Theme: Introduction in Matlab.

Theoretical notions

Introduction

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

Typical uses of MATLAB include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical-computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN.

Perhaps the easiest way to visualize MATLAB is to think of it as a full-featured calculator. Like a basic calculator, it does simple math such as addition, subtraction, multiplication, and division. Like a scientific calculator, it handle complex numbers, square roots and powers, logarithms, and trigonometric operations such as sine, cosine, and tangent. Like a programmable calculator, it can be used to store and retrieve data; you can create, execute, and save sequences of commands to automate the computation of important equations; you can make logical comparisons and control the order in which commands are executed.

Polynomials

In Matlab, a polynomial is represented by a vector. To create a polynomial in Matlab, simply enter each coefficient of the polynomial into the vector in descending order. For instance, let's say you have the following polynomial:

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

To enter this into Matlab, just enter it as a vector in the following manner

```
»x = [1 3 -15 -2 9]

x =
     1   3  -15  -2   9
```

Matlab can interpret a vector of length n+1 as an nth order polynomial. Thus, if your polynomial is missing any coefficients, you must enter zeros in the appropriate place in the vector. For example,

$$s^4 + 1$$

would be represented in Matlab as:

```
»y = [1 0 0 0 1]
```

You can find the value of a polynomial using the `polyval` function. For example, to find the value of the above polynomial at s=2,

```
»z = polyval(y,2)

     z =
          17
```

You can also extract the roots of a polynomial. This is useful when you have a high-order polynomial such as

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

Finding the roots would be as easy as entering the following command;

```
»roots([1 3 -15 -2 9])  %or »roots(x)

     ans =
        -5.5745
         2.5836
        -0.7951
         0.7860
```

Let's say you want to multiply two polynomials together. The product of two polynomials is found by taking the convolution of their coefficients. Matlab's function `conv` that will do this for you.

```
»x = [1 2];
»y = [1 4 8];
»z = conv(x,y)

     z =
          1   6   16   16
```

Dividing two polynomials is just as easy. The `deconv` function will return the remainder as well as the result. Let's divide z by y and see if we get x.

```
»[xx, R] = deconv(z,y)

     xx =
          1   2

     R =
          0   0   0   0
```

As you can see, this is just the polynomial/vector x from before. If y had not gone into z evenly, the remainder vector would have been something other than zero.

If you want to add two polynomials together which have the same order, a simple z=x+y will work (the vectors x and y must have the same length). In the general case, the user-defined function, *polyadd* can be used. To use *polyadd*, copy the function into an m-file, and then use it just as you would any other function in the Matlab toolbox. Assuming you had the polyadd function stored as a m-file, and you wanted to add the two uneven polynomials, x and y, you could accomplish this by entering the command:

```
»z = polyadd(x,y)

  x =
     1  2

  y =
     1  4  8

  z =
     1  5  10
```

Exercise: Given the polynomials: x = $s^4 - 2s - 3$ and y = $s^3 - 7s - 6$

a) Find the roots for both of them
b) Multiply the two polynomials then find the roots for the new polynomial z.
c) Divide y by x and get the remainder

Plotting

The most common plot used for engineers and scientists is the *xy* plot. Occasionally, we may like or asked to use a logarithmic scale on one or both of the axes.

The Matlab command for generating linear and logarithmic plot of the vectors x and y are the following:

