MATLAB: Operations

In this tutorial, the reader will learn about how to different matrix and polynomial operations.

Addition and subtraction

Create the following matrices in MATLAB:

$A = \begin{bmatrix} 1.2 & 10 & 15 \\ 3 & 5.5 & 2 \\ 4 & 6.8 & 7 \end{bmatrix},$	$B = \begin{bmatrix} 5 & 1 & 3 \\ 9 & 0.8 & 8 \\ 2 & 4 & 6 \end{bmatrix}$
>> A=[1.2 10 15;3 5.5 2; 4 6.8 7]	>> B=[5 1 3;9 0.8 8; 2 4 6]
A =	в =
1.2000 10.0000 15.0000	5.0000 1.0000 3.0000
3.0000 5.5000 2.0000 4.0000 6.8000 7.0000	9.0000 0.8000 8.0000 2.0000 4.0000 6.0000

To create a *C* matrix that is the sum of *A* and *B*

>> C=A+B		
с =		
6.2000	11.0000	18.0000
12.0000	6.3000	10.0000
6.0000	10.8000	13.0000

To create a D matrix that is the subtracts of B from A

>> D=A-B		
D =		
-3.8000 -6.0000 2.0000	9.0000 4.7000 2.8000	12.0000 -6.0000 1.0000

To create G matrix by adding 2 to A matrix. Since you adding a scalar to matrix, MATLAB adds 2 to each element in A, such as

>> G=A+2		
G =		
3.2000	12.0000	17.0000
5.0000	7.5000	4.0000
6.0000	8.8000	9.0000

Matrix multiplication

The inner dimensions of two matrices must agree to perform matrix multiplication and the dimension of the resulting matrix is the two outer dimensions, such as:

$$A_{nxm} * B_{mxr} = C_{nxr}$$

Enter the following matrices in MATLAB:

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad y = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 0 \\ 1 & 2 & 5 \end{bmatrix}$$

Multiple *x***y*

```
>> x*y
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

MATLAB will prompt you with dimension error. Now, multiply x with the transpose of y. This will yield a 3x3 matrix because x_{3x1} and y'_{1x3}

>> x*y'		
ans =		
4	5	6
8	10	12
12	15	18

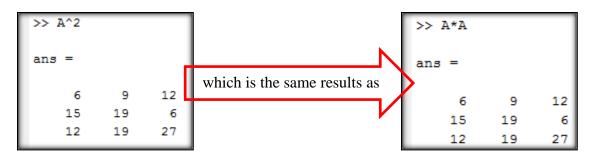
Note if you multiply transpose of x by y, the operation will yield only one number because x'_{1x3} and y_{3x1}

>> x'*y	
ans =	
32	

Note a scalar can either multiply a matrix or be multiplied by matrix and the results would be the same, such as:

>> 5*A			>> A*5		
ans =			ans =		
5	5	10	5	5	10
15	20	0	15	20	0
5	10	25	5	10	25

Note you can raise a square matrix to any power, i.e. A² because MATLAB would perform A*A operation which does not violate the dimension rules. The same is not true for non-square matrices.



Now try to calculate x^2 (remember x was not a square matrix, it is a column vector)

```
>> x^2
??? Error using ==> mpower
Inputs must be a scalar and a square matrix.
```

However, you can square each element in x, y or A by using element-wise operator the period (.), MATLAB calculated the square if each element such as

```
>> x.^2
ans =
1
4
9
```

Think about the element-wise operator as if you writing a *for-loop* to perform a certain mathematical operation on each element on a matrix.

<u>Array Division</u>

This tutorial won't cover matrix division because that will require some introductory remarks about the solution of system of linear equations (A*x=B). In what follows, the discussion will cover array division instead. Consider the expressions x./y, x.\y, A./B and A.\B. First enter the following matrices.

$$x = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \quad y = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}, \\ A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 9 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & 5 & 6 \\ 7 & 6 & 5 \end{bmatrix}$$

>> x.\y			>> x./y		
ans =			ans =		
4.0000	2.5000	2.0000	0.2500	0.4000	0.5000

You can think about it as if it is an element by element operation, where the first operation (x.y) is dividing y by x element-by-element as if you are calculating y./x, such as



<u>Matrix inversion</u>

Use *inv*(*A*) function to calculate the inverse of a square non-singular ($|A|\neq 0$) matrix X. You can check whether or not the determinate is zero by using *det*() function.

```
>> A=[1 3 4;5 6 7;6 7 7]
A =
     1
          3
                4
    5
          6
                7
     6
          7
                7
>> det(A)
ans =
  10.0000
>> inv(A)
ans =
   -0.7000
            0.7000
                      -0.3000
   0.7000
            -1.7000
                      1.3000
   -0.1000
             1.1000
                      -0.9000
```

Characteristic equation, eigenvalues and eigenvectors

Given a matrix A, one can calcuate the characheristic equation using poly(*matrix*) function, then can calculate the roots of this equation that is the eigenvalues of the matrix using roots(charac_eqn), such as:

