10. The Linear Quadratic Regulator

- Regulation and the least squares formulation of regulation
- The LQR problem formulation
- Constrained optimization formulation
- Dynamic programming
- example: path optimization
- Solving the Hamilton-Jacobi equation
- The Riccati recursion
- Summary of LQR solution via DP
- Example: force on mass
- The steady-state regulator
- Time-varying systems and tracking problems
- Infinite-horizon problems
- The Algebraic Riccati equation

The Key Points of This Section

- idea of regulation; keep the output small, using as little input as possible
- multi-objective problem: allows trade-off to be made between input effort and regulation
- can be formulated as a large least squares problem

- instead, solve it via dynamic programming
- solution is *Riccati recursion*; much faster to compute
- controller is *linear state feedback* $u(t) = K_t x(t)$
- we often use the steady-state solution; to find it, solve the *Algebraic Riccati Equation*

Regulation

usual discrete-time system

$$x(t+1) = Ax(t) + Bu(t) \qquad x(0) = x_0$$
$$y(t) = Cx(t)$$

multiobjective problem

- *regulation:* keep $y(t)$ small on $t = 0, \dots, N-1$; we'd like to keep small

$$J_{\text{out}} = \sum_{t=0}^{N-1} \|y(t)\|^2$$

- using *low input effort*; we'd like to keep small

$$J_{\text{in}} = \sum_{t=0}^{N-1} \|u(t)\|^2$$

least-squares formulation

as before we have

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} 0 & & & & \\ CB & 0 & & & \\ CAB & CB & 0 & & \\ \vdots & & & \ddots & \\ CA^{N-2}B & CA^{N-3}B & \dots & CB & 0 \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ u(2) \\ \vdots \\ u(N-1) \end{bmatrix} + \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{N-1} \end{bmatrix} x(0)$$

$$= L u_{\mathsf{seq}} + M x_0$$

multiobjective least squares problem:

$$\begin{aligned} J_{\mathsf{out}}(u_{\mathsf{seq}}) + \mu J_{\mathsf{in}}(u_{\mathsf{seq}}) &= \|L u_{\mathsf{seq}} + M x_0\|^2 + \mu \|u_{\mathsf{seq}}\|^2 \\ &= \left\| \begin{bmatrix} L \\ \sqrt{\mu} I \end{bmatrix} u_{\mathsf{seq}} + \begin{bmatrix} M x_0 \\ 0 \end{bmatrix} \right\|^2 \end{aligned}$$

least-squares solution is *open-loop*; does not use measurements of $x(t)$ on $t = 0, \dots, N-1$

cost function

$$J(u_{\mathsf{seq}}) = J_{\mathsf{out}}(u_{\mathsf{seq}}) + \mu J_{\mathsf{in}}(u_{\mathsf{seq}}) = \sum_{t=0}^{N-1} \|y(t)\|^2 + \mu\|u(t)\|^2$$

$$= \sum_{t=0}^{N-1} x(t)^T C^T C x(t) + \mu u(t)^T u(t)$$

we'll use the slightly more general cost function

$$\boxed{J(u_{\mathsf{seq}}) = \sum_{t=0}^{N-1} \Big(x(t)^T Q x(t) + u(t)^T R u(t)\Big) + x(N)^T Q_f x(N)}$$

where

$$Q \geq 0 \qquad Q_f \geq 0 \qquad R > 0$$

are called *state cost*, *final state cost*, and *input cost* matrices

cost function

$$J(u_{\mathsf{seq}}) = \sum_{t=0}^{N-1} \Big(x(t)^T Q x(t) + u(t)^T R u(t) \Big) + x(N)^T Q_f x(N)$$

- N is called the *time horizon*
- first term measures *state deviation*
- second term measures *input size* or *actuator authority*
- last term measures *final state deviation*
- Q, R set relative weights of state deviation and input usage
- $R > 0$ means any (nonzero) input adds to the cost J
- we often use $Q = Q_f = C^T C$ and $R = \mu I$

LQR problem

$$\boxed{\text{find } u(0), \ldots, u(N-1) \text{ that minimizes } J(u_{\mathsf{seq}})}$$

Constrained Optimization

the LQR problem can be written as the *constrained optimization*

$$\begin{aligned} \text{minimize} \quad & \sum_{t=0}^{N-1} \Big(x(t)^T Q x(t) + u(t)^T R u(t) \Big) + x(N)^T Q_f x(N) \\ \text{subject to} \quad & x(0) = x_0 \\ & x(t+1) = Ax(t) + Bu(t) \qquad \text{for } t = 0, \dots, N-1 \end{aligned}$$

- the variables are $u(0), \dots, u(N-1), x(0), \dots, x(N)$
- the dynamics are *constraints*

