An Introductory Course to Model Predictive Control

Jun Chen, Ph.D.

Fall 2022

Contents

Chapter 1

Introduction to Control Theory

The primary objective of control system is to make some object (called system, or plant, or process) behave as we desire. There are a lot of examples of control systems around us:

- Room temperature control
- Car/bicycle driving
- Cruise control or speed control
- Shooting a basketball
- Process control chemical reaction
- etc.

A classical closed-loop feedback control diagram is shown in Fig. 1.1, where

- r is called the reference, or set-point, or desired output.
- y is the variable we want to control, also called "output variable".
- e is the difference between the reference input and the measurement output y_m, also called "error".

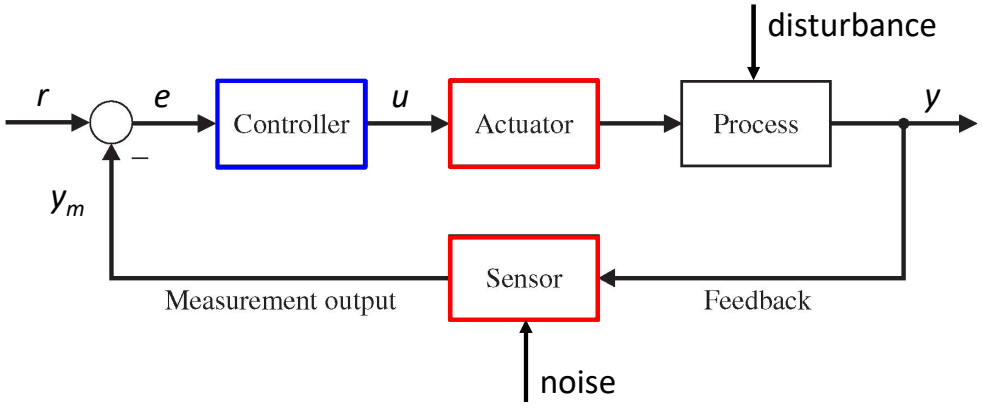

Figure 1.1: Classical closed-loop feedback control diagram [1].

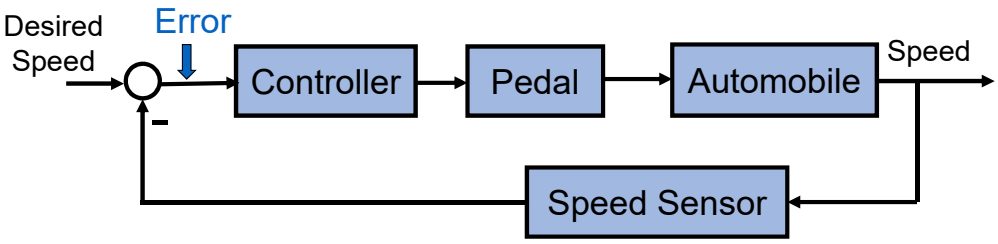

Figure 1.2: Vehicle speed control diagram.

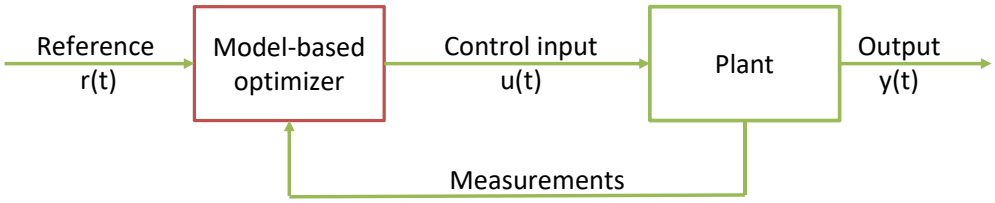

Figure 1.3: Closed-loop feedback control diagram using model-based optimal control method.

- u is the control signal, or control input, which is computed based on the error and applied to the actuator.

The four major blocks of Fig. 1.1 are process, sensor, actuator, and controller. In particular,

- process refers to the system being control, which is subject to disturbance.
- sensor is the measurement device that measures the output of the process. In general, sensor is subject to measurement noise.
- actuator is the one applies control input the the plant.
- controller is the hardware and software that computes a value for control input.

An example of vehicle speed control is shown in Fig. 1.2. In this case, the process is "Automobile", the actuator is "Pedal", the sensor is a "Speed sensor", and the controller can be a "microcomputer" for autonomous vehicles or a "human driver" for human driven vehicles.

Many modern control methods do not explicitly compute the error signal e, though the goal is still to minimize it. Furthermore, in this course, we would focus on model-based optimal control, and do not consider process and sensor dynamics (unless stated otherwise). Therefore, the diagram that we use throughout this course is given in Fig. 1.3.

Chapter 2

Linear Systems

2.1 Continuous-Time Linear Time Invariant Systems

Definition 2.1.1 (**CT LTI System**). *In general, continuous-time linear invariant systems can be described by the following state space equation:*

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.1a}$$
$$y(t) = Cx(t) + Du(t), \tag{2.1b}$$

with initial condition $x(0) = x_0$. Note that $x(t) \in \mathbb{R}^{n_x}$, $u(t) \in \mathbb{R}^{n_u}$ and $y(t) \in \mathbb{R}^{n_y}$. $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_u \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$ are all real-valued matrices.

Equ. (2.1a) is called *state equation* and consists of a set of n_x first order differential equations, while Equ. (2.1b) is called *output equation* and consists of a set of n_y linear algebraic equations. The term $Du(t)$ is called *direct feed through* and is often ignored in many applications (due to actuator dynamics).

Example 2.1.1 (Battery ECM adopted from [2]). *The dynamic of a battery model can often be modeled by a second order equivalent circuit model (ECM), as shown in Fig. 2.1, where the open circuit voltage is denoted as V_{oc} and the terminal voltage is denoted as v. Let the input to be the current i with the convention that a positive value of i denotes discharging and negative value of i denotes charging. Let $x(t) = [x_1, x_2, x_3]^T$ where x_1 is the state-of-charge (SOC) of the battery, $x_2 = V_1$, and $x_3 = V2$. Furthermore, let the input $u = i$ and the output $y = V_{oc} - v$. Then the*

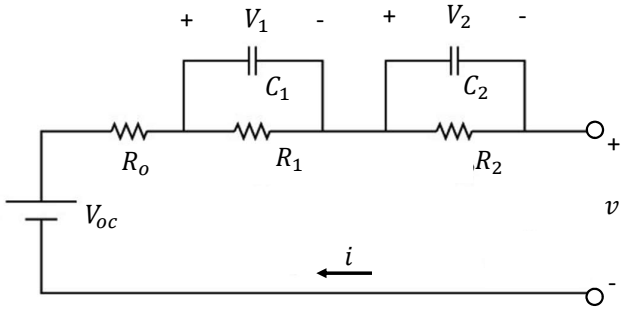

Figure 2.1: Second order equivalent circuit model for battery.

dynamic of the model can be represented as follows.

$$\dot{x}_1 = -\frac{1}{3600C_b}u \tag{2.2a}$$

$$\dot{x}_2 = -\frac{1}{C_1R_1}x_2 + \frac{1}{C_1}u \tag{2.2b}$$

$$\dot{x}_3 = -\frac{1}{C_2R_2}x_3 + \frac{1}{C_2}u \tag{2.2c}$$

$$y = V_{oc} - v = x_2 + x_3 + R_o u \tag{2.2d}$$

Putting it into matrix form, we have

$$\underbrace{\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & -\frac{1}{C_1R_1} & 0 \\ 0 & 0 & -\frac{1}{C_2R_2} \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} -\frac{1}{3600C_b} \\ \frac{1}{C_1} \\ \frac{1}{C_2} \end{bmatrix}}_{B} u \tag{2.3a}$$

$$y = \underbrace{\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}}_{C}\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} R_o \end{bmatrix}}_{D} u. \tag{2.3b}$$

Let $C_b = 2.3$, $R_o = 0.0262$, $R_1 = 0.0433$, $C_1 = 1974.1$, $R_2 = 0.0749$, and $C_2 = 4254.7$, then we have

$$\dot{x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.0117 & 0 \\ 0 & 0 & -0.0031 \end{bmatrix} x + \begin{bmatrix} -1.208 \times 10^{-4} \\ 5.066 \times 10^{-4} \\ 2.35 \times 10^{-4} \end{bmatrix} u \tag{2.4a}$$

$$y = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} x + \begin{bmatrix} 0.0262 \end{bmatrix} u. \tag{2.4b}$$

2.1.1 System Response

The advantage of using LTI models is the ease of solution and analysis. Equ. (2.1a) can be solved to get[1]:

$$x(t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau, \tag{2.5a}$$

where $e^{At} \in \mathbb{R}^{n_x \times n_x}$ is the matrix exponential defined as for any matrix X

$$e^X := \sum_{n=0}^{\infty} \frac{1}{n!}X^n = I + \frac{1}{1!}X + \frac{1}{2!}X^2 + \frac{1}{3!}X^3 + \cdots$$

Then the output response can be obtained from Equ. (2.1b) as:

$$y(t) = Ce^{At}x_0 + C\int_0^t e^{A(t-\tau)}Bu(\tau)d\tau + Du(t). \tag{2.5b}$$

The time-domain solution Equ. (2.5) is useful for analyzing the system properties, but is hard to compute as it involves matrices multiplication for infinite number of times. Another approach to

[1] Details can be found in [3, Chapter 4.2].

