EE3004 : Assignment 5 Report

Varun Sundar, EE16B068

November 7, 2018

1 Install Instructions

We use *ipython* notebooks for the following assignment. The repository includes the relevant pip requirements.

Ensure python 3.6 is installed, and run:

```
pip3 install -r requirements.txt
jupyter lab
```

This should open a browser window with the code snippets.

2 Question 1

```
In []: import numpy as np
        import math
        import scipy
        from control.matlab import *
        import matplotlib.pyplot as plt
        plt.style.use('ggplot')
        pie=np.pi
```

We design for $M_p = 17$, $T_s = 3$

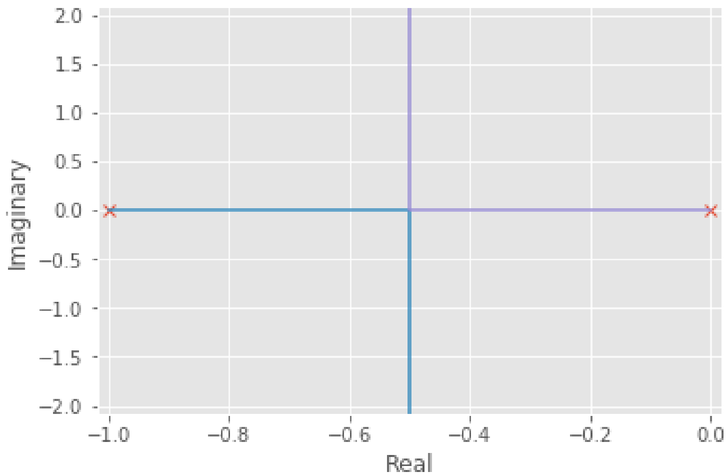
```
In []: real=-(4.0/Ts)
         imag=4*pie/(Ts*np.log(100/Mp))
         pole1=complex(real,imag)
         pole2=complex(real,imag)
         theta=180-np.angle(pole1,deg=True)
         wn=np.abs(pole1)
         print (pole1,pole2)
         print(theta,wn)
```

```
(-1.3333333333333333+2.3639346657109117j) (-1.3333333333333333+2.3639346657109117j)
60.57563766283309 2.714031112851792
```

```
In []: (num,den)=zpk2tf([],[0,-1],1)
       H=tf(num,den)
       print(H)
```

```
   1
-------
s^2 + s
```

```
In []: rlocus(H)
```




```
In []: anl=np.angle(evalfr(H,pole1),deg=True)
       print (anl)
```

```
142.54939989916585
```

2.1 Part 3

Clearly this doesnot lie on the root locus, hence we need a lead compensator. We calculate ϕ, the necessary angle difference. In addition we calculate θ the positive angle made by the dominant pole, by either $arccos(\zeta)$ or from the angle directly.

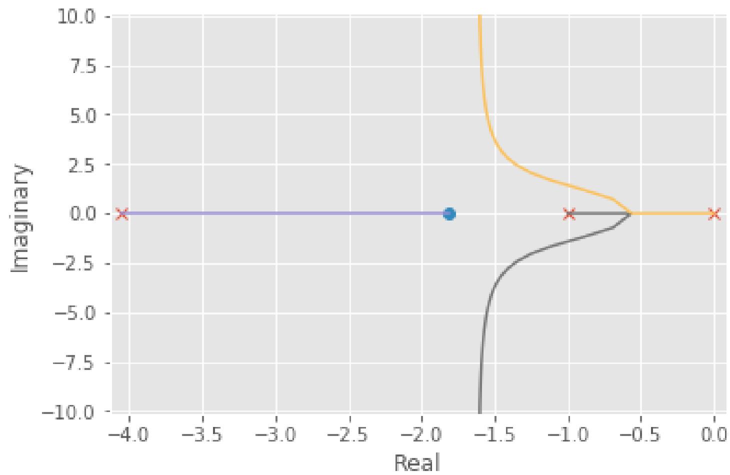
```
phi=-180+th
print(f"Phi {phi}")
theta=180-np.angle(pole1,deg=True)
print(f"Theta {theta}")
gamma=0.5*(180-theta-phi)
print(f"Gamma {gamma}")
alpha=np.sin(gamma*pie/180)*np.sin((theta+gamma+phi)*pie/180)/np.sin((theta+gamma)*pie/
print(f"Alpha {alpha}")
zc=wn*np.sin(gamma*pie/180)/np.sin((theta+gamma)*pie/180)
pc=zc/alpha
print (f"zc {zc} pc {pc}")
```

```
Phi 37.45060010083418
Theta 60.57563766283309
Gamma 40.986881118166366
Alpha 0.44819287447848166
zc 1.8169680716950904 pc 4.053986966681072
```

In []:
```python
Hnew = H * tf([1,zc],[1,pc])
Hnew
```

Out[49]:
```
                s + 1.817
        -------------------------
        s^3 + 5.054 s^2 + 4.054 s
```

