6.310 : Nov. 9, 2026

Using LQR

① DEFINE MODEL (PREFERABLY STATE SPACE WITH KNOWN STATES): CONTROL INPUT, IMPORTANT OUTPUTS (E.G., CONTROLED OUTPUT)

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t)$$

② SELECT THE COST OUTPUT (LEAVE SOME PARAMETERS TO CHOOSE)

$$e(t) = C_e x(t) + D_e u(t)$$

(THIS DEFINES LQR WEIGHT MATRICES $Q = C_e^T C_e$, $R = D_e^T D_e$, $N = C_e^T D_e$, ACCORDING TO

$$|C_e x + D_e u|^2 = (C_e x + D_e u)^T (C_e x + D_e u) = (x^T C_e^T + u^T D_e^T)(C_e x + D_e u)$$
$$= x^T C_e^T C_e x + u^T D_e^T C_e x + x^T C_e^T D_e u + u^T D_e^T D_e u$$
$$= x^T Q x + 2 x^T N u + u^T R u$$

③ RUN $ct.lqr(A, B, Q, R, N)$ (OR $ct.lqr(A, B, QR)$ IF $N=0$):

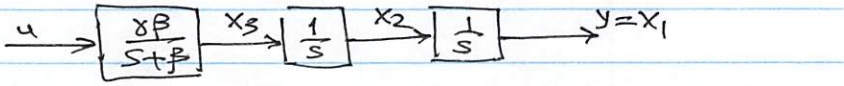
$$K, S, E = ct.lqr(A, B, Q, R, N) \quad \| \quad E: \text{eig}(A - BK)$$
$$K, S, E = ct.lqr(A, B, Q, R) \quad \| \quad S: x_0^T S x_0 = \min \int_0^\infty |e(t)|^2 dt$$

ALSO DESIGN ANY AUXILIARY FEEDBACK GAIN, IF NECESSARY (FOR EXAMPLE, DEFINE $K_r = -\frac{1}{C(A-BK)^{-1}B}$ TO USE IN FEEDBACK $u(t) = K_r y_d(t) - Kx(t)$)

④ USE SIMULATION (E.G., CLOSED LOOP STEP RESPONSE $y_d \to y$ AND $y_d \to u$) TO OBSERVE THE RESULTING PERFORMANCE

⑤ ITERATE COST PARAMETER ADJUSTMENTS, LQR OPTIMIZATION, AND SIMULATION UNTIL SATISFACTORY RESULTS ARE OBTAINED

Ex.: PROP MODEL



① $x = \begin{bmatrix} y \\ \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $\dot{x}(t) = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\beta \end{bmatrix}}_{A_0} x(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \gamma\beta \end{bmatrix}}_{B_0} u(t)$, $y(t) = \underbrace{[1, 0, 0]}_{C_y} x(t) + \underbrace{0}_{D_y} \cdot u(t)$

② $e = \begin{bmatrix} y \\ h \cdot u \end{bmatrix} = \underbrace{\begin{bmatrix} C_y \\ 0 \end{bmatrix}}_{C_e} x + \underbrace{\begin{bmatrix} D_y \\ h \end{bmatrix}}_{D_e} u \to Q = C_e^T C_e = \begin{bmatrix} C_y^T & 0 \end{bmatrix} \begin{bmatrix} C_y \\ 0 \end{bmatrix} = C_y^T C_y$

$$R = D_e^T D_e = \begin{bmatrix} 0 & h \end{bmatrix} \begin{bmatrix} 0 \\ h \end{bmatrix} = h^2$$
$$N = C_e^T D_e = \begin{bmatrix} C_y^T & 0 \end{bmatrix} \begin{bmatrix} 0 \\ h \end{bmatrix} = 0_{3\times 1}$$

LOOKING FOR FASTEST SETTLING TO WITHIN 1% OF y_d, WHILE MAINTAINING $|u(t)| \leq 5$ FOR UNIT STEP y_d] OUR SPECS

IN PYTHON: Be, Ga = 25, 50

```
Aϕ, Bϕ, Cy = np.array([[0,1,0],[0,0,1],[0,0,-Be]]), np.array([[0,0,Be*Ga]]).T,
             np.array([[1, 0, 0]])
K, S, E = ct.lqr(Aϕ, Bϕ, Cy.T@Cy, 1.0)   # 1.0 = h*h
```

K, S, E ← EIGENVALUES OF $A_o - B_o K$; $x_0^T S x_0 = \min \int_0^\infty |e(t)|^2 dt$

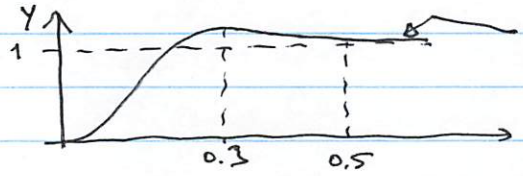
SIMULATION:

```
Acl = Aϕ - Bϕ@K
Kr = -1.0/(Cy@np.linalg.solve(Acl, Bϕ))
Gr2y = ct.ss(Acl, Bϕ@Kr, Cy, 0)   # yd → y, closed loop
Gr2u = ct.ss(Acl, Bϕ@Kr, -K, Kr)  # yd → u, closed loop
```

