ELECTRONICS AND COMMUNICATIONS DEPT.
FACULTY OF ENGINEERING
CAIRO UNIVERSITY
GIZA, 12613, EGYPT

قسم الإلكترونيات و الاتصالات الكهربية
كلية الهندسة
جامعة القاهرة
الجيزة –جمهورية مصر العربية

Third Year
Control Subject

A brief overview of:

MATLAB Control Toolbox

Table of Contents

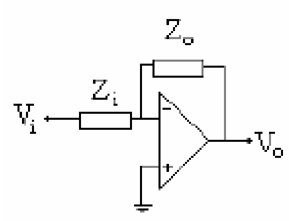
1. System Identification

1.1. Transfer function

The transfer function is the direct relationship between system output and its input regardless of the internal components of the system. Every system that has an input and output may be considered a system that has a transfer function. For example, the electronic buffer is a system that has an input voltage *V_i* and output voltage *V_o* such that the transfer function of the system is: $\frac{V_o}{V_i} = 1$

The operational amplifier in the shown configuration represents a system with a transfer function:

$$\frac{V_o}{V_i} = -\frac{Z_o}{Z_i}$$



Note that *Z_o* or *Z_i* may be a capacitor, coil, resistor, or combination of them, i.e. it will be a function of S (S-domain).

So, electric and electronic components and basic circuits may be considered as systems and can have a transfer function.

In our course, we concern with control systems that have their transfer function in the S-domain using Laplace transform. In order to deal with MATLAB to analyze any system and control it we should first define this system to MATLAB by any method of system identification methods (will be mentioned one by one), in this part of the lab. we will focus on defining the system using its transfer function (it will be your task to get the transfer function in the S-domain using lectures and tutorials.) then use the given transfer function to identify the system to MATLAB.

So, let’s see an example:

Assume that the transfer function of a system is given as follows and you are required to define this system to MATLAB:

$$\frac{\theta_o}{\theta_i} = \frac{1}{S^2 + 5S + 6}$$

To define any system, we should give it a name and then specify its T.F. numerator and denominator coefficients.

First, let the system name be ***sys***, then let's identify the denominator coefficients:

Take a look at the polynomial in the denominator of the T.F. and define it as we learned in the ***Introduction to MATLAB*** chapter.

Let's name the denominator polynomial ***denom*** and the numerator polynomial ***num*** then define these polynomials to MATLAB as follows:

```
>> num=[1];
>>denom=[1 5 6];
```

Then define the system ***sys*** through its components ***num*** & ***denom*** as follows:

```
>>sys=tf(num,denom)
```

Which means that the instruction **tf** is the one that is used for defining systems through their transfer function (note that it's an abbreviation of **T**ransfer **F**unction). The format of this instruction is as follows:

Sys_name **= tf (***T.F._numerator_polynomial* **,** *T.F._denominator_polynomial***)**

Another example:

Define the following system to MATLAB using its shown transfer function:

$$\frac{Y(s)}{U(s)} = \frac{2S + 3}{S(S^2 + 2) + (3S^2 + 5)}$$

The used code will be:

```
>> a=[2 3];
>>b=[1 3 2 5];
>>system=tf(a,b)
```

You can get the same results using the following code in which we define **s** as a symbol in the transfer function of **system**:

```
>> s=tf('s');
>> system= (2*s+3)/(s^3+3*s^2+2*s+5)
```

1.2. System zeros, poles, and gain

Another method of defining systems is through knowing their zeros (roots of the numerator of the transfer function of this system), poles (roots of the denominator of the transfer function of this system), and gain (over all constant gain of the transfer function).

So, let's have an example, to learn how to use this method.

Example:

A system is defined by the shown transfer function:

$$\frac{Y(s)}{R(s)} = \frac{3(S+2)}{S(S+1)(S+3)}$$

And you are required to define this system to MATLAB.

Note that the system zeros are got by: (S+2)=0 $\rightarrow$ S=-2 , also system poles can be got using the same manner to be: S = 0, -1, -3. The system gain is of course 3.