Dynamic Programming Solution

- gives an *efficient, recursive* method to solve LQR problem
- gives a *feedback* solution
- useful and important idea on its own

for $t = 0, \ldots, N$ define the *value function* V_t by

$$\boxed{\begin{gathered} V_t(z) = \min_{u(t),\ldots,u(N-1)} \sum_{\tau=t}^{N-1} \Big(x(\tau)^T Q x(\tau) + u(\tau)^T R u(\tau) \Big) + x(N)^T Q_f x(N) \\ \text{subject to } x(t) = z, \text{ and } x(\tau+1) = A x(\tau) + B u(\tau) \end{gathered}}$$

- $V_t(z)$ gives the *min-cost-to-go* starting from state z at time t
- $V_0(x_0)$ is the min LQR cost

dynamic programming

we'll find that

- V_t is *quadratic*, i.e., $V_t(z) = z^T P_t z$, where $P_t = P_t^T$ and $P_t \geq 0$
- P_t can be found *recursively*, working backward from $t = N$
- the optimal u is easily expressed in terms of P_t

cost-to-go with no time left is just final state cost

$$V_N(z) = z^T Q_f z$$

therefore $P_N = Q_f$

dynamic programming principle

now suppose we know $V_{t+1}(z)$, what is the optimal choice for $u(t)$?

choice of $u(t)$ affects

- current cost incurred (through $u(t)^T R u(t)$)
- where we land, i.e., $x(t+1)$ (hence the min-cost-to-go from $x(t+1)$)

dynamic programming principle

$$\boxed{V_t(z) = \min_w \Big(z^T Q z + w^T R w + V_{t+1}(Az + Bw) \Big)}$$

dynamic programming principle

let's look at this again

$$\boxed{V_t(z) = \min_w \Big(z^T Q z + w^T R w + V_{t+1}(Az + Bw) \Big)}$$

- $z^T Q z + w^T R w$ is cost incurred at time t if $u(t) = w$
- $V_{t+1}(Az + Bw)$ is min-cost-to-go from where you land at $t + 1$

follows from fact that we can minimize in any order

$$\min_{w_1,\ldots,w_k} f(w_1,\ldots,w_k) = \min_{w1} \underbrace{\left(\min_{w_2,\ldots,w_k} f(w_1,\ldots,w_k) \right)}_{\text{a function of } w_1}$$

