Tutorial on Optimal Control – LQR

Tremaine Consulting Group 10/29/2023

This tutorial introduces the concept of *Time-varying Optimal Control* and introduces the *Linear Quadratic Regulator* (LQR). The basic approach is to poise the control problem in an optimization framework. The motivation is that optimization methods provide a convenient method to select the placement of closed loop poles using intuitive trade-offs between state error and control effort. Also, the method works very well with both SISO and MIMO systems.

The LQR method was introduced by *R. Kalman* in 1960 in a series of three papers and was the beginning of optimization in control systems. The reference for this material is *Franklin, Powell & Workman, "Digital Control of Dynamic Systems", 2nd Ed, Addison-Wesley,* and *Bryson and Ho, "Applied Optimal Control", Hemisphere Publishing*.

Math alert: this tutorial is math intensive in the solution development but the final results are very tractable and easy to use in every-day control engineering. The first solution will will apply to *time-varying* optimal control. The results are then easily extended to the steady-state solution.

Solving the LQR

The problem solved here assumes a linear time-invariant (LTI) discrete-time system. Consider a dynamic system defined by:

$$\begin{aligned} \mathbf{x}(\mathbf{k+1}) &= \mathbf{A}\cdot\mathbf{x}(\mathbf{k}) + \mathbf{B}\cdot\mathbf{u}(\mathbf{k}) \\ \mathbf{x}(\mathbf{k}) &\triangleq \mathbb{R}^n \\ \mathbf{u}(\mathbf{k}) &\triangleq \mathbb{R}^m \end{aligned} \qquad [1]$$

The problem to be solved is to design a control **u**(k) based on the system states **x**(k) that will drive the terminal **x**(k) to a desired end state. Note, the problem formulation does not need information on the output matix **C**.

An Optimization Problem

The problem is cast as an optimization problem where we minimize a cost function given by the following.

$$J = \frac{1}{2}\sum_{k=0}^{k=N}\left[\mathbf{x}^{\mathbf{T}}(\mathbf{k})\mathbf{Q}\mathbf{x}(\mathbf{k}) + \mathbf{u}^{\mathbf{T}}(\mathbf{k})\mathbf{R}\mathbf{u}(\mathbf{k})\right] \qquad [2]$$

The weighting matrices **Q** and **R** are defined to be positive definite. The problem can be restated that we wish to minimize

$$\text{min:}\ J = \frac{1}{2}\sum_{k=0}^{k=N}\left[\mathbf{x}^{\mathbf{T}}(\mathbf{k})\mathbf{Q}\mathbf{x}(\mathbf{k}) + \mathbf{u}^{\mathbf{T}}(\mathbf{k})\mathbf{R}\mathbf{u}(\mathbf{k})\right] \qquad [3]$$

subject to the constraint

$$\text{s.t.:}\ -\mathbf{x}(\mathbf{k+1}) + \mathbf{A}\cdot\mathbf{x}(\mathbf{k}) + \mathbf{B}\cdot\mathbf{u}(\mathbf{k}) = \mathbf{0},\ \text{k} = 0,1,2...\text{N}. \quad [4]$$

This is a constrained minima problem that we can solve using the method of *Lagrange multipliers*. The Lagrange function is

$$\check{J} = \frac{1}{2}\sum_{k=0}^{k=N}\left[\mathbf{x^T(k)Qx(k)} + \mathbf{u^T(k)Ru(k)} + \lambda^{\mathbf{T}}\mathbf{(k+1)(-x(k+1) + A\cdot x(k) + B\cdot u(k))}\right] \quad [5]$$

The solution is found as the minimum of $\check{J}$ with respect to **x**(k), **u**(k) and λk). Continuing with the minimization by setting the gradients to zero we arrive at the following three equations.