In []:
```python
rlocus(Hnew);
```



In []:
```python
mag = np.abs(evalfr(Hnew,pole1))
phase = np.angle(evalfr(Hnew,pole1),deg=True)
print (f"Phase at pole {phase}")
K=1.0/mag
print(f"K {K}")
Hlead=K*Hnew
print(f"Hlead compensator only {Hlead}")
```

```
Phase at pole -180.0
K 9.678165381551548
Hlead compensator only
     9.678 s + 17.58
-------------------------
s^3 + 5.054 s^2 + 4.054 s
```

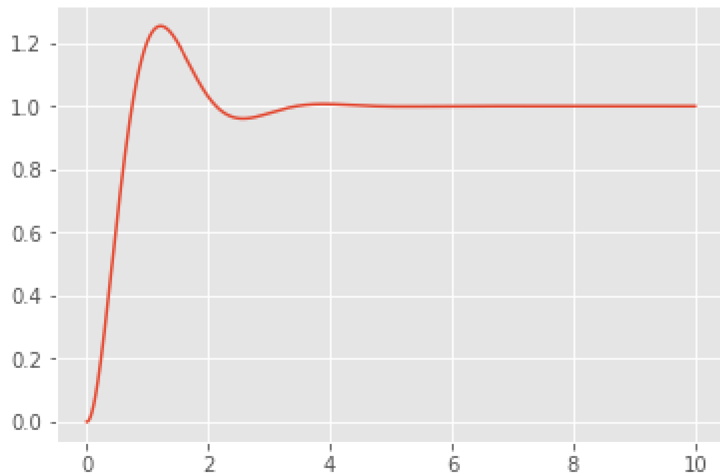
In [ ]:
```
Kv=K*zc/pc
print (Kv)
```

```
4.337684762035719
```

2.2 Plot with only lead compensator

In [ ]:
```
T = linspace(0,10,1000)
(y,T)=step(feedback(Hlead),T)
plt.plot(T,y)
```

Out[54]:
```
[<matplotlib.lines.Line2D at 0x1c15f30b00>]
```



In [ ]:
```
stepinfo(T,y)
```

```
OS 25.39881214713917 %
Tr 0.6706706706706707
Ts 2.032032032032032
```

2.3 $\alpha * \beta = 1$

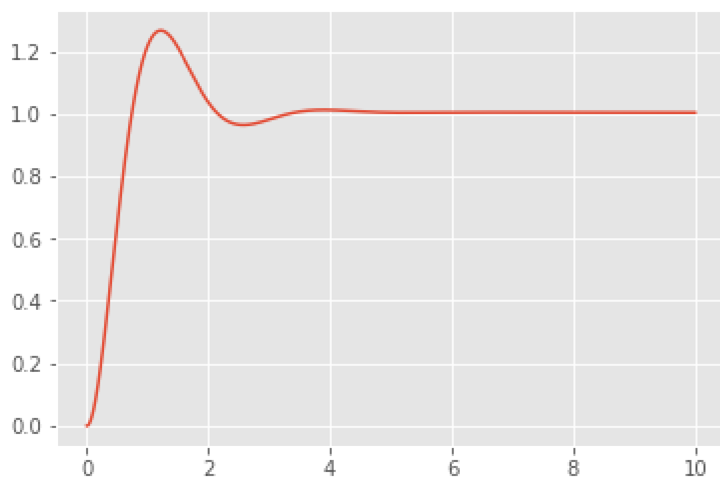
In [ ]:
```
beta=1.0/alpha
zg=0.05
pg=zg/beta
Hlag=tf([1,zg],[1,pg])
Hleadlag=Hlead*Hlag
Hleadlag
```

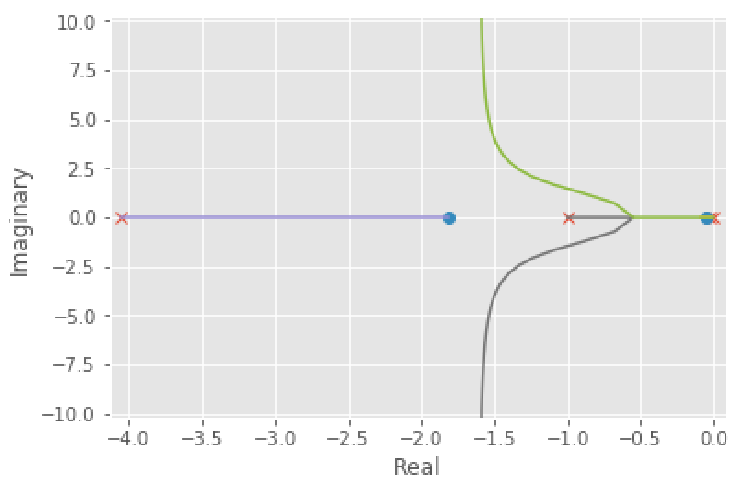
Out[56]:

```
        9.678 s^2 + 18.07 s + 0.8792
---------------------------------------
s^4 + 5.076 s^3 + 4.167 s^2 + 0.09085 s
```