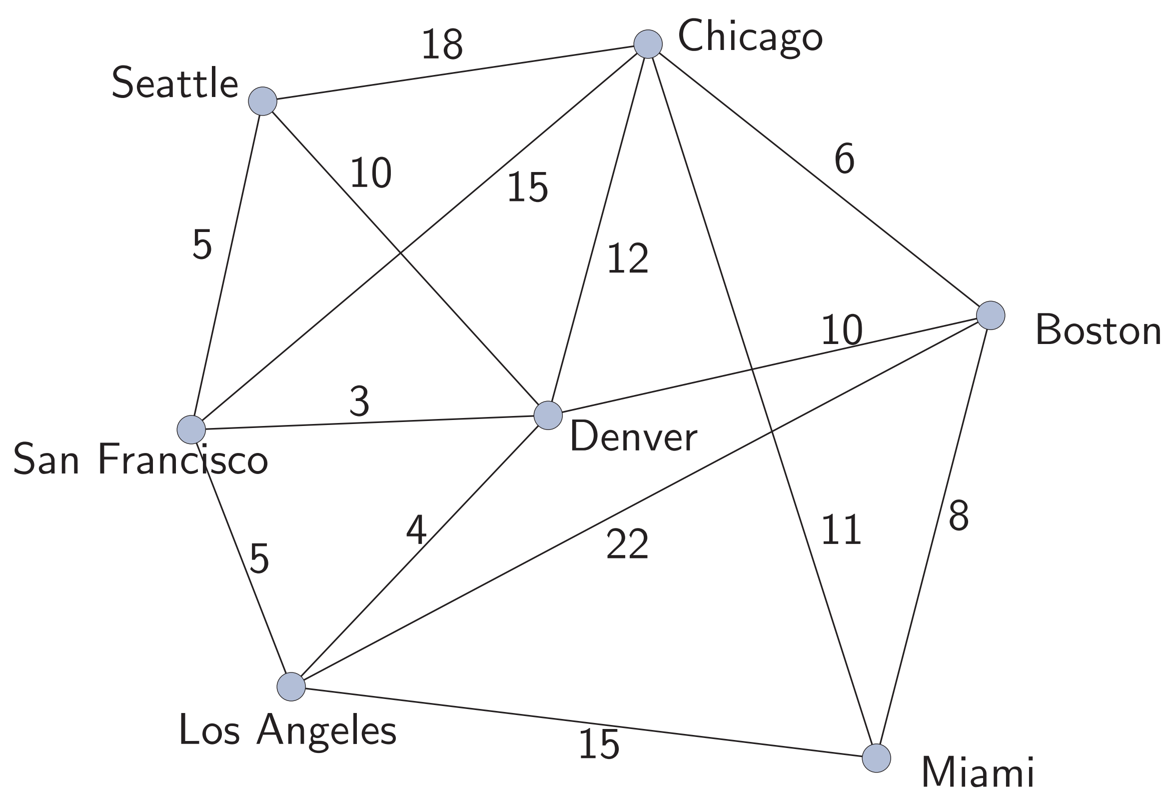
in words

min-cost-to-go from where you are = min over

(current cost incurred + min-cost-to-go from where you land)

Example: Path Optimization

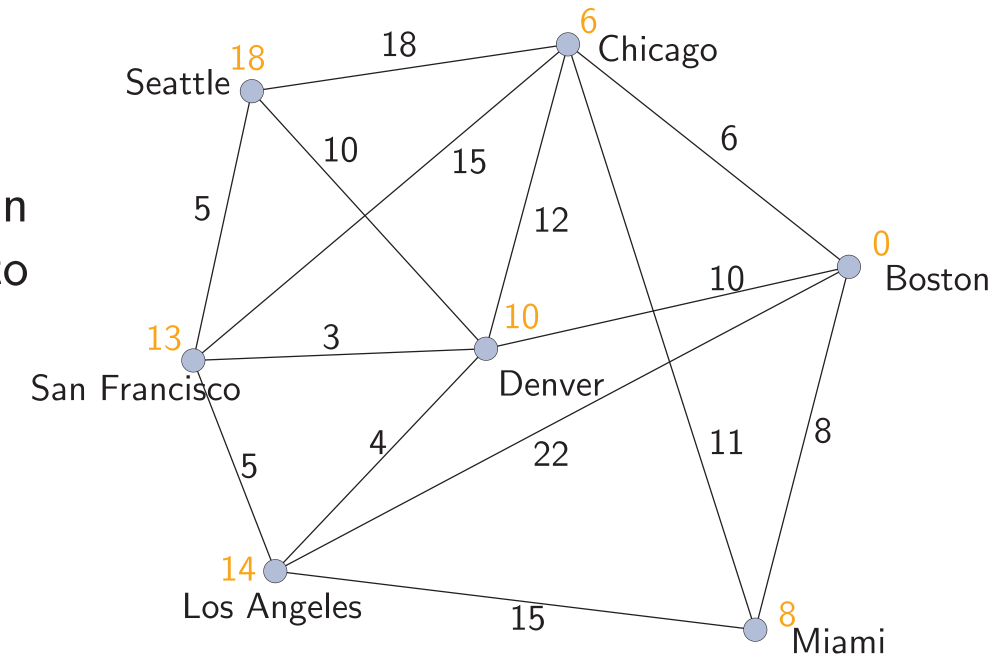
- want to find min cost route or path from Seattle to Boston
- you can fly from point i to point j at cost c_{ij}



dynamic programming approach

value function

$$V(j) = \text{min cost from point } j \text{ to Boston}$$



- if we know $V(j)$ for all j, then we can find min cost route from any point to Boston

- dynamic programming principle

$$V(i) = \min_j \big(A_{ij} + V(j)\big)$$

- from point i, fly to j where

$$A_{ij} + V(j) = \min_{j \text{ adjacent}} \big(A_{ij} + V_j\big)$$

Hamilton-Jacobi equation

$$V_t(z) = \min_w \left(z^T Q z + w^T R w + V_{t+1}(Az + Bw) \right)$$

- called DP, Bellman, or Hamilton-Jacobi equation
- gives V_t recursively, in terms of V_{t+1}
- any minimizing w gives optimal $u(t)$

DP has many applications beyond LQR, e.g.,

- optimal flow control in communication networks
- optimization in finance

solving the Hamilton-Jacobi equation

we know $V_N(z) = z^T P_N z$ where $P_N = Q_f$

by DP

$$V_{N-1}(z) = z^T Q z + \min_w \Big(w^T R w + (Az + Bw)^T P_N (Az + Bw) \Big)$$

solve by setting derivative w.r.t. w to zero

$$2w^T R + 2(Az + Bw)^T P_N B = 0$$

hence optimal w is

$$w_{\text{opt}} = -(R + B^T P_T B)^{-1} B^T P_N A z$$

solving the Hamilton-Jacobi equation

and so

$$\begin{aligned} V_{N-1}(z) &= z^T Q z + w_{\text{opt}}^T R w_{\text{opt}} + (Az + Bw_{\text{opt}})^T P_N (Az + Bw_{\text{opt}})\Big) \\ &= z^T (Q + A^T P_N A - A^T P_N B (R + B^T P_N B)^{-1} B^T P_N A) z \end{aligned}$$

