18 LINEAR QUADRATIC REGULATOR

18.1 Introduction

The simple form of loopshaping in scalar systems does not extend directly to multivariable (MIMO) plants, which are characterized by transfer matrices instead of transfer functions.
The notion of optimality is closely tied to MIMO control system design. Optimal controllers, i.e., controllers that are the *best* possible, according to some figure of merit, turn out to generate only stabilizing controllers for MIMO plants. In this sense, optimal control solutions provide an automated design procedure - we have only to decide what figure of merit to use. The linear quadratic regulator (LQR) is a well-known design technique that provides practical feedback gains.

18.2 Full-State Feedback

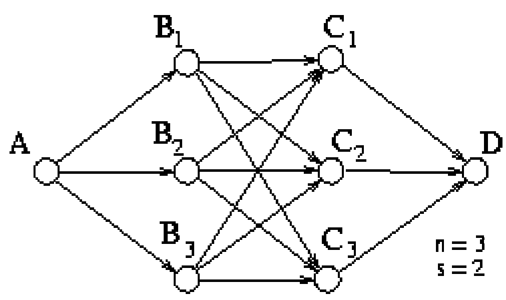
For the derivation of the linear quadratic regulator, we assume the plant to be written in state-space form $\dot{x} = Ax + Bu$, and that *all* of the n states x are available for the controller. The feedback gain is a matrix K, implemented as $u = -K(x - x_{desired})$. The system dynamics are then written as:

$$\dot{x} = (A - BK)x + Kx_{desired}. \qquad (186)$$

$x_{desired}$ represents the vector of desired states, and serves as the external input to the closed-loop system. The ``A-matrix'' of the closed loop system is $(A - BK)$, and the ``B-matrix'' of the closed-loop system is K. The closed-loop system has exactly as many outputs as inputs: n. The column dimension of B equals the number of channels available in u, and must match the row dimension of K. Pole-placement is the process of placing the poles of $(A - BK)$ in stable, suitably-damped locations in the complex plane.

18.3 Dynamic Programming

There are at least two conventional derivations for the LQR; we present here one based on *dynamic programming*, due to R. Bellman. The key observation is best given through a loose example: Suppose that we are driving from Point A to Point C, and we ask what is the shortest path in miles. If A and C represent Los Angeles and Boston, for example, there are *many* paths to choose from! Assume that one way or another we have found the best path, and that a Point B lies along this path, say Las Vegas. Let X be an arbitrary point east of Las Vegas. If we were to now solve the optimization problem for getting from only Las Vegas to Boston, this same arbitrary point X would be along the new optimal path as well.
The point is a subtle one: the optimization problem from Las Vegas to Boston is easier than that from Los Angeles to Boston, and the idea is to use this property *backwards* through time to evolve the optimal path, beginning in Boston.



Example: Nodal Travel. We now add some structure to the above experiment. Consider now traveling from point A (Los Angeles) to Point D (Boston). Suppose there are only three places to cross the Rocky Mountains, B_1, B_2, B_3, and three places to cross the Mississippi River, C_1, C_2, C_3.[3]
By way of notation, we say that the path from A to B_1 is AB_1. Suppose that all of the paths (and distances) from A to the B-nodes are known, as are those from the B-nodes to the C-nodes, and the C-nodes to the terminal point D. There are nine unique paths from A to D.
A brute-force approach sums up the total distance for all the possible paths, and picks the shortest one. In terms of computations, we could summarize that this method requires nine additions of three numbers, equivalent to eighteen additions of two numbers. The *comparison* of numbers is relatively cheap.

The dynamic programming approach has two steps. First, from each B-node, pick the best path to D. There are three possible paths from B_1 to D, for example, and nine paths total from the B-level to D. Store the best paths as $B_1D|_{opt}, B_2D|_{opt}, B_3D|_{opt}$. This operation involves nine additions of two numbers.
Second, compute the distance for each of the possible paths from A to D, *constrained to the optimal paths from the* B*-nodes onward*: $AB_1 + B_1D|_{opt}$, $AB_2 + B_2D|_{opt}$, or $AB_3 + B_3D|_{opt}$. The combined path with the shortest distance is the total solution; this second step involves three sums of two numbers, and total optimization is done in twelve additions of two numbers.
Needless to say, this example gives only a mild advantage to the dynamic programming approach over brute force. The gap widens vastly, however, as one increases the dimensions of the solution space. In general, if there are s layers of nodes (e.g., rivers or mountain ranges), and each has width n (e.g., n river crossing points), the brute force approach will take (sn^s) additions, while the dynamic programming procedure involves only $(n^2(s-1)+n)$ additions. In the case of $n=5$, $s=5$, brute force requires 6250 additions; dynamic programming needs only 105.

18.4 Dynamic Programming and Full-State Feedback

We consider here the regulation problem, that is, of keeping $x_{desired} = 0$. The closed-loop system thus is intended to reject disturbances and recover from initial conditions, but not necessarily follow y-trajectories. There are several necessary definitions. First we define an instantaneous *penalty* function $l(x(t), u(t))$, which is to be greater than zero for all nonzero x and u. The *cost* associated with this penalty, along an optimal trajectory, is

$$J = \int_0^\infty l(x(t), u(t))dt, \tag{187}$$

i.e., the integral over time of the instantaneous penalty. Finally, the *optimal return* is the cost of the optimal trajectory remaining after time t:

$$V(x(t), u(t)) = \int_t^\infty l(x(\tau), u(\tau))d\tau. \tag{188}$$

.
We have directly from the dynamic programming principle

$$V(x(t), u(t)) = \min_u \left\{ l(x(t), u(t))\delta t + V(x(t+\delta t), u(t+\delta t)) \right\}. \tag{189}$$

