Indian Institute of Technology Bombay

EE 324: Control Systems Lab

Problem Sheet 7 Report

March 14, 2021

Student Name	*Roll Number*
Mantri Krishna Sri Ipsit	180070032

Contents

1 Question 1

The open loop transfer function given is

$$G(s) = \frac{1}{(s+3)(s+4)(s+12)}$$

1.1 Part a

We have $z = 0.01$ and $\zeta = 0.2$. Hence the slope of the constant ζ line drawn on the root locus will be 4.8989. Using the root locus, we get $K = 666.3$.

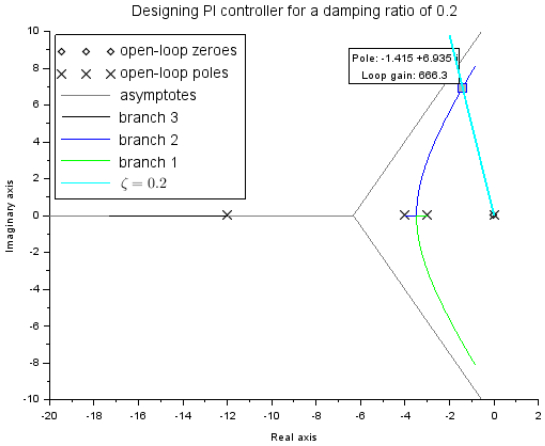


Figure 1: PI controller for $\zeta = 0.2$ and $z = 0.01$

1.2 Part b

We have to design a PI controller to obtain undamped natural frequencies of 8 rad/s and 9 rad/s. This means the root locus should intersect the imaginary axis at $\pm j8$ and $\pm j9$ respectively. IF we take $z = 0.01$ and proceed, then we get $K = 953.4$ for 8 rad/s and $K = 1329$ for 9 rad/s respectively.

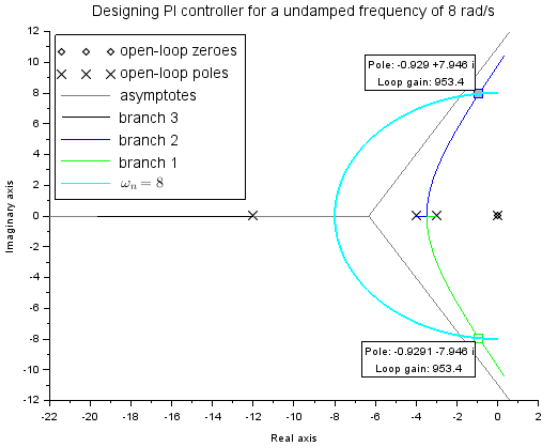


Figure 2: PI controller for $\omega_n = 8$ and $z = 0.01$

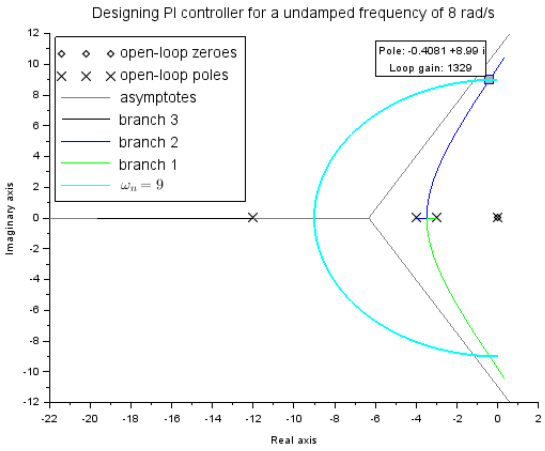
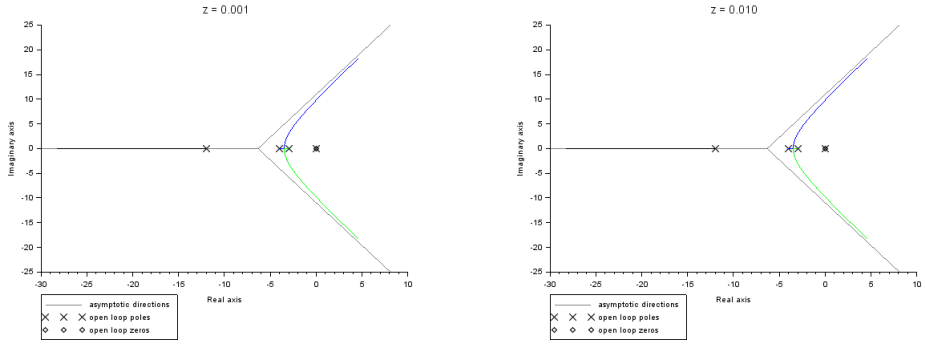


