البرمجة الإجرائية

Lecture No. 5

Matrix(2)

ميكاترونيك-سنة أولى-فصل أول

Dr. Eng. Essa Alghannam
Ph.D. Degree in Mechatronics Engineering

2024

Additional Array Functions

[u,v,w] = find(A)	Computes the arrays u and v, containing the row and column indices of the nonzero elements of the matrix *A*, and the array w, containing the values of the nonzero elements. The array w may be omitted.
length(A)	Computes either the number of elements of A if A is a vector or the largest value of m or n if A is an *m* × *n* *m*atrix.

Additional Array Functions

max(A)

Returns the algebraically largest element in A if **A** is a vector.

Returns a row vector containing the largest elements in each column if **A** is a matrix.

If any of the elements are complex, max(A) returns the elements that have the largest magnitudes.

Additional Array Functions

[x,k] = max(A)

Similar to max(A) but stores the maximum values in the row vector x and their indices in the row vector k.

min(A)

Like max but returns minimum values.

and

[x,k] = min(A)

Additional Array Functions

```
>> S=[ 1 2 3 0 5 6;7 8 9 10 0 12; 13 14 0 16
17 18; 0 20 21 0 23 24; 0 26 0 28 29 30]

S =

   1   2   3   0   5   6
   7   8   9  10   0  12
  13  14   0  16  17  18
   0  20  21   0  23  24
   0  26   0  28  29  30

>> find(S)
```

Linear Indexing of nonzero elements

```
ans =

     1
     2
     3
     6
     7
     8
     9
    10
    11
    12
    14
    17
    18
    20
    21
    23
    24
    25
    26
    27
    28
    29
    30
```

row and column subscripts of nonzero elements

```
>> [u v w]=find(S)

u =     v =     w =

     1       1       1
     2       1       7
     3       1      13
     1       2       2
     2       2       8
     3       2      14
     4       2      20
     5       2      26
     1       3       3
     2       3       9
     4       3      21
     2       4      10
     3       4      16
     5       4      28
     1       5       5
     3       5      17
     4       5      23
     5       5      29
     1       6       6
     2       6      12
     3       6      18
     4       6      24
     5       6      30
```

Additional Array Functions

X =

```
 1   3   5
 7   0  11
13  15  17
19   0  23
25  27  29
 2   4   6
 8  10   0
14  16  18
20  22   0
26  28  30
```

```
>> t=max(X)
t =
   26   28   30

>> d=min(X)
d =
    1    0    0

>> [f1,f2]=max(X)
f1 =
   26   28   30


f2 =
   10   10   10
```

```
>> [f1,f2]=min(X)
f1 =
    1    0    0


f2 =
    1    2    7
```

```
>> INDEX=find(X>25)

INDEX =

   10
   15
   20
   25
   30
```

```
>> INDEX=find(X==25)

INDEX =

    5
```

Additional Array Functions

```
>> INDEX=find(X>25 && X==25)
```

Operands to the || and && operators must be convertible to logical scalar values.

```
>> INDEX=find(X>25 & X==25)

INDEX =

  0×1 empty double column vector
```

```
>> INDEX=find(X>25 | X==25)

INDEX =

     5
    10
    15
    20
    25
    30
```

Additional Array Functions

size(A)	Returns a row vector [m n] containing the sizes of the *m* x *n* array A.
sort(A)	Sorts each column of the array A in ascending order and returns an array the same size as A.
sum(A)	Sums the elements in each column of the array A and returns a row vector containing the sums.

The function size(A) returns a row vector [m n] containing the sizes of the $m \times n$ array **A**. The length(A) function computes either the number of elements of **A** if A is a vector or the largest value of m or n if **A** is an $m \times n$ matrix.

For example, if

$$\mathbf{A} = \begin{bmatrix} 6 & 2 \\ -10 & -5 \\ 3 & 0 \end{bmatrix}$$

then max(A) returns the vector [6,2];

min(A)returns the vector [-10, -5];

size(A) returns [3, 2];

and length(A) returns 3.

