Homework4

September 23, 2022

[1]:
```
import numpy as np
import sympy as sp
from sympy.plotting import plot
import matplotlib.pyplot as plt
from IPython.display import display, Latex, Math
def eq_disp(varstring, expr):
    display(Latex(f"${varstring}={sp.latex(expr)}$"))

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))

s = sp.symbols('s')
t, zeta, omega = sp.symbols('t, zeta, omega', positive=True, real=True)
```

1 E5.19

From the Transfer function we extract ω_n and ζ

[2]:
```
omega = sp.sqrt(7)
zeta = sp.Rational(3175, 1000)/2/omega
R = 1/s
T = omega**2/(s**2 + 2*zeta*omega*s + omega**2)
Y = R*T
eq_disp('Y(s)', Y)
```

$Y(s) = \frac{7}{s\left(s^2+\frac{127s}{40}+7\right)}$

The formula for percent overshoot is:

$$\sigma_p\% = \frac{y\left(T_p\right) - y(\infty)}{y(\infty)} \times 100\%$$

where

$$\left.\frac{dy(t)}{dt}\right|_{t=T_p} = 0|_{(\text{first })}$$

First we must find $y(t)$

Transform to the time domain

[3]:
```
y = sp.inverse_laplace_transform(Y.apart(), s, t)
eq_disp('y', y)
```

$y = \left(e^{\frac{127t}{80}} - \frac{127\sqrt{28671}\sin\left(\frac{\sqrt{28671}t}{80}\right)}{28671} - \cos\left(\frac{\sqrt{28671}t}{80}\right)\right)e^{-\frac{127t}{80}}$

The derivative in the laplace domain is obtained simply by multiplying by s (in case initial conditions are zero)

[4]:
```
dy = sp.inverse_laplace_transform((s*Y).apart(), s, t)
eq_disp('\\frac{dy}{dt}', sp.N(dy,3))
```

$\frac{dy}{dt} = 3.31e^{-\frac{127t}{80}}\sin\left(\frac{\sqrt{28671}t}{80}\right)$

Peak time is then calculated

[5]:
```
sol = sp.solve(dy.evalf(5), t)
Tp = min([x for x in sol if x != 0])
eq_disp('T_p', sp.N(Tp,3))
```

$T_p = 1.48$

Now the steady state value is calculated by

$$\lim_{t\to\infty} y(t)$$

[6]:
```
y_ss = sp.limit(y, t, sp.oo)
eq_disp('y(\infty)', y_ss)
```

$y(\infty) = 1$

Finally %overshoot can be calculated

[7]:
```
sigma_p = ((y.subs(t, Tp) - y_ss)/y_ss*100).evalf()
eq_disp('\\sigma_p\%', sp.N(sigma_p,3))
```

$\sigma_p\% = 9.48$

The setling time at 2% threshold is calculated by:

$$T_s = 4\tau = \frac{4}{\xi\omega_n}$$

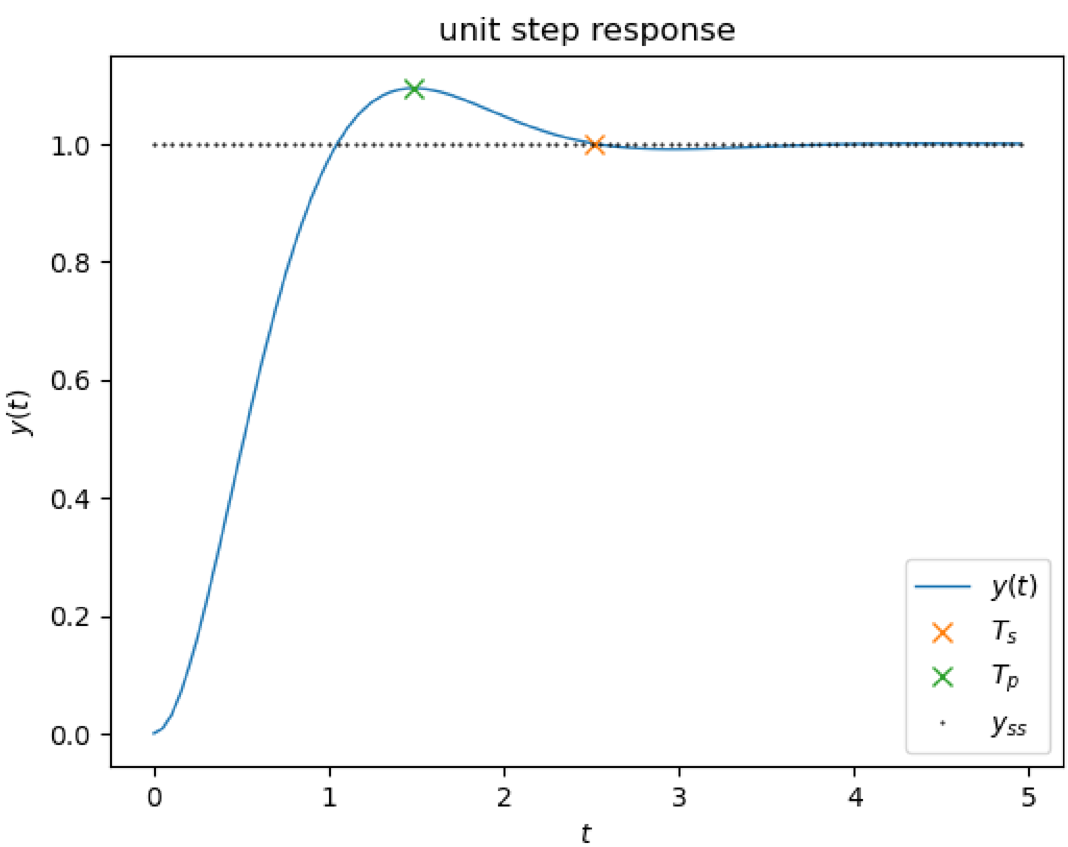
[8]:
```
Ts = 4/(zeta*omega)
eq_disp('T_s', sp.N(Ts,3))
```