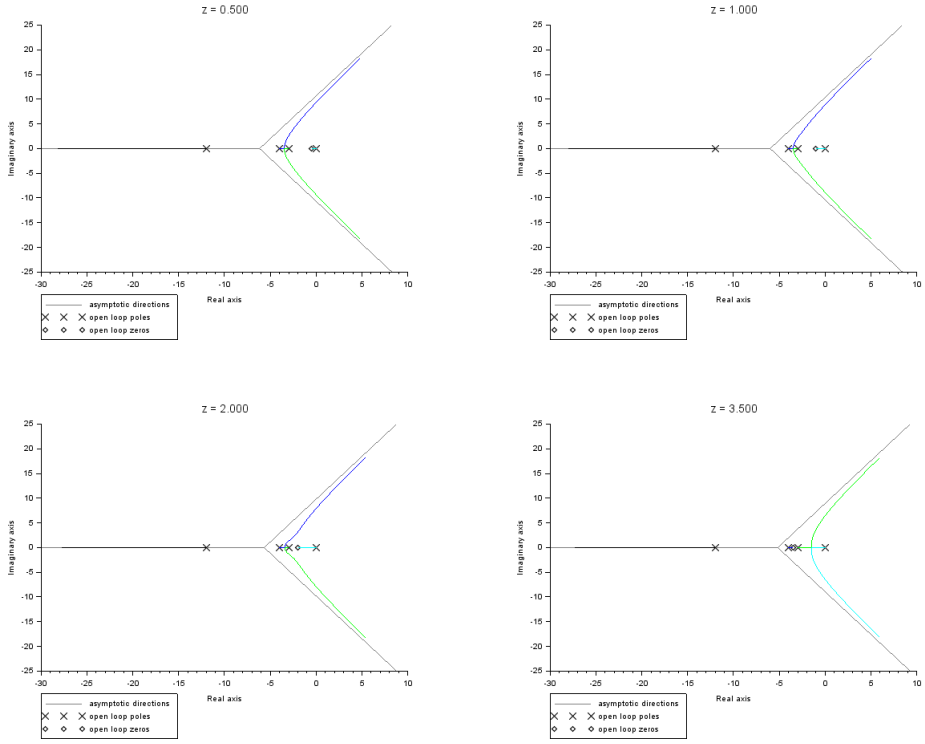
Figure 3: PI controller for $\omega_n = 8$ and $z = 0.01$

1.3 Part c



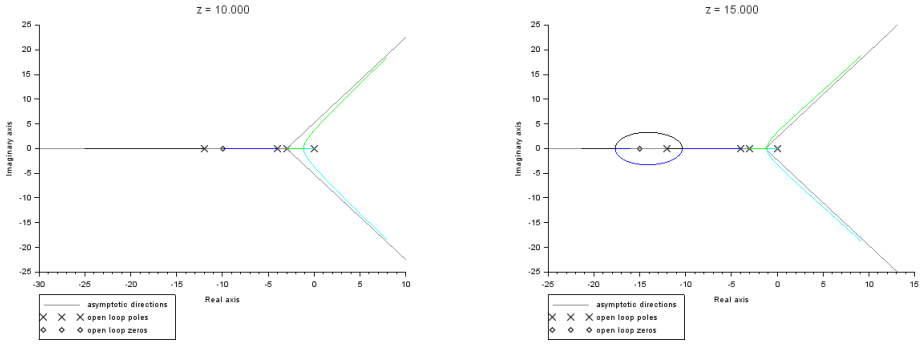
1.4 Part d

No, it is not possible to change the pole locations of a system using a PI controller without changing the damping raito. This is because a PI controller essentially increases the Type of the systems so that the steady state error of the system goes to zero. A PI controller is designed in such a way that the angle contribution due to origin and the zero of the PI controller cancel out each other and the root locus does not change much. This can also be observed from the plots in part c above. As the zero moves farther away from the origin, the root locus changes drastically.