Additional Array Functions

```
>> X

X =

    1    3    5
    7    0   11
   13   15   17
   19    0   23
   25   27   29
    2    4    6
    8   10    0
   14   16   18
   20   22    0
   26   28   30
```

```
>> X=sort(X)

X =

    1    0    0
    2    0    0
    7    3    5
    8    4    6
   13   10   11
   14   15   17
   19   16   18
   20   22   23
   25   27   29
   26   28   30
```

```
>> sum(X)

ans =

   135   125   139
```

```
>> size(X)

ans =

    10     3
```

```
>> length(X)

ans =

    10
```

Additional Array Functions

```
>> prod(X)

ans =

   1.0e+09 *

    5.0348         0         0
```

Multidimensional Arrays

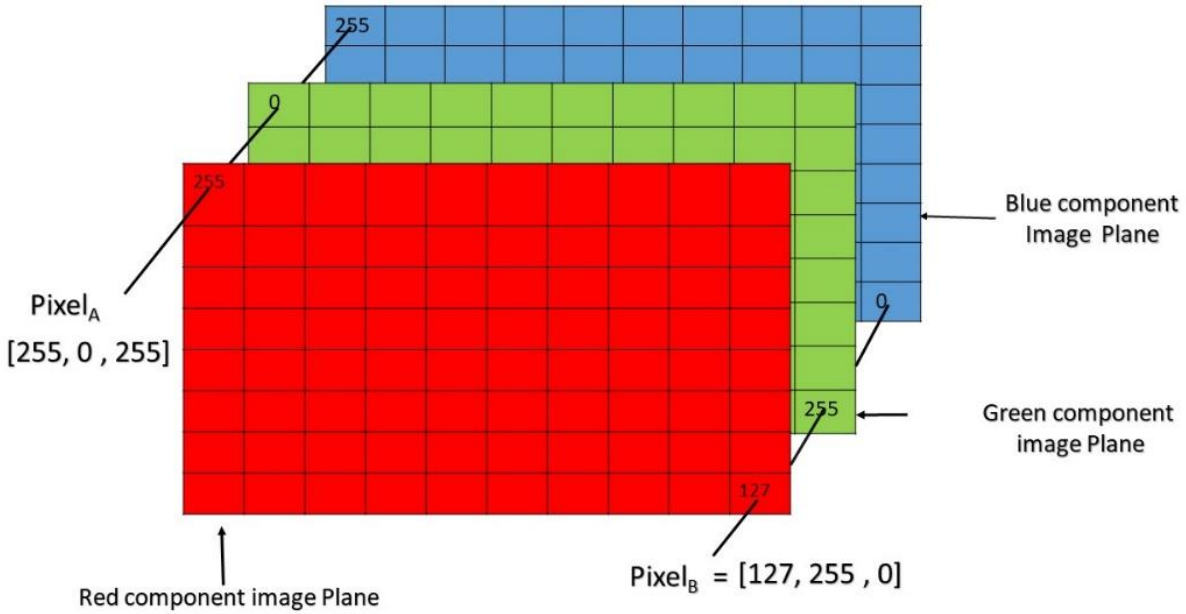
Consist of two-dimensional matrices "layered" to produce a third dimension. Each "layer" is called a *page*.

cat(n,A,B,C, ...)	Creates a new array by concatenating the arrays A,B,C, and so on along the dimension n.

Multidimensional Arrays

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

A =
   1   2
   3   4

>> B=2*A

B =
   2   4
   6   8

>> C=3*A
C =

   3   6
   9  12
```

```
>> D=cat(3,A,B,C)

