HW9

October 22, 2022

Group number
3

Group members
Alexander Stoustrup
Mathias Tyranski
Benjamin Simonsen

[65]:
```
import numpy as np
from scipy.optimize import minimize
import scipy.signal as si
import sympy as sp
import control as ct
from typing import List
from sympy.plotting import plot
import matplotlib.pyplot as plt
from IPython.display import display, Latex, Math, Image
%matplotlib inline

def eq_disp(varstring, expr, unit=""):
    display(Latex(f"${varstring}={sp.latex(expr)} \: {unit}$"))

def reduce_feedback(G_fwd, G_bwd):
    """Assumes feedback is deducted from signal, if not
    change sign of feedback"""
    return sp.simplify(G_fwd/(1+G_fwd*G_bwd))

def RHarray(coeffs: List):
    # first 2 rows from coefficients
    n = len(coeffs)
    arr = sp.zeros(n, n//2+2)
    i = 0
    for i in range(0,n,2):
        arr[0, i//2] = coeffs[i]
    for i in range(1,n,2):
        arr[1, i//2] = coeffs[i]

    for j in range(2, arr.shape[0]):
        for i in range(arr.shape[1]-1):
            a0 = arr[j-2,0]
            a3 = a1 = arr[j-1,i+1]
            a1 = arr[j-1,0]
            a2 = arr[j-2,i+1]
            arr[j, i] = (a1*a2-a0*a3)/a1
    return arr
```

1 P9.21

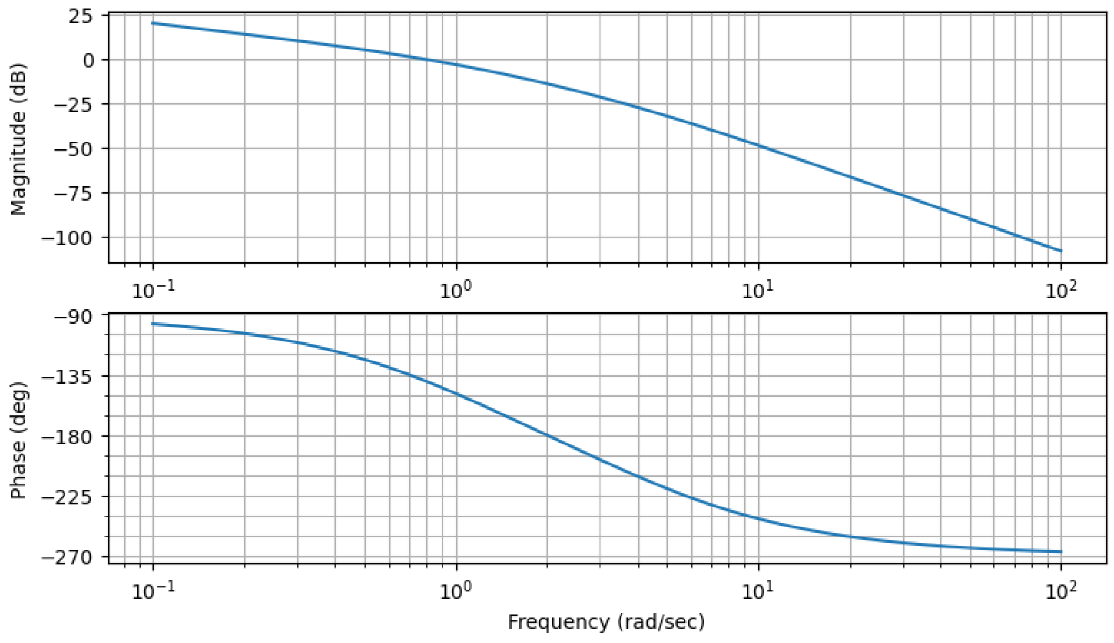
[66]:
```
K, s = sp.symbols('K, s')
L = K/(s*(s+1)*(s+4))
eq_disp('L(s)', L)
T = reduce_feedback(L, 1)
eq_disp('T(s)', T)
```

$L(s) = \frac{K}{s(s+1)(s+4)}$

$T(s) = \frac{K}{K+s(s+1)(s+4)}$

1.1 a) Bode plot

[67]:
```
K = 4
s = ct.tf('s')
sys = K/(s*(s+1)*(s+4))

fig, ax = plt.subplots(figsize=(9, 5))
mag, phase, omega = ct.bode(sys, dB=True)
```



1.2 b) Gain margin

Rewriting the loop transfer function to the frequency domain on the following form to be able to extract τ_1 and τ_2

$$L(j\omega) = \frac{K}{j\omega\,(j\omega\tau_1 + 1)\,(j\omega\tau_2 + 1)}$$

[68]:
```
K, s, j, omega = sp.symbols('K, s, j, omega')
L = K*(1/4)/(j*omega*(j*omega+1)*(j*omega/4+1))
eq_disp('L(s)', L)
```

$L(s) = \frac{0.25K}{j\omega\left(\frac{j\omega}{4}+1\right)(j\omega+1)}$

With

[69]:
```
tau1 = 1/4
tau2 = 1
K = 4
eq_disp('\\tau_1', tau1)
eq_disp('\\tau_2', tau2)
```

$\tau_1 = 0.25$

$\tau_2 = 1$

Evaluating the gain margin by

$$G.M. = 20log\left(\left[\frac{K\tau_1\tau_2}{\tau_1+\tau_2}\right]^{-1}\right) \tag{1}$$

[70]:
```
GM_K4 = (20*sp.log((K*0.25*tau1*tau2/(tau1 + tau2))**(-1), 10)).evalf()
eq_disp('G. M.(K=4)', round(GM_K4,1), 'dB')
```

$G.M.(K = 4) = 14.0\,dB$

1.3 c) K providing gain margin of 12 dB

Eq. 1 is solved for K with a gain margin of $12dB$

$$12dB = 20log\left(\left[\frac{K\tau_1\tau_2}{\tau_1+\tau_2}\right]^{-1}\right)$$

[71]:
```
K = sp.symbols('K')
K_12 = sp.solve(20*sp.log((K*0.25*tau1*tau2/(tau1 + tau2))**(-1), 10) - 12,␣
 ↪K)[0]
eq_disp('K(G.M.=12 dB)', round(K_12, 1))
```

$K(G.M. = 12dB) = 5.0$

1.4 d) K for steady-state error of 25 % of the magnitude A for ramp input

```
[72]: K, s, A = sp.symbols('K, s, A')
      L = K/(s*(s+1)*(s+4))
      eq_disp('L(s)', L)
```

$L(s) = \frac{K}{s(s+1)(s+4)}$

```
[73]: R = A*1/s**2
      E = sp.simplify(R/(1 + L))
      eq_disp('E(s)', E)
```

$E(s) = \frac{A(s+1)(s+4)}{s(K+s(s+1)(s+4))}$

$$e_{ss} = \lim_{s \to 0} sE(s)$$

```
[74]: e_ss = sp.limit(sp.simplify(s*E), s, 0)
      eq_disp('e_{ss}', e_ss)
```

$e_{ss} = \frac{4A}{K}$

With

$$e_{ss} = 0.25A$$

```
[75]: K_25p = sp.solve(e_ss - 0.25*A, K)[0]
      eq_disp('K(e_{ss}=0.25A)', K_25p)
```

$K(e_{ss} = 0.25A) = 16.0$

1.4.1 Checking if this K results in a stable system

Finding the coefficients of the denominator

```
[76]: p, q = T.as_numer_denom()

      coeffs = sp.Poly(q, s).coeffs()
      eq_disp('q(s)', sp.Poly(q, s))
      for i, k in enumerate(coeffs):
          display(Latex(f"${f's^{len(coeffs)-1-i}'}: {sp.latex(k)}$"))
```