All the above figures have been generated using the following code:

```
clc; clear;
s = poly(0, 's');
g = 1 / ((s+3)*(s+4)*(s+12));
G = syslin('c', g);
//-----------------------------
// Part a
zeta = 0.2;
slope = sqrt(1 - zeta^2) / zeta;
x = -2:0.001:0;
y = -slope * x;
z = 0.01;
ga = g * (s + z) / s;
Ga = syslin('c', ga);
scf();
evans(Ga, 1000);
plot(x, y, 'c-', 'LineWidth', 2);
L = legend(['open-loop zeroes', 'open-loop poles', 'asymptotes',...
'branch 3', 'branch 2', 'branch 1', "$\zeta = 0.2$"]);
L.font_size = 3;
L.legend_location = "in_upper_left";
title(["Designing PI controller for a damping ratio of 0.2"], 'fontsize', 3);
poi = -1.416 + %i*6.935; // from the root locus
Ka = 666.3; // from the root locus
disp("K for damping ratio of 0.2 = ");
disp(Ka);
//-----------------------------
// Part b
```




```
y = -8:0.01:8;
x = -sqrt(64 - y^2);
gb = g * (s + z) / s;
Gb = syslin('c', gb);
scf();
evans(Gb, 2000);
plot(x, y, 'c-', 'LineWidth', 2);
L = legend(['open-loop zeroes', 'open-loop poles', 'asymptotes',...
'branch 3', 'branch 2', 'branch 1', "$\omega_n = 8$"]);
L.font_size = 3;
L.legend_location = "in_upper_left";
title(["Designing PI controller for a undamped frequency of 8 rad/s"], 'fontsize', 3);
Kb1 = 953.4; // from the root locus
disp("K for undamped natural frequency = 8 rad/s");
disp(Kb1);
y = -9:0.01:9;
x = -sqrt(81 - y^2);
scf();
evans(Gb, 2000);
plot(x, y, 'c-', 'LineWidth', 2);
L = legend(['open-loop zeroes', 'open-loop poles', 'asymptotes',...
'branch 3', 'branch 2', 'branch 1', "$\omega_n = 9$"]);
L.font_size = 3;
L.legend_location = "in_upper_left";
title(["Designing PI controller for a undamped frequency of 8 rad/s"], 'fontsize', 3);
Kb2 = 1329; // from the root locus
disp("K for undamped natural frequency = 9 rad/s");
disp(Kb2);
//-----------------------------
// Part c
Z = [0.001, 0.01, 0.5, 1, 2, 3.5, 10, 15];
for i=1:size(Z, 2)
    z = Z(i);
    gc = g * (s + z) / s;
    Gc = syslin('c', gc);
    scf();
    evans(Gc, 10000);
    title([sprintf("z = %0.3f", z)], 'fontsize', 3)
end
```

2 Question 2

Ratio of zero magnitude to pole magnitude is 20. We have to design a lag compensator. This means the magnitude of pole should be larger than that of the zero of this controller.

2.1 Part a

We have the transfer function

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

This is an overdamped system. We want 10% OS. Using root locus design method, we find that a proportional controller with a gain $K = 4.437$ will work.

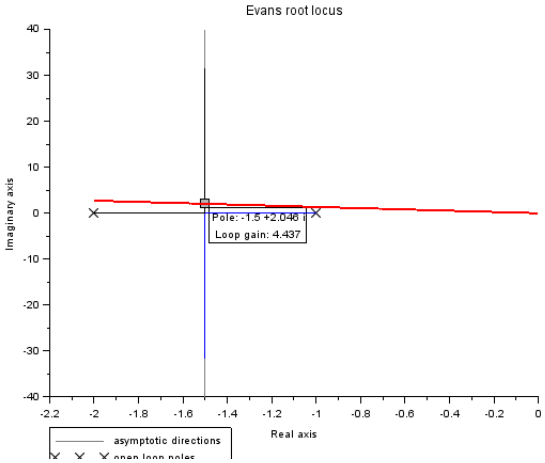


Figure 4: P controller for 10% OS

2.2 Part b

The steady state error will be

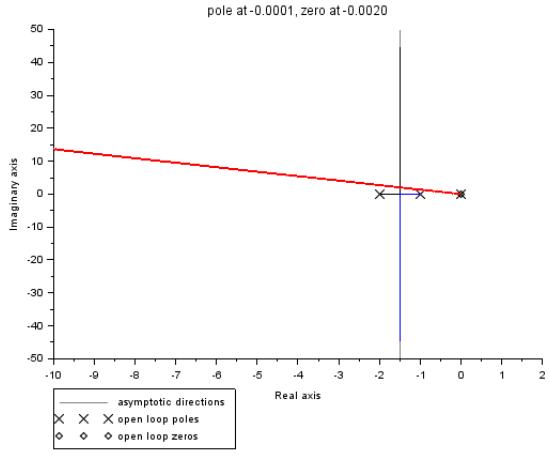
$$e(\infty) = \frac{1}{1 + KG(0)} = 0.3107037$$

