3.41) Control Design via State Space

Let's revist state space modeling quickly

dynamics

$\dot{x} = Ax + Bu$ ← inputs

$y = Cx + Du$ ← output equation

A controller is trying to achieve a set of goals for y, subject to input u

→ We want to alter the behavior in A

Recall also that the eigenvalues of A are the poles of the system

→ these are also the things we have been trying to move around with our classical control techniques

→ It follows that by transforming from SS to s-space, we can use all our friendly classical contols techniques

→ However, we can also work directly through the SS matrix, and these techniques, called "Modern Control" apply to a wider range of system types including non-linear and MIMO

→ In this course, however, we'll focus on applying them to linear systems

Our classical methods have focused on controlling the locations of the dominant second order poles, then crossing our fingers that the other poles and zeros don't mess up our second order approximation too much.

- → We have a limited number of "tuning knobs"
  - → one gain
  - → one or two compensator poles or zeros

Wouldn't it be nice to just place ALL the closed loop poles wherever we want them?

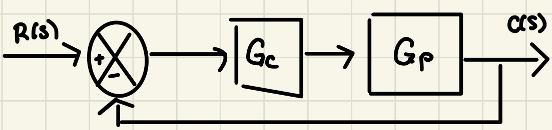
- → need n adjustable parameters to place n poles
- → and we need a way to adjust them efficently

Enter Modern Control Methods!!!

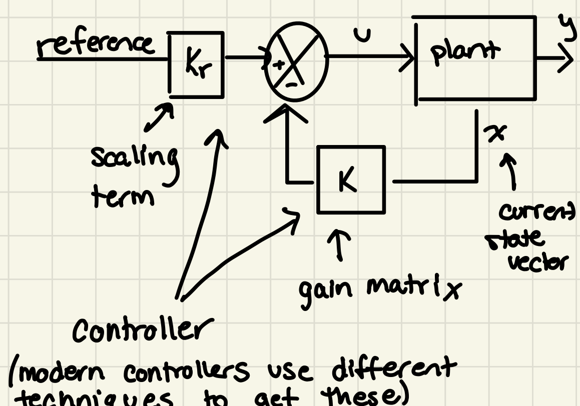
"Modern" control methods work a little differently

Let's look at a block diagram:

Classical Controls



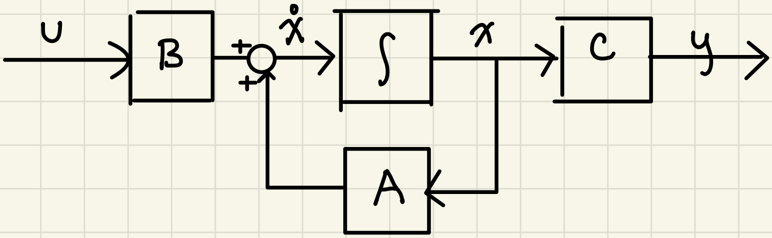
State Space Methods



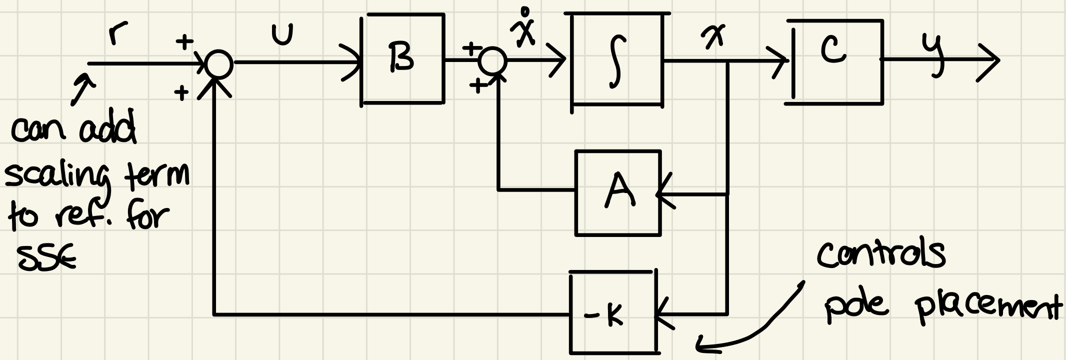
- → Pole Placement
- * → LQR *

Let's look at this block diagram more closely

$$\dot{x} = Ax + Bu$$
$$y = Cx$$



What we'd really like to do is feed back a gain on each x to the input u, allowing us to place the closed loop poles anywhere we want



→ Each state variable x_i has its own gain K_i

→ $-Kx$ is the feedback vector

The state equations for this new system are

$$\dot{x} = Ax + Bu = Ax + B(-Kx + r) = (A - BK)x + Br$$
$$y = Cx$$

The goal is to find the values of Kis to get to the desired closed loop pole locations. So here are some important questions:

1) How do I find the closed loop poles?
2) How do I decide where I want them?
3) How do I solve for the Kis to achieve this goal?

