L5: Linear Programming and Connections with Continuous Control

EECE 571N | Sequential Decision Making | Fall 2025

Cyrus Neary | cyrus.neary@ubc.ca

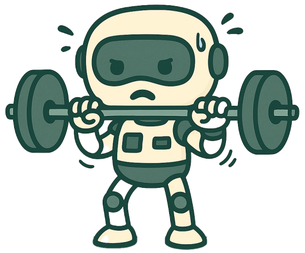
Last class

Bellman's equation and operator

$$V^{\pi}(s) = \sum_{a \in A} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^{\pi}(s') \right] \qquad (T_{\pi}V)(s) = \sum_{a \in A} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V(s') \right]$$

Bellman's optimality equation and operator

$$V^{*}(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^{*}(s') \right] \qquad (T_{*}V)(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V(s') \right]$$



Policy iteration

Policy evaluation

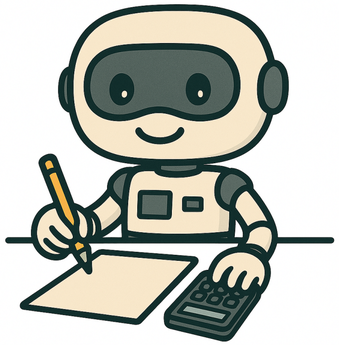
$$V_{\pi} = (I - \gamma P_{\pi})^{-1} R_{\pi}$$

For k = 0, 1, 2, …

$$V_{k+1} \leftarrow T_{\pi} V_k$$

Policy improvement

$$\pi'(s) \in \text{argmax}_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^{\pi}(s') \right]$$



Value iteration

For k = 0, 1, 2, …

$$V_{k+1} \leftarrow T_{*} V_k$$

$$\pi^{*}(s) \in \text{argmax}_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^{*}(s') \right]$$

Linear Programming Formulations of MDPs

Linear programs

https://en.wikipedia.org/wiki/Linear_programming#Standard_form

Standard form [edit]

Standard form is the usual and most intuitive form of describing a linear programming problem. It consists of the following three parts:

- A **linear (or affine) function to be maximized**

  e.g. $f(x_1, x_2) = c_1 x_1 + c_2 x_2$

- **Problem constraints** of the following form

  e.g.

  $$\begin{aligned} a_{11} x_1 + a_{12} x_2 &\leq b_1 \\ a_{21} x_1 + a_{22} x_2 &\leq b_2 \\ a_{31} x_1 + a_{32} x_2 &\leq b_3 \end{aligned}$$

- **Non-negative variables**

  e.g.

  $$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

$\max_x c^T x$

$Ax \leq b$

$x \geq 0$

The problem is usually expressed in *matrix form*, and then becomes:

$$\max\{\, \mathbf{c}^\mathsf{T}\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^n \wedge A\mathbf{x} \leq \mathbf{b} \wedge \mathbf{x} \geq 0 \,\}$$

Other forms, such as minimization problems, problems with constraints on alternative forms, and problems involving negative variables can always be rewritten into an equivalent problem in standard form.

Intuition and big picture

Key idea: Find the value function that satisfies Bellman optimality as a set of linear constraints.

In comparison:

- Value iteration finds a fixed point of the Bellman operator.
- LP solutions directly solve constraints with optimization algorithms.

Why study LP solutions to MDPs?

- Easy to incorporate additional constraints.
- Easy to incorporate different optimization objectives.
- Intuition on MDP solutions, and elegant theory.

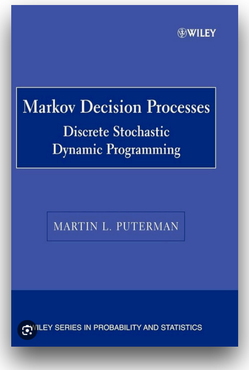
Upper bound on optimal value function

If you can find a candidate value function $f(s)$ such that

$$f(s) \geq R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) f(s'), \quad \forall a \in A_s, s \in S$$

Then $f(s)$ provides an upper bound on the MDP's optimal value function.

$$f(s) \geq V^*(s)$$



See Theorem 6.2.2

Primal LP Formulation

Idea: find the smallest candidate function $f(s)$, subject to the constraint that $f(s)$ upper bounds the MDP's optimal value function.

Primal LP Formulation

Idea: find the smallest candidate function $f(s)$, subject to the constraint that $f(s)$ upper bounds the MDP's optimal value function.

$$\min_{f} \sum_{s \in S} \alpha(s) f_s$$

$$\text{s.t.} \quad f_s \geq R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s, a) f'_s, \quad \forall a \in A_s, s \in S$$

$$f_s \in \mathbb{R}^{|S|} \text{ unconstrained}$$

Primal LP Formulation

Idea: find the smallest candidate function $f(s)$, subject to the constraint that $f(s)$ upper bounds the MDP's optimal value function.

$$\min_{f} \sum_{s \in S} \alpha(s) f_s$$

$$\text{s.t.} \quad f_s \geq R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s, a) f'_s, \quad \forall a \in A_s, s \in S$$

$$f_s \in \mathbb{R}^{|S|} \text{ unconstrained}$$

Considering MDPs with finite state and action spaces, we need to search over vectors $f \in \mathbb{R}^{|S|}$.

Solving this LP yields the MDP's optimal value function.

Dual LP formulation

It is more informative to consider the dual LP.

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

$$\text{s.t.} \quad \sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

Dual linear programs

https://en.wikipedia.org/wiki/Dual_linear_program

Dual linear program

5 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

The **dual** of a given linear program (LP) is another LP that is derived from the original (the **primal**) LP in the following schematic way:

- Each variable in the primal LP becomes a constraint in the dual LP;
- Each constraint in the primal LP becomes a variable in the dual LP;
- The objective direction is inversed – maximum in the primal becomes minimum in the dual and vice versa.

The **weak duality theorem** states that the objective value of the dual LP at any feasible solution is always a bound on the objective of the primal LP at any feasible solution (upper or lower bound, depending on whether it is a maximization or minimization problem). In fact, this bounding property holds for the optimal values of the dual and primal LPs.

The **strong duality theorem** states that, moreover, if the primal has an optimal solution then the dual has an optimal solution too, *and the two optima are equal*.[1]

Form of the dual LP [edit]

Suppose we have the linear program:

Maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \leq \mathbf{b}$, $\mathbf{x} \geq 0$.

We would like to construct an upper bound on the solution. So we create a linear combination of the constraints, with positive coefficients, such that the coefficients of $\mathbf{x}$ in the constraints are at least $\mathbf{c}^T$. This linear combination gives us an upper bound on the objective. The variables $\mathbf{y}$ of the dual LP are the coefficients of this linear combination. The dual LP tries to find such coefficients that *minimize* the resulting upper bound. This gives the following LP:[1]:81–83

Minimize $\mathbf{b}^T\mathbf{y}$ subject to $A^T\mathbf{y} \geq \mathbf{c}$, $\mathbf{y} \geq 0$

This LP is called the *dual of* the original LP.

Primal LP

$$\min_{f \in \mathbb{R}^{|S|}} \sum_s \alpha(s) f_s$$

$$\text{s.t.} \quad f_s \geq R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) f_{s'} \quad \forall s,a$$

Dual LP steps:

1. Introduce multiplier variables $x_{s,a} \geq 0$ for every primal constraint associated with pair (s,a)
2. Define lagrangian $\mathcal{L}(f_s, x_{s,a})$
3. Define dual function $g(x_{s,a}) = \inf_{f_s} \mathcal{L}(f_s, x_{s,a})$ which lower bounds the primal optimum f^*.
4. Maximize the dual function $g(x_{s,a})$ to find a maximum lower bound to the primal problem, which will be our solution.

Primal $\min_{f \in \mathbb{R}^{|S|}} \sum_s \alpha(s) f_s$

$$\text{s.t. } f_s \geq R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) f_{s'} \quad \forall s,a$$

Step 1 Consider variables $x_{s,a} \geq 0 \quad \forall (s,a)$.