D(:,:,1) =

   1   2
   3   4
D(:,:,2) =

   2   4
   6   8
D(:,:,3) =

   3   6
   9  12
```



Pixel of an RGB image are formed from the corresponding pixel of the three component images

Matrix-Arithmetic Operations

The Arithmetic Operations includes the following:

Addition (+), subtraction (-), multiplication (*), division (/), power (^).

matrix operations.

Putting a period (.) in front of the arithmetic operator to perform element-by-element.

multiplication (.*), division (./), power (.^).

Operators (Element by Element)	
	.* element-by-element multiplication
	./ element-by-element division
	.^ element-by-element power

Matrix-Arithmetic Operations

Symbol	Operation	Form	Examples
+	Scalar-array addition	A + b	[6,3]+2=[8,5]
-	Scalar-array subtraction	A – b	[8,3]-5=[3,-2]
+	Array addition	A + B	[6,5]+[4,8]=[10,13]
-	Array subtraction	A – B	[6,5]-[4,8]=[2,-3]
.*	Array multiplication	A.*B	[3,5].*[4,8]=[12,40]
./	Array right division	A./B	[2,5]./[4,8]=[2/4,5/8]
.\	Array left division	A.\B	[2,5].\[4,8]=[2\4,5\8]
.^	Array exponentiation	A.^B	[3,5].^2=[3^2,5^2]
			2.^[3,5]=[2^3,2^5]
*	ضرب مصفوفتين عدد أعمدة الأولى يساوي عدد أسطر الثانية		[3,5].^[2,4]=[3^2,5^4]

(لهما أحجام متساوية)

Array Addition and Subtraction

For example:

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} = \begin{bmatrix} 15 & 6 \\ -2 & 17 \end{bmatrix}$$

Array subtraction is performed in a similar way.

```
>>A = [6,-2;10,3];
>>B = [9,8;-12,14]
>>A+B
ans =
   15     6
   -2    17
```

Multiplying a matrix **A** by a scalar w produces a matrix whose elements are the elements of **A** multiplied by w. For example:

$$3\begin{bmatrix} 2 & 9 \\ 5 & -7 \end{bmatrix} = \begin{bmatrix} 6 & 27 \\ 15 & -21 \end{bmatrix}$$

This multiplication is performed in MATLAB as follows:

```
>>A = [2, 9; 5,-7];
>>3*A
ans =
   6    27
  15   -21
```

Vector and matrix

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

>> y=[1 2];
>> x+y

ans =

    2    4
    4    6
>> y=[1 2 3;1 2 3];
>> x+y
Matrix dimensions must agree.
```

الرفع إلى أس

```
>> eye(2,4)

ans =

     1     0     0     0
     0     1     0     0

>> x=3;
>> x^ans
Error using  ^  (line 51)
Incorrect dimensions for raising a matrix to a power. Check
that the matrix is square and
the power is a scalar. To perform elementwise matrix
powers, use '.^'.

>> x.^ans

ans =
```

A.^x يرفع كل عنصر من A إلى الأس x

```
     3     1     1     1
     1     3     1     1
```

```
>> ans.^x

ans =

    27     1     1     1
     1    27     1     1

>> ans^x
Error using  ^  (line 51)
Incorrect dimensions for raising a matrix to a power. Check that the matrix is
square and
the power is a scalar. To perform elementwise matrix powers, use '.^'.
```

x.^A يرفع x إلى الأس الموافق لكل عنصر من A

الرفع إلى أس

A يجب أن تكون مربعة

A^x يرفع المصفوفة A إلى الأس x

```
>> 1+eye(2,2)
ans =

     2     1
     1     2

>> x=3;
>> ans^x

ans =

    14    13
    13    14
```

Examples

Given x and y:

```
>> x=rand(3)

x =

    0.9575    0.9706    0.8003
    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
```

```
>> y=rand(3)

y =

    0.9157    0.6557    0.9340
    0.7922    0.0357    0.6787
    0.9595    0.8491    0.7577
```

Product

```
>> x*y

ans =

    2.4136    1.3421    2.1595
    1.7780    0.7874    1.6584
    0.9335    0.4788    0.7962
```

Addition

```
>> x+y

ans =

    1.8732    1.6263    1.7343
    1.7571    0.9929    0.8206
    1.1171    1.3345    1.1795
```

Subtraction

```
>> x-y

ans =

    0.0418    0.3149   -0.1337
    0.1727    0.9215   -0.5368
   -0.8019   -0.3638   -0.3360
```

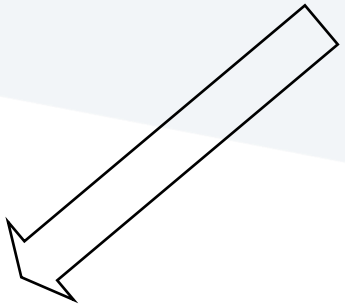
Transpose

```
>> x'

ans =

    0.9575    0.9649    0.1576
    0.9706    0.9572    0.4854
    0.8003    0.1419    0.4218
```

```
>> x'==transpose(x)

ans =

  3×3 logical array

   1   1   1
   1   1   1
   1   1   1
```

Example

```
A = [1 2 3; 5 1 4; 3 4 -1]
   A =
         1     2     3
         5     1     4
         3     4    -1
```



```
x = A(1,:)
x=
     1   2   3
```

```
y = A(3 ,:)
y=
     3   4  -1
```

```
b = x .* y
b=
     3   8  -3
```

```
c = x . / y
c=
  0.33   0.5   -3
```

```
d = x .^2
d=
     1   4   9
```

K= x^2

Erorr:

??? Error using ==> mpower Matrix must be square.