In []:
```
T=linspace(0,10,1000)
(y,T)=step(Hleadlag/(1+Hleadlag),T)
plt.plot(T,y)
stepinfo(T,y);
print ("Kv:",Kv*beta)
rlocus(Hleadlag);
```

```
OS 26.16900489320202 %
Tr 0.6706706706706707
Ts 2.042042042042042
Kv: 20.0
```





2.4 Part 4, 6 Independent α, β

We set $\beta = Kv_{desired} / Kv_{obtained}$, when obtained from the lead compensator.

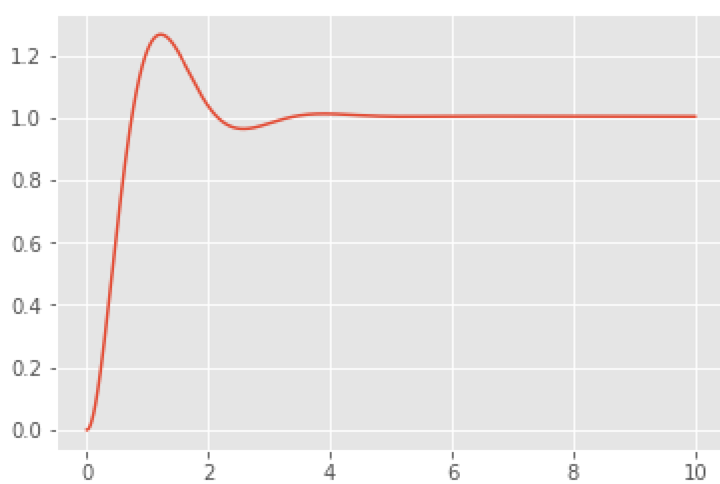
```
In []: Kv_desire=20
        beta=Kv_desire/Kv
        zg=0.05
        pg=zg/beta
        H_lag=tf([1,zg],[1,pg])
        Hlead_lag=Hlead*Hlag
        print(Hlead_lag)
```

```
      9.678 s^2 + 18.07 s + 0.8792
---------------------------------------
s^4 + 5.076 s^3 + 4.167 s^2 + 0.09085 s
```

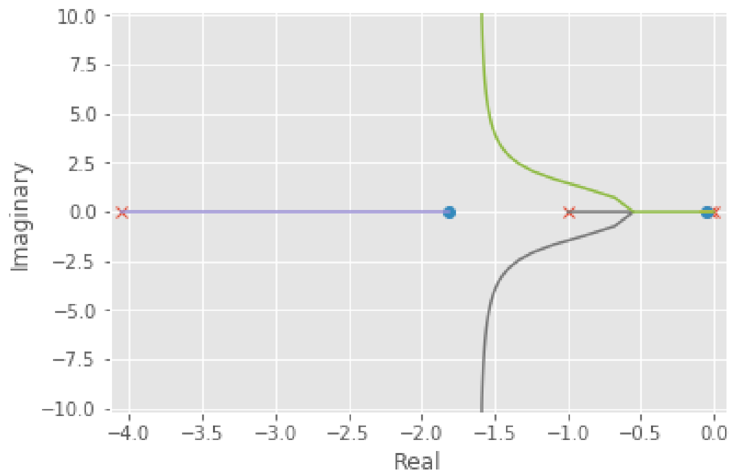
We plot the step response and the root locus of the above system.

```
In []: T=linspace(0,10,1000)
        (y,T)=step(feedback(Hlead_lag),T)
        plt.plot(T,y)
        stepinfo(T,y);
```

```
OS 26.16900489320171 %
Tr 0.6706706706706707
Ts 2.042042042042042
```



```
In []: rlocus(Hleadlag);
```



3 Question 3

```
In []: import numpy as np
          import scipy
          import matplotlib.pyplot as plt
          import control
          import math
          plt.style.use('ggplot')
          from sympy.solvers import solve
          from sympy import Symbol
          import sympy as sp
```

We define a few helper functions ...

```
In []: def get_params(Mp,ts):
              '''
              Mp in percent
              '''

              x = np.log(Mp/100)**2
              zeta = x / (x + np.pi**2)
              zeta = np.sqrt(zeta)

              wn = 4/ts/zeta
```

```
    return zeta, wn

def polar(z):
    '''
    Get polar representation
    '''
    a= z.real
    b= z.imag
    r = math.hypot(a,b)
    theta = math.atan2(b,a)
    return r,theta # use return instead of print.

def get_poles(zeta,wn):
    '''
    Get poles f4om
    '''
    return (complex(-zeta*wn, wn*np.sqrt(1-zeta**2)), complex(-zeta*wn ,- wn*np.sqrt(1

def get_alpha(gamma, theta, phi):
    '''
    Returns alpha from gamma.

    All angles in radians
    '''
    num = np.sin(gamma) * np.sin(theta + gamma + phi)
    denom = np.sin(theta + gamma) * np.sin(gamma + phi)
    return num/denom

def get_zc(wn, gamma, theta):
    '''
    Get zc
    '''
    num = wn* np.sin(gamma)
    denom = np.sin(theta + gamma)
    return num/denom
```

