Linear Quadratic Control

Leo Jaeeun Kim
University of Michigan
General Motors

ABSTRACT

Linear Quadratic Control (LQC) is a fundamental framework for optimizing the performance of dynamic systems, making it indispensable in autonomous systems, robotics, and aerospace applications. This paper explores the theoretical principles and practical implementations of LQC, focusing on its ability to balance performance metrics such as stability, energy efficiency, and trajectory accuracy. In autonomous driving, LQC is employed for tasks like trajectory planning, stability control, and obstacle avoidance. In robotics, it supports precise manipulation and navigation, while in aerospace, it governs flight path optimization and attitude stabilization. By solving quadratic cost functions under linear dynamic constraints, LQC enables robust control strategies even in complex, dynamic environments. The study highlights LQC's computational efficiency and adaptability, emphasizing opportunities for extending its applications to nonlinear systems and integrating emerging technologies such as machine learning to enhance real-time decision-making.

INTRODUCTION

The rapid advancement of autonomous systems, robotics, and aerospace technologies has significantly increased the need for precise and efficient control strategies. Linear Quadratic Control (LQC) has emerged as a key tool in these domains, providing an optimal control framework that minimizes a quadratic cost function under linear dynamic constraints. By balancing objectives such as energy efficiency, trajectory accuracy, and system stability, LQC enables robust and reliable control across diverse applications.

In autonomous driving, LQC ensures vehicle safety and performance through trajectory planning, stability control, and collision avoidance, leveraging real-time sensor data to adapt to dynamic road conditions. In robotics, LQC supports motion planning, robotic arm manipulation, and mobile platform control. In aerospace, it plays a crucial role in flight control, attitude stabilization, and trajectory optimization, benefiting from its analytical tractability and computational efficiency.

This paper explores the theoretical foundations of LQC and its practical applications across these interconnected fields, focusing on constrained least squares optimization, state-space modeling, and feedback control strategies. Limitations in handling nonlinear dynamics are addressed, with future directions such as hybrid control approaches and machine learning integration proposed to enhance applicability in complex scenarios.

With the foundational concepts of Linear Quadratic Regulation (LQR) covered, this study transitions into hands-on problem-solving using Python. Tools like the CVX library for convex optimization will be used to demonstrate key concepts, including feedback gain computation for proportional controllers (similar to LQR) and system controllability. This integration emphasizes LQC's significance in addressing modern engineering challenges.

METHOD

Linear least squares problem is a problem of minimizing object function, linear system.

$$f(x) = \frac{1}{2}\left||Ax - b|\right|^2$$
$$\nabla f(x) = A^T Ax - A^T b$$
$$H(x) = A^T A$$

A(Matrix): Represents a transformation or system of equations, typically of size $m \times n$, where m is the number of equations (or data points) and n is the number of variables (or features). ***x(Vector):*** The unknown variable or of size of $n \times 1$ we aim to solve for. **b(Vector):** A given vector of size $m \times 1$, representing the target or observed values in a least squares problem.

The function f(x) represents the squared error loss function, commonly used in linear regression.

The gradient($\nabla f(x)$) is the first derivative of the function and shows how the function changes with respect to *x*, representing the direction and rate of the steepest ascent. In optimization, setting the gradient to zero helps identify critical points, which can be local minima, maxima, or saddle points. The Hessian*(H(x))*, a matrix of second-order derivatives, provides information about the function's curvature. When the Hessian is positive definite, it ensures a unique global minimum, guaranteeing an optimal solution.

1) Linear Regression with L2- regularizer

This principle is fundamental in machine learning, regularized linear regression helps improve model performance by preventing overfitting. One common approach is Ridge Regression, which uses L2 regularization to penalize large weights.

