In [ ]:
```
import numpy as np # numerical library
import matplotlib.pyplot as plt # plotting library
%config InlineBackend.figure_format='retina' # high-res plots
import control.matlab as ctm # matlab layer for control systems library
import control as ct # use regular control library for a few things
ct.set_defaults('statesp', latex_repr_type='separate') # ABCD matrices
```

1.

a.

In [ ]:
```
T = 0.1
damping = 1/5
tau = 1
s_p_m = tau/damping
s_p_a = np.pi-np.arccos(damping)
s_p = s_p_m*np.exp(s_p_a*1j)
z_p = np.exp(s_p*T)
M = 0.01
K = 44.567
plant = ctm.tf(1, [M, 0, K], inputs='u', outputs='y')
plantd = ct.c2d(plant, T, 'zoh')
leadangle = -(np.angle(ctm.evalfr(plantd, z_p))-np.pi)

zp = 0
zeroangle = leadangle + np.angle(z_p - zp)
zz_pre = np.imag(z_p)/np.tan(zeroangle)
zz = np.real(z_p) - zz_pre
```

In [ ]:
```
controller = ctm.tf([1, -zz], [1, -zp], T)
loop = controller*plantd
k = 1/np.abs(ctm.evalfr(loop, z_p))
lead_compensator = ctm.tf(k*controller)
display(lead_compensator)
```

$$\frac{72.85z - 60.67}{z} \quad dt = 0.1$$

b.