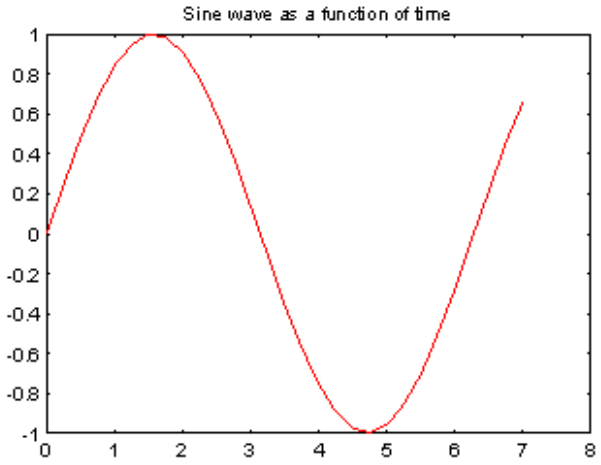
Plot(x,y)	*generate a linear plot of the values of x and y.*
Semilogx(x,y)	*generate a plot of the values of x and y using a logarithmic scale for x and a linear scale for y.*
Semilogy(x,y)	*generate a plot of the values of x and y using a logarithmic scale for y and a linear scale for x.*
Loglog(x,y)	*generate a plot of the values of x and y using logarithmic scales for both x and y.*

It is also easy to create plots in Matlab. Suppose you wanted to plot a sine wave as a function of time. First make a time vector (the semicolon after each statement tells Matlab we don't want to see all the values) and then compute the sin value at each time.

```
create an M-file and put the data inside

    x=0:0.25:7;
    y = sin(x);
    plot(x,y)
    xlabel('Time axis')
```

```
ylabel('Amplitude')
title('Sine wave as a function of time')
grid
```



We can also do a discrete plot of the sine wave,

```
stem(x,y)
```

We can also plot with colors or plot with discrete dots. For further help in plot, you can always refer to the help on plot

```
plot(x,y,'ro')
```

Moreover, we can hold the graph and add another plot on the same plot using the 'hold' command,

```
v=0:0.25:7;
w=cos(v);
hold on
plot(v,w)
hold off
```

A simple way to plot multiple curves on the same graph is to use multiple arguments in a plot command, as in

```
plot(x,y,v,w)
%add grid from workspace
%plot(x,y,v,w,'o')
```

Basic plotting is very easy in Matlab, and the `plot` command has extensive add-on capabilities. Now we will generate a polar plot of the angle thetas (in radians) between o and 2π.
Hint: do help on polar and see how it works

```
thetta = 0 : 2*pi/100 : 2*pi;
r = thetta/(2*pi);
polar(thetta,r,'b')
hold
```

```
polar(thetta,r,'.'), title ('Polar Plot')
hold
```

The subplot command allows you to split the graph window into sub-windows. H = SUBPLOT (m,n,p), or SUBPLOT (mnp), breaks the Figure window into an m-by-n matrix of small axes.

```
subplot(2,1,1) , plot(x,y)
 subplot(2,1,2) , plot(thetta,r)
```

Control

Linear and control system analysis and design begin with the model of real systems. Theses models, which are mathematical representations of such things as chemical processes, machinery, and electrical circuits, are used to study the dynamic response of real systems. The mathematical techniques used by Matlab to design and analyze these systems assume processes that are physically realizable, linear and time invariant (LTI).

Matlab uses models in the form of ***transfer functions*** or ***state space equations***, thus allowing both *classical* and *modern* control system design and analysis techniques to be used.

1. ***Transfer functions*** can be expressed as a polynomial, a ratio of polynomials, or one of two factored forms: zero-pole-gain or partial fraction form.

2. ***State space system*** models are particularly well-suited to Matlab because they are a matrix-based expression.

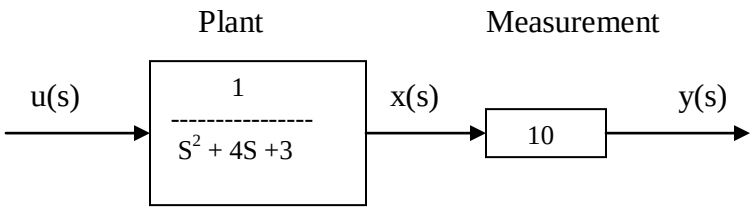
Either model form can be expressed in **continuous-time (analog)**, or **discrete-time (digital)** forms.

