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

Written By:

Mostafa Adly
Tarek Abubakr

Table of Contents

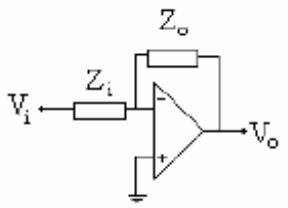
1. System Identification

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 → 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:

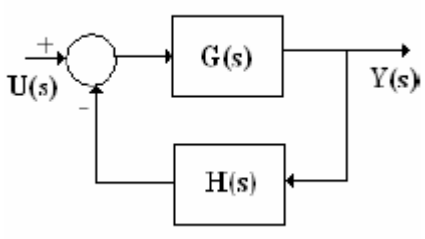
```
>>sys=zpk([ ],[-1 -4],1)
```

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 feed back 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 from 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 feed back 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)**

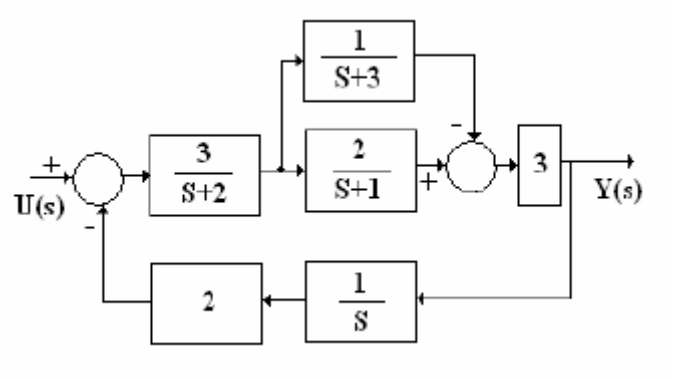
Note that the **+1** is applied only for **positive feedback** systems while **-1** is applied for **negative feedback** systems (the -1 may be canceled as it is the default value for MATLAB).

1.4. Block diagrams

Sometimes systems may be defined using their block diagram. So, how to define a system in block diagram form for MATLAB? And how to get its transfer function without involving in block-diagrams' reduction rules?
Let's analyze this method using an example:

Example:

Define the following system to MATLAB and hence show how to get its transfer function.



You can use the rules of reduction of block diagrams to simplify this system and get the transfer function of this system Y(s)/U(s).

Step (1)

Define each block as a separate sub-system and assume that the sub-systems are:

$$Sys1 = \frac{3}{S+2} \qquad Sys2 = \frac{1}{S+3} \qquad Sys3 = \frac{2}{S+1} \qquad Sys4 = 3 \qquad Sys5 = \frac{1}{S} \qquad Sys6 = 2$$

Now, define each sub-system to MATLAB:

```
>>sys1=tf([3],[1 2]);
>>sys2=tf([1],[1 3]);
>>sys3=tf([2],[1 1]);
>>sys4=tf([3],[1]);
```

```
>>sys5=tf([1],[1 0]);
>>sys6=tf([2],[1]);
```

Step (2)

Now, these sub-systems should be appended to get the overall system connecting them using the append instruction as follows:

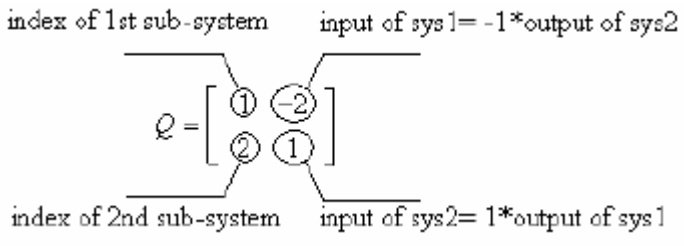
```
>>sys=append(sys1,sys2,sys3,sys4,sys5,sys6);
```

Step (3)

MATLAB should know how these sub-systems are connected through a connection matrix, in which the connections between sub-systems are indicated.

The connection matrix is formed as follows:

Each sub-system has a row in this matrix, so the first column in each row is the index of each sub-system (1,2,3,4, or 5). The remaining elements in each row specify the sub systems to which the sub-system whose index is indicated in the first column is connected. If the connection is due to a summing element and of subtraction operation, the index of the sub-system of connection should be of negative sign. For example:



This means that sys1 is connected to sys2 through a summing element in which the output of sys2 is subtracted from the certain input forming the input of sys2 then the output of sys1 is fed as an input to sys2 without any inversion of sign.

Note that the index of the sub-systems is got from the order of the sub-system in the

append instruction.

Now, the connection matrix of our example is as follows:

$$Q = \begin{pmatrix} 1 & -6 & 0 \\ 2 & 1 & 0 \\ 3 & 1 & 0 \\ 4 & -2 & 3 \\ 5 & 4 & 0 \\ 6 & 5 & 0 \end{pmatrix}$$

Which means that the input of sys1 comes from the output of sys6 only, the input of sys2 comes from the output of sys1 only, the input of sys3 comes from the output of sys1 only, the input of sys4 comes from the outputs of sys3 and sys2 but inverted, the input of sys5 comes from the output of sys4, and finally the input of sys6 comes from the output of sys5.

Now, define this matrix to MATLAB:

```
>> Q=[1 -6 0; 2 1 0; 3 1 0; 4 -2 3; 5 4 0; 6 5 0];
```

Step (4)

Finally, the main input and output of the system should be specified. This is done by specifying the sub-system in which the main input is fed and the sub-system from which the main output is got using the connect instruction as follows:

```
>>Final_sys=connect(sys,Q,[1],[4])
```

Which means that the final connected system is the connection of the appended system through a connection matrix Q where the main input of the system is fed to sys1 of the block diagram and the main output is taken out of sys4.

The final complete code will be:

```
>>sys1=tf([3],[1 2]);
>>sys2=tf([1],[1 3]);
>>sys3=tf([2],[1 1]);
>>sys4=tf([3],[1])
>>sys5=tf([1],[1 0]);
>>sys6=tf([2],[1]);
```

```
>>sys=append(sys1,sys2,sys3,sys4,sys5,sys6);
>> Q=[1 6 0; 2 1 0; 3 1 0; 4 -2 3; 5 4 0; 6 5 0];
>>Final_sys=connect(sys,Q,[1],[4])
```

Then you will get the overall transfer function of the system.