OPEN LOOP $\begin{cases} \dot{x} = A_o x + B_o u \\ y = C_y x \end{cases}$ ↓ closed loop with $u = K_r y_d - Kx$

CLOSED LOOP $\begin{cases} \dot{x} = (A_o - B_o K)x + K_r y_d \\ y = C_y x \\ u = -Kx + K_r y_d \end{cases}$

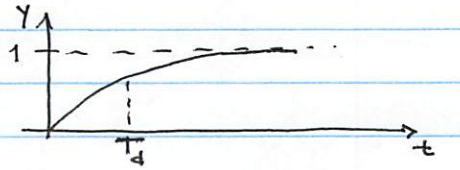
SIMULATION SHOWS LQR DESIGN WITH $h = 1$ UNDERUTILIZES CONTROL ($\max |u| \sim 1$). REDUCING h TO 0.2 PICKS UP THE SLACK, AND SPEEDS UP THE RESPONSE $y_d \to y_d$. BUT THERE IS STILL SOME OVERSHOOT... WHAT TO DO NEXT?



RESPONSE WITH $h = 0.2$

THE FAMILIAR "BETTER" RESPONSE:

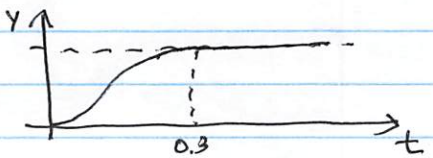
$$T_d \dot{y} + y = y_d$$



HOW ABOUT USING $e = \begin{bmatrix} T_d \dot{y} + y \\ hu \end{bmatrix}$ INSTEAD OF $e = \begin{bmatrix} y \\ hu \end{bmatrix}$?

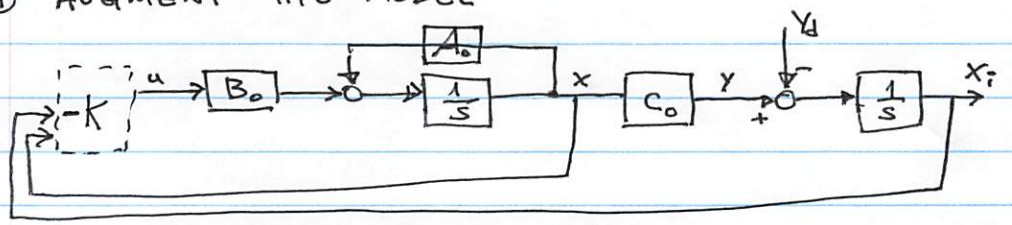
```
Td = 0.06          # FOUND BY TRIAL AND ERROR
Cyyd = Cy@Aϕ       # dy/dt = Cyyd * x
K, S, E = ct.lqr(Aϕ, Bϕ, Cyyd.T@Cyyd, 0.04)
```

THIS REDUCES 1% SETTLING TIME, AND (MOSTLY) ELIMINATES THE OVERSHOOT



USING K_r DEPENDS ON KNOWING A, B, C VERY WELL.
CAN WE TRY THE "INTEGRAL" PART?

① AUGMENT THE MODEL



$$\frac{d}{dt}\underbrace{\begin{bmatrix} x \\ x_i \end{bmatrix}}_{\text{NEW STATE: } \tilde{x}} = \underbrace{\begin{bmatrix} A_0 & 0 \\ C_0 & 0 \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x \\ x_i \end{bmatrix}}_{\tilde{x}} + \underbrace{\begin{bmatrix} B_0 \\ 0 \end{bmatrix}}_{\tilde{B}} u + \underbrace{\begin{bmatrix} 0 \\ -1 \end{bmatrix}}_{B_r \leftarrow \text{ignored by lqr design}} Y_d$$

Cost? $e = \begin{bmatrix} x_i \\ hu \end{bmatrix}$ IS MOST STRAIGHTFORWARD

$e = \begin{bmatrix} x_i + T_{di}\, y \\ hu \end{bmatrix}$ ALLOWS ONE TO CONTROL OVERSHOOT BETTER

IN PYTHON:

```
Cxi = np.array([[0,0,0,1]])
Cyi = np.array([[1,0,0,0]])
A1,Bu,Br = np.block([[Aφ, np.zeros(3,1)], Cyi]),
           np.block([[Bφ,], np.zeros(1)]), Cxi.T
Tdi = 0.05
Cxiy = Cxi + Tdi * Cyi
K,S,E = ct.lqr(A1,Bu, Cxiy.T@Cxiy, 0.000005))
Acl = A - Bu@K
Gr2y = ct.ss(Acl, Br, Cy, 0)
Gr2u = ct.ss(Acl, Br, -K, 0)
```

WITH INTEGRAL CONTROL, COMMAND IS MUCH MORE EFFECTIVE: WE CAN DROP "h" DRAMATICALLY!

} WE ARE NOT USING K_r ANY MORE!

QUIZ: IS THERE ANY NEED TO USE "K_r" WITH INTEGRAL CONTROL?
I.E., COULD $u = K_r Y_d - K\tilde{x}$ BE ANY BETTER THAN $u = -K\tilde{x}$?

ANSWER: NO! $u = -K\tilde{x}$ WILL GUARANTEE PERFECT STEADY-STATE TRACKING OF CONSTANT Y_d, HENCE THE CALCULATION OF "K_r" WOULD YIELD $K_r = 0$