$$\frac{\partial \check{J}}{\partial \mathbf{u(k)}} = \mathbf{u^T(k)R} + \lambda^{\mathbf{T}}\mathbf{(k+1)B} = \mathbf{0} \qquad [6]$$

$$\frac{\partial \check{J}}{\partial \lambda\mathbf{(k+1)}} = \mathbf{-x(k+1) + Ax(k) + Bu(k)} = \mathbf{0} \qquad [7]$$

$$\frac{\partial \check{J}}{\partial \mathbf{x(k)}} = \mathbf{x^T(k)Q} - \lambda^{\mathbf{T}}\mathbf{(k)} + \lambda^{\mathbf{T}}\mathbf{(k+1)A} = \mathbf{0} \qquad [8]$$

The three equations [6], [7], and [8] are the *control* equation, *state* equation and *adjoint* equation.
The *adjoint* equation can be written as a backward difference equation

$$\lambda^{\mathbf{T}}\mathbf{(k)} = \mathbf{A^T}\lambda\mathbf{(k+1)} + \mathbf{Qx(k)}. \qquad [9]$$

You can think of this equation as starting at k= N+1 and iterating backwards to k = 0.
The above equations can be can be written more concisely in the following three equations,

$$\mathbf{x(k+1)} = \mathbf{Ax(k)} + \mathbf{Bu(k)}, \qquad [10a]$$

$$\mathbf{u(k)} = -\mathbf{R^{-1}}\lambda^{\mathbf{T}}\mathbf{(k+1)}, \qquad [10b]$$

$$\lambda\mathbf{(k+1)} = \mathbf{A^T}\lambda\mathbf{(k)} - \mathbf{A^TQx(k)}. \qquad [10c]$$

The three equations summarized in (10) are a set of coupled difference equations defining the optimal solution of x(k), λ(k) and u(k) given the initial or final conditions.

Boundary Conditions

The initial conditions on **x**(k) must be given. The initial condition $\lambda(0)$ is not known, but we can work backwards from time k = N+1. From the cost function [2], **u**(N) must be zero to minimize J, because **u**(N) has no effect on **x**(N). This yields $\lambda(N+1)$ = 0 from [10b].

From (9) the boundary conditions are then,

$$\lambda(N) = \mathbf{Qx(N)},\ \mathbf{x(0)} \text{ is given.} \qquad [11]$$

The solution of [10a, b, c] from Bryson and Ho (1969) assumes

$$\lambda(k) = \mathbf{S(k)x(k)}. \qquad [12]$$

This transforms the two-point boundary value problem in **x** and λ to one in **S** with a single-point boundary condition. With the definition from [12] and the control equation [10b] we can write the control as

$$\begin{aligned}\mathbf{u}(\mathbf{k}) &= -\left(\mathbf{R}+\mathbf{B}^{\mathbf{T}}\mathbf{S}(\mathbf{k}+\mathbf{1})\mathbf{B}\right)^{-1}\mathbf{B}^{\mathbf{T}}\mathbf{S}(\mathbf{k}+\mathbf{1})\mathbf{A}\mathbf{x}(\mathbf{k}) \\ &= -\bar{\mathbf{R}}\mathbf{B}^{\mathbf{T}}\mathbf{S}(\mathbf{k}+\mathbf{1})\mathbf{A}\mathbf{x}(\mathbf{k})\end{aligned} \qquad [13]$$

After several substitutions using the new definition of $\lambda(\mathbf{k})$ in [12] and the above **u**(k) into the set of equations 10[a], 10[b] and 10[c] the variables $\mathbf{x}$ and λ can be eliminated, (see *"Franklin & Powell"*).

The final result is:

$$\begin{aligned}&\mathbf{S}(\mathbf{k}) = \mathbf{A}^{\mathbf{T}}\mathbf{M}(\mathbf{k}+\mathbf{1})\mathbf{A}+\mathbf{R}, \text{ where} \\ &\mathbf{M}(\mathbf{k}+\mathbf{1}) = \mathbf{S}(\mathbf{k}+\mathbf{1}) - \mathbf{S}(\mathbf{k}+\mathbf{1})\mathbf{B}\left[\mathbf{R}+\mathbf{B}^{\mathbf{T}}\mathbf{S}(\mathbf{k}+\mathbf{1})\mathbf{B}\right]^{-\mathbf{1}}\mathbf{B}^{\mathbf{T}}\mathbf{S}(\mathbf{k}+\mathbf{1})\end{aligned} \qquad [14]$$

The Algebraic Riccati Equation (ARE)

The equation in [14] is the discrete algebraic Riccati equation. It can be solved recursively [16] using the boundary conditions,

$$\begin{aligned}\mathbf{S}(\mathbf{N}) &= \mathbf{Q} \\ \mathbf{K}(\mathbf{N}) &= 0\end{aligned} \qquad [15]$$

I will not derive the recursion equation. Those interested should look at references "*Franklin & Powell*", or "*Bryson and Ho*".

