In [1]:

```python
import pandas as pd
import numpy as np
from numpy.linalg import inv
import math as m
from math import sqrt
import sympy as sp
from sympy import collect, simplify, expand, fraction, latex
from IPython.display import display, Markdown, Math
import matplotlib.pyplot as plt
sp.init_printing(use_latex='mathjax')
```

In [2]:

```python
class numden_coeff:
    def __init__(self, expr, order_num, order_denum, symb):
        self.num, self.denum = fraction(expr)
        self.symb = symb
        self.common_factor = None
        self.lst_denum_coeff = self.build_lst(self.denum, order_denum)
        self.lst_num_coeff = self.build_lst(self.num, order_num)
#         display(Markdown(r"Numerator coefficients (\beta)"), self.lst_num_coeff[::-1])
#         display(Markdown(r"Denominator coefficients (alpha)"), self.lst_denum_coeff[::-1])


    def build_lst(self, poly, order):
        lst = [expand(poly).coeff(self.symb**i) for i in range((order), 0, -1)]
        lst.append(poly.subs(self.symb,0))
        if (self.common_factor == None):
            self.common_factor = lst[0]

        lst = [simplify(lst[i]/self.common_factor) for i in range(order + 1)]
        return lst
```

Problem 1

Part 1

In [3]:

```python
c, d, s, zeta, omega, r_1, s_0, s_1, a_0 = sp.symbols('c d s zeta omega r_1 s_0 s_1 a_0')
a = 1
b = 1
G1 = b/(s + a)
G2 = c/(s+d)
G = collect(expand(G1*G2), s)
B, A = fraction(G)
B_minus = B
```

$G_1(s)G_2(s) = G(s) = \frac{c}{d + s^2 + s(d+1)}$

Therefore

$B = c$

$A = d + s^2 + s(d+1)$

Since $Deg(B)$ is clearly 0, $B^+ = 1$ and $B^- = c$

In [4]:

```python
A_m = s**2 + 2*s + 1
B_m = 1
G_m = B_m/A_m
B_m_prime = B_m/B_minus
```

A_m is given to be $s^2 + 2s + 1$. Letting the desired model take the form of

$G_m = \frac{\omega^2}{s^2+2\zeta\omega+\omega^2}$

ω and ζ are equivalent to 1. Since $\omega = 1$, B_m must be equal to 1 which yeilds

$G_m = \frac{1}{s^2 + 2s + 1}$

$$Deg(A_0) = Deg(A) - Deg(B^+) - 1 = 2 - 0 - 1 = 1$$

$$Deg(A_c) = 2(Deg(A)) - 1 = 2 * 2 - 1 = 3$$

$$Deg(R) = Deg(S) = Deg(A_c) - Deg(A) = 3 - 2 = 1$$

In [5]:

```python
A_0 = s + a_0
R_prime = s + r_1
R_ = R_prime
S_ = s_0*s + s_1
T_ = A_0*B_m_prime
```

Since $Deg(B^+) = 0$ then $Deg(R^{'}) = 1$ and therfore

$R = B^+R^{'} = R^{'} = r_1 + s$

Additionally

$A_0 = a_0 + s$

$S = ss_0 + s_1$

$T = A_0B_m^{'} = \frac{a_0 + s}{c}$

In [6]:

```python
LHS = collect(expand(A*R_prime + B_minus*S_), s)
RHS = collect(expand(A_0*A_m), s)
equ = sp.Eq(LHS,RHS)
r_1_ = sp.solve(sp.Eq(LHS.coeff(s**2),RHS.coeff(s**2)), r_1)[0]
s_0_ = sp.solve(sp.Eq(LHS.coeff(s**1),RHS.coeff(s**1)), s_0)[0]
s_1_ = sp.solve(sp.Eq(LHS.subs(s,0),RHS.subs(s,0)), s_1)[0]
```

The Diophantine equation $AR^{'} + B^-S = A_0A_m$ in terms of control parameters is given by

$cs_1 + dr_1 + s^3 + s^2\left(d + r_1 + 1\right) + s\left(cs_0 + dr_1 + d + r_1\right) = a_0 + s^3 + s^2\left(a_0 + 2\right) + s\left(2a_0 + 1\right)$

