Chapter 8: Introduction to Systems Control

8.1 System Stability from Pole-Zero Locations (S-domain)

Case 1. Simple Poles with *L<N*: Consider an n-order transfer function in s-domain

$$H(s)=\frac{N(s)}{D(s)}=\frac{C_1}{s-p_1}+\frac{C_2}{s-p_2}+\cdots+\frac{C_N}{s-p_N}=\frac{C_1}{s-(s_1+jw_1)}+\cdots+\frac{C_N}{s-(s_N+jw_N)} \tag{8.1}$$

Rules:

a. If all p_i are real then it is obvious that: $w_1=w_2=\cdots=w_N\equiv 0$ and impulse response is simply a sum of real exponentials:

$$h(t)=(C_1e^{s_1t}+C_2e^{s_2t}+\cdots+C_Ne^{s_Nt}).u(t) \tag{8.2}$$

- If all $s_i<0$ then the roots (poles) are on negative real axis and each exponential term decays as *t* increases. Thus, the system is *unconditionally stable* for bounded inputs.
- If any $s_i>0$ then some of the poles are on the positive real-axis and as a result the term for this particular pole will diverge as t increases. The system will be *generally unstable* for any input. However, it is *unconditionally unstable for bounded inputs*.

b. If the system has complex roots then they must occur in complex conjugate pairs, i.e., if s_1+jw_1 is a root with a coefficient C_1, then s_1-jw_1 is also a root with constant C_1^*. For a simple pole-pair at location: $s_k=s_k\pm jw_k$ the impulse response due to these two terms will be:

$$\begin{aligned} h_k(t) &= C_k.e^{s_kt}.e^{jw_kt}.u(t)+C_k^*.e^{s_kt}.e^{-jw_kt}.u(t) \\ &= 2|C_k|.e^{s_kt}.Cos(w_kt+q_k).u(t) \end{aligned} \tag{8.3}$$

$$where\, q_k=a\tan\left(\frac{\mathrm{Im}\{C_k\}}{\mathrm{Re}\{C_k\}}\right)$$

- If $s_k<0$ then the system is *unconditionally stable* for bounded inputs since the pole-zero map is on the left half plane as before.

- If $s_k > 0$ then the system is *generally unstable* since the pole-zero map is sometimes on the right half plane and other times on the LHP.

Case 2. Simple Poles on jw-axis:

In this case, pole-zero map passes through jw-axis; at that instant $s_k = 0$ and the pair of roots are purely imaginary and hence, the system is *oscillatory* with an impulse response:

$$h_k(t) = 2|C_k|.Cos(w_k t + q_k) \tag{8.4}$$

Rules:

- For bounded input signals, the output signal will be bounded oscillatory or simply bounded. But if the input is also sinusoidal with the same w_k, the output will be of the form:

$$Y(s) = \frac{B.s}{(s^2 + w_k^2)^2} \quad \Leftrightarrow \quad y(t) = \frac{B}{2w_k}.t.Sin(w_k t) \tag{8.5}$$

which will *grow unboundedly* as *t* increases.

Case 3. Multiple-Order (Repeated) Poles in LHP of s-Plane:

Let us assume the order of the terms is *m*, then we will have an impulse response:

$$h_k(t) = |C_k|.t^{m-1}.e^{s_k t}.Cos(w_k t + q_k) \quad for \quad s_k < 0 \tag{8.6}$$

Rules:

Since $e^{s_k t}$ will decay faster than the term t^{m-1} increases and the response in (8.6) will be *BIBO stable* if the input is bounded.

Case 4. Multiple-Order (Repeated) Poles on jw-axis:

Again let us assume the order of the terms is *m*, then we will have an impulse response:

$$h_k(t) = |C_k|.t^{m-1}.Cos(w_k t + q_k) \quad for \quad s_k < 0 \tag{8.7}$$

Rules:

Here t^{m-1} increases as *t* grows and the system will be *unstable*.

Case 5. Multiple-Order (Repeated) Poles in RHP of s-Plane:

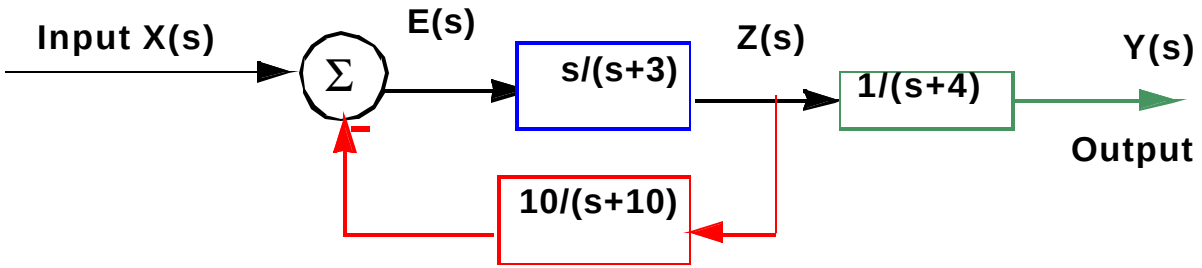
The response for these poles will be:

$$h_k(t) = |C_k|.t^{m-1}.e^{s_k t}.Cos(w_k t + q_k) \quad but \quad s_k > 0 \tag{8.8}$$

Rules:

Again, both $e^{s_k t}$ and t^{m-1} will grow as *t* increases. Expectedly, the system will be *unstable*.

Example 8.1: Determine if the following feedback system is stable.



The following set of equations governs this system.

$$E(S) = X(s) - Z(s).\frac{10}{s+10}; \qquad Z(s) = E(s).\frac{s}{s+3}; \qquad Y(s) = Z(s).\frac{1}{s+4}$$

The overall transfer function of this control system can be obtained as follows:

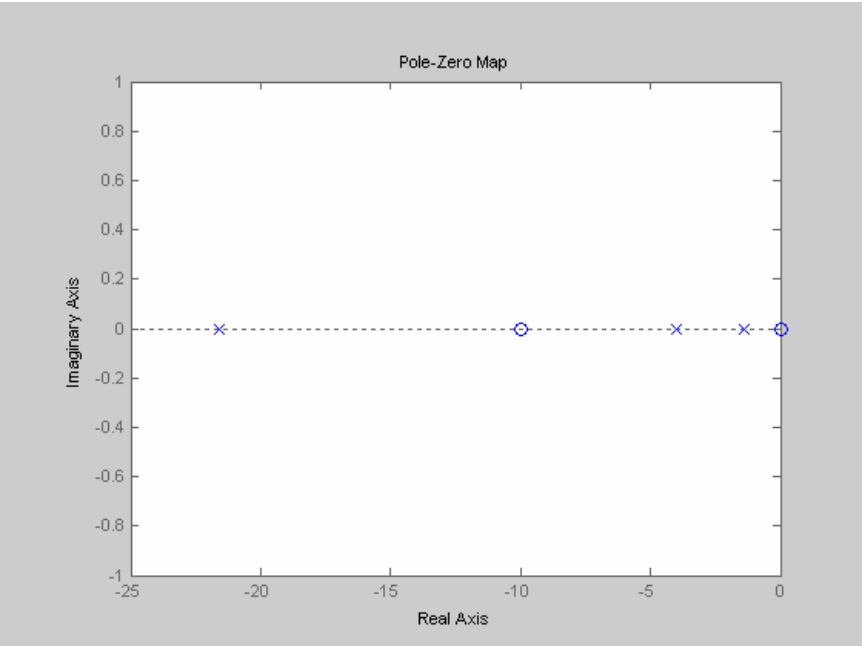
$$H_1(s) \equiv \frac{Z(s)}{X(s)} = \frac{s/(s+3)}{1 + \frac{s}{s+3}.\frac{10}{s+10}} = \frac{s^2 + 10s}{s^2 + 23s + 30}$$

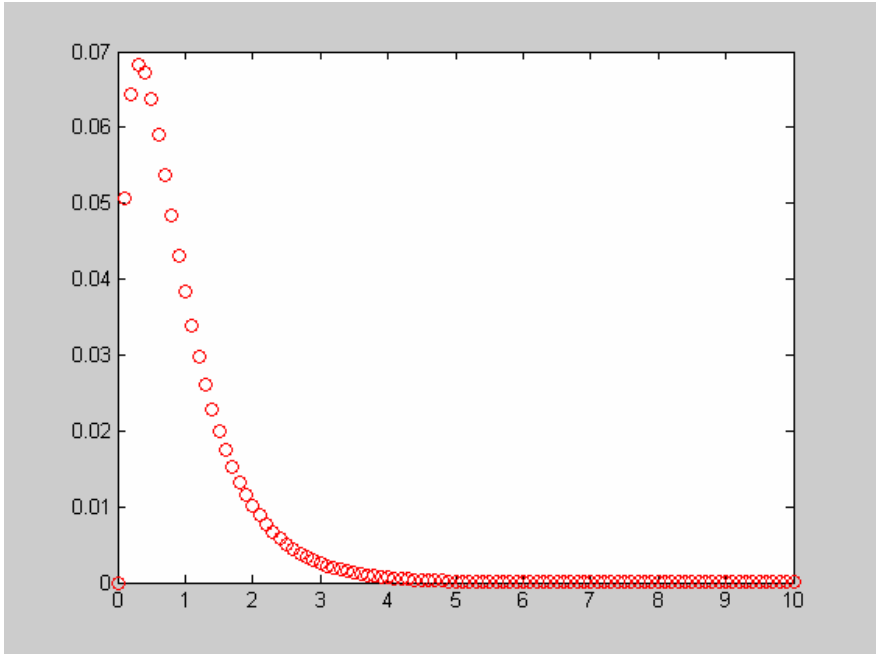
and

$$H(s) \equiv \frac{Y(s)}{X(s)} = \frac{Y(s)}{Z(s)} \cdot \frac{Z(s)}{X(s)} = \frac{1}{s+4} \cdot \frac{s^2+10s}{s^2+23s+30} = \frac{s^2+10s}{s^3+27s^2+122s+120}$$

As we have done in a few examples by now, the best way to finish this problem is to resort to Matlab tools.

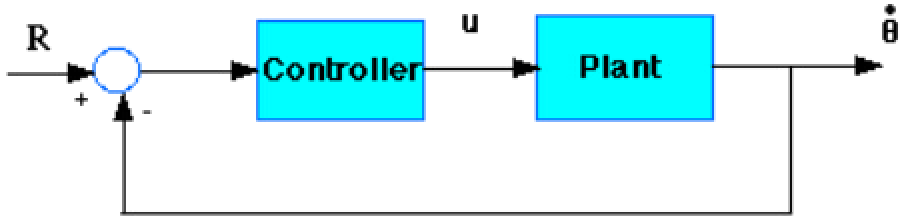
```
%EXAMPLE 8.1
   b=[1,10,0];
   a=[1,27,122,120];
 %pole-zero map                %Step-Input Response
   pzmap(b,a);                    time=0:0.1:10;
   figure;                        response=step(b,a,time);
                                  plot(time,response,'rO');  axis;
```





As we can see observe from the above pole-zero map, both zeros and poles are all real with z_1=0; z_2=-10, p_1=-1.388; p_2=-4; and p_3=-21,6. Therefore, the system is stable for bounded inputs, which is clearly demonstrated in the step-response plot.

Example 8.1: The system block diagram and the dynamic equations and the open-loop transfer function of DC Motor Speed are:



$$s(Js+b).\Theta(s) = K.I(s); \qquad (Ls+R).I(s) = V - K.s.\Theta(s)$$

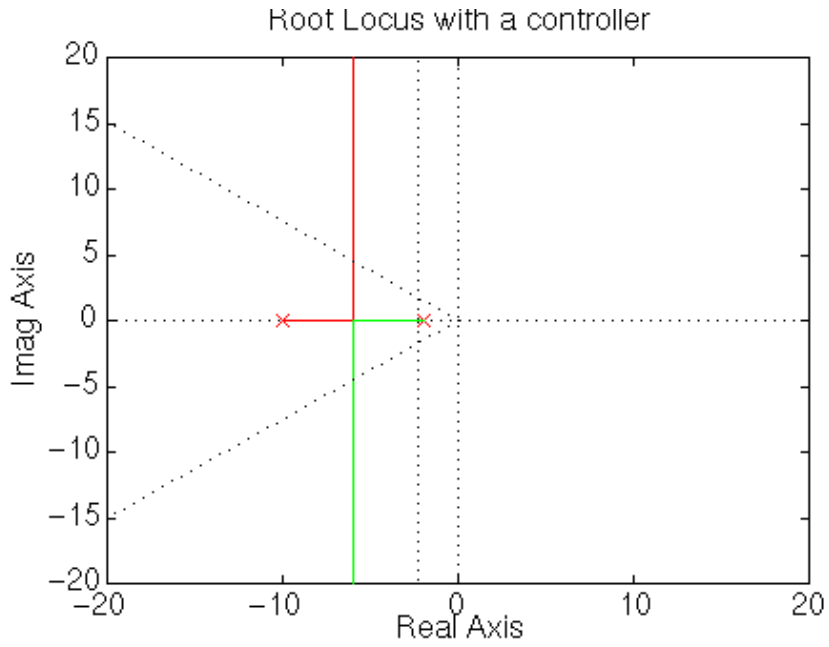
$$\frac{q'}{V} = \frac{K}{(Js+b).(Ls+R)+K^2} \tag{8.9}$$