$q(s) = \operatorname{Poly}\left(s^3 + 5s^2 + 4s + K, s, domain = \mathbb{Z}[K]\right)$

$s^3 : 1$

$s^2 : 5$

$s^1 : 4$

$s^0 : K$

Finding Routh-Hurwitz array

[77]:
```
arr = RHarray(coeffs)
arr
```

[77]:
$$\begin{bmatrix} 1 & 4 & 0 & 0 \\ 5 & K & 0 & 0 \\ 4 - \frac{K}{5} & 0 & 0 & 0 \\ K & 0 & 0 & 0 \end{bmatrix}$$

So when the two roots lie on the imaginary axis the gain is

[78]:
```
K_mstable = sp.solve(arr[2,0], K)[0]
eq_disp('K', K_mstable)
```

$K = 20$

For $K < 20$ the system will be stable so this gain can be utilized and achieve acceptable performance

2 AP9.11

[79]:
```
s = sp.symbols('s')
t = sp.symbols('t', positive=True)
T_delay = 1 #s
PO_max = 0.1
Gc = 5/(s*(s + 10))
G = sp.exp(-s*T_delay)
T = reduce_feedback(Gc*G, 1)
eq_disp('G_c(s)', Gc)
eq_disp('G(s)', G)
eq_disp('T(s)', T)
```

$G_c(s) = \frac{5}{s(s+10)}$

$G(s) = e^{-s}$

$T(s) = \frac{5}{s(s+10)e^{s}+5}$

Ignoring the time delay

[80]:
```
T_ndelay = reduce_feedback(Gc, 1)
eq_disp('T(s)', T_ndelay)
```

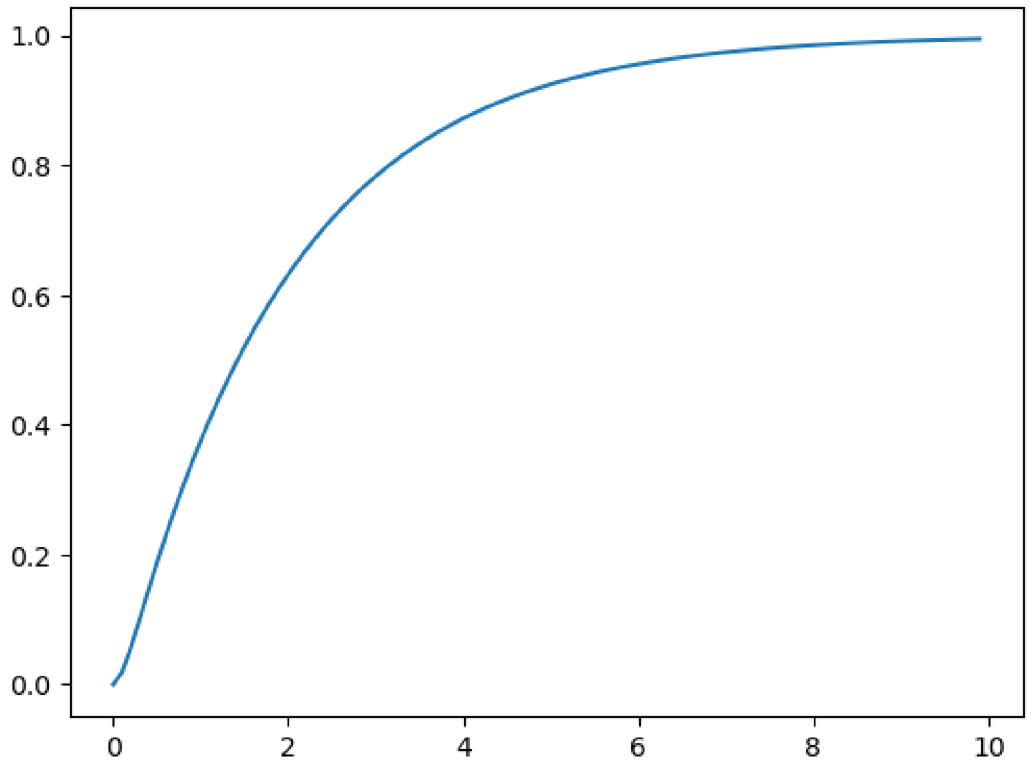
$T(s) = \frac{5}{s(s+10)+5}$

[81]:
```
R = 1/s
Y_ndelay = R*T_ndelay
y_ndelay = sp.inverse_laplace_transform(Y_ndelay, s, t)
eq_disp('y(t)', y_ndelay)
```

$$y(t) = \frac{\left(-(2+\sqrt{5})e^{4\sqrt{5}t} + 4e^{t(2\sqrt{5}+5)} - 2 + \sqrt{5}\right)e^{-t(2\sqrt{5}+5)}}{4}$$

```
[82]: y_plot = sp.lambdify(t, y_ndelay)
      t_span = np.r_[0:10:0.1]
      plt.plot(t_span, y_plot(t_span))
```

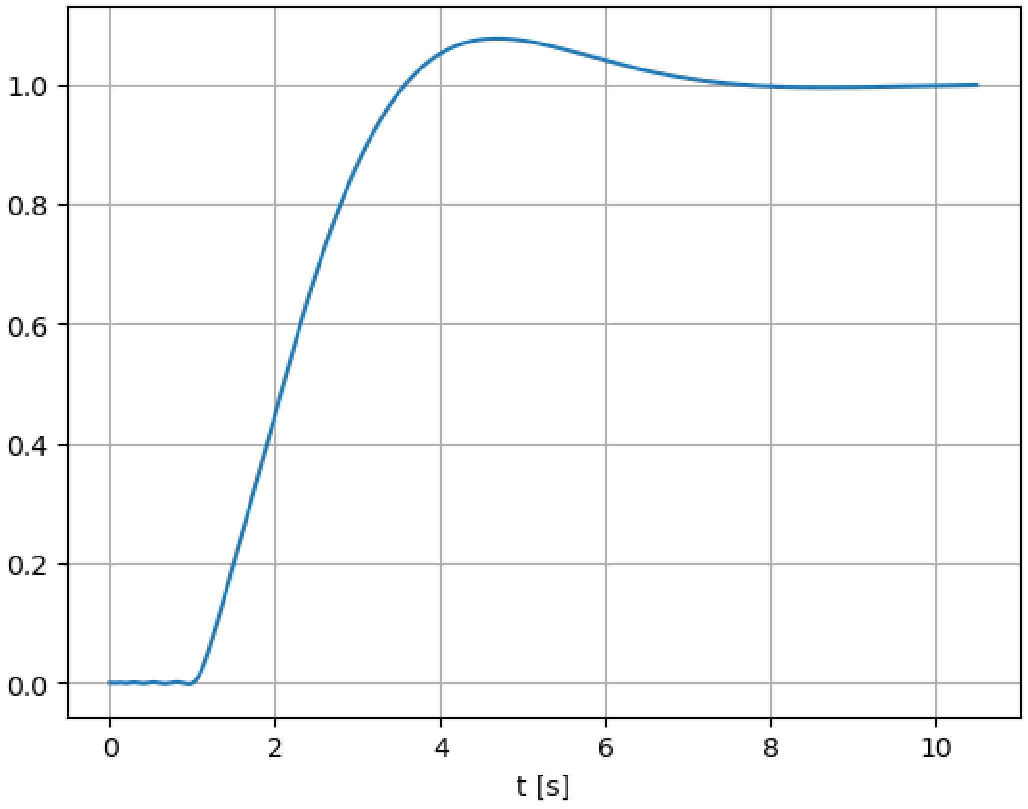
```
[82]: [<matplotlib.lines.Line2D at 0x1c60586bb20>]
```