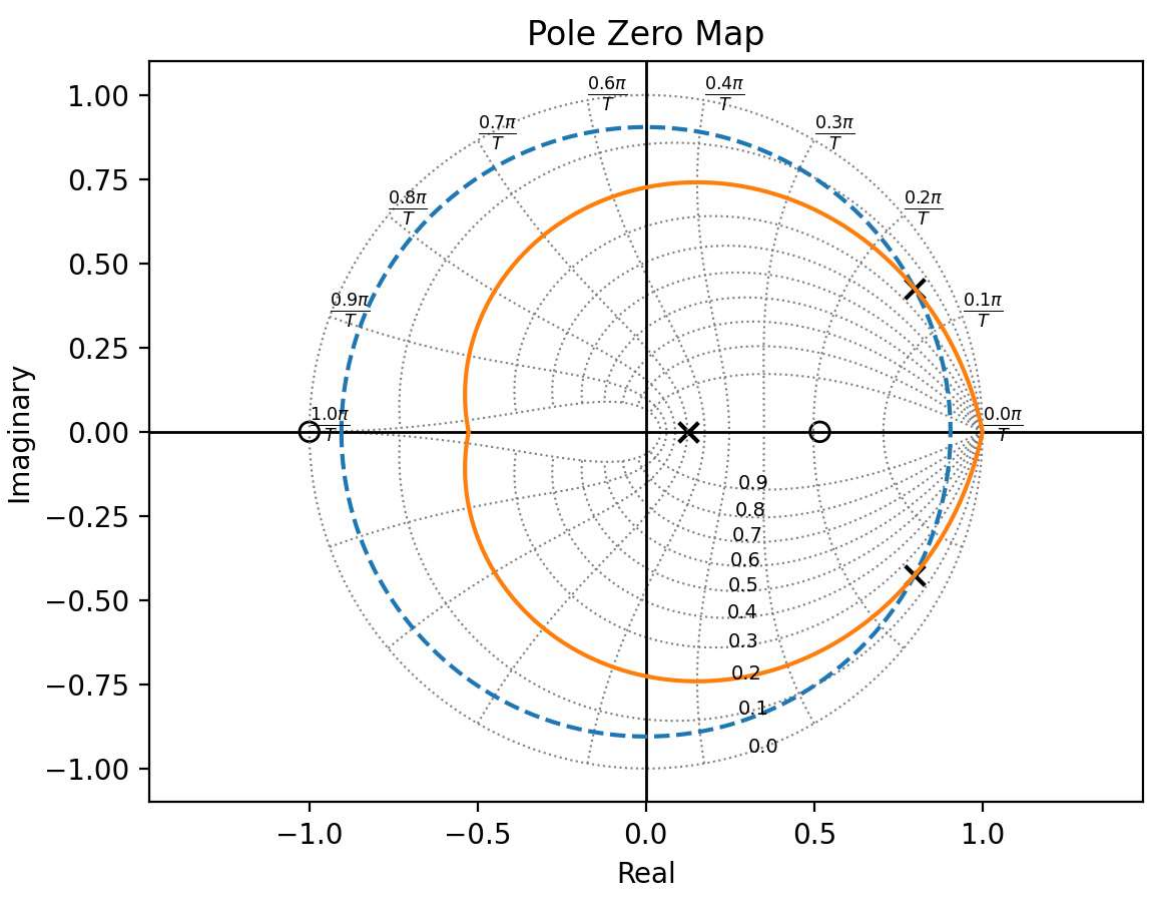
In [ ]:
```
def plot_constant_lines_z(damping, omegan, T=1, c=None, plot_omegan=False):
    """plot lines of constant damping ratio (damping), time constant
    (1/damping*omegan) and omegan on the z-plane"""
    nyquist_freq = np.pi/T
    freqs = np.linspace(-nyquist_freq, nyquist_freq, 201, endpoint=True)
    # constant time constant is a vertical line in s-plane at -damping*omegan
    timeconstant_line_s = -damping*omegan*np.ones_like(freqs) + 1j*freqs
    timeconstant_line_z = np.exp(timeconstant_line_s*T)
    plt.plot(timeconstant_line_z.real, timeconstant_line_z.imag,
             '--', c=c, label=f'$\damping\omega_n$={damping*omegan:.3f}')
    # constant damping ratio damping is a radial line from origin in splane
    damping_line_splane = -abs(freqs)/np.tan(np.arccos(damping)) + 1j*freqs
    damping_line_zplane = np.exp(damping_line_splane*T)
    plt.plot(damping_line_zplane.real, damping_line_zplane.imag,'-', c=c,
            label=f'$\damping$={damping:.3f}')
    if plot_omegan: # circle centered at origin in s-plane
        angles = np.linspace(np.pi/2, 3./2*np.pi, 31, endpoint=True)
        omegan_line_s = omegan*np.exp(1j*angles)
        omegan_line_z = np.exp(omegan_line_s*T)
        plt.plot(omegan_line_z.real, omegan_line_z.imag,
                 c=c, ls='-.', label=f'$\omega_n$={omegan:.3f}')
```