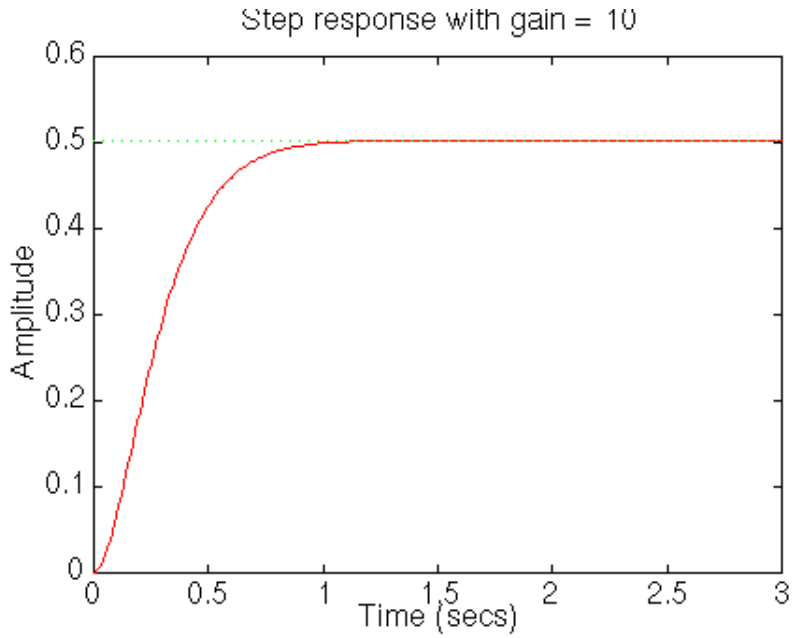
With a 1 rad/sec step reference, the design criteria are:

- Settling time less than 2 seconds
- Overshoot less than 5%
- Steady-state error less than 1%

Now let's design a controller using the **root locus** approach. The main idea of root locus design is to find the closed-loop response from the open-loop root locus plot. Then by adding zeros and/or poles to the original plant, the closed-loop response can be modified to improve stability or for other tasks.

```
% Root-Locus plot for motor speed
    J=0.01;  b=0.1;  K=0.01;  R=1;  L=0.5;
    num=K;  den=[(J*L) ((J*R)+(L*b)) ((b*R)+K^2)];
    sgrid(.8,0); sigrid(2.3)                    %User defined function to obtain coordinates of a point via hair-line
    rlocus(num,den)
        title('Root Locus without a controller')
```





Finding the appropriate gain using the rlocfind command:

In order a DC motor run as desired, we need the settling time and the overshoot to be as small as possible. Large damping corresponds to points on the root locus near the real axis. A fast response corresponds to points on the root locus far to the left of the imaginary axis. To find the gain corresponding to a point on the root locus, we can use the rlocfind command. We can find the gain and plot the step response using this gain all at once.

```
[k,poles] = rlocfind(num,den)
[numc,denc]=cloop(k*num,den,-1);
t=0:0.01:3;
step(numc,denc,t)
title('Step response with gain')
```

As you can see, the system is overdamped and the settling time is about one second, so the overshoot and settling time requirements are satisfied. The only problem we can see from this plot is the steady- state error

of about 50%. If we increase the gain to reduce the steady-state error, the overshoot becomes too large (Try this yourself). We need to add a lag controller to reduce the steady-state error.

Adding a lag controller

From the plot we see that this is a very simple root locus. The damping and settling time criteria were met with the proportional controller. The steady-state error is the only criterion not met with the proportional controller. A lag compensator can reduce the steady-state error. By doing this, we might however increase our settling time. Try the following lag controller first:

$$(s+1)/(s+0.1) \qquad (8.10)$$

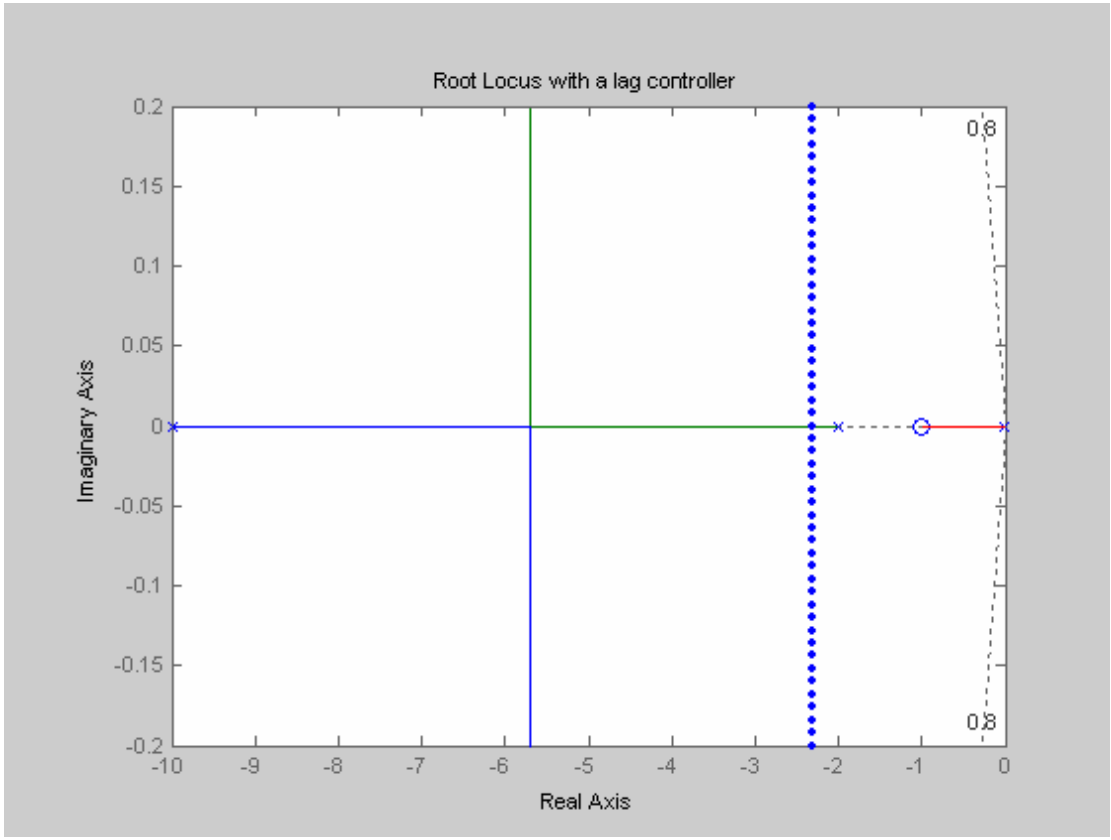
This can be done by changing the m-file to look like the following:

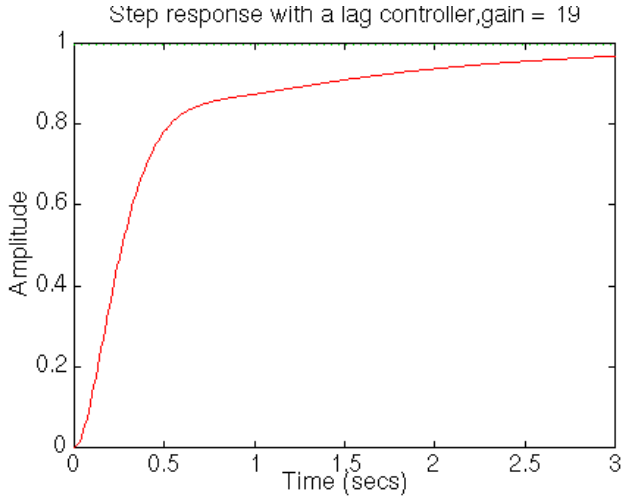
```
J=0.01; b=0.1; K=0.01;  R=1;  L=0.5;
num=K;
den=[(J*L) ((J*R)+(L*b)) ((b*R)+K^2)];

z1=1;  p1=0.01;
numa = [1 z1];  dena = [1 p1];          % numerator and denominator of the controller
numb=conv(num,numa);                    % the numerator and denominator of the overall open-loop transfer function

rlocus(numb,denb)
sgrid(.8,0)
sigrid(2.3)
title('Root Locus with a lag controller')
```

We get the following root locus, which looks very similar to the original one. Now let's close the loop and see the closed-loop step response by entering the following code at the end of our m-file:

```
[k,poles]=rlocfind(numb,denb)
[numc,denc]=cloop(k*numb,denb,-1);
t=0:0.01:3;
step(numc,denc,t)
title('Step response with a lag controller')
```

Rerun this m-file in the Matlab command window. When prompted to select a point, pick one that is near the damping requirement (diagonal dotted line). You should get the a plot similar to the following:


Root Locus with a lag controller
Imaginary Axis
Real Axis
0.8
0.8
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
0

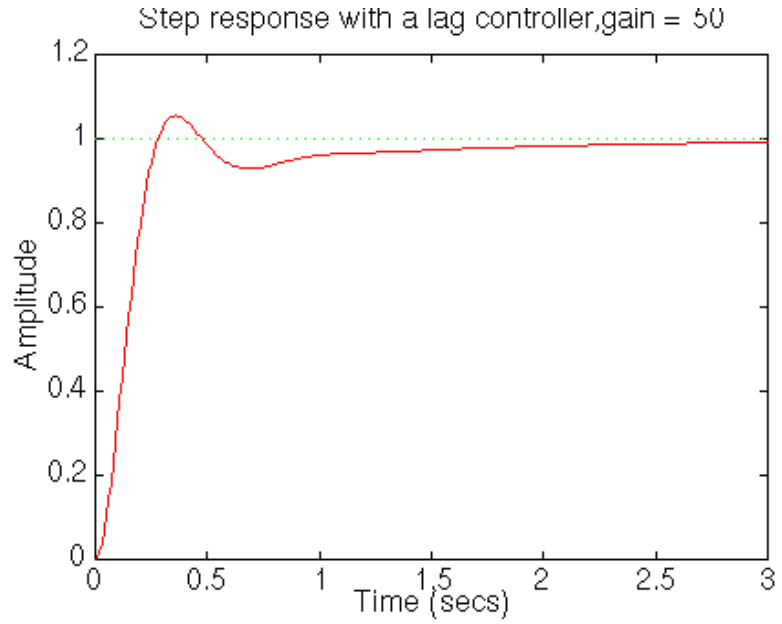

Step response with a lag controller,gain = 19
Amplitude
Time (secs)
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3

8.8

Our gain should be about 20. As you can see the response is not quite satisfactory.

You may also note that even though the gain was selected to correlate with a position close to the damping criterion, the overshoot is not even close to five percent. This is due to the effect of the lag controller kicking in at a later time than the plant. (its pole is slower). What this means is that we can go beyond the dotted lines that represent the limit, and get the higher gains without worrying about the overshoot

Now we rerunI our m-file, place the gain just above the white, dotted line. Keep trying until you get a satisfactory response. It should look similar to the following (we used a gain of around 50):

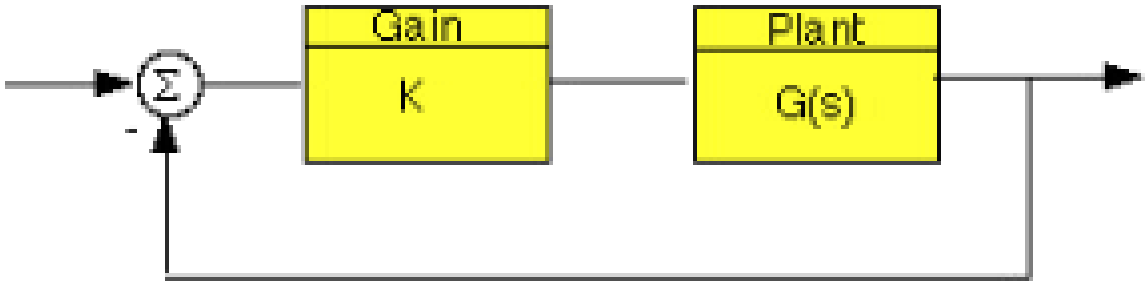


The steady-state error is smaller than 1%, and the settling time and overshoot requirements have been met. As you can see, the design process for root locus is very much a trial and error process. That is why it is nice to plot the root locus, pick the gain, and plot the response all in one step. If we had not been able to get a satisfactory response by choosing the gains, we could have tried a different lag controller, or even added a lead controller.

8.2 System Analysis & design based on Frequency Response

The frequency response is a representation of the system's response to sinusoidal inputs at varying frequencies. The output of a linear system to a sinusoidal input is a sinusoid of the same frequency but with a different magnitude and phase. The ***frequency response*** is defined as the magnitude and phase differences between the input and output sinusoids. Let us see how we can use the open-loop frequency response of a system to predict its behavior in closed-loop.