***Sys_name*=append(*Sub-sys1* , *Sub-sys2* , *Sub-sys3* , ……..)**

***Sys*=connect(*Appended_system*, *Connection matrix*, *main i/p subsys*, *main o/p subsys*)**

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 Response

2.1. The step response

The response of a system is the shape and characteristics of the output of this system for a certain input. In the analysis of systems, some basic test input signals are used such as the unit step, the impulse, and the ramp function.

In this section, the response of the system for a step input will be taken into consideration while the response for different inputs will be discussed later.

To get the step response of a certain system using MATLAB, the system should first be defined using either block diagrams, transfer functions, or zeros-poles-gain, then use the following function syntax.

step (*System_name*, *time_duration_of_analysis*)

In this case, the response of the system will be displayed in the time interval specified by the second input argument of the step function.

Example:

A system has the following transfer function:

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

and you are required to get the step response of this system and hence get the value of the *rise time, peak time, settling time, system overshoot,* and the *steady state error*.

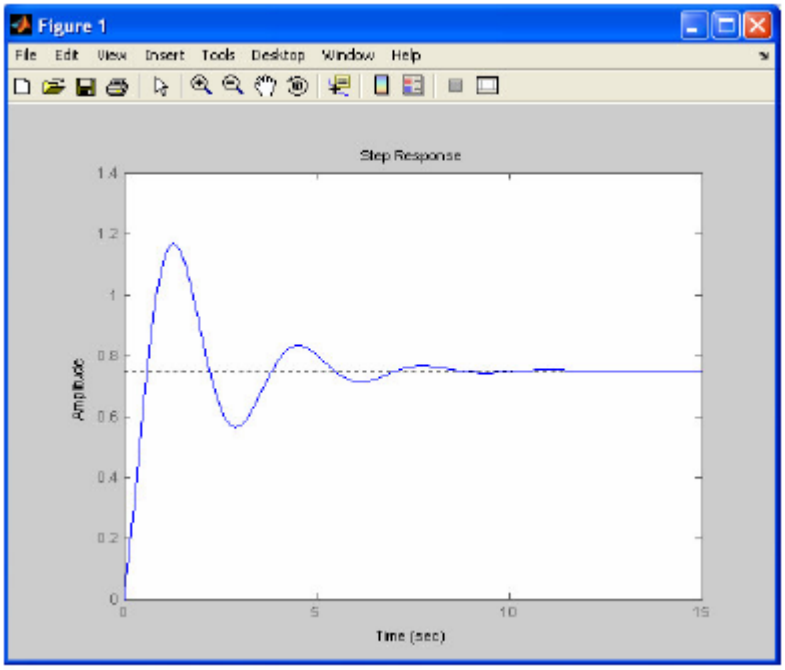
Solution:

A suitable period of analysis is about 15 seconds as an estimation (or by trial), so the time period is set to be 15 seconds with step size of 0.01 second. The code will be as follows.

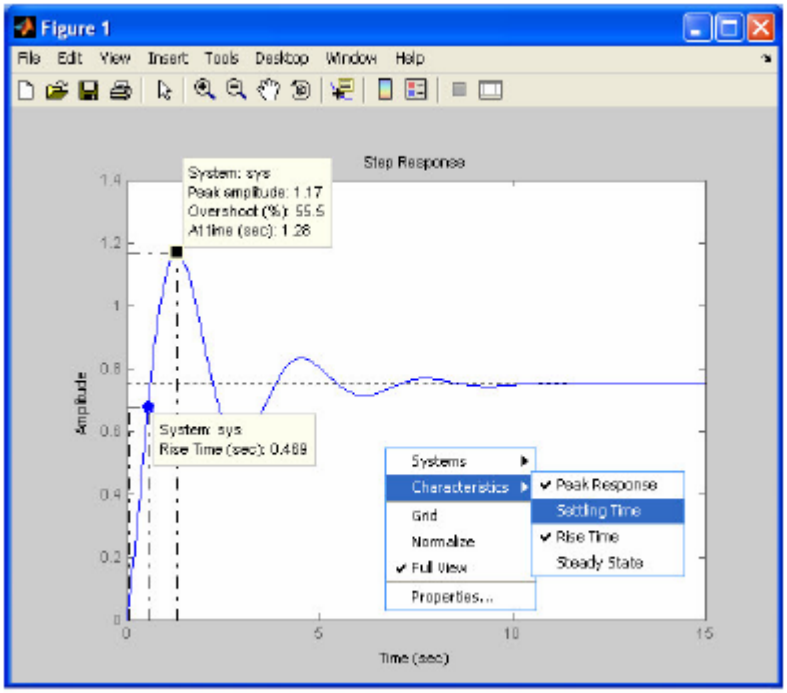
```
>> t=0:0.01:15;
```

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

The output will be displayed as in the following window:



On the previously shown window, right click your mouse and then activate the characteristics as shown:



It's now clear that the transient and steady state characteristics can be obtained from the figure or the window in which the step response is displayed using the right mouse key and then choosing characteristics to select between any one of the characteristics.

2.2. The impulse response

In the same manner, the response of the system to a unit impulse is called the impulse response which can be got using the following function syntax.

impulse (*System_name*, *time_duration_of_analysis*)

The output will also be displayed in a window that allows right clicking to get the characteristics of the response.