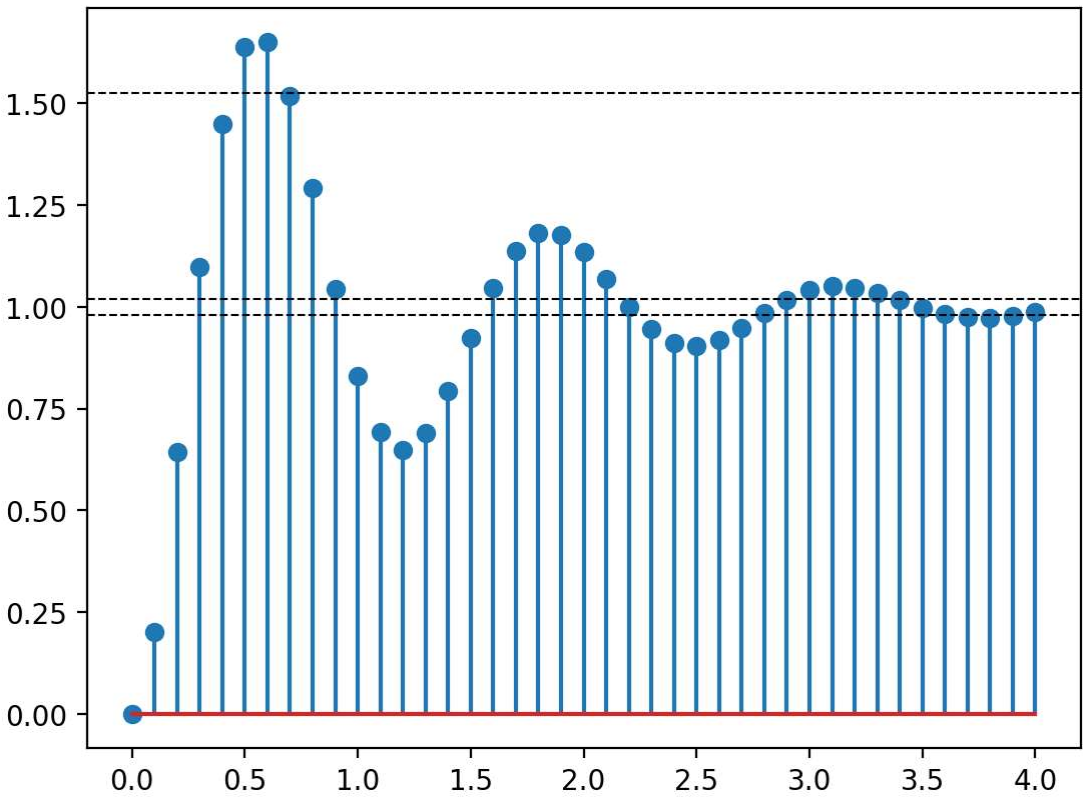
In [ ]:
```
PO = 1 + np.exp(-np.pi*damping/np.sqrt(1-damping**2))

kff = ctm.tf2ss(K, 1, inputs='yref', outputs='u')
C = ctm.ss(k*controller, inputs='e', outputs='u')
sum = ct.summing_junction(inputs=['yref', '-y'], outputs='e')
sys = ct.interconnect((kff, C, sum, plantd), inputs='yref', outputs='y')
ctm.pzmap(sys, grid=True)
plot_constant_lines_z(damping, tau/damping, T)

plt.figure()
y, t = ctm.step(sys, 4)
plt.stem(t,y)
plt.plot()
plt.axhline(PO, ls='--', c='k', lw=0.75)
plt.axhline(0.98, ls='--', c='k', lw=0.75)
plt.axhline(1.02, ls='--', c='k', lw=0.75)
```

Out[ ]:
```
<matplotlib.lines.Line2D at 0x1d9b1d2d290>
```





c.

The maximum percent overshoot of the system with lead compensator in the plot is higher than the one from the Simple Oscillator Model due to the extra zero designed in the lead compensator.

d.

The settling time is predicted as the formula does.

e.