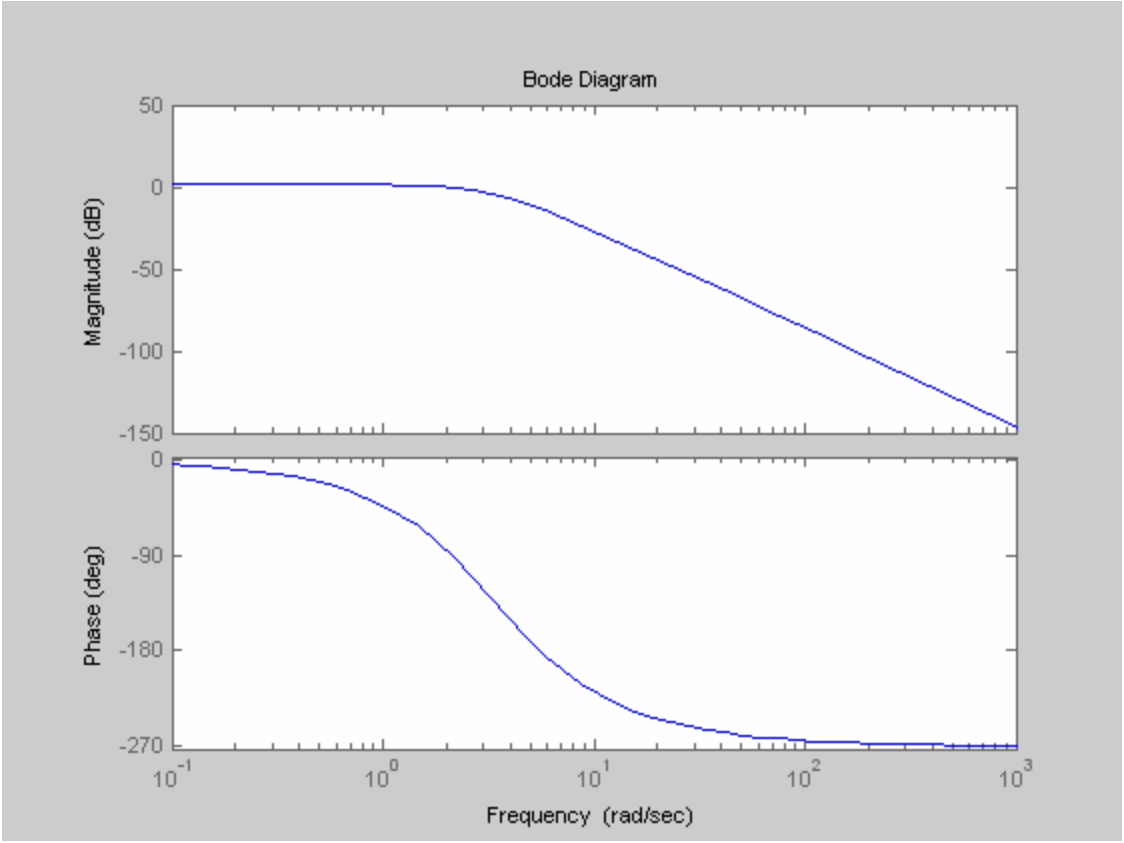
To plot the frequency response, we create a vector of frequencies (varying between zero or "DC" and infinity) and compute the value of the plant transfer function at those frequencies. If $G(s)$ is the open loop transfer function of a system and *w* is the frequency vector, we then plot $G(jw)$ vs. w. Since $G(jw)$ is a complex number, we can plot both its magnitude and phase (the Bode plot) or its position in the complex plane (the Nyquist plot).



Example: 8.3 Bode Plots: Matlab has a tool for plotting Bode plots, where the frequency is on a logarithmic scale, the phase is given in degrees, and the magnitude is given as the gain in decibels.

$$H(s) = \frac{50}{s^3 + 9s^2 + 30s + 40}$$

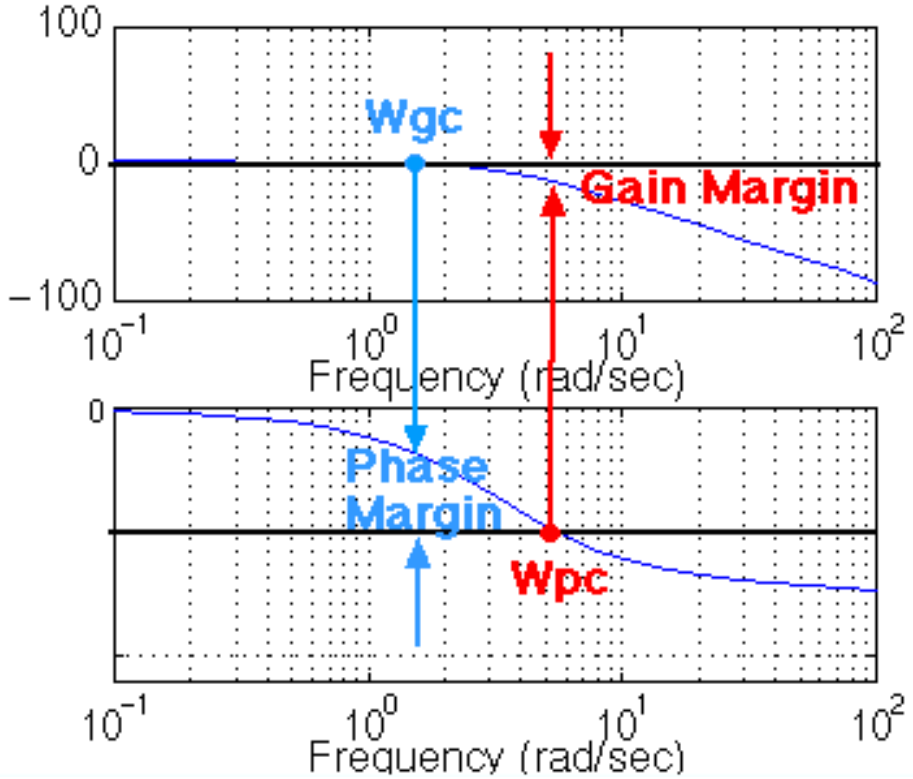
```
num=[50]
denom=[1 9 30 40]
bode(num,denom)
```



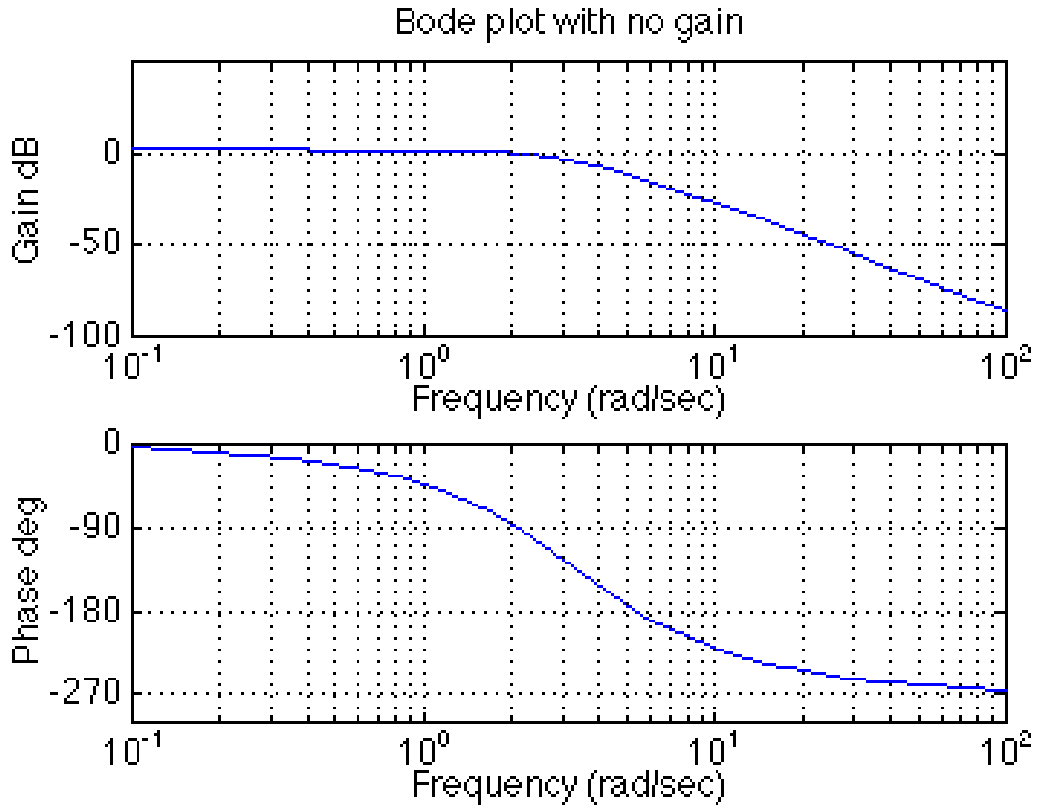
Gain and Phase Margin:

- The gain margin is defined as the change in open loop gain required to make the system unstable. Systems with greater gain margins can withstand greater changes in system parameters before becoming unstable in closed loop.
- The phase margin is defined as the change in open loop phase shift required to make a closed loop system unstable. The phase margin also measures the system's tolerance to time delay. If there is a time delay greater than 180/*Wpc* in the loop (where *Wpc* is the frequency where the phase shift is 180 deg), the system will become unstable in closed loop.

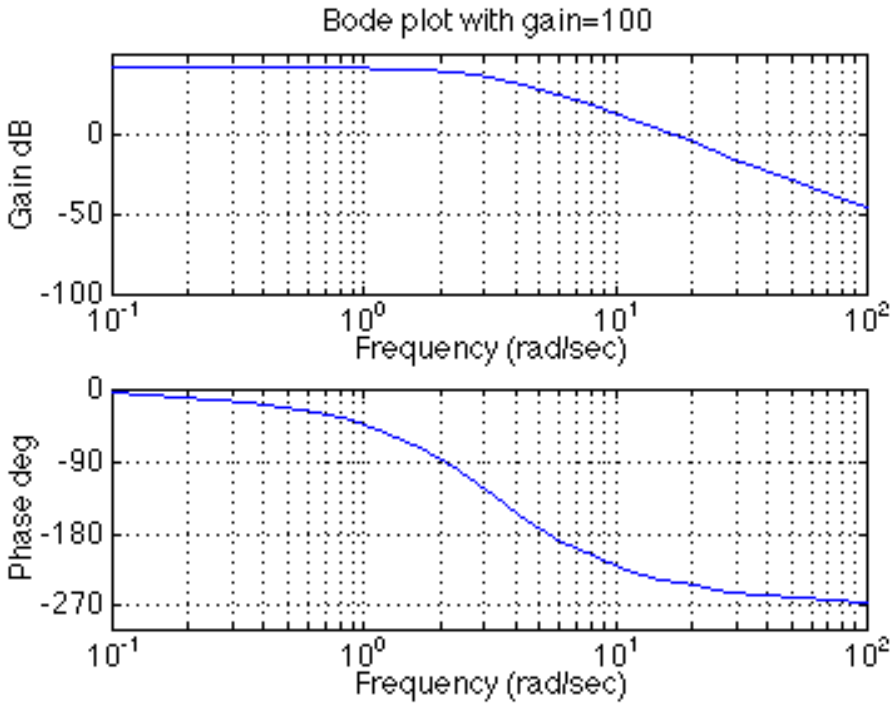
- The time delay can be thought of as an extra block in the forward path of the block diagram that adds phase to the system but has no effect the gain. That is, a time delay can be represented as a block with magnitude of 1 and phase w*time_delay (in radians/second).



One nice thing about the phase margin is that you don't need to replot the Bode in order to find the new phase margin when changing the gains. If you recall, adding gain only shifts the magnitude plot up. This is the equivalent of changing the y-axis on the magnitude plot. Finding the phase margin is simply the matter of finding the new cross-over frequency and reading off the phase margin. For example, suppose we use the above case bode(50,[1 9 30 40]). We will get the following bode plot:



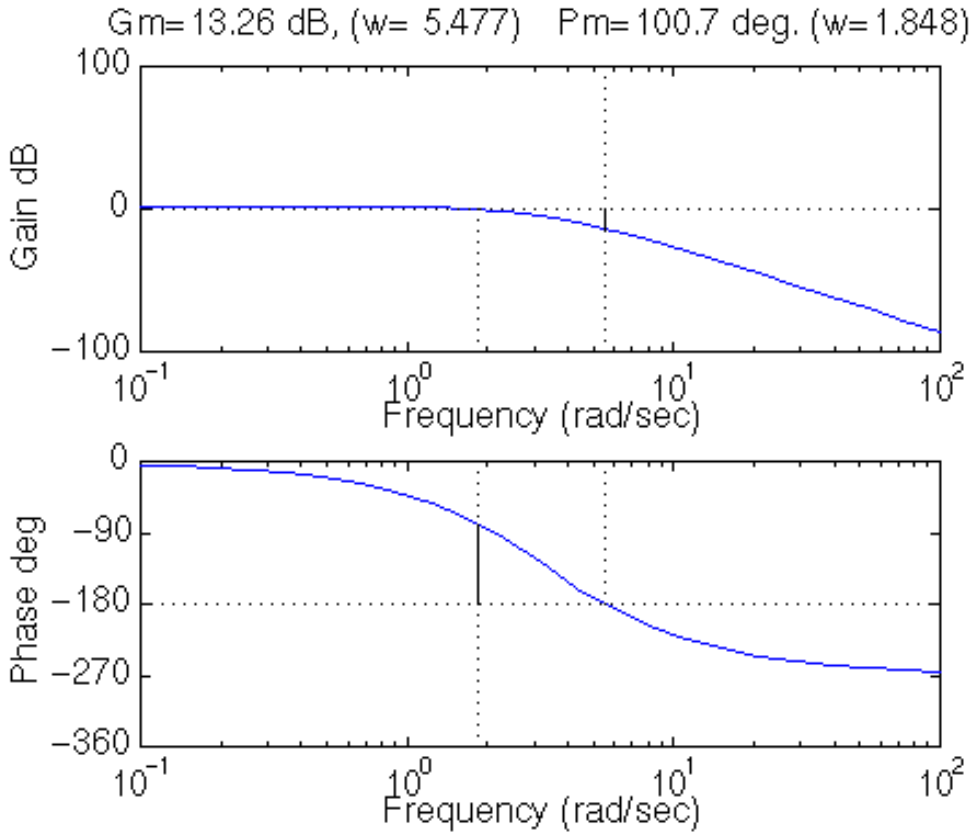
- You should see that the phase margin is about 100 degrees.
- Suppose we add a gain of 100, by entering the command bode(100*50,[1 9 30 40]). We should get the following plot (note I changed the axis so the scale would be the same as the plot above, your bode plot may not be exactly the same shape, depending on the scale used):



As you can see the phase plot is exactly the same as before, and the magnitude plot is shifted up by 40dB (gain of 100). The phase margin is now about -60 degrees. This same result could be achieved if the y-axis of the magnitude plot was shifted down 40dB. Try this, look at the first Bode plot, find where the curve crosses the -40dB line, and read off the phase margin. It should be about -60 degrees, the same as the second Bode plot.