The procedure for solving the time-varying optimal gain is summarized in this pseudo code:

```
$\mathbf{S}(N) = \mathbf{Q}, \mathbf{K}(N) = 0;$
for k= N:1
  $\mathbf{M}(k) = \mathbf{S}(k) - \mathbf{S}(k)\cdot\mathbf{B}\cdot[\mathbf{R} + \mathbf{B}^T\cdot\mathbf{S}(k)\cdot\mathbf{B}]^{-1}\cdot\mathbf{B}^T\cdot\mathbf{S}(k)$
  $\mathbf{K}(k-1) = [\mathbf{R} + \mathbf{B}^T\cdot\mathbf{S}(k)\cdot\mathbf{B}]^{-1}\cdot\mathbf{B}^T\cdot\mathbf{S}(k)\cdot\mathbf{A}$
  $\mathbf{S}(k-1) = \mathbf{A}^T\cdot\mathbf{M}(k)\cdot\mathbf{A} + \mathbf{Q}$
```

[16]

To apply the control for any initial conditions on **x** the state equation and control law are iterated using the stored optimal gain **K**(k).

$$\begin{aligned}&\mathbf{x}(\mathbf{k}+\mathbf{1}) = \mathbf{A}\mathbf{x}(\mathbf{k}) + \mathbf{B}\mathbf{u}(\mathbf{k}), \\ &\mathbf{u}(\mathbf{k}) = -\mathbf{K}(\mathbf{k})\mathbf{x}(\mathbf{k})\end{aligned} \qquad [17]$$

The optimal gain can be computed using the boundary condition **K**(N) = 0. **K** can be precomputed with a known length N, but does *not* require knowledge of the initial state **x**[0]. The optimal **K(**k**)** will change with each time step. We will see this in the example given later.

The Steady State LQR Case

The optimal control gain that minimizes the cost function in the infinite time case is also called the Linear Quadratic *steady state regulator* (LQR) case and it has a constant gain.

In the time-varying case it is found that the gain reaches a constant value we can all **K∞.** This value is a constant we call the solution to the LQR *regulator* problem. One way we could solve for this is to use a “large” terminal value of N and iterate backward in time to k=0, using [16].

Another solution is to look for a steady state solution to the Riccati equation [14]. In steady state **S**(k) = **S**(k+1). We will call this S∞ and simplify the Riccati solution to:

$$\mathbf{S}\infty = -\mathbf{A^T}\left[\mathbf{S}\infty - \mathbf{S}\infty\mathbf{B}\bar{\mathbf{R}}^{-1}\mathbf{B^T}\mathbf{S}\infty\right]\mathbf{A} + \mathbf{Q} \qquad [18]$$

where, $\bar{\mathbf{R}} = \mathbf{R} + \mathbf{B^T}\mathbf{S}\infty\mathbf{B}$

The equation in [18] is called the *algebraic Riccati equation.* Knowing S∞ is positive definite it can be solved using numerical methods.

Luckily for us, we can turn to using modern tools to solve the LQR problem and obtain the optimal feedback gain with a simple function call in MATLAB or Python.

LQR In Practice

In MATLAB the solution to the LQR problem is bundled in the *lqr* command and works with continuous time or discrete time.