Which yeilds

$r_1 = a_0 - d + 1$

$s_0 = \frac{2a_0 - dr_1 - d - r_1 + 1}{c}$

$s_1 = \frac{a_0 - dr_1}{c}$

Part 2

ODE of Plant

In [7]:

```python
y_s, u_s = sp.symbols('y(s) u(s)')

ode_RHS = ((-A.coeff(s**1)*s - A.subs(s,0))*y_s) + (B.coeff(s**2)*s**2 + B.coeff(s**1)*s**1 + B.subs(s,0))*u_s
ode_RHS
```

Out[7]: $cu(s) + y(s)(-d + s(-d-1))$

The ODE of 2^{nd} order describing the process is given by

$s^2y(s) = -(d+1)sy(s) - dy(s) + cu(s)$

where p is the time shifting operator. The reliance of the RHS of the equation on derivatives can be changed to integrals by filtering the input ($u(s)$) and output ($y(s)$) of the plant by a filter whose denominator polynomial is greater order than the derivative. The above equation becomes

$s^2y_f(s) = -(d+1)sy_f(s) - dy_f(s) + cu_f(s)$

$\Rightarrow s^2H_fy(s) = -(d+1)sH_fy(s) - dH_fy(s) + cH_fu(s)$

$\Rightarrow \frac{s^2}{A_m}y(s) = -(d+1)\frac{s}{A_m}y(s) - d\frac{1}{A_m}y(s) + c\frac{1}{A_m}u(s)$

For simplicity, let $(d+1) = x$. The ODE then becomes

$\Rightarrow \frac{s^2}{A_m}y(s) = -x\frac{s}{A_m}y(s) - d\frac{1}{A_m}y(s) + c\frac{1}{A_m}u(s)$

This equation can be further simplified as

$\Rightarrow y_2(s) = -xy_1(s) - dy_0(s) + cu_0(s)$

Bilinear Transformation of Filtered ODE

In [8]:

```python
H_f = 1/A_m
```

The filter $H_f(s)$ is given to be

$H_f(s) = \frac{1}{A_m} = \frac{1}{s^2 + 2s + 1}$

This filter ,and the ODE above, are however, in terms of s and are therfore, in continuous time domain. To converte the filter to discrete time (q), a bilinear transformation will be performed. i.e.

$s \rightarrow \frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}$

The ODE can now be represented in the discret time domain by

$y_i(kT) = H_i(q^{-1})y(kT) = \frac{s^i}{A_m(s)}\Bigg|_{s=\frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}}y(kT), \quad u_i(kT) = \frac{s^i}{A_m(s)}\Bigg|_{s=\frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}}u(kT)$

In [9]:

```python
T, q = sp.symbols('T q')

bilinear_T = (2/T)*((1 - q**(-1))/(1 + q**(-1)))

H_fy1 = collect(simplify(expand((s*H_f).subs(s,bilinear_T))), q)
H_fy0 = collect(simplify(expand((H_f).subs(s,bilinear_T))), q)
H_fu0 = collect(simplify(expand((H_f).subs(s,bilinear_T))), q)

obj_H_fy1 = numden_coeff(H_fy1, 2, 2, q)
obj_H_fy0 = numden_coeff(H_fy0, 2, 2, q)
obj_H_fu0 = numden_coeff(H_fu0, 2, 2, q)

aH_fy1 = obj_H_fy1.lst_denum_coeff
bH_fy1 = obj_H_fy1.lst_num_coeff
aH_fy0 = obj_H_fy0.lst_denum_coeff
bH_fy0 = obj_H_fy0.lst_num_coeff
aH_fu0 = obj_H_fu0.lst_denum_coeff
bH_fu0 = obj_H_fu0.lst_num_coeff
```

For $H_1(q^{-1})y(kT)$, the coefficients of the denominator αy_1 are

$\alpha y_1 = \left[1, \frac{2(T-2)}{T+2}, \frac{T^2 - 4T + 4}{T^2 + 4T + 4}\right]$

(ordered by powers of q going from q^0 to q^-2) and the coefficients of the numerator βy_1 are

$\beta y_1 = \left[\frac{2T}{T^2 + 4T + 4}, 0, -\frac{2T}{T^2 + 4T + 4}\right]$

which are also ordered by powers of q going from q^0 to q^-2. Similarly, the coefficients for the denominator (α) and numerator (β) of y_0 and u_0 are

$\alpha y_0 = \left[1, \frac{2(T-2)}{T+2}, \frac{T^2 - 4T + 4}{T^2 + 4T + 4}\right]$

$\beta y_0 = \left[\frac{T^2}{T^2 + 4T + 4}, \frac{2T^2}{T^2 + 4T + 4}, \frac{T^2}{T^2 + 4T + 4}\right]$