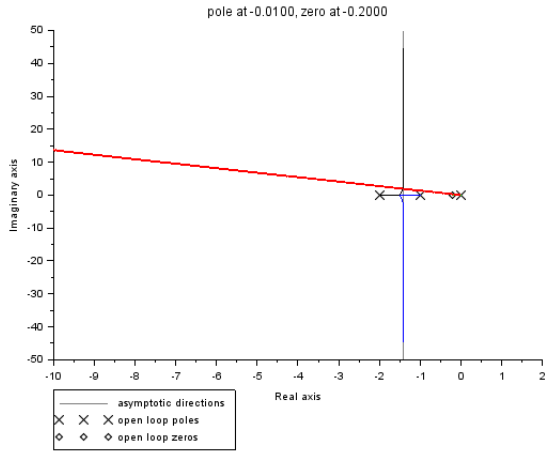
After adding the lag compensator with pole at -0.0001 and zero at -0.002. Hence the steady state error after compensation is

$$e(\infty) = \frac{1}{1 + KG(0)|z/p|} = 0.022041$$

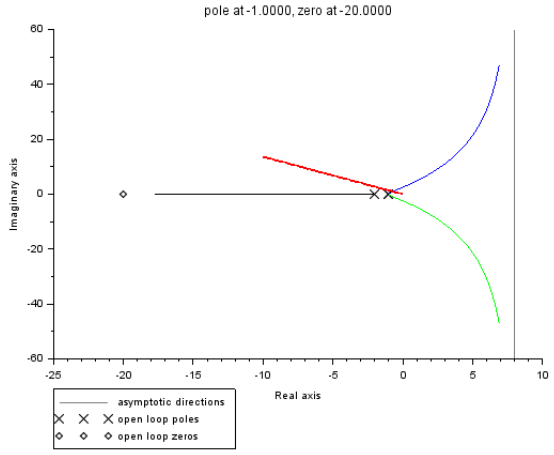
2.3 Part c

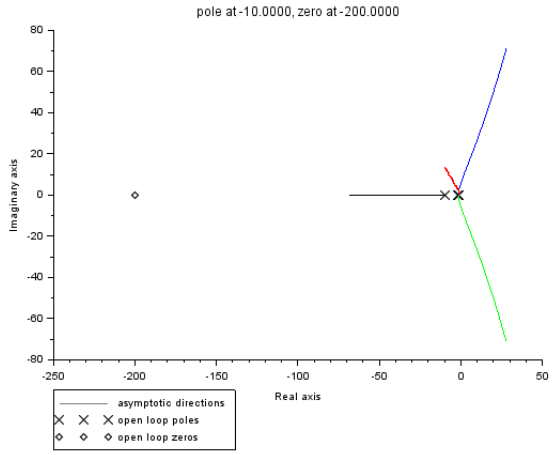
Here I varied the pole locations as [0.0001, 0.01, 1, 10, 20]. The corresponding plots are as follows: We can see that as the pole moves farther away from origin, time taken to reach the given %OS increases, i.e., the transient response lasts longer.