B=x*y

Erorr:

??? Error using ==> mtimes Inner matrix dimensions must agree.

Array or *Element-by-element* multiplication is defined only for arrays having the same size. The definition of the product x.*y, where x and y each have *n* elements, is

$x.*y = [x(1)y(1), x(2)y(2), \ldots, x(n)y(n)]$

if x and y are row vectors. For example, if

$x = [2, 4, -5],\ y = [-7, 3, -8]$

then z = x.*y gives

$$\mathbf{z} = [2(-7), 4(3), -5(-8)] = [-14, 12, 40]$$

$$>> x = [2, 4, -5],\ y = [-7, 3, -8]$$

If x and y are column vectors, the result of x.*y is a column vector. For example z = (x′).*(y′) gives

$$\mathbf{z} = \begin{bmatrix} 2(-7) \\ 4(3) \\ -5(-8) \end{bmatrix} = \begin{bmatrix} -14 \\ 12 \\ 40 \end{bmatrix}$$

Note that x′ is a column vector with size 3 × 1 and thus does not have the same size as y, whose size is 1 × 3.

Thus for the vectors x and y the operations x′.*y and y.*x′ are not defined in MATLAB and will generate an error message.

The array operations are performed between the elements in corresponding locations in the arrays. For example, the array multiplication operation A.*B results in a matrix C that has the same size as A and B and has the elements $c_{ij} = a_{ij} b_{ij}$. For example, if

$$\mathbf{A} = \begin{bmatrix} 11 & 5 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -7 & 8 \\ 6 & 2 \end{bmatrix}$$

then C = A.*B gives this result:

$$\mathbf{C} = \begin{bmatrix} 11(-7) & 5(8) \\ -9(6) & 4(2) \end{bmatrix} = \begin{bmatrix} -77 & 40 \\ -54 & 8 \end{bmatrix}$$

Built-in MATLAB functions and Matrix

The built-in MATLAB functions such as sqrt(x) and exp(x) automatically operate on array arguments to produce an array result the same size as the array argument x.

Thus these functions are said to be *vectorized* functions.

For example, in the following session the result y has the same size as the argument x.

```
>>x = [4, 16, 25];
>>y = sqrt(x)
y =
  2  4  5
```

Built-in MATLAB functions and Matrix

However, when multiplying or dividing these functions, or when raising them to a power, you must use element-by-element operations if the arguments are arrays.

For example, to compute $z = (e^y \sin x) \cos^2 x$, you must type

```
z = exp(y).*sin(x).*(cos(x).^2)
```

```
z = exp(y).*sin(x).*(cos(x)).^2
```

You will get an error message if the size of x is not the same as the size of y. The result z will have the same size as x and y.

Matrix

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

```
>> exp(y)
ans =

    2.7183    2.7183    2.7183
    2.7183    2.7183    2.7183
    2.7183    2.7183    2.7183
```

```
>> sin(y)
ans =
    0.8415    0.8415    0.8415
    0.8415    0.8415    0.8415
    0.8415    0.8415    0.8415
```

```
>> y/10
ans =

    0.1000    0.1000    0.1000
    0.1000    0.1000    0.1000
    0.1000    0.1000    0.1000
```

كافة العمليات التي أجريناها حتى الآن كانت تتم على قيم سلمية
يمكن استخدام المصفوفات لإجراء نفس العمليات على عدد كبير من المعطايت

يمكن إجراء العمليات الرياضية بين القيم السلمية و المصفوفات مباشرة، و تكون بمثابة إجراء العملية بين القيمة السلمية و كل عنصر من عناصر المصفوفة على حدى

