Linear Quadratic Regulator in Three Ways

Indirect Shooting, Quadratic Programming, and Riccati Equation

Zirui Zhang

Cheng Kar-Shun Robotics Institute
The Hong Kong University of Science and Technology

September 21, 2025

Problem Formulation

Consider a discrete-time linear system:

$$x_{n+1} = A_n x_n + B_n u_n$$

Quadratic cost function:

$$\min_{x_{1:N}, u_{1:N-1}} J = \underbrace{\sum_{n=1}^{N-1} \left[\frac{1}{2} x_n^\top Q_n x_n + \frac{1}{2} u_n^\top R_n u_n \right]}_{\text{running cost}} + \underbrace{\frac{1}{2} x_N^\top Q_N x_N}_{\text{terminal cost}}$$

Assumptions:

- (A_n, B_n) is *controllable* and (A_n, C_n) is *observable*
- $Q_n \succeq 0, R_n \succeq 0, Q_N \succeq 0$

Problem Formulation and Optimality Conditions

Consider the deterministic discrete-time optimal control problem:

$$\begin{aligned} \min_{x_{1:N}, u_{1:N-1}} \quad & \sum_{n=1}^{N-1} l(x_n, u_n) + l_F(x_N) \\ \text{s.t.} \quad & x_{n+1} = f(x_n, u_n) \\ & u_n \in \mathcal{U} \end{aligned}$$

The first-order necessary conditions for optimality can be derived using:

- The Lagrangian framework (special case of KKT conditions)
- Pontryagin's Minimum Principle (PMP)

Lagrangian Formulation

Form the Lagrangian:

$$L = \sum_{n=1}^{N-1} l(x_n, u_n) + \lambda_{n+1}^\top (f(x_n, u_n) - x_{n+1}) + l_F(x_N)$$

Define the **Hamiltonian**:

$$H(x_n, u_n, \lambda_{n+1}) = l(x_n, u_n) + \lambda_{n+1}^\top f(x_n, u_n)$$

Rewrite the Lagrangian using the Hamiltonian:

$$L = H(x_1, u_1, \lambda_2) + \left[\sum_{n=2}^{N-1} H(x_n, u_n, \lambda_{n+1}) - \lambda_n^\top x_n \right] + l_F(x_N) - \lambda_N^\top x_N$$

Optimality Conditions

Take derivatives with respect to x and λ:

$$
\begin{aligned}
\frac{\partial L}{\partial \lambda_n} &= \frac{\partial H}{\partial \lambda_n} - x_{n+1} = f(x_n, u_n) - x_{n+1} = 0 \\
\frac{\partial L}{\partial x_n} &= \frac{\partial H}{\partial x_n} - \lambda_n^\top = \frac{\partial l}{\partial x_n} + \lambda_{n+1}^\top \frac{\partial f}{\partial x_n} - \lambda_n^\top = 0 \\
\frac{\partial L}{\partial x_N} &= \frac{\partial l_F}{\partial x_N} - \lambda_N^\top = 0
\end{aligned}
$$

For u, we write the minimization explicitly to handle constraints:

$$
\begin{aligned}
u_n = \arg\min_u H(x_n, u, \lambda_{n+1}) \\
\text{s.t. } u \in \mathcal{U}
\end{aligned}
$$

Summary of Necessary Conditions

The first-order necessary conditions can be summarized as:

$$
\begin{aligned}
x_{n+1} &= \nabla_{\lambda} H(x_n, u_n, \lambda_{n+1}) \\
\lambda_n &= \nabla_x H(x_n, u_n, \lambda_{n+1}) \\
u_n &= \arg\min_u H(x_n, u, \lambda_{n+1}), \quad \text{s.t. } u \in \mathcal{U} \\
\lambda_N &= \frac{\partial l_F}{\partial x_N}
\end{aligned}
$$