We design for $M_p = 5$ %, and T_s is 0.02s, K_v is 1%. And we shall test if the design satisifies for T_r.

```
In []: Mp, ts = 5, 0.02
       zeta,wn = get_params(Mp,ts)
       p1,p2 = get_poles(zeta,wn)
```

```
Zeta 0.6901067305598216 wn 289.8102440440741
Required dominant poles (-200.00000000000003+209.73787820249777j), (-200.00000000000003-209.7378
```

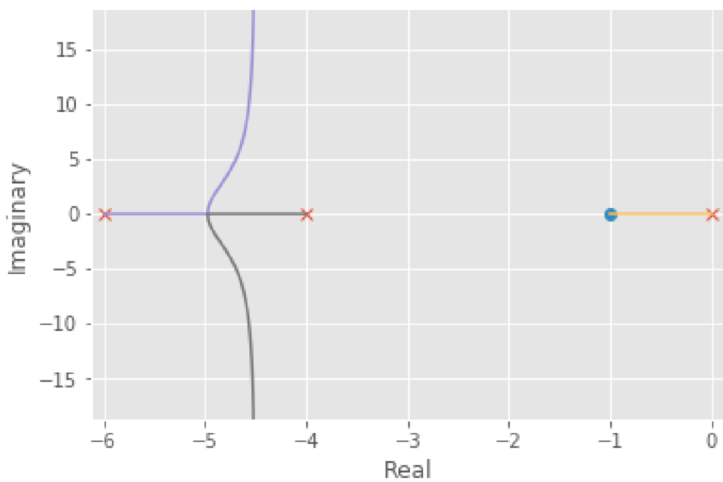
3.1 Part a

This is feasible if (-200.00 +/- 209.737j) lies on the root locus, as dominant poles.

```
In []: sys = control.TransferFunction([240,240],[1,10,24,0])
```

```
System open loop is
    240 s + 240
-------------------
s^3 + 10 s^2 + 24 s
```

```
In []: control.root_locus(sys);
```



Hence choice 1 is not feasible.

3.2 Part b

We first analyse the existent system, understand if the transient parameters need improvement or the steady state. We then desgin appropriately, with the dominant pole assumption.

With [U+FFFD][U+FFFD] $G_c(s) = K_c\alpha * (1 + Ts)/(1 + \alpha * Ts)$.

Needed K_v is 100 (1% error). Current K_v is 10.

Hence, $K_c * \alpha = 10$.

We determine θ, angle of compensation needed at (-200, +/- 209.73)

```
In []: polar(control.evalfr(sys, complex(200,-209.73)))[1]

Out[179]: 1.596108226418121

In []: np.pi - 1.596108226418121

Out[180]: 1.5454844271716721
```

We need -88.54 deg as compensation.
Besides this, we need to meet:

- Angle sum = π
- Magnitude at roots = 1
- $K_c * \alpha = 10$

We treat this as a lead compensation case.

```
In []: phi = 85.97404456267036 * np.pi/180
        theta = np.arccos(zeta)
        gamma = (1.596108226418121 - theta)/2
        print(f"Gamma {gamma * 180/np.pi}")
        alpha = get_alpha(gamma = gamma, phi = phi, theta = theta)
        print(f"Alpha {alpha}")
        temp_sys = alpha * control.TransferFunction([T,1],[T*alpha, 1]) * sys
        Kc = 1/polar(control.evalfr(temp_sys,complex(-200,209.73)))[0]
        print(f"Kc {Kc}")
```

```
Gamma 22.54441156486525
Alpha 0.183975416417848
Kc 752.8216104925237
```

To determine T, we use the magnitude condition.

```
In []: cur_mag = polar(control.evalfr(sys, complex(-200,209.73)))[0]
        print(f"Current magnitude without compensator at requisite roots {cur_mag}")
```

```
Current magnitude without compensator at requisite roots 0.002919450489681809

Z_c 119.09350502020902
T 0.00839676353324482

Solved parameters :

Alpha 0.183975416417848
Kc 54.355088275967454
T 0.00839676353324482
```

3.3 Part 3

We perform a grid search, while looking for the following:

- Angle sum = π
- Magnitude at roots = 1
- $K_c * \alpha = 10$

We keep varying α and T.
We try:

- grid search
- gradient descent

```
In []: aa = np.arange(0.1,2.0,(0.2-0.1)/1000)
       tt = np.arange(0, 0.01, 0.01/100)
       ll = [] # loss array

       def loss1(a,t, pole = complex(-200,209.73)):
           '''
           Angle compensation
           '''
           diff = polar(1 + t*pole)[1] - polar(1 + a*t*pole)[1] - 1.5454844271716721
           diff = np.abs(diff)
           return diff

       def loss2(a,t, pole = complex(-200,209.73)):
           '''
           Magnitude compensation
           '''
           diff = polar(1 + t*pole)[0] / polar(1 + a*t*pole)[0] - 1/(10*cur_mag)
           return np.abs(diff)

       def loss(a,t, pole = complex(-200,209.73)):
           return loss1(a,t, pole) + 0*loss2(a,t,pole)

       l_min = None
       a_min = None
       t_min = None

       for a in aa:
           for t in tt:
               l = loss(a,t)
               if not l_min or (l_min > l):
                   l_min = l
                   a_min = a
                   t_min = t
               ll.append(loss)

       print(f"Loss {l_min} a_min {a_min} t_min {t_min}")
       print(f"Angle loss {loss1(a_min,t_min)}")
       #print(f"Mag loss {loss2(a_min, t_min)}")
```

```
Loss 1.884514472694221e-06 a_min 0.16690000000000194 t_min 0.0077
Angle loss 1.884514472694221e-06
Mag loss 32.09852385559125
```

3.4 Part 4,5 with Part 2

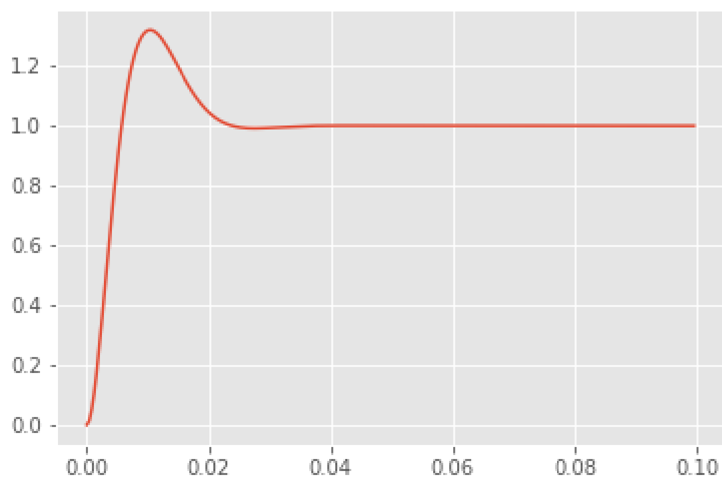
Clearly, we have used a lead compensator.

The compensated transfer function is:

```
Compensator
1.163 s + 138.5
---------------
0.001545 s + 1

Final Sys
    279.1 s^2 + 3.352e+04 s + 3.324e+04
-------------------------------------------
0.001545 s^4 + 1.015 s^3 + 10.04 s^2 + 24 s
```

The transient responses associated with this are:

```
In []: tt, yy = control.step_response(control.feedback(comp_sys), T = np.arange(0,0.1,0.04/100))
         plt.plot(tt,yy)
```



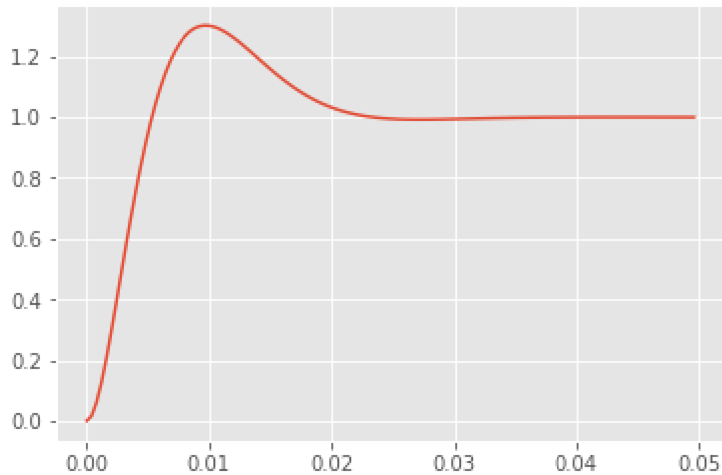
3.5 Part 4,5 with Part 3

Clearly, we have used a lead compensator.
The compensated transfer function is:

```
Compensator
1.224 s + 159
--------------
0.001285 s + 1

Final Sys
    293.8 s^2 + 3.845e+04 s + 3.816e+04
```

```
-------------------------------------------
0.001285 s^4 + 1.013 s^3 + 10.03 s^2 + 24 s
```

```
In []: tt, yy = control.step_response(control.feedback(comp_sys2), T = np.arange(0,0.05,0.04/100
```



Which has a much smaller T_s. We also note that in the above two cases, no lag compensation is needed, since the K_v obtained is good enough.