(after some ugly algebra)

we conclude that V_{N-1} is quadratic: $V_{N-1}(z) = z^T P_{N-1} z$ where

$$P_{N-1} = Q + A^T P_N A - A^T P_N B (R + B^T P_N B)^{-1} B^T P_N A$$

Riccati recursion

this recursion works for all t

once we know $V_t(z) = z^T P_t z$ is quadratic, we find that V_{t-1} is as well, i.e., $V_{t-1}(z) = z^T P_{t-1} z$ where

$$\boxed{P_{t-1} = Q + A^T P_t A - A^T P_t B(R + B^T P_t B)^{-1} B^T P_t A}$$

since we know $P_N = Q_f$, we can find $P_{N-1}, P_{N-2}, \ldots, P_0$ in turn by recursion, backward in time

called *Riccati recursion* for P_t; then the optimal control input is

$$\boxed{u_{\text{opt}}(t) = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A x(t)}$$

Summary of LQR Solution via DP

1. set $P_N = Q_f$

2. for $t = N-1, \dots, 0$

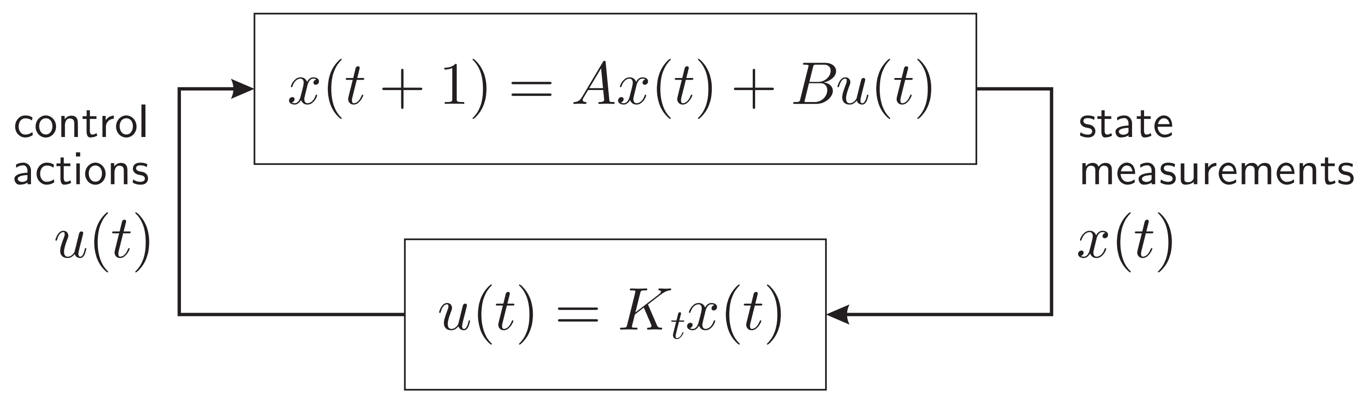
   compute P_t from P_{t+1} via

$$P_t = Q + A^T P_{t+1} A - A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$$

3. define $K_t = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$

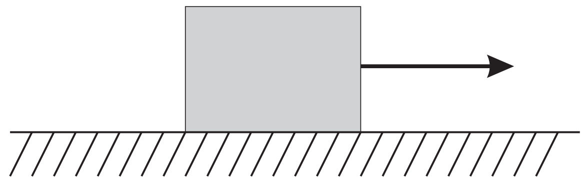
4. optimal u is given by $u_{\mathsf{opt}}(t) = K_t x(t)$

properties of LQR solution



- optimal u is a linear function of the state (called *linear state feedback*)
- recursion for min cost-to-go runs backward in time
- solves least squares problem in Nm variables much faster than direct least squares method
- compute $K_0, \ldots, K_{N-1}$ in advance; we don't need to know $x(0)$
- at run-time, $u(t) = K_t x(t)$; we need only know $x(t)$, not past states

example: force on mass



Newton's law discretized, with sample period $h = 1$ gives

$$x(t+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix} u(t)$$
$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t)$$