Thus we can define this system as follows:

```
>>zeros=[-2];
>>poles=[0 -1 -3];
```

```
>>gain=3;
>>system=zpk(zeros,poles,gain)
```

Note that you can define this system in only one step as follows:

```
>>system=zpk([-2],[0 -1 -3],3)
```

Also, we can define this system directly using the transfer function method as follows:

```
>>system=tf([3 6],[1 4 3 0])
```

This means that the format of the zpk function is as follows:

Sys_name **= zpk (***System_zeros* **,** *System_poles, overall_gain***)**

Note that the zeros, poles, and gain of the system may be given without the transfer function, then you are required to define the system. Of course we are desired only in getting the zeros and the poles of the system without any care for how we got them.

Another example:

Define the system given by the shown T.F. to MATLAB:

$$\frac{Y}{R}=\frac{1}{S^2+5S+4}\ .$$

The code will be:

```
>>a=[ ];
>>b=[-1 -4];
>>k=1;
>>sys=zpk(a,b,k)
```

Or directly in one step:

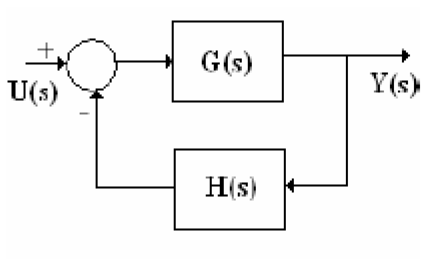
```
>>sys=zpk([ ],[-1 -4],1)
```

Note that this system doesn't have any zeros as the polynomial of the numerator of the transfer function can't equal zero any way. So, we define the system zeros to be an empty matrix as described.

1.3. Feedback systems

In this part, we will only care for using the previously discussed methods in interconnecting system components such as feedback systems to define the overall system to MATLAB.

So, let's consider the simple feedback system shown:



The feed forward T.F. G(s) and the feedback transfer function H(s) are simply systems that can be defined to MATLAB either by transfer function method or zeros-poles-gain method. Then we can form a feedback from these systems to get the overall system definition.

Example:

A closed loop system has a feed forward transfer function $G(S)=\frac{2}{S+1}$ and a feedback transfer function $H(S)=\frac{3}{S}$, you are required to get the overall transfer function of the feedback system.

Solution:

G(s) and H(s) should be defined separately as if isolated systems, then they should be combined to get the required feedback system using the following code:

```
>> G=zpk([ ],[-1],[2]);
>> H=zpk([ ],[0],[3]);
>> system=feedback(G,H)
```

Using the tf instruction we can build a similar code as follows:

```
>> G=tf([2],[1 1]);
>> H=tf([3],[1 0]);
>> system=feedback(G,H)
```

What about positive feedback?

The feedback instruction assumes negative feedback connection but if it's required to build a positive feedback system, an additional input argument is added as will be illustrated in the following example:

Example:

Get the transfer function of the system SYS whose feed forward transfer function is (2/S) and feedback transfer function is 0.25 knowing that SYS is a positive feedback system.

Solution:

```
>> G=tf([2],[1 0]);
>> H=tf([1],[4]);
>> SYS=feedback(G,H,+1)
```

Now, in general the feedback instruction has the following syntax:

*Sys_name***=feedback(***Feedforward_transfer_function***,***Feedback_transfer_function***,±1)**

Note that the **+1** is applied only for **positive feedback** systems while **-1** is applied for **negative feedback** systems (the -1 may be canceled as it is the default value for MATLAB).

1.4. State Space Representation

There are several different ways to describe a system of linear differential equations. The **state-space representation** is given by the equations:

$$\begin{cases}\underline{\dot{X}} = A\underline{X} + B\underline{U} \\ \underline{Y} = C\underline{X} + D\underline{U}\end{cases}$$

where X is an n by 1 vector representing the state (commonly position and velocity variables in mechanical systems), U is a scalar representing the input (commonly a force or torque in mechanical systems), and Y is a scalar representing the output. The matrices A (n by n), B (n by 1), and C (1 by n) determine the relationships between the state and input and output variables. Note that there are n first-order differential equations. State space representation can also be used for systems with multiple inputs and outputs (MIMO), but we will only use single-input, single-output (SISO) systems in these tutorials.