$T_s = 2.52$

1.1 b)

To verify our results we plot $y(t)$ and anotate the plot with the peak time T_p and the settling time T_s

[9]:
```
y_f = sp.lambdify(t, y)
tspan = np.r_[0:5:0.05]
plt.plot(tspan, y_f(tspan), linewidth=1)
plt.plot(float(Ts), y_f(float(Ts)),linestyle='None', markersize=7, marker='x')
plt.plot(float(Tp), y_f(float(Tp)),linestyle='None', markersize=7, marker='x')
plt.plot(tspan, float(y_ss)*np.ones(len(tspan)), 'k.', markersize=1)
plt.legend(['$y(t)$', '$T_s$', '$T_p$', '$y_{ss}$'])
plt.xlabel('$t$')
plt.ylabel('$y(t)$')
plt.title('unit step response')
plt.show()
```



[10]:
```
import numpy as np
import sympy as sp
from sympy.plotting import plot
import matplotlib.pyplot as plt
```

```python
from IPython.display import display, Latex, Math, Image
def eq_disp(varstring, expr):
    display(Latex(f"${varstring}={sp.latex(expr)}$"))

# Function for displaying expressions
def eq_disp_unit(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))
```

2 P5.7

[11]:
```python
# Moment of inertia of the equipment and man [kg*m^2]
I = 25

s, t, K_1, K_2, K_3sym = sp.symbols('s, t, K_1, K_2, K_3')
```

2.1 a)

Input

$$R(t) = t \Leftrightarrow R(s) = \frac{1}{s^2}$$

The transfer function is found

[12]:
```python
G1 = (K_1*K_2/(I*s))/(1 + K_1*K_2*K_3sym/(I*s))

G2 = 1/s

T = sp.simplify(G1*G2/(1 + G1*G2))
eq_disp('T(s)', T)
```

$T(s) = \frac{K_1 K_2}{K_1 K_2 + s(K_1 K_2 K_3 + 25s)}$

The tracking error

$$E(s) = R(s) - Y(s) = R(s) - T(s)R(s) = R(s)(1 - T(s))$$

So we get

$$E(s) = \frac{1}{s^2}(1 - T(s))$$

[13]:
```python
# Input
R = 1/(s**2)
```

```python
E = sp.simplify(1/s**2*(1-T))

eq_disp('E(s)', E)
```