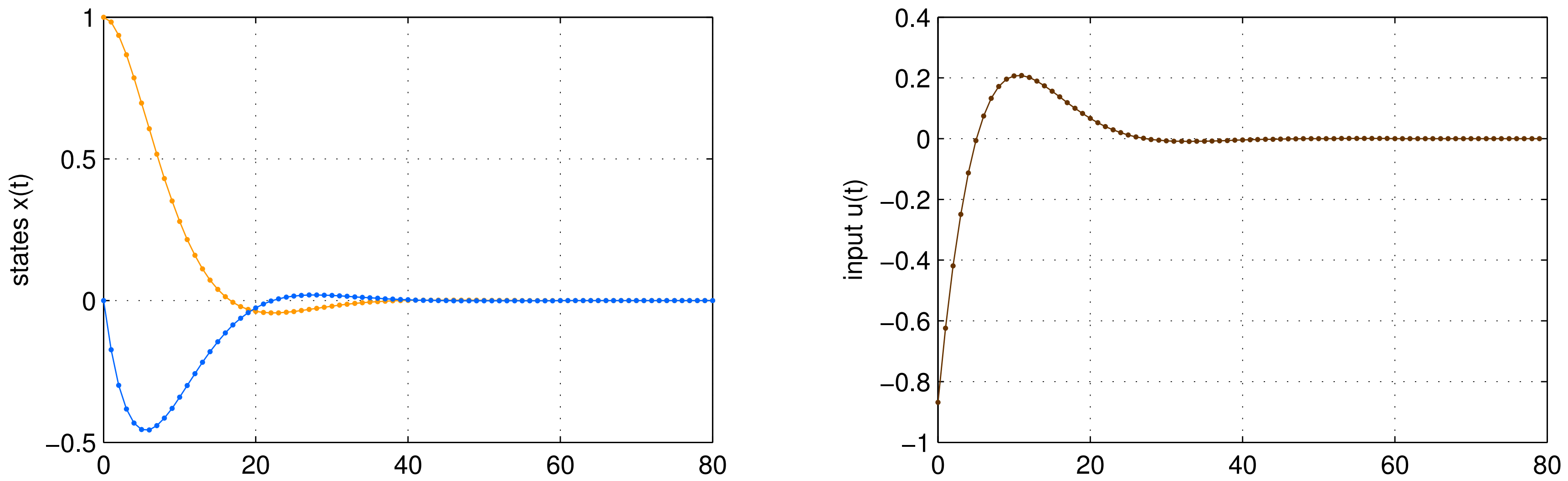
Cost function

$$J_{\mathsf{out}}(u_{\mathsf{seq}}) + \mu J_{\mathsf{in}}(u_{\mathsf{seq}}) = \sum_{t=0}^{N-1} \Big(y(t)^T y(t) + \mu u(t)^T u(t) \Big) + y(N)^T y(N)$$

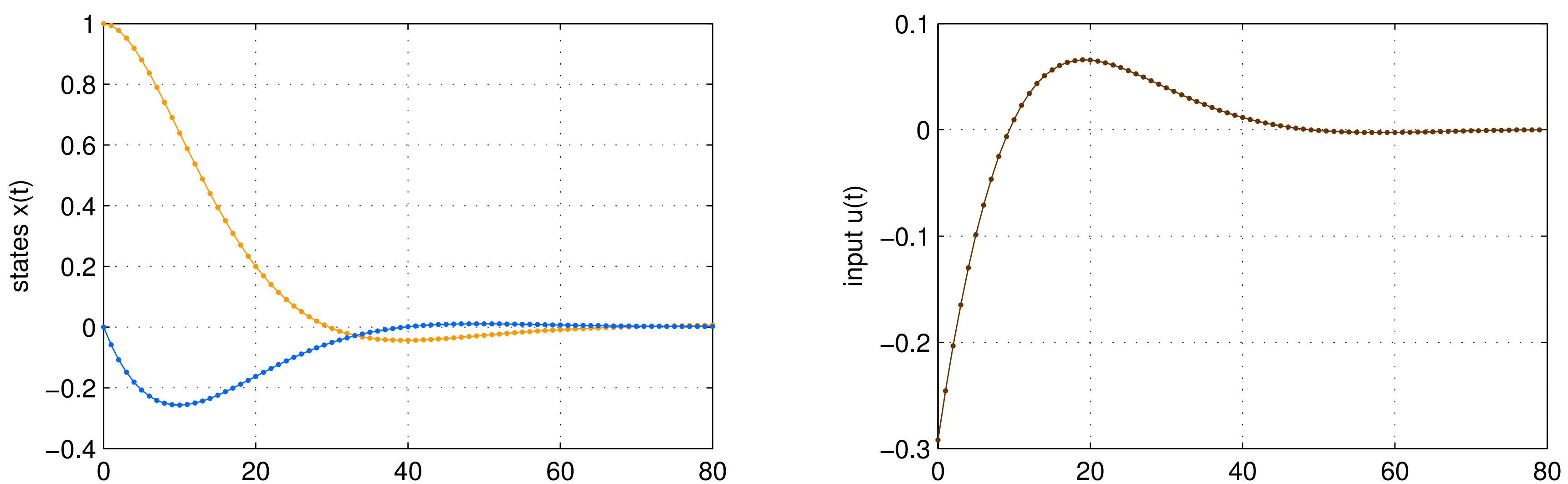
corresponds to $Q = Q_f = C^T C$ and $R = \mu I$.