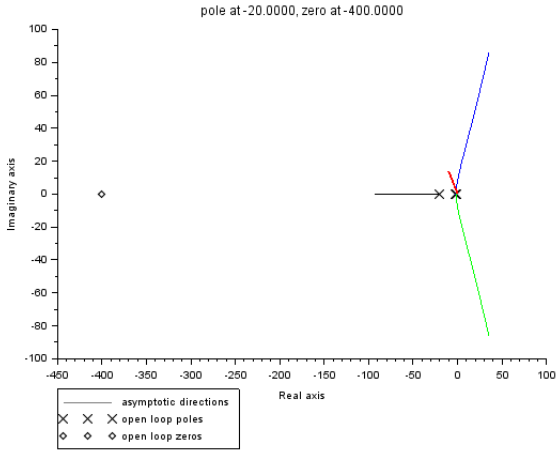



```
clc; clear;
s = poly(0, 's');
ratio = 20;
//--------------------
// Part a
g = 1 / (s^2 + 3*s + 2);
slope = %pi / log(10);
G = syslin('c', g);
scf();
evans(G, 1000);
x = -10:0.0001:0;
y = -slope * x;
plot(x, y, 'r-', 'LineWidth', 2);
K = 4.437; // constant gain for 10% OS
disp("K for 10 % OS = ");
disp(K);
//--------------------
// Part b
sse_original = 1 / (1 + K * 0.5);
disp("Original Steady State Error = ");
```






```
disp(sse_original);
// choose pole very close to origin
pole = 0.0001;
zero = ratio * pole;
g_lag_comp = (s + zero) * g / (s + pole);
G_lag_comp = syslin('c', g_lag_comp);
scf();
evans(G_lag_comp, 1000);
sse_lag_comp = 1 / (1 + (K * 20 * 0.5));
disp("New Steady State Error = ");
disp(sse_lag_comp);
//--------------------
// Part c
Poles = [0.0001, 0.01, 1, 10, 20];
for i=1:size(Poles, 2)
    pole = Poles(i);
    zero = pole * ratio;
    g_lag_comp = (s + zero) * g / (s + pole);
    G_lag_comp = syslin('c', g_lag_comp);
    scf();
```




```
41      evans(G_lag_comp, 2000);
42      plot(x, y, 'r-', 'LineWidth', 2);
43      title(sprintf("pole at -%.4f, zero at -%.4f", pole, zero));
44  end
```

3 Question 3

3.1 Part a

We have to design a lead compensator. The 2% settling time of the system in Q2a is 2.337s. So the required 2% settling here is 1.1679s. We also want the %OS to be 10%. Using the root locus method, we get the point on root locus which satisfies both these properties as

$$p = -3.42509740 + j4.67312190$$

For the lead compensator, we should choose a zero suitably left and the pole should be more left. We choose

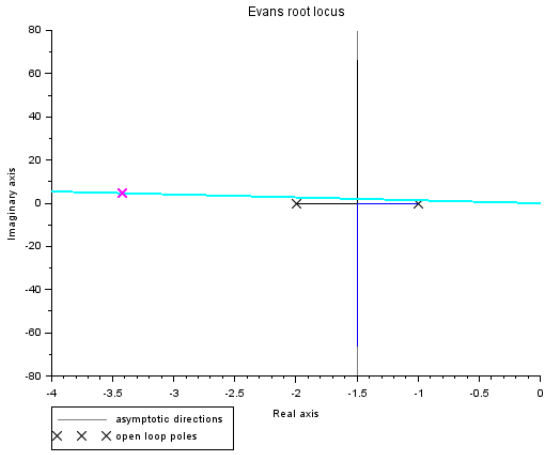


Figure 5: $T_s = 1.1679s$, $\%OS = 10$

the zero to be at -6 and then find the pole using the angle conditions. Then we get $p_c = -18.94350523$ and

the transfer function as

$$G_1(s) = \frac{4.437s + 26.622}{s^3 + 21.943505s^2 + 58.830516s + 37.88701}$$

3.2 Part b

The PD controller satisfying the same specs will have a zero at -8.19941903. The plant's transfer function will be