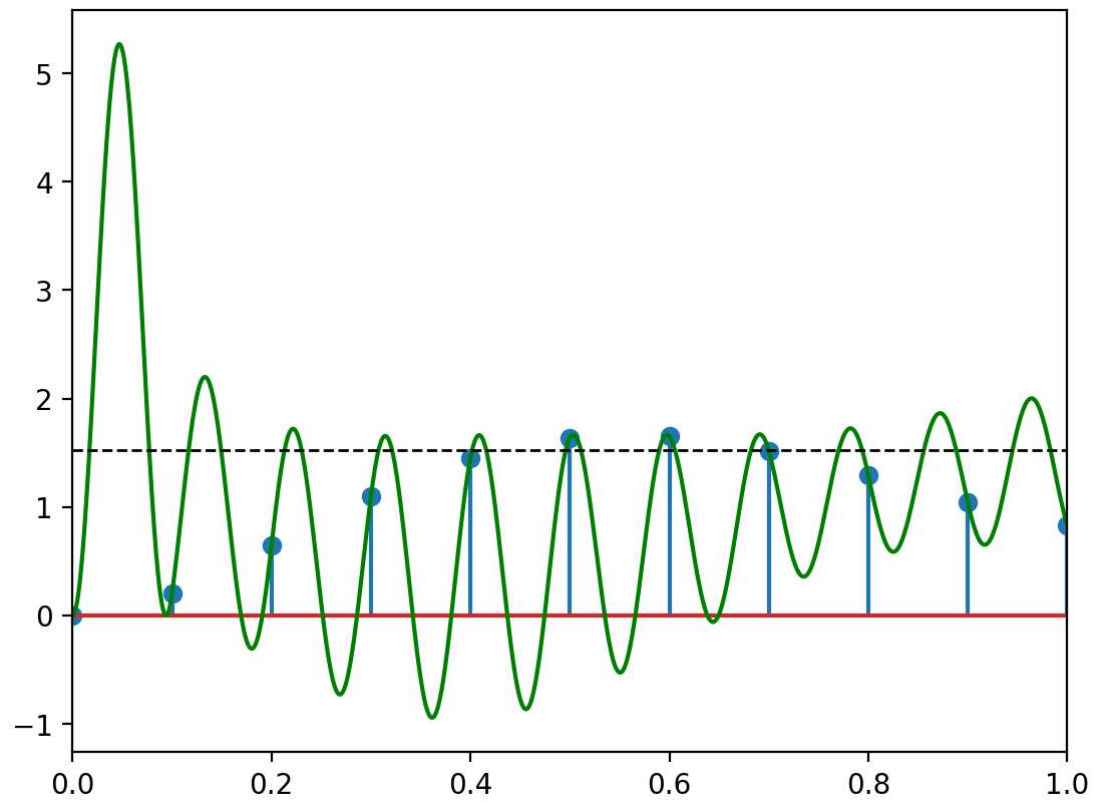
In [ ]:
```python
def sampled_data_system(sysd, simulation_dt):

    assert ct.isdtime(sysd, True), "sysd must be discrete-time"
    sysd = ct.ss(sysd) # convert to state-space if not already
    nsteps = int(round(sysd.dt / simulation_dt))
    assert np.isclose(nsteps, sysd.dt/simulation_dt), \
        "simulation_dt must be an integral multiple of sysd.dt"
    st = 0
    y = np.zeros((sysd.noutputs, 1))
    def updatefunction(t, x, u, params):
        nonlocal st
        if st == 0: # is it time to sample?
            x = sysd._rhs(t, x, u)
        st += 1
        if st == nsteps:
            st = 0
        return x
    def outputfunction(t, x, u, params):
        nonlocal y
        if st == 0: # is it time to sample?
            y = sysd._out(t, x, u)
        return y
    return ct.ss(updatefunction, outputfunction, dt=simulation_dt,
                 name=sysd.name, inputs=sysd.input_labels,
                 outputs=sysd.output_labels, states=sysd.state_labels)
```

In [ ]:
```
dt = 0.001
plant_sim = ct.c2d(plant, dt, 'zoh')
controller_sim = sampled_data_system(C, dt)
sys_sim = ct.interconnect((kff, controller_sim, sum, plant_sim), inputs='yref', outputs='y')
time = np.arange(0, 1, dt)
inp = np.ones_like(time)
t_sim, y_sim  = ct.input_output_response(sys_sim, time, inp)

plt.stem(t, y)
plt.plot(t_sim, y_sim, 'g')
plt.xlim(0, 1)
plt.axhline(PO, c='k', linestyle='--', lw=1)
```

Out[ ]:
```
<matplotlib.lines.Line2D at 0x1d9b73fbc90>
```



f.

The maximum of percent overshoots in discrete time and continous time are higher than the PO from Simple Oscillator Model.

2.

a.