$\alpha u_0 = \left[1, \frac{2(T-2)}{T+2}, \frac{T^2 - 4T + 4}{T^2 + 4T + 4}\right]$

$\beta u_0 = \left[\frac{T^2}{T^2 + 4T + 4}, \frac{2T^2}{T^2 + 4T + 4}, \frac{T^2}{T^2 + 4T + 4}\right]$

Note that $\alpha y_0 = \alpha u_0$ and $\beta y_0 = \beta u_0$

In [10]:

```python
y_k, y_k_1, y_k_2  = sp.symbols('y(k) y(k-1) y(k-2)')
u_k, u_k_1, u_k_2  = sp.symbols('u(k) u(k-1) u(k-2)')
y1_k_1, y1_k_2 = sp.symbols('y_{1}(k-1) y_{1}(k-2)')
y0_k_1, y0_k_2 = sp.symbols('y_{0}(k-1) y_{0}(k-2)')
u0_k_1, u0_k_2 = sp.symbols('u_{0}(k-1) u_{0}(k-2)')

y1_k = -y1_k_1*aH_fy1[1] - y1_k_2*aH_fy1[2] + y_k*bH_fy1[0] + y_k_2*bH_fy1[2]
y0_k = -y0_k_1*aH_fy0[1] - y0_k_2*aH_fy0[2] + y_k*bH_fy0[0] + y_k_1*bH_fy0[1] + y_k_2*bH_fy0[2]
u0_k = -u0_k_1*aH_fu0[1] - u0_k_2*aH_fu0[2] + u_k*bH_fu0[0] + u_k_1*bH_fu0[1] + u_k_2*bH_fu0[2]
```

The difference equations for $y_i(kT)$ and $u_i(kT)$ are

$y_1(kT) = \frac{2Ty(k)}{T^2 + 4T + 4} - \frac{2Ty(k-2)}{T^2 + 4T + 4} - \frac{2y_1(k-1)(T-2)}{T+2} - \frac{y_1(k-2)\left(T^2 - 4T + 4\right)}{T^2 + 4T + 4}$

$y_0(kT) = \frac{T^2y(k)}{T^2 + 4T + 4} + \frac{2T^2y(k-1)}{T^2 + 4T + 4} + \frac{T^2y(k-2)}{T^2 + 4T + 4} - \frac{2y_0(k-1)(T-2)}{T+2} - \frac{y_0(k-2)\left(T^2 - 4T + 4\right)}{T^2 + 4T + 4}$

$u_0(kT) = \frac{T^2u(k)}{T^2 + 4T + 4} + \frac{2T^2u(k-1)}{T^2 + 4T + 4} + \frac{T^2u(k-2)}{T^2 + 4T + 4} - \frac{2u_0(k-1)(T-2)}{T+2} - \frac{u_0(k-2)\left(T^2 - 4T + 4\right)}{T^2 + 4T + 4}$

Therefore,

$y_2(kT) = [-y_1(kT) \; -y_0(kT) \; u_0(kT)][x \; d \; c]^T = \phi^T\theta$

Bilinear Transformation of Control Signal u(t)

In [11]:

```python
T_R = T_/R_
S_R = S_/R_

T_subd = T_
R_subd = R_.subs(r_1, r_1_)
S_subd = collect(expand(S_.subs([(s_0,s_0_), (s_1,s_1_), (r_1, r_1_)])), s)

T_R_subd = T_subd/R_subd
S_R_subd = simplify(S_subd/R_subd)

# bilinear transformation of T/R and S/R
TR = collect(simplify(expand(T_R_subd.subs(s, bilinear_T))), q)
SR = collect(simplify(expand(S_R_subd.subs(s, bilinear_T))), q)
```

The control signal of the system is given by

$u(t) = \frac{T}{R}u_c(t) - \frac{S}{R}y(t) = \frac{a_0 + s}{c\left(r_1 + s\right)}u_c(t) - \frac{ss_0 + s_1}{r_1 + s}y(t) = \frac{a_0 + s}{c\left(a_0 - d + s + 1\right)}u_c(t) - \frac{-a_0d + a_0 + d^2 - d - s\left(a_0d - a_0 - d^2 + d\right)}{c\left(a_0 - d + s + 1\right)}y(t)$

This however, must also be converted to the discrete time doamin with a bilinear transformation as well. This will be done by directly performing the transformation on $\frac{T}{R}$ and $\frac{S}{R}$ (no filtering) and using the α and β coefficients to derive difference equations for $u_c(kT)$ and $y(kT)$ respectivley.