4 Question 4

```
In [2]: import numpy as np
        import math
        import scipy
        from control.matlab import *
        import control as cont
        import matplotlib.pyplot as plt
        pie=np.pi
```

Stepinfo function which gives all the useful information like overshoot, rise time and settle time from the response function

```
In [23]: def stepinfo(t,yout):
             print ("Overshoot",(yout.max()/yout[-1]-1)*100,'%')
             print ("Time rise",t[next(i for i in range(0,len(yout)-1) if yout[i]>yout[-1]*.90)]
             print ("Time settle",t[next(len(yout)-i for i in range(2,len(yout)-1) if abs(yout[-

In [24]: M=10
         ts=2
```

Calculating tau and wn from formulas

$$M = 100e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}}$$

$$ts = \frac{4}{\zeta\omega_n}$$

In [25]:
```
tau=np.sqrt(np.log(M/100)**2/(np.log(M/100)**2+pie**2))
wn=4/(tau*ts)
```

Calculating gamma and omega from the formulas

$$\gamma = \tan^{-1}\frac{2\zeta}{\sqrt{\sqrt{1+4\zeta^4}-2\zeta^2}}$$

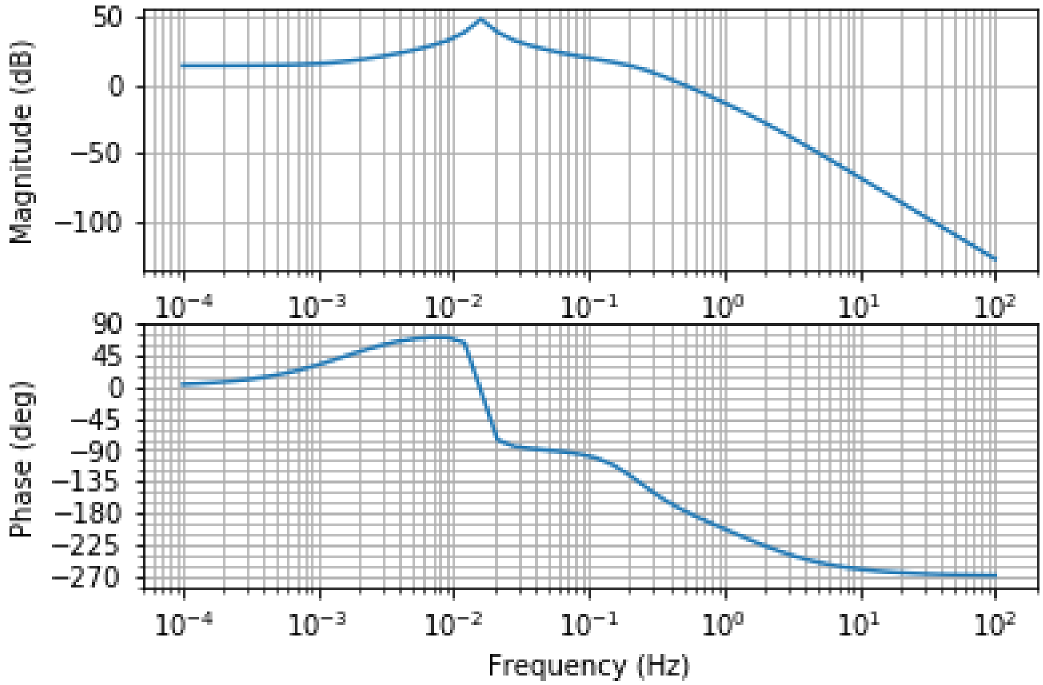
$$\omega_{gc} = \omega_n\sqrt{\sqrt{1+4\zeta^4}-2\zeta^2}$$

In [26]:
```
gamma=np.arctan(2*tau/np.sqrt(np.sqrt(1+4*tau**4)-2*tau**2))*180/pie
wgc=wn*np.sqrt(np.sqrt(1+4*tau**4)-2*tau**2)
print(f"Gamma {gamma} wgc {wgc}")
print(f"w natural {wn}")

(num,den)=zpk2tf([-1,-0.01],[-10],100)
den1=tf([1],[1,2,2])
den2=tf([1],[1,0.02,0.0101])
H=tf(num,den)
H=H*den1*den2
print(f"Transfer function {H}")
```

```
Gamma 58.59306826496366 wgc 2.4422747101330673
w natural 3.38320725639016
Transfer function
                    100 s^2 + 101 s + 1
-----------------------------------------------------------
s^5 + 12.02 s^4 + 22.25 s^3 + 20.56 s^2 + 0.6222 s + 0.202
```

