Nonlinear Least Squares Trajectory Exploration

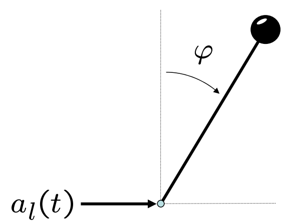
John Hauser

ver. Dec 2012

We consider the problem for finding a trajectory of $\dot{x} = f(x, u)$ that is close to a specified curve $\xi_d = (x_d(\cdot), u_d(\cdot))$ in a weighted L_2 sense. In particular, given symmetric positive definite matrices Q, R, and P_1, we seek to (locally) minimize the least squares functional

$$h(\xi) = \int_0^T \|x(\tau) - x_d(\tau)\|_Q^2/2 + \|u(\tau) - u_d(\tau)\|_R^2/2 \; d\tau + \|x(T) - x_d(T)\|_{P_1}^2/2$$

over trajectories $\xi = (x(\cdot), u(\cdot)) \in \mathcal{T}$. For simplicity, we only require that the desired curve ξ_d be continuous on $[0, T]$.



We will work through the details using the pictured driven inverted pendulum system. The dynamics is given by

$$\ddot{\varphi} = (g/l)\sin\varphi - (1/l)\cos\varphi \; u$$

where the control u is taken to be the pivot point lateral acceleration a_l. In state space form, we have

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} f_1(x, u) \\ f_2(x, u) \end{bmatrix} = \begin{bmatrix} x_2 \\ (g/l)\sin x_1 - (u/l)\cos x_1 \end{bmatrix}$$

The linearization about a trajectory $\xi = (x(\cdot), u(\cdot))$ is given by

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ (g/l)\cos x_1(t) + (u(t)/l)\sin x_1(t) & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -(1/l)\cos x_1(t) \end{bmatrix} v$$

defining $A(x(t), u(t))$ and $B(x(t), u(t))$. We will also make use of the second derivative of $f(\cdot, \cdot)$. Each $D^2 f_i(x, u)$ has a matrix representation: $D^2 f_1(x(t), u(t)) = 0_{3\times 3}$ and

$$D^2 f_2(x(t), u(t)) = \left[\begin{array}{cc|c} -(g/l)\sin x_1(t) + (u(t)/l)\cos x_1(t) & 0 & (1/l)\sin x_1(t) \\ 0 & 0 & 0 \\ \hline (1/l)\sin x_1(t) & 0 & 0 \end{array}\right].$$

A suitable $K(\cdot)$ for finite horizon regulation may be obtained by solving a linear quadratic optimal control problem. To wit, one may choose (with subscript r meaning regulator)

$$K_r(t) = R_r^{-1} B(t)^T P_r(t)$$

where $P_r(\cdot)$ satisfies the Riccati equation

$$\dot{P}_r + A(t)^T P_r + P_r A(t) - P_r B(t) R_r^{-1} B(t)^T P_r + Q_r = 0, \quad P_r(T) = P_{1r}, \tag{1}$$

or, equivalently,

$$\dot{P}_r + A(t)^T P_r + P_r A(t) - K_r(t)^T R_r K_r(t) + Q_r = 0, \quad P_r(T) = P_{1r},$$

with $Q_r = Q_r^T > 0$, $R_r = R_r^T > 0$, and $P_{1r} = P_{1r}^T > 0$. (The matrices Q_r, R_r, and P_{1r} here need not be related to cost function above.) The terminal value P_{1r} is often chosen in a fashion to make it approximately compatible with Q_r and R_r and the linearized system dynamics. For instance, suppose that $(x(T), u(T)) = (x_{eq}, u_{eq})$ is an equilibrium point, $f(x_{eq}, u_{eq}) = 0$, with controllable linearization (A_{eq}, B_{eq}) and let $P_{1r} = P_{1r}^T > 0$ be the stabilizing solution to the associated algebraic Riccati equation. Then the extension of $K_r(\cdot)$ (constant on $t \geq T$) stabilizes the corresponding extension of ξ (also constant on $t \geq T$). Naturally, these comments are also useful in the selection of P_1 for the least squares functional $h(\cdot)$ above.