example: states and inputs

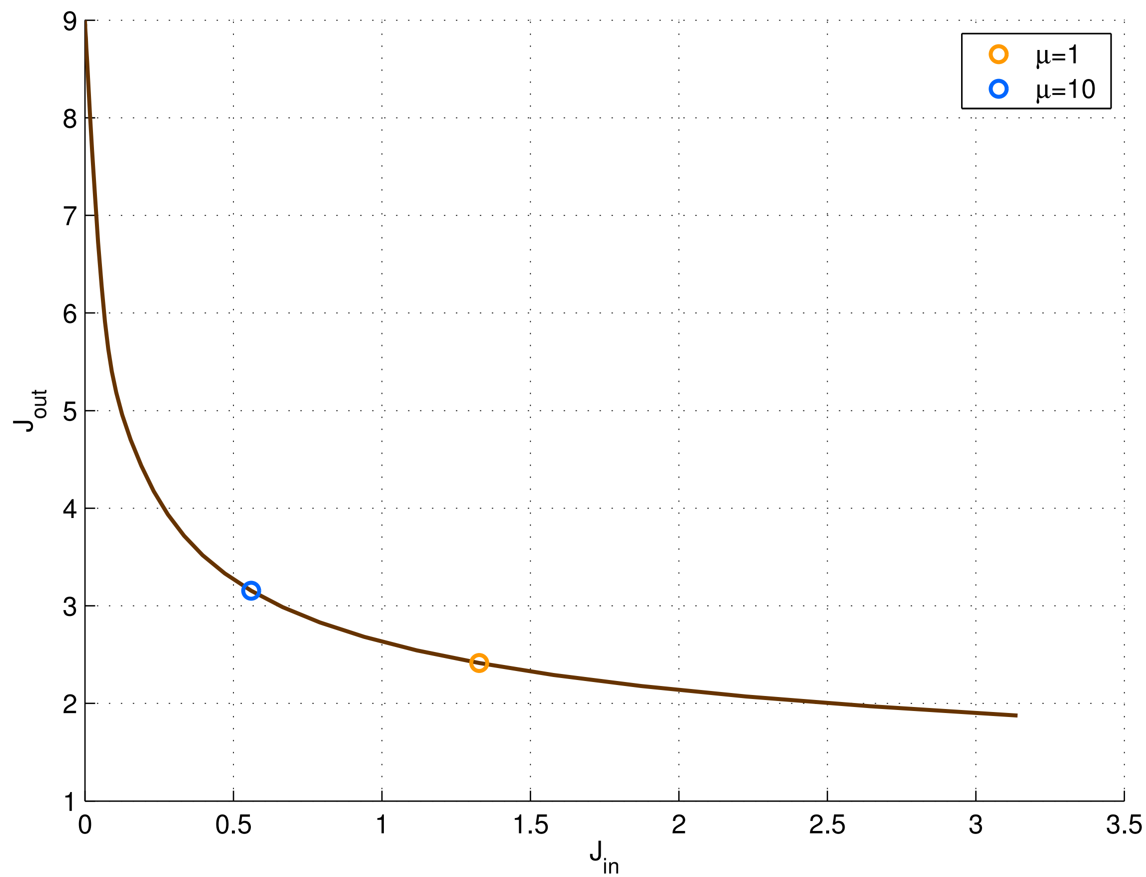
for $\mu = 1$



for $\mu = 10$, controller uses less control effort



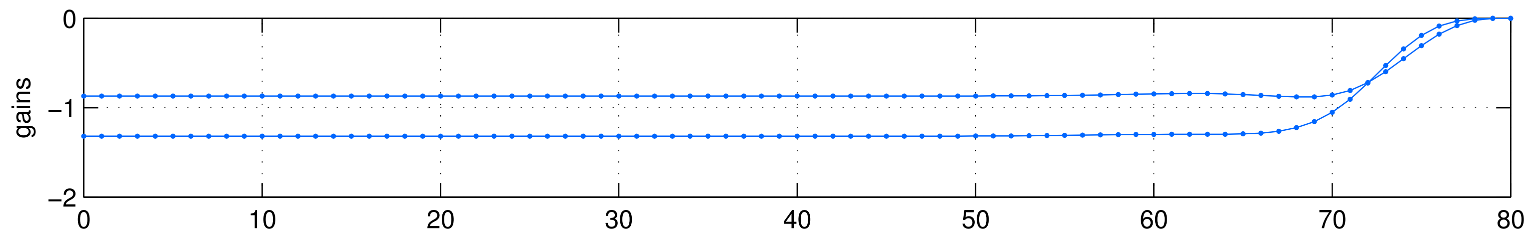
example: trade-off curve



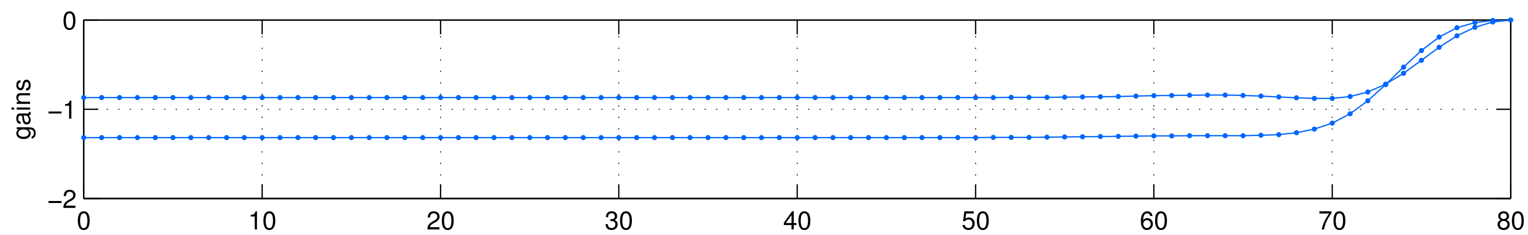
example: gains

state feedback gains for various Q_f; note convergence

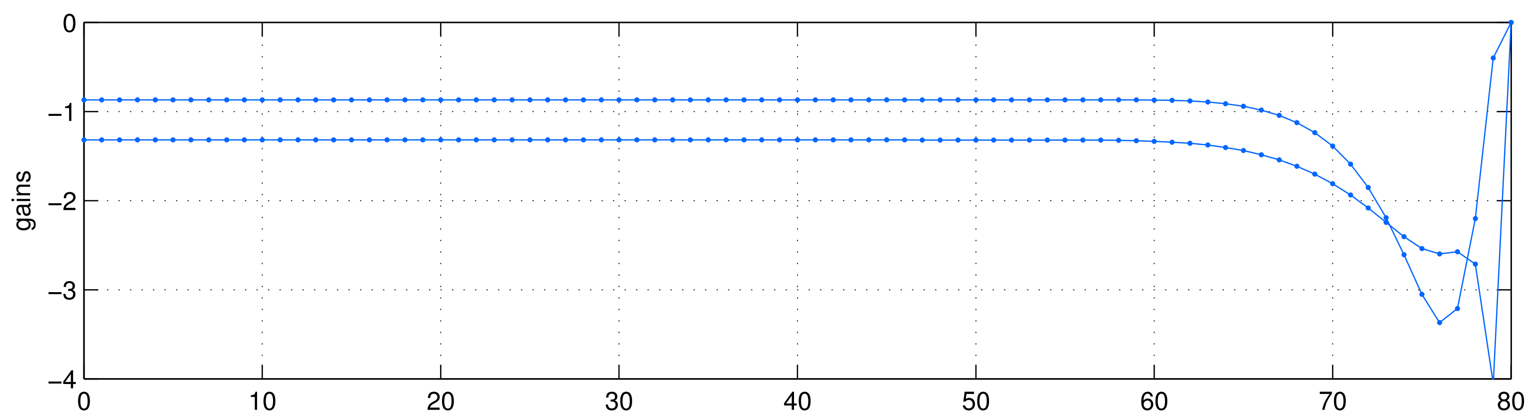
$Q_f = 0$



$Q_f = Q$



$Q_f = 100I$



steady-state regulator

usually P_t rapidly converges as t decreases below N

limit or steady-state value P_{ss} satisfies

$$\boxed{P_{\mathrm{ss}} = Q + A^T P_{\mathrm{ss}} A - A^T P_{\mathrm{ss}} B (R + B^T P_{\mathrm{ss}} B)^{-1} B^T P_{\mathrm{ss}} A}$$

which is called the (discrete-time) *Algebraic Riccati Equation* (ARE)

- P_{ss} can be found by iterating the Riccati recursion, or by direct methods
- for t not close to the horizon, LQR optimal control is approximately a linear, constant state feedback

$$u(t) = K_{\mathrm{ss}} x(t) \qquad K_{\mathrm{ss}} = -(R + B^T P_{\mathrm{ss}} B)^{-1} B^T P_{\mathrm{ss}} A$$

- very widely used in practice

Time-varying Systems

LQR is readily extended to handle time-varying systems

$$x(t+1) = A(t)x(t) + B(t)u(t)$$

and time-varying cost matrices

$$J = \sum_{t=0}^{T-1} \Big(x(t)^T Q(t) x(t) + u(t)^T R(t) u(t) \Big) + x(T)^T Q_f x(T)$$

(so Q_f is really $Q(T)$)

DP solution is readily extended, but (of course) there need not be a steady-state solution

Tracking Problems

we consider LQR cost with state and input offsets

$$J = \sum_{t=0}^{T-1} \big(x(t) - \bar{x}(t)\big)^T Q \big(x(t) - \bar{x}(t)\big) + \sum_{t=0}^{T-1} \big(u(t) - \bar{u}(t)\big)^T R \big(u(t) - \bar{u}(t)\big)$$

(we drop the final state term for simplicity)

here $\bar{x}(t)$ and $\bar{u}(t)$ are given desired state and input trajectories.