Define Lagrangian

$$\mathcal{L}(f_s, x_{s,a}) = \sum_s \alpha(s) f_s - \sum_{s,a} x_{s,a} \left[f_s - \left(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) f_{s'} \right) \right]$$

$$= \sum_s \alpha(s) f_s - \sum_{s,a} x_{s,a} f_s + \sum_{s,a} x_{s,a} R(s,a) + \gamma \sum_{s,a} x_{s,a} \sum_{s' \in S} T(s'|s,a) f_{s'}$$

$$= \gamma \sum_{s' \in S} f_{s'} \sum_{s,a} x_{s,a} T(s'|s,a) = \gamma \sum_{s \in S} f_s \sum_{s',a} x_{s',a} T(s|s',a)$$

$$= \sum_{s,a} x_{s,a} R(s,a) + \sum_s f_s \left[\alpha(s) - \sum_a x_{s,a} + \gamma \sum_{s',a} x_{s',a} T(s|s',a) \right]$$

$$\mathcal{L} = \sum_{s,a} x_{s,a} R(s,a) + \sum_s f_s \left[\alpha(s) - \sum_a x_{s,a} + \gamma \sum_{s',a} x_{s',a} T(s|s',a) \right]$$

Step 3 Define the dual function $g(x_{s,a}) = \inf_{f_s} \mathcal{L}(f_s, x_{s,a})$

- Note that without constraints, $\inf \mathcal{L} = -\infty$
- To avoid this, and get a useful dual function

dual constraints: $\alpha(s) - \sum_a x_{s,a} + \gamma \sum_{s',a} x_{s',a} T(s|s',a) = 0$

$\Rightarrow$ dual function $g(x_{s,a}) = \sum_{s,a} x_{sa} R(s,a)$

Step 4

$$\max \sum_{s,a} x_{s,a} R(s,a)$$

$$\text{s.t. } \sum_a x_{s,a} = \alpha(s) + \gamma \sum_{s',a} x_{s',a} T(s|s',a)$$

$$x_{s,a} \geq 0$$

Dual LP.

Dual LP formulation

occupancy measure

Interpretation: $x_{s,a}$ represents the total discounted ~~joint probability~~ under initial-state distribution $\alpha(s)$ that the system occupies state s and chooses action a.

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

Maximize discounted sum of total rewards.

$$\text{s.t.} \quad \sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

Dual LP formulation

Interpretation: $x_{s,a}$ represents the total discounted joint probability under initial-state distribution $\alpha(s)$ that the system occupies state s and chooses action a.

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

Maximize discounted sum of total rewards.

How much can I in state s'?

$$\text{s.t.} \quad \sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

Constraints enforce MDP dynamics: LHS is related to occupancy of s', RHS is the probability of starting in state s' summed with all probabilities of transitioning from (s,a) to s' for all $s \in S, a \in A_s$.

Dual LP formulation

Interpretation: $x_{s,a}$ represents the total discounted joint probability under initial-state distribution $\alpha(s)$ that the system occupies state s and chooses action a.

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

Maximize discounted sum of total rewards.

$$\text{s.t.} \quad \sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

Constraints enforce MDP dynamics: LHS is related to occupancy of s', RHS is the probability of starting in state s' summed with all probabilities of transitioning from (s,a) to s' for all $s \in S, a \in A_s$.

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

Occupancy variables must be positive.

Connecting $x_{s,a}$ with $\pi(a \mid s)$

Occupancy measure

Interpretation: $x_{s,a}$ represents the total discounted joint probability under initial-state distribution $\alpha(s)$ that the system occupies state s and chooses action a.

Connecting $x_{s,a}$ with $\pi(a \mid s)$

Interpretation: $x_{s,a}$ represents the total discounted joint probability under initial-state distribution $\alpha(s)$ that the system occupies state s and chooses action a.

Theorem 6.9.1 (Puterman):

A. For each $\pi \in \Pi^{MR}$, $s \in S$, and $a \in A_s$ define $x^{\pi}_{s,a}$ by

$$x^{\pi}_{s,a} = \sum_{j \in S} \alpha(j) \sum_{t=1}^{\infty} \gamma^t P^{\pi}(S_t = s, A_t = a \mid S_0 = j)$$

Then $x^{\pi}_{s,a}$ is a feasible solution to the dual problem.

Connecting $x_{s,a}$ with $\pi(a \mid s)$

occupancy measure.

Interpretation: $x_{s,a}$ represents the total discounted ~~joint probability~~ under initial-state distribution $\alpha(s)$ that the system occupies state s and chooses action a.

Theorem 6.9.1 (Puterman):

A. For each $\pi \in \Pi^{MR}$, $s \in S$, and $a \in A_s$ define $x^{\pi}_{s,a}$ by

$$x^{\pi}_{s,a} = \sum_{j \in S} \alpha(j) \sum_{t=1}^{\infty} \gamma^t P^{\pi}(S_t = s, A_t = a \mid S_0 = j)$$

Then $x^{\pi}_{s,a}$ is a feasible solution to the dual problem.

B. Suppose $x_{s,a}$ is a feasible solution to the dual problem, then, for each $s \in S$, $\Sigma_{a \in A_s} x_{s,a} > 0$. Define the randomized stationary policy $\pi_x(a \mid s)$ by:

$$\pi_x(a \mid s) = \frac{x_{s,a}}{\sum_{a' \in A_s} x_{s,a'}}$$

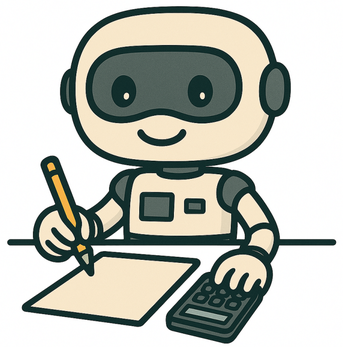
Then $x^{\pi_x}_{s,a}$ as defined in the equation above is a feasible solution to the dual LP, and $x^{\pi_x}_{s,a} = x_{s,a}$ for all $a \in A_s$ and $s \in S$.

In words, what does this theorem tell us?

- Direct correspondance between $x_{s,a}$ and policies
- From any feasible soln to the dual LP, we can extract a policy.
- Given a policy, π, there is a corresponding x_{sa}
  - related to the discounted occupancy measure of pairs (s,a) under policy π.

Under the given interpretation of $x_{s,a}$, what does the dual objective $\sum_{s \in S} \sum_{a \in A_s} R(s,a) x_{s,a}$ represent?

$$\sum_s \sum_a x_{s,a} R(s,a) = \sum_s \alpha(s) V^{\pi_x}(s)$$



How can we use the results of this theorem to find policies?

First, solve the dual LP. Then...

$$\pi(a|s) = \frac{x_{s,a}}{\sum_{a' \in A} x_{s,a'}}$$

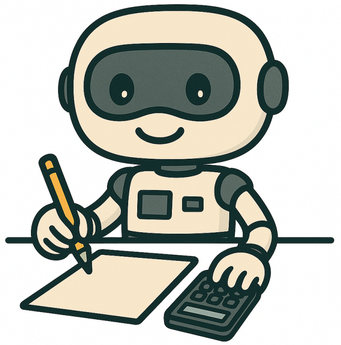
What kinds of additional constraints can we model with the dual problem?