*[K,S,CLP] = **lqr**(SYS,Q,R,N) calculates the optimal gain matrix K for the continuous or discrete state-space model SYS. lqr also returns the solution S of the associated algebraic Riccati equation and the closed-loop poles CLP = EIG(A-B*K). The matrix N is set to zero when omitted.*

*[K,S,CLP] = **lqr**(A,B,Q,R,N) is an equivalent syntax for continuous-time models with dynamics dx/dt = Ax+Bu.*

The command *dlqr* can also be used for the discrete-time case, *dlqr Linear-quadratic regulator design for discrete-time systems.*

[K,S,CLP] = ***dlqr****(A,B,Q,R,N) calculates the optimal gain matrix K such that the state-feedback control u[n] = -Kx[n] minimizes the cost function*

*J = Sum {x'Qx + u'Ru + 2*x'Nu}*

subject to the state dynamics x[n+1] = Ax[n] + Bu[n].

The matrix N is set to zero when omitted. Also returned are the Riccati equation solution S and the closed-loop eigenvalues CLP

$$A'SA - S - (A'SB+N)(R+B'SB)^{-1}(B'SA+N') + Q = 0 ,$$

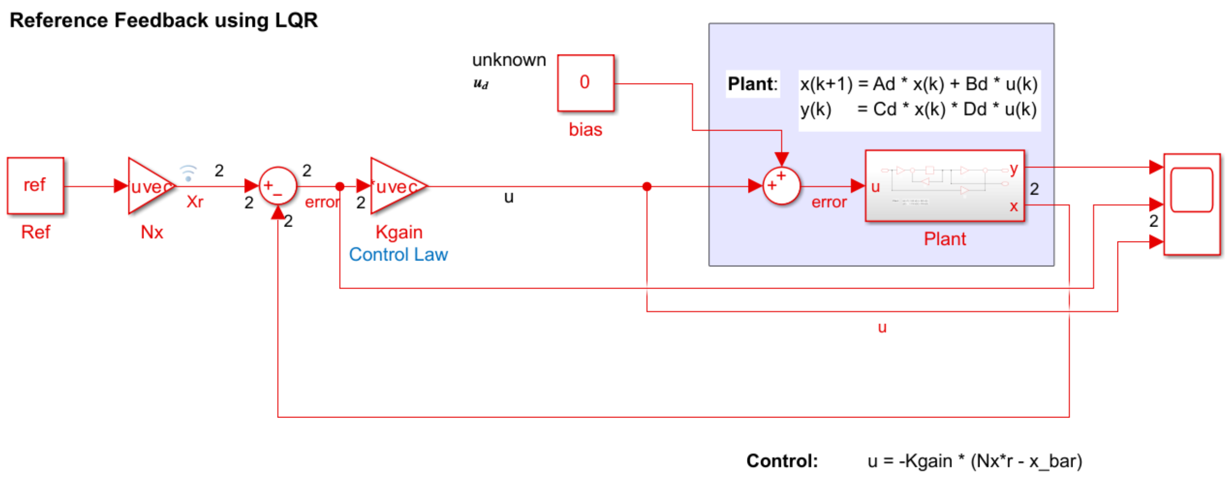
$$CLP = EIG(A-B*K) .$$

The LQR solution is also available in *Python* using the *control* package *control.lqr* module. The Example problems is solved in both MATLAB/SIMULINK 2023a as well as in Python version 3.9.

LQR Example – Time-varying

This problem example is a double integrator system with viscous damping constant kv, and force. constant kf, controlled with full-state feedback and reference feed-forward. The Reference command is 10 units. The sample time is Ts=0.25. The MATLAB script ***lqr_ex1.m*** does all the setting of parameters and calculating the time-varying gains. The SIMULINK file is not needed for this portion.

Figure 1 – Block Diagram lqr_ex1_sim.slx

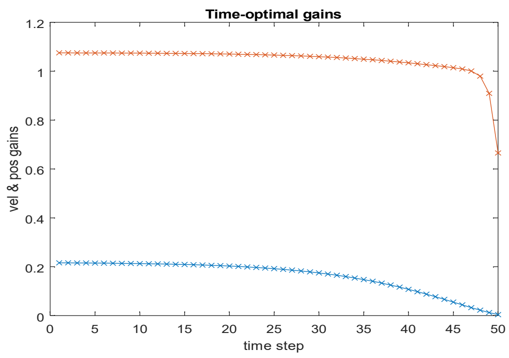

Reference Feedback using LQR
unknown
u_d
0
bias
Plant:
x(k+1) = Ad * x(k) + Bd * u(k)
y(k)   = Cd * x(k) * Dd * u(k)
ref
Ref
uvec
Nx
Xr
2
2
error
2
uvec
Kgain
Control Law
u
error
u
y
x
2
Plant
2
u
Control:
u = -Kgain * (Nx*r - x_bar)