We can find the gain and phase margins for a system directly, by using Matlab. The command margin returns the gain and phase margins, the gain and phase cross over frequencies, and a graphical representation of these on the Bode plot. In our problem:

```
margin(50,[1 9 30 40])
```



Bandwidth Frequency

The bandwidth frequency is defined as the frequency at which the **closed-loop** magnitude response is equal to -3 dB. However, when we design via frequency response, we are interested in predicting the closed-loop behavior from the open-loop response. Therefore, we will use a second-order system approximation and say that the bandwidth frequency equals the frequency at which the **open-loop** magnitude response is between -6 and - 7.5dB, assuming the open loop phase response is between -135 deg and -225 deg.

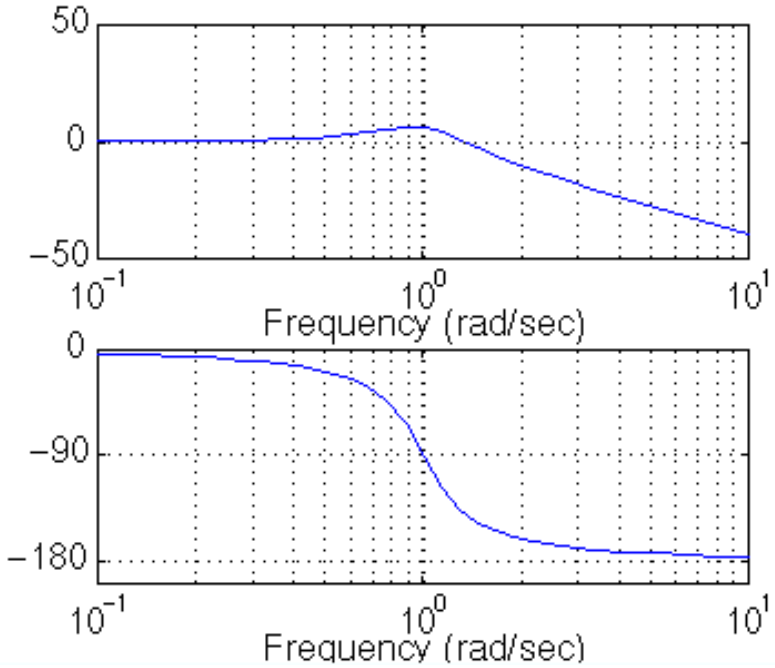
In order to illustrate the importance of the bandwidth frequency, we will show how the output changes with different input frequencies. We will find that sinusoidal inputs with frequency less than *Wbw* (the bandwidth frequency) are tracked "reasonably well" by the system. Sinusoidal inputs with frequency greater than *Wbw* are attenuated (in magnitude) by a factor of 0.707 or greater (and are also shifted in phase).

Example: 8.3 Bandwidth from Bode Plots*:* Consider a second order system transfer function:

$$H(s) = \frac{1}{s^2 + 0.5 * s + 1}$$

Bandwidth frequency by looking at the Bode plot:

```
bode (1, [1 0.5 1 ])
```



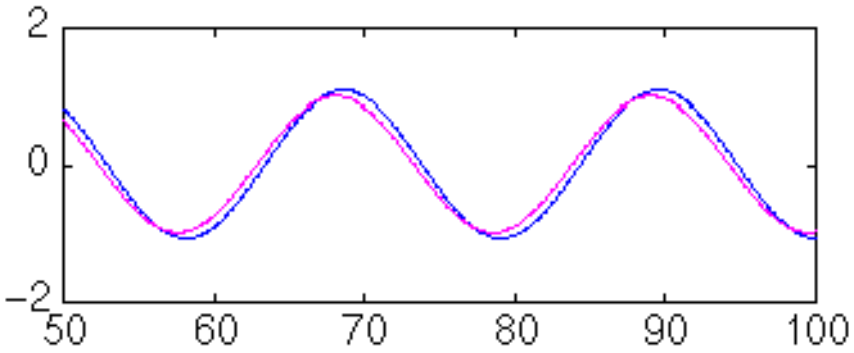
Since this is the closed-loop transfer function, our bandwidth frequency will be the frequency corresponding to a gain of -3 dB.

- Looking at the plot, we find that it is approximately 1.4 rad/s.
- We can also read off the plot that for an input frequency of 0.3 radians, the output sinusoid should have a magnitude about one and the phase should be shifted by perhaps a few degrees (behind the input).
- For an input frequency of 3 rad/sec, the output magnitude should be about -20dB (or 1/10 as large as the input) and the phase should be nearly -180 (almost exactly out-of-phase).

- We can use the lsim command to simulate the response of the system to sinusoidal inputs.

1. Consider a sinusoidal input with a frequency lower than Wbw. We must also keep in mind that we want to view the steady state response. Therefore, we will modify the axes in order to see the steady state response clearly (ignoring the transient response).

```
w= 0.3;
num = 1;   den = [1 0.5 1 ];
t=0:0.1:100;
u = sin(w*t);
[y,x] = lsim(num,den,u,t);
plot(t,y,t,u); axis([50,100,-2,2])
```

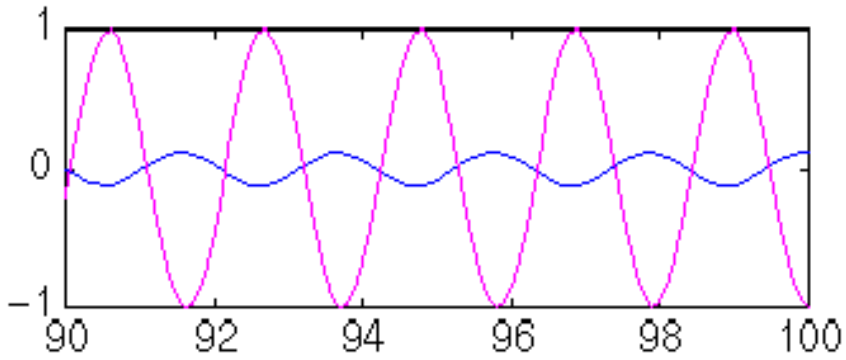


Note that the output (blue) tracks the input (purple) fairly well; it is perhaps a few degrees behind the input as expected.

2. If we set the frequency of the input **higher than the bandwidth frequency** for the system, we get a very distorted response (with respect to the input):

```
 w = 3;
num = 1;  den = [1 0.5 1 ];
t=0:0.1:100;
u = sin(w*t);
```

```
[y,x] = lsim(num,den,u,t);
plot(t,y,t,u);   axis([90, 100, -1, 1])
```



Note that the magnitude is about 1/10 that of the input, as predicted, and that it is almost exactly out of phase (180 degrees behind) the input.

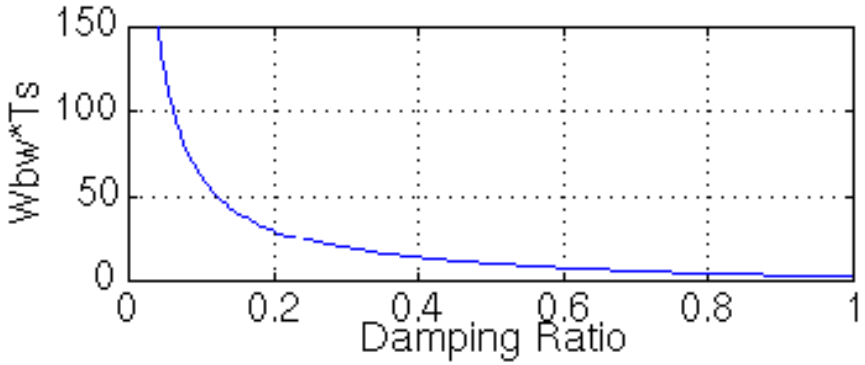
Closed-loop performance

In order to predict closed-loop performance from open-loop frequency response, we need to have several concepts clear:

1. The system must be stable in open loop if we are going to design via Bode plots.
2. If the gain cross over frequency is less than the phase cross over frequency(i.e. Wgc < Wpc), then the closed-loop system will be stable.
3. For second-order systems, the closed-loop damping ratio is approximately equal to the phase margin divided by 100 if the phase margin is between 0 and 60 deg. We can use this concept with caution if the phase margin is greater than 60 deg.
4. For second-order systems, a relationship between damping ratio, bandwidth frequency and **settling time** is given by:

$$\omega_{BW} = \omega_n \sqrt{\left(1-2\zeta^2\right)+\sqrt{\zeta^4-4\zeta^2+2}} \qquad \omega_n = \frac{4}{T_s \zeta} \qquad \omega_{BW} = \frac{4}{T_s \zeta^2}\sqrt{\left(1-2\zeta^2\right)+\sqrt{\zeta^4-4\zeta^2+2}}$$

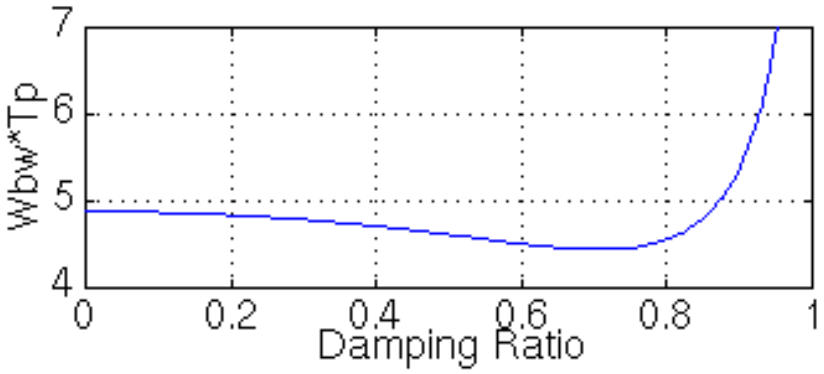
In order to make our life easier, we usually plot these equations instead of plugging in the values:



Similarly, a relationship between damping ratio, bandwidth frequency and **rise time** is given by:

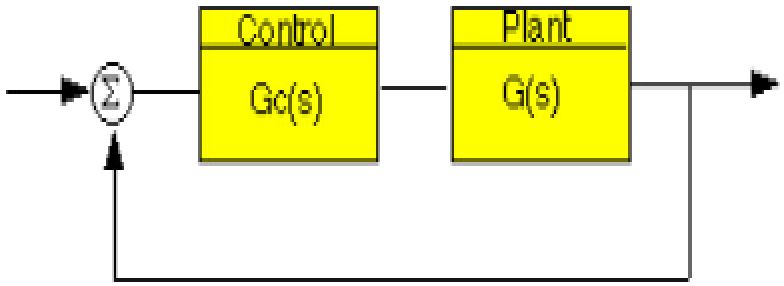
$$\omega_n = \frac{\pi}{T_P\sqrt{1-\zeta^2}} \qquad \omega_{BW} = \frac{\pi}{T_P\sqrt{1-\zeta^2}}\sqrt{\left(1-2\zeta^2\right)+\sqrt{\zeta^4-4\zeta^2+2}}$$

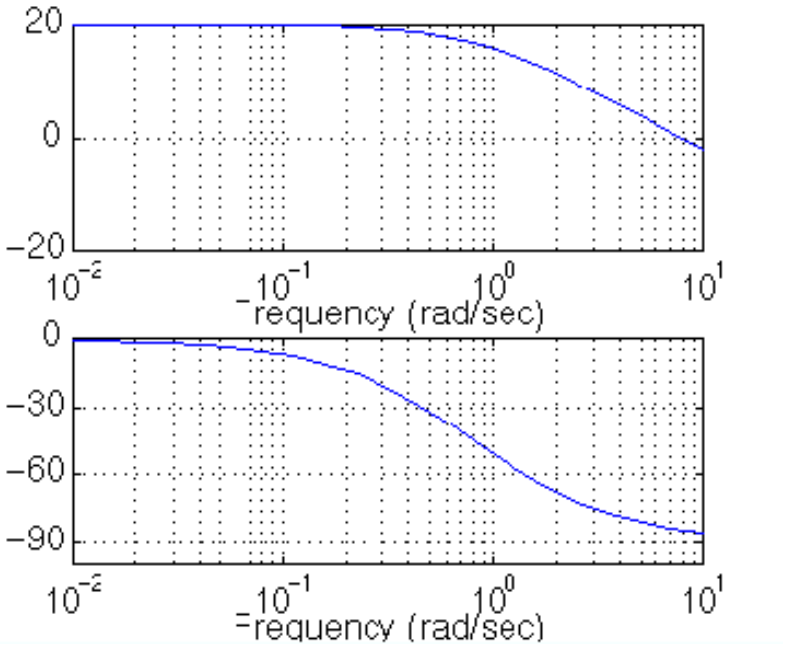
Again, we usually look at the graph instead of plugging in the values.



However, a very rough estimate that you can use is that the bandwidth is approximately equal to the natural frequency.