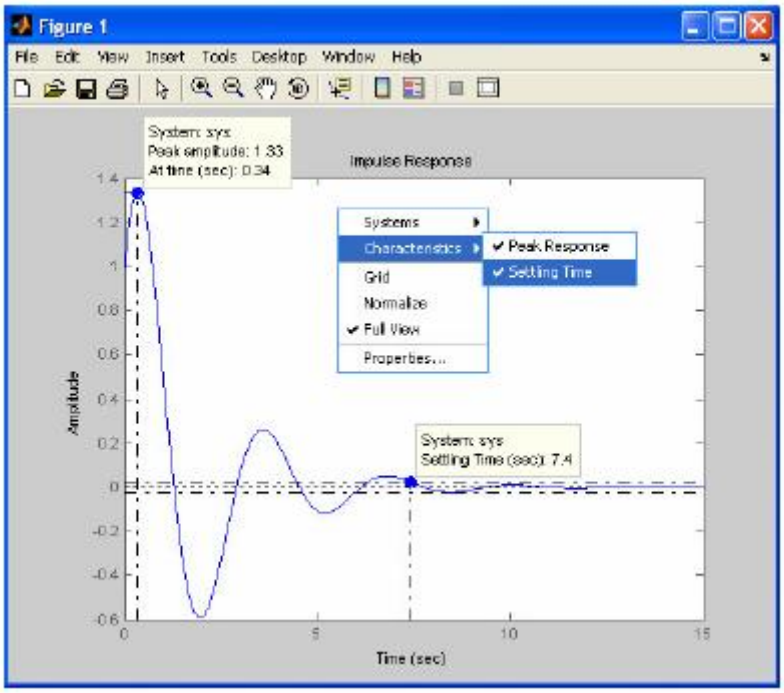
Example:

For the previously discussed example, it's required to get the impulse response of this system and hence find the peak response and the settling time.

Solution:

```
>> t=0:0.01:15;
>> sys=tf([1 3],[1 1 4]);
>> impulse(sys,t);
```

The output will be displayed as shown:



2.3. Arbitrary input response

Sometimes, there may be a need to check the system response if the input is a sinusoidal input or in general an arbitrary input signal. In this case, you may use the following function syntax.

lsim(*System_name* **,** *Input_function* **,** *time_duration_of_analysis* **)**

where the input function is a function of the time of analysis which means that it is a vector of the same length as the time vector.

Example:

A system is defined by the following transfer function:

$$\frac{O/P}{I/P} = \frac{2}{S^2 + 2S + 6}$$

And it's required to get the response of the system for an input signal u(t), where:

a) U(t) is a unit ramp function, where U(t)=t.

b) U(t) is a cosine function, where U(t)=cos(t).

Solution:

In this case, the time period of analysis should be specified first then the input function is defined and finally the LINEAR SIMULATION function should be applied on the defined system as follows.

```
>> sys=tf([2],[1 2 6]);
>> t=0:0.01:10;
>> u1=t;
>> u2=cos(t);
>> figure(1);
>> lsim(sys,u1,t);
>> grid;
```

```
>> figure(2);
>> lsim(sys,u2,t);
>> grid;
```

In this code, u1 is a ramp function as it increases with time in the same rate as the time vector t, while u2 is a cosine function. Both signals are applied to the same system in a time interval specified by t and the output of each input signal is displayed in a separate figure using the *FIGURE* instruction. The figures will look like the following ones.

Note that the linear simulation function can also display the peak response of the system output using the right mouse key and then the characteristics option. The plots displayed in each figure represent the input signal (in gray) and the output (in blue).

Now, it's clear that MATLAB is helpful for getting the response of the system for any input function form.

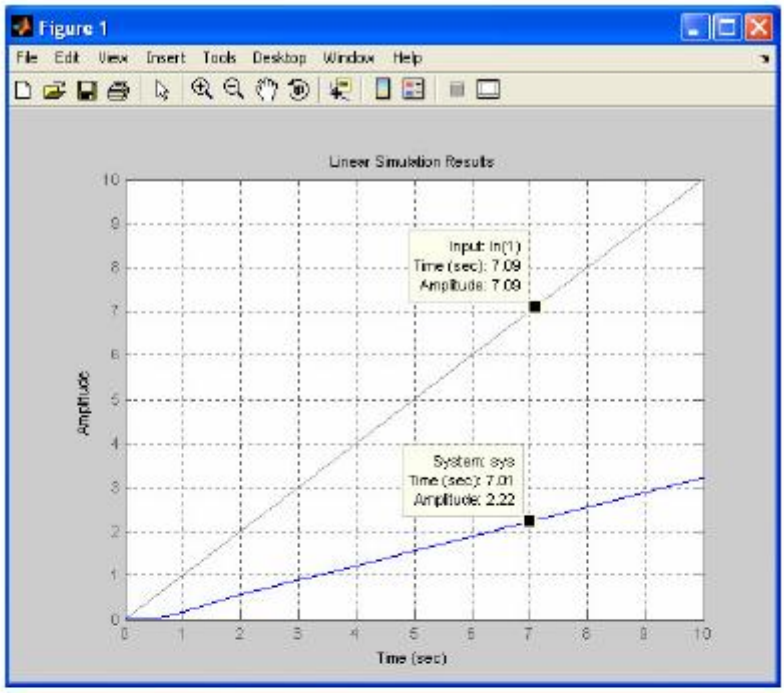


Figure 1
File Edit View Insert Tools Desktop Window Help
Linear Simulation Results
Input: In(1)
Time (sec): 7.09
Amplitude: 7.09
System: sys
Time (sec): 7.01
Amplitude: 2.22
Amplitude
Time (sec)

