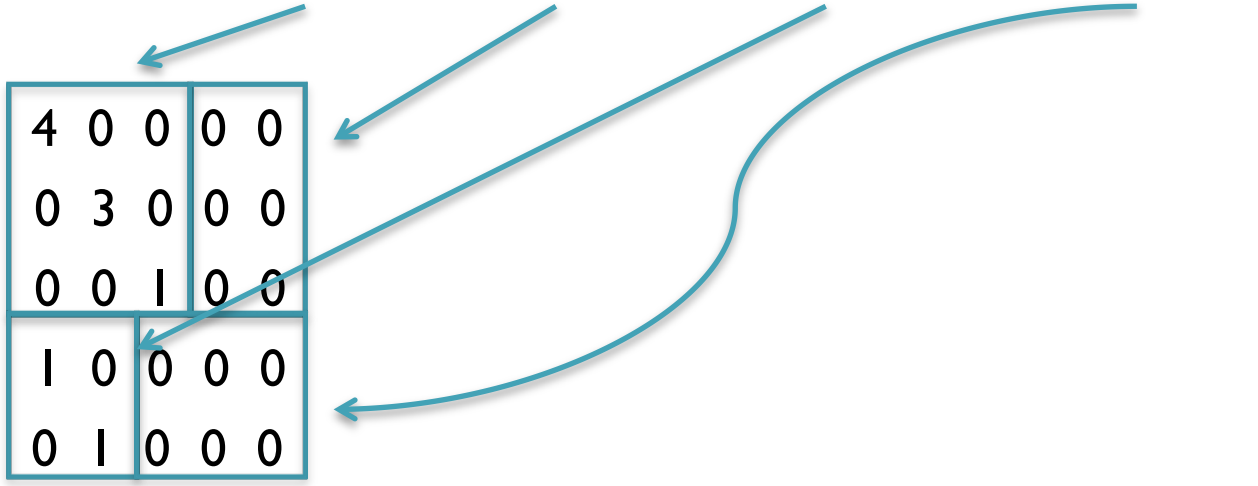
```
>> diag(diag(diagmat2))
ans =
     4  0  0
     0  3  0
     0  0  1
```

Examples

```
>> combinedmat = [diagmat2    zeros(3,2) ;   eye(2)    zeros(2,3)]
```

Examples

```
>> combinedmat = [diagmat2    zeros(3,2) ;   eye(2)    zeros(2,3)]
```



Matrix Operations

- Addition and subtraction operate entry-wise. Matrices need to be the same size.

```
>> A = [1 2 ; 3 4],    B = [ 1 1 ; 1 1];

>> C = A + B
C =
        2   3
        4   5
```

Online tutorial:
http://patrickjmt.com/matrices-basic-matrix-operations-add-subtract-multiply-by-constant/

Matrix Operations

- Multiplication, division, and to-the-power operate matrix-wise.

```
>> A = [1 2 ; 3 4],    B = [ 1 1 ; 1 1];

>> C = A * B
C =
        3   3
        7   7
```

Online tutorial:
http://patrickjmt.com/matrices-multiplying-a-matrix-by-another-matrix/

Matrix Operations

- To do element-by-element operations, use .* , ./, .^

```
>> A = [1 2 ; 3 4],    B = [ 1  1 ; 1  1];

>> C = A .* B
C =
        1   2
        3   4
```

Logical Operators

==	equal
~=	not equal
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
&	AND
|	OR
~	NOT

true and false

- true and false are functions

```
>> dat = 1;

>> true(dat)
ans =
        1

>> false(dat)
ans =
        0
```

See m file example

- Example of repmat (see help repmat)