Example: 8.4 Let's use these concepts to design a controller for the following system:





$$G(s) = \frac{10}{1.25 * s + 1}$$

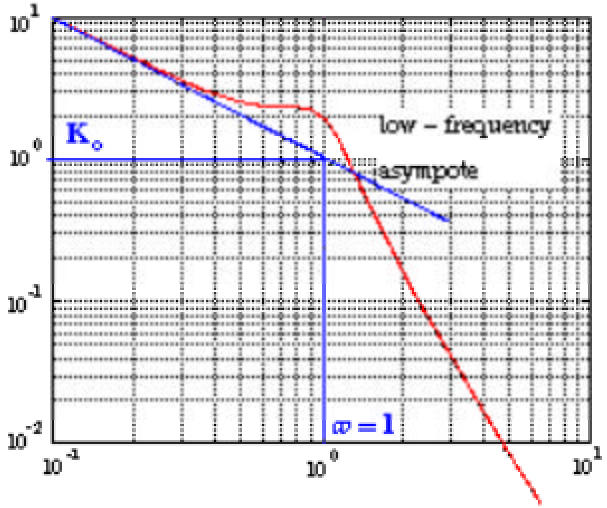
The design must meet the following specifications: (1) zero steady state error, (2) maximum overshoot must be less than 40% and (3) settling time must be less than 2 seconds.

There are two ways of solving this problem: one is graphical and the other is numerical. Within Matlab, the graphical approach is best, so that is the approach we will use. First, let's look at the Bode plot. Create an m-file with the following code:

```
num = 10;
den = [1.25,1];
bode(num, den)
```

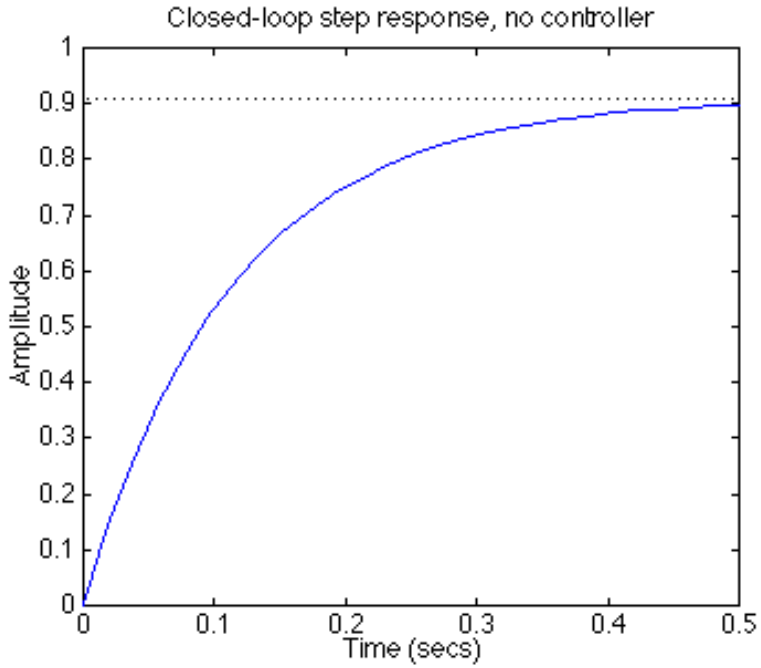
Observations: There are several several characteristics of the system that can be read directly from this Bode plot.

- First of all, we can see that the bandwidth frequency is around 10 rad/sec.
- Since the bandwidth frequency is roughly the same as the natural frequency, the rise time is 1.8/BW=1.8/10=1.8 seconds. This is a rough estimate, so we can say the rise time is about 2 seconds.
- The phase margin for this system is approximately 95 degrees. This corresponds to a damping of PM/100=95/100=0.95. Plugging in this value into the equation relating overshoot and the damping ratio (or consulting a plot of this relation), we find that the damping ratio corresponding to this overshoot is approximately 1%. The system will be close to being overdamped.
- The steady-state error can be read directly off the Bode plot as well. The constant (Kp, Kv, or Ka) are located at the intersection of the low frequency asymptote with the w=1 line. (Just extend the low frequency line to the w=1 line.) The magnitude at this point is the constant.
- Since the Bode plot of this system is a horizontal line at low frequencies (slope = 0), we know this system is of type zero. Therefore, the intersection is easy to find. The gain is 20dB (magnitude 10). What this means is that the constant for the error function is 10.
- The steady-state error is 1/(1+Kp)=1/(1+10)=0.091. If our system was type one instead of type zero, the constant for the steady-state error would be found in a manner similar to the following



Let's check our predictions by looking at a step response plot. This can be done by adding the following two lines of code into the Matlab command window.

```
[numc,denc] = cloop(num,den,-1);
step(numc,denc)
```



Observations:

- Our predictions were very good.
- The system has a rise time of about 2 s, is overdamped, and has a steady-state error of about 9%.
- Now we need to choose a controller that will allow us to meet the design criteria.
- We choose a PI controller because it will yield zero steady state error for a step input.

$$G_C(s) = \frac{K * (s + a)}{s}$$

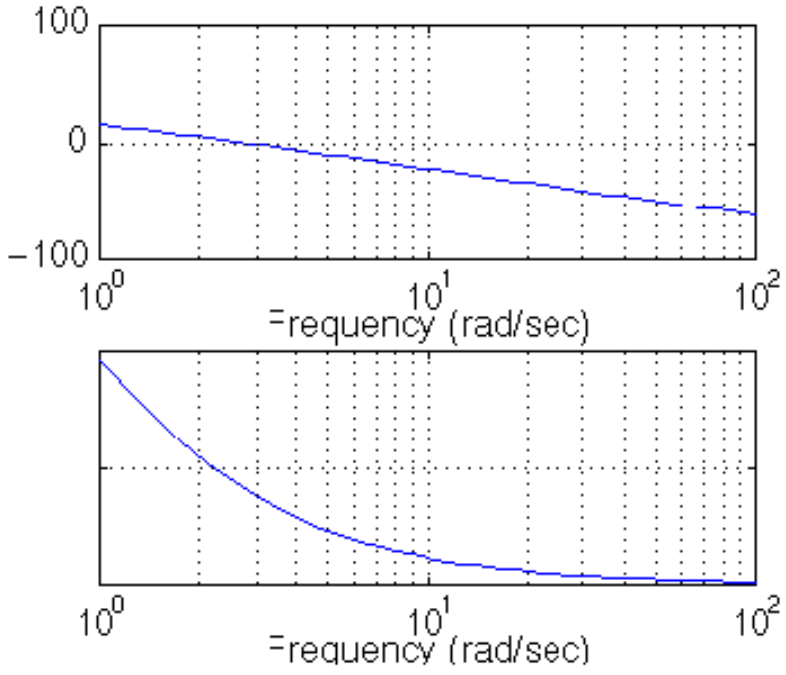
PI controller has a zero at a, which we can choose. This gives us additional design flexibility to help us meet our criteria.

1. The first thing we need to find is the damping ratio corresponding to a percent overshoot of 40%:

Plugging in this value into the equation relating overshoot and damping ratio (or consulting a plot of this relation), we find that the damping ratio corresponding to this overshoot is approximately 0.28. Therefore, our phase margin should be approximately 30 degrees. From our Ts*Wbw vs damping ratio plot, we find that Ts*Wbw ~ 21. We must have a bandwidth frequency greater than or equal to 12 if we want our settling time to be less than 1.75 seconds which meets the design specs.

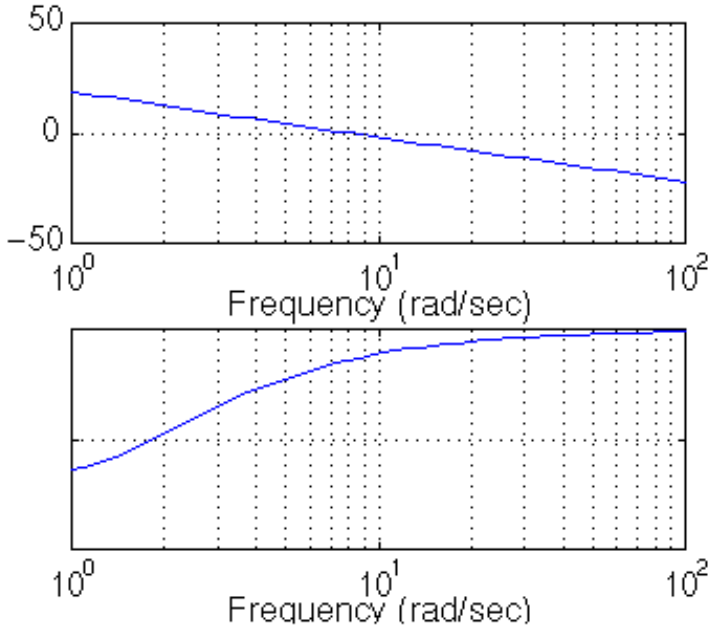
2. Now that we know our desired phase margin and bandwidth frequency, we can start our design. Remember that we are looking at the open-loop Bode plots. Therefore, our bandwidth frequency will be the frequency corresponding to a gain of approximately -7 dB.

```
num = [10];
den = [1.25, 1];
numPI = [1];
denPI = [1 0];
newnum = conv(num,numPI);
newden = conv(den,denPI);
bode(newnum, newden, logspace(0,2))
```



Our phase margin and bandwidth frequency are too small. We will add gain and phase with a zero. Let's place the zero at 1 for now and see what happens. Change your m-file to look like the following:

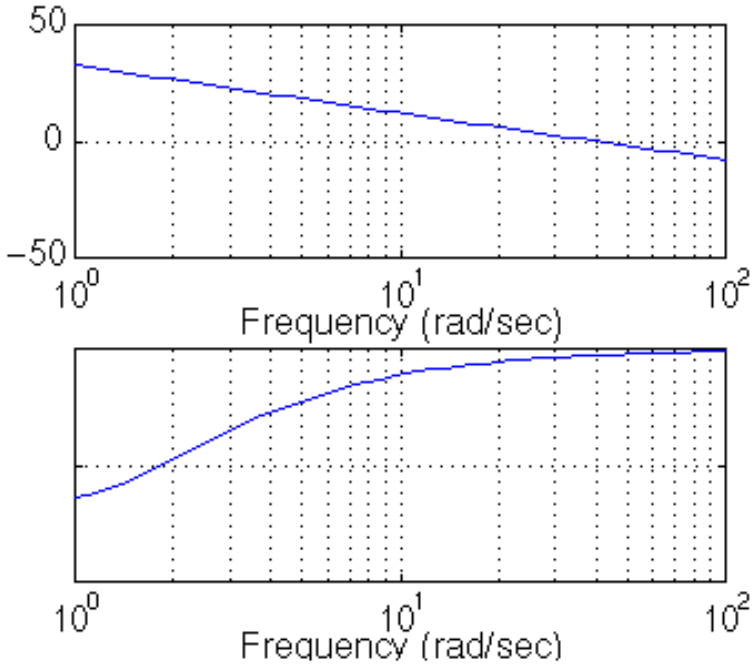
```
num = [10];
den = [1.25, 1];
numPI = [1 1];
denPI = [1 0];
newnum = conv(num,numPI);
newden = conv(den,denPI);
bode(newnum, newden, logspace(0,2))
```



- The zero at 1 with a unit gain gives us a satisfactory answer.
- Phase margin is greater than 60 degrees (even less overshoot than expected) and
- Bandwidth frequency is approximately 11 rad/s, which will give us a satisfactory response.
- Although satisfactory, the response is not quite as good as we would like.

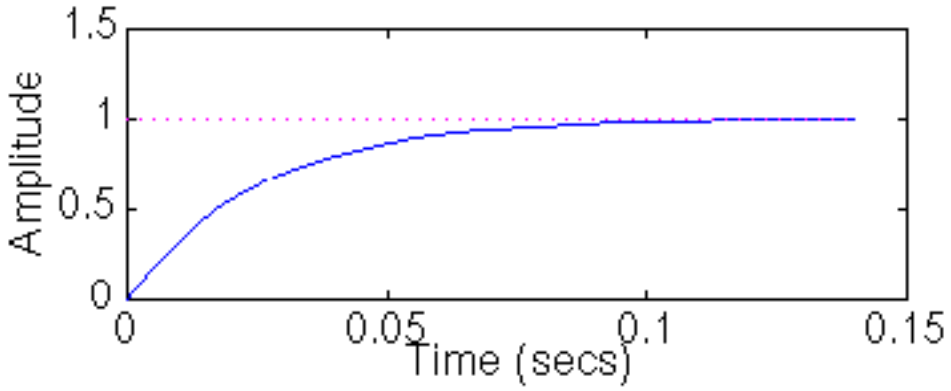
- Now let's try to get a higher bandwidth frequency without changing the phase margin too much. Let's try to increase the gain to 5 and see what happens .This will make the gain shift and the phase will remain the same.

```
num = [10];
den = [1.25, 1];
numPI = 5*[1 1];
denPI = [1 0];
newnum = conv(num,numPI);
newden = conv(den,denPI);
bode(newnum, newden, logspace(0,2))
```



That looks really good. Let's look at our step response and verify our results with the following lines in the code:

```
[clnum,clden] =cloop(newnum,newden,-1);
step(clnum,clden)
```



As you can see, our response is better than we had hoped for. However, we are not always quite as lucky and usually have to play around with the gain and the position of the poles and/or zeros in order to achieve our design requirements.