Transfer functions models

The transfer function is expressed as a ratio of two polynomials, where the numerator polynomial is simply a scalar. For systems having a single input and a single output (SISO), the form for writing transfer function is

$$H(s) = \frac{B(s)}{A(s)} = \frac{b_0 s^n + b_1 s^{n-1} + b_2 s^{n-2} + ... + b_n}{a_0 s^n + a_1 s^{n-1} + a_2 s^{n-2} + ... + a_n}$$

Block diagrams are frequently used to show how the transfer functions and the input and output variables of a system are related.



More generally, the numerator of this transfer function can be a three-dimensional matrix for multiple-input-multiple-output (MIMO) systems.

Very often, the numerator and denominator of a transfer function are factored into the zero-pole gain form, which is

$$H(s) = K\frac{(s-z_1)(s-z_2)...(s-z_n)}{(s-p_1)(s-p_2)...(s-p_n)}$$

Moreover transfer function can also be written in the partial-fraction expansion

$$H(s) = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + ... + \frac{r_n}{s-p_n} + k(s)$$

The residue function performs a partial fraction expansion.

```
[r,p,k] = residue(B,A)
```

where

r vector contains the coefficients r_i

p vector contains the coefficients p_i

k vector contains the values of k_n

Let us consider the following transfer function:

$$H(s) = \frac{10}{s^2 + 4s + 3}$$

```
B = [10]
A=[1 4 3]
[r,p,k]= residue(B,A)
```

Now let us change the B to [1 0 0] and see what happens.

State Space Models

Using the matrix notation, any continuous time system model can be written as the following state space model

$$x' = Ax + Bu$$
$$y = Cx + Du$$

Where A, B, C and D are the state space representation of the system.

Example: `A = [0 1; -3 -4], B=[0 1]', C=[10 0]` and `D=0`

The state space equations for discrete time systems are also very similar to the continuous systems:

$$x[n+1] = Ax[n] + Bx[n]$$
$$y[n] = Cx[n] + Du[n]$$

Where n represents the current sample and n+1 indicates the next sample.

The commands that are used for creation the LTI model are presented in the following table

Table 1. The commands for creation the LTI models.

Command	Description
ss	Create state-space model .
tf	Create a transfer function.
tfdata	Retrieve transfer function data.
zpk	Create a zero-pole-gain model.
zpkdata	Retrieve zero-pole-gain data.

Exemple: Let is given the following transfer function $H(s) = \frac{s+2}{s^2+2s+5}$. This function can be presented in Matlab using the following commands:

num=[1 2]
den=[1 2 5]
sis=tf(num,den)

The result:
transfer function:

s + 2
----
s^2 + 2s +5

Example: Let it be given the following values $k = 20;\ z_1 = -1;\ p_1 = -4;\ p_2 = -5$. For creation the a zero-pole-gain model is using the *zpk* function in the following form
» z=[-1]; p=[-4 –5]
» sys=zpk(z,p,20)
» step(sys)
Result :
Zero/pole/gain:
$$\frac{20(s+1)}{(s+4)(s+5)}$$
Example: Let is given state space model described by the following matrixes
$A = \begin{bmatrix} 0 & 1 \\ -5 & -2 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$, $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$.
In Matlab the command ss is using in the following form:
» sys = ss([0 1; -5 -2], [0 3], [0 1], 0)

Model Conversion

Matlab has a number of functions that make it easy to covert from one model form to another and to convert continuous-time systems into discrete-time systems. We can do some help on control toolbox and take a look on the functions.