4.1 Bode Plot of H

In [27]:
```
bode(H);
```

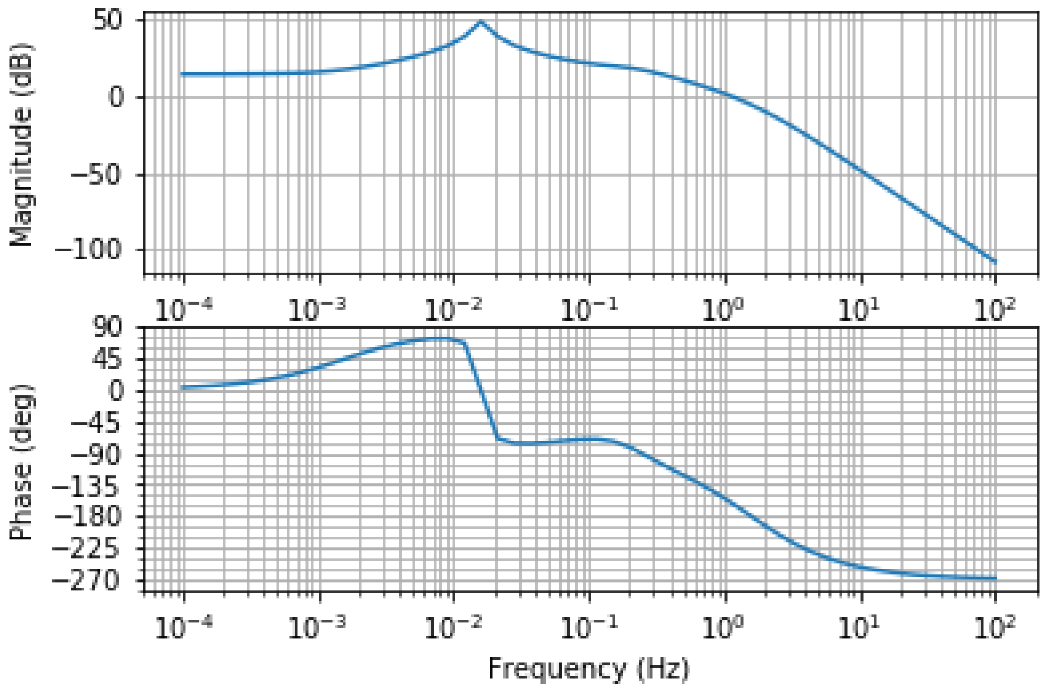



```
In [28]: [gm,pm,wg,wm]=margin(H)
         print (f"gain-margin {gm} phase-margin {pm} \ngain cross over freq {wg} phase cross ove

         phi=gamma-pm
         wm=wm
         print(f"\nPhi {phi} w_m {wm}")

         alpha=(1-np.sin(phi*pie/180))/(1+np.sin(phi*pie/180))
         print(f"\nAlpha {alpha}")

         T=1/np.sqrt(alpha)/wm
         print(f"T {T}")

         Gc=tf([T,1],[alpha*T,1])
         Hnew=H*Gc
         print(f"\nHnew {Hnew}")
```

```
gain-margin 1.214662566961802 phase-margin 3.7857102810716015
gain cross over freq 3.4359780521729917 phase cross over freq 3.1344441272838512

Phi 54.80735798389206 w_m 3.1344441272838512

Alpha 0.1005828635112053
T 1.0059524889079905
```

```
Hnew
                        100.6 s^3 + 201.6 s^2 + 102 s + 1
------------------------------------------------------------------------------
0.1012 s^6 + 2.216 s^5 + 14.27 s^4 + 24.33 s^3 + 20.62 s^2 + 0.6426 s + 0.202
```

```
In [45]: bode(Hnew)
         plt.show()
         [gm,pm2,wg,wm2]=margin(Hnew)
         print(f"Phase margin {pm2} Gain margin {wg2}")
         print(f"Phi new {phi}")
```




```
Phase margin 20.823595702607207 Gain margin 6.246114286537816
Phi new 54.80735798389206
```

Since there is a huge difference between the phase margin and the desired value, we increment phi by 60 degrees

```
In [43]: phi_one=phi+60
         print(f"Phi altered {phi_one}")

         alpha_one=(1-np.sin(phi_one*pie/180))/(1+np.sin(phi_one*pie/180))
```

```
        print(f"Alpha {alpha_one}")

        mag10 = 20*np.log10(1/np.sqrt(alpha_one))
        print(f"decibels magnitude {mag10}")

        mag_one=np.sqrt(alpha_one)
        print(f"Magnitude descent needed to offset {mag_one}")
```