8.3 System Analysis & design based on Nyquist Diagrams

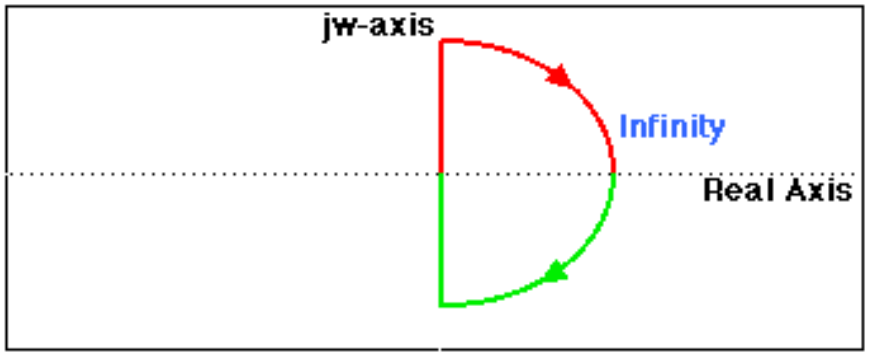
The Nyquist plot allows us to predict the stability and performance of a closed-loop system by observing its open-loop behavior. The Nyquist criterion can be used for design purposes regardless of open-loop stability (remember that the Bode design methods assume that the system is stable in open loop). Therefore, we use this criterion to determine closed-loop stability when the Bode plots display confusing information.

Note: The Matlab nyquist command does not provide an adequate representation for systems that have open-loop poles in the *jw*-axis. Therefore, we suggest that you copy the nyquist1.m file as a new m-file. This m-file creates more accurate Nyquist plots, since it take into account poles and zeros on the *jw*-axis.

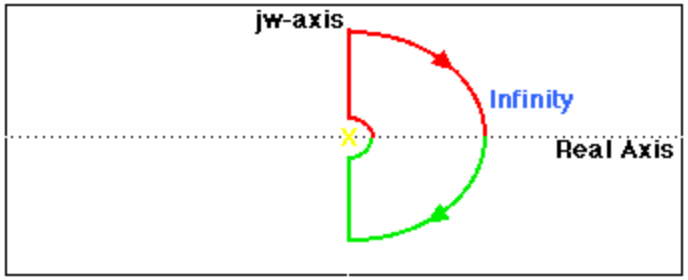
The Nyquist diagram is basically a plot of $G(jw)$, where $G(s)$ is the open-loop transfer function and w is a vector of frequencies which encloses the entire right-half plane.

- In drawing the Nyquist diagram, both positive and negative frequencies (from zero to infinity) are taken into account. We will represent positive frequencies in red and negative frequencies in green.

- The frequency vector used in plotting the Nyquist diagram usually looks like this (if you can imagine the plot stretching out to infinity):



- If we have open-loop poles or zeros on the *jw*-axis, $G(s)$ will not be defined at those points, and we must loop around them when we are plotting the contour. Such a contour would look as follows:



- Note that the contour loops around the pole on the jw axis. As we mentioned before, the Matlab nyquist command does not take poles or zeros on the jw axis into account and therefore produces an incorrect plot. Therefore, If we have a pole on the jw axis, we have to use nyquist1. If there are no poles or zeros on the jw-axis, or if we have pole-zero cancellation, we can use either the nyquist command or nyquist1.m.

The Cauchy criterion

The Cauchy criterion (from complex analysis) states that when taking a closed contour in the complex plane, and mapping it through a complex function $G(s)$, the number of times that the plot of $G(s)$ encircles the origin is equal to:

1. Number of zeros of $G(s)$ enclosed by the frequency contour
2. Minus the number of poles of $G(s)$ enclosed by the frequency contour.
3. Encirclements of the origin are counted as positive if they are in the same direction as the original closed contour or negative if they are in the opposite direction.

In studying feedback controls, we are not interested in $G(s)$ as in the closed-loop transfer function: $\frac{G(s)}{1+G(s)}$

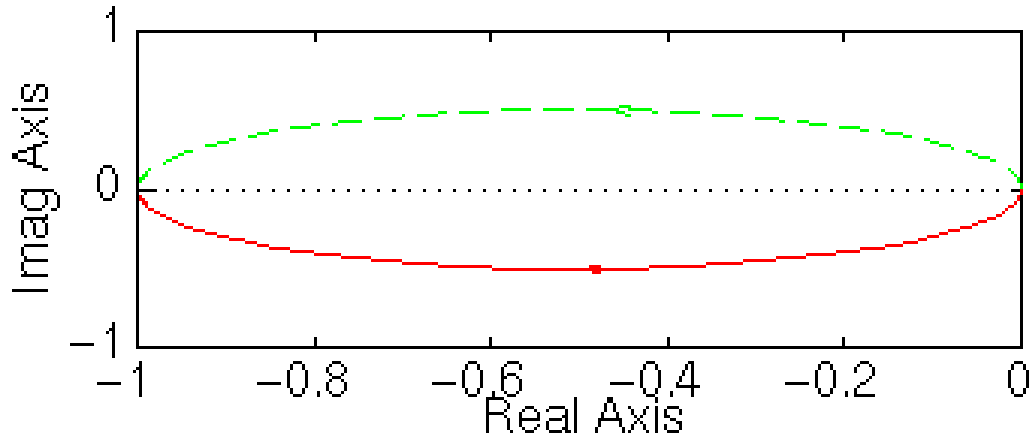
If $1+G(s)$ encircles the origin, then $G(s)$ will enclose the point -1. Since we are interested in the closed-loop stability, we want to know if there are any closed-loop poles (zeros of $1+G(s)$) in the right-half plane.

Therefore, the lnyquist1.m command plots the Nyquist diagram using a logarithmic scale and preserves the characteristics of the -1 point.

Example: 8.5 Consider the following transfer functions:

Case 1: $G(s)=\frac{0.5}{s-0.5}$

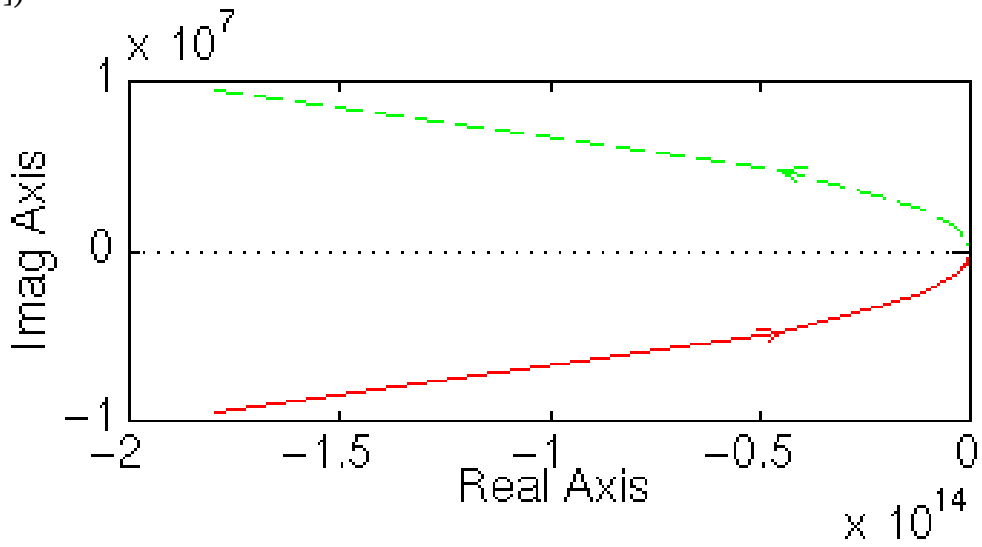
```
nyquist (0.5,[1 -0.5])
```



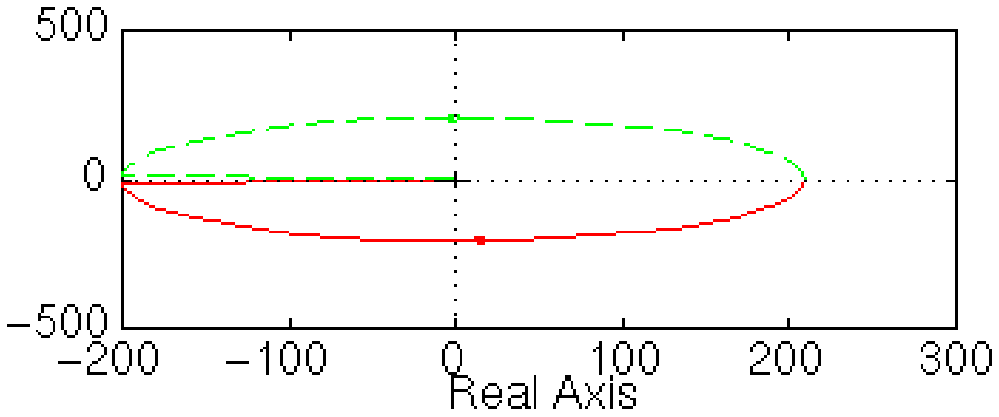
Case 2: $G(s) = \dfrac{s+2}{s^2}$

Note: this function has a pole at the origin. Let us see the difference between using the nyquist, nyquist1, and lnyquist1 commands with this particular function.

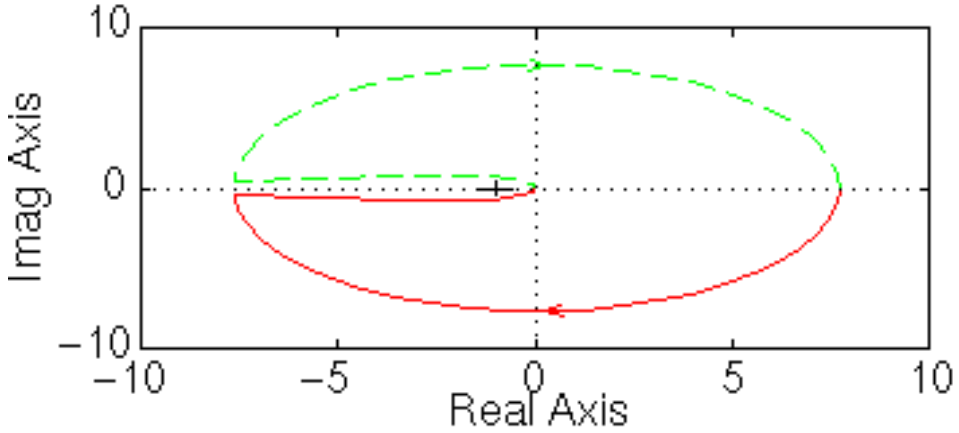
```
nyquist([1 2], [1 0 0])
```



nyquist1([1 2], [1 0 0])



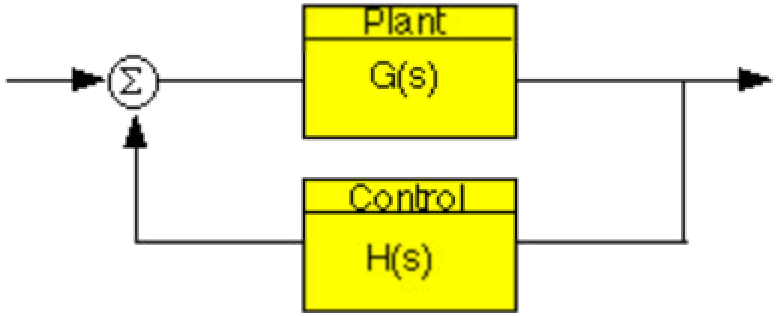
lnyquist1([1 2], [1 0 0])



Note that the nyquist plot is not the correct one, the nyquist1 plot is correct, but it's hard to see what happens close to the -1 point, and the lnyquist1 plot is correct and has an appropriate scale.

Closed Loop Stability

Consider the negative feedback system:



From the Cauchy criterion that the number *N* of times that the plot of $G(s)H(s)$ encircles -1 is equal to:

- Number *Z* of zeros of 1 + $G(s)H(s)$ enclosed by the frequency contour
- Minus the number P of poles of 1 + G(s)H(s) enclosed by the frequency contour (N = Z - P).

Keeping careful track of open- and closed-loop transfer functions, as well as numerators and denominators, you should convince yourself that:

- the zeros of 1 + *G(s)H(s)* are the poles of the closed-loop transfer function
- the poles of 1 + *G(s)H(s)* are the poles of the open-loop transfer function.

The Nyquist criterion then states that:

- *P* = the number of open-loop (unstable) poles of *G(s)H(s)*
- *N* = the number of times the Nyquist diagram encircles -1
- clockwise encirclements of -1 count as positive encirclements
- counter-clockwise (or anti-clockwise) encirclements of -1 count as negative encirclements
- *Z* = the number of right half-plane (positive, real) poles of the closed-loop system

The important equation which relates these three quantities is: $Z = P + N$

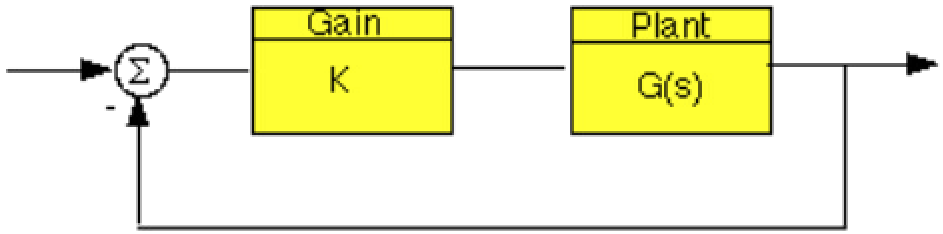
Note: This is only one convention for the Nyquist criterion. Another convention states that: A positive *N* counts the counter-clockwise or anti-clockwise encirclements of -1. The *P* and *Z* variables remain the same. In this case the equation becomes $Z = P - N$.

It is very important to learn how to count the number of times that the diagram encircles -1. Therefore, we will go into some detail to help you visualize this. You can view this movie as an example.