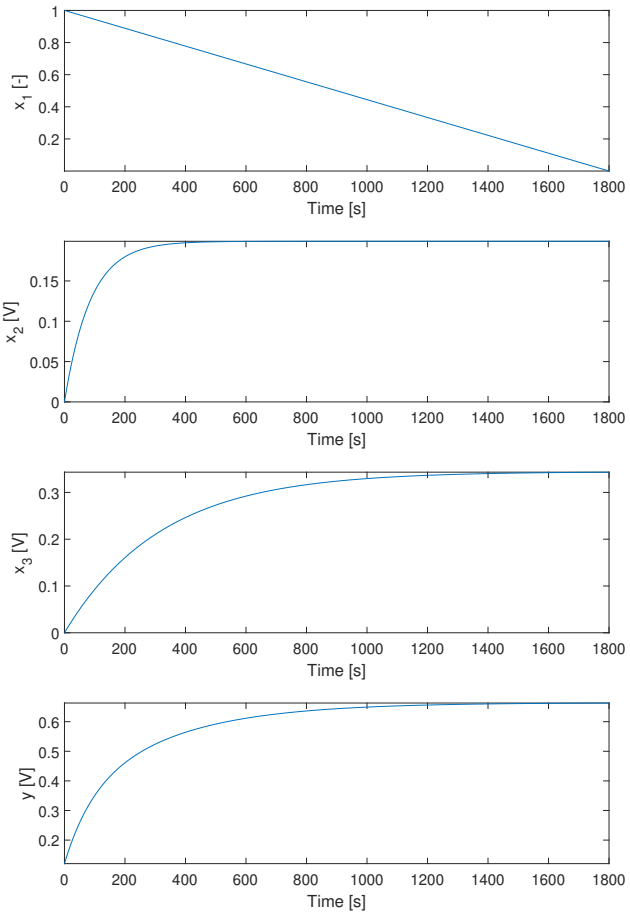

Figure 2.2: Battery system response for constant discharge current.

obtain solution of Equ. (2.1) is to use Laplace transformation. Apply Laplace transformation to Equ. (2.1), we have

$$\begin{aligned} sX(s) - x_0 &= AX(s) + BU(s) \\ sX(s) - AX(s) &= (sI - A)X(s) = x_0 + BU(s) \\ X(s) &= (sI - A)^{-1}x_0 + (sI - A)^{-1}BU(s) \\ Y(s) &= CX(s) + DU(s) = C(sI - A)^{-1}x_0 + C(sI - A)^{-1}BU(s) + DU(s) \end{aligned}$$

In other words, the solution of Equ. (2.1) in complex domain can be represented as

$$X(s) = (sI - A)^{-1}x_0 + (sI - A)^{-1}BU(s) \tag{2.6a}$$

$$Y(s) = C(sI - A)^{-1}x_0 + C(sI - A)^{-1}BU(s) + DU(s) \tag{2.6b}$$

Example 2.1.2 (Battery Simulation)**.** *Consider the battery model derived in Example 2.1.1 and Equ. (2.4). If we supply a constant u of 4.6 Ah with initial state of* $x_0 = [1, 0, 0]^T$*, then the system response is shown in Fig. 2.2. As can be seen, the dynamic of* x_2 *is faster than that of* x_3*, making the precise voltage control challanging.*

Remark 2.1.1. *The reason system (2.1) is called linear system is as follows. Suppose* $y_1(t)$ *is the system response of (2.1) when the input is* $u_1(t)$ *and the initial condition is* $x_1(0)$*. And* $y_2(t)$

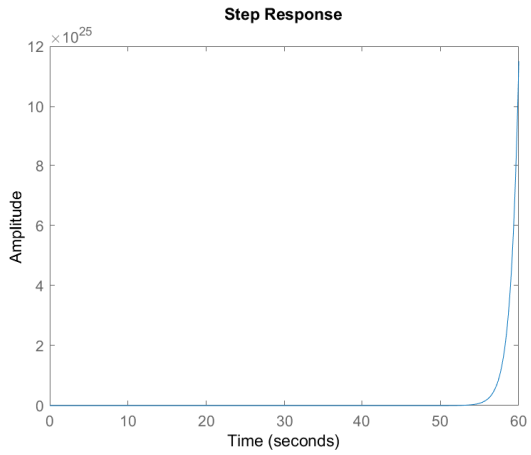

Figure 2.3: Step response for system in Example 2.1.3.

is the system response of (2.1) when the input is $u_2(t)$ and the initial condition is $x_2(0)$. Then $\alpha_1 y_1(t) + \alpha_2 y_2(t)$ is the system response of (2.1) when the input is $\alpha_1 u_1(t) + \alpha_2 u_2(t)$ and the initial condition is $\alpha_1 x_1(0) + \alpha_2 x_2(0)$. In other words, the system satisfies superposition property. A system is called nonlinear if the superposition property does not hold.

2.1.2 Stability

Definition 2.1.2 (**Bounded-input**). *An input $u(t)$ is said to be bounded if there exists an finite constant bound $\bar{u}$ such that for all t*

$$|u(t)| \leq \bar{u}.$$

Definition 2.1.3 (**BIBO Stability**). *A system is said to be bounded-input bounded-output stable (BIBO stable) if every bounded input generates a bounded output, assuming zero initial condition (i.e., $x_0 = 0$). In other words,*

$$\exists \bar{u} \ s.t. \ |u(t)| \leq \bar{u} \implies \exists \bar{y} \ s.t. \ |y(t)| \leq \bar{y}.$$

Example 2.1.3 (Unstable System). *Consider the following system*

$$\dot{x} = x + u$$
$$y = x$$

Its step response is shown in Fig. 2.3, which is unbounded even if the input is bounded (constant value of 1). Therefore, the above system is not BIBO stable.

Theorem 2.1.1. *System (2.1) is BIBO stable if and only if every pole has negative real part.*

Proof. See [3, Chapter5.2]. □

Theorem 2.1.2. *System (2.1) is BIBO stable if every eigenvalue of A has negative real part.*

Proof. See [3, Chapter5.2], where the fact that every pole of system (2.1) is an eigenvalue of A. □

Note that if A has an eigenvalue that lies in the right hand side of the complex plane, then the system may still be stable. This is because not every eigenvalue of A is a pole of system (2.1).

Example 2.1.4 (Stable System with Positive Eigenvalue)**.** *Let* $A = \begin{bmatrix} 2 & 0 \\ 0 & -3 \end{bmatrix}$, $B = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$, *and* $D = 0$. *Apparently the eigenvalues of A are 2 and -3. One can compute the transfer function of the system as*

$$sI - A = \begin{bmatrix} s-2 & 0 \\ 0 & s+3 \end{bmatrix}$$

$$(sI - A)^{-1} = \begin{bmatrix} \frac{1}{s-2} & 0 \\ 0 & \frac{1}{s+3} \end{bmatrix}$$

$$G(s) = C\left(sI - A\right)^{-1} B + D = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{s-2} & 0 \\ 0 & \frac{1}{s+3} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{s+3} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \frac{2}{s+3}$$

Therefore the system has only one pole, i.e., -3, and therefore it is BIBO stable.

Theorem 2.1.3. *If system (2.1) is both controllable and observable*[2]*, then it is BIBO stable if and only if all eigenvalues of A have negative real part.*

BIBO stability focus on the evolution of output $y(t)$, while in many situations, the evolutaion of state $x(t)$ is equally important. Next we discuss internal stability.

Definition 2.1.4 (**Asymptotic stability**)**.** *System (2.1a) is asymptotically stable if its zero-input response* $x(t)$ *for every finite initial condition* x_0 *is bounded and approaches 0 as* $t \to \infty$.

Theorem 2.1.4. *System (2.1a) is asymptotically stable if and only if all eigenvalues of A have negative real part.*

Example 2.1.5 (Unstable Systems)**.** *Let's revisit the system in Example 2.1.4. As discussed in Example 2.1.4, the system is BIBO stable. However, the eigenvalues of A are 2 and -3, and therefore the system is not asymptotically stable.*

2.1.3 Controllability and Observability

Definition 2.1.5 (**Controllability**)**.** *System (2.1), or* (A, B)*, is said to be controllable if for any initial condition* x_0 *and any final state* x_f*, there exists an input* $u(t)$ *that transfers* x_0 *to* x_f *in finite time.*

Theorem 2.1.5. *System (2.1), or* (A, B)*, is controllable if and only if the controllability matrix*

$$\mathcal{C} = \underbrace{\begin{bmatrix} B & AB & A^2B & \cdots & A^{n_x-1}B \end{bmatrix}}_{n_x \text{ blocks}} \tag{2.7}$$

has rank n_x *(i.e., full row rank).*

Example 2.1.6 (Battery Controllability)**.** *Consider the battery model derived in Example 2.1.1 and Equ. (2.4). We have* $n_x = 3$ *and*

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B \end{bmatrix} = \frac{1}{10000} \begin{bmatrix} -1.208 & 0 & 0 \\ 5.066 & -0.059 & 0.001 \\ 2.350 & -0.007 & 0 \end{bmatrix}$$

[2]See Section 2.1.3 for the definition of controllability and observability.

$$rank(\mathcal{C}) = 3 = n_x$$

Therefore, (A, B) of the battery system (2.4) is controllable.

Definition 2.1.6 (**Observability**)**.** *System (2.1), or (A, C), is said to be observable if for any unknown initial state x_0, there exists a finite time t_f such that x_0 can be determined by the knowledge of input u and output y over time frame $[0, t_f]$.*

Theorem 2.1.6. *System (2.1), or (A, C), is observable if and only if the observability matrix*

$$\mathcal{O} = \left.\begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n_x-1} \end{bmatrix}\right\} n_x \text{ blocks} \tag{2.8}$$

has rank n_x (i.e., full column rank).

Example 2.1.7 (Battery Observability)**.** *Consider the battery model derived in Example 2.1.1 and Equ. (2.4). We have $n_x = 3$ and*

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & -0.0117 & -0.0031 \\ 0 & 0.0001 & 0 \end{bmatrix}$$
$$rank(\mathcal{O}) = 2 < n_x$$

Therefore, (A, C) of the battery system (2.4) is unobservable.

The unobservability of system (2.4) is due to the fact that there is no way to infer initial SOC by observing the input and output. As an example, let's simulate system (2.4) at two different initial SOC conditions, i.e., 1 and 0.5. In both cases the input is a constant of 4.6 Ah. As shown in Fig. 2.4, though the initial conditions for SOC are different, the output are identical. Hence there is no way to infer the initial SOC by observing the input and output.