Given a dataset $\{(x_i, t_i)\}_{i=1}^N$,where *x* is the input(features) and *t* is the output(target), aim to find a linear model that explains data. The function *y(x;w)* is a linear combination of basis functions $\phi_j(x)$ weighted by parameters w_j.

$$y(x; w) = \sum_{j=0}^{n} w_j \phi_j(x) = w^T \phi(x)$$

Here, w is a vector of weights and $\phi(x)$ is vector of basis function. The first basis function $\phi_0 = 1$ and w_0 serves as a bias term. In Ridge Regression, we minimize the sum of squared errors between predicted values and actual targets, plus a penalty on large weights:

$$\frac{1}{2}\sum_{i=0}^{N}(t_i - w^T\phi(x_i)^2 + \frac{\lambda}{2}||w||^2$$

The goal is to find weights w that minimize the sum of squared errors between predicted values $y(x_i; w)$and actual targets t_i with an L2 regularization term to prevent overfitting. The problem is rewritten in matrix form, where:

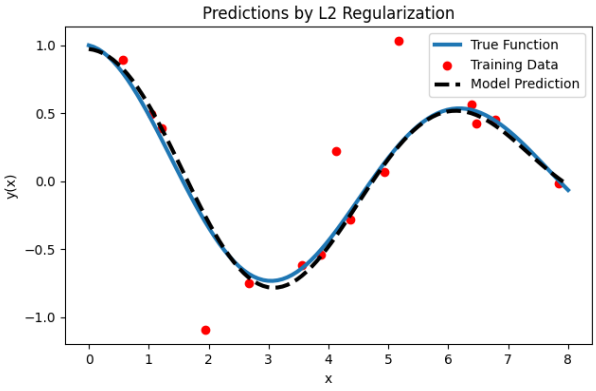
$$f(w) \coloneqq \frac{1}{2}||t - \Phi w||^2 + \frac{\lambda}{2}||w||^2$$

Φ is the design matrix containing basis functions for all training samples. The gradient of loss function is

$$\nabla f(w) = \Phi^{\mathrm{T}}\Phi w - \Phi^{\mathrm{T}}t + \lambda w$$

and setting ∇*f(w)* to 0 to find the optimal weights:

$$w^* = (\Phi^{\mathrm{T}}\Phi + \lambda I)^{-1}\Phi^{\mathrm{T}}t$$



The example code for L2 regularization can be found in https://github.com/woa0425/Leo_ADAS_script/blob/main/LQC/LQC_python/L2_regularization.py

2) Least squares with equality constraints

In least squares problems, we minimize the sum of squared errors between observed data and model predictions. When equality constraints are added, we require the solution to satisfy specific linear constraints, even while minimizing the least squares objective.

Minimize: $\| Ax - b \|^2$ Subject to: $Cx = d$

A: Design matrix that represents the relationship between the independent variables(Inputs) and the dependent variable(output) in a linear model. Role is maps the unknown variable x to the observation vector b. Each row corresponds to an observation and each column represents a feature or basis function. If A = 3×2 matrix, it has 3 observations and 2 features.

b: Observation vector,A column vector containing the observed values of the dependent variable for the given input. Rerpresents the data we are trying to fit using the model. A vector of size equal to the number of observations.

C: Constraint matrix, A matrix defining the linear equality constraints that the solution x must satisfy. Imposes specific restrictions or conditions on the solution. Each row represents one constraint, and each column corresponds to a variable in x.

d: Right-hand side of the constraints, A column vector representing the values that the constraints must equal. Specifies the exact values the constraints should satisfy.

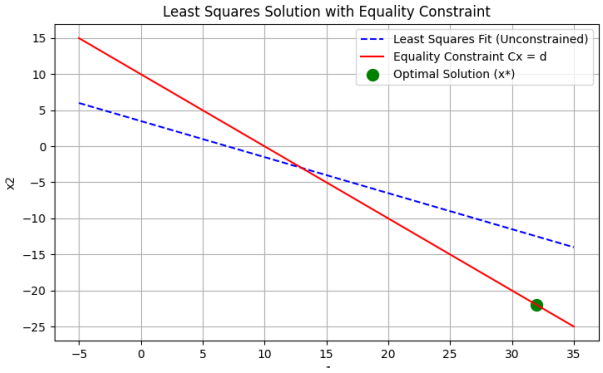
x: variable to solve

λ: Mathematical method used to solve optimization problems with constraints. They allow us to find the maximum or minimum of a function subject to one or more constraints. λ are the Lagrange multipliers.