In [ ]:
```
T = 0.1
damping = 1/5
tau = 1/(2/3)
s_p_m = tau/damping
s_p_a = np.pi-np.arccos(damping)
s_p = s_p_m*np.exp(s_p_a*1j)
z_p = np.exp(s_p*T)
M = 0.01
B = 0.0628
plant = ctm.tf(1, [M, B, 0], inputs='u', outputs='y')
plantd = ct.c2d(plant, T, 'zoh')
K = 1/np.abs(ctm.evalfr(plantd, z_p))
K
```

Out[ ]:
```
0.6242584844354816
```

b.

The transfer function op is a Type 1 system.

In [ ]:
```
Kv = K/0.06283
Kv
```

Out[ ]:
```
9.935675384935248
```

c.

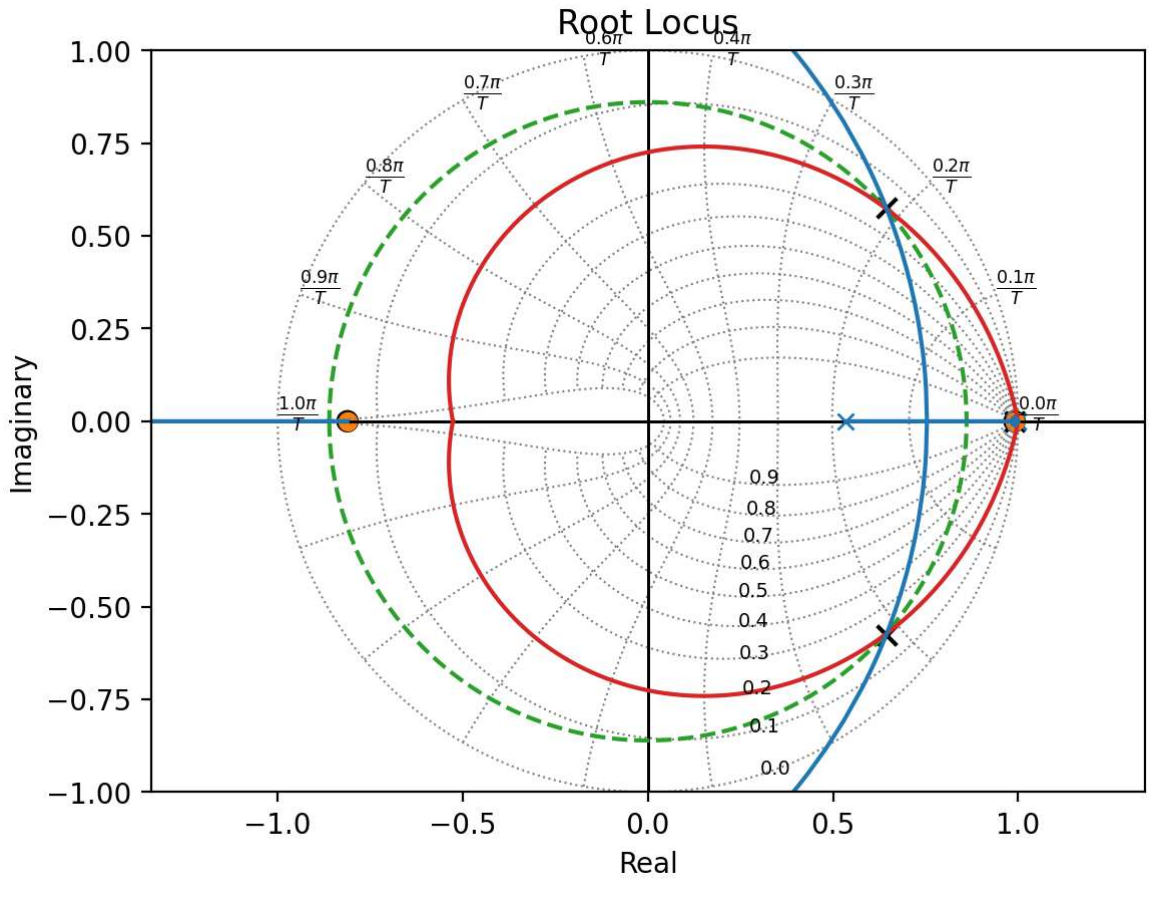
In [ ]:
```
alpha = 10
zz = 0.99
zp = 1 - (1 - zz)/alpha
controller = ct.tf([1, -zz], [1, -zp], T)
display(controller)
```