The bilinear transformations of $\frac{T}{R}$ and $\frac{S}{R}$ are

$\frac{T}{R}\Bigg|_{s=\frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}} = \frac{-Ta_0 + q\left(-Ta_0 - 2\right) + 2}{c\left(-Ta_0 + Td - T + q\left(-Ta_0 + Td - T - 2\right) + 2\right)}$

$\frac{S}{R}\Bigg|_{s=\frac{2(1-\frac{1}{q})}{T(1+\frac{1}{q})}} = \frac{Ta_0d - Ta_0 - Td^2 + Td - 2a_0d + 2a_0 + 2d^2 - 2d + q\left(Ta_0d - Ta_0 - Td^2 + Td + 2a_0d - 2a_0 - 2d^2 + 2d\right)}{c\left(-Ta_0 + Td - T + q\left(-Ta_0 + Td - T - 2\right) + 2\right)}$

In [12]:

```python
obj_TR = numden_coeff(TR, 1, 1, q)
obj_SR = numden_coeff(SR, 1, 1, q)

aTR = obj_TR.lst_denum_coeff
bTR = obj_TR.lst_num_coeff

aSR = obj_SR.lst_denum_coeff
bSR = obj_SR.lst_num_coeff
```

For $\frac{T}{R}$, the coefficients of the numerator and denominator are

$\alpha\frac{T}{R} = \left[1, \frac{Ta_0 - Td + T - 2}{Ta_0 - Td + T + 2}\right]$

and

$\beta\frac{T}{R} = \left[\frac{Ta_0 + 2}{c\left(Ta_0 - Td + T + 2\right)}, \frac{Ta_0 - 2}{c\left(Ta_0 - Td + T + 2\right)}\right]$

while the coefficients of the numerator and denominator for $\frac{S}{R}$ are

$\alpha\frac{S}{R} = \left[1, \frac{Ta_0 - Td + T - 2}{Ta_0 - Td + T + 2}\right]$

and

$\beta\frac{S}{R} = \left[\frac{-Ta_0d + Ta_0 + Td^2 - Td - 2a_0d + 2a_0 + 2d^2 - 2d}{c\left(Ta_0 - Td + T + 2\right)}, \frac{-Ta_0d + Ta_0 + Td^2 - Td + 2a_0d - 2a_0 - 2d^2 + 2d}{c\left(Ta_0 - Td + T + 2\right)}\right]$

In [13]:

```python
uc_k, uc_k_1 = sp.symbols('u_{c}(k) u_{c}(k-1)')

uk = -u_k_1*aTR[1] + uc_k*bTR[0] + uc_k_1*bTR[1] - y_k*bSR[0] - y_k_1*bSR[1]
```

The difference equation representing the control signal becomes

$u(k) =$

$-\frac{u(k-1)\left(Ta_0 - Td + T - 2\right)}{Ta_0 - Td + T + 2} + \frac{u_c(k)\left(Ta_0 + 2\right)}{c\left(Ta_0 - Td + T + 2\right)} + \frac{u_c(k-1)\left(Ta_0 - 2\right)}{c\left(Ta_0 - Td + T + 2\right)} - \frac{y(k)\left(-Ta_0d + Ta_0 + Td^2 - Td - 2a_0d + 2a_0 + 2d^2 - 2d\right)}{c\left(Ta_0 - Td + T + 2\right)} - y(k-1)\left(-\right.$

Bilinear Transformation of Control Signal $G(s)$

In [14]:

```python
G_ = collect(simplify(expand(G.subs(s, bilinear_T))), q)

obj_G_ = numden_coeff(G_, 2, 2, q)

aG_ = obj_G_.lst_denum_coeff
bG_ = obj_G_.lst_num_coeff

yk = -y_k_1*aG_[1] - y_k_2*aG_[2] + u_k*bG_[0] + u_k_1*bG_[1] + u_k_2*bG_[2]
```

Performing a bilinear transformation on $G(s)$ yeilds

$G(kT) = \frac{T^2c\left(q^2 + 2q + 1\right)}{T^2d - 2Td - 2T + q^2\left(T^2d + 2Td + 2T + 4\right) + q\left(2T^2d - 8\right) + 4}$