$$L(x, \lambda) = \| Ax - b \|^2 + \lambda^T(Cx - d)$$

$$\begin{bmatrix} A^TA & C^T \\ C & 0 \end{bmatrix}\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} A^Tb \\ d \end{bmatrix}$$

We can solve for x that the least squares solution that satisfies the constraint. The goal of least squares is to find the minimum value which represents the sum of squared differences (residuals) between the observed data (b) and the predicted data (Ax). This is for the equality constraints. The difference between equality constraints and inequality constraints lies in how they restrict the solution space of an optimization problem. An equality constraint requires that a specific condition be exactly satisfied. An inequality constraint requires that a specific condition be satisfied within a range. The unconstrained solution minimizes the error $\| Ax - b \|^2$ but does not necessarily satisfy additional constraints. When an equality constraint $Cx = d$ is imposed, the solution is adjusted using Lagrange multipliers to enforce feasibility, even if it increases the residual error. The unconstrained fit provides the best approximation, while the constrained solution ensures compliance with imposed conditions, highlighting a trade-off between accuracy and constraint satisfaction.



The example code for Least squares with equality constraints can be found in https://github.com/woa0425/Leo_ADAS_script/blob/main/LQC/LQC_python/LS_equality_constratins.py

3) LQR

The Linear Quadratic Control (LQC) is an optimal control framework for designing controllers for linear systems with a quadratic cost function. The goal is to compute a control input *u(t)* that: Drives the system to a desired state (e.g., stabilize it, track a trajectory). Minimizes a quadratic cost function, which penalizes both: the deviation of the state from the desired state and the control effort (energy usage). The Linear Quadratic Regulator (LQR) offers several benefits by optimizing system performance while balancing control effort. It ensures an optimal trade-off between maintaining system states close to the desired trajectory, penalized by the matrix Q, and minimizing control input effort, penalized by R, which prevents excessive actuator usage and reduces energy consumption. The feedback control law derived from minimizing the cost function enhances system stability, ensuring robust behavior even in the presence of disturbances or uncertainties. Additionally, LQR improves energy efficiency by minimizing unnecessary control actions, prolonging actuator lifespan. Its predictable and tunable behavior allows designers to adjust performance priorities using Q and R, enabling faster state convergence or reduced control effort as needed. Moreover, LQR provides robust control in multi-variable systems by balancing trade-offs among all states and control inputs, ensuring an optimized and stable response.

In the Linear Quadratic Regulator (LQR) optimization problem for a discrete-time system, the goal is to minimize a quadratic cost function that balances output deviations and control effort, with a trade-off parameter ρ adjusting the priority between these objectives. This optimization framework ensures that state evolution follows the system dynamics

$$x_{t+1} = A_t x_t + B_t u_t$$

x_t: State of the system at time t
u_t: Control input at time t
A_t, B_t: System matrices that describe how the state evolves with time and control

Boundary Conditions:

$$x_1 = x^{init}, x_T = x^{des}$$

The total cost to minimize is

$$J_{total} = J_{output} + \rho J_{input}$$
$$J_{output} = ||y_1||^2 + \cdots + ||y_T||^2 = ||C_1 x_1||^2 + \cdots + ||C_T x_T||^2$$
$$J_{input} = ||u_1||^2 + \cdots + ||u_{T-1}||^2$$

Output cost (J_{output}) penalizes deviations of system output and input cost (J_{input})penalizes the control effort (energy). Trade-off parameter ρ is a positive scalar used to balance the importance of minimizing output deviation and minimizing control effort. Larger ρ places more weight on reducing control input and smaller ρ focuses more on achieving accurate outputs. The cost function to minimize in the LQR problem is given by:

$$J = \int (x^T Q x + u^T R u) dt$$

where Q penalizes deviations in the state x, and penalizes the control input u. The optimal solution requires solving the Algebraic Riccati Equation (ARE), where P is a symmetric positive-definite matrix that satisfies:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

Once P is determined, the optimal feedback gain is computed as

$$K = R^{-1}B^T P$$

This gives the control law of u = -*Kx*

4) LQR Implementation

The Linear Quadratic Regulator (LQR) is implemented to stabilize an inverted pendulum on a moving cart, a well-known benchmark in control theory. The system is inherently unstable, requiring continuous feedback control to maintain the pendulum in an upright position. The LQR technique provides an optimal feedback control law that balances state deviation and control effort.