Remark 2.1.2. *One of the reasons that the SOC in Example 2.1.7 is not observable is due to the fact that the V_{oc} is not measured in this particular example. In reality, V_{oc} depends on SOC and is measured. In this case the battery ECM is an observable system.*

2.1.4 Linearization

Real systems are often nonlinear. Fortunately nonlinear system can usually be linearized to obtain the form of Equ. (2.1) as follows. Consider a nonlinear system

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.9a}$$
$$y(t) = h(x(t), u(t)), \tag{2.9b}$$

and an equilibrium nominal operating point $(\bar{u}, \bar{x}, \bar{y})$ such that

$$0 = f(\bar{x}, \bar{u})$$
$$\bar{y} = h(\bar{x}, \bar{u}).$$

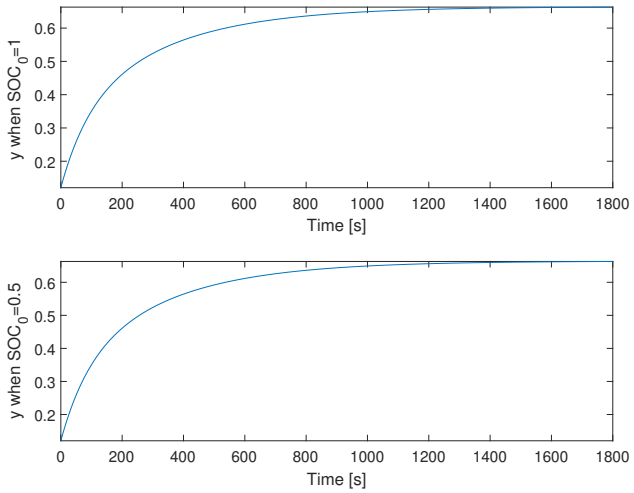

Figure 2.4: Illustration of the unobservability of the battery model model (2.4).

Define the following change of variables

$$\begin{aligned}\hat{u} &= u(t) - \bar{u}\\ \hat{x} &= x(t) - \bar{x}\\ \hat{y} &= y(t) - \bar{y}\end{aligned}$$

Then the nonlinear system (2.9) can be approximated using first order Taylor expansion, as follows.

$$\begin{aligned}\frac{d\hat{x}}{dt} = \dot{x}(t) &\approx f(\bar{x},\bar{u}) + \left.\frac{\partial f(x(t),u(t))}{\partial x(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} (x(t)-\bar{x}) + \left.\frac{\partial f(x(t),u(t))}{\partial u(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} (u(t)-\bar{u})\\ &= \left.\frac{\partial f(x(t),u(t))}{\partial x(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \hat{x} + \left.\frac{\partial f(x(t),u(t))}{\partial u(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \hat{u}\\ \hat{y} &\approx h(\bar{x},\bar{u}) + \left.\frac{\partial h(x(t),u(t))}{\partial x(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} (x(t)-\bar{x}) + \left.\frac{\partial h(x(t),u(t))}{\partial u(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} (u(t)-\bar{u}) - \bar{y}\\ &= \left.\frac{\partial h(x(t),u(t))}{\partial x(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \hat{x} + \left.\frac{\partial h(x(t),u(t))}{\partial u(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \hat{u}\end{aligned}$$

In other words, the linearized system would have the following form

$$\frac{d\hat{x}}{dt} = A\hat{x} + B\hat{u} \tag{2.10a}$$

$$\hat{y} = C\hat{x} + D\hat{u} \tag{2.10b}$$

with

$$A = \left.\frac{\partial f(x(t),u(t))}{\partial x(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \tag{2.10c}$$

$$B = \left.\frac{\partial f(x(t),u(t))}{\partial u(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \tag{2.10d}$$

$$C = \left.\frac{\partial h(x(t),u(t))}{\partial x(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \tag{2.10e}$$

$$D = \left.\frac{\partial h(x(t),u(t))}{\partial u(t)}\right|_{x(t)=\bar{x},u(t)=\bar{u}} \tag{2.10f}$$

Example 2.1.8 (Linearization)**.** *Consider the following nonlinear system*

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1,x_2,x_3,u) = x_1^2 + x_1x_2 + x_3u + 6\\
\dot{x}_2 &= f_2(x_1,x_2,x_3,u) = x_1 + x_1^2x_2 + u - 19\\
\dot{x}_3 &= f_3(x_1,x_2,x_3,u) = x_1 + x_2u + x_3 - 3\\
y &= h(x_1,x_2,x_3,u) = x_1 + x_3^3 + u
\end{aligned}$$

Given a nominal point that $\bar{u}=1$*,* $\bar{x}_1=-2$*,* $\bar{x}_2=5$*,* $\bar{x}_3=0$ *and* $\bar{y}=-1$*. Then we have*

$$\begin{aligned}
A &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2\bar{x}_1+\bar{x}_2 & \bar{x}_1 & \bar{u} \\ 1+2\bar{x}_1\bar{x}_2 & \bar{x}_1^2 & 0 \\ 1 & \bar{u} & 1 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 1 \\ -19 & 4 & 0 \\ 1 & 1 & 1 \end{bmatrix}\\
B &= \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \\ \frac{\partial f_3}{\partial u} \end{bmatrix} = \begin{bmatrix} \bar{x}_3 \\ 1 \\ \bar{x}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}\\
C &= \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} & \frac{\partial h}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3\bar{x}_3^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\\
D &= \begin{bmatrix} \frac{\partial h}{\partial u} \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}
\end{aligned}$$

Note that Equ. (2.10c) can be evaluated using finite difference, as shown in the Listings 2.4 and 2.5.

2.2 Discrete-Time Linear Time Invariant Systems

Given the continuous-time LTI system (2.1), which is copied as follows,

$$\begin{aligned}
\dot{x}(t) &= Ax(t) + Bu(t)\\
y(t) &= Cx(t) + Du(t),
\end{aligned}$$

they can be discretized to obtain a disctrete-time LTI system

$$x((k+1)T_s) = A_dx(kT_s) + B_du(kT_s) \tag{2.11a}$$
$$y(kT_s) = C_dx(kT_s) + D_du(kT_s), \tag{2.11b}$$

where T_s is the *sampling time* or *step size*. The notation of kT_s is usually simplified as k, and (2.11) can be simplified as

$$x(k+1) = A_dx(k) + B_du(k) \tag{2.12a}$$
$$y(k) = C_dx(k) + D_du(k), \tag{2.12b}$$

2.2.1 Discretization

Next, we provide two ways to discretize (2.1) into (2.12).

Exponential discretization. The exponential discretization can be derived based on the solution of continuous-time LTI system. According to Equ. (2.5), we have

$$\begin{aligned}
x(k) = x(kT_s) &= e^{AkT_s}x_0 + \int_0^{kT_s} e^{A(kT_s-\tau)}Bu(\tau)d\tau \\
x(k+1) = x((k+1)T_s) &= e^{A(k+1)T_s}x_0 + \int_0^{(k+1)T_s} e^{A((k+1)T_s-\tau)}Bu(\tau)d\tau \\
&= e^{AT_s}\left(e^{AkT_s}x_0 + \int_0^{(k+1)T_s} e^{A(kT_s-\tau)}Bu(\tau)d\tau\right) \\
&= e^{AT_s}\left(e^{AkT_s}x_0 + \int_0^{kT_s} e^{A(kT_s-\tau)}Bu(\tau)d\tau + \int_{kT_s}^{(k+1)T_s} e^{A(kT_s-\tau)}Bu(\tau)d\tau\right) \\
&= e^{AT_s}\left(e^{AkT_s}x_0 + \int_0^{kT_s} e^{A(kT_s-\tau)}Bu(\tau)d\tau\right) + \int_{kT_s}^{(k+1)T_s} e^{A((k+1)T_s-\tau)}Bu(\tau)d\tau \\
&= \underbrace{e^{AT_s}}_{A_d}x(k) + \underbrace{\left(\int_0^{T_s} e^{A\bar{\tau}}d\bar{\tau}\right)B}_{B_d}u(k)
\end{aligned}$$

Note that here we used the fact that in discrete time system, $u(t) = u(k)$ for all $kT_s \le t < (k+1)T_s$. In other words, zero-order-hold is applied to the input $u(t)$. Therefore, for exponential discretization, we have[3]

$$A_d = e^{AT_s}, \quad B_d = \left(\int_0^{T_s} e^{A\bar{\tau}}d\bar{\tau}\right)B, \quad C_d = C, \quad D_d = D \tag{2.13}$$

Forward Euler discretization. The forward Euler discretization, or Euler discretization, is much simpler compared to exponential discretization. Given a sampling time T_s, the derivation of $x(t)$ can be approximated as

$$\dot{x}(t) \approx \frac{x(t+T_s) - x(t)}{T_s} \tag{2.14}$$

Therefore, for $t = kT_s$, we have

$$\begin{aligned}
x(k+1) &\approx x(k) + \dot{x}(kT_s)T_s = x(k) + \dot{x}(t)T_s \\
&= x(k) + (Ax(t) + Bu(t))T_s = x(k) + (Ax(k) + Bu(k))T_s \\
&= \underbrace{(AT_s + I)}_{A_d}x(k) + \underbrace{BT_s}_{B_d}u(k)
\end{aligned}$$

Therefore, we have for Euler discretization,

$$A_d = AT_s + I, \quad B_d = BT_s, \quad C_d = C, \quad D_d = D \tag{2.15}$$

Remark 2.2.1. *Comparing the two discretization methods, Euler discretization is simpler to understand and implement, and requires less computation, while Exponential discretization requires more onboard computation due to the computation of matrix exponential. However, Exponential discretization often yields more accurate discrete-time representation of the original continuous-time systems.*

[3]The formula for B_d here is not directly implementatble. See [3, Chapter 4.2.1] for alternative formula that is relatively easier to implement.