Transpose

```
>> d=[2+3i  3+4i  6+5i  0]
d =
  2.0000 + 3.0000i   3.0000 + 4.0000i   6.0000 + 5.0000i   0.0000 + 0.0000i
```

```
>> d'

ans =

  2.0000 - 3.0000i
  3.0000 - 4.0000i
  6.0000 - 5.0000i
  0.0000 + 0.0000i
```

```
>> d.'

ans =

  2.0000 + 3.0000i
  3.0000 + 4.0000i
  6.0000 + 5.0000i
  0.0000 + 0.0000i
```

```
>> transpose(d)

ans =

  2.0000 + 3.0000i
  3.0000 + 4.0000i
  6.0000 + 5.0000i
  0.0000 + 0.0000i
```

```
>> transpose(d)==d.'

ans =

  4×1 logical array

   1
   1
   1
   1
```

Transpose

```
>> d=[2+3i  3+4i  6+5i  0; 2  30+40i  60+50i  0]
d =

   2.0000 + 3.0000i   3.0000 + 4.0000i   6.0000 + 5.0000i   0.0000 + 0.0000i
   2.0000 + 0.0000i  30.0000 +40.0000i  60.0000 +50.0000i   0.0000 + 0.0000i
```

```
>> d'

ans =

   2.0000 - 3.0000i   2.0000 + 0.0000i
   3.0000 - 4.0000i  30.0000 -40.0000i
   6.0000 - 5.0000i  60.0000 -50.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
>> d.'

ans =

   2.0000 + 3.0000i   2.0000 + 0.0000i
   3.0000 + 4.0000i  30.0000 +40.0000i
   6.0000 + 5.0000i  60.0000 +50.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
>> transpose(d)==d.'

ans =

  4×2 logical array

   1   1
   1   1
   1   1
   1   1
```

Transpose

```
>> d=[2   3    0; 6   30    0; 1 1 1]
d =
    2    3    0
    6   30    0
    1    1    1

>> d'
ans =

    2    6    1
    3   30    1
    0    0    1
```

```
>> d.'
ans =
    2    6    1
    3   30    1
    0    0    1

>> transpose(d)
ans =
    2    6    1
    3   30    1
    0    0    1
```

Matrix determine

det(X) is the determinant of the square matrix X.

```
>> d=[2   3    0; 6   30    0; 1 1 1]
d =
    2    3    0
    6   30    0
    1    1    1

>> det(d)
ans =

   42.0000
```

2(30-0)-3(6-0)+0(6-30)=60-18=42

Matrix inverse

inv Matrix inverse.

inv(X) is the inverse of the square matrix X.

A warning message is printed if X is badly scaled or nearly singular.

```
>> inv(d)
ans =

    0.7143   -0.0714         0
   -0.1429    0.0476         0
   -0.5714    0.0238    1.0000
```

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

A =

     1     3     1
     1     2     1
     1     5     0
```

```
>> det(A)

ans =

     1
```

```
>> A^(-1)

ans =

    -5     5     1
     1    -1     0
     3    -2    -1
```

```
>> inv(A)