$E(s) = \frac{K_1 K_2 K_3 + 25s}{s(K_1 K_2 + s(K_1 K_2 K_3 + 25s))}$

The steady state error e_{ss} is found by

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

[14]:
```python
e_ss = sp.limit(s*E, s, 0)
eq_disp('e_{ss}', e_ss)
```

$e_{ss} = K_3$

The maximum steady state error is

$$e_{ss,max} = 0.01m$$

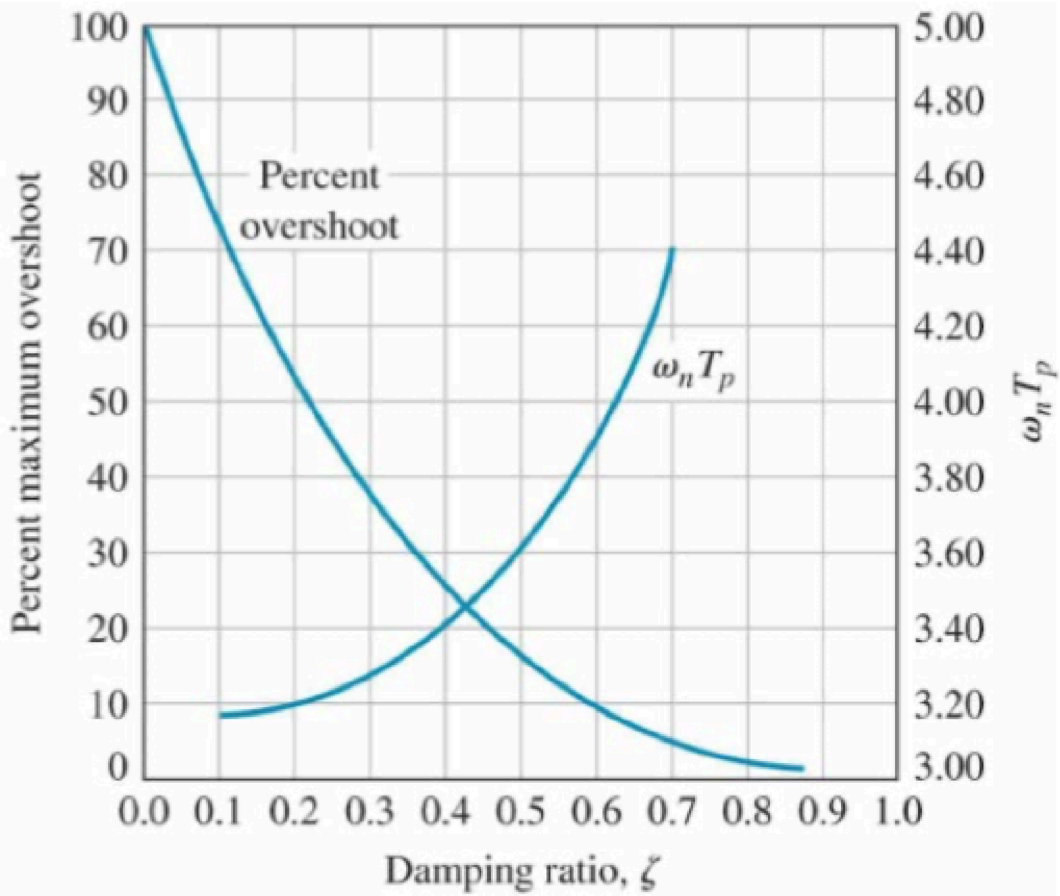
[15]:
```python
e_ss_max = 0.01
K_3 = sp.solve(e_ss - e_ss_max, K_3sym)[0]
eq_disp_unit('K_3', K_3, 'm')
```

$K_3 = 0.01\, m$

2.2 b)

[17]:
```python
Image('P.O..png')
```

[17]:



From the diagram it is seen that for a percent overshoot (P.O.) of 10% the corresponding damping ratio is $\zeta = 0.6$

[18]:
```python
zeta = 0.6
T_a = T.subs(K_3sym, K_3)
eq_disp('T',T_a)
```

$T = \frac{K_1 K_2}{K_1 K_2 + s(0.01 K_1 K_2 + 25s)}$

Solve for the n poles

[19]:
```python
p, q = (T).as_numer_denom()
poles = sp.solve(q,s)

for i, pole in enumerate(poles):
    eq_disp(f'-s_{i}', pole)
```

$-s_0 = -\frac{K_1 K_2 K_3}{50} - \frac{\sqrt{K_1 K_2 (K_1 K_2 K_3^2 - 100)}}{50}$

$-s_1 = -\frac{K_1K_2K_3}{50} + \frac{\sqrt{K_1K_2(K_1K_2K_3^2-100)}}{50}$

The poles 1 and 2 are given by

$$s_{1,2} = -\zeta\omega_n \mp j\omega_n\sqrt{1-\zeta^2}$$

This means that

$$\zeta\omega_n = \frac{K_1K_2 0.01}{50} \tag{1}$$

The response of a second order system is given by

$$T(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

$$Y(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}R(s)$$

In this case:

[20]:
```
eq_disp('T', sp.expand(T_a))
```

$T = \frac{K_1K_2}{0.01K_1K_2s + K_1K_2 + 25s^2}$

Meaning that

$$\omega_n^2 = \frac{K_1K_2}{25} \tag{2}$$

Using equations 1 and 2 the constant K_1K_2 is found

[21]:
```
K1K2sym,omegasym = sp.symbols('K1K2,omega_n')
eq1 = sp.Eq(zeta*omegasym,K1K2sym*K_3/50)
eq2 = sp.Eq(omegasym**2,K1K2sym/25)
K_1K_2_b,omega_n_b = sp.solve([eq1,eq2],(K1K2sym,omegasym))[1]

eq_disp('\\omega_n',omega_n_b)
eq_disp('K_1K_2',K_1K_2_b)
```