Example:

Consider the following system:

$$\begin{cases}\underline{\dot{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 980 & 0 & -2.8 \\ 0 & 0 & -100\end{bmatrix}\underline{x} + \begin{bmatrix} 0 \\ 0 \\ 100\end{bmatrix}u \\ y = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\underline{x} + [0]u\end{cases}$$

Enter the system matrices to MATLAB workspace as follows:

```
>> A= [0 1 0; 980 0 -2.8; 0 0 -100];

>> B= [0; 0; 100];

>> C= [1 0 0];

>> D= [0];

>> sys= SS(A, B, C, D)

sys =


  A =

        x1    x2    x3

   x1     0     1     0

   x2   980     0  -2.8
```

```
 x3    0    0  -100


B =
      u1
 x1    0
 x2    0
 x3  100


C =
     x1  x2  x3
 y1   1   0   0


D =
     u1
 y1   0


Continuous-time state-space model.
```

1.4.1. Canonical Forms

1.4.1.1. Modal Form

csys = canon(sys,'modal') returns a realization csys in modal form, that is, where the real eigenvalues appear on the diagonal of the matrix and the complex conjugate eigenvalues appear in 2-by-2 blocks on the diagonal of A.
For a system with eigenvalues $(\lambda_1, \sigma \pm j\omega)$, the modal matrix is of the form:

$$\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \sigma & \omega \\ 0 & -\omega & \sigma \end{bmatrix}$$

1.4.1.2. Companion Form

csys = canon(sys,'companion') produces a companion realization of sys where the characteristic polynomial of the system appears explicitly in the rightmost column of the A matrix. For a system with characteristic polynomial

$$P(s) = s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n$$

the corresponding companion A matrix is

$$A = \begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 & -a_n \\ 1 & 0 & 0 & \cdots & 0 & -a_{n-1} \\ 0 & 1 & 0 & . & \vdots & \vdots \\ \vdots & 0 & . & . & \vdots & \vdots \\ 0 & . & . & 1 & 0 & -a_2 \\ 0 & \cdots & \cdots & 0 & 1 & -a_1 \end{bmatrix}$$

For state-space models sys,

```
>> [csys,T] = canon(A,B,C,D,'type')
```

also returns the state coordinate transformation T relating the original state vector x and the canonical state vector x_c.

$$x_c = Tx$$

This syntax returns T=[] when sys is not a state-space model.

Transfer functions or zero-pole-gain models are first converted to state space using ss.

The transformation to modal form uses the matrix P of eigenvectors of the A matrix. The modal form is then obtained as

$$\begin{cases} \dot{\underline{x}}_c = P^{-1}AP\underline{x}_c + P^{-1}Bu \\ y = CP\underline{x}_c + Du \end{cases}$$

The state transformation T returned is the inverse of P.

Note:

The modal transformation requires that A the matrix be diagonalizable. A sufficient condition for diagonalizability is that A has no repeated eigenvalues.

The companion transformation requires that the system be controllable from the first input. The companion form is often poorly conditioned for most state-space computations; avoid using it when possible.

1.5. Transformations

In this section, the transformation from any representation method to any other one and getting the data of any representation will be covered. First, getting the information of any representation should be covered as follows:

Assume that a certain system SYS is defined for MATLAB in transfer function form and you are required to get the parameters of this representation which are the polynomial of the numerator and the polynomial of the denominator of the transfer function corresponding to this system. In this case, you may use the **tfdata** instruction.

Example:

The transfer function of a certain system is given by:

$$\frac{Y(S)}{U(S)} = \frac{2S+3}{S^2+5S+6}$$

And it's defined for MATLAB using the TF form with the system name SYS, so it's required to get the parameters of the TF representation using MATLAB.

Solution:

```
>> [num,denom]=tfdata(SYS)
```

Now, ***num*** contains the polynomial of the numerator of the transfer function of SYS, which is [2 3] while the polynomial of the denominator is in ***denom*** [1 5 6]. If the same system is defined for MATLAB using the ZPK method and you are required to get the poles, zeros, and gain of the system; it will be suitable to use the zpk data instruction as follows:

```
>> [z,p,k]=zpkdata(SYS)
```

Now, ***Z*** contains the zeros of the system, ***P*** contains the poles of the system, and ***K*** contains the overall gain of the system. Z=[-1.5] , P=[-2 -3], K=[2].