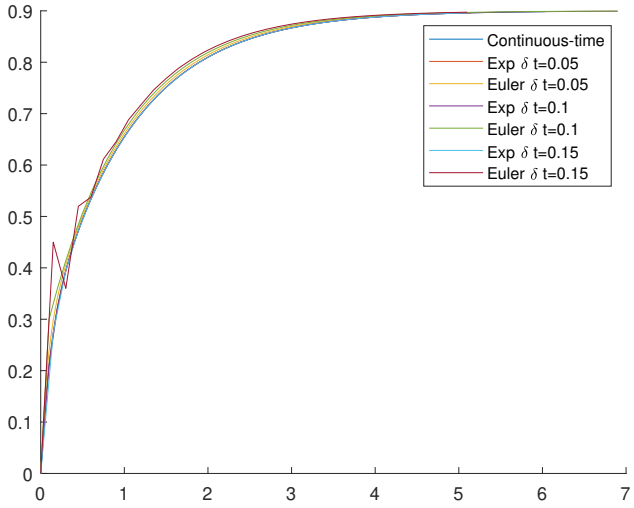

Figure 2.5: Comparison of different discretization methods.

Example 2.2.1 (Comparison of Discretization Methods)**.** *Consider the following CT LTI system*

$$\dot{x}(t) = \begin{bmatrix} -1 & 0 \\ -3 & -10 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 2 \end{bmatrix} u(t)$$

Let $T_s = 0.01$*. Its exponential discretization can be computed using Matlab command "c2d" as*

$$x(k+1) = \begin{bmatrix} 0.99 & 0 \\ -0.0284 & 0.9048 \end{bmatrix} x(k) + \begin{bmatrix} 0.01 \\ 0.0189 \end{bmatrix} u(k)$$

while it Euler discretization is given as follows

$$x(k+1) = \begin{bmatrix} 0.99 & 0 \\ -0.03 & 0.9 \end{bmatrix} x(k) + \begin{bmatrix} 0.01 \\ 0.02 \end{bmatrix} u(k).$$

The difference of them can be larger when T_s *increases. As shown in 2.5, the exponential discretization provides a fairly accurate representation of the continuous-time systems, while the Euler discretization deteriorate when sampling time increases. In fact, Euler discretization will lead to unstable system if* T_s *is too large.*

From now on, we will focus on discrete-time system. To simplify the notation, we will drop the subscript on the matrices, and drop the direct feedthrough term. In other words, we will use the following representation for DT LTI system.

$$x(k+1) = Ax(k) + Bu(k) \tag{2.16a}$$
$$y(k) = Cx(k), \tag{2.16b}$$

with initial value $x(0) = x_0$. Often times we will place time index as subscript instead of within bracket, as follows.

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k.$$

Note that dropping direct feedthrough term does not incur any loss of generality, as the direct feedthrough can be treated as a one step delay by augmenting the state space as follows.

$$\begin{aligned}\bar{x}(k+1) &= \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \bar{x}(k) + \begin{bmatrix} B \\ I \end{bmatrix} u(k) \\ \bar{y}(k) &= \begin{bmatrix} C & I \end{bmatrix} \bar{x}(k) = Cx(k) + u(k-1).\end{aligned}$$

2.2.2 System Response

Given a discrete-time LTI system (2.16), its solution can be obtained by computing (2.16a) recursively, as follows.

$$\begin{aligned}x(1) &= Ax(0) + Bu(0) \\ x(2) &= Ax(1) + Bu(1) = A\left(Ax(0) + Bu(0)\right) + Bu(1) = A^2x(0) + ABu(0) + Bu(1) \\ x(3) &= Ax(2) + Bu(2) = A^3x(0) + A^2Bu(0) + ABu(1) + Bu(2)\end{aligned}$$

Then for any $k > 0$, we have

$$x(k) = A^k x(0) + \sum_{j=0}^{k-1} A^{k-1-j} Bu(j) \tag{2.17a}$$

$$y(k) = Cx(k) = CA^k x(0) + \sum_{j=0}^{k-1} CA^{k-1-j} Bu(j) \tag{2.17b}$$

Example 2.2.2 (Response of Discrete-time Battery ECM)**.** *Consider the battery model derived in Example 2.1.1 and Equ. (2.4a). Then we have the discrete-time model given as*

$$x(k+1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 - 0.0117T_s & 0 \\ 0 & 0 & 1 - 0.0031T_s \end{bmatrix} x(k) + \begin{bmatrix} -1.208 \times 10^{-4} \\ 5.066 \times 10^{-4} \\ 2.35 \times 10^{-4} \end{bmatrix} T_s u(k) \tag{2.18}$$

When $T_s = 1$,

$$x(k+1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.9883 & 0 \\ 0 & 0 & 0.9969 \end{bmatrix} x(k) + \begin{bmatrix} -1.208 \times 10^{-4} \\ 5.066 \times 10^{-4} \\ 2.35 \times 10^{-4} \end{bmatrix} u(k)$$

Then its response can be computed by (2.17a), which is plotted in Fig. 2.6.

2.2.3 Stability

Definition 2.2.1 (**Asymptotic stability**)**.** *System (2.16a) is asymptotically stable if its zero-input response $x(k)$ for every finite initial condition x_0 is bounded and approaches 0 as $k \to \infty$.*

Theorem 2.2.1. *System (2.16a) is asymptotically stable if and only if all eigenvalues of A have magnitudes less than 1.*

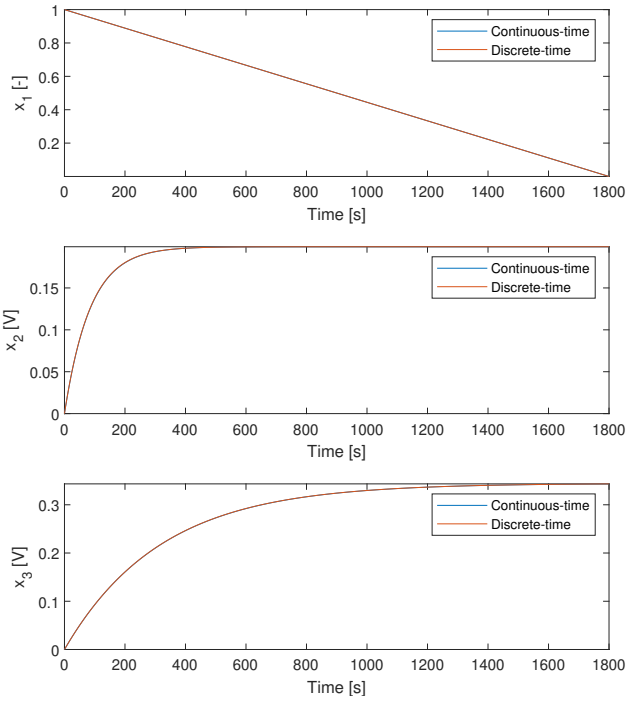

Figure 2.6: Response of the discrete-time battery model (2.18) when $T_s = 1$.

Example 2.2.3 (Stability of Discrete-time Battery ECM)**.** *Consider the discrete-time battery model derived in Example 2.2.2 and Equ. (2.18). It is clear that the eigenvalues of the A matrix are given by 1,* $1 - 0.0117T_s$*, and* $1 - 0.0031T_s$*. Therefore the sysetm is not asymptotically stable since the SOC will remain at its initial value if the external input* $u(k)$ *is 0.*

However, its second and third state is asymptotically stable provided that $T_s < 2/0.0017 = 170.94$*. When* $T_s = 171$*, the system is unstable as the eigenvalues are 1, -1.0005, and 0.4634. And it step response is unbounded, as plotted in Fig. 2.7.*

2.2.4 Controllability and Observability

The controllability and observability for discrete-time LTI system is very similar to those for continuous-time LTI systems.

Definition 2.2.2 (**Controllability**)**.** *System (2.16), or* (A, B)*, is said to be controllable if for any initial condition* x_0 *and any final state* x_f*, there exists an input* $u(k)$ *that transfers* x_0 *to* x_f *in finite time.*

Theorem 2.2.2. *System (2.16), or* (A, B)*, is controllable if and only if the controllability matrix*

$$\mathcal{C}_d = \underbrace{\begin{bmatrix} B & AB & A^2B & \cdots & A^{n_x-1}B \end{bmatrix}}_{n_x \text{ blocks}} \tag{2.19}$$

has rank n_x *(i.e., full row rank).*

Definition 2.2.3 (**Observability**)**.** *System (2.16), or* (A, C)*, is said to be observable if for any unknown initial state* x_0*, there exists a finite time step* k_f *such that* x_0 *can be determined by the knowledge of input* $u(k)$ *and output* $y(k)$ *over time frame* $k = 0, 1, \ldots, k_f$*.*

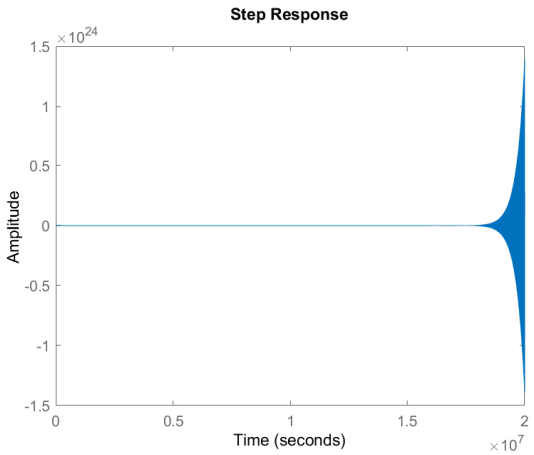

Figure 2.7: Response of the discrete-time battery model (2.18) when $T_s = 171$.

Theorem 2.2.3. *System (2.16), or (A, C), is observable if and only if the observability matrix*

$$\mathcal{O}_d = \left.\begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n_x-1} \end{bmatrix}\right\} n_x \text{ blocks} \tag{2.20}$$

has rank n_x (i.e., full column rank).