With the time delay:

```
[83]: # Make padé approximation of time delay
      # 10th order approximation
      n_pade = 10
      num_pade, den_pade = ct.pade(T_delay, n_pade)
      H_pade = ct.tf(num_pade, den_pade)

      s = ct.tf('s')
      Gc = 5/(s*(s + 10))

      # Connect Gc and time delay in series to get loop function
      L = ct.series(Gc, H_pade)
```

```python
# Reduce feedback loop
T = ct.feedback(L, 1)

# Get step response
t, y = ct.step_response(T)

# Plot
plt.plot(t, y)
plt.xlabel('t [s]')
plt.grid ()
```



The steadu state error is approximated

[84]:
```python
e_ss = 1 - y[-1]
eq_disp('e_{ss}', round(e_ss,2))
```

$e_{ss} = 0.0$

The percent overshoot

$$P.O. = \frac{y(T_p) - y(\infty)}{y(\infty)}$$

is found

[85]:
```
PO = (np.max(y) - 1)/1
eq_disp('P.O.', round(PO,3)*100, '\\%')
```

$P.O. = 7.6\,\%$

So the percent overshoot and steady state error specifications are satisfied.

Next, we find the gain margin, phase margin and cut-off frequencies of each of them. The time delay results in a phase shift so we find the margins for the controller transfer function

[86]:
```
gm, pm, wcg, wcp = ct.margin(Gc)
```

We get the phase margin and the cut-off frequency for the phase margin

[87]:
```
eq_disp('\\phi_{pm}', round(pm,1), '^\\circ')
eq_disp('\\omega_{cp}', round(wcp,2), 'rad/s')
```

$\phi_{pm} = 87.1\,^\circ$

$\omega_{cp} = 0.5\,rad/s$

The critical time delay is

$$T_c = \frac{\phi_{pm}}{\omega_{cp}}$$

[88]:
```
T_c = pm/(wcp*180/np.pi)
eq_disp('T_c', round(T_c,4))
```

$T_c = 3.0456$

So

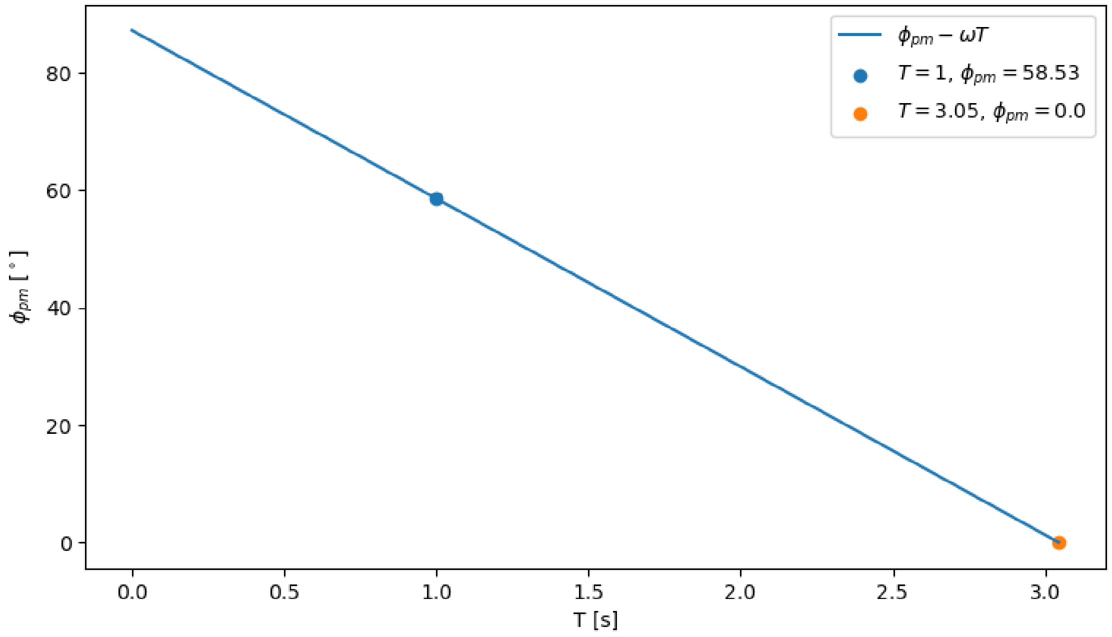
$$T \le T_c = \frac{\phi_{pm}}{\omega_{cp}}$$

The time delay e^{-sT} results in a phase shift

$$\phi(\omega) = -\omega T$$

[89]:
```
phi = lambda T_d: T_d*wcp*180/np.pi
T_span = np.r_[0:5:T_c]

fig, ax = plt.subplots(figsize=(9, 5))
ax.plot(T_span, pm-phi(T_span))
ax.scatter(T_delay, pm-phi(T_delay))
ax.scatter(T_c, pm-phi(T_c))
ax.legend(['$\phi_{pm}-\omega T$', f'$T={round(T_delay,2)}$, ' + '$\phi_{pm}=$'␣
↪+ f'{round(pm-phi(T_delay),2)}', f'$T={round(T_c,2)}$, ' + '$\phi_{pm}=$' +␣
↪f'{round(pm-phi(T_c),2)}'])
ax.set_xlabel('T [s]')
ax.set_ylabel('$\phi_{pm}$ $[^\circ]$')
```

```
[89]: Text(0, 0.5, '$\\phi_{pm}$ $[^\\circ]$')
```