- Constraints on discounted occupancy for particular state-action pairs

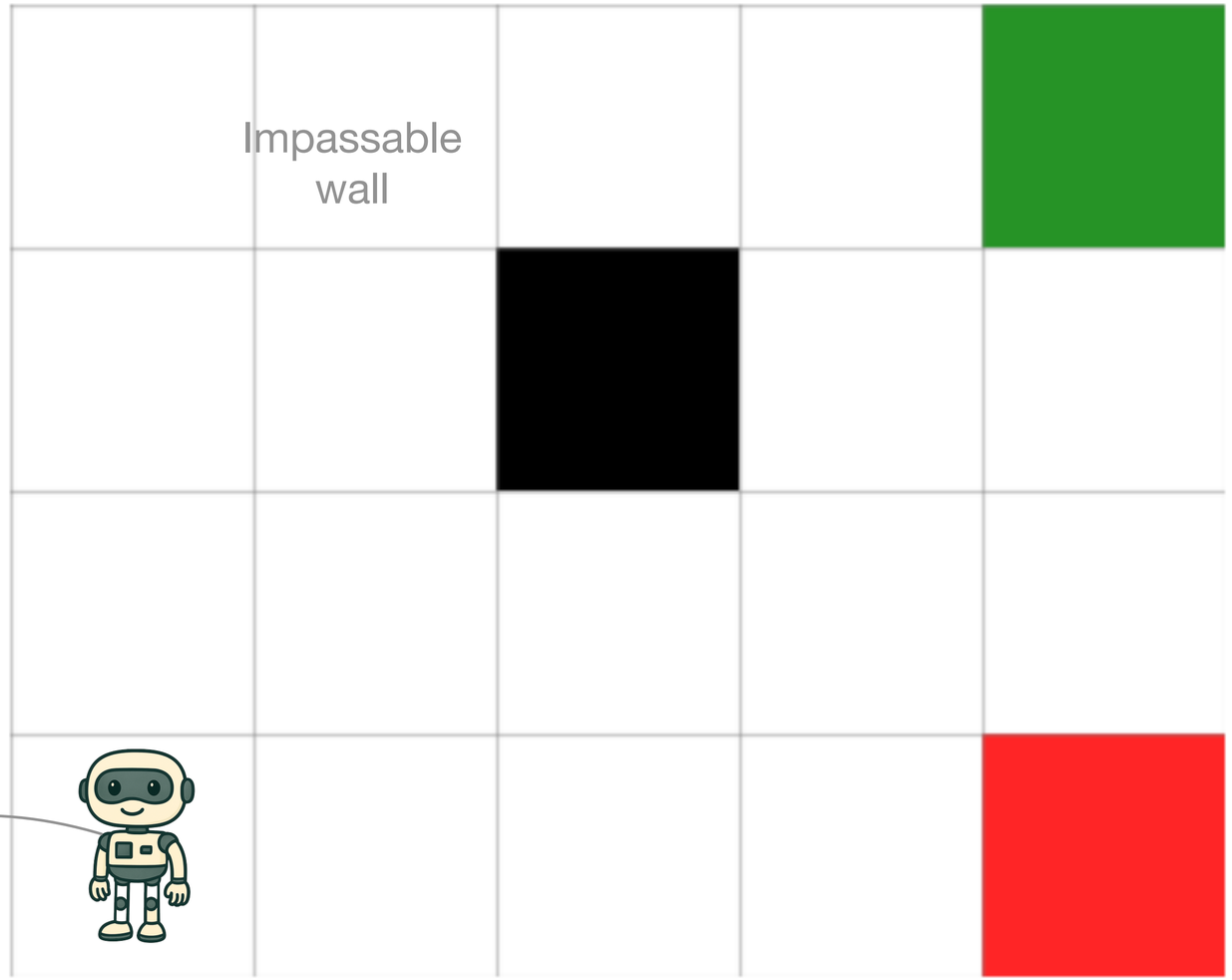
What if we wanted to solve undiscounted finite horizon problems? Can we still use the dual LP?

- Yes! Add time to each variable x_{sat}. Need to now include an extra constraint for every instant of the time horizon.

- If $\gamma = 1$, x_{sat} is a probability. Prob of taking action a from state s at time t, under the associated policy.



Worked example



LP Solution

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

$$\sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

Solve using scipy.optimize.linprog()

LP Solution

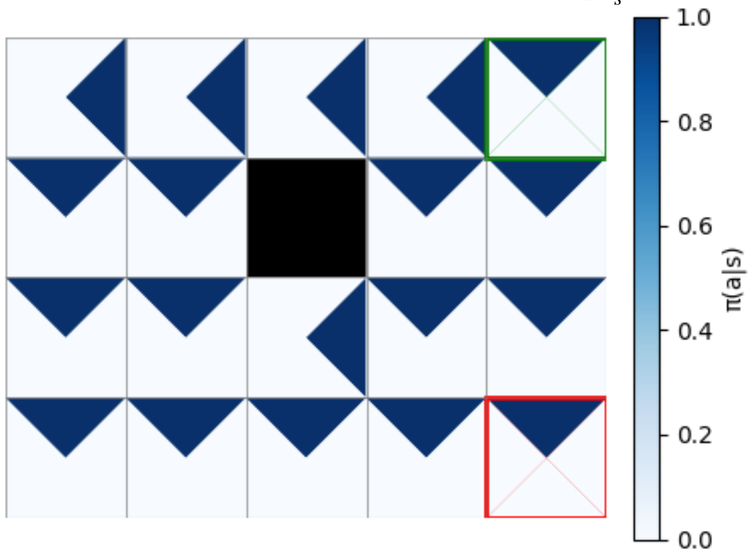
$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

$$\sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

Solve using scipy.optimize.linprog()

Optimal policy $\pi_x(a \mid s) = \dfrac{x_{s,a}}{\sum_{a' \in A_s} x_{s,a'}}$



LP Solution

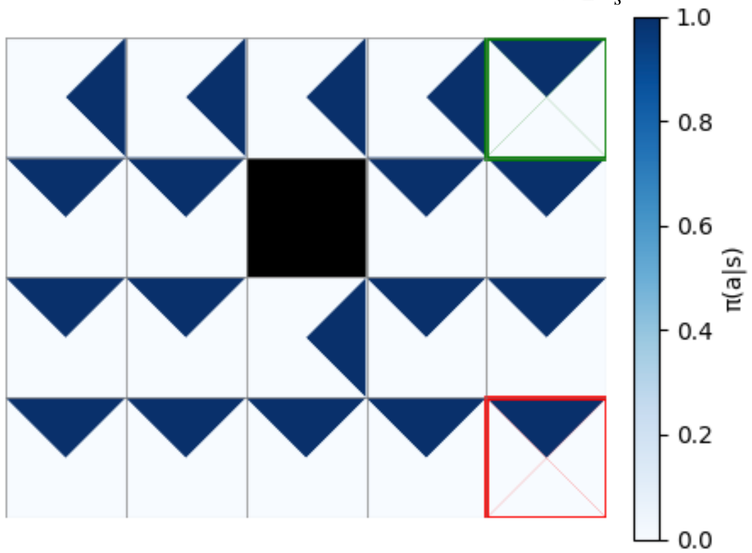
$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

$$\sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

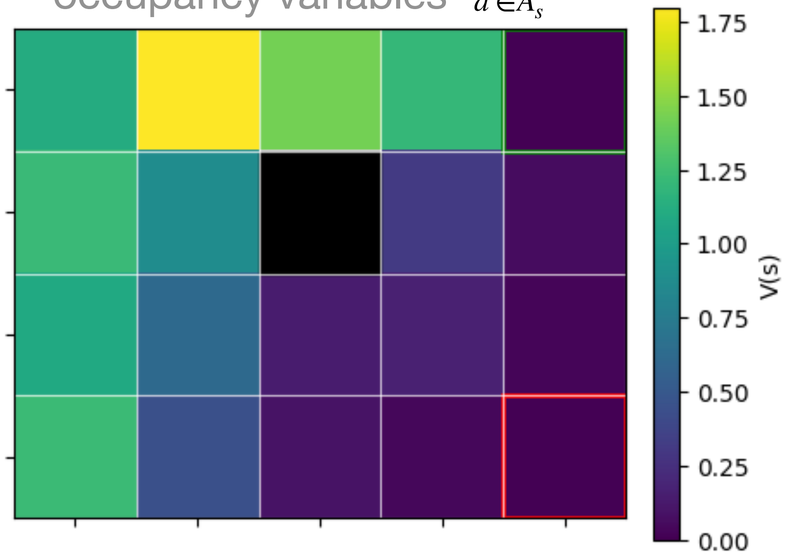
$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

Solve using scipy.optimize.linprog()

Optimal policy $\pi_x(a \mid s) = \frac{x_{s,a}}{\sum_{a' \in A_s} x_{s,a'}}$



Discounted state-occupancy variables $\sum_{a' \in A_s} x_{s,a'}$



LP Solution with constraints

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

$$\sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \leq 0.1, \quad \forall a \in A_s, s \in S_{constraint}$$

Add constraint enforcing low occupancy in a state a long the agent's previously planned path.