Example 2.2.4 (Controllability and Observability of DT LTI)**.** *Given the following DT LTI system*

$$x(k+1) = \begin{bmatrix} -2 & 0 \\ p & -3 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(k)$$
$$y(k) = \begin{bmatrix} 0 & 1 \end{bmatrix} x(k),$$

where p is a parameter of the system. To determine its controllability and observability, we have

$$\mathcal{C} = \begin{bmatrix} B & AB \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 0 & p \end{bmatrix}$$
$$\mathcal{O} = \begin{bmatrix} C \\ CA \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ p & -3 \end{bmatrix}$$

Therefore, when $p \neq 0$, we have $rank(\mathcal{C}) = rank(\mathcal{O}) = 2 = n_x$, so the system is both controllable and observable. On the other hand, when $p = 0$, we have $rank(\mathcal{C}) = rank(\mathcal{O}) = 1 < n_x$, so the system is uncontrollable and unobservable.

In fact, when $p = 0$, the dynamics of x_2 is decoupled from the dynamic of x_1. Since the input u can only impact x_1, it has not control authority over x_2, making the system uncontrollable. In the meanwhile, since the output only depends on x_2, there is no way one can infer the initial condition of x_2, making the system unobservable.

Example 2.2.5 (Simplified Vehicle Dynamics)**.** *Consider the following simplified lateral vehicle dynamics model, modified from [4]:*

$$\dot{v} = a \tag{2.21a}$$
$$\dot{p}_y = v \sin\phi \tag{2.21b}$$
$$\dot{\phi} = \frac{v}{l} \tan\zeta, \tag{2.21c}$$

where v is the vehicle speed, p_y is the lateral position of the vehicle, and ϕ is the yaw angle of the vehicle, all in global coordinate. l is the wheelbase, a and ζ are the two control inputs, namely vehicle acceleration and steering angle. Denote $x = \begin{bmatrix} v & p_y & \phi \end{bmatrix}^T$, $y = x$, and $u = \begin{bmatrix} a & \zeta \end{bmatrix}^T$. Given a nominal point $\bar{x} = \begin{bmatrix} \bar{v} & \bar{p}_y & 0 \end{bmatrix}^T$ and $\bar{u} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, we have the linearized model given by

$$\frac{d\hat{x}}{dt} = A\hat{x} + B\hat{u}$$

with

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ \sin\bar{\phi} & 0 & \bar{v}\cos\bar{\phi} \\ \frac{\tan\bar{\zeta}}{l} & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \bar{v} \\ 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & \frac{\bar{v}}{l\cos^2\bar{\zeta}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & \frac{\bar{v}}{l} \end{bmatrix}$$

Given a sampling time T_s, one can further discretize the linearized system to get a DT LTI system as follows.

$$\hat{x}_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \bar{v}T_s \\ 0 & 0 & 1 \end{bmatrix} \hat{x}_k + \begin{bmatrix} T_s & 0 \\ 0 & 0 \\ 0 & \frac{\bar{v}}{l}T_s \end{bmatrix} \hat{u}_k$$

Comparison of the nonlinear model, linearized CT LTI model, and discretized DT LTI model is plotted in Fig. 2.8.

2.3 Matlab Codes

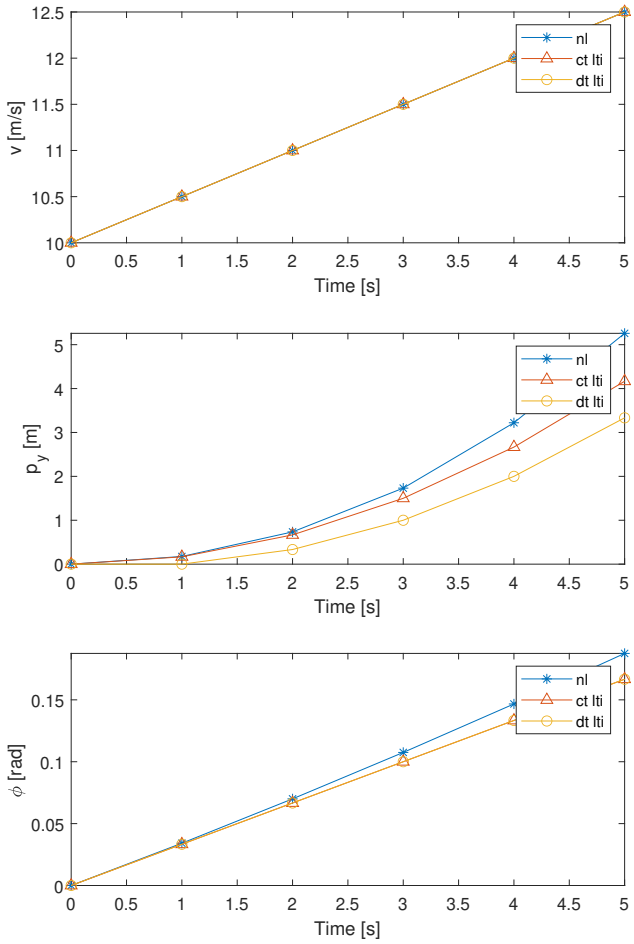

Figure 2.8: Response of the simplified lateral vehicle dynamic model in Example 2.2.5 when $\bar{v} = 10$, $\bar{p}_y = 0$, $T_s = 1$, and $u = [0.5, 0.01]^T$.

```
clear all; clc; close all;

Cb = 2.3;
Ro = 0.0262;
R1 = 0.0433;
C1 = 1974.1;
R2 = 0.0749;
C2 = 4254.7;

A = [0 0 0; 0 -1/C1/R1 0; 0 0 -1/C2/R2];
B = [-1/3600/Cb;1/C1;1/C2];
C = [0 1 1];
D = [Ro];

sys = ss(A,B,C,D);
x0 = [1;0;0];

t = 0:1800;
u = 2*Cb*ones(size(t));

[Y,T,X] = lsim(sys,u,t,x0);

f=figure;

pos = get(f, 'Position');
pos(2)=pos(2)/10;
pos(4) = pos(4)*2;
f.Position = pos;

subplot(4,1,1), plot(T,X(:,1)),
xlabel('Time [s]'), ylabel('x_1 [-]'), axis tight
subplot(4,1,2), plot(T,X(:,2)),
xlabel('Time [s]'), ylabel('x_2 [V]'), axis tight
subplot(4,1,3), plot(T,X(:,3)),
xlabel('Time [s]'), ylabel('x_3 [V]'), axis tight
subplot(4,1,4), plot(T,Y),
xlabel('Time [s]'), ylabel('y [V]'), axis tight
```

Listing 2.1: MATLAB Commands for Example 2.1.2.

```
>> A=[2  0;0  -3]; B=[1;2]; C=[0  1]; D=0; sys=ss(A,B,C,D)

sys =

  A =
       x1   x2
   x1   2    0
   x2   0   -3

  B =
       u1
   x1   1
   x2   2

  C =
       x1   x2
   y1   0    1

  D =
       u1
   y1   0

Continuous-time state-space model.

>> zpk(sys)

ans =

     2
  -------
  (s+3)
```

Listing 2.2: MATLAB Commands for Example 2.1.4.

```
clear all; clc; close all;

Cb = 2.3;
Ro = 0.0262;
R1 = 0.0433;
C1 = 1974.1;
R2 = 0.0749;
C2 = 4254.7;

A = [0 0 0; 0 -1/C1/R1 0; 0 0 -1/C2/R2];
B = [-1/3600/Cb;1/C1;1/C2];
C = [0 1 1];
D = [Ro];

sys = ss(A,B,C,D);

>> rank(ctrb(A,B))

ans =

     3

>> rank(obsv(A,C))

ans =

     2

t = 0:1800;
u = 2*Cb*ones(size(t));

[Y1,T1,~] = lsim(sys,u,t,[1;0;0]);

[Y2,T2,~] = lsim(sys,u,t,[0.5;0;0]);

f=figure;

subplot(2,1,1), plot(T1,Y1),
xlabel('Time [s]'), ylabel('y when SOC_0=1'), axis tight
subplot(2,1,2), plot(T2,Y2),
xlabel('Time [s]'), ylabel('y when SOC_0=0.5'), axis tight
```

Listing 2.3: MATLAB Commands for Examples 2.1.6 and 2.1.7

```
clear; close all; clc

x0=[-2;5;0];
u0=1;
delta = 1e-6;
f0 = f(x0,u0)
y0 = h(x0,u0)
A = zeros(3,3);
C = zeros(1,3);

for n = 1 : 3
    xd = x0;
    xd(n) = xd(n) + delta;
    f1 = f(xd,u0);
    A(:,n) = (f1-f0)/delta;
    h1 = h(xd,u0);
    C(n) = (h1-y0)/delta;
end

B = (f(x0,u0+delta)-f0)/delta;
D = (h(x0,u0+delta)-y0)/delta;

function xdot = f(x,u)
xdot = zeros(size(x));
xdot(1) = x(1)^2+x(1)*x(2)+x(3)*u+6;
xdot(2) = x(1)+x(1)^2*x(2)+u-19;
xdot(3) = x(1)+x(2)*u+x(3)-3;
end

function y = h(x,u)
y = x(1)+x(3)^3+u;
end
```

Listing 2.4: MATLAB Commands for Example 2.1.8 - Part 1.

```
>> f0

f0 =

     0
     0
     0

>> y0

y0 =

    -1

>> A

A =

    1.0000   -2.0000    1.0000
  -19.0000    4.0000         0
    1.0000    1.0000    1.0000

>> B

B =

         0
    1.0000
    5.0000

>> C

C =

    1.0000         0         0

>> D

D =

    1.0000
```

Listing 2.5: MATLAB Commands for Example 2.1.8 - Part 2.

```
clear all; clc; close all;

A = [-1  0; -3  -10];
B = [1;2];
C = [1  1];
D = 0;

sys = ss(A,B,C,D);
x0 = [1;0];

Ts = 0.01;

[A1,B1] = c2d(A,B,Ts) % Exponential discretization
A2 = A*Ts+eye(2) % Euler discretization
B2 = B*Ts % Euler discretization

A1 =

    0.9900         0
   -0.0284    0.9048


B1 =

    0.0100
    0.0189


A2 =

    0.9900         0
   -0.0300    0.9000


B2 =

    0.0100
    0.0200
```