[*numerator,denomenator*]=tfdata(*System_name_defined_in_tf_format*)

[*zeros, poles, gain*]=zpkdata(*System_name_defined_in_zpk_format*)

Transformation:

Suppose that a system is defined using append/connect instruction or zpk instruction and you would like to get this system in transfer function form. In this case, you may use the tf instruction in the shown syntax.

***System_representation_in_TF_form* = tf (*System_representation_ in_any_form*)**

Example:

A system is defined for MATLAB in the zpk form with name SYSTEM and it's required to get the system representation in transfer function form.

Solution:

```
>> SYSTEM=tf(SYSTEM)
```

Now, assume that the system is defined in the tf from and it's required to get the system representation in the zpk form and hence get the zeros, poles, and gain of the system.

System_representation_in_ZPK_form **= zpk (***System_representation_ in_any_form***)**

```
>> SYSTEM=zpk(SYSTEM);
>>[Z,P,K]=zpkdata(SYSTEM)
```

2. *System characteristics*

2.1. *Stability*

Stability is the most important property of the system as it implies that if the system is achievable or not. There are a lot of techniques to analyze the stability of any system but in this lab. manual, we will focus on using MATLAB for stability analysis.

Because stability of a system can be determined through the location of its poles, it will be more useful to ask MATLAB about the location of these poles then examine their location by direct look or using a simple code programming.

Remember that stable system have their poles in the left half plane while unstable ones have their poles in the right half plane.

To get the location of the system poles using MATLAB, you may use the following instruction syntax.

eig(*System_name***) or pole(***System_name***)**

The output of this instruction is a vertical vector containing the location of the poles of the system under test. Simply, you can use if conditions to check that the location of the poles is in the left half plane for stable system.

Example:

A system is defined by the shown transfer function:

$$\frac{V_o}{V_i} = \frac{S+3}{S^2+S+4}$$

And it's required to check the stability of this system.

Solution:

The code in this case may be:

```
>> sys=tf([1 3],[1 1 4]);
>> eig(sys) or Pole(sys)
```

Now, by a simple look at the ***ans*** variable generated by MATLAB that contain the location of the poles of the system, you can decide whether this system is stable or not. The ***ans*** variable will contain: -0.5000 + 1.9365i   -0.5000 - 1.9365i

Which means that the system is stable.

Note also that the code may be:

```
>> sys=tf([1 3],[1 1 4]);
>> poles=eig(sys);
>> if poles(:)<0
disp('System is stable');
else
disp('System is unstable or critically stable');
end
```

<u>PZMAP method:</u>

The ***pzmap*** function displays a map for the system on which the poles and zeros of this system are displayed. So, by looking at this map you can directly deduce the stability of the system.

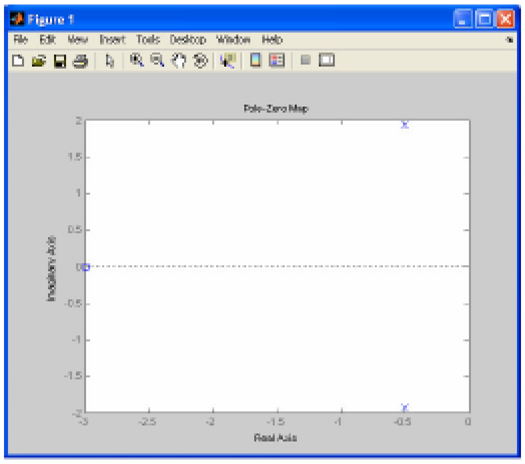
You can use the following syntax for the pzmap function.

pzmap(*System_name*)

Now, the code will be:

```
>> sys=tf([1 3],[1 1 4]);
>> pzmap(sys);
```

The map contains o's to represent the locations of the zeros and x's for poles.



2.2. *Controllability*

Controllability is the ability to control the locations of the poles of a certain system to get a desired system response. Systems are classified into controllable and uncontrollable systems through a simple mathematical check. In this section, classifying systems into controllable or uncontrollable will be taken into consideration.