LP Solution with constraints

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

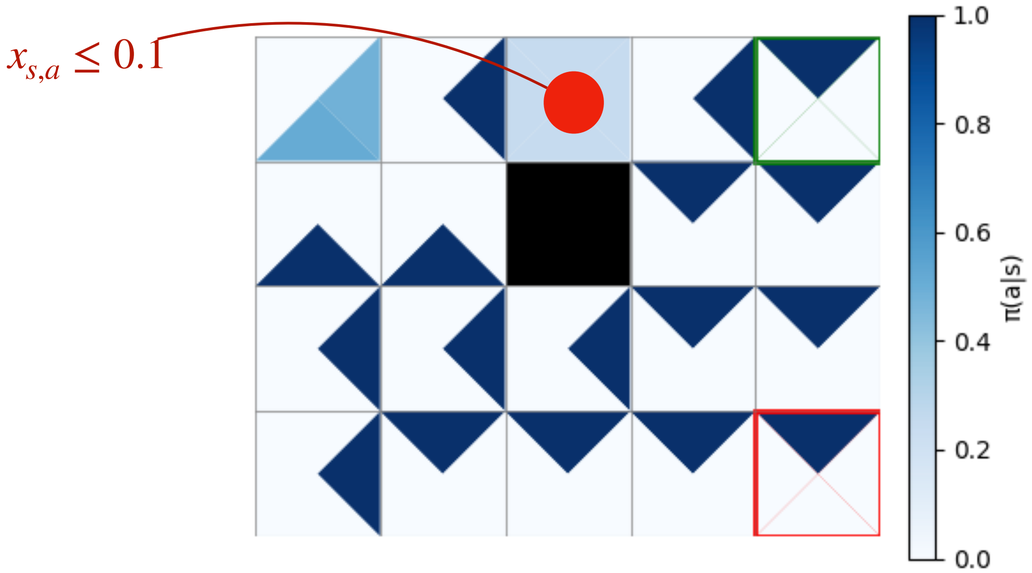
$$\sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \leq 0.1, \quad \forall a \in A_s, s \in S_{constraint}$$

Add constraint enforcing low occupancy in a state a long the agent's previously planned path.

Optimal policy $\pi_x(a \mid s) = \dfrac{x_{s,a}}{\sum_{a' \in A_s} x_{s,a'}}$



LP Solution with constraints

$$\max_{x_{s,a} \in \mathbb{R}^{|S| \cdot |A|}} \sum_{s \in S, a \in A} R(s,a) x_{s,a}$$

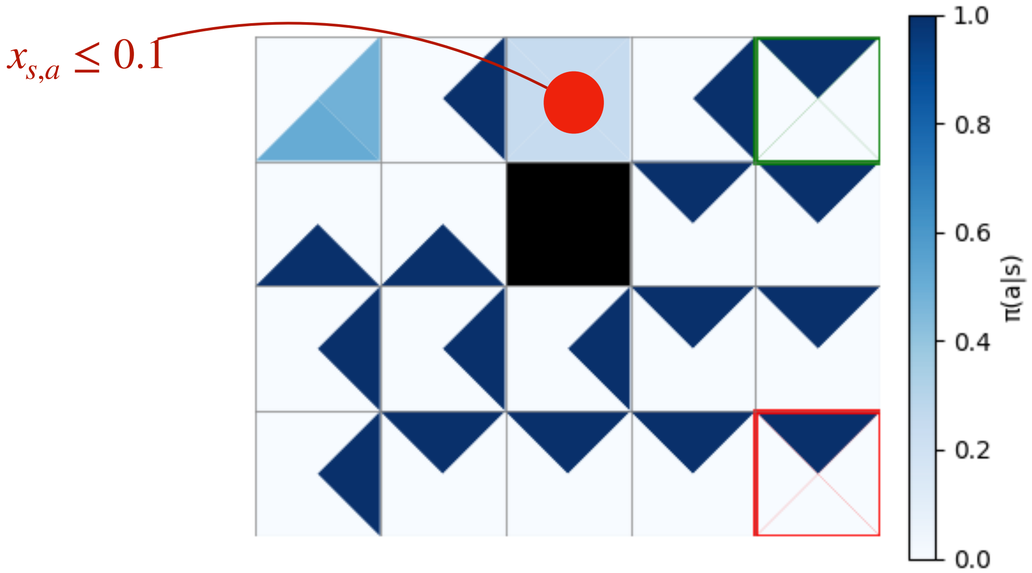
$$\sum_{a \in A_{s'}} x_{s',a} = \alpha(s') + \gamma \sum_{s \in S, a \in A_s} T(s' \mid s, a) x_{s,a} \quad \forall a \in A_s, s \in S$$

$$x_{s,a} \geq 0, \quad \forall a \in A_s, s \in S$$

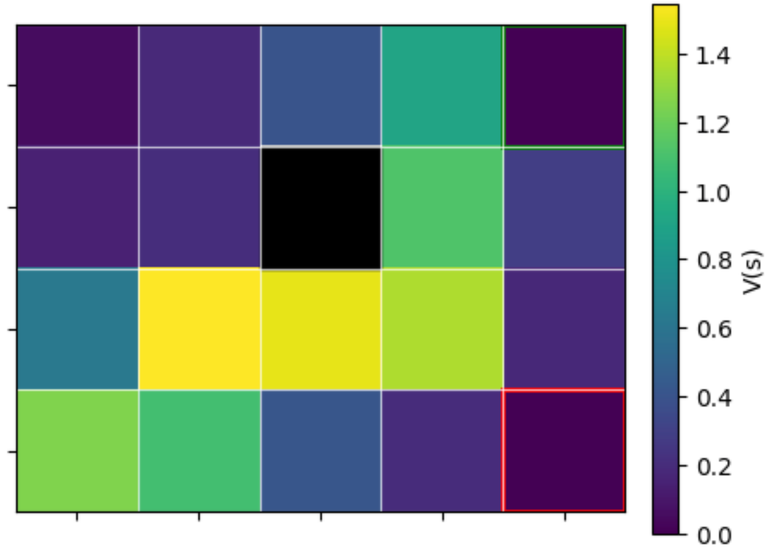
$$x_{s,a} \leq 0.1, \quad \forall a \in A_s, s \in S_{constraint}$$

Add constraint enforcing low occupancy in a state a long the agent's previously planned path.

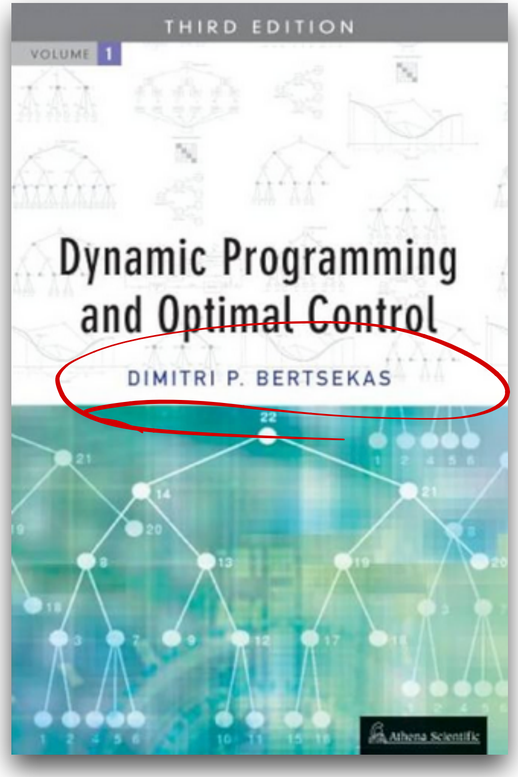
Optimal policy $\pi_x(a \mid s) = \dfrac{x_{s,a}}{\sum_{a' \in A_s} x_{s,a'}}$



Discounted state-occupancy variables $\sum_{a' \in A_s} x_{s,a'}$



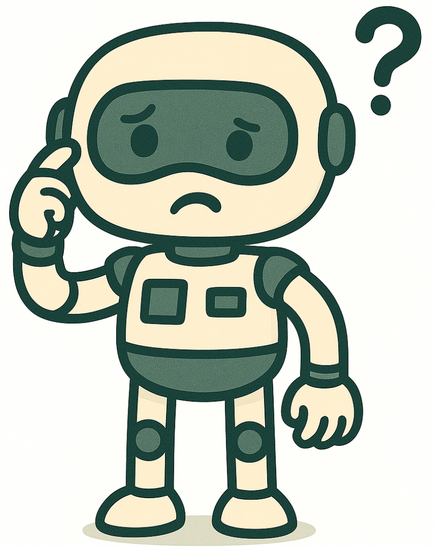
Dynamic programming in continuous control: Hamilton-Jacobi-Bellman Equations & LQR



Today: Chapter 3

So far we've seen Bellman's equations for discrete-time systems with finite states and actions.

How ~~are~~ do similar dynamic programming ideas show up in continuous-time optimal control problems with compact state and action sets?