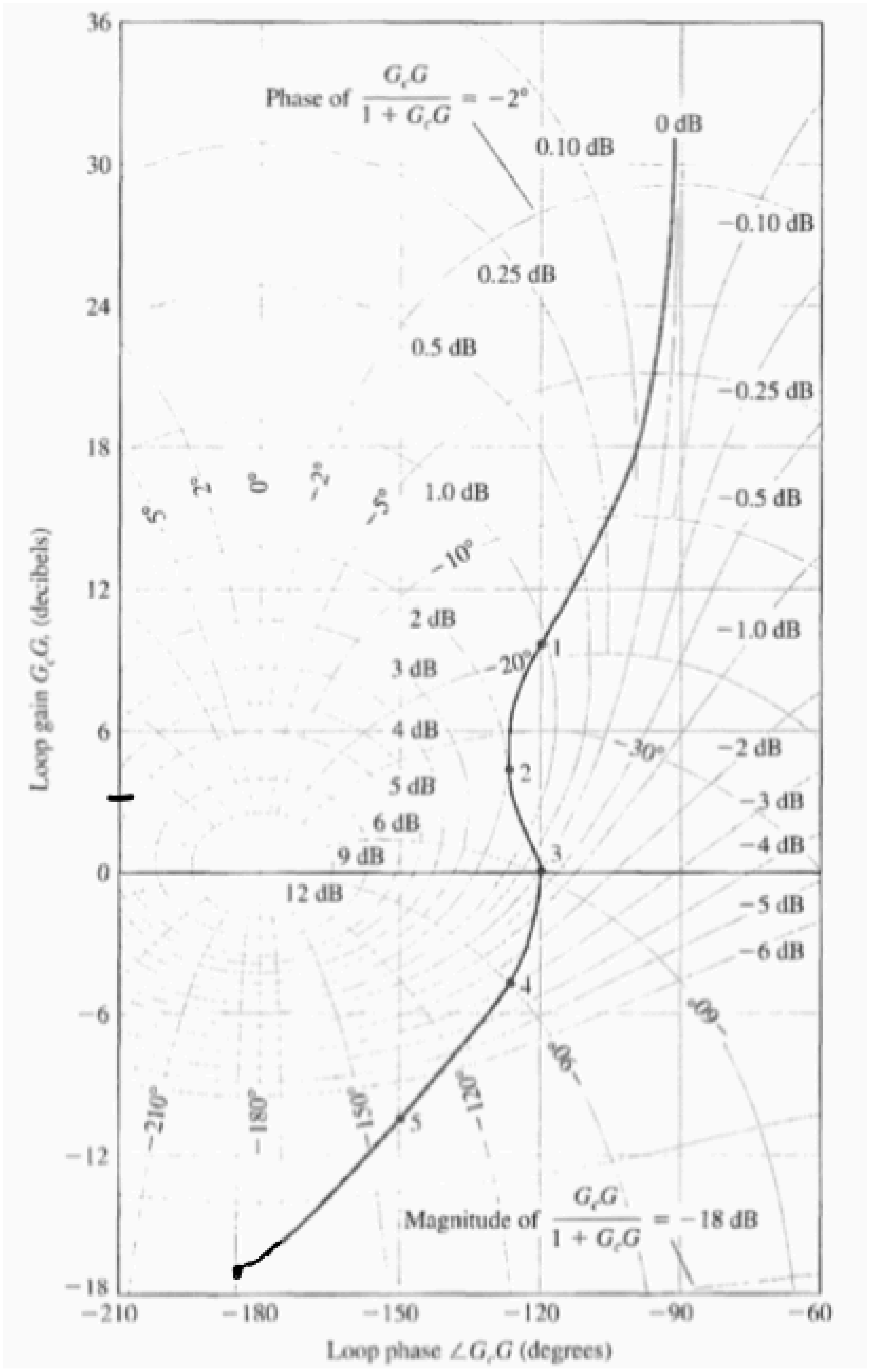


3 DP9.1

3.1 a)

```
[90]: Image('Gain and phase margin.png', width='400px')
```

```
[90]:
```



Phase of $\frac{G_cG}{1+G_cG} = -2°$
0 dB
0.10 dB
-0.10 dB
0.25 dB
0.5 dB
-0.25 dB
1.0 dB
-0.5 dB
5°
2°
0°
-2°
-5°
-10°
2 dB
-1.0 dB
3 dB
-20°
1
4 dB
-30°
-2 dB
2
5 dB
6 dB
-3 dB
9 dB
3
-4 dB
12 dB
-5 dB
-6 dB
4
-60°
-90°
-210°
-180°
-150°
-120°
5
Magnitude of $\frac{G_cG}{1+G_cG} = -18$ dB
36
30
24
18
12
6
0
-6
-12
-18
-210
-180
-150
-120
-90
-60
Loop gain G_cG (decibels)
Loop phase $\angle G_cG$ (degrees)

From the plot the gain and phase margins are read

[91]:
```
GM = 17
PM = 60
eq_disp('G.M.', GM, 'dB')
eq_disp('P.M.', PM, '^\\circ')
```

$G.M. = 17\,dB$

$P.M. = 60\,^\circ$

3.2 b)

The resonant peak is found at point 2 on the plot. Here the resonant peak and frequency are

[92]:
```
Mp = 2
omega_p = 5
eq_disp('M_p', Mp, 'dB')
eq_disp('\\omega_p', omega_p, 'rad/s')
```

$M_p = 2\,dB$

$\omega_p = 5\,rad/s$

3.3 c)

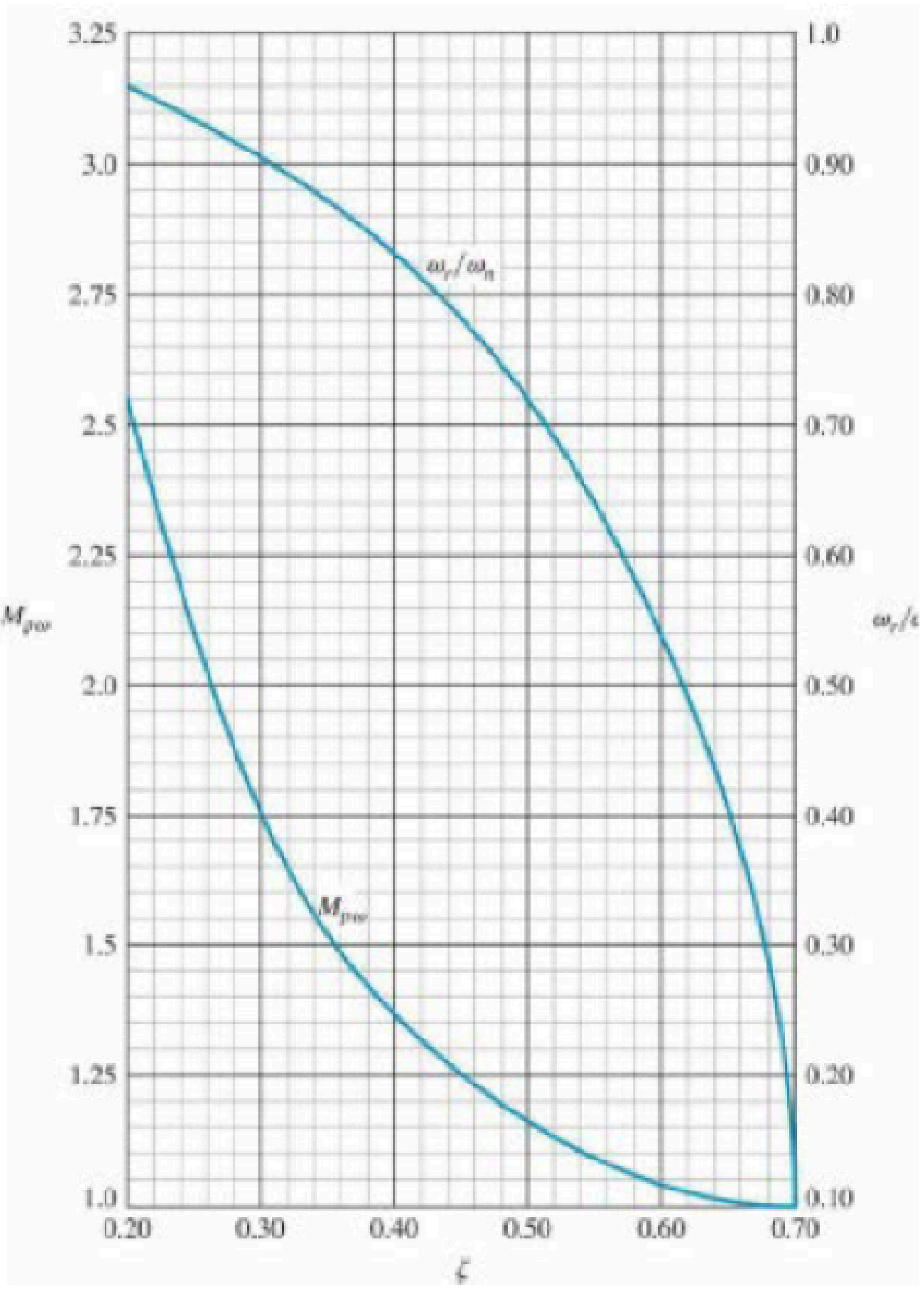
The bandwidth is at point 4, where the curve crosses the $-3dB$ curve

[93]:
```
omega_B = 20
eq_disp('\\omega_B', omega_B, 'rad/s')
```

$\omega_B = 20\,rad/s$

[94]:
```
Image('Resonant peak, frequency and damping ratio.png')
```

[94]:



Converting the resonant peak to linear scale

```
[95]: Mp_lin = 10**(Mp/20)
      eq_disp('M_p', round(Mp_lin,2))
```

$M_p = 1.26$

So we get the damping ratio and the ratio of the resonant frequency to the natural frequency

```
[96]: xi = 0.45
      wr_pr_wn = 0.69
      eq_disp('\\xi', xi)
      eq_disp('\\frac{\\omega_r}{\\omega_n}', wr_pr_wn)
```

$\xi = 0.45$

$\frac{\omega_r}{\omega_n} = 0.69$

So the natural frequency and thereby the settling time are

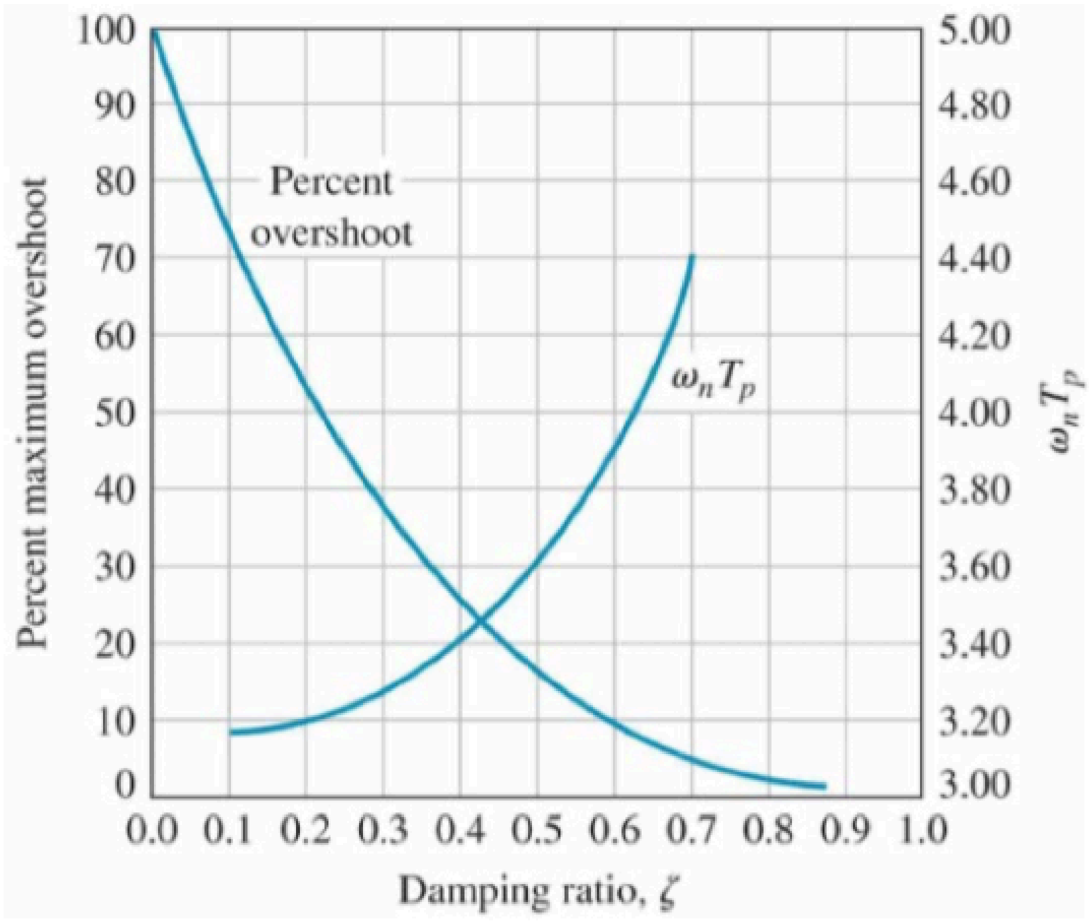
```
[97]: wn = omega_p/wr_pr_wn
      Ts = 4/(xi*wn)
      eq_disp('\\omega_n', round(wn,2), 'rad/s')
      eq_disp('T_s', round(Ts,2), 's')
```

$\omega_n = 7.25\, rad/s$

$T_s = 1.23\, s$

```
[98]: Image('PO.png')
```

```
[98]:
```



So the percent overshoot is

```
[99]: PO = 0.2
      eq_disp('P.O.', PO*100, '\\%')
```

$P.O. = 20.0\,\%$

3.4 d)

The P.O. of 30 % is equivalent to a damping ratio of

```
[100]: xi = 0.35
       xi
```

```
[100]: 0.35
```

```
[101]: Mp = 20*np.log10(1.5)
       Mp
```

```
[101]: 3.5218251811136247
```

The first plot shows $G_c(j\omega)G(j\omega)/K$. So we need to lower the gain by approximately 10 dB to land between the closed loop line of -3 and -4 dB.

4 DP9.9

5 a)

The transferfunction is given by

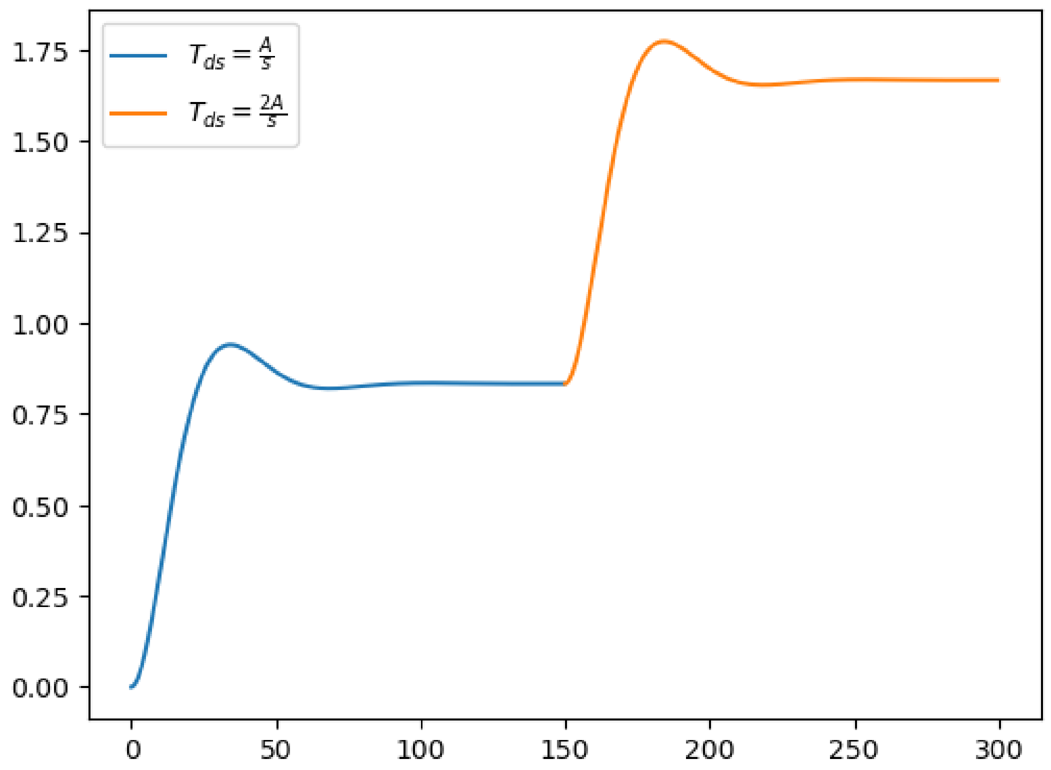
```
[102]: s, A, Kp, K1, T2d, T0, G1s, G2s, Gcs = sp.symbols(r's, A, K_p, K1, T_{2d}, T0,␣
         ↪G_1, G_2, G_c')
       tau1 = 10
       tau2 = 50
       G1 = 1/100/((tau1*s+1)*(tau2*s+1))
       G2 = 1/((tau1*s+1)*(tau2*s+1))
       Gc = Kp + K1/s
       T2 = reduce_feedback(Gcs*G1s,1)*T2d + reduce_feedback(1, Gcs*G1s)*G2s*T0
       eq_disp('T_2', T2)
```

$$T_2 = \frac{G_1 G_c T_{2d}}{G_1 G_c + 1} + \frac{G_2 T_0}{G_1 G_c + 1}$$

5.1 b)