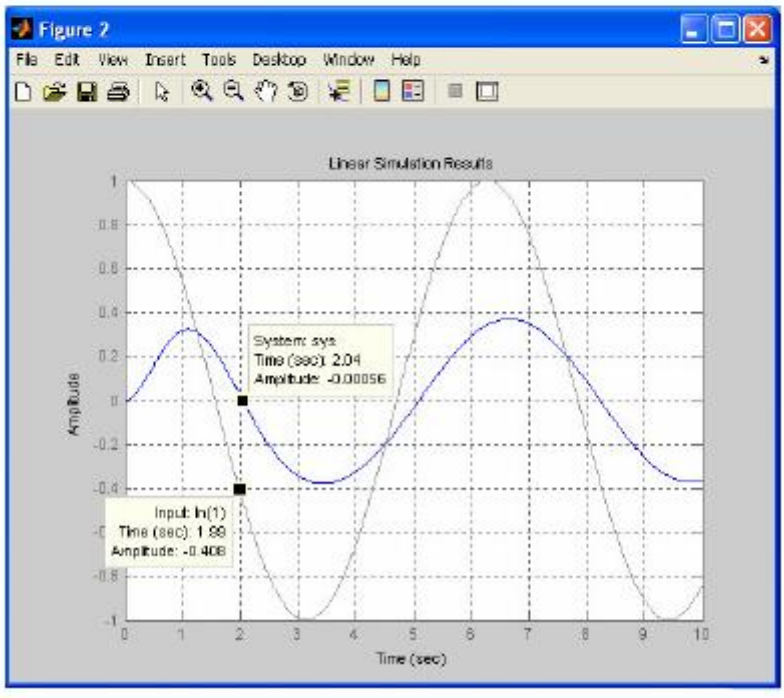


Figure 2
File Edit View Insert Tools Desktop Window Help
Linear Simulation Results
System: sys
Time (sec): 2.04
Amplitude: -0.00056
Input: In(1)
Time (sec): 1.99
Amplitude: -0.408
Amplitude
Time (sec)

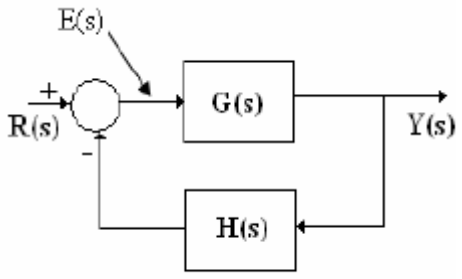
2.4. Steady state response

The response of any system has two phases; the **transient phase** in which it is desired to get the rise time, peak time, settling time, and maximum peak - and the **steady state phase** in which it is desired to get the steady state final value of the output and the steady state error. The steady state error is the shift or the difference between the desired output and the current fed back output.

Steady state error = Desired output - Current fed back output

This means that the steady state error for the shown system can be calculated as follows:

$$\therefore E(s) = R(s) - H(s)Y(s)$$
$$\because Y(s) = G(s)E(s)$$
$$\therefore \mathrm{E(s)} = \mathrm{R(s)} - \mathrm{G(s)H(s)E(s)}$$
$$\therefore E(s)[1 + G(s)H(s)] = R(s)$$
$$\therefore E(s) = \frac{R(s)}{1 + G(s)H(s)}$$



Then, How to get the steady state error given the error function?
You can use the final value theorem:

$$e_{ss} = \lim_{t \to \infty} e(t) = \lim_{s \to 0} SE(s)$$

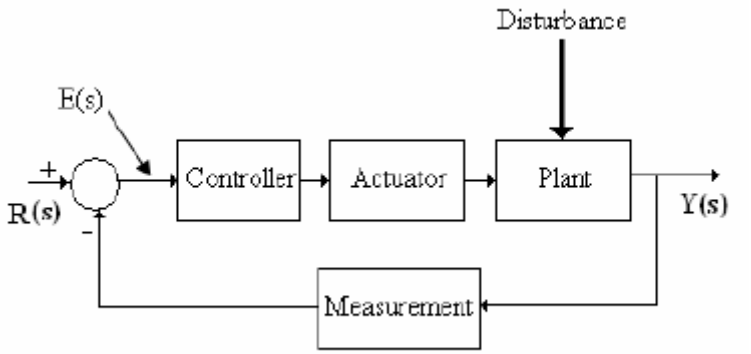
So the steady state error can be calculated using the general following formula:

$$e_{ss} = \lim_{s \to 0} \frac{SR(s)}{1 + G(s)H(s)}$$

The used code will be:

```
>> syms t s
>> rt=.....; % any time function
>> r=laplace(rt);
>> g=.....; % any s-domain function
>> h=.....; % any s-domain function
>> ess=limit(s*r/(1+g*h),0)
```

Sometimes the error signal is called the "error actuating signal" because while there is an error, the plant G(s) is actuated but if error equals zero the plant is deactivated. Usually the error signal is fed to either a controller to control the process through an actuator or directly to the actuator if there was no need for a controller.



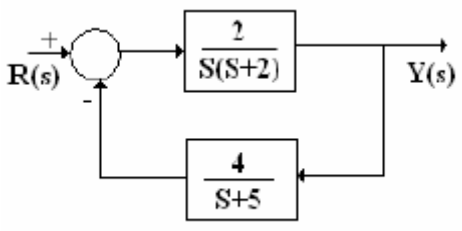
There may be a special case in which the **input is a step function** and the **feedback transfer function H(s)= constant C.** In this case the steady state error can be calculated using the following formula:

$$e_{ss} = R_{ss} - CY_{ss} = A - CY_{ss}$$

Where A is the amplitude of the step function of the input, Y_{ss} is the steady state value of the output which can be got from the step response using MATLAB step function. Revise the rules of calculating the steady state error using the error coefficients mentioned in the lecture.

Example (1):

A feed back control system is shown in the following figure:



It's required to get the steady state error for an input R(t)=3 and R(t)=4t.

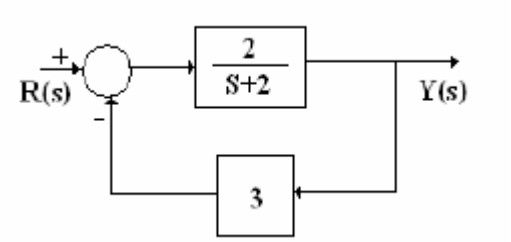
Solution:

```
>> syms t s
>> rt1=3*t/t; % due to a bug in MATLAB 7
>> rt2=4*t;
>> r1=laplace(rt1);
>> r2=laplace(rt2);
>> g=2/(s*(s+2));
>> h=4/(s+5);
>> ess1=limit(s*r1/(1+g*h),0)
>> ess2=limit(s*r2/(1+g*h),0)
```

It is clear that ess1=0 and ess2=5.