Problem setting

Consider a continuous-time dynamic system

State vector $\in \mathbb{R}^n$

Control vector $\in \mathbb{R}^m$

Terminal time

$$\dot{x}(t) = f(x(t), u(t)), \quad 0 \leq t \leq T$$

$$x(0) = x_0$$

Initial condition

Problem setting

Consider a continuous-time dynamic system

State vector $\in \mathbb{R}^n$

Control vector $\in \mathbb{R}^m$

Terminal time

$$\dot{x}(t) = f(x(t), u(t)), \quad 0 \leq t \leq T$$

$$x(0) = x_0$$

Initial condition

Objective is to minimize a cost function of the form

$$h(x(T)) + \int_0^T g(x(t), u(t))dt$$

Terminal cost

Running cost

Example

A mass moves on a line under influence of a force u.
Let $x_1(t)$ and $x_2(t)$ are position and velocity of the mass, respectively.

Initial condition $x_0 = \begin{bmatrix} x_{1,0} \\ x_{2,0} \end{bmatrix}$ ← initial position, ← initial velocity

Objective Bring the mass to $x_F = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. ~~constraint $\|u(t)\| \leq 1$~~

Dynamics $\dot{x}(t) = f(x, u)$, $m = 1$

$$\dot{x}(t) = \begin{bmatrix} x_2(t) \\ u(t) \end{bmatrix} \rightsquigarrow f(x, u)$$

Cost function: $h(x(T)) = |x_1(T) - x_{F,1}|^2 + |x_2(T) - x_{F,2}|^2$

$g(x(t), u(t)) = 0 \quad \forall t \in [0, T]$

This is a very simple example of an optimal control problem.

The Hamilton-Jacobi-Bellman PDE

Optimal “cost-to-go” function

$$\nabla_t V^*(t,x) = -\min_{u \in U} \left[g(x,u) + \nabla_x V^*(t,x) \cdot f(x,u) \right]$$

The Hamilton-Jacobi-Bellman PDE

Optimal “cost-to-go” function

$$\nabla_t V^*(t,x) = -\min_{u \in U} \left[g(x,u) + \nabla_x V^*(t,x) \cdot f(x,u) \right]$$

Change in V^* at state x
over infinitesimal dt

The Hamilton-Jacobi-Bellman PDE

Optimal “cost-to-go” function

$$\nabla_t V^*(t,x) = -\min_{u \in U} \left[g(x,u) + \nabla_x V^*(t,x) \cdot f(x,u) \right]$$

Change in V^* at state x over infinitesimal dt

Minimization problem over infinitesimal dt

The Hamilton-Jacobi-Bellman PDE

Optimal “cost-to-go” function

$$\nabla_t V^*(t,x) = -\min_{u \in U} \left[g(x,u) + \nabla_x V^*(t,x) \cdot f(x,u) \right]$$

Change in V^* at state x over infinitesimal dt

Minimization problem over infinitesimal dt

Optimal value at the resulting infinitesimally changed state x

How does this relate to the Bellman equations from last class?
Why does the continuous setting yield a differential equation?

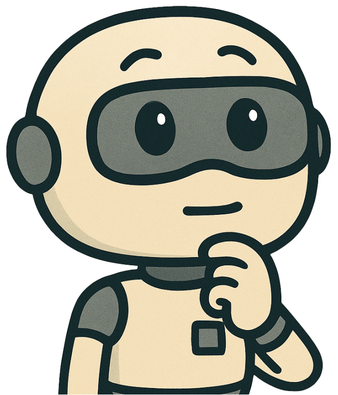
$$\nabla_t V^*(t,x) = -\min_{u \in U} \left[g(x,u) + \nabla_x V^*(t,x) \cdot f(x,u) \right]$$

vs.

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^*(s') \right]$$

- In discrete settings, Bellman equations tell us recurrent relations over states with discrete jumps in state (which we could model with discrete Δt)

- HJB equations tell the same thing as $\Delta t \to 0$.



Deriving the Hamilton-Jacobi-Bellman PDE

Step 1: Discretize the system in time.

Original system $\dot{x}(t) = f(x(t), u(t)), \quad 0 \leq t \leq T \qquad x(0) = x_0$

Cost function $h(x(T)) + \int_0^T g(x(t), u(t))dt$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 1: Discretize the system in time.

Original system $\dot{x}(t) = f(x(t), u(t)), \quad 0 \le t \le T \qquad x(0) = x_0$

Cost function $h(x(T)) + \int_0^T g(x(t), u(t))dt$

Define $\delta = \frac{T}{N}$ $\quad x_k = x(k\delta), \quad k = 0,1,2,\ldots,N$ $\quad u_k = u(k\delta), \quad k = 0,1,2,\ldots,N$

Δt ↗

Deriving the Hamilton-Jacobi-Bellman PDE

Step 1: Discretize the system in time.

Original system $\dot{x}(t) = f(x(t), u(t)), \quad 0 \le t \le T \qquad x(0) = x_0$

Cost function $h(x(T)) + \int_0^T g(x(t), u(t))dt$

Define $\delta = \frac{T}{N}$ $\qquad x_k = x(k\delta), \quad k = 0,1,2,\ldots,N$

$u_k = u(k\delta), \quad k = 0,1,2,\ldots,N$

Discrete system $x_{k+1} = x_k + f(x_k, u_k) \cdot \delta, \quad k = 0,1,2,\ldots N-1$

Discrete cost $h(x_N) + \sum_{k=1}^{N-1} g(x_k, u_k) \cdot \delta$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 2: Apply Bellman optimality equations to discrete system.

$$\tilde{V}^*(N\delta, x) = h(x)$$ ——— Terminal cost

Deriving the Hamilton-Jacobi-Bellman PDE

Step 2: Apply Bellman optimality equations to discrete system.

initial condition

$$\tilde{V}^*(N\delta, x) = h(x)$$ — Terminal cost

$$\tilde{V}^*(k\delta, x) = \min_{u \in U} \left[g(x,u) \cdot \delta + \tilde{V}^*(k\delta + \delta, x + f(x,u) \cdot \delta) \right], \quad k = 0, \ldots, N-1$$

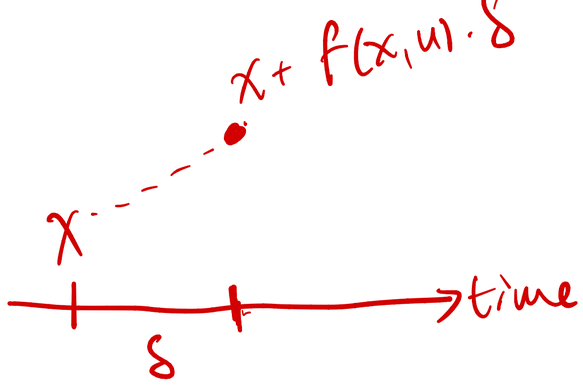
Discrete optimality equations

Discrete cost-to-go function

next time

next state, given dynamics

Note! No expectations, because there is no stochasticity in these equations.



Deriving the Hamilton-Jacobi-Bellman PDE

Step 3: Taylor expand $\tilde{V}^*$ about the point $(k\delta, x)$ in time,

(assuming required differentiability properties on $\tilde{V}^*$)

next δ instant in time

$$\tilde{V}^*(k\delta + \delta, x + f(x,u)\cdot\delta) = \tilde{V}^*(k\delta, x) + \nabla_t \tilde{V}^*(k\delta, x)\cdot\delta + \nabla_x \tilde{V}^*(k\delta, x)\cdot f(x,u)\cdot\delta + o(\delta)$$

Where $o(\delta)$ represents second-order terms satisfying $lim_{\delta\to 0}\frac{o(\delta)}{\delta} = 0$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 4: Plug the Taylor expansion into discrete Bellman equation from before.

$$\tilde{V}^*(k\delta, x) = \min_{u \in U} \left[g(x,u) \cdot \delta + \tilde{V}^*(k\delta + \delta, x + f(x,u) \cdot \delta) \right], \quad k = 0, \ldots, N-1$$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 4: Plug the Taylor expansion into discrete Bellman equation from before.

$$\tilde{V}^*(k\delta, x) = \min_{u \in U} \left[g(x,u) \cdot \delta + \tilde{V}^*(k\delta + \delta, x + f(x,u) \cdot \delta) \right], \quad k = 0, \ldots, N-1$$

$$\tilde{V}^*(k\delta, x) = \min_{u \in U} [g(x,u) \cdot \delta + \tilde{V}^*(k\delta, x) + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta$$
$$\nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta)], \quad k = 0, \ldots, N-1$$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 5: Subtract $\tilde{V}^*(k\delta, x)$ from both sides, and take the limit as $\delta \to 0$.