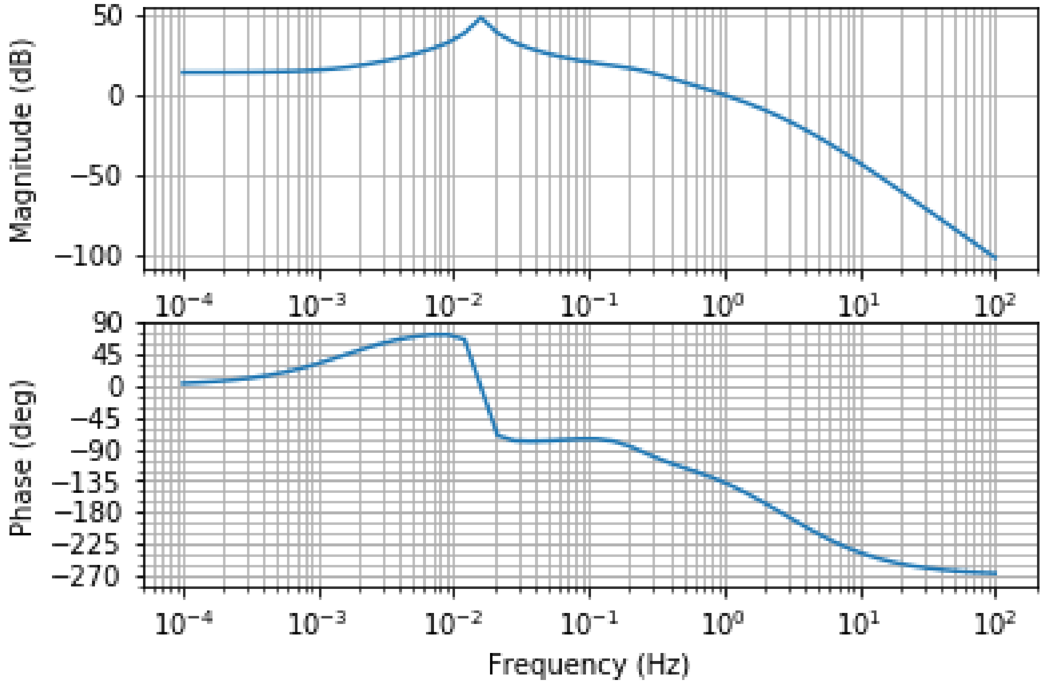
```
Phi altered 114.80735798389206
Alpha 0.04836989761311832
decibels magnitude 13.154248319025266
Magnitude descent needed to offset 0.21993157484344608
```

To calculate the frequency at which magnitude is square root of alpha, we find the gain crossover frequency of the transfer function H/square root of alpha

```
In [40]: [gm1,pm1,wm1,wg1]=margin(H/mag_one)
         print(f"Cross over {wg1}")

         T_1=1/np.sqrt(alpha_one)/wg1
         print(f"T_1 {T_1}")

         Gc_one=tf([T_1,1],[alpha_one*T_1,1])
         Hnew_one=Gc_one*H
         print(Hnew_one)
```

```
Cross over 6.246114286537809
T_1 0.7279515743095443

                      72.8 s^3 + 173.5 s^2 + 101.7 s + 1
-----------------------------------------------------------------------------
0.03521 s^6 + 1.423 s^5 + 12.8 s^4 + 22.97 s^3 + 20.58 s^2 + 0.6293 s + 0.202
```

```
In [41]: bode(Hnew_one)
         plt.show()
         [gm2,pm2,wm2,wg2]=margin(Hnew_one)
         print(f"Phase margin {pm2} Gain margin {wg2}")
```

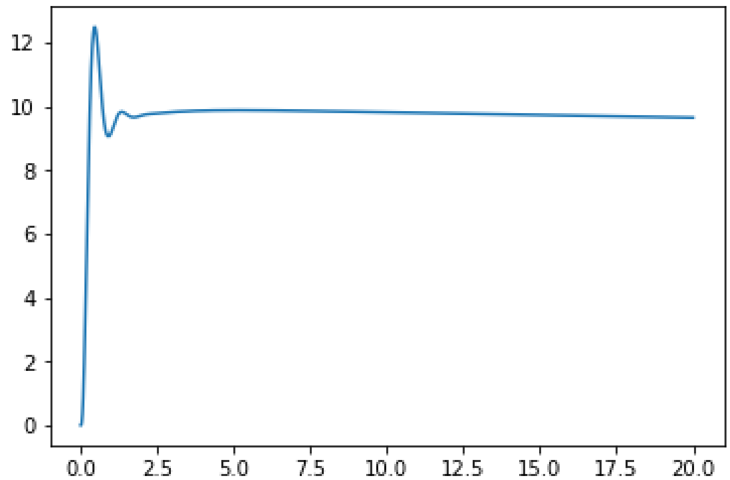


Phase margin 42.84878119640814 Gain margin 6.246114286537816

4.2 Step Response of Closed Loop System

```
In [42]: T, yout = cont.step_response(feedback(Hnew_one,1), np.linspace(0,20,10000))
         plt.plot(T,yout*10)
         stepinfo(T,yout*10)
```

```
Overshoot 29.495260712314074 %
Time rise 0.266026602660266
Time settle 8.586858685868586
```


12
10
8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0