Model Conversion Functions

Function	Purpose
c2d	continuous state space to discrete state space
d2c	discrete state space to continuous state space
residue	partial fraction expansion
ss2tf	state space to transfer function
ss2zp	state space to zero pole gain
tf2ss	transfer function to state space
tf2zp	transfer function to zero pole gain
zp2ss	zero pole gain to state space
zp2tf	zero pole gain to transfer function

Example:
»sisc=tf ([1 -1],[1 2 5]) % is created the continue transfer function
»step (sisc) % is presented the step response
»sisd=c2d (sisc,0.01) % the continue model is converted to discrete model
»hold on % the creation of two graphics on the same plot.
»step (sisd) % creation the step response
Ex:
»sisd=tf ([1 -1],[1 2 5], 0.1) % is created discret transfer function with sample time 0.1
»step (sisd)
»sisc=d2c (sisd) % conversation the discrete model to continue model
»hold on
»step (sisc)

Generating a Step Response in MATLAB

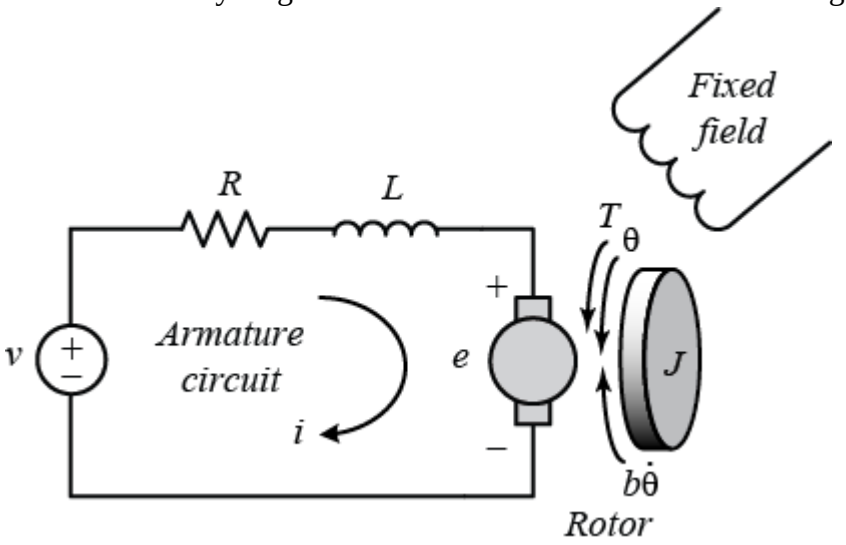
The `step` function is one of most useful functions in MATLAB for control design. Given a system representation, the response to a step input can be immediately plotted, without need to actually solve for the time response analytically. A step input can be described as a change in the input from zero to a finite value at time $t = 0$. By default, the `step` command performs a unit step (i.e. the input goes from zero to one at time $t = 0$).

The basic syntax for calling the step function is the following, where `sys` is a defined LTI object.

```
step(sys)
```

DC motor modeling

A common actuator in control systems is the DC motor. It directly provides rotary motion and, coupled with wheels or drums and cables, can provide translational motion. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure.



For this example, we will assume that the input of the system is the voltage source (*V*) applied to the motor's armature, while the output is the rotational speed of the shaft d(*theta*)/dt. The rotor and shaft are assumed to be rigid. We further assume a viscous friction model, that is, the friction torque is proportional to shaft angular velocity.

The physical parameters for our example are:

```
(J)     moment of inertia of the rotor
(b)     motor viscous friction constant
(Ke)    electromotive force constant
(Kt)    motor torque constant
(R)     electric resistance
(L)     electric inductance
```

In general, the torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field. In this example we will assume that the magnetic field is constant and, therefore, that the motor torque is proportional to only the armature current *i* by a constant factor *Kt* as shown in the equation below. This is referred to as an armature-controlled motor.

$$M_e = k_t \cdot i(t)$$

The back emf, *e*, is proportional to the angular velocity of the shaft by a constant factor *Ke*.

$$U_{em} = k_e \dot{\theta}$$

From the figure above, we can derive the following governing equations based on Newton's 2nd law and Kirchhoff's voltage law.

$$J\ddot{\theta} + b\dot{\theta} = k_t i;$$

$$L\frac{di}{dt} + Ri = U_a - k_e \dot{\theta}.$$

Transfer Function

Applying the Laplace transform, the above modeling equations can be expressed in terms of the Laplace variable *s*.