In continuous time, these become:

$$
\begin{aligned}
\dot{x} &= \nabla_{\lambda} H(x, u, \lambda) \\
-\dot{\lambda} &= \nabla_x H(x, u, \lambda) \\
u &= \arg\min_{\tilde{u}} H(x, \tilde{u}, \lambda), \quad \text{s.t. } \tilde{u} \in \mathcal{U} \\
\lambda(t_F) &= \frac{\partial l_F}{\partial x}
\end{aligned}
$$

Application to LQR Problems

For LQR problems with quadratic cost and linear dynamics:

$$
\begin{aligned}
l(x_n, u_n) &= \frac{1}{2}(x_n^\top Q_n x_n + u_n^\top R_n u_n) \\
l_F(x_N) &= \frac{1}{2} x_N^\top Q_N x_N \\
f(x_n, u_n) &= A_n x_n + B_n u_n
\end{aligned}
$$

The necessary conditions simplify to:

$$
\begin{aligned}
x_{n+1} &= A_n x_n + B_n u_n \\
\lambda_n &= Q_n x_n + A_n^\top \lambda_{n+1} \\
\lambda_N &= Q_N x_N \\
u_n &= -R_n^{-1} B_n^\top \lambda_{n+1}
\end{aligned}
$$

This forms a linear two-point boundary value problem.

Indirect Shooting Algorithm for LQR

Procedure:

1. Make initial guess for control sequence $u_{1:N-1}$
2. **Forward pass:** Simulate dynamics to get state trajectory $x_{1:N}$
3. **Backward pass:**
   - Set terminal costate: $\lambda_N = Q_N x_N$
   - Compute costate trajectory: $\lambda_n = Q_n x_n + A_n^\top \lambda_{n+1}$
   - Compute control adjustment: $\Delta u_n = -R_n^{-1} B_n^\top \lambda_{n+1} - u_n$
4. **Line search:** Update controls $u_n \leftarrow u_n + \alpha \Delta u_n$
5. Iterate until convergence

LQR as Quadratic Programming Problem

Assume x_1 is given, define the decision variable vector and the block-diagonal matrix:

$$z = \begin{bmatrix} u_1 \\ x_2 \\ u_2 \\ \vdots \\ x_N \end{bmatrix}, \qquad H = \begin{bmatrix} R_1 & & & & \\ & Q_2 & & & \\ & & R_2 & & \\ & & & \ddots & \\ & & & & Q_N \end{bmatrix}$$

The dynamics constraints can be expressed as

$$\underbrace{\begin{bmatrix} B_1 & -I & & & & \\ & A_2 & B_2 & -I & & \\ & & & \ddots & & \\ & & & A_{N-1} & B_{N-1} & -I \end{bmatrix}}_{C} \begin{bmatrix} u_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \underbrace{\begin{bmatrix} -A_1 x_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{d}$$

QP Formulation and KKT Conditions

The LQR problem becomes the QP:

$$\min_z J = \frac{1}{2} z^\top H z \quad \text{subject to} \quad Cz = d$$

The Lagrangian of this QP is:

$$\mathcal{L}(z, \lambda) = \frac{1}{2} z^\top H z + \lambda^\top (Cz - d)$$

The KKT conditions are:

$$\begin{aligned} \nabla_z \mathcal{L} &= Hz + C^\top \lambda = 0 \\ \nabla_\lambda \mathcal{L} &= Cz - d = 0 \end{aligned}$$

This leads to the linear system:

$$\begin{bmatrix} H & C^\top \\ C & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$$

We get the exact solution by solving one linear system!

KKT System Structure for LQR

The KKT system for LQR has a highly structured sparse form, consider an $N = 4$ case:

$$
\begin{bmatrix}
R_1 & & & & & & B_1^T & \\
& Q_2 & & & & & I & A_2^T \\
& & R_2 & & & & & B_2^T \\
& & & \color{green}{Q_3} & & & \color{green}{-I} & \color{green}{A_3^T} \\
& & & & \color{red}{R_3} & & & \color{red}{B_3^T} \\
& & & & & \color{blue}{Q_4} & & \color{blue}{-I} \\
B_1 & -I & & & & & & \\
& A_2 & B_2 & -I & & & & \\
& & A_3 & B_3 & -I & & &
\end{bmatrix}
\begin{bmatrix}
u_1 \\ x_2 \\ u_2 \\ x_3 \\ u_3 \\ x_4 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4
\end{bmatrix}
=
\begin{bmatrix}
0 \\ 0 \\ 0 \\ \color{green}{0} \\ \color{red}{0} \\ \color{blue}{0} \\ -A_1 x_1 \\ 0 \\ 0
\end{bmatrix}
$$