Another way of looking at it is to imagine you are standing on top of the -1 point and are following the diagram from beginning to end. Now ask yourself: How many times did I turn a full 360 degrees? Again, if the motion was clockwise, N is positive, and if the motion is anti-clockwise, N is negative.

Knowing the number of right-half plane (unstable) poles in open loop (P), and the number of encirclements of -1 made by the Nyquist diagram (N), we can determine the closed-loop stability of the system. If Z = P + N is a positive, nonzero number, the closed-loop system is unstable.

Example: 8.6 We can use the Nyquist diagram to find the range of gains for a closed-loop unity feedback system to be stable.



$$G(s) = \frac{s^2 + 10.s + 24}{s^2 - 8.s + 15}$$

This system has a gain K which can be varied in order to modify the response of the closed-loop system. However, we will see that we can only vary this gain within certain limits, since we have to make sure that our

closed-loop system will be stable. The first thing we need to do is find the number of positive real poles in our open-loop transfer function:

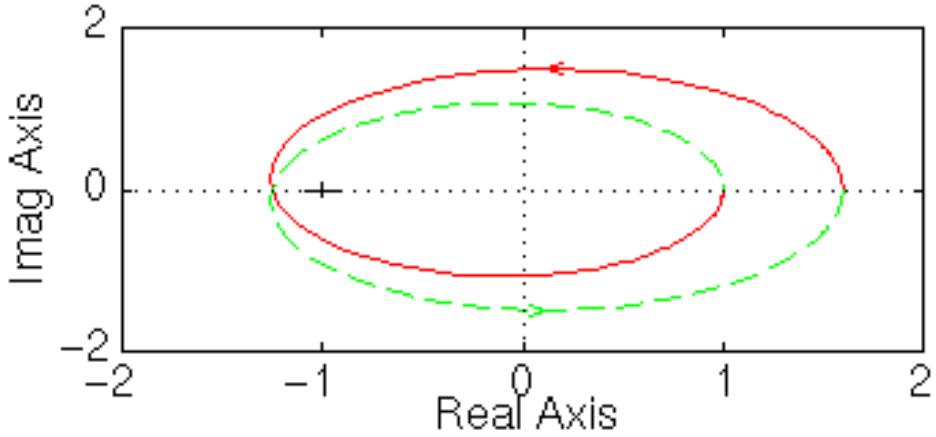
```
roots([1 -8 15])
```

ans =5; 3

The poles of the open-loop transfer function are both positive. Therefore, we need two anti-clockwise (N = -2) encirclements of the Nyquist diagram in order to have a stable closed-loop system (Z = P + N). If the number of encirclements is less than two or the encirclements are not anti-clockwise, our system will be unstable.

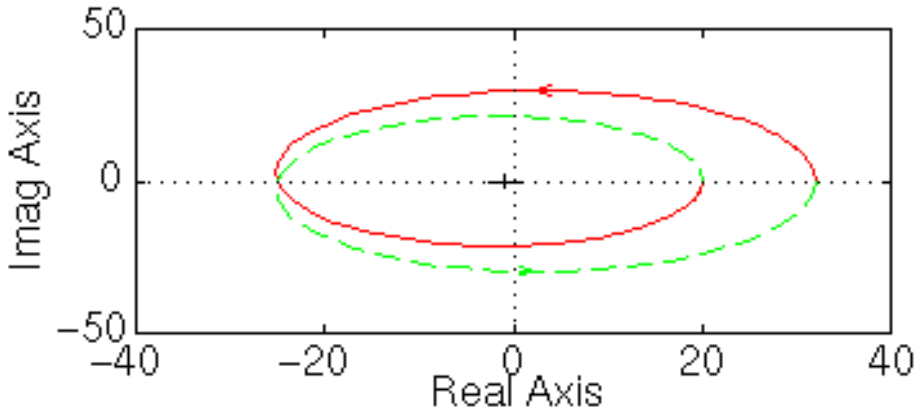
Let's look at our Nyquist diagram for a gain of 1:

```
nyquist([ 1 10 24], [ 1 -8 15])
```



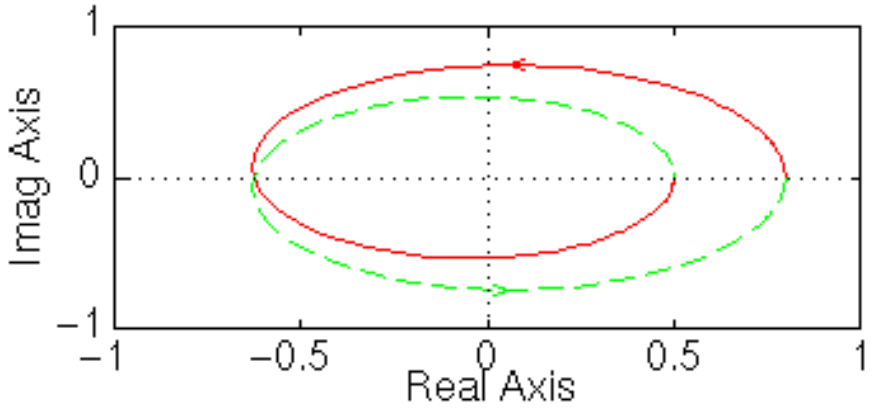
There are two anti-clockwise encirclements of -1. Therefore, the system is stable for a gain of 1. Now we will see how the system behaves if we increase the gain to 20:

```
nyquist(20*[ 1 10 24], [ 1 -8 15])
```



The diagram expanded. Therefore, we know that the system will be stable no matter how much we increase the gain. However, if we decrease the gain to 0.5, the diagram will contract and the system might become unstable.

```
nyquist(0.5*[ 1 10 24], [ 1 -8 15])
```



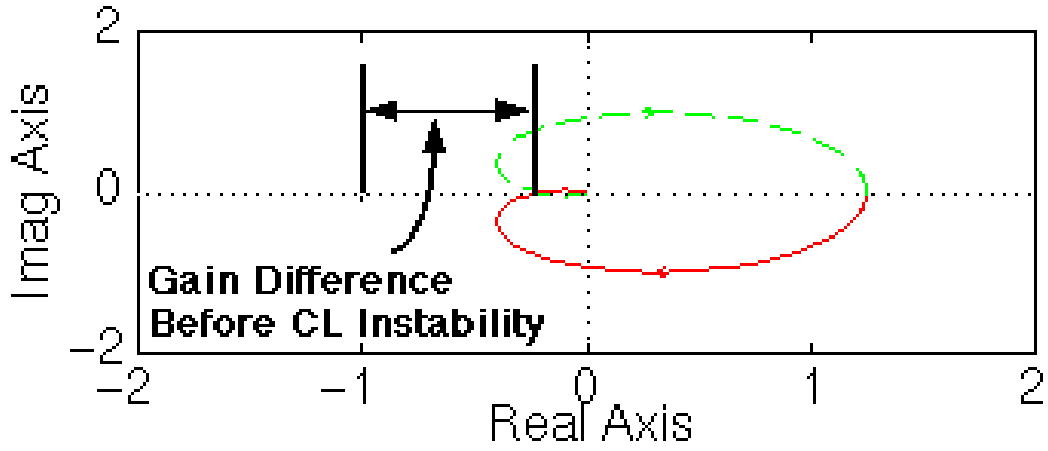
The system is now unstable. By trial and error we find that this system will become unstable for gains less than 0.80. We can verify our answers by zooming in on the Nyquist plots as well as by looking at the closed-loop steps responses for gains of 0.79, 0.80, and 0.81.

Gain Margin

We already defined the gain margin as the change in open-loop gain expressed in decibels (dB), required at 180 degrees of phase shift to make the system unstable. Now we are going to find out where this comes from. Suppose that we have a system that is stable if there are no Nyquist encirclements of -1, such as :

$$G(s) = \frac{50}{s^3 + 9.s^2 + 30.s + 40}$$

Looking at the roots, we find that we have no open loop poles in the RHP and therefore no closed-loop poles in the RHP if there are no Nyquist encirclements of -1. Now, how much can we vary the gain before this system becomes unstable in closed loop?



Observations:

1. The open-loop system represented by this plot will become unstable in closed loop if the gain is increased past a certain boundary.
2. The negative real axis area between -1/a (defined as the point where the 180 degree phase shift occurs, where the diagram crosses the real axis) and -1 represents the amount of increase in gain that can be tolerated before closed-loop instability.
3. If the gain is equal to a, the diagram will touch the -1 point:

$$G(jw) = \frac{-1}{a} \quad \Rightarrow a^{*}.G(jw) = -1$$

Therefore, we say that the gain margin is '*a*' units. Since the gain margin is normally measured in decibels. Hence, the gain margin in dB is :

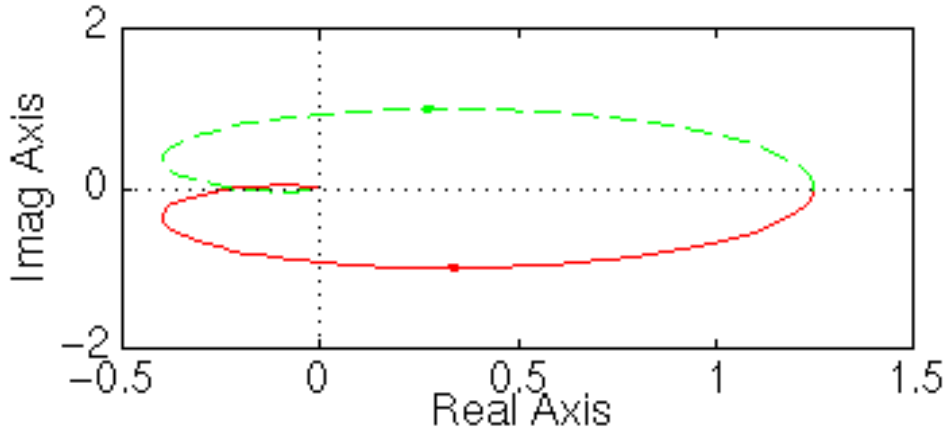
$$GM = 20 * \log_{10}(a)\ dB$$

Gain margin of the stable, open-loop transfer function we viewed before. Recall that the function is:

$$G(s) = \frac{50}{s^3 + 9.s^2 + 30.s + 40}$$

and that the Nyquist diagram can be viewed by typing:

```
nyquist (50, [1 9 30 40 ])
```



To find the gain margin we must find 'a', as defined above. To do this, we need to find the point where there is exactly 180 degrees of phase shift. This means that the transfer function at this point is real (has no imaginary part). The numerator is already real, so we just need to look at the denominator. When *s* = *j*w*, the only terms in the denominator that will have imaginary parts are those which are odd powers of s. Therefore, for $G(jw)$ to be real, we must have:

$$-jw^3 + 30.jw = 0$$

which means *w=0* (this is the rightmost point in the Nyquist diagram) or $w = \sqrt{30}$. We can then find the value of $G(jw)$ at this point using a matlab tool called: polyval:

```
polyval(50,j*w)/polyval([1 9 30 40],j*w)
```

Answer: -0.2174 + 0i.

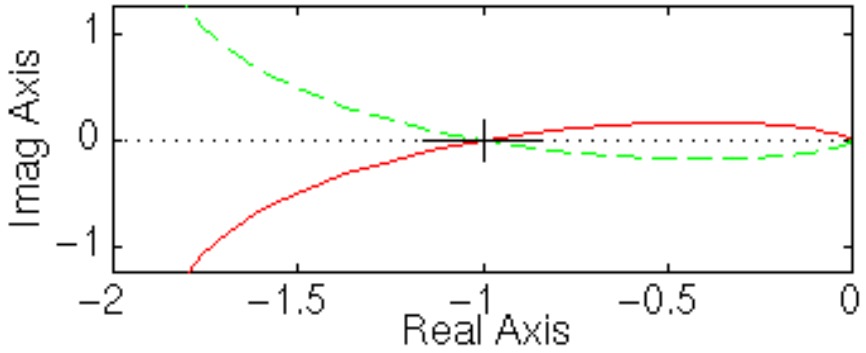
The imaginary part is zero, so we know that our answer is correct. We can also verify by looking at the Nyquist plot again. The real part also makes sense. Now we can proceed to find the gain margin.

We found that the 180 degrees phase shift occurs at *-0.2174 + 0i*. This point was previously defined as *-1/a*. Therefore, we now have '*a*', which is the gain margin and in dB.

$$1/a = -0.1274 \quad \Rightarrow a = 4.6 \quad \Rightarrow GM = 20 * \log_{10}(4.6) = 13.26\ dB$$

We now have our gain margin of a = 4.6 and by zooming-in on the Nyquist plot: a = 4.6

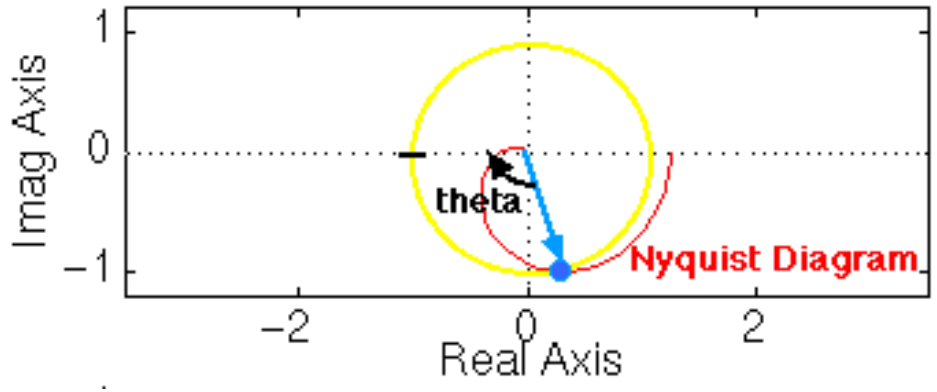
```
nyquist(a*50,[1 9 30 40])
```



The plot appears to go right through the -1 point.