Assuming $A = 1$ we can plot the response of the system

```
[103]: s = ct.tf('s')
       G1 = 1/100/((tau1*s+1)*(tau2*s+1))
       G2 = 1/((tau1*s+1)*(tau2*s+1))
       Gc = 500
       T2 = ct.feedback(Gc*G1, 1)
       t1, y1 = ct.step_response(T2, np.r_[0:150:0.5])
       t2, y2 = ct.step_response(T2+T2.dcgain(), np.r_[150:300:0.5])
       plt.plot(t1, y1)
       plt.plot(t2, y2)
       plt.legend([r"$T_{ds}=\frac{A}{s}$", r"$T_{ds}=\frac{2A}{s}$"])
       plt.show()
```



5.2 c)

The steady state error for $T_{ds} = \frac{2}{s}$ is given by

[104]:
```
e_ss = 2-(T2+T2.dcgain()).dcgain()
eq_disp("e_{ss}", round(e_ss,3))
```

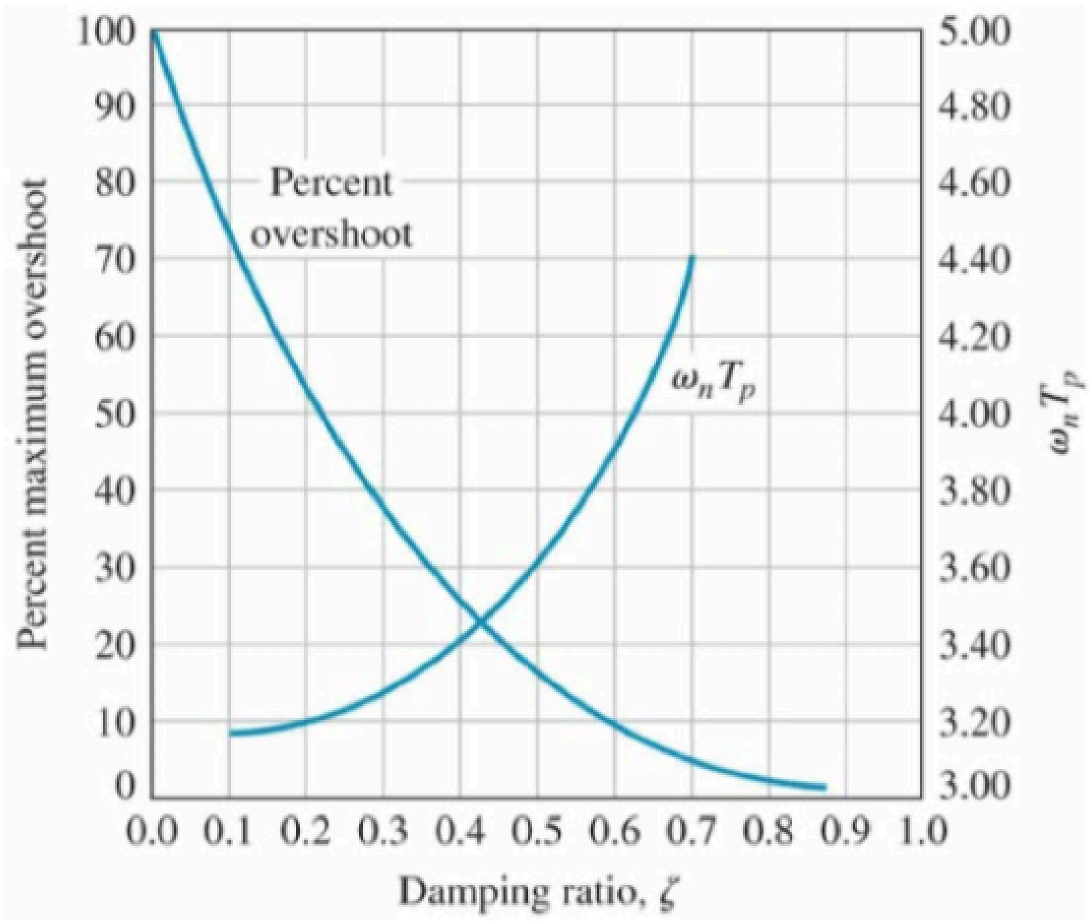
$e_{ss} = 0.333$

5.3 d)

From the figure the damping ratio must be $\zeta > 0.6$

[105]:
```
Image('PO.png')
```

[105]:



[106]:
```
Gc = 1/s
L = Gc*G1
L
```

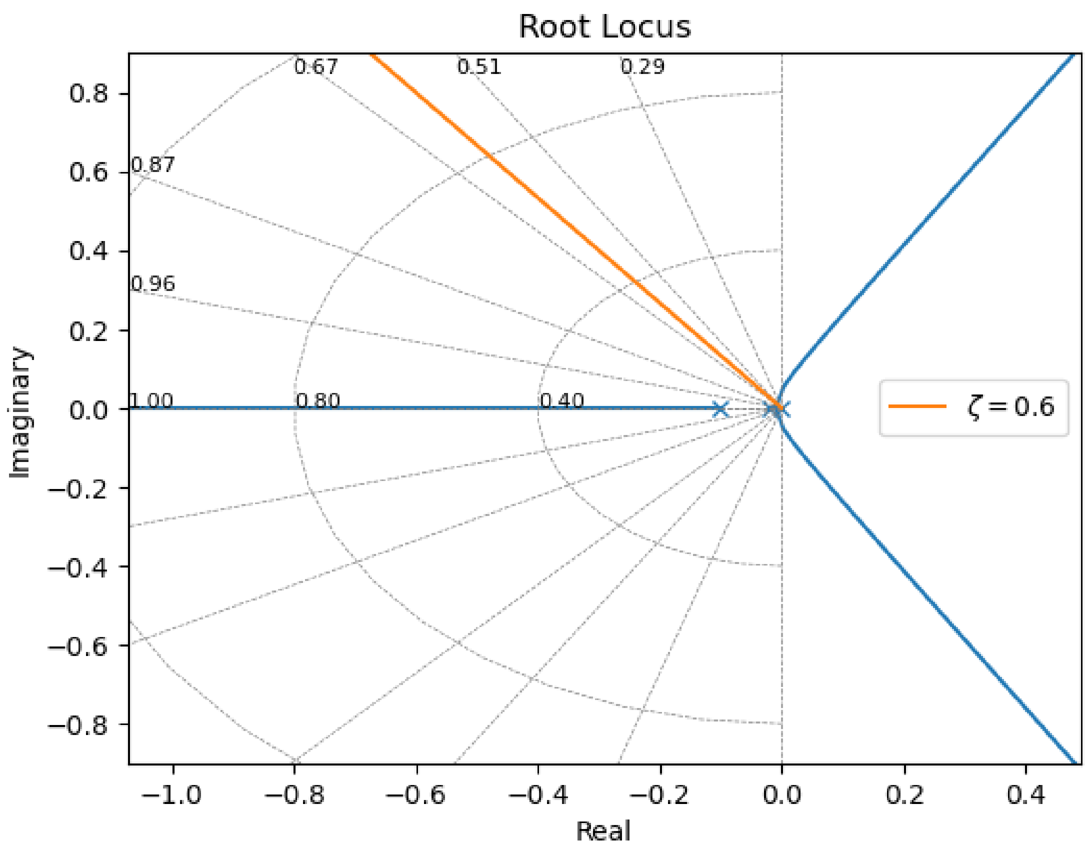
[106]:

$$\frac{0.01}{500s^3 + 60s^2 + s}$$

We plot the damping ratio requirement on the root locus and determine the gain

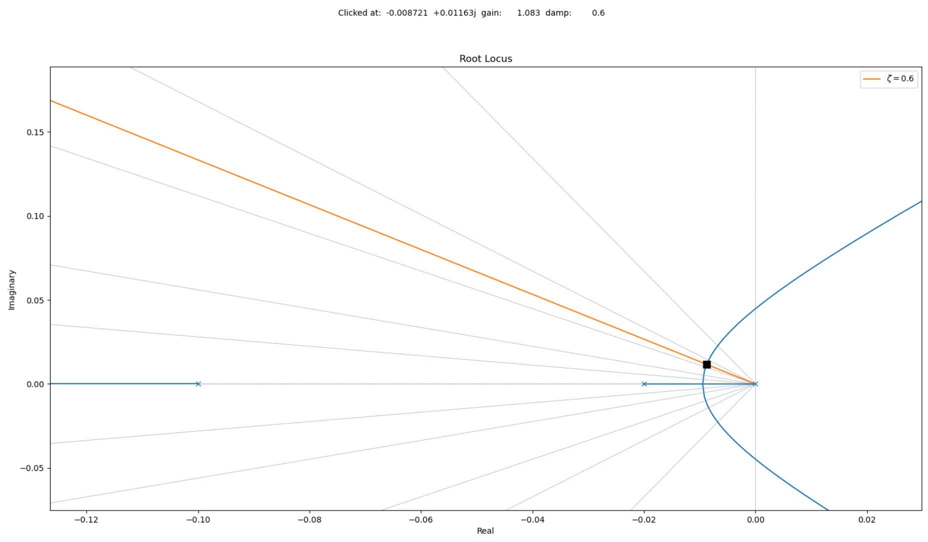
[107]:
```
zeta = 0.6
theta = np.arcsin(zeta)
a = -np.cos(theta)/np.sin(theta)
tspan = np.r_[0:-10:-1]
damp = a*tspan
a = ct.root_locus(L)
handle= plt.plot(tspan, damp, label=r"$\zeta=0.6$")
plt.legend(handles=handle)
```

```
[107]: <matplotlib.legend.Legend at 0x1c602d0ceb0>
```



```
[108]: Image("DP9_9.png")
```

```
[108]:
```



Clicked at: -0.008721 +0.01163j gain: 1.083 damp: 0.6

From the root locus we can see that the gain must be $K < 1.089$ to meet the calculated damping ratio criteria.
We chose $K = 1$

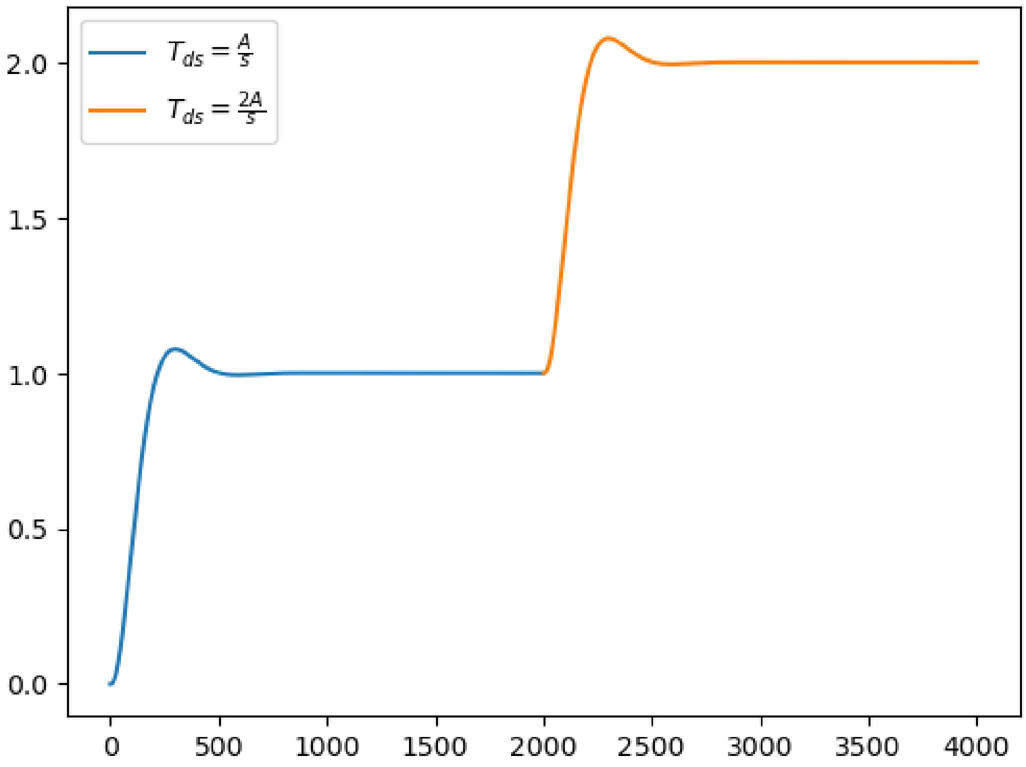
```
[109]: Gc=1/s
       T = ct.feedback(Gc*G1, 1)
       ct.step_info(T)
```

```
[109]: {'RiseTime': 140.28630808602514,
        'SettlingTime': 439.74361957734806,
        'SettlingMin': 0.9022355487060528,
        'SettlingMax': 1.0770685327703315,
        'Overshoot': 7.706853277033154,
        'Undershoot': 0,
        'Peak': 1.0770685327703315,
        'PeakTime': 300.8062182998424,
        'SteadyStateValue': 1.0}
```

We can see that the desired overshoot is now met

We can now plot the result as in b) for a unit step input

```
[110]: t1, y1 = ct.step_response(T, np.r_[0:2000:0.5])
       t2, y2 = ct.step_response(T+T.dcgain(), np.r_[2000:4000:0.5])
       plt.plot(t1, y1)
       plt.plot(t2, y2)
       plt.legend([r"$T_{ds}=\frac{A}{s}$", r"$T_{ds}=\frac{2A}{s}$"])
       plt.show()
```



The steady state error for $T_{ds} = \frac{2}{s}$ is given by

```
[111]: e_ss = 2-(T+T.dcgain()).dcgain()
       eq_disp("e_{ss}", round(e_ss,3))
```

$e_{ss} = 0.0$

5.4 e)

From the settling time and overshoot requirement, we can find the desired ζ and ω, then we can find the equivalent ideal pole position of the second order system