We know bits of these answers:

1) The eigenvalues of A are the poles. For the closed loop system, the eigenvalues of $(A - BK)$ will be the closed loop poles

2) We know a lot already about how to find a desired pole location from a set of design requirements

3) We need to set the answers to 1) and 2) equal to each other

Let's look at an example

Example (A really simple one)

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 6 & -1 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u$$

$$y = [1 \quad 0]\, x$$

We can find the poles of the plant through the eigenvalues of A

$$\det(A - \lambda I) = 0$$

$$\det\left(\begin{bmatrix} -\lambda & 1 \\ 6 & -1-\lambda \end{bmatrix}\right) = \lambda^2 + \lambda - 6 = 0$$

$$\lambda = -3,\ 2$$

plant is unstable

So let's add feedback and find the poles as a function of the k_i

$$A_{CL} = A - BK = \begin{bmatrix} 0 & 1 \\ 6 & -1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} [K_1 \quad K_2]$$

$$= \begin{bmatrix} -k_1 & 1-k_2 \\ 6 & -1 \end{bmatrix}$$

$$\det(A_{CL} - \lambda I) = 0$$

$$\det(A_{CL} - \lambda I) = \det\left(\begin{bmatrix} -K_1-\lambda & 1-K_2 \\ 6 & -1-\lambda \end{bmatrix}\right) = 0$$

$$(-K_1-\lambda)(-1-\lambda) - (1-K_2)(6) = 0$$

$$\lambda^2 + (1+K_1)\lambda + K_1 - 6 + 6K_2 = 0$$

$$\lambda^2 + (1+K_1)\lambda + (K_1 - 6 + 6K_2) = 0 \qquad \text{eqn } ①$$

Now let's choose a desired pole location.

We'll call them $-p_1$ and $-p_2$.

So we want $(\lambda+p_1)(\lambda+p_2) = 0$

$$\lambda^2 + (p_1+p_2)\lambda + p_1 p_2 = 0 \qquad \text{eqn } ②$$

we want eqn ①, which is the char. eqn for our CL system to be our desired char eqn, which is eqn ②

so eq ① = eqn ②

These are equal when

$$\lambda^2 + \boxed{(1+K_1)}\lambda + \boxed{(K_1 - 6 + 6K_2)} = 0$$

$$= \qquad\qquad =$$

$$\lambda^2 + \boxed{(p_1+p_2)}\lambda + \boxed{p_1 p_2} = 0$$

$(1+K_1) = (p_1 + p_2)$ and $(K_1 - 6 + 6K_2) = p_1 p_2$

so if we want poles at -3 and -5:

$1 + K_1 = 3 + 5 = 8 \quad \Rightarrow K_1 = 7$

$K_1 - 6 + 6K_2 = (3)(5) = 15$

$7 - 6 + 6K_2 = 15$

$1 + 6K_2 = 15$

$K_2 = 14/6 = 2.33$

if we want poles at -2 and -1

$(1 + K_1) = 2 + 1 = 3 \qquad K_1 = 2$

$K_1 - 6 + 6K_2 = (2)(1) = 2$

$2 - 6 + 6K_2 = 2$

$6K_2 = 6$

$K_2 = 1$

and so by solving for these ks we can put our poles anywhere we want!

You can imagine that as the number of state variables increase that this quickly becomes difficult to do by hand

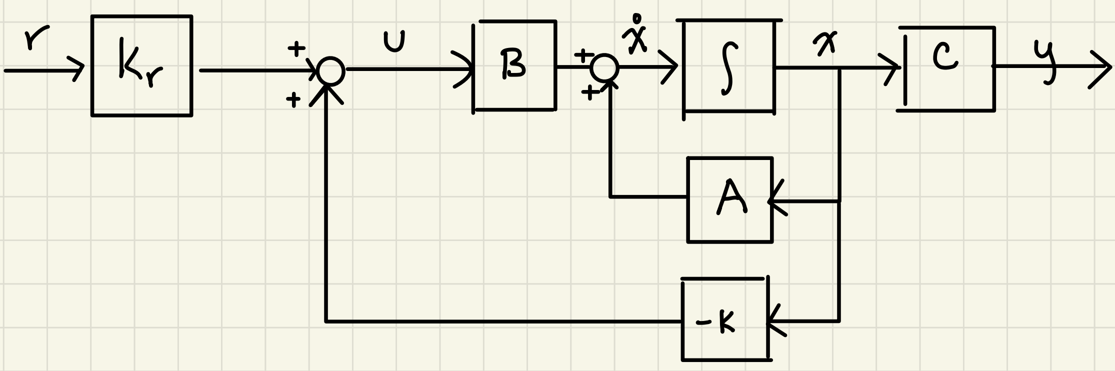
So let's repeat our example in Matlab

poleplacementex1.m

Okay, so it's stable, but we have a SS error problem.

We need to scale the system by some value to reduce the SSE. We call this k_r

there is an easy way to do this → dcgain computes the low frequency gain of a system. We know the low frequency gain is related to SSE. So we can just adust this gain value to get to our desired SSE



$$K_r = \frac{1}{\text{dcgain}}$$

The state equations for this new system are

$$\dot{x} = Ax + Bu = Ax + B(-Kx + k_r r) = (A - BK)x + Bk_r r$$
$$y = Cx$$

This technique is called pole placement and it lets you solve for a set of gains, one for each state, that let you put your poles ANYwhere

so this is great! why would we ever need anything else? I know how to put anything into SS, and then I write all of 6 lines of Matlab code and BAM- perfect control, every time!

Well.... it's not always that easy....

These gains are sort of like PD controllers → think about a 2-state system → one gain for the state x_1, one for the derivative state x_2

→ important practical difference

→ PD controllers find D in controller

→ PP controllers feed D back as a state

You need to know the states to feed them back!

→ observe directly w/ sensor

→ estimate them from what you can observe

Practical considerations on possible response time

Beyond that, we actually need to be able to influence the states through the control signal u. If we can't actually influence them, we can't control them.

So our states need to be controllable and observable or this isn't going to work!