$$\tilde{V}^*(k\delta, x) = \min_{u \in U}[g(x,u) \cdot \delta + \tilde{V}^*(k\delta, x) + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta$$

$$\nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta)], \quad k = 0, \ldots, N-1$$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 5: Subtract $\tilde{V}^*(k\delta, x)$ from both sides, and take the limit as $\delta \to 0$.

$$\tilde{V}^*(k\delta, x) = \min_{u \in U}[g(x,u) \cdot \delta + \tilde{V}^*(k\delta, x) + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta$$

$$\nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta)], \quad k = 0, \ldots, N-1$$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 5: Subtract $\tilde{V}^*(k\delta, x)$ from both sides, and take the limit as $\delta \to 0$.

$$\tilde{V}^*(k\delta, x) = \min_{u \in U}[g(x,u) \cdot \delta + \tilde{V}^*(k\delta, x) + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta$$

$$\nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta)], \quad k = 0, \ldots, N-1$$

Assume $\lim_{\delta \to 0, k \to \infty, k\delta \to t} \tilde{V}^*(k\delta, x) = V^*(t, x), \quad$ for all t, x

Deriving the Hamilton-Jacobi-Bellman PDE

Step 5: Subtract $\tilde{V}^*(k\delta, x)$ from both sides, and take the limit as $\delta \to 0$.

$$\tilde{V}^*(k\delta, x) = \min_{u \in U}[g(x,u) \cdot \delta + \tilde{V}^*(k\delta, x) + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta$$

$$\nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta)], \quad k = 0, \ldots, N-1$$

Assume $\lim_{\delta \to 0, k \to \infty, k\delta \to t} \tilde{V}^*(k\delta, x) = V^*(t,x), \quad$ for all t, x

$$0 = \min_{u \in U}\left[g(x,u) \cdot \delta + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta + \nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta)\right], \quad k = 0, \ldots, N-1$$

$\tilde{V}^*(t,x)$ $\quad \tilde{V}^*(t,x)$ $\quad 0$

Deriving the Hamilton-Jacobi-Bellman PDE

Step 5: Subtract $\tilde{V}^*(k\delta, x)$ from both sides, and take the limit as $\delta \to 0$.

$$\tilde{V}^*(k\delta, x) = \min_{u \in U}[g(x,u) \cdot \delta + \tilde{V}^*(k\delta, x) + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta$$

$$\nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta)], \quad k = 0, \ldots, N-1$$

Assume $\lim_{\delta \to 0, k \to \infty, k\delta \to t} \tilde{V}^*(k\delta, x) = V^*(t,x), \quad$ for all t, x

$$0 = \min_{u \in U} \left[g(x,u) \cdot \delta + \nabla_t \tilde{V}^*(k\delta, x) \cdot \delta + \nabla_x \tilde{V}^*(k\delta, x) \cdot f(x,u) \cdot \delta + o(\delta) \right], \quad k = 0, \ldots, N-1$$

$\tilde{V}^*(t,x)$ $\qquad$ $\tilde{V}^*(t,x)$ $\qquad$ 0

Divide by δ and rearrange.

$$\Longrightarrow \boxed{\nabla_t V^*(t,x) = -\min_{u \in U} \left[g(x,u) + \nabla_x V^*(t,x) \cdot f(x,u) \right]}$$

How can we use the HJB PDE to solve control problems?

$$\nabla_t V^*(t,x) = -\min_{u \in U} \left[g(x,u) + \nabla_x V^*(t,x) \cdot f(x,u) \right]$$

Proposition 3.2.1: (Sufficiency Theorem) Suppose $V(t,x)$ is a solution to the HJB equation; that is, V is continuously differentiable in t and x, and is such that

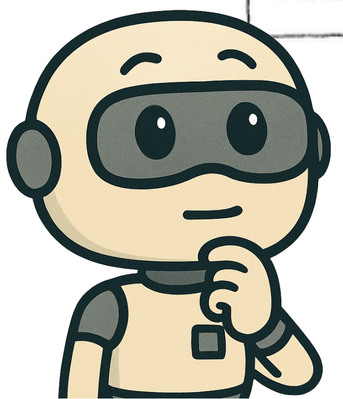
$$0 = \min_{u \in U} \left[g(x,u) + \nabla_t V(t,x) + \nabla_x V(t,x)' f(x,u) \right], \qquad \text{for all } t, x, \tag{3.2}$$

$$V(T,x) = h(x), \qquad \text{for all } x. \tag{3.3}$$

Suppose also that $\mu^*(t,x)$ attains the minimum in Eq. (3.2) for all t and x. Let $\{x^*(t) \mid t \in [0,T]\}$ be the state trajectory obtained from the given initial condition $x(0)$ when the control trajectory $u^*(t) = \mu^*\big(t, x^*(t)\big)$, $t \in [0,T]$ is used [that is, $x^*(0) = x(0)$ and for all $t \in [0,T]$, $\dot{x}^*(t) = f\big(x^*(t), \mu^*(t, x^*(t))\big)$; we assume that this differential equation has a unique solution starting at any pair (t,x) and that the control trajectory $\{\mu^*(t, x^*(t)) \mid t \in [0,T]\}$ is piecewise continuous as a function of t]. Then V is equal to the optimal cost-to-go function, i.e.,

$$V(t,x) = J^*(t,x), \qquad \text{for all } t, x.$$

Furthermore, the control trajectory $\{u^*(t) \mid t \in [0,T]\}$ is optimal.



In discrete setting: Value iteration to solve for V^*, and the we solved one-step minimization problems using that value function to get optimal policy.

Now Solve the HJB PDE to get $V(t,x)$. Plug it into (3.2) and solve minimization problem for u at each t, to get optimal control policy.

Example: Linear-Quadratic Problems

Consider the n-dimensional linear system

$$\dot{x}(t) = Ax(t) + Bu(t) \qquad A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}$$

and the quadratic cost function

$$x(T)'Q_T x(T) + \int_0^T \left(x(t)'Qx(t) + u(t)'Ru(t)\right) dt$$

$Q_T, Q \in \mathbb{R}^{n\times n}$ Symmetric positive semidefinite

$R \in \mathbb{R}^{m\times m}$ Symmetric positive definite

The HJB equations become

$$V(T, x) = x'Q_T x$$

$$0 = \min_{u\in\mathbb{R}^{m\times m}} \left[x'Qx + u'Ru + \nabla_t V(t,x) + \nabla_x V(t,x)'(Ax + Bu)\right]$$

Example: Linear-Quadratic Problems

Try a solution of the form

$$V(t,x) = x'K(t)x, \quad \text{for some symmetric } K(t) \in \mathbb{R}^{n\times n}$$

Example: Linear-Quadratic Problems

Try a solution of the form

$$V(t,x) = x'K(t)x, \quad \text{for some symmetric } K(t) \in \mathbb{R}^{n\times n}$$

Then we have $\nabla_x V(t,x) = 2K(t)x$ and $\nabla_t V(t,x) = x'\dot{K}(t)x$

Example: Linear-Quadratic Problems

Try a solution of the form

$$V(t,x) = x'K(t)x, \quad \text{for some symmetric } K(t) \in \mathbb{R}^{n\times n}$$

Then we have $\nabla_x V(t,x) = 2K(t)x$ and $\nabla_t V(t,x) = x'\dot{K}(t)x$

Substituting into our HJB equations we get

$$V(T,x) = x'Q_T x$$

$$0 = \min_{u\in\mathbb{R}^{m\times m}} \left[x'Qx + u'Ru + x'\dot{K}(t)x + 2x'K(t)Ax + 2x'K(t)Bu\right]$$

Example: Linear-Quadratic Problems

$$V(T, x) = x' Q_T x$$

$$0 = \min_{u \in \mathbb{R}^{m \times m}} \left[x'Qx + u'Ru + x'\dot{K}(t)x + 2x'K(t)Ax + 2x'K(t)Bu \right]$$

Solve the minimization in u by setting the derivative to zero.