Deriving the Riccati Recursion

Start from the terminal condition (blue equation):

$$Q_4 x_4 - \lambda_4 = 0 \Rightarrow \lambda_4 = Q_4 x_4$$

Move to the previous equation (red equation):

$$R_3 u_3 + B_3^\top \lambda_4 = R_3 u_3 + B_3^\top Q_4 x_4 = 0$$

Substitute $x_4 = A_3 x_3 + B_3 u_3$:

$$R_3 u_3 + B_3^\top Q_4 (A_3 x_3 + B_3 u_3) = 0$$

Solve for u_3:

$$u_3 = -\underbrace{(R_3 + B_3^\top Q_4 B_3)^{-1} B_3^\top Q_4 A_3}_{K_3} x_3$$

Deriving the Riccati Recursion (Cont'd)

Now consider the green equation:

$$Q_3 x_3 - \lambda_3 + A_3^\top \lambda_4 = 0$$

Substitute $\lambda_4 = Q_4 x_4$ and $x_4 = A_3 x_3 + B_3 u_3$:

$$Q_3 x_3 - \lambda_3 + A_3^\top Q_4 (A_3 x_3 + B_3 u_3) = 0$$

Substitute $u_3 = -K_3 x_3$:

$$Q_3 x_3 - \lambda_3 + A_3^\top Q_4 (A_3 x_3 - B_3 K_3 x_3) = 0$$

Solve for λ_3:

$$\lambda_3 = \underbrace{(Q_3 + A_3^\top Q_4 (A_3 - B_3 K_3))}_{P_3} x_3$$

Riccati Recursion Formula

We now have a recursive relationship. Generalizing:

$$
\begin{aligned}
P_N &= Q_N \\
K_k &= (R_k + B_k^\top P_{k+1} B_k)^{-1} B_k^\top P_{k+1} A_k \\
P_k &= Q_k + A_k^\top P_{k+1} (A_k - B_k K_k)
\end{aligned}
$$

This is the celebrated **Riccati equation**.

The solution process involves:

1. A **backward Riccati pass** to compute P_k and K_k for $k = N-1, \ldots, 1$
2. A **forward rollout** to compute $x_{1:N}$ and $u_{1:N-1}$ using $u_k = -K_k x_k$

Computational Complexity

Naive QP Solution: Treats problem as one big least-squares.

- Computational cost: $\mathbf{O}[\mathbf{N}^3(\mathbf{n}+\mathbf{m})^3]$
- Must be re-solved from scratch for any change.

Riccati Recursion: Exploits the temporal structure.

- Computational cost: $\mathbf{O}[\mathbf{N}(\mathbf{n}+\mathbf{m})^3]$
- **Exponentially faster** for long horizons (large N).

The Riccati Solution is More Than Just Fast:

- It provides a **ready-to-use feedback policy**: $\mathbf{u_k} = -\mathbf{K_k}\mathbf{x_k}$
- This policy is **adaptive**: optimal for *any* initial state x_1, not just a single one.
- It enables **real-time control** by naturally rejecting disturbances.
- And it delivers the **exact same optimal solution** as the QP.

Summary

Finite-Horizon Problems

- Use Riccati recursion backward in time
- Store gain matrices K_n
- Apply time-varying feedback

Infinite-Horizon Problems

- Solve algebraic Riccati equation offline
- Use constant gain matrix K_∞
- Implement simple state feedback
- Algebraic Riccati Equation (ARE):

$$P_\infty = Q + A^\top P_\infty A - A^\top P_\infty B(R + B^\top P_\infty B)^{-1} B^\top P_\infty A$$

Thank you for listening !

Zirui Zhang