The minimization of $V(x(t), u(t))$ is made by considering all the possible control inputs u in the time interval $(t, t+\delta t)$. As suggested by dynamic programming, the return at time t is constructed from the return at $t+\delta t$, and the differential component due to $l(x,u)$. If V is smooth and has no explicit dependence on t, as written, then

$$\begin{aligned} V(x(t+\delta t), u(t+\delta t)) &= V(x(t), u(t)) + \frac{\partial V}{\partial x}\frac{dx}{dt}\delta t + h.o.t. \longrightarrow \\ &= V(x(t), u(t)) + \frac{\partial V}{\partial x}(Ax(t) + Bu(t))\delta t. \end{aligned} \tag{190}$$

Now control input u in the interval $(t, t+\delta t)$ cannot affect $V(x(t), u(t))$, so inserting the above and making a cancellation gives

$$0 = \min_u \left\{ l(x(t), u(t)) + \frac{\partial V}{\partial x}(Ax(t) + Bu(t)) \right\}. \tag{191}$$

We next make the assumption that $V(x,u)$ has the following form:

$$V(x,u) = \frac{1}{2}x^T P x, \tag{192}$$

where P is a symmetric matrix, and positive definite.[45] It follows that

$$\begin{aligned} \frac{\partial V}{\partial x} &= x^T P \longrightarrow \\ 0 &= \min_u \left\{ l(x,u) + x^T P (Ax + Bu) \right\}. \end{aligned} \tag{193}$$

We finally specify the instantaneous penalty function. The LQR employs the special quadratic form

$$l(x,u) = \frac{1}{2}x^T Q x + \frac{1}{2}u^T R u, \tag{194}$$

where Q and R are both symmetric and positive definite. The matrices Q and R are to be set by the user, and represent the main ``tuning knobs'' for the LQR. Substitution of this form into the above equation, and setting the derivative with respect to u to zero gives

$$0 = u^T R + x^T P B \qquad (195)$$
$$u^T = -x^T P B R^{-1}$$
$$u = -R^{-1} B^T P x.$$

The **gain matrix** for the feedback control is thus $K = R^{-1} B^T P$. Inserting this solution back into equation 194, and eliminating u in favor of x, we have

$$0 = \frac{1}{2} x^T Q x - \frac{1}{2} x^T P B R^{-1} B^T P + x^T P A x.$$

All the matrices here are symmetric except for PA; since $x^T P A x = x^T A^T P x$, we can make its effect symmetric by letting

$$x^T P A x = \frac{1}{2} x^T P A x + \frac{1}{2} x^T A^T P x,$$

leading to the final matrix-only result

$$0 = Q + PA + A^T P - P B R^{-1} B^T P. \qquad (196)$$

This equation is the *matrix algebraic Riccati equation* (MARE), whose solution P is needed to compute the optimal feedback gain K. The MARE is easily solved by standard numerical tools in linear algebra.

18.5 Properties and Use of the LQR

Static Gain. The LQR generates a static gain matrix K, which is not a dynamical system. Hence, the order of the closed-loop system is the same as that of the plant.

Robustness. The LQR achieves infinite gain margin: $k_g = \infty$, implying that the loci of (PC) (scalar case) or $(det(I + PC) - 1)$ (MIMO case) approach the origin along the imaginary axis. The LQR also guarantees phase margin $\gamma \geq 60$ degrees. This is in good agreement with the practical guidelines for control system design.

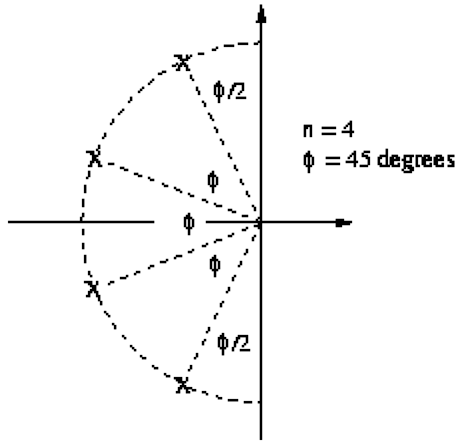
Output Variables. In many cases, it is not the states x which are to be minimized, but the output variables y. In this case, we set the weighting matrix $Q = C^T Q' C$, since $y = Cx$, and the auxiliary matrix Q' weights the plant output.

Behavior of Closed-Loop Poles: Expensive Control. When $R >> C^T Q' C$, the cost function is dominated by the control effort u, and so the controller minimizes the control action itself. In the case of a completely stable plant, the gain will indeed go to zero, so that the closed-loop poles approach the open-loop plant poles in a manner consistent with the scalar root locus.

The optimal control *must* always stabilize the closed-loop system, however, so there should be some account made for unstable plant poles. The expensive control solution puts stable closed-loop poles at the *mirror* images of the unstable plant poles.

Behavior of Closed-Loop Poles: Cheap Control. When $R << C^T Q' C$, the cost function is dominated by the output errors y, and there is no penalty for using large u. There are two groups of closed-loop poles. First, poles are placed at stable plant zeros, and at the mirror images of the unstable plant zeros. This part is akin to the high-gain limiting case of the root locus. The remaining poles assume a Butterworth pattern, whose radius increases to infinity as R becomes smaller and smaller.

The Butterworth pattern refers to an arc in the stable left-half plane, as shown in the figure. The angular separation of n closed-loop poles on the arc is constant, and equal to $180^\circ / n$. An angle $90^\circ / n$ separates the most lightly-damped poles from the imaginary axis.



---

[3] Apologies to readers not familiar with American geography.
[4] Positive definiteness means that $x^T P x > 0$ for all nonzero x, and $x^T P x = 0$ if $x = 0$.
[5] This suggested form for the optimal return can be confirmed after the optimal controller is derived.