The linearized state-space model of the inverted pendulum is represented as:

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

Where *x* is the state vectors: $x = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T$ and u is the control force applied to the cart. The system matrices are derived from the physical parameters of the system:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(M+m)g}{lm} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & \frac{1}{M} & 0 & -\frac{1}{lM} \end{bmatrix}^T$$

where: M is the mass of the cart, m is the mass of the pendulum, l is the pendulum length, g is gravitational acceleration.

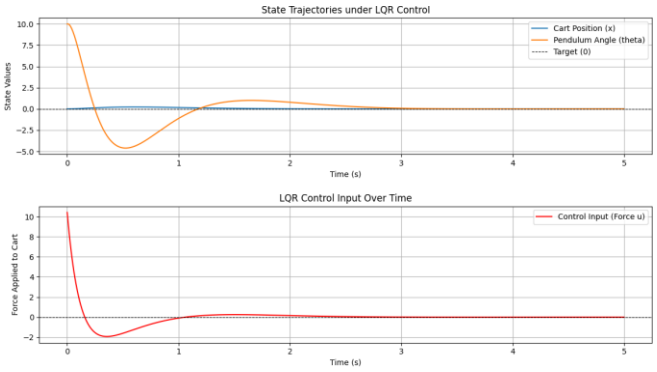
The control objective is to stabilize the pendulum upright (θ = 0) while minimizing cart displacement.

Initially, the pendulum starts with a 10-degree deviation from the upright position, representing a perturbed state. The trajectory of θ shows a rapid correction as the controller applies optimal forces to drive the pendulum back to its upright equilibrium. Similarly, the cart's position is adjusted to counteract the pendulum's motion while maintaining minimal displacement. The quick convergence of the state trajectories to zero indicates that the LQR successfully stabilizes the system while ensuring that the pendulum remains upright with minimal oscillations.

The LQR control input over time represents the force applied to the cart to achieve stabilization. Initially, the control force exhibits a higher magnitude as the system compensates for the large initial deviation. Over time, the control input gradually decreases, demonstrating LQR's energy-efficient approach to stabilization. The diminishing control effort highlights the optimal trade-off between state correction and minimal control energy, achieved through the weight matrices Q and R. The relatively small control effort after the initial transient phase confirms that LQR efficiently reduces oscillations and stabilizes the system without excessive actuator force.

Furthermore, the computed closed-loop eigenvalues of the system matrix ($A-Bk$) are all negative, confirming asymptotic stability of the controlled system. This ensures that the state variables will not diverge and that the system will maintain equilibrium under LQR regulation. The effectiveness of the control strategy is also evident in the smooth trajectory of the pendulum angle, as opposed to abrupt corrective actions, which could destabilize the system.

Overall, the results validate that LQR effectively stabilizes the inverted pendulum by optimally balancing state deviation correction and control effort minimization. The state trajectories confirm smooth convergence to equilibrium, while the control input profile highlights energy-efficient stabilization



The example code inverted pendulum on a moving cart can be found in

https://github.com/woa0425/Leo_ADAS_script/blob/main/LQC/LQC_python/LQR_InvertedPendulum.py

5) Karush-Kuhn-Tucker (KKT)

The Karush-Kuhn-Tucker (KKT) conditions are a set of necessary conditions that a solution must satisfy for a constrained optimization problem (both equality and inequality constraints). These conditions generalize the Lagrange multiplier method to handle inequality constraints as well. The KKT conditions combine stationarity, primal feasibility, dual feasibility, and complementary slackness. Details shown below:

- Stationarity: The gradient of the Lagrangian must be zero.
- Primal Feasibility: The constraints must be satisfied.
- Dual Feasibility: The Lagrange multipliers for the inequality constraints must be non-negative.
- Complementary Slackness: For each inequality constraint, either the constraint is active or the corresponding Lagrange multiplier is zero.