Phase Margin

We have defined the phase margin as the change in open-loop phase shift required at unity gain to make a closed-loop system unstable.



From our previous example we know that this particular system will be unstable in closed loop if the Nyquist diagram encircles the -1 point. However, we must also realize that if the diagram is shifted by theta degrees, it will then touch the -1 point at the negative real axis, making the system marginally stable in closed loop. Therefore, the angle required to make this system marginally stable in closed loop is called the phase margin (measured in degrees). In order to find the point we measure this angle from, we draw a circle with radius of 1, find the point in the Nyquist diagram with a magnitude of 1 (gain of zero dB), and measure the phase shift needed for this point to be at an angle of 180 deg.

APPENDIX:

Function nyquist1:

This function is a modified version of the nyquist command of matlab and has been jointly prepared by the good folks at Carnegie-Mellon and University of Michigan's CTM Group[1]. It has all the same attributes as the original, with a few improvements. Nyquist1.m takes poles on the imaginary axis into account when creating the Nyquist plot, and plots around them. Furthermore, this function outputs the number of open-loop right hand plane poles, the number of anti-clockwise encirclements, and the number of closed-loop right half plane poles onto your screen.

Copy the following text into a file nyquis1t.m, and put it in the same directory as the Matlab software, or in a directory which is contained in Matlab's search path. Be sure to copy from the document source. We have observed some very peculiar bugs appearing when copying from the text appearing in the browser window.

```
function [reout,imt,w] = nyquist1(a,b,c,d,iu,w)
%NYQUIST1 Nyquist frequency response for continuous-time linear systems.
%
%       This Verison of the  NYQUIST Command takes into account poles at the
%       jw-axis and loops around them when creating the frequency vector  in order
%       to produce the appropriate Nyquist Diagram (The NYQUIST command does
%       not do this and therefore produces an incorrect plot when we have poles in the
%       jw axis).   As an added feature, this function outputs the number of open loop right
%        hand plane poles, the number of anti-clockwise encirclements, and the number of
%       closed loop right half plane poles on your screen.
%
%       NOTE: This version of NYQUIST1 does not account for pole-zero
%       cancellation.  Therefore, the user must simplify the transfer function before using
%       this command.

%NYQUIST1 Nyquist frequency response for continuous-time linear systems.
%       NYQUIST(A,B,C,D,IU) produces a Nyquist plot from the single input
%       IU to all the outputs of the system:
```

[1] This program and many other nice control systems tutorials can be found at: http://www.engin.umich.edu/group/ctm/basic/basic.html

```
%           .                          -1
%           x = Ax + Bu          G(s) = C(sI-A) B + D
%           y = Cx + Du      RE(w) = real(G(jw)), IM(w) = imag(G(jw))
%
%      The frequency range and number of points are chosen automatically.
%
%      NYQUIST1(NUM,DEN) produces the Nyquist plot for the polynomial
%      transfer function G(s) = NUM(s)/DEN(s) where NUM and DEN contain
%      the polynomial coefficients in descending powers of s.
%
%      NYQUIST1(A,B,C,D,IU,W) or NYQUIST(NUM,DEN,W) uses the user-supplied
%       freq. vector W which must contain the frequencies, in radians/sec,
%      at which the Nyquist response is to be evaluated.  When invoked
%      with left hand arguments,
%           [RE,IM,W] = NYQUIST(A,B,C,D,...)
%           [RE,IM,W] = NYQUIST(NUM,DEN,...)
%      returns the frequency vector W and matrices RE and IM with as many
%      columns as outputs and length(W) rows.  No plot is drawn on the
%      screen.
%      See also: LOGSPACE,MARGIN,BODE, and NICHOLS.

%      J.N. Little 10-11-85
%      Revised ACWG 8-15-89, CMT 7-9-90, ACWG 2-12-91, 6-21-92,
%           AFP 2-23-93
%           WCM 8-31-97
%
% ****************************************************************
%  Modifications made to the nyquist - takes into account poles on jw axis.
%  then goes around these to make up frequency vector
%


if nargin==0, eval('exresp("nyquist")'), return, end

% --- Determine which syntax is being used ---
if (nargin==1),
```

```
        error('Wrong number of input arguments.');

elseif (nargin==2),     % Transfer function form without frequency vector
        num  = a; den = b;
        w = freqint2(num,den,30);
        [ny,nn] = size(num); nu = 1;

elseif (nargin==3),     % Transfer function form with frequency vector
        num = a; den = b;
        w = c;
        [ny,nn] = size(num); nu = 1;

elseif (nargin==4),     % State space system, w/o iu or frequency vector
        error(abcdchk(a,b,c,d));
        w = freqint2(a,b,c,d,30);
        [iu,nargin,re,im]=mulresp('nyquist',a,b,c,d,w,nargout,0);
        if ~iu, if nargout, reout = re; end, return, end
        [ny,nu] = size(d);

elseif (nargin==5),     % State space system, with iu but w/o freq. vector
        error(abcdchk(a,b,c,d));
        w = freqint2(a,b,c,d,30);
        [ny,nu] = size(d);

else
        error(abcdchk(a,b,c,d));
        [ny,nu] = size(d);

end

if nu*ny==0, im=[]; w=[]; if nargout~=0, reout=[]; end, return, end

% ***************************************************************************
% depart from the regular nyquist program here
% now we have a frequency vector, a numerator and denominator
% now we create code to go around all poles and zeroes here.
```

```
if (nargin==5) | (nargin ==4) | (nargin == 6)
       [num,den]=ss2tf(a,b,c,d)
end
tol = 1e-6;  %defined tolerance for finding imaginary poles
z = roots(num);
p = roots(den)
% ***** If all of the poles are at the origin, just move them a tad to the left***
if norm(p) == 0
 length_p = length(p);
 p = -tol*ones(length_p,1);
 den = den(1,1)*[1  tol];
 for ii = 2:length_p
  den = conv(den,[1  tol])
 end
end

zp = [z;p];        % combine the zeros and poles of the system
nzp = length(zp);  % number of zeros and poles
ones_zp=ones(nzp,1);
%Ipo = find((abs(real(p))=0)); %index poles with zero real part + non-neg imag part
Ipo = find((abs(real(p))< tol) & (imag(p)>=0)); %index poles with zero real part + non-neg imag part
if  ~isempty(Ipo)   %
% **** only if we have such poles do we do the following:*************************
po = p(Ipo); % poles with 0 real part and non-negative imag part
% check for distinct poles
[y,ipo] = sort(imag(po));  % sort imaginary parts
po = po(ipo);
dpo = diff(po);
idpo = find(abs(dpo)> tol);
idpo = [1;idpo+1];   % indexes of the distinct poles

po = po(idpo);   % only distinct poles are used
nIpo = length(idpo); % # of such poles
originflag = find(imag(po)==0);  % locate origin pole
```

```
s = [];  % s is our frequency response vector
w = sqrt(-1)*w;  % create a jwo vector to evaluate all frequencies with
for ii=1:nIpo % for all Ipo poles

      [nrows,ncolumns]=size(w);
      if nrows == 1
            w = w.';  % if w is a row, make it a column
      end;
      if nIpo == 1
            r(ii) = .1;
      else           % check distances to other poles and zeroes
            pdiff = zp-po(ii)*ones_zp  % find the differences between
            % poles you are checking and other poles and zeros
             ipdiff = find(abs(pdiff)> tol) % ipdiff is all nonzero differences

            r(ii)=0.2*min(abs(pdiff(ipdiff))) % take half this difference
            r(ii)=min(r(ii),0.1)  % take the minimum of this diff.and .1
      end;
      t = linspace(-pi/2,pi/2,25);
      if (ii == originflag)
            t = linspace(0,pi/2,25);
       end;    % gives us a vector of points around each Ipo
      s1 = po(ii)+r(ii)*(cos(t)+sqrt(-1)*sin(t));  % detour here
       s1 = s1.';  % make sure it is a column

      % Now here I reconstitute s - complex frequency - and
      % evaluate again.

      bottomvalue = po(ii)-sqrt(-1)*r(ii);  % take magnitude of imag part
       if (ii ==  originflag)  % if this is an origin point
            bottomvalue = 0;
      end;
      topvalue = po(ii)+sqrt(-1)*r(ii); % the top value where detour stops
       nbegin = find(imag(w) < imag(bottomvalue)); %
       nnbegin = length(nbegin); % find all the points less than encirclement
      if (nnbegin == 0)& (ii ~= originflag)    % around jw root
```

```
            sbegin = 0
        else sbegin = w(nbegin);
        end;
        nend = find(imag(w) > imag(topvalue));  % find all points greater than
        nnend = length(nend);    % encirclement around jw root
        if (nnend == 0)
            send = 0
        else send = w(nend);
        end
        w = [sbegin; s1; send];  % reconstituted half of jw axis
end
else
        w = sqrt(-1)*w;
end
%end  % this ends the loop for imaginary axis poles
% ************************************************************
% back to the regular nyquist program here
% Compute frequency response
if (nargin==2)|(nargin==3)
        gt = freqresp(num,den,w);
else
        gt = freqresp(a,b,c,d,iu,w);
end
% ************************************************************

%       nw = length(gt);
%       mag = abs(gt);   % scaling factor added
%       ang = angle(gt);
%       mag = log2(mag+1);    % scale by log2(mag) throughout

%       for n = 1:nw
%           h(n,1) = mag(n,1)*(cos(ang(n,1))+sqrt(-1)*sin(ang(n,1)));
%       end;  % recalculate G(jw) with scaling factor

%       gt = h;
% ************************************************************
```

```
ret=real(gt);
imt=imag(gt);

% If no left hand arguments then plot graph.
if nargout==0,
  status = ishold;
  plot(ret,imt,'r-',ret,-imt,'g--')

%  plot(real(w),imag(w))
% modifications added here

 % ****************************************
       % modifications added here to count encirclements
      [numc,denc] = tfchk(num,den);
       % create the + and - reflection of G(jw) on imag axis
      gw = [(ret + j*imt); numc(1)/denc(1); (flipud(ret) - j*flipud(imt))]; %
       %look at G(jw)
      [Ngw,Mgw] = size(gw);  % size of evaluated G(jw)
      gwp1 = gw + ones(Ngw,Mgw);
      % moves origin from 0 to -1, so we
       %  can count encirclements around -1 now
       initial_angle = rem(180/pi*angle(gwp1(1)) + 360, 360);
       % define initial angle
      angle_gwp1 = rem(180/pi*angle(gwp1) - initial_angle + 720,360);
       % angle from origin for all points
      dagw = diff(angle_gwp1);
       % subtract off initial angle find degrees of  encirclement
      tolerance = 180;
       % define tolerance - where encirclement counter "flips" over
      Ipd = find(dagw < -tolerance);
       % number of anti-clockwise encirclements
      Ind = find(dagw > tolerance);
       % number of clockwise encirclemtents
      Nacw = max(size(Ipd)) - max(size(Ind));   % number of encirclements
      % Nyquist Criterion says  Z = P - N
      P = length(find(p>0))
```

```
      disp('P = number of Open loop poles in rhp');
      N = Nacw
      disp('N =  number of anti-clockwise encirclements')
       Z = P - N     %
      disp('Z = number of closed loop poles in rhp')

%*****************************************

   set(gca, 'YLimMode', 'auto')
   limits = axis;
   % Set axis hold on because next plot command may rescale
   set(gca, 'YLimMode', 'auto')
   set(gca, 'XLimMode', 'manual')
   hold on
   % Make arrows
   for k=1:size(gt,2),
       g = gt(:,k);
       re = ret(:,k);
       im = imt(:,k);
       sx = limits(2) - limits(1);
       [sy,sample]=max(abs(2*im));
       arrow=[-1;0;-1] + 0.75*sqrt(-1)*[1;0;-1];
       sample=sample+(sample==1);
       reim=diag(g(sample,:));
       d=diag(g(sample+1,:)-g(sample-1,:));
       % Rotate arrow taking into account scaling factors sx and sy
       d = real(d)*sy + sqrt(-1)*imag(d)*sx;
       rot=d./abs(d);          % Use this when arrow is not horizontal
       arrow = ones(3,1)*rot'.*arrow;
       scalex = (max(real(arrow)) - min(real(arrow)))*sx/50;
       scaley = (max(imag(arrow)) - min(imag(arrow)))*sy/50;
       arrow = real(arrow)*scalex + sqrt(-1)*imag(arrow)*scaley;
       xy =ones(3,1)*reim' + arrow;
       xy2=ones(3,1)*reim' - arrow;
       [m,n]=size(g);
```

```
        hold on
        plot(real(xy),-imag(xy),'r-',real(xy2),imag(xy2),'g-')
    end
    xlabel('Real Axis'), ylabel('Imag Axis')

    limits = axis;
    % Make cross at s = -1 + j0, i.e the -1 point
    if limits(2) >= -1.5  & limits(1) <= -0.5 % Only plot if -1 point is not far out.
        line1 = (limits(2)-limits(1))/50;
        line2 = (limits(4)-limits(3))/50;
        plot([-1+line1, -1-line1], [0,0], 'w-', [-1, -1], [line2, -line2], 'w-')
    end

    % Axis
    plot([limits(1:2);0,0]',[0,0;limits(3:4)]','w:');

    if ~status, hold off, end    % Return hold to previous status
    return % Suppress output
end
%reout = ret;
%   plot(real(p),imag(p),'x',real(z),imag(z),'o');


% End of the package
```