Example (2):

A feed back control system is shown in the following figure:



It's required to get the steady state error for an input R(t)=unit step and R(t)=10 units.

Solution:

```
>> syms t s
>> rt1=1*t/t; % due to a bug in MATLAB 7
>> rt2=10*t/t;
>> r1=laplace(rt1);
>> r2=laplace(rt2);
>> g=2/(s+2);
>> h=3;
>> ess1=limit(s*r1/(1+g*h),0)
```

```
>> ess2=limit(s*r2/(1+g*h),0)
```

It is clear that ess1=1/4 and ess2=5/2.

You may solve it using the special case because H(s)=constant (:H(s)=3) and the input is a unit step.

```
>> G=tf([2],[1 2]);
>> H=tf([3],[1]);
>> sys=feedback(G,H);
>> t=0:0.1:10;
>> step(sys,t)
```

Form the step response, you can find that $Y_{ss} = 0.25$ and hence $e_{ss} = 1 - 3*0.25 = 0.25$. You can also use the following instruction:

```
>> s1=step(sys,t);
>> ess1=1-3*s1(end)
>> s2=10*step(sys,t);
>>ess2=10-3*s2(end)
```

2.5. Disturbance analysis

Continuous-time control systems may have some disturbances that affect their responses. The disturbance is by default unpredictable which leads to an unpredictable output. So, it is usually important to check the response of the system with respect to the disturbance. You can evaluate the effect of the disturbance on the by applying a zero test input signal which means that the output will be only due to the disturbance signal.

You can find that the output of any system is given by the following general form:

$$Y(s) = \frac{Y(s)}{R(s)}_{D(s)=0} * R(s) + \frac{Y(s)}{D(s)}_{R(s)=0} * D(s)$$

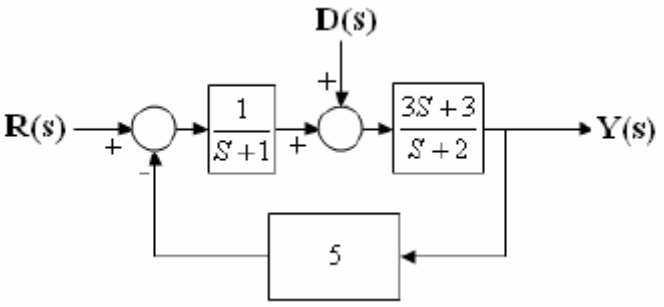
Where Y(s) is the output of the system, R(s) is the input of the system, and D(s) is the disturbance.

$\frac{Y(s)}{R(s)}_{D(s)=0}$ is the transfer function relating the output to the input while the disturbance is zero and $\frac{Y(s)}{D(s)}_{R(s)=0}$ is the transfer function relating the output to the disturbance calculated when R(s)=0.

Some times you can estimate the value of the output under the effect of the disturbance when the input is zero and hence it will be possible to subtract this value of the output from the output at any time to get an output free from the effect of the disturbance.

$$Y_{free}(s) = Y(s) - \frac{Y(s)}{D(s)}_{R(s)=0} * D(s) = \frac{Y(s)}{R(s)}_{D(s)=0} * R(s)$$

MATLAB enables us to get these transfer functions by specifying the main input and the disturbance in the connect instruction as follows:




```
>> G1=tf([1],[1 1]);
>> G2=tf([3 3],[1 2]);
>> G3=tf([5],[1]);
>> sys=append(G1,G2,G3);
>> Q=[1 -3;2 1;3 2];
>> Final_sys=connect(sys,Q,[1 2],[2])
```

3. System characteristics

3.1. 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
```

PZMAP method:

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.

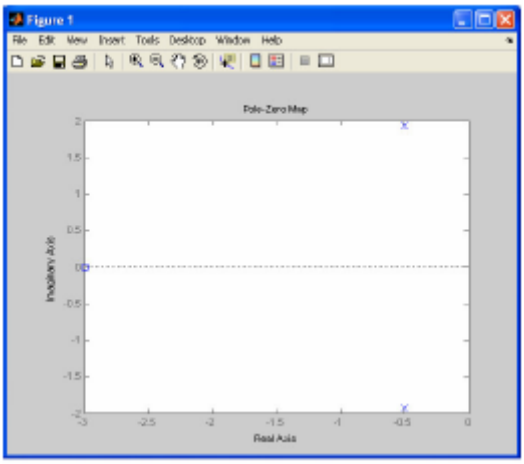
You can use the following syntax for the pzmap function.

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.



3.2. 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.

3.3. 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.

3.4. Damping

MATLAB can help you in analyzing systems by getting their damping ratio _ and natural frequency ω_n for further analysis and calculations of the transient response parameters.
To do so, you can use the following syntax.

[w,z]=damp(*System_name***)**

The output argument **w** is a vertical vector containing the natural frequency of each pole in the system, while **z** is also a vertical vector containing the damping ratio of each pole in the system.

Example:

A system is defined by the following transfer function:

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

It's required to get the settling time of this system.

Solution:

The settling time of a system is calculated by the formula:

$$t_s = \frac{4}{\eta \omega_n}$$

so it's now required to get the damping ratio zeta and the natural frequency ω_n of the system. The code will be:

```
>> sys=tf([1 2],[1 1 4]);
>> [w,z]=damp(sys);
>> ts=4./(w.*z)
```

Note that the code uses the dot operator because **w** and **z** are both vectors and it's required to deal with their adjacent components. Why?

You can answer this by typing the following instruction and noticing the output.