The relationship between the Linear Quadratic Regulator (LQR) and the Karush-Kuhn-Tucker (KKT) conditions lies in their connection to optimal control and constrained optimization. LQR solves an unconstrained quadratic cost minimization problem for a linear system, typically using the Riccati equation. When constraints on states or inputs are introduced, the problem becomes a constrained optimization task, where the KKT conditions define the necessary optimality conditions. The Lagrange multipliers in KKT correspond to the costate variables in the Hamiltonian formulation, linking LQR to the Pontryagin Minimum Principle and demonstrating how constrained optimization extends classical LQR solutions.

Closed-form solution of LQR via KKT condition

KKT condition for standard QP(Quadratic Programming)
For QP in form, with P >0:

$$\min \frac{1}{2} s^T P_s + Q^T s, \qquad s.t. Cs = d$$

$$L(s,u,\lambda) = \frac{1}{2} s^T P_s + q^T s + \lambda(Cs + d)$$
$$\nabla_x L(s,u,\lambda) = P_s + q + C^T \lambda = 0$$
$$\nabla_\lambda L(s,u,\lambda) = C_x - d = 0$$

Which is:

$$\begin{bmatrix} P & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} s \\ \lambda \end{bmatrix} = \begin{bmatrix} -q \\ d \end{bmatrix}$$

6) KKT Implementation

This study implements and compares two discrete-time linear quadratic regulator (LQR) approaches to stabilize the system $x_{k+1} = A_d x_k + B_d u_k$, where x_k is the state vector and u_k is the control input at step k. The system matrices A_d and B_d arise from discretizing the continuous-time dynamics:

$$\dot{x}(t) = A_c x(t) + B_c u(t)$$

using a Taylor expansion to approximate the matrix exponential. Both controllers seek to minimize a quadratic cost balancing state deviations (weighted by Q) and control effort (weighted by R). However, they differ in how they solve for optimal controls. KKT-based LQR uses a finite-horizon preview of length horizon, solving a large linear system (the KKT system) at every step. On the other hand, simple LQR solves the discrete Riccati equation for an infinite-horizon gain K, i.e., $u_k = -Kx_k$

The first method, referred to as KKT-based LQR, applies a finite-horizon preview by aggregating all future states and control inputs into a single decision vector. A quadratic cost matrix P encodes penalties on state deviation and control effort, and a constraint matrix C enforces the linear dynamics over the horizon. Finited-horizon quadratic cost:

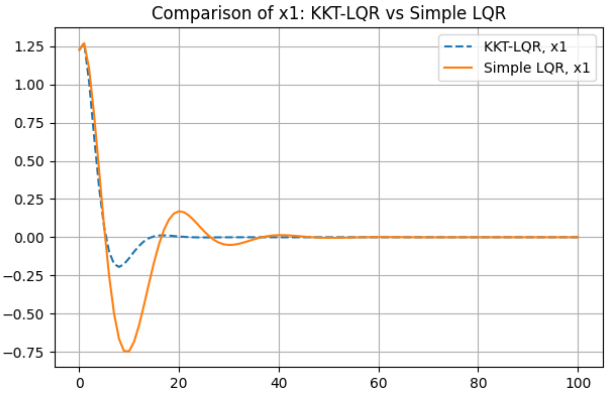
$$J = \sum_{k=0}^{N-1} (x_N^T \mathrm{Q} x_k + u_k^T \mathrm{R} u_k) + x_N^T \mathrm{Q} x_N$$