Suppose now that we have obtained a $K_r(\cdot)$ and we wish to evaluate $\mathcal{P}(\xi)$ and $g(\xi) = h(\mathcal{P}(\xi))$ for some $\xi = (\alpha(\cdot), \mu(\cdot))$ that is not necessarily a trajectory. This is easily accomplished by integrating the augmented system

$$\begin{array}{rclrcl}
\dot{x} & = & f(x, u) & x(0) & = & x_0, \\
u & = & \mu(t) + K_r(t)\,[\alpha(t) - x] & & & \\
\dot{x}_{n+1} & = & \|x - x_d(t)\|_Q^2/2 + \|u - u_d(t)\|_R^2/2 & x_{n+1}(0) & = & 0
\end{array} \tag{2}$$

over $[0, T]$ and noting that

$$g(\xi) = h(\mathcal{P}(\xi)) = x_{n+1}(T) + \|x(T) - x_d(T)\|_{P_1}^2/2\,.$$

The system (2) can be implemented in `matlab` using an S-function system with

- state (x, x_{n+1}),
- input $(\alpha(t), \mu(t), K_r(t), x_d(t), u_d(t))$, $t \in [0, T]$,

and

- output u.

The S-function for evaluating the nonlinear projection operator $\mathcal{P}(\xi)$ together with the cost functional $h(\mathcal{P}(\xi))$ is implemented in `nonl_K.c` which is accessed using the associated `simulink` model `nonl_KS.mdl`. The executable `nonl_K.mex*` is made using `mex nonl_K.c` in the `matlab` command window. The dynamics and cost functions (plus derivatives) are specified using a `dynamics(...)` function and a `cost(...)` function, placed in files called `dynamics.c` and `cost.c`, respectively.

Note that we have followed the convention wherein the argument t is shown *only* for functions that depend *explicitly* on time. The time dependent functions $x(t)$, $u(t)$, $x_{n+1}(t)$, $t \in [0, T]$ are determined (*implicitly*) by solving the differential equation (2).

Cheat Sheet

Given $\xi = (x(\cdot), u(\cdot)) \in \mathcal{T}$, find

$$\zeta = (z(\cdot), v(\cdot)) = \arg \min_{\zeta \in T_\xi \mathcal{T}} \; Dh(\xi) \cdot \zeta + (1/2) D^2 g(\xi) \cdot (\zeta, \zeta)$$

Solve, backward in time,

$$\begin{aligned}
K_o &= R_o^{-1}(S_o^T + B^T P) \\
-\dot{P} &= A^T P + PA - K_o^T R_o K_o + Q_o, \quad P(T) = P_1, \\
-\dot{r} &= (A - BK_o)^T r + a - K_o^T b, \qquad r(T) = r_1, \\
-\dot{q} &= (A - BK_r)^T q + a - K_r^T b, \qquad q(T) = r_1, \\
v_o &= -R_o^{-1}(B^T r + b)
\end{aligned}$$

$$\begin{aligned}
a &= l_x^T && = Q\,(x(t) - x_d(t)) \\
b &= l_u^T && = R\,(u(t) - u_d(t)) \\
Q_o &= l_{xx} + \sum q_k f_{k,xx} && = Q + \sum q_k f_{k,xx} \\
R_o &= l_{uu} + \sum q_k f_{k,uu} && = R + \sum q_k f_{k,uu} \\
S_o &= l_{xu} + \sum q_k f_{k,xu} && = 0 + \sum q_k f_{k,xu} \\
r_1 &= m_x^T && = P_1\,(x(T) - x_d(T)) \\
P_1 &= m_{xx}
\end{aligned}$$

and then, forward in time,

$$\begin{aligned}
\dot{z} &= Az + Bv, && z(0) = 0 \\
v &= -K_o z - R_o^{-1}(B^T r + b) \\
&= -K_o z + v_o \\
\dot{z}_{n+1} &= a^T z + b^T v, && z_{n+1}(0) = 0 \\
\dot{z}_{n+2} &= z^T Q_o z + 2 z^T S_o v + v^T R_o v, && z_{n+2}(0) = 0
\end{aligned}$$

so that, with $\zeta = (z(\cdot), v(\cdot))$,

$$\begin{aligned}
Dh(\xi) \cdot \zeta &= z_{n+1}(T) + r_1^T z(T) \\
D^2 g(\xi) \cdot (\zeta, \zeta) &= z_{n+2}(T) + z(T)^T P_1 z(T)
\end{aligned}$$

When $f(\cdot, \cdot)$ is control-affine, $R_o = R$.

Note: the term R_o will have different meanings in the backward and forward time calculations when using a quadratic form other than $D^2 g(\xi)$ in computing the descent direction. In that case, the expression $v = -K_o z + v_o$ *must* be used since the *optimal* R_o going backwards is then *not* the second order R_o given, but rather whatever has been chosen (to ensure that the LQ problem is solvable). In contrast, the R_o used in computing z_{n+2} (going forward) is *always* the one associated with $D^2 g(\xi)$. (Perhaps a different notation might be developed for the non-Newton direction case...)