```
>> p=poly(A)
p =
    1.0000 -14.0000 -33.0000 -10.0000
```

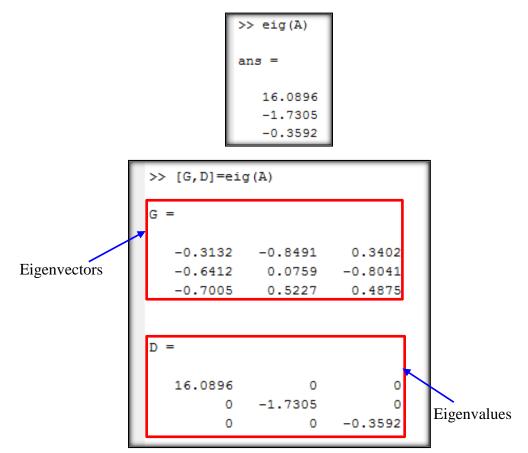
The way to interpret the output of the poly(matrix) function is rewrite as a nth order polynomial, where the order of polynomial is number of terms -1(i.e. 4-1 = 3 in the example above). So the above characteristic equation can be rewrites as:

$$s^3 - 14s^2 - 33s - 10 = 0$$

Next, calculate the roots of the characteristic equation, such

```
>> roots(p)
ans =
    16.0896
    -1.7305
    -0.3592
```

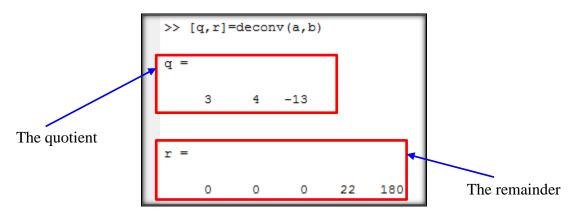
To calculate the eigenvalues and eigenvectors of the a matrix, use the eig(matrix) function.



Polynomial Operations

Convolution (product) of polynomial, the product of two polynomials is the convolution of the coefficients. Use *conv(a,b)* function to calculate the product

Deconvolution (division) of polynomials can be done using deconv(a,b) function, where to calculate the quotient and remainder.



This output mean

 $3s^4 + 10s^3 + 25s^2 + 36s + 50 = (s^2 + 2s + 10)(3s^2 + 4s - 13) + 22s + 180$

Polynomial evaluation, to evaluate a polynomial at given point use *polyval(polynomial, value)*, such

```
>> p=[2 1 3];
>> polyval(p,3)
ans =
24
```

Partial fraction expansion, dividing two polynomials can be represented as sum of fractions and direct term. To calculate the partial fraction of a transfer function, use *residue(numerator,denominator)* function. For example, find the partial fraction expansion of the following transfer function:

$$G(s) = \frac{2s^3 + 2s^2 + 3s + 6}{s^3 + 6s^2 + 11s + 6}$$
$$G(s) = \frac{-6}{s+3} + \frac{-4}{s+2} + \frac{3}{s+1} + 2$$

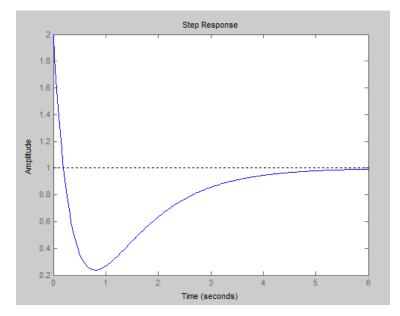
Note that we used *printsys(num,den,'s')* function to reconstruct the transfer function. Now, let's apply *residue()* function to calculate the partial fraction expression, such as:

```
>> [r,p,k]=residue(num,den)
r =
        -6.0000
        -4.0000
        3.0000
p =
        -3.0000
        -2.0000
        -1.0000
k =
        2
```

Useful functions for ME384students:

Syntax	Description
[z,p,k]=tf2zp(num,den)	Calculate the zeros, poles and gain from transfer function
[num,den]=zp2tf(z,p,k)	Calculate the transfer function from zeros, poles and gain
[A,B,C.D]=tf2ss(num,den)	Calculate the state-space representation from transfer function
[num,den]=ss2tf(A,B,C,D)	Calculate the transfer function from the state-space representation
[A,B,C,D]=zp2ss(z,p,k)	Calculate the state-space representation from zeros, poles and gain
[z,p,k]=ss2zp(A,B,C,D)	Calculate the zeros, poles and gain from state-space representation
sys=tf(num,den)	Create the transfer function from numerator and denominator
step(sys)	Generate the step response of dynamic system
sys=series(sys1,sys2)	Series equivalent of two transfer functions
sys=parelle(sys1,sys2)	Parallel equivalent of two transfer functions
Sys=feedback(sysg,sysh)	Equivalent transfer function of entire feedback system

```
>> num=[2 5 3 6];
>> den=[1 6 11 6];
>> printsys(num,den,'s')
num/den =
   2 s^3 + 5 s^2 + 3 s + 6
   s^3 + 6 s^2 + 11 s + 6
>> [A, B, C, D]=tf2ss(num, den)
A =
       -11 -6
    -6
       0 0
    1
    0
          1
               0
в =
    1
    0
    0
с =
    -7
        -19
               -6
D =
    2
>> sys=tf(num,den)
Transfer function:
2 s^3 + 5 s^2 + 3 s + 6
s^3
    + 6 s^2 + 11 s +
                        6
   step(sys)
```



>>