ans =

    -5     5     1
     1    -1     0
     3    -2    -1
```

Matrix inverse

The inverse of a square matrix A with a non zero determinant is the adjoint matrix divided by the determinant

$$A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A)$$

The adjoint matrix is the transpose of the cofactor matrix.

The cofactor matrix is the matrix of determinants of the minors A_{ij} multiplied by -1^{i+j}.

Matrix inverse

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

Determinant = 9

$$\text{Cofactor matrix} = \begin{bmatrix} + \begin{vmatrix} 1 & 1 \\ 2 & 3 \end{vmatrix} & - \begin{vmatrix} -1 & 1 \\ 1 & 3 \end{vmatrix} & + \begin{vmatrix} -1 & 1 \\ 1 & 2 \end{vmatrix} \\ - \begin{vmatrix} 2 & 0 \\ 2 & 3 \end{vmatrix} & + \begin{vmatrix} 1 & 0 \\ 1 & 3 \end{vmatrix} & - \begin{vmatrix} 1 & 2 \\ 1 & 2 \end{vmatrix} \\ + \begin{vmatrix} 2 & 0 \\ 1 & 1 \end{vmatrix} & - \begin{vmatrix} 1 & 0 \\ -1 & 1 \end{vmatrix} & + \begin{vmatrix} 1 & 2 \\ -1 & 1 \end{vmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 4 & -3 \\ -6 & 3 & -0 \\ 2 & -1 & 3 \end{bmatrix}$$

$$\text{Adjoint matrix} = \text{Transpose of cofactor matrix} = \begin{bmatrix} 1 & -6 & 2 \\ 4 & 3 & -1 \\ -3 & -0 & 3 \end{bmatrix}$$

$$A^{-1} = \text{Inverse of } A = \begin{bmatrix} 1/9 & -6/9 & 2/9 \\ 4/9 & 3/9 & -1/9 \\ -3/9 & 0 & 3/9 \end{bmatrix}$$

$$A\,A^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrix inverse

$$A^{-1} = \frac{1}{|A|}\begin{pmatrix} \begin{vmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{vmatrix} & \begin{vmatrix} a_{1,3} & a_{1,2} \\ a_{3,3} & a_{3,2} \end{vmatrix} & \begin{vmatrix} a_{1,2} & a_{1,3} \\ a_{2,2} & a_{2,3} \end{vmatrix} \\ \begin{vmatrix} a_{2,3} & a_{2,1} \\ a_{3,3} & a_{3,1} \end{vmatrix} & \begin{vmatrix} a_{1,1} & a_{1,3} \\ a_{3,1} & a_{3,3} \end{vmatrix} & \begin{vmatrix} a_{1,3} & a_{1,1} \\ a_{2,3} & a_{2,1} \end{vmatrix} \\ \begin{vmatrix} a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{vmatrix} & \begin{vmatrix} a_{1,2} & a_{1,1} \\ a_{3,2} & a_{3,1} \end{vmatrix} & \begin{vmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{vmatrix} \end{pmatrix}$$

Adjoint Of a Matrix 2 x 2

Let A be the 2 x 2 matrix and is given by:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Then, the adjoint of this matrix is:

$$adj\ A = \begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix}$$ ← منقول مصفوفة المعاملات

Here,

A_{11} = Cofactor of a_{11} = a22

A_{12} = Cofactor of a_{12} =-a21

A_{21} = Cofactor of a_{21} =-a12

A_{22} = Cofactor of a_{22} =a11

Matrix inverse

The inverse of a 2x2 matrix can be written explicitly, namely

inverse of A

$$\begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix}^{-1} = \frac{1}{a11 * a22 - a12 * a21} \begin{bmatrix} a22 & -a12 \\ -a21 & a11 \end{bmatrix}$$

Division

/ Slash or right matrix divide.

A/B is the matrix division of B into A, which is roughly the same as A*inv(B)

\ Backslash or left matrix divide.

- A\B is the matrix division of A into B, which is roughly the same as inv(A)*B
- X = A\B is the solution to the equation A*X = B
- A\eye(size(A)) produces the inverse of A.

Examples

```
>> A=[1 2 0; -1 1 1; 1 2 3]
A =
     1     2     0
    -1     1     1
     1     2     3

>> det(A)

ans =
     9

>> inv(A)
ans =
    0.1111   -0.6667    0.2222
    0.4444    0.3333   -0.1111
   -0.3333         0    0.3333
```

```
>> A\eye(size(A))
ans =

    0.1111   -0.6667    0.2222
    0.4444    0.3333   -0.1111
   -0.3333         0    0.3333

>> eye(size(A))/A
ans =

    0.1111   -0.6667    0.2222
    0.4444    0.3333   -0.1111
   -0.3333         0    0.3333
```

Examples

```
>> B=eye(3)
B =
     1     0     0
     0     1     0
     0     0     1
```

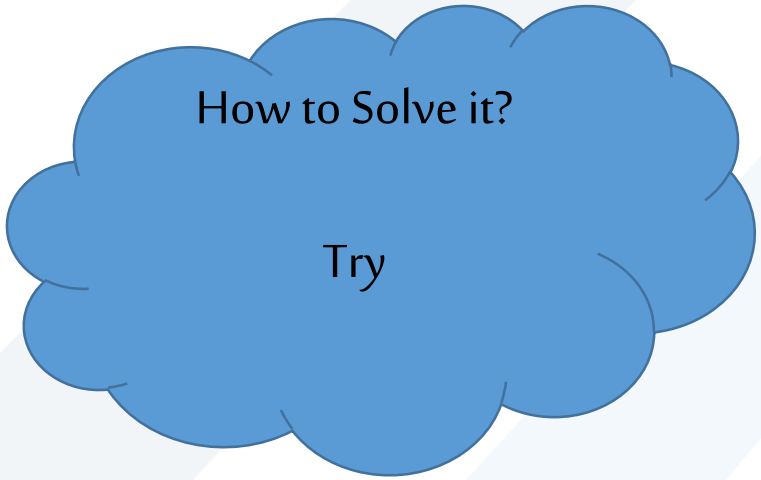
```
>> A/B
ans =
     1     2     0
    -1     1     1
     1     2     3
```