To do so, the ***ctrb***, ***rank***, and ***ss*** functions will be used. Usually, you will use the following syntax:

```
rank(ctrb(ss(System_name)))
```

If the answer of this instruction is equal to the system's order, the system is controllable, otherwise the system is uncontrollable.

Example:

A system is defined by the transfer function:

$$\frac{Y(S)}{U(S)} = \frac{2S+3}{S^2+5S+6}$$

And you are required to check whether this system is controllable or not.

Solution:

First, define the system, then use the above syntax.

```
>> sys=tf([2 3],[1 5 6]);
>> rank(ctrb(ss(sys)))
```

The answer of the final instruction is equal to **2** and the system under check is of 2nd order (the highest power of S in the denominator of the transfer function), so the system is controllable. The previous syntax first transforms your system into state space form (one of the system representation methods) then gets the controllability matrix of the state space format of the system and finally checks its rank which should be equal to the system order for controllable system.

2.3. Observability

Observability is the ability to determine system states through the observation of its output in finite time intervals. Systems are classified into observable and unobservable systems through a simple mathematical check. In this section, classifying systems into observable and unobservable will be taken into consideration.

To do so, the ***obsv***, ***rank***, and ***ss*** functions will be used. Usually, you will use the following syntax:

rank(obsv(ss(*System_name*)))

If the answer of this instruction is equal to the system's order, the system is observable, otherwise the system is unobservable.

Example:

A system is defined by the transfer function:

$$\frac{Y(S)}{U(S)} = \frac{S+1}{S^2+S+4}$$

And you are required to check whether this system is observable or not.

Solution:

First, define the system, then use the above syntax.

```
>> sys=tf([1 1],[1 1 4]);
>> rank(obsv(ss(sys)))
```

The answer of the final instruction is equal to **2** and the system under check is of 2nd order, so the system is observable. Similarly as in the controllability check, the previous syntax first transforms your system into state space form then gets the observability matrix of the state space format of the system and finally checks its rank which should be equal to the system order for observable system.

3. *Frequency analysis*

3.1. *Bode plots & system margins*

MATLAB can help getting the frequency response of any system using bode plots. First, you should define the feed forward transfer function of the system. Then, use the following syntax.

bode(*System_feed_forward_transfer_function***)**

Example:

Given the feed forward transfer function of a certain system:

$$G(S) = \frac{1}{S(S^2 + S + 1)}.$$

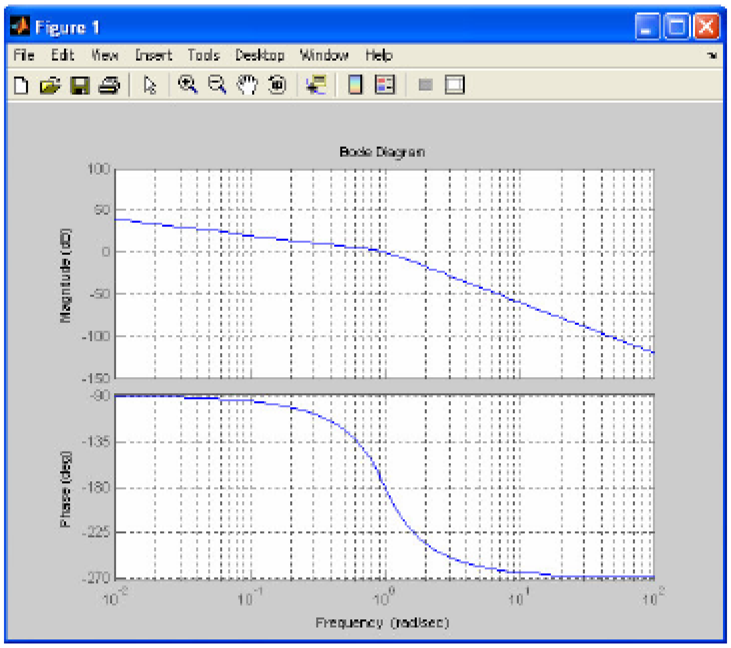
You are required to plot the bode plots of this system.