Example: Linear-Quadratic Problems

$$V(T, x) = x' Q_T x$$

$$0 = \min_{u \in \mathbb{R}^{m \times m}} \left[x' Q x + u' R u + x' \dot{K}(t) x + 2 x' K(t) A x + 2 x' K(t) B u \right]$$

Solve the minimization in u by setting the derivative to zero.

$$\nabla_u \left[x' Q x + u' R u + x' \dot{K}(t) x + 2 x' K(t) A x + 2 x' K(t) B u \right] = 0$$

$$\implies 2 B' K(t) x + 2 R u = 0$$

$$\implies \boxed{u = -R^{-1} B' K(t) x}$$

Example: Linear-Quadratic Problems

Substitute $u = -R^{-1}B'K(t)x$ into

$$0 = \min_{u \in \mathbb{R}^{m \times m}} \left[x'Qx + u'Ru + x'\dot{K}(t)x + 2x'K(t)Ax + 2x'K(t)Bu \right]$$

Example: Linear-Quadratic Problems

Substitute $u = -R^{-1}B'K(t)x$ into

$$0 = \min_{u \in \mathbb{R}^{m \times m}} \left[x'Qx + u'Ru + x'\dot{K}(t)x + 2x'K(t)Ax + 2x'K(t)Bu \right]$$

We get

$$0 = x' \left(\dot{K}(t) + K(t)A + A'K(t) - K(t)BR^{-1}B'K(t) + Q \right) x, \quad \forall (t, x)$$

Example: Linear-Quadratic Problems

Substitute $u = -R^{-1}B'K(t)x$ into

$$0 = \min_{u \in \mathbb{R}^{m \times m}} \left[x'Qx + u'Ru + x'\dot{K}(t)x + 2x'K(t)Ax + 2x'K(t)Bu \right]$$

We get

$$0 = x' \left(\dot{K}(t) + K(t)A + A'K(t) - K(t)BR^{-1}B'K(t) + Q \right) x, \quad \forall (t, x)$$

So, for $V(t,x) = x'K(t)x$ to solve the HJB equation, $K(t)$ must satisfy the Continuous-Time Riccati Equation:

$$\boxed{\dot{K}(t) = -K(t)A - A'K(t) + K(t)BR^{-1}B'K(t) - Q}$$

With terminal condition $\boxed{K(T) = Q_T}$

Example: Linear-Quadratic Problems

To get the optimal controller, start at the terminal condition $K(T) = Q_T$

Solve the Riccati equation backwards in time to get $K(t)$.

Plug $K(t)$ into $u^* = -R^{-1}B'K(t)x$ to get the optimal controller.

What about if we had solved the time-discretized version of the linear quadratic problem?

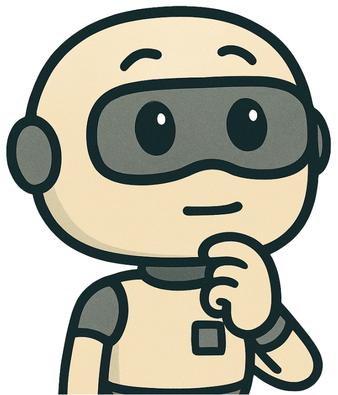
- we get discrete time Riccati equations

$$K_T = Q_T$$

$$K_{t-1} = Q + A' K_t A - A' K_t B \left(B' K_t B + R\right)^{-1} B' K_t A$$

↳ start at end, and recursively solve this system of equation.

- In continuous time, solve CTRE backwards in time. get our controller interms of that solution.



A numerical example

Double integrator system: $x = \begin{bmatrix} p \\ v \end{bmatrix}$ Position / velocity

$\dot{p} = v$ Time derivative of position IS velocity

$\dot{v} = u$ Acceleration is given by control input u

In matrix form:

$$\underbrace{\begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} p \\ v \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B} u$$

Set initial condition $x = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, and cost components $Q = Q_T = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$, $R = [1.0]$

$h(x(T)) = 10(x_1 - 0)^2 + 1(x_2 - 0)^2$

A numerical example

```python
def double_integrator_matrices():
    A = np.array([[0.0, 1.0],
                  [0.0, 0.0]])
    B = np.array([[0.0],
                  [1.0]])
    return A, B


def riccati_rhs(A, B, Q, R, P):
    """
    Compute RHS of the *backward* Riccati ODE:
        -Pdot = A^T P + P A - P B R^{-1} B^T P + Q
    Return Pdot (for forward time), so the ODE is:
        dP/dt = -(A^T P + P A - P B R^{-1} B^T P + Q)
    """
    Rinv = np.linalg.inv(R)
    PB = P @ B
    BRB = PB @ (Rinv @ (B.T @ P))
    return -(A.T @ P + P @ A - BRB + Q)
```

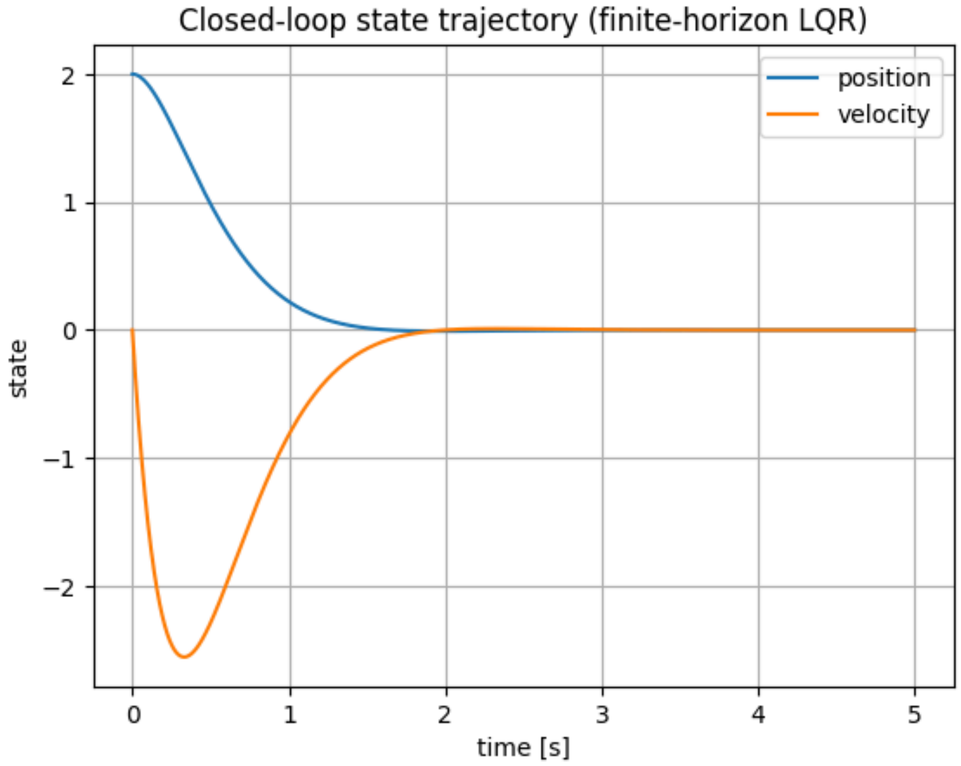
```python
def riccati_finite_horizon(A, B, Q, R, S, T, num_points=1000):
    """
    Integrate the matrix RDE backward from t=T to t=0.
    Returns:
      t_grid (ascending),
      P_seq: (num_points, n, n),
      K_seq: (num_points, m, n)
    """
    n = A.shape[0]
    # Flatten P to integrate as a vector
    def f(t, Pflat):
        P = Pflat.reshape(n, n)
        # enforce symmetry numerically
        P = 0.5 * (P + P.T)
        dP = riccati_rhs(A, B, Q, R, P)
        # keep symmetry in derivative as well
        dP = 0.5 * (dP + dP.T)
        return dP.ravel()

    # integrate from T down to 0 (solve_ivp supports decreasing t_span)
    t_span = (T, 0.0)
    t_eval_desc = np.linspace(T, 0.0, num_points)
    sol = solve_ivp(
        f, t_span, S.ravel(), t_eval=t_eval_desc, method="RK45", rtol=1e-8, atol=1e-10
    )
    if not sol.success:
        raise RuntimeError(f"Riccati integration failed: {sol.message}")

    # reverse to ascending time for convenience
    t_grid = sol.t[::-1]
    P_flat_seq = sol.y[:, ::-1].T  # shape (num_points, n*n)
    P_seq = P_flat_seq.reshape(-1, n, n)
    # clean symmetry
    P_seq = 0.5 * (P_seq + np.transpose(P_seq, (0, 2, 1)))

    # K(t) = R^{-1} B^T P(t)
    Rinv = np.linalg.inv(R)
    K_seq = np.array([Rinv @ (B.T @ P) for P in P_seq])  # (N, m, n)

    return t_grid, P_seq, K_seq
```