Listing 2.6: MATLAB Commands for Examples 2.2.1.

```
clear all; clc; close all;

Cb = 2.3;
Ro = 0.0262;
R1 = 0.0433;
C1 = 1974.1;
R2 = 0.0749;
C2 = 4254.7;

Ts = 1;
A = [0 0 0; 0 -1/C1/R1 0; 0 0 -1/C2/R2];
Ad = A*Ts+eye(3);
B = [-1/3600/Cb;1/C1;1/C2];
Bd = B*Ts;
C = [0 1 1];
D = [Ro];

sys = ss(A,B,C,D);
x0 = [1;0;0];

t = 0:1800;
u = 2*Cb*ones(size(t));

[Y,T,X] = lsim(sys,u,t,x0);

Xd = zeros(size(X));
Xd(1,:) = X(1,:);
for t = 1 : 1800
    Xd(t+1,:) = (Ad^t*x0)';
    for j = 1 : t
        Xd(t+1,:) = Xd(t+1,:) + (Ad^(t-j)*Bd*u(j))';
    end
end

f=figure;
pos = get(f, 'Position'); pos(2)=pos(2)/10; pos(4) = pos(4)*1.5;
f.Position = pos;

subplot(3,1,1), plot(T,X(:,1),T,Xd(:,1)), xlabel('Time [s]'),
ylabel('x_1 [-]'), axis tight, legend('Continuous-time','Discrete-time')
subplot(3,1,2), plot(T,X(:,2),T,Xd(:,2)), xlabel('Time [s]'),
ylabel('x_2 [V]'), axis tight, legend('Continuous-time','Discrete-time')
subplot(3,1,3), plot(T,X(:,3),T,Xd(:,3)), xlabel('Time [s]'),
ylabel('x_3 [V]'), axis tight, legend('Continuous-time','Discrete-time')
```

Listing 2.7: MATLAB Commands for Example 2.2.2.

```
clear all; clc; close all;

% Voc = 3.2;
Cb = 2.3;
Ro = 0.0262;
R1 = 0.0433;
C1 = 1974.1;
R2 = 0.0749;
C2 = 4254.7;

Ts = 1;
A = [0 0 0; 0 -1/C1/R1 0; 0 0 -1/C2/R2];
Ad = A*Ts+eye(3);
B = [-1/3600/Cb;1/C1;1/C2];
Bd = B*Ts;
C = [0 1 1];
D = [Ro];

eig(Ad)

ans =

    0.9883
    0.9969
    1.0000

Ts = 171;
Ad = A*Ts+eye(3);
Bd = B*Ts;

eig(Ad)

ans =

   -1.0005
    0.4634
    1.0000
```

Listing 2.8: MATLAB Commands for Example 2.2.3.

```
clear all; clc; close all;

p = 1;
A=[-2  0;p  -3];
B = [1;0];
C = [0,1];

>> rank(ctrb(A,B))
ans =
     2

>> rank(obsv(A,C))
ans =
     2

p = 0;
A=[-2  0;p  -3];

>> rank(ctrb(A,B))
ans =
     1

>> rank(obsv(A,C))
ans =
     1
```

Listing 2.9: MATLAB Commands for Example 2.2.4.

```
close all; clear variables; clc

l = 3;
xbar = [10;0;0]; ubar = [0;0];
x0 = xbar;
Ts = 1; tspan = 0:Ts:5;
u = [0.5;0.01];
N = numel(tspan);

[tn,xn] = ode45(@(t,x)lvd(t,x,u,l), tspan, x0);

A = [0 0 0;0 0 xbar(1);0 0 0];
B = [1 0;0 0;0 xbar(1)/l];
C = eye(3);
lsys = ss(A,B,C,0);
[yl,tl,xl] = lsim(lsys,repmat(u,1,N),tspan,x0);

Ad = [1 0 0;0 1 xbar(1)*Ts;0 0 1];
Bd = [Ts 0;0 0;0 xbar(1)/l*Ts];
td = tl; xd = zeros(size(xl));
for n = 2:N
    xd(n,:) = (Ad*xd(n-1,:)'+Bd*u)';
end
xd = xd + repmat(x0',N,1);

f=figure;
pos = get(f, 'Position'); pos(2)=pos(2)/10; pos(4) = pos(4)*2;
f.Position = pos;

subplot(3,1,1), plot(tn,xn(:,1),'*-',tl,xl(:,1),'^-',td,xd(:,1),'o-'),
xlabel('Time [s]'), ylabel('v [m/s]'), axis tight,
legend('nl','ct lti','dt lti')
subplot(3,1,2), plot(tn,xn(:,2),'*-',tl,xl(:,2),'^-',td,xd(:,2),'o-'),
xlabel('Time [s]'), ylabel('p_y [m]'), axis tight,
legend('nl','ct lti','dt lti')
subplot(3,1,3), plot(tn,xn(:,3),'*-',tl,xl(:,3),'^-',td,xd(:,3),'o-'),
xlabel('Time [s]'), ylabel('\phi [rad]'), axis tight,
legend('nl','ct lti','dt lti')

function xdot = lvd(t,x,u,l)
xdot = zeros(3,1);
xdot(1) = u(1);
xdot(2) = x(1)*sin(x(3));
xdot(3) = x(1)/l*tan(u(2));
end
```

Listing 2.10: MATLAB Commands for Example 2.2.5.

Chapter 3

Numerical Optimization

We start this chapter by discussing the general mathematical formulation of optimization problems. We will discuss how we can formulate real-time control problem into optimizations problem.

3.1 Optimization Problems

This section discuss the general mathematical formulation of optimization problems, which is the minimization or maximization of certain objective function subject to constraints. Without loss of generality, we focus on the case of minimization. Note that the maximization problem can be converted into minimization problem by flipping the sign of the objective function.

3.1.1 Mathematical Formulation

The mathematical formulation of optimization problem is given as follows [5]

$$\min_{x} \quad f(x) \qquad \text{subject to } \begin{cases} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \leq 0, & i \in \mathcal{I}, \end{cases} \tag{3.1}$$

where

- $x \in \mathbb{R}^n$ is the optimization variables;
- $f(x)$ is the objective function, or cost function, that maps x to a scalar; and
- c_i are constraint functions, with $\mathcal{E}$ being the index for equality constraints and $\mathcal{I}$ the index for inequality constraints.

Given an optimization problem (3.1), the set of points for x that satisfy the constraints are called *feasible region.*

Example 3.1.1 (Minimization Problem)**.** *Consider the problem*

$$\min_{x_1,x_2} \quad (x_1-3)^2 + 2(x_2-2)^2 \qquad \textit{subject to } \begin{cases} x_1^2 - x_2 \leq 0, \\ x_1 + x_2 \leq 1. \end{cases} \tag{3.2}$$

Then in this case

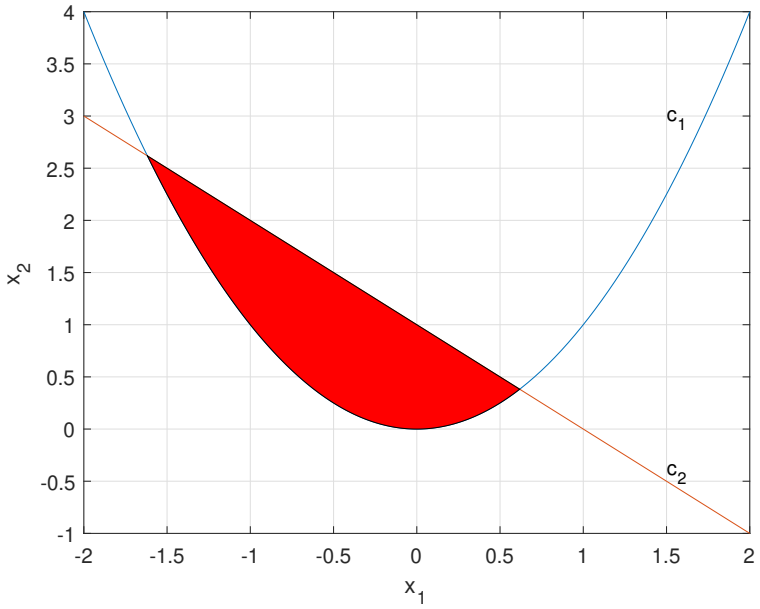

Figure 3.1: Feasible region for Example 3.1.1.

- $x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \in \mathbb{R}^2$;
- $f(x) = (x_1 - 3)^2 + 2(x_2 - 2)^2$; *and*
- $c_1(x) = x_1^2 - x_2$ *and* $c_2(x) = x_1 + x_2 - 1$, *or simply* $c(x) = \begin{bmatrix} x_1^2 - x_2 \\ x_1 + x_2 - 1 \end{bmatrix} \in \mathbb{R}^2$, *with* $\mathcal{I} = \{1, 2\}$ *and* $\mathcal{E} = \emptyset$.

Furthermore, the feasible region is illustrated in Fig. 3.1.

Example 3.1.2 (Parameter Estimation)**.** *Consider the simplified vehicle dynamic model discussed in Example 2.2.5, where the DT LTI model is given as follows.*

$$x_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \bar{v}T_s \\ 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} T_s & 0 \\ 0 & 0 \\ 0 & \frac{\bar{v}}{l}T_s \end{bmatrix} u_k = Ax_k + B(l)u_k \tag{3.3}$$

Now, suppose that the wheelbase l is unknown. To estimate l, we can collect data using constant input $u = [0.5; 0.01]$. Denote the collected data as $\bar{x}_k$, $k = 1, \ldots, K$, as shown in Fig. 3.2. Then the parameter l can be estimated by solving the following optimization problem.

$$\min_{l, x_k} \sum_{k=1}^{K} (x_k - \bar{x}_k)^2 \qquad \textit{subject to} \quad \begin{cases} x_{k+1} = Ax_k + B(l)u, k = 1, \ldots, K-1 \\ 1 \le l \le 4 \end{cases}$$

where we enforce bound constraints on the parameter l. Using the code in Listing 3.1, one can solve the above optimization problem, which gave $\hat{l} = 3.0151$ in this example, which can be validated by integrating the model (3.3), as shown in Fig. 3.2 also.