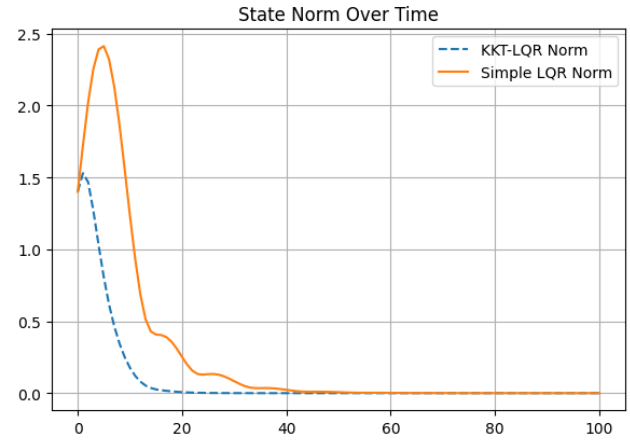
Subject to the dynamics $x_{k+1} = A_d x_k + B_d u_k$. By stacking all $(x_0, x_1, \ldots, x_N, u_0, \ldots, u_{N-1})$ into a single decision vector. The code builds a cost matrix P capturing Q and R along the diagonal and a constraint matrix C encoding the linear relation $x_{k+1} - A_d x_k - B_d u_k = 0$. Solve the KKT system

$$\begin{bmatrix} P & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} s \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$$

The second method, called Simple LQR, computes a time-invariant feedback gain K from the algebraic Riccati equation, yielding the control law $u_k = -Kx_k$, assuming the pair (A_d, B_d) is stabilizable. Both methods rely on a simulation function that advances the system over a specified number of steps and records the resulting trajectories for visualization. By adjusting the weight matrices Q and R, the trade-off between rapid convergence and minimized control effort is made evident, illustrating the distinctions between the finite-horizon KKT approach and the traditional infinite-horizon solution.

In the results, the KKT-based LQR allocates more aggressive control actions by weighting state deviations more heavily relative to control inputs. This emphasis encourages rapid convergence to the desired trajectory at each step, particularly because the method re-solves with updated initial conditions and a finite preview horizon. Meanwhile, the simple LQR approach relies on a single infinite-horizon Riccati solution that often penalizes control effort more strongly, leading to smaller adjustments and slower convergence. Consequently, the KKT-based formulation appears to stabilize the system more effectively under the chosen cost matrices.





The example code for Closed-form solution of LQR via KKT condition can be found in https://github.com/woa0425/Leo_ADAS_script/blob/main/LQC/LQC_python/LQR_via_KKT.py

CONCLUSION

. Linear Quadratic Control (LQC) is a fundamental and versatile framework for optimizing dynamic systems by balancing stability, energy efficiency, and trajectory accuracy. This study examined its theoretical foundations and practical applications, demonstrating its effectiveness in diverse fields such as autonomous driving, robotics, and aerospace. By incorporating constrained optimization through Karush-Kuhn-Tucker (KKT) conditions, LQR extends to complex scenarios where adaptive and robust control strategies are required. The implementation methods discussed, including solutions via the Algebraic Riccati Equation (ARE) and quadratic programming formulations, emphasize LQC's computational efficiency and analytical rigor.

In autonomous systems, LQR plays a critical role in trajectory planning, stability management, and collision avoidance, ensuring safe and efficient navigation. In robotics, it enables precise motion control and manipulation, while in aerospace applications, it contributes to flight stability and optimal trajectory tracking. The integration of constrained least squares optimization further enhances LQC's applicability in scenarios

requiring strict control effort limitations or state evolution constraints.

Despite its strengths in handling linear systems, LQR faces challenges in real-world applications due to nonlinear dynamics. Addressing these limitations, future research directions include extending LQR to nonlinear systems through hybrid control approaches, integrating reinforcement learning for adaptive control, and leveraging machine learning techniques to enhance real-time decision-making. By bridging classical control methodologies with advanced computational strategies, LQC continues to drive progress in intelligent and autonomous control systems across multiple engineering domains.

REFERENCES

[1] University of Michigan, NAVARCH 565 – Lecture 05: Linear Quadratic Control, Fall 2023.
[2] University of Michigan, NAVARCH 565 – Lecture 06: Linear Quadratic Control Example, Fall 2023.
[3] University of Michigan CURLY Teaching, "ROB 530 Public – Linear Regression Code Examples," GitHub Repository. Available: https://github.com/UMich-CURLY-teaching/UMich-ROB-530-public/tree/main/code-examples/Python/linear_regression.