$$G_2(s) = \frac{36.380822 + 4.437s}{2 + 3s + s^2}$$

```
clc; clear;
s = poly(0, 's');
g = 1 / (s^2 + 3*s + 2);
G = syslin('c', 4.437 * g);
G_old = G /. syslin('c', 1, 1);
disp("Uncompensated Transfer function")
disp(G_old);
settling_t = 2.3357;
disp(sprintf("Old settling time = %.4f", settling_t));
slope = %pi / log(10);
x = -4:0.001:0;
y = -slope * x;
scf();
evans(G, 1000);
plot(x, y, 'c-', 'LineWidth', 2);
new_settling_t = settling_t / 2;
disp(sprintf("New settling time = %.4f", new_settling_t));
new_real_part = -4 / new_settling_t;
disp(sprintf("Real part of new dominant pole = %.8f", new_real_part));
new_imaginary_part = -slope * new_real_part;
disp(sprintf("Imaginary part of new dominant pole = %.8f", new_imaginary_part));
plot(new_real_part, new_imaginary_part, 'mx', 'LineWidth', 2);
// assume the zero of the lead compensator is at -6
z_c = 6;
[z, p, _] = tf2zp(G);
if z ~= [] then
    zeroes = zeros(size(z, 1), 2);
end
if p ~= [] then
    poles = zeros(size(p, 1), 2);
end
for i=1:size(p, 1)
    pi = p(i);
    pi = [real(pi), imag(pi)];
    poles(i, :) = pi;
end
for i=1:size(z, 1)
    zi = z(i);
    zi = [real(zi), imag(zi)];
    zeroes(i, :) = zi;
end
angles = 0;
for i=1:size(poles, 1)
    vector = [new_real_part, new_imaginary_part] - poles(i, :);
    angles = angles - atan(vector(2), vector(1))
```

```
end
angles = angles + atan(new_imaginary_part, z_c + new_real_part);
angle_by_new_pole = angles - %pi + 2*%pi;
disp(sprintf("Required angle to be compensated by the pole = %.4f", angle_by_new_pole));
p_c = (new_imaginary_part / tan(angle_by_new_pole)) - new_real_part;
disp(sprintf("p_c = %.8f", p_c));
g_comp = (s + z_c) * 4.437 * g / (s + p_c);
G_comp = syslin('c', g_comp);
disp(G_comp);
K = 0.01:1:100;
//----------------------------------------
// Part b
angles = 0;
for i=1:size(poles, 1)
    vector = [new_real_part, new_imaginary_part] - poles(i, :);
    angles = angles - atan(vector(2), vector(1))
end
angle_by_new_zero = -angles + %pi;
disp(sprintf("Required angle to be compensated by the zero = %.4f", angle_by_new_zero));
z_c = (new_imaginary_part / tan(angle_by_new_zero)) - new_real_part;
disp(sprintf("z_c = %.8f", z_c));
g_pd = g * 4.437 * (s + z_c);
G_pd = syslin('c', g_pd);
disp(G_pd);
```

4 Question 4

4.1 Part a

$$G(s) = \frac{1}{s^2 + 5s + 6}$$

We vary the frequency of the system and plot the outputs for sinusoidal inputs. Frequency is varied from $0.2Hz$ to $5Hz$ and the gains, as well as phase differences are calculated both from the plots, as well as analytically.

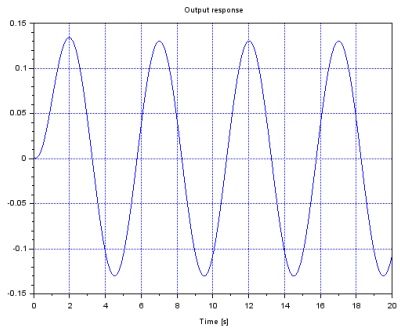
The table showing the analytically and numerically calculated gains and phase differences is shown.
The outputs are shown as follows in Figure 7

The above computation is done using the following Scilab code

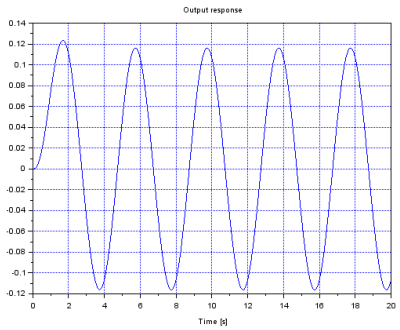
```
s = poly(0,'s');
   G = 1/(s^2+5*s+6);
   TF = syslin('c',G);

   f = [0.2,0.25,0.5,1,2,5];
   t = 0:0.01:20;
   half_length = int(length(t)/2);
   gain = [0,0,0,0,0,0]
   phase_diff = [0,0,0,0,0,0];

   for i=1:length(f)
       x = sin(2*%pi*f(i)*t);
       y = csim(x,t,TF);
```

Frequency (Hz)	Gain (plot)	Gain (math)	Phase (plot)	Phase (math)
0.2	0.130162	0.1301641	0.9550442	0.9576568
0.25	0.1161163	0.1161188	1.1466813	1.1481217
0.5	0.0618059	0.061814	1.8221237	1.812334
1	0.0217744	0.0217816	2.3876104	2.3879661
2	0.00608	0.0060829	2.7646021	2.7494153
5	0.0009863	0.0010066	2.8274334	2.9828124