The state equations are **Ad**= [1.0000 0.2497
0 0.9975]
Bd= [0.0624
0.4994]

Using MATLAB, the Controllability matrix is rank 2, so the system is controllable. The state weighting matrix **Q** will be set to [2 0; 0 0.1] to put a heavy penalty on position and a small penalty on velocity.

Next, the feedback gain was calculated assuming N=50 samples. This is a plot of the time-varying gains:

Figure - 2: Time-varying gain with N=50 samples



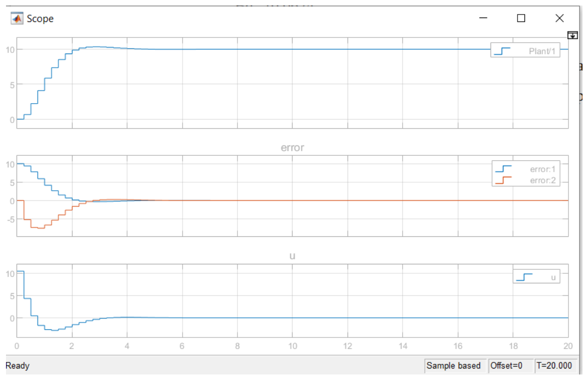
The gain K at time step k=50 is K=[0.2161 1.0752] and at k=1 the gain is [0.0042 0.6656]. For the infinite time case we will show the LQR gain to be K=[0.2161 1.0752] .

LQR Example – Infinite Time

In this example it is the same system as described in the time-varying example. The only change is how we calculate the feedback gain.

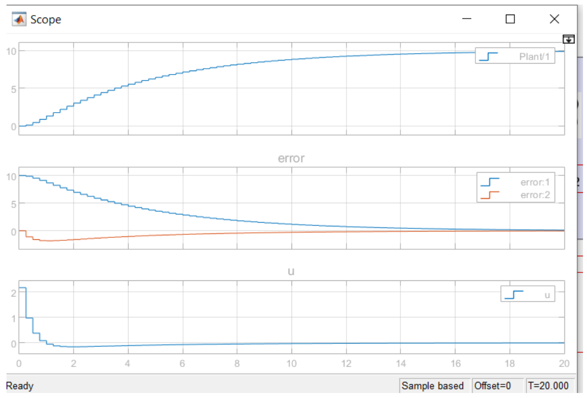
The state weighting matrix **Q** will initially be set to [2 0; 0 0.1] to put a heavy penalty on position and a small penalty on velocity. The R weighting matrix will be set to 1.0. Using *lqr,* the feedback gain is **Kgain** = [.0423 1.0428]. The closed loop eigenvalues are located at z = [0.7058 +/- 0.2088i].

Figure 3 – Infinite time LQR response – Heavy penalty on position



In the next plot the **Q** matrix was changed to [0.1 0; 0 2.0] to put a heavy penalty on velocity and a small penalty on position. The R weighting matrix remained set to 1.0. Using lqr, the feedback gain is **Kgain** = [0.2180   1.0759]. The closed loop eigenvalues are located at z = [0.9455, 0.5012].

Figure 4: – Infinite time LQR response – Heavy penalty on velocity



A Python file, *lqr_example.py* is provided at github that does all the above in the one file. The MATLAB files are also provided.

Closing Remarks

The LQR approach was developed by Richard Kalman in 1960 and was the real beginning of the use of optimization in control systems. The solution provides a time-varying solution as well as a steady state solution.

The LQR method is very accessible in MATLAB or Python and provides a means of calculating the optimal control gain witch is designed to penalize system states as chosen by the designer. The approach works well with both MIMO and SISO systems. The requirements are the system must be LTI and (A,B) must be *stabilizable*.