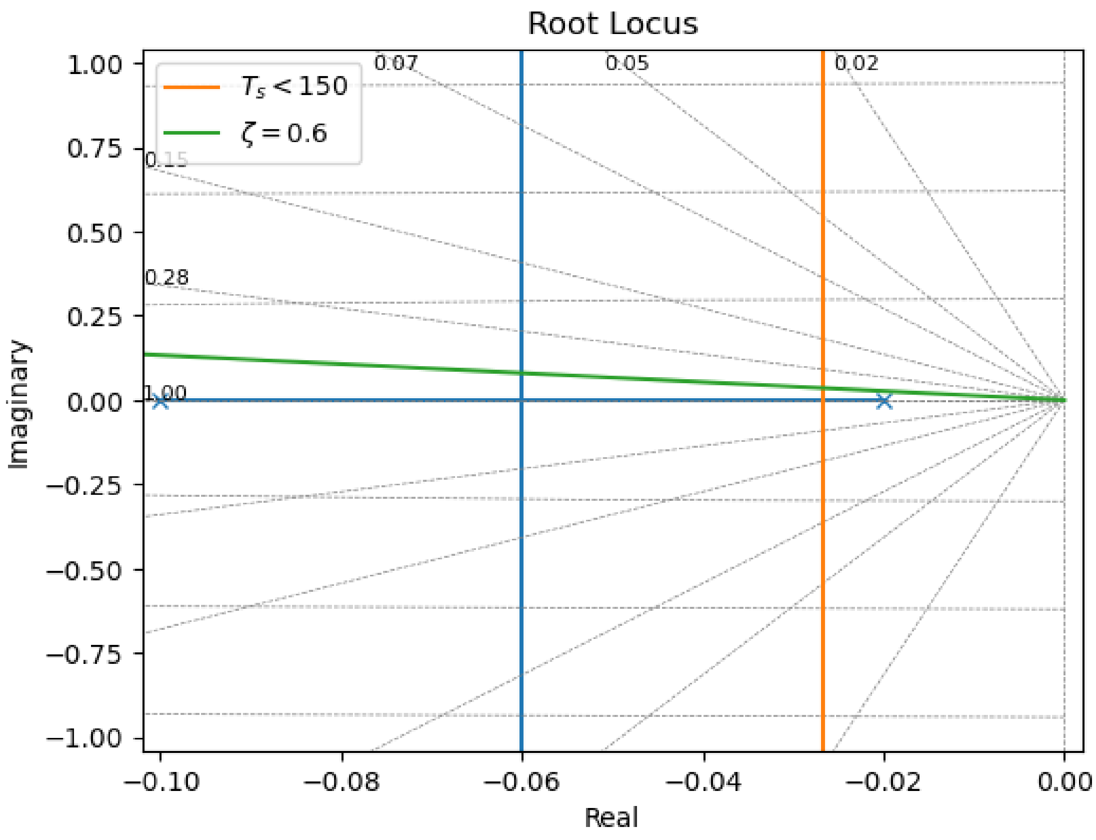
```
[112]: s = ct.tf('s')
       Ts=150
       zeta = 0.6
       omega = 4/(Ts*zeta)
```

We plot the root locus of the plant with the constraint lines that represents the system requirements

```
[114]: b = ct.root_locus(G1)
       theta = np.arcsin(zeta)
       a = -np.cos(theta)/np.sin(theta)
       tspan = np.r_[0:-2000:-1]
```

```
damp = a*tspan
handles= []
handles.append(*plt.plot(np.array([-omega*zeta, -omega*zeta]), np.array([-1500,␣
↪1500]), label=f"$T_s<{round(Ts,3)}$"))
handles.append(*plt.plot(tspan, damp, label=r"$\zeta=0.6$"))
plt.legend(handles=handles)
```

[114]:
```
<matplotlib.legend.Legend at 0x1c607bca670>
```



The ideal pole placement of second order system is given by:

[115]:
```
dp = complex(-zeta*omega, (omega)*np.sqrt(1-zeta**2))
dp
```

[115]:
```
(-0.02666666666666667+0.035555555555555556j)
```

From the magnitude and the angle criterion we can determine the controller pole placement and gain, that will result in the desired pole position

[116]:
```
plant_poles = (G1*1/s).poles()
plant_zeros = (G1*1/s).zeros()
```

[117]:
```
p_angles = []
p_mag = []
for p in plant_poles:
    p_angles.append(np.angle(dp - p, deg=True))
    p_mag.append(np.abs(dp - p))
```

Apply the angle criterion to determine the controler pole location

[121]:
```
z_angle = (180 + sum(p_angles))%360
z = sp.symbols('z')
z = float(sp.solve(np.tan(z_angle*np.pi/180) - dp.imag/(dp.real+z), z)[0])
eq_disp('z', z)
```

$z = 0.0372960372960373$

Apply magnitude criterion to determine the gain

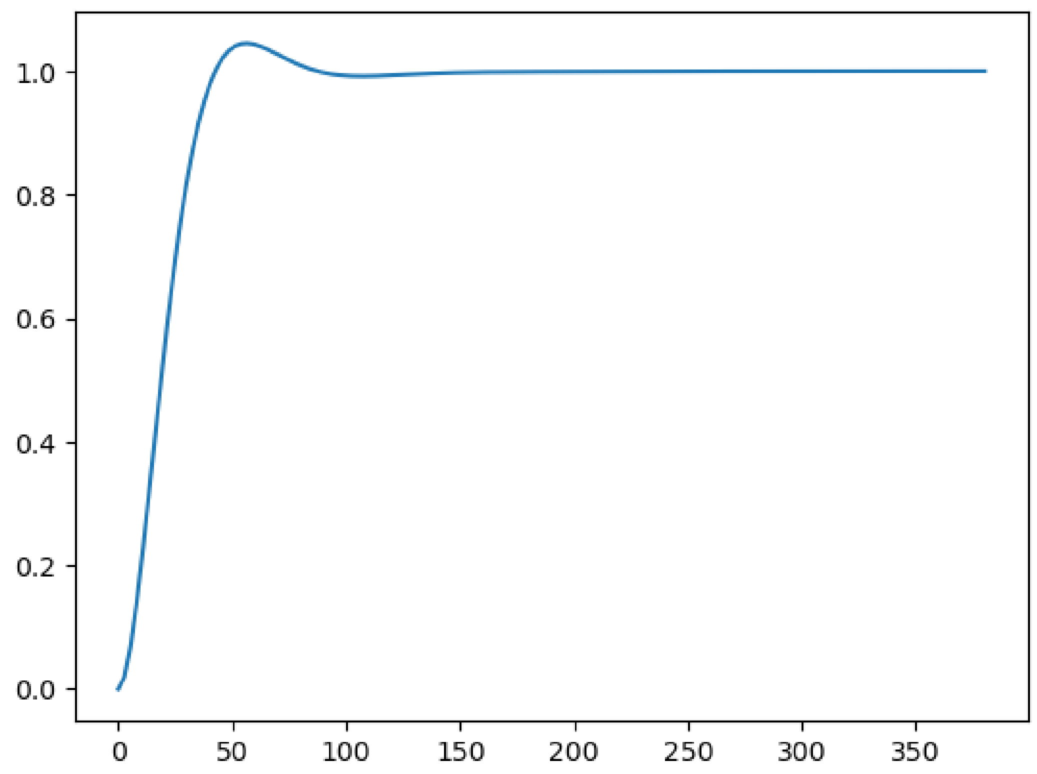
[122]:
```
K = abs(dp+z)/np.prod(p_mag)
eq_disp('K',K)
```

$K = 283.216783216783$

Lastly the step response is plotted. Through trial and error the pole position is decreased to half, to meet the requirements

[120]:
```
T = ct.feedback(G1*K*(s+z*0.5)/s)
plt.plot(*ct.step_response(T))
ct.step_info(T)
```

[120]:
```
{'RiseTime': 26.967260104194203,
 'SettlingTime': 75.50832829174377,
 'SettlingMin': 0.9130208737447244,
 'SettlingMax': 1.0449291014062252,
 'Overshoot': 4.492910140622519,
 'Undershoot': 0,
 'Peak': 1.0449291014062252,
 'PeakTime': 56.63124621880783,
 'SteadyStateValue': 1.0}
```


1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
350