DP solution is readily extended, even to time-varying tracking problems

Infinite Horizon LQR Problem

discrete-time system

$$x(t+1) = Ax(t) + Bu(t) \qquad x(0) = x_0$$

infinite-horizon LQR problem: choose $u(0), u(1), \ldots$ to minimize

$$J = \sum_{t=0}^{\infty} \Big(x(t)^T Q x(t) + u(t)^T R u(t) \Big)$$

with given constant state and input weighting matrices

$$Q \geq 0 \qquad R > 0$$

we have an infinite number of variables

finiteness of the cost

problem: it's possible that $J = \infty$ for all possible input sequences $u(0), u(1), \ldots$

for example

$$x(t+1) = 2x(t) + 0u(t) \qquad x(0) = 1$$

this cannot happen if (A, B) is controllable; then for any x_0 there's an input sequence

$$u(0), u(1), \ldots, u(n-1), 0, 0, \ldots$$

that steers x to zero at $t = n$ and keeps it there

for this u, $J < \infty$, and so $\min\limits_u J < \infty$ for any x_0

dynamic programming solution

define the *value function* $V : \mathbb{R}^n \to \mathbb{R}$

$$V(z) = \min_{u(0),u(1),\ldots} \sum_{t=0}^{\infty} \Big(x(t)^T Q x(t) + u(t)^T R u(t) \Big)$$

subject to $x(0) = z$, $x(t+1) = Ax(t) + Bu(t)$ for all $t \geq 0$

- $V(z)$ is the min-cost-to-go, starting from state z
- doesn't depend on time-to-go, which is always ∞; infinite horizon problem is *shift-invariant*

Hamilton-Jacobi equation

the HJ equation is

$$\boxed{V(z) = \min_w \Big(z^T Q z + w^T R w + V(Az + Bw) \Big)}$$

fact: V is quadratic, i.e., $V(z) = z^T P z$ where $P = P^T$ and $P \geq 0$
(can be argued directly from first principles)

so

$$z^T P z = \min_w \Big(z^T Q z + w^T R w + (Az + Bw)^T P (Az + Bw) \Big)$$

and the minimizing w is

$$w_{\mathsf{opt}} = -(R + B^T P B)^{-1} B^T P A z$$

Riccati equation

the HJ equation is

$$\begin{aligned} z^T P z &= z^T Q z + w_{\text{opt}}^T R w_{\text{opt}} + (Az + Bw_{\text{opt}})^T P (Az + Bw_{\text{opt}}) \\ &= z^T \big(Q + A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A\big) z \end{aligned}$$

this must hold for all z, so P satisfies the ARE

$$\boxed{P = Q + A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A}$$

compared to finite-horizon LQR problem

- value function and optimal state feedback gains are time-invariant
- we don't have a recursion to compute P; we only have the ARE

solving the ARE

if A, B is controllable and A, C is observable (recall $Q = C^T C$)

- the ARE has exactly one positive semidefinite solution P
- the Riccati recursion

$$P_{k-1} = Q + A^T P_k A - A^T P_k B(R + B^T P_k B)^{-1} B^T P_k A$$

  converges to P as $k \to -\infty$, from any 'initial condition' $P(T)$

so infinite-horizon LQR is the same as steady-state finite horizon LQR

summary of infinite-horizon LQR

to find the optimal infinite-horizon LQR controller

1. solve the ARE

$$P = Q + A^T P A - A^T P B(R + B^T P B)^{-1} B^T P A$$

   in Matlab, can use `dlqr`

2. set K to

$$K = -(R + B^T P B)^{-1} B^T P A$$

   and use state-feedback controller $u(t) = Kx(t)$

the closed-loop system

we have $x(t+1) = Ax(t) + Bu(t)$, so with the controller $u(t) = Kx(t)$,

$$x(t+1) = (A + BK)x(t)$$

is closed-loop system stable? look at the example

$$x(t+1) = 2x(t) + u(t) \qquad Q = 0, \quad R = 1$$

optimal control is $u(t) = 0$; so closed-loop system is unstable

if A, B is controllable and A, C is observable, then the closed-loop system is stable