To which the coefficients of the numerator and denominator are

$\beta G(kT) = \left[\frac{T^2c}{T^2d + 2Td + 2T + 4}, \frac{2T^2c}{T^2d + 2Td + 2T + 4}, \frac{T^2c}{T^2d + 2Td + 2T + 4}\right]$

and

$\alpha G(kT) = \left[1, \frac{2\left(T^2d - 4\right)}{T^2d + 2Td + 2T + 4}, \frac{T^2d - 2Td - 2T + 4}{T^2d + 2Td + 2T + 4}\right]$

The difference equation representing the output of the plant is therefore given by

$y(k) = \frac{T^2cu(k)}{T^2d + 2Td + 2T + 4} + \frac{2T^2cu(k-1)}{T^2d + 2Td + 2T + 4} + \frac{T^2cu(k-2)}{T^2d + 2Td + 2T + 4} - \frac{2y(k-1)\left(T^2d - 4\right)}{T^2d + 2Td + 2T + 4} - \frac{y(k-2)\left(T^2d - 2Td - 2T + 4\right)}{T^2d + 2Td + 2T + 4}$

part 3

In [15]:

```python
T_val = 1
a_0_val = 1

y1_k = y1_k.subs(T,T_val)
y0_k = y0_k.subs(T,T_val)
u0_k = u0_k.subs(T,T_val)
yk = yk.subs(T,T_val)
uk = uk.subs([(T,T_val), (a_0, a_0_val)])

y1_k_func = sp.lambdify([y_k, y_k_2, y1_k_1, y1_k_2], y1_k)
y0_k_func = sp.lambdify([y_k, y_k_1, y_k_2, y0_k_1, y0_k_2], y0_k)
u0_k_func = sp.lambdify([u_k, u_k_1, u_k_2, u0_k_1, u0_k_2], u0_k)
yk_func = sp.lambdify([u_k, u_k_1, u_k_2, y_k_1, y_k_2], yk)
uk_func = sp.lambdify([u_k_1, uc_k, uc_k_1, y_k, y_k_1], uk)
```

The for the implementation of the design, a sampling period of 1 ($T = 1$) and an observer polynomial parameter of 1 ($a_0 = 1$) will be used. The difference equations for $y_1(kT), y_0(kT), u_0(kT), u(kT)$ and $y(kT)$ become

$y_1(kT) = \frac{2y(k)}{9} - \frac{2y(k-2)}{9} + \frac{2y_1(k-1)}{3} - \frac{y_1(k-2)}{9}$

$y_0(kT) = \frac{y(k)}{9} + \frac{2y(k-1)}{9} + \frac{y(k-2)}{9} + \frac{2y_0(k-1)}{3} - \frac{y_0(k-2)}{9}$

$u_0(kT) = \frac{u(k)}{9} + \frac{2u(k-1)}{9} + \frac{u(k-2)}{9} + \frac{2u_0(k-1)}{3} - \frac{u_0(k-2)}{9}$

$y(kT) = \frac{cu(k)}{3d+6} + \frac{2cu(k-1)}{3d+6} + \frac{cu(k-2)}{3d+6} - \frac{2y(k-1)(d-4)}{3d+6} - \frac{y(k-2)(2-d)}{3d+6}$

$u(kT) = \frac{du(k-1)}{4-d} + \frac{3u_c(k)}{c(4-d)} - \frac{u_c(k-1)}{c(4-d)} - \frac{y(k)\left(3d^2 - 6d + 3\right)}{c(4-d)} - \frac{y(k-1)\left(-d^2 + 2d - 1\right)}{c(4-d)}$

In [16]:

```python
sample_depth = 100
sample_range = range(sample_depth)

t = [i for i in sample_range]
u_c = np.ones(sample_depth)*-1
u_c[np.where([m.sin(t[i]*m.pi/20)>=0 for i in sample_range])] = 1

c = 2
d = 0.5
x = d + 1

theta0 = np.array([x, d, c]).reshape(-1,1)
theta_hat = [np.array([0]*3).reshape(-1,1)]

y = [0]*2
u = [0]

yf1 = [0]
yf0 = [0]
uf0 = [0]

lamb = 0
I = np.identity(3)
p = 100*I

for i in range(1,sample_depth):
    phi = np.array([-yf1[-1], -yf0[-1], uf0[-1]]).reshape(-1,1)
    theta_hat.append(theta_hat[-1] + T_val*(p@phi)*(phi.T@theta0 - phi.T@theta_hat[-1]))
    p = p + T_val*(lamb*I - p@phi@phi.T)@p



plt.plot(t,u_c)
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x29d54289d60>]
```