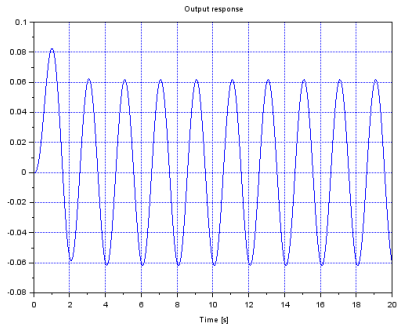
Table 1: Table of values



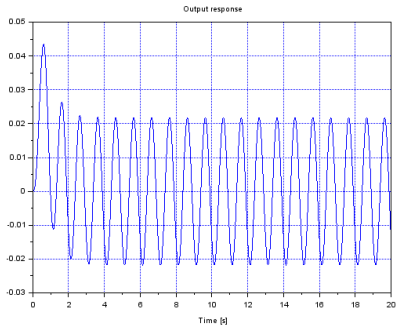
(a) Frequency $= 0.2Hz$



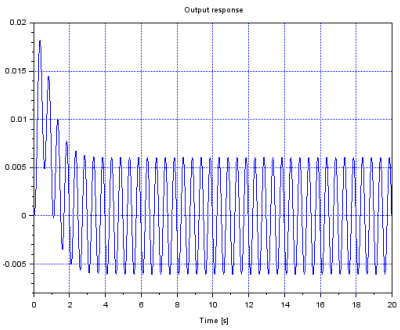
(b) Frequency $= 0.25Hz$



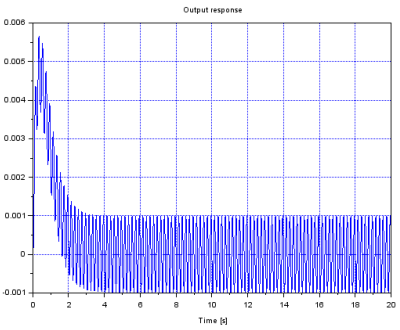
(c) Frequency $= 0.5Hz$



(d) Frequency $= 1Hz$



(e) Frequency $= 2Hz$



(f) Frequency $= 5Hz$

Figure 6: Outputs of system for varying frequencies

```
        amp_in = (max(x(half_length:$))-min(x(half_length:$)))/2;
        amp_out = (max(y(half_length:$))-min(y(half_length:$)))/2;
        gain(i) = amp_out/amp_in;
        temp1 = find(x(length(t)-400:length(t))==max(x(length(t)-400:
        length(t))));
        temp2 = find(y(length(t)-400:length(t))==max(y(length(t)-400:
        length(t))));
        t1 = t(temp1(1)+length(t)-400-1);
        t2 = t(temp2(1)+length(t)-400-1);
        phase_diff(i) = (t2-t1)2%pi*f(i);
        scf();
        clf();
        plot(t,y,'b');
        xlabel('Time [s]');
        title('Output response');
        xgrid(2);
    end

    num_gain = [0,0,0,0,0,0];
    num_phase = [0,0,0,0,0,0];

    for j=1:length(f)
        jw = %i*2*%pi*f(j);
        Gain = 1/((jw)^2+5*(jw)+6);
        num_gain(j) = abs(Gain);
        num_phase(j) = atan(imag(Gain)/real(Gain));
    end
```

4.2 Part b

The frequency must be in units of $rad.s^{-1}$

4.3 Part c

$$G(s) = \frac{60}{s^3 + 6s^2 + 11s + 6}$$

We vary the frequency of the system and plot the outputs for sinusoidal inputs. Frequency is varied from $0.2Hz$ to $5Hz$ and the gains, as well as phase differences are calculated both from the plots, as well as analytically.

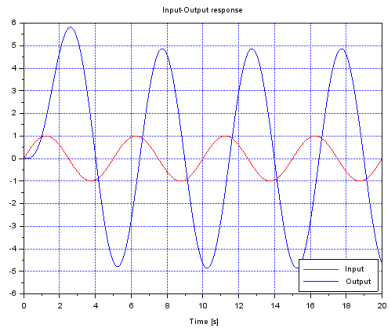
The table showing the analytically and numerically calculated gains and phase differences is shown.
The outputs are shown as follows in Figure 7

The above computation is done using the following Scilab code

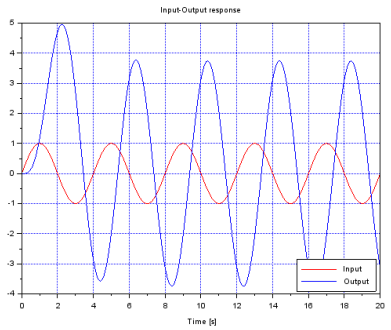
```

    s = poly(0,'s');
    G = 60/(s^3+6s^2+11s+6);
    TF = syslin('c',G);

    f = [0.2,0.25,0.5,1,2,5];
    t = 0:0.01:20;
    half_length = int(length(t)/2);
```