$$\frac{z - 0.99}{z - 0.999} \quad dt = 0.1$$

In [ ]:
```
cl_lag = ctm.feedback(K*controller*plantd, 1)
ctm.pzmap(cl_lag)
ctm.rlocus(plantd*controller)
```

```
plot_constant_lines_z(damping, tau/damping, T)
plt.axis((-1,1,-1,1))
```

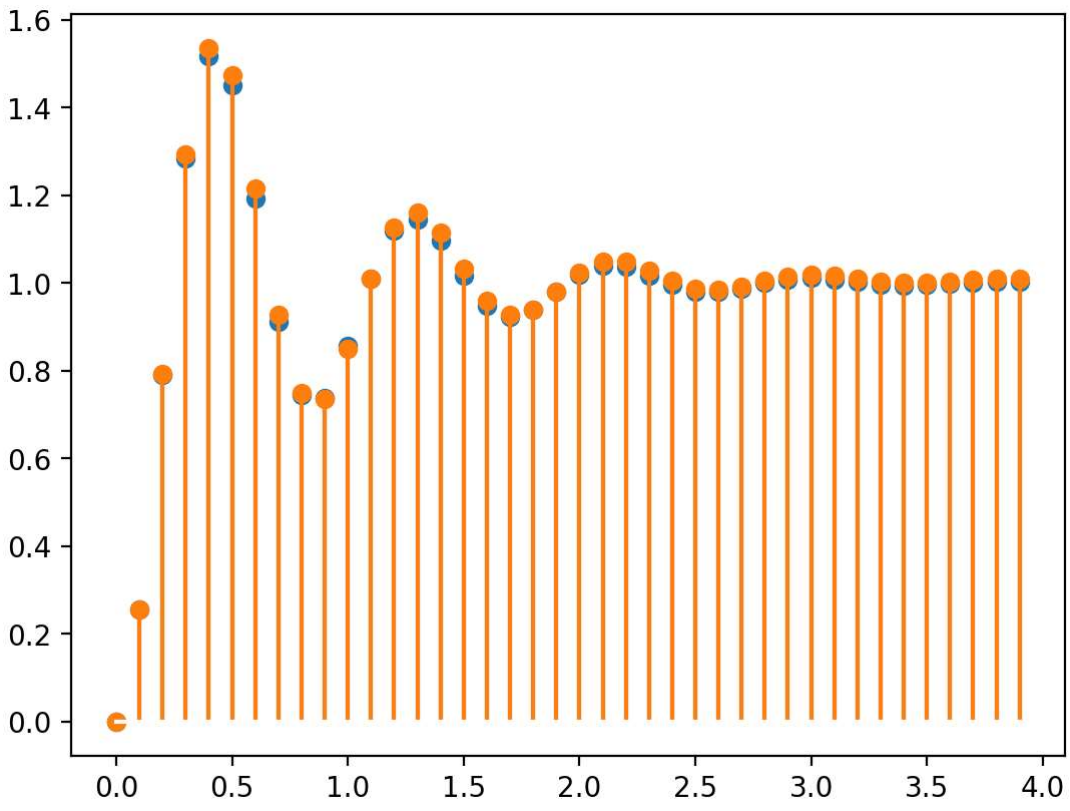
Out[ ]: (-1.0, 1.0, -1.0, 1.0)



d.

```
time = np.arange(0, 4, T)
cl = ctm.feedback(K*plantd, 1)
y, _ = ctm.step(cl, time)
ylag, _ = ctm.step(cl_lag, time)
plt.stem(time, y, linefmt='C0', basefmt='w', markerfmt='C0o')
plt.stem(time, ylag, linefmt='C1', basefmt='w', markerfmt='C1o')
```