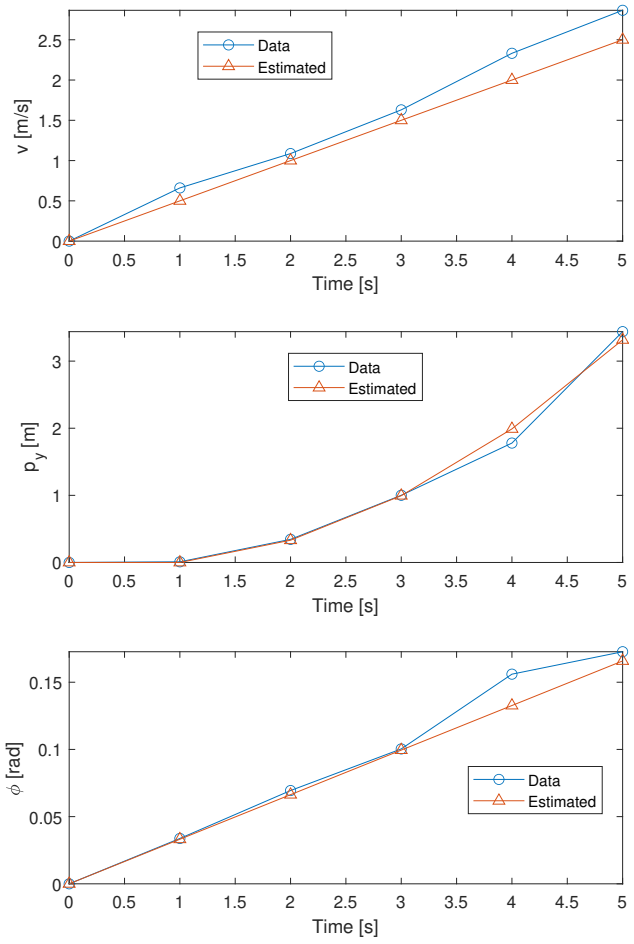

Figure 3.2: Comparison of data and model response using estimated wheelbase.

Example 3.1.3 (Steering Control)**.** *Consider the simplified vehicle dynamic model discussed in Example 3.1.2. With $l = 3$, $\bar{v} = 10$ and $T_s = 0.2$, the DT LTI model is given as follows.*

$$x_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0.2 & 0 \\ 0 & 0 \\ 0 & \frac{2}{3} \end{bmatrix} u_k = Ax_k + Bu_k$$

Now suppose the initial conditions are specified as $v_0 = 0$, $p_{y,0} = -2$, and $\phi_0 = -0.2$.

Now, suppose that the vehicle speed is satisfactory, and we want to find a sequence of steering command so that the vehicle can be steered back to $p_y = 0$ and $\phi = 0$ within 5 seconds, i.e., in $K = 25$ time steps. This can be formulated as following optimization problems.

$$\min_{u_k, x_k} \quad \sum_{k=1}^{K} \|x_k\|_Q^2 \qquad \textit{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad k = 1, \ldots, K-1 \\ u_k(1) = 0, \quad k = 1, \ldots, K-1 \\ u_l \le u_k(2) \le u_u, \quad k = 1, \ldots, K-1 \end{cases}$$

Here Q is the weight matrix. Note that the second constraint is essentially requesting the longitudinal acceleration to be 0. By using the Matlab command "ga", an implementation of Genetic Algorithm, a sequence of steering input can be found, as shown in Figs. 3.3 and 3.4, where the first plot shows results with bound constraints of $[-0.2, 0.2]$ while the second plot is with tighter constraints of $[-0.1, 0.1]$. It is obvious that with tighter constraint the vehicle takes longer time to return back to its equilibrium point.

3.1.2 Types of Optimization Problems

Optimization problems can be classified according to the optimization variables, objective function, and/or constraint functions.

Continuous v.s. discrete optimization. When the feasible region of the optimization variables x is a finite set, then the optimization problem is often called *discrete optimization* problem. Otherwise when the feasible set is uncountably infinite, then the problem is referred to as *continuous optimization* problem. For example, in autonomous driving, the path planning algorithm needs to decide among "keep in current lane", "change to left lane", "change to right lane", etc. In this case the feasible set is discrete and the problem is a discrete optimization problem. On the other hand, if it has been decided to "change to left lane", then the calculation of optimal steering angle belongs to continuous optimization problem since the feasible region in this case is uncountably infinite.

In many scenarios, the optimization variables x has to take on integer values. In this case, the problem is referred to as *integer programming* problems, which is a special case of discrete optimization problem. In addition, if only some elements of x are restricted to integers, while the remaining can be continuous, then it is often referred to as *mixed integer programming* problem. An example of the integer programming is the determination of gear shift to optimize fuel efficiency and ride comfort. If we further consider throttle position, meaning, co-optimization of transmission and throttle, then the problem becomes mixed integer programming.

Constrained v.s. unconstrained optimization. Another way to classify the optimization problem (3.1) is based on the objective function and constraints. When there is no constraint, i.e., $\mathcal{E} = \mathcal{I} = \emptyset$, the problem is called *unconstrained optimization* problems. For example, the following

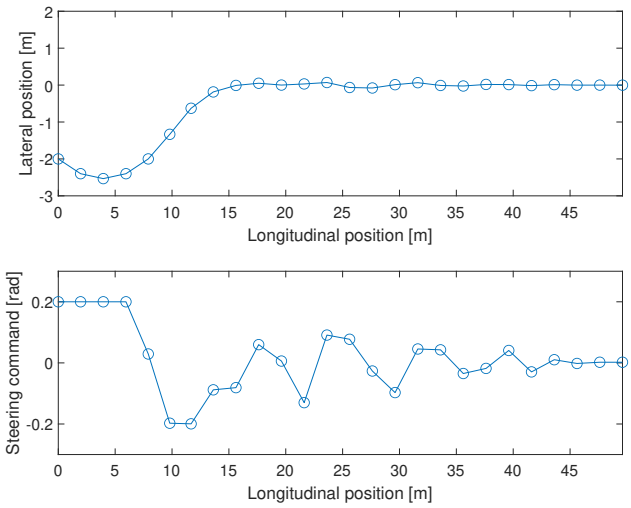

Figure 3.3: Optimal solution for Example 3.1.3.

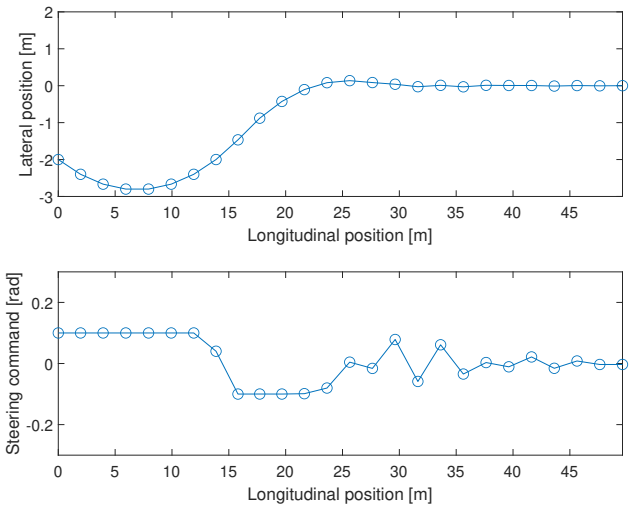

Figure 3.4: Optimal solution, with tighter constraints, for Example 3.1.3.

is an unconstrained optimization problem.

$$\min_{x_1,x_2} \quad (x_1-3)^2+2(x_2-2)^2,$$

whose solution is $x_1^*=3$ and $x_2^*=2$. Unconstrained optimization is very important and can be useful in a lot of applications. For example, in normal driving condition, the speed cruise control unit may not need to constrain the acceleration a since it is regulated in the objective function as $\beta_1 a^2+\beta_2 \dot{a}^2$.

On the other hand, the *constrained optimization* problem is such that at least one constraint is present in (3.1). Problem (3.2) is one example of constrained constrained optimization problem. In the above speed cruise control example, if abnormal driving condition is also considered such that abrupt braking may be possible, one may want to introduce upper bound and lower bound constraints on the acceleration such that $a_l \le a \le a_u$.

In many research, constrained optimization problem can be converted to unconstrained one to utilize the rich methodologies developed for unconstrained optimization problem. For example, problem (3.2) can be equivalently presented as

$$\min_{x_1,x_2} \quad (x_1-3)^2+2(x_2-2)^2+f_1(x_1,x_2)+f_2(x_1,x_2),$$

where

$$f_1(x_1,x_2)=\begin{cases}0 & \text{if } x_1^2-x_2\le 0,\\ \infty & \text{otherwise.}\end{cases} \qquad \text{and} \qquad f_2(x_1,x_2)=\begin{cases}0 & \text{if } x_1+x_2-1\le 0,\\ \infty & \text{otherwise.}\end{cases}$$

Linear v.s. nonlinear programming. When the objective function and all constraints (if any) are linear, then the problem is referred to as *linear programming*. On the other hand, when at least one of the constraints or the objective function is nonlinear, then the problem is referred to as *nonlinear programming*. Problem (3.2) is a nonlinear programming, while the following is an example of linear programming:

$$\min_{x_1,x_2} \quad (x_1-3)+2(x_2-2) \qquad \text{subject to } \begin{cases}x_1-x_2\le 0,\\ -x_1-x_2\le 1.\end{cases}$$

Stochastic v.s. deterministic optimization. In some scenarios, there is randomness in either the objective function, or the constraints. In this case, the problem is considered to be *stochastic optimization*. For example, the following is a stochastic optimization problem.

$$\min_{x_1,x_2} \quad (x_1-3)^2+2(x_2-2)^2+\sigma x_1^4 \qquad \text{subject to } x_1+\sigma\le 0$$

where σ is a random variable with standard normal distribution. Different approaches can be used to solve stochastic optimization problem. One is to find the solution x^* so that the expected objective function is minimized and the constraints are satisfied in an expected sense. Another way is called *chance-constrained optimization*, in which the above constraint can be replaced with

$$Pr(x_1+\sigma\le 0)>\mu$$

where μ is a predefined lower bound. In other words, we ensure that the constraints are satisfied with certain probability. Finally, the *robust optimization* requires that the constraints are satisfied all the time, even for the worst case scenario. For the above problem, there is no such solution.