$$s(Js+b)\theta(s) = k_t i(s);$$
$$(Ls+R)i(s) = U_a - k_e\theta(s).$$

We arrive at the following open-loop transfer function by eliminating *I*(*s*) between the two above equations, where the rotational speed is considered the output and the armature voltage is considered the input.

$$\frac{\dot{\theta}(s)}{U_a(s)} = \frac{k_t}{(Ls+R)(Js+b)+k_e k_t}.$$

Task:

1. For the given variants :
   a) Find the roots of the polynomial.
   b) Multiply the polynomial with f=x+1.
   c) Divide the obtained polynomial by f=x+1.
   d) Present the plot of the polynomial.

Variants

1. 24*x^4+50*x^3+35*x^2+10*x+1=0;
2. 30*x^4+61*x^3+41*x^2+11*x+1=0;
3. 36*x^4+72*x^3+47*x^2+12*x+1=0;
4. 42*x^4+83*x^3+53*x^2+13*x+1=0;
5. 48*x^4+94*x^3+59*x^2+14*x+1=0;
6. 60*x^4+116*x^3+71*x^2+16*x+1=0;
7. 66*x^4+127*x^3+77*x^2+17*x+1=0;
8. 72*x^4+138*x^3+83*x^2+18*x+1=0;
9. 78*x^4+149*x^3+89*x^2+19*x+1=0;
10. 84*x^4+160*x^3+95*x^2+20*x+1=0;
11. 90*x^4+171*x^3+101*x^2+21*x+1=0;
12. 96*x^4+182*x^3+107*x^2+22*x+1=0;
13. 102*x^4+193*x^3+113*x^2+23*x+1=0;
14. 108*x^4+204*x^3+119*x^2+24*x+1=0;
15. 114*x^4+215*x^3+125*x^2+25*x+1=0;
16. 120*x^4+226*x^3+131*x^2+26*x+1=0;
17. 32*x^4+64*x^3+42*x^2+11*x+1=0;
18. 40*x^4+78*x^3+49*x^2+12*x+1=0;
19. 48*x^4+92*x^3+56*x^2+13*x+1=0;
20. 56*x^4+106*x^3+63*x^2+14*x+1=0;
21. 64*x^4+120*x^3+70*x^2+15*x+1=0;
22. 72*x^4+134*x^3+77*x^2+16*x+1=0;
23. 80*x^4+148*x^3+84*x^2+17*x+1=0;

2. For the given variants :
   a) Present for each transfer function the step response.
   b) Convert the continue model to discrete model and present in the one plot the step response for continues and discrete model.
   c) Convert transfer function to state space.
   d) Convert transfer function to zero pole gain.