```
>> damp(sys)
```

The output is as follows:

Eigenvalue	Damping	Freq. (rad/s)
-5.00e-001 + 1.94e+000i	2.50e-001	2.00e+000
-5.00e-001 - 1.94e+000i	2.50e-001	2.00e+000

Which means that the first element in the damping vector is related to the first one in the frequency vector and so on.

3.5. Root locus

It's a method of analyzing systems and compensation design using the possible root locus. It simplifies the method of gain adjustment to get a desired pole location. To plot the root locus of a system, the system's feed forward transfer function should be first defined then use the following syntax.

rlocus(*System_feed_forward_transfer_function*)

Example:

Given the feed forward transfer function of a certain system:

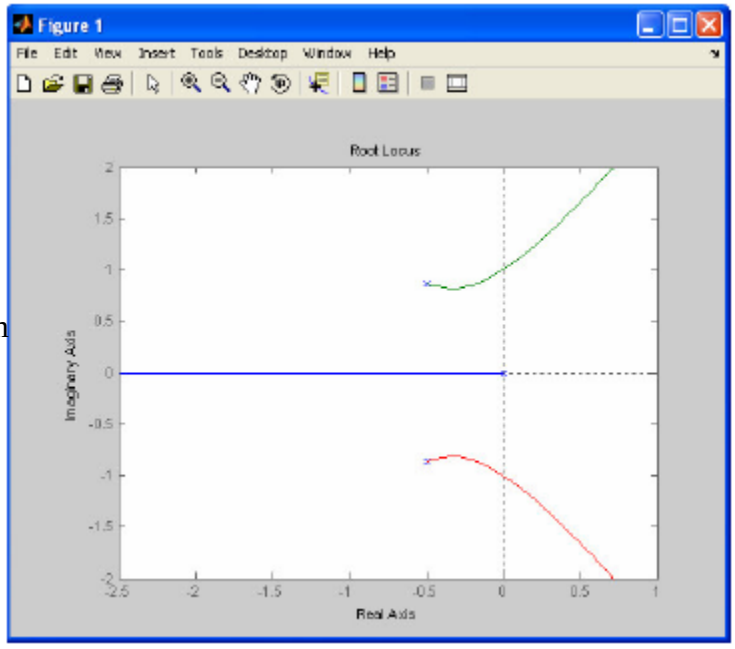
$$G(S) = \frac{1}{S(S^2 + S + 1)}.$$

You are required to plot the root locus of this system, hence find the value of the system gain such that the poles of the system lie at: 0, -0.244± 0.831i.

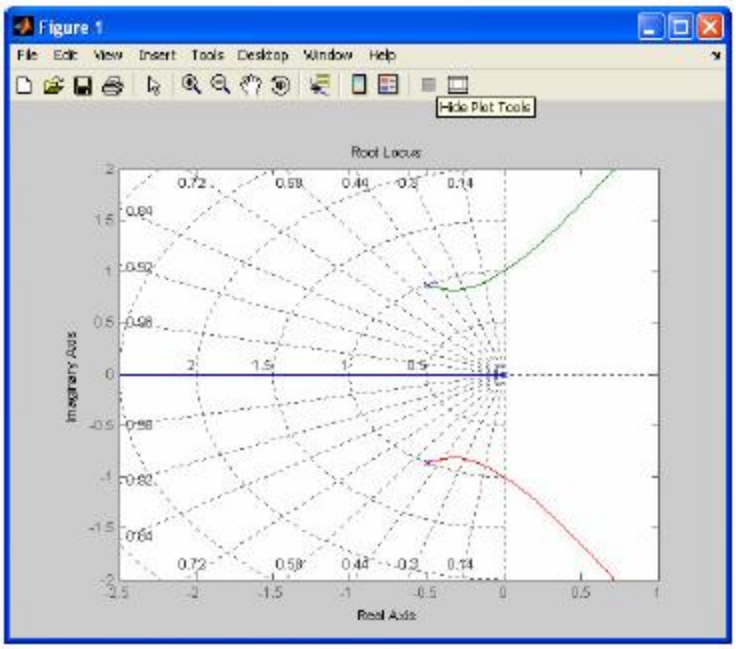
S olution:

On the MATLAB command window, type:

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



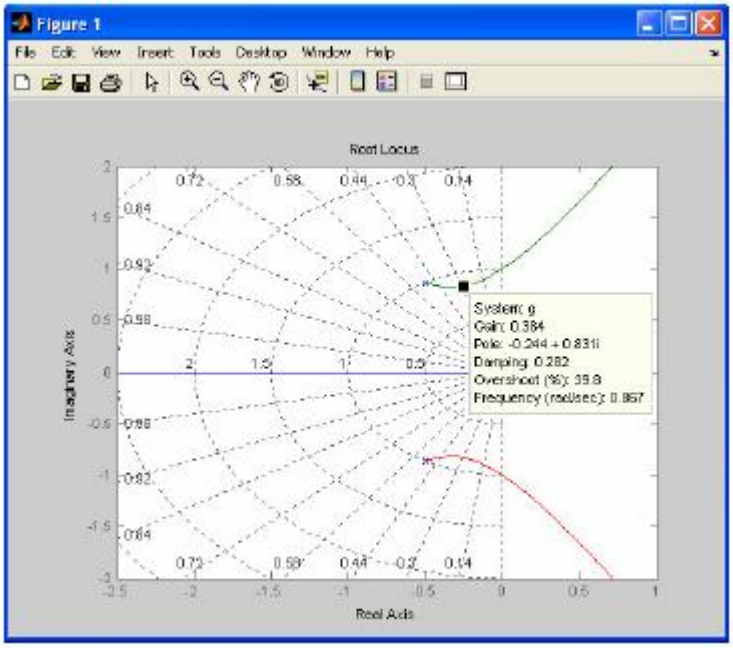
The shown window will appear.
The plot specifies the locations of the poles and their possible loci.
For better identification for the system the values of zeta and ω_n should be displayed. To do so, you need to right click on the shown window and select Grid from the pop-up menu.
The result is shown in the figure at the beginning of the next page.



Now, constant _ lines appear, also constant ω_n lines appear.
Thus full identification of the system is possible.
Currently, the poles of the system lie at: 0, -0.5± 0.866i.
In order to locate the poles of the system in another location, simple gain adjustment should be made.

To do so using MATLAB, just click any point on the plots of the locus then drag it to check different gains and pole locations as shown in the next figure. As shown, it's now possible to change the system gain to get another location for the poles.

From this figure, the suitable value of the gain is: **0.384**.
Note that the question may be formed in many other forms such as asking about the value of the gain that ensure certain damping ration, percent overshoot, or frequency.
All you have to do, is to drag the cursor on the plots and get that gain to ensure a certain property for the system.



4. Frequency analysis

4.1. 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:

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

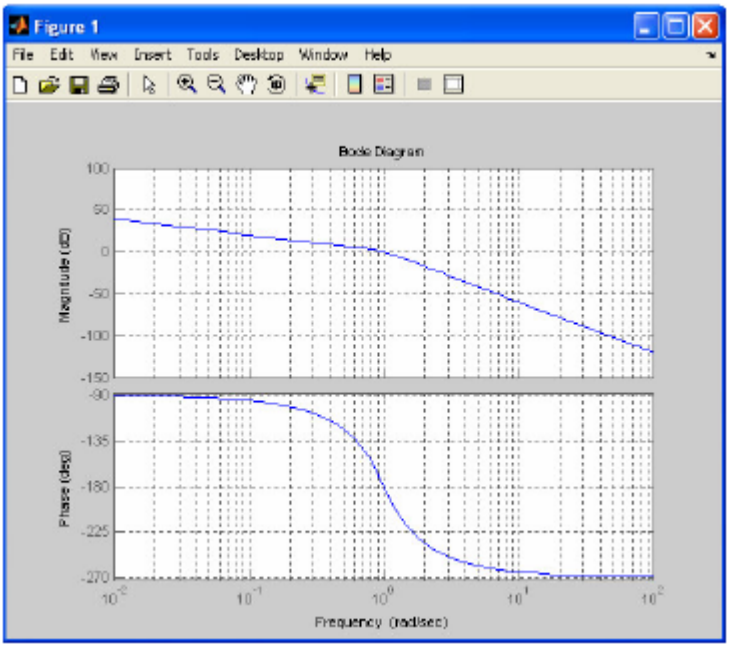
You are required to plot the bode plots of this system.

Solution:

On the MATLAB command window, type:

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

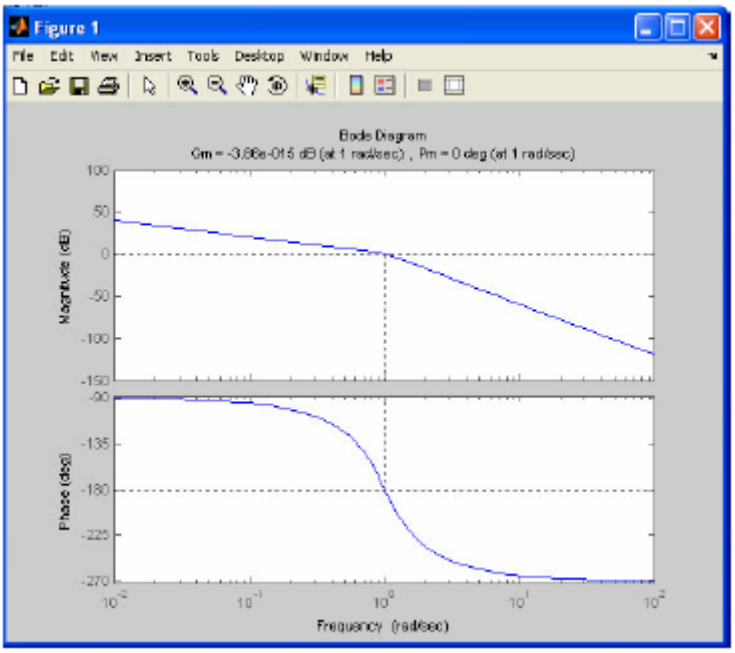
You will get the shown figure after right clicking the figure window and choosing Grid.



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

GainMargin: 1.0000

PMFrequency: 1.0000

PhaseMargin: 0

DMFrequency: 1.0000

DelayMargin: 0

Stable: 1

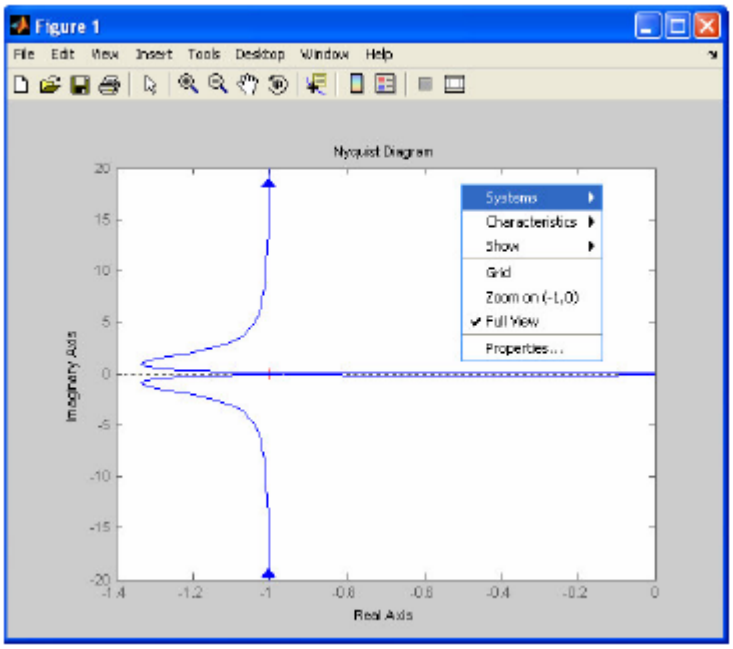
4.2. Nyquist plots

It's another method of analyzing system's frequency response. You can use MATLAB to get the Nyquits response of any system using the following syntax.

nyquist(*System_feed_forward_transfer_function***)**

But note that the resulting figure is some how different from our ordinary plotted Nyquist plots because we are used to plot the absolute value of the feed forward transfer function versus its angle, while MATLAB calculates both the real part and the imaginary part of the frequency response of the feed forward transfer function. For the previously illustrated example, type:

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



From the pop-up menu, you can choose grid or display system characteristics on the plot.

5. Control Design and compensation

5.1. Compensation using root locus

Sometimes you are required to design a compensator to achieve specific requirements on the given system. The design of the compensator is some how 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 and sometimes a very complicated process in which you need adding poles or zeros to the system using lead, lag, or lead-lag compensators. In this part we will focus on the simple gain adjustment using root locus method then the compensation by adding zeros or poles will be covered in the next section.

Example:

A unity feed back control system has a feed forward transfer function:

$$G(S) = \frac{1}{S(S^2 + 3S + 2)}$$

Assuming a unit step input of the system:

a) Find the peak time of the system.

b) Find the percent overshoot of the system.

c) Design a compensator to have an overshoot of not more than 5%.

Solution:

Using MATLAB:

```
>> G=tf([1],[1 3 2 0]);
>> sys=feedback(G,1);
>> t=0:0.01:30;
>> step(sys,t)
```

From the step response figure use the characteristics (peak response) to get the peak time and the overshoot.

$(i) t_p = 6.08 \sec$

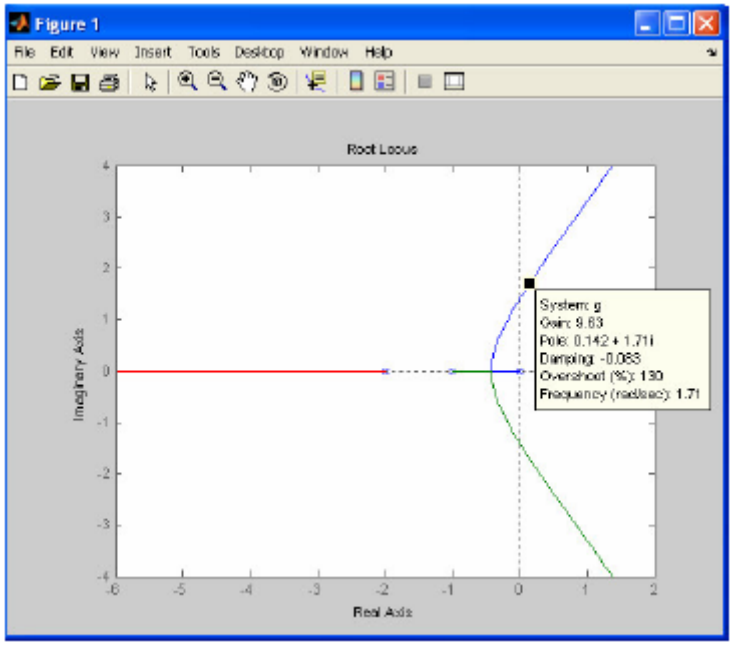
$(ii) Overshoot = 14.5\%$

To design the suitable controller, let's first assume simplicity and check the effect of simple gain adjustment using root locus for the whole system as follows:

On MATLAB type:

```
>> rlocus(G)
```

The following window will appear:



Dragging until we can adjust the gain to have overshoot less than 5%, the suitable value of the gain will be: **0.646**.

Now, check your results using MATLAB command window:

```
>> K=0.646;
>> new_sys=feedback(K*G,1);
>> step(new_sys,t)
```

From the step response figure, check the new values of the peak time and the overshoot:

$(i) t_p = 8.85 \sec$

$(ii) Overshoot = 3.99\%$

5.2. *Compensation using MATLAB GUI*

This is the general method of compensation in which we depend on defining the system transfer function then use MATLAB GUI (Graphical User Interface) to control the system.

sisotool(*System_transfer_function*)

Example:

Try to compensate the system of the given transfer function to achieve different requirements:

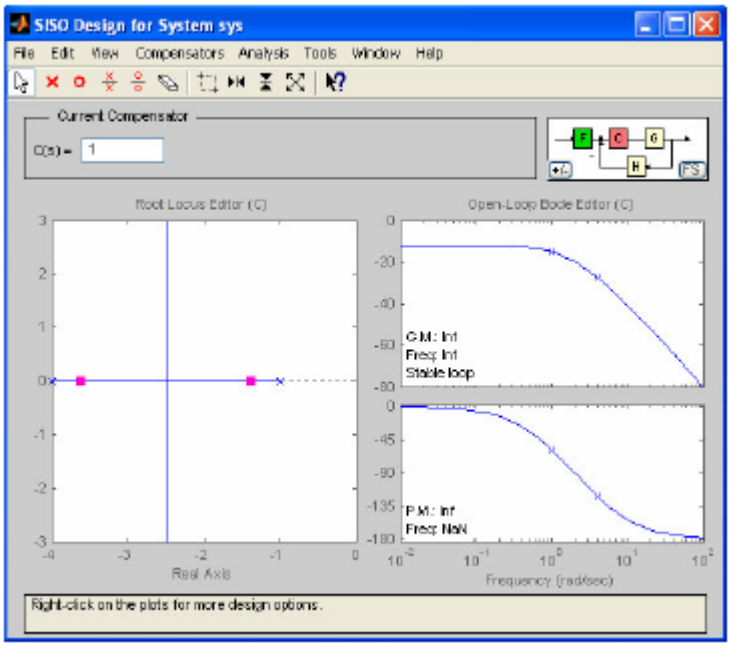
$$\frac{Y(S)}{U(S)} = \frac{1}{S^2 + 5S + 4}$$

Solution:

On the MATLAB command window, type:

```
>> sys=tf([1],[1 5 4])
>> sisotool(sys)
```

The following GUI will appear after a few seconds:



Note that this method is only valid for Single-Input Single-Output systems SISO, while Multi-Input Multi-Output systems MIMO are not valid. Now, you are able to design a compensator and place it where you want. Also it's possible to add poles or zeros and specify their locations, you can also adjust a gain for the compensator and check the effect of each modification on the bode plots (frequency response of the new system) and on the root locus plot.

Check each menu and try to full use this tool in your designs.

Finally, it's a wide tool that you can use to get any suitable kind of compensators to achieve any type of requirements.

- For more details on any fi\unction on the matlob, just write:

>> help *control*

>> help *function_name*