```
>> B/A
ans =
    0.1111   -0.6667    0.2222
    0.4444    0.3333   -0.1111
   -0.3333         0    0.3333
```

```
>> B\A
ans =
     1     2     0
    -1     1     1
     1     2     3
```

مثال

عدل برنامج إيجاد جذور معادلة من الدرجة الثانية ليستطيع إيجاد جذور عدة معادلات دفعة واحدة
أي أن a,b,c أشعة وكذلك x1,x2 حلول المعادلات أشعة



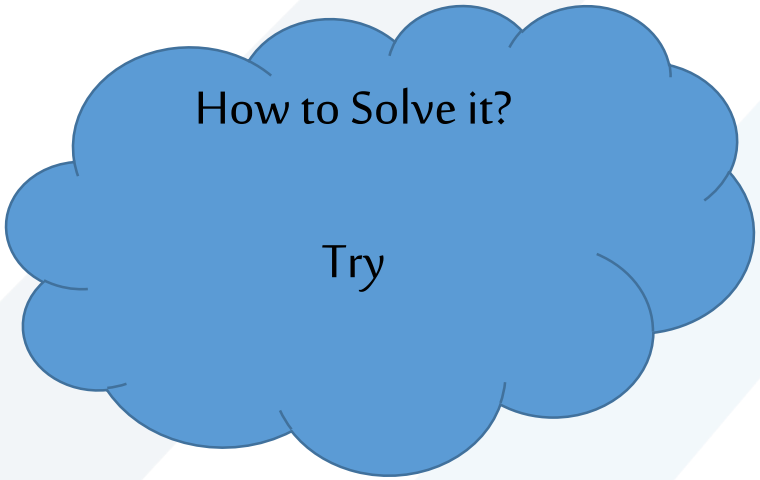
Untitled.m

```
a=input('input a as a 1*n vector:');
b=input('input b as a 1*n vector:');
c=input('input c as a 1*n vector:');

delta=b.^2 -4*a.*c;

x1=(-b+sqrt(delta))./(2*a);
x2=(-b-sqrt(delta))./(2*a);
```

```
>> Untitled
input a as a 1*n vector:[ 1 2 3]
input b as a 1*n vector:[4 5 6]
input c as a 1*n vector:[1 2 3]

>> delta
delta =
   12    9    0

>> x1,x2
x1 =
  -0.2679   -0.5000   -1.0000
x2 =
  -3.7321   -2.0000   -1.0000
```

مثال

اكتب برنامج لحل n معادلة ب n مجهول



اكتب برنامج لحل n معادلة ب n مجهول

$$[A]_{n*n} * [X]_{n*1} = [B]_{n*1}$$

$$[A]^{-1}_{n*n} * [A]_{n*n} * [X]_{n*1} = [A]^{-1}_{n*n} * [B]_{n*1}$$

$$[I]_{n*n} * [X]_{n*1} = [A]^{-1}_{n*n} * [B]_{n*1}$$

$$[X]_{n*1} = [A]^{-1}_{n*n} * [B]_{n*1}$$

اكتب برنامج لحل n معادلة ب n مجهول

Untitled2.m

```
A=input('input a as a n*n matrix:');
B=input('input b as a n*1 vector:');

x=A\B;
disp('result is')
disp(x)
```

```
>> Untitled2
input a as a n*n matrix:[1 2 3;1 2 1;4 5 6]
input b as a n*1 vector:[6; 6 ;6]
result is
   -6
    6
    0
```

جَامعة
المَنارة
MANARA UNIVERSITY

Thanks .