   Variants

1. $\frac{1}{4s^2+3s+1};\frac{3s+2}{4s^2+3s+1};\frac{1}{10s+1}.$
2. $\frac{6s+2}{8s^2+2s+1};\frac{1}{15s^2+3s};\frac{7s^2+8s+1}{4s^2+3s+1}.$
3. $\frac{1}{10s+1};\frac{1}{12s^2+9s+1};\frac{s^2+9s}{4s^2+3s+1}.$
4. $\frac{4s}{8s^2+4s};\frac{4}{25s^2+3s+1};\frac{1}{3s+1}.$
5. $\frac{8s^2+4s}{14s^2+13s+1};\frac{4s+1}{6s+1};\frac{4}{s^2+3s+1}.$
6. $\frac{8s^2+s}{10s^2+3s+1};\frac{4s}{3s+1};\frac{1}{8s^2+5s+1}.$
7. $\frac{4}{2s^2+2s+1};\frac{8s^2+4s}{9s^2+2s+1};\frac{1}{2s+1}.$
8. $\frac{10}{3s+1};\frac{8s^2+4s}{6s^2+7s+1};\frac{1}{5s^2+2s+1}.$
9. $\frac{1}{3s^2+s+1};\frac{s^2+4s+1}{4s^2+3s+1};\frac{1}{8s^2+4s}.$
10. $\frac{2s^2+2s}{8s^2+3s};\frac{1}{5s^2+7s+1};\frac{2s^2+8s+1}{4s^2+3s+1}.$
11. $\frac{1}{2s^2+6s+1};\frac{9s^2+3s}{4s^2+3s+1};\frac{1}{7s+1}.$
12. $\frac{3s^2+2s+1}{7s^2+3s+1};\frac{1}{9s^2+3s+1};\frac{3s+1}{5s+1}.$
13. $\frac{3s^2+2s}{5s^2+3s+1};\frac{2s}{8s^2+3s};\frac{2}{4s^2+3s+1}.$
14. $\frac{1}{6s^2+8s+1};\frac{s^2+4s+1}{2s^2+4s+1};\frac{1}{7s^2+8s+1}.$
15. $\frac{s^2+4s}{3s^2+2s+1};\frac{1}{7s^2+8s+1};\frac{2s}{2s^2+3s+1}.$
16. $\frac{1}{3s^2+8s+1};\frac{3s+1}{4s^2+7s+1};\frac{2}{8s+1}.$
17. $\frac{s+2}{4s^2+9s+1};\frac{1}{5s^2+3s};\frac{3s^2+4s}{4s^2+3s+1}.$
18. $\frac{1}{10s+1};\frac{1}{14s^2+7s+1};\frac{s^2+4s}{4s^2+3s+1}.$
19. $\frac{4s+1}{9s^2+4s};\frac{4}{20s^2+3s+1};\frac{s}{3s+1}.$
20. $\frac{5s^2+2s}{14s^2+7s+1};\frac{10s+1}{6s+1};\frac{4}{8s^2+3s+1}.$

21. $\frac{1}{19s^2+6s+1}; \frac{9s^2+3s}{4s^2+3s+1}; \frac{s}{8s+1}.$

22. $\frac{3s^2+2}{18s^2+3s+1}; \frac{1}{15s^2+3s+1}; \frac{3s+1}{5s+1}.$

23. $\frac{3s}{5s^2+3s+1}; \frac{2s}{5s^2+3s+1}; \frac{2}{10s^2+3s+1}.$

3. For the given variant simulate the model of the DC motor

Variants	J, kgm^2/s^2	b, Nms	k_t, Nm/A	k_e, Nm/A	R, Ω	L
1	0.001	0.12	0.011	0.011	1.1	0.1
2	0.002	0.13	0.012	0.012	1.2	0.2
3	0.003	0.14	0.013	0.013	1.3	0.3
4	0.0035	0.15	0.014	0.014	1.4	0.4
5	0.0025	0.16	0.015	0.015	1.5	0.5
6	0.0015	0.17	0.016	0.016	1.6	0.6
7	0.0022	0.18	0.017	0.017	1.7	0.7
8	0.0011	0.19	0.018	0.018	1.8	0.8
9	0.0014	0.2	0.019	0.019	1.9	0.9
10	0.0017	0.21	0.011	0.011	2.0	1.0
11	0.0018	0.22	0.011	0.011	1.1	0.1
12	0.0019	0.23	0.012	0.012	1.2	0.2
13	0.002	0.24	0.013	0.013	1.3	0.3
14	0.0021	0.25	0.014	0.014	1.4	0.4
15	0.0022	0.26	0.015	0.015	1.5	0.5
16	0.0023	0.27	0.016	0.016	1.6	0.6
17	0.0024	0.28	0.017	0.017	1.7	0.7
18	0.0025	0.29	0.018	0.018	1.8	0.8
19	0.0026	0.3	0.019	0.019	1.9	0.9
20	0.0027	0.31	0.011	0.011	2.0	1.0
21	0.0028	0.32	0.011	0.011	1.1	0.1
22	0.0029	0.33	0.012	0.012	1.2	0.2
23	0.003	0.35	0.013	0.013	1.3	0.3