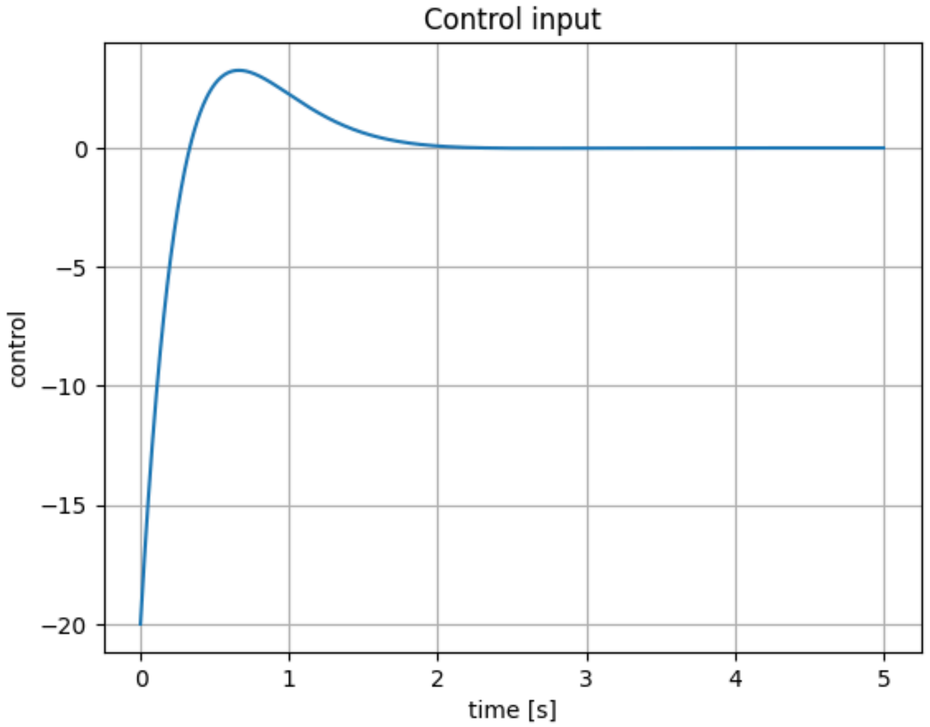
A numerical example

```python
def simulate_closed_loop(A, B, K_t, t_grid, x0):
    """
    Simulate xdot = (A - B K(t)) x, u = -K(t) x using 4th-order Runge-Kutta.
    K_t: array of shape (N, m, n) corresponding to t_grid (ascending)
    """
    n = A.shape[0]
    m = B.shape[1]
    # interpolate each entry of K over time
    K_interp = interp1d(t_grid, K_t, axis=0, kind="linear", fill_value="extrapolate")

    def f(t, x):
        K = K_interp(t)  # (m, n)
        u = -K @ x
        return (A @ x + B @ u).ravel()

    x_traj = [x0.copy()]
    u_traj = []
    for k in range(len(t_grid) - 1):
        t, h = t_grid[k], t_grid[k + 1] - t_grid[k]
        x = x_traj[-1]

        # RK4
        k1 = f(t, x)
        k2 = f(t + 0.5 * h, x + 0.5 * h * k1)
        k3 = f(t + 0.5 * h, x + 0.5 * h * k2)
        k4 = f(t + h, x + h * k3)
        x_next = x + (h / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)

        # store
        K_now = K_interp(t)
        u_now = -(K_now @ x)
        u_traj.append(u_now)
        x_traj.append(x_next)

    x_traj = np.vstack(x_traj)           # (N, n)
    u_traj = np.vstack(u_traj + [u_traj[-1]])  # pad to length N
    return x_traj, u_traj
```

A numerical example

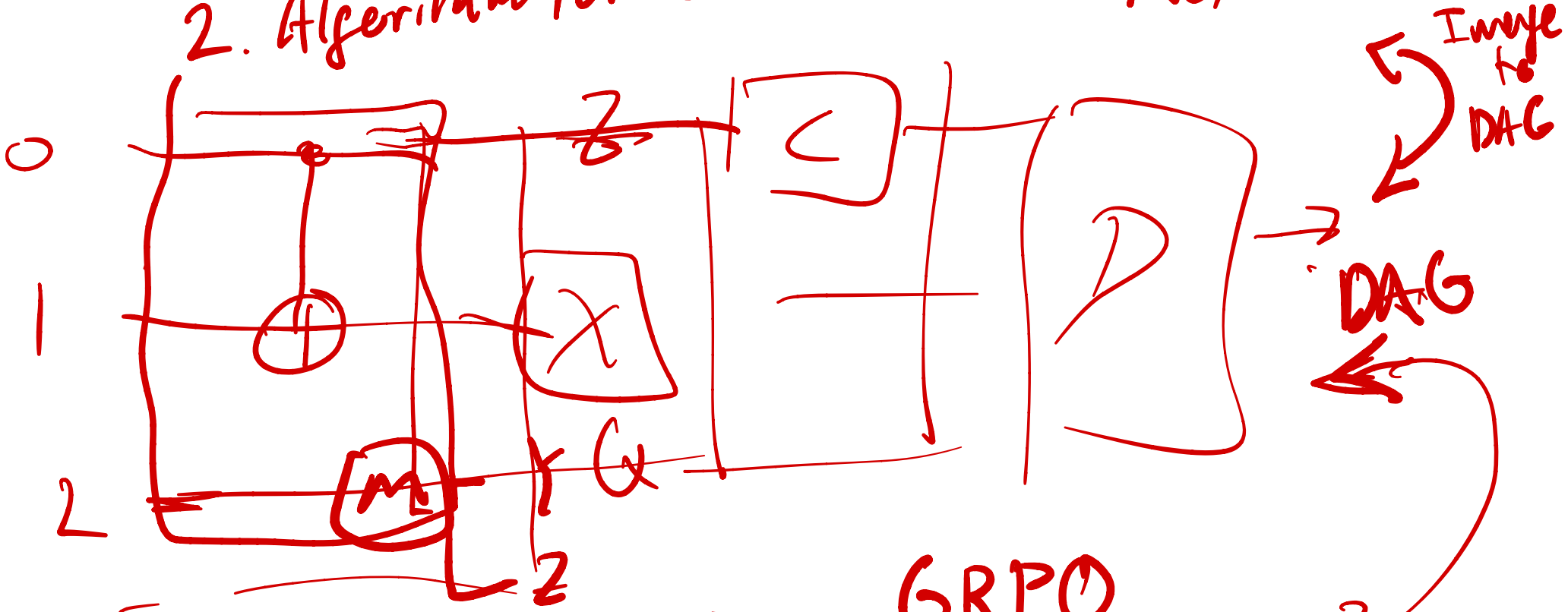
```python
# Problem data
A, B = double_integrator_matrices()
# Weights (tune as desired)
Q = np.diag([10.0, 1.0])   # penalize position more than velocity
R = np.array([[0.1]])      # control effort penalty
S = Q.copy()               # terminal cost
T = 5.0                    # horizon (seconds)

# Solve finite-horizon RDE
t, P_seq, K_seq = riccati_finite_horizon(A, B, Q, R, S, T, num_points=1501)

# Example: simulate closed-loop from some initial state
x0 = np.array([2.0, 0.0])  # start at position 2m, zero velocity
x_traj, u_traj = simulate_closed_loop(A, B, K_seq, t, x0)
```





1. Computer vision ← Leave out for now

2. Algorithm for conversion ← RL/SDM



GRPO

```
def fc1:
    cNot(0,1)
    x(1)
    c[0];D
```

```
draw(fc1)
```

{cNot, S, x, T}

$$T > t \Rightarrow T^4 > t^4 \Rightarrow \frac{1}{t^4} > \frac{1}{T^4}$$

$$P(\bar{F}_{a,t}) \leq \frac{2}{t^4}$$

$$1 - \sum_{a \in A} \sum_{t \in [T]} \frac{2}{t^4} = 1 - k \sum_{t=1}^{T} \frac{2}{t^4}$$

$$\sum_{t=1}^{T} \frac{2}{t^4} > \sum_{t=1}^{T} \frac{2}{T^4}$$

$$-k \sum_{t=1}^{T} \frac{2}{t^4} < -k \sum_{t=1}^{T} \frac{2}{T^4}$$