<u>Solution:</u>

On the MATLAB command window, type:

```
>> G=tf([1],[1 1 1 0]);
>> bode(G)
```

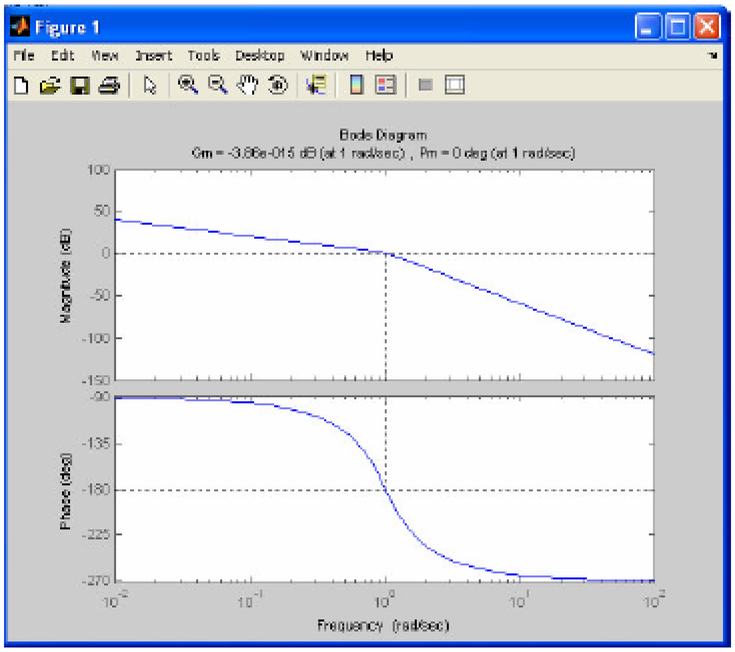
You will get the shown figure after right clicking the figure window and choosing Grid.



But it will be better to use the **margin** instruction for the previous example as it not only displays the frequency response via bode plots but also calculates the system parameters such as: Gain Margin GM, Phase Margin PM, gain crossover frequency ω_{gc} , and phase crossover frequency ω_{pc} . You can use the following syntax for that purpose.

```
margin(System_feed_forward_transfer_function)
```

The result will be as shown below:



Finally, if you would like to get the parameters of the system directly or without bode plots, you can use the **allmargin** instruction for that purpose.

```
allmargin(System_feed_forward_transfer_function)
```

The output in the case of the previous example will be:

```
GMFrequency: 1.0000                DMFrequency: 1.0000
GainMargin: 1.0000                 DelayMargin: 0
PMFrequency: 1.0000                Stable:1
PhaseMargin: 0
```

4. Control Design

4.1. Pole-placement Design

Sometimes you are required to design a compensator to achieve specific requirements on the given system. The design of the compensator is somehow a logical operation. It may be a very simple process in which the gain of the system should be adjusted to a certain value using either an amplifier or an attenuator as a compensator at each state. In this part we will focus on the simple gain adjustment using pole-placement method.

Given the single- or multi-input system

$$\dot{\underline{x}} = A\underline{x} + Bu$$

and a vector p of desired self-conjugate closed-loop pole locations, 'place' computes a gain matrix K such that the state feedback u=-Kx places the closed-loop poles at the locations p. In other words, the eigenvalues of match the entries of p (up to the ordering).

$K = place(A, B, p)$ computes a feedback gain matrix K that achieves the desired closed-loop pole locations p, assuming all the inputs of the plant are control inputs. The length of p must match the row size of A.

Example:

Consider the following SSR:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 980 & 0 & -2.8 \\ 0 & 0 & -100 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 100 \end{bmatrix}$$

Obtain the state-feedback control gains such that the new poles are $-10 \pm 10i, -20$.

Solution:

Using MATLAB:

```
>> A = [0 1 0;980 0 -2.8; 0 0 -100];
>> B = [0; 0; 100];
>> p1 = -10+10i;
>> p2 = -10-10i;
>> p3 = -20;
>> Kc = place(A,B,[p1, p2, p3])
```

```
Kc =


 -154.2857   -5.6429   -0.6000
```