Frequency (Hz)	Gain (plot)	Gain (math)	Phase (plot)	Phase (math)
0.2	4.8629388	4.8630133	1.8598229	1.8578317
0.25	3.7416878	3.7415522	2.151991	2.151021
0.5	1.1248882	1.1249441	3.0787608	3.1087928
1	0.205292	0.2054132	3.7699112	3.6682510
2	0.0289914	0.0289523	-2.136283	-2.136176
5	0.0019129	0.0019215	4.3982297	4.3993219

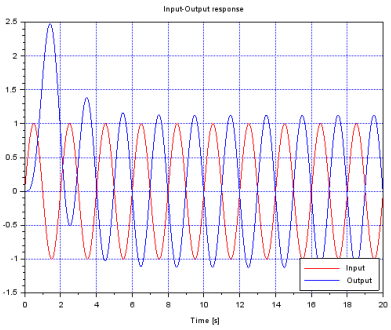
Table 2: Table of values



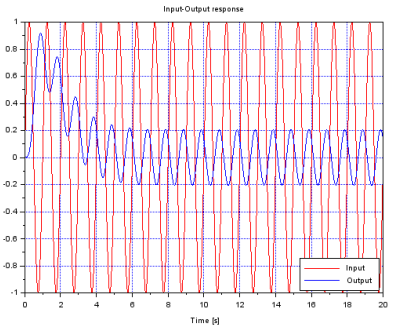
(a) Frequency $= 0.2Hz$



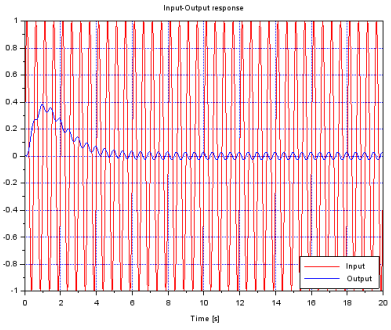
(b) Frequency $= 0.25Hz$



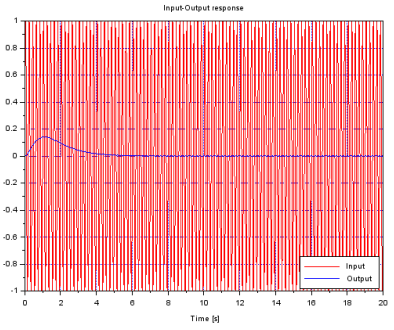
(c) Frequency $= 0.5Hz$



(d) Frequency $= 1Hz$



(e) Frequency $= 2Hz$



(f) Frequency $= 5Hz$

Figure 7: Outputs of system for varying frequencies

```
    gain = [0,0,0,0,0,0]
    phase_diff = [0,0,0,0,0,0];

    for i=1:length(f)
        x = sin(2*%pi*f(i)*t);
        y = csim(x,t,TF);
        amp_in = (max(x(half_length:$))-min(x(half_length:$)))/2;
        amp_out = (max(y(half_length:$))-min(y(half_length:$)))/2;
        gain(i) = amp_out/amp_in;
        temp1 = find(x(length(t)-400:length(t))==max(x(length(t)-400:
        length(t))));
        temp2 = find(y(length(t)-400:length(t))==max(y(length(t)-400:
        length(t))));
        t1 = t(temp1(1)+length(t)-400-1);
        t2 = t(temp2(1)+length(t)-400-1);
        phase_diff(i) = (t2-t1)2%pi*f(i);
        scf();
        clf();
        plot(t,x,'r');
        plot(t,y,'b');
        xlabel('Time [s]');
        title('Output response');
        xgrid(2);
    end

    num_gain = [0,0,0,0,0,0];
    num_phase = [0,0,0,0,0,0];

    for j=1:length(f)
        jw = %i*2*%pi*f(j);
        Gain = 1/((jw)^2+5*(jw)+6);
        num_gain(j) = abs(Gain);
        num_phase(j) = atan(imag(Gain)/real(Gain));
    end
```

The phase angle between input and output is π when the imaginary part of the transfer function becomes 0 and the real part negative i.e. at $\omega = \sqrt{11} rad.s^{-1}$

Therefore, required frequency for π phase difference $= 0.528 Hz$

The numerator did not have any role in this calculation.