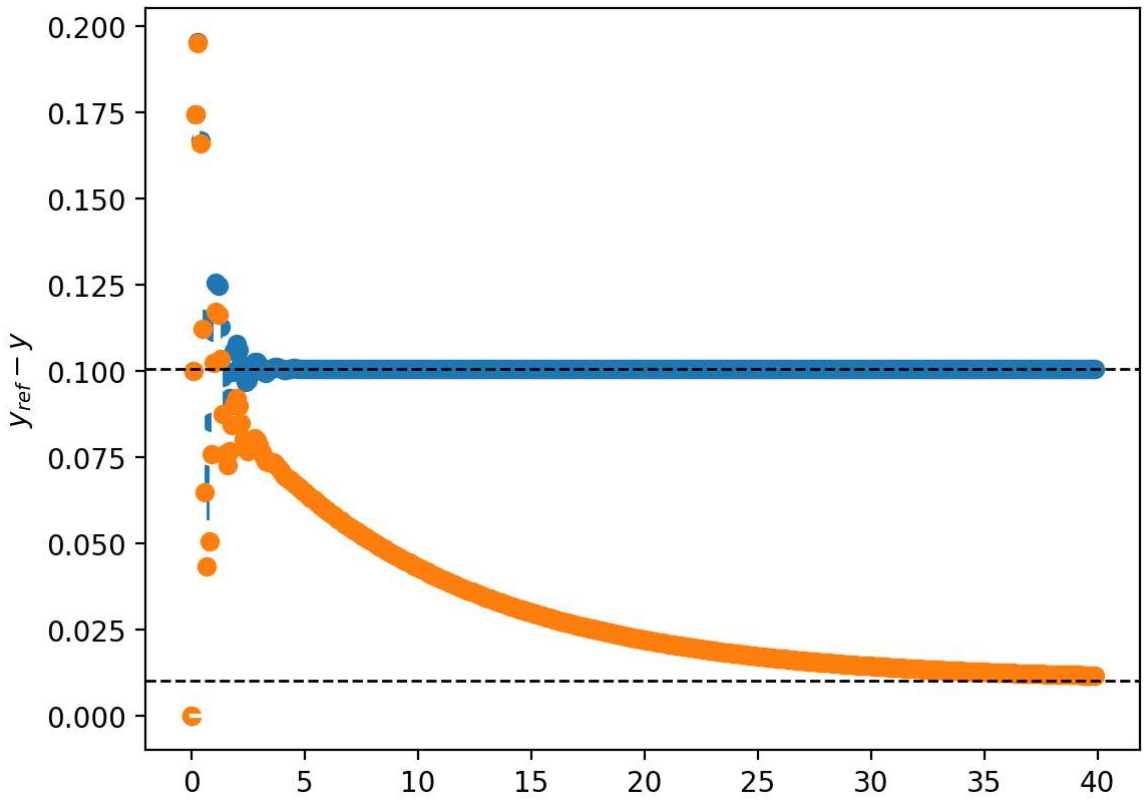
Out[ ]: <StemContainer object of 3 artists>



e.

```
time = np.arange(0, 40, T)
ramp = ctm.tf(T, [1, -1], T)
yref,_ = ctm.step(ramp, time)
y, _ = ctm.step(ramp*cl, time)
ylag, _ = ctm.step(ramp*cl_lag, time)
plt.stem(time, yref-y, linefmt='w', basefmt='w', markerfmt='C0o')
plt.stem(time, yref-ylag, linefmt='w', basefmt='w', markerfmt='C1o')
plt.ylabel(r'$y_{ref} - y$')
plt.axhline(1/Kv, ls='--', c='k', lw=1)
plt.axhline(1/(10*Kv), ls='--', c='k', lw=1)
```

Out[ ]: <matplotlib.lines.Line2D at 0x1d9b2e1cc10>



f.

The length of the time constant of red exponential decay is determined by the third pole of the close loop system.