$\omega_n = 120.0$

$K_1K_2 = 360000.0$

2.3 c)

[22]:
```
# Input
R = 1/s

Y = sp.expand(T*R)

eq_disp('Y(s)', Y)
```

$Y(s) = \frac{K_1K_2}{K_1K_2K_3s^2 + K_1K_2s + 25s^3}$

Where the $Y(s)$ can be written on the form

$$\begin{aligned} Y(s) &= \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} R(s) \\ &= \frac{\omega_n^2}{s(s^2 + 2\zeta\omega_n s + \omega_n^2)} \end{aligned}$$

So

$$\omega_n^2 = \frac{K_1 K_2}{25}$$

and

$$2\zeta\omega_n = \frac{K_1 K_2 K_3}{25}$$

This gives the $y(t)$ in the time domain as

$$y(t) = 1 - \frac{1}{\beta} e^{-\zeta\omega_n t} sin(\omega_n \beta t + \theta)$$

Where

$$\begin{aligned} \beta &= \sqrt{1 - \zeta^2} \\ \theta &= cos^{-1}\zeta \end{aligned}$$

[23]:
```
beta, zeta, omega_n, theta = sp.symbols('beta,zeta,omega_n,theta')
y = sp.simplify(1 - 1/beta*sp.exp(-zeta*omega_n*t)*sp.sin(omega_n*beta*t+theta))
eq_disp('y(t)',y)
```

$y(t) = 1 - \frac{e^{-\omega_n t \zeta} \sin(\beta\omega_n t + \theta)}{\beta}$

Integrating this w.r.t to time from 0 to ∞

Squaring $y(t)$ and integrating the result gives the ISE (see example 5.7 in the book)

$$ISE = \int_0^\infty \left(1 - \frac{1}{\beta} e^{-\zeta\omega_n t} sin(\omega_n \beta t + \theta)\right)^2 dt$$

[24]:
```
from sympy import *
ISE = sp.integrate(y**2, (t, 0, oo))
eq_disp('ISE',ISE)
```

$ISE = \frac{\int\limits_0^\infty \left(\beta e^{\omega_n t \zeta} - \sin(\beta\omega_n t + \theta)\right)^2 e^{-2\omega_n t \zeta}\, dt}{\beta^2}$

[25]:
```
omega_n = sp.sqrt(K1K2sym/25)
eq_disp('\\omega_n',omega_n)

zeta = K1K2sym*K_3sym/(2*25*omega_n)
```

```
eq_disp('\\zeta',zeta)
```

$\omega_n = \frac{\sqrt{K1K2}}{5}$

$\zeta = \frac{\sqrt{K1K2}K_3}{10}$

[26]:
```
beta = sp.sqrt(1-zeta**2)
eq_disp('\\beta',beta)

theta = sp.acos(zeta)
eq_disp('\\theta',theta)
```

$\beta = \sqrt{-\frac{K1K2K_3^2}{100} + 1}$

$\theta = \operatorname{acos}\left(\frac{\sqrt{K1K2}K_3}{10}\right)$

[27]:
```
y = sp.simplify(1 - 1/beta*sp.exp(-zeta*omega_n*t)*sp.sin(omega_n*beta*t+theta))
eq_disp('y(t)',y)
```

$$y(t) = 1 - \frac{10e^{-\frac{K1K2K_3t}{50}}\sin\left(\frac{\sqrt{K1K2}t\sqrt{-K1K2K_3^2+100}}{50} + \operatorname{acos}\left(\frac{\sqrt{K1K2}K_3}{10}\right)\right)}{\sqrt{-K1K2K_3^2+100}}$$

Squaring $y(t)$ and integrating the result gives the ISE as in example 5.7 in the book (We consulted xuping about this last part of the problem and was told it was okay to skip since we have not yet learned the theory)
The following approach doesnt work

[ ]:
```
ISE = sp.integrate(y**2, (t, 0, oo))
eq_disp('ISE',ISE)
```

Differentiating the ISE w.r.t. K_1K_2

[ ]:
```
ISE_diff = sp.diff(ISE,K1K2sym)
ISE_diff
```

Setting the differentiated ISE equal to 0 to find the minimum

[ ]:
```
K1K2 = sp.solve(ISE_diff,K1K2sym)
eq_disp('K_1K_2', K1K2)
```