Stochastic optimization arises in may real world applications. For example, in automated driving, the force provided by the road surface is high stochastic, depending on the type of the road as well as the dry/wet condition. All these make the vehicle dynamic model inherently a stochastic model.

Example 3.1.4 (Stochastic Steering Control). *Let's revisit Example 3.1.3, but this time we consider randomness in steering actuation. The optimization problem is formulated as follows.*

$$\min_{u_k,x_k} \quad \sum_{k=1}^{K} \|x_k\|_Q^2 \qquad \textit{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad k = 1,\dots,K-1 \\ u_k(1) = 0, \quad k = 1,\dots,K-1 \\ u_l \le u_k(2) + \sigma \le u_u, \quad k = 1,\dots,K-1 \end{cases}$$

Here σ denotes a random variable with uniform distribution from $[-0.05, 0.05]$. In other words, the steering component is not very robust and may actually randomly deviate from the commanded angle. In this case, if one decides to solve above optimization problem such that the expected constraint functions are satisfied, then one would use the exact approach as presented in Example 3.1.3. If one decides to solve this problem using robust optimization, i.e., to ensure that all constraints are satisfied at all times, then one needs to solve the following problem with tighter bound.

$$\min_{u_k,x_k} \quad \sum_{k=1}^{K} \|x_k\|_Q^2 \qquad \textit{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad k = 1,\dots,K-1 \\ u_k(1) = 0, \quad k = 1,\dots,K-1 \\ u_l + 0.05 \le u_k(2) \le u_u - 0.05, \quad k = 1,\dots,K-1 \end{cases}$$

The results are shown in Figs. 3.5 and 3.6. It can be observed that, if one choose to deal with the expected constraint satisfaction, then the system responds faster. However, there is a chance that the systems is actually perform worst and constraints are violated. On the other hand, for robust optimization, the system is always slower due to tighter constraints. However, the system is more robust against noise and the constraints are always satisfied.

3.2 Optimal Solution and Optimality Conditions

3.2.1 Mathematical Definition

We first define the global and local optimizer of a function.

Definition 3.2.1 (Global and Local Optimizer). *A point x^* is a global minimizer of $f(x)$ if $f(x^*) \le f(x)$ for all feasible x. A point x^* is a local minimizer of $f(x)$ if there is a feasible neighbor $\mathcal{N}$ such that $f(x^*) \le f(x)$ for all $x \in \mathcal{N}$.*

A convenient notation for global minimizer is denoted as follows

$$x^* = \arg\min_x f(x)$$

Example 3.2.1 (Global and Local Optimizer). *Consider the following function*

$$f(x) = x^2 \cos\left(\frac{1}{x}\right) + 2x^2,$$

which is illustrated in Fig. 3.7. As can be seen $x^* = 0$ is the global minimizer of $f(x)$, while the function has infinitely many local minimizer.

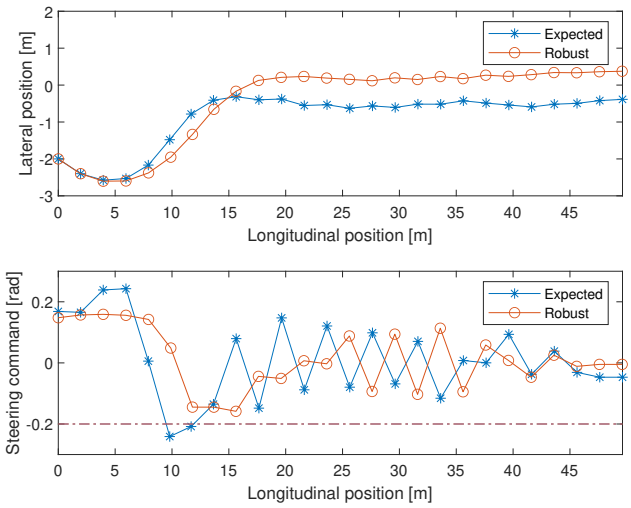

Figure 3.5: Comparison with expected and robust constraint enforcement for Example 3.1.4.

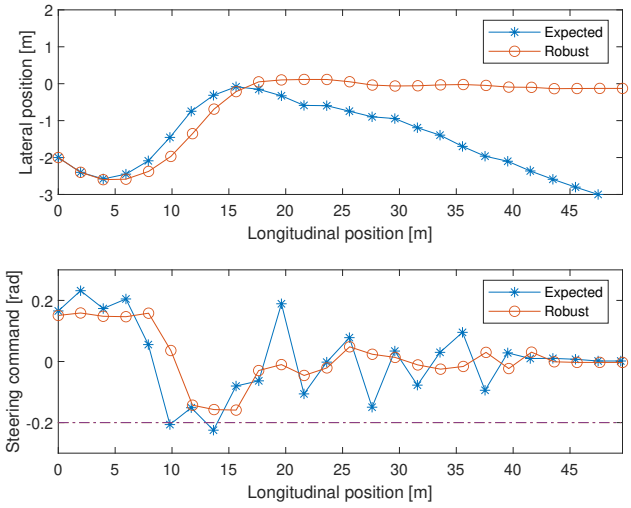

Figure 3.6: Comparison with expected and robust constraint enforcement for Example 3.1.4.

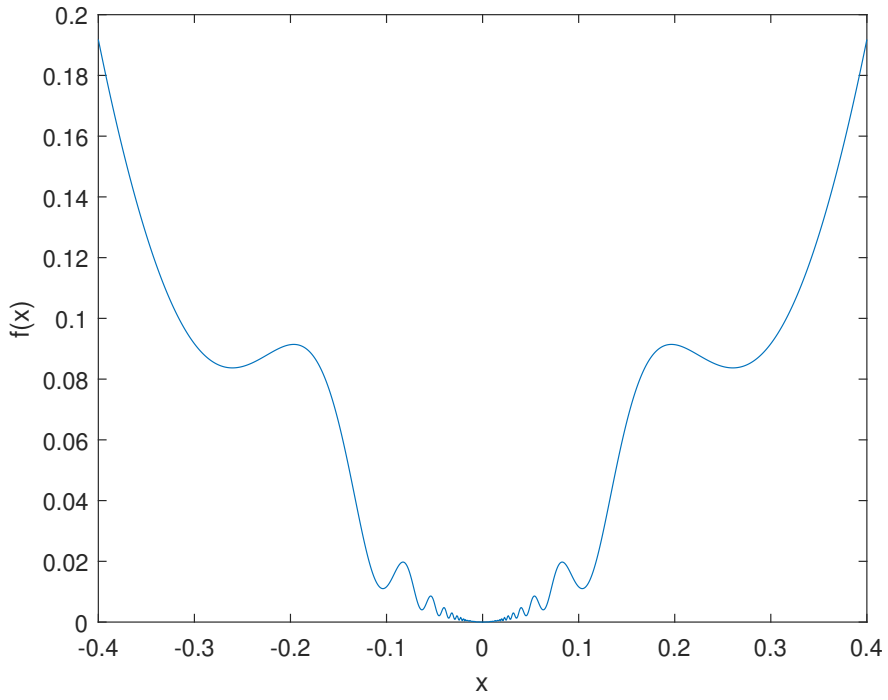

Figure 3.7: Illustration of global and local minimizer.

Given the constrained optimization problem (3.1), x^* is said to be the solution to (3.1) if x^* satisfies constraints and is a global minimizer of $f(x)$ over feasible range. In other words, x^* satisfies

$$c_i(x^*) = 0, \quad i \in \mathcal{E}$$
$$c_i(x^*) \leq 0, \quad i \in \mathcal{I}$$
$$f(x^*) \leq f(x) \text{ for all } x \text{ such that } \begin{cases} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \leq 0, & i \in \mathcal{I}. \end{cases}$$

So far, we have been focusing on the definition of the optimal solution. The remaining of this section will focus on deriving conditions for optimal solution to constrained optimization problem. We start by walking through several examples.

3.2.2 Numerical Examples

Example 3.2.2 (Scalar Unconstrained Optimization)**.** *Consider the following optimization problem.*

$$\min_x f(x) = (x-2)^2 \tag{3.4}$$

Then it is obvious that $x^* = 2$ is a solution to (3.4). To demonstrate this, one only needs to show that $f(2) \leq f(x)$ for all $x \in \mathbb{R}$. Given any arbitrary $x \in \mathbb{R}$, we have

$$f(x) = (x-2)^2 \geq 0 = f(2).$$

Therefore $x^* = 2$ is the optimal solution of (3.4). Furthermore, one can also show that at the optimal solution $x^* = 2$, the following is satisfied.

$$\frac{df(x)}{dx} \mid_{x=x^*} = 2(x-2) \mid_{x=x^*} = 0.$$

In other words, at the optimal solution, the gradient of the objective function is 0 in this example.

Example 3.2.3 (Multi-variable Unconstrained Optimization)**.** *Consider the following optimization problem.*

$$\min_{x_1,x_2} f(x_1,x_2) = (x_1-2)^2 + (x_2-3)^2 \tag{3.5}$$

Then it is obvious that $x^* = \begin{bmatrix}2 & 3\end{bmatrix}^T$ *is a solution to (3.5). To demonstrate this, one only needs to show that* $f(x^*) \leq f(x)$ *for all* $x \in \mathbb{R}^2$*. Given any arbitrary* $x \in \mathbb{R}^2$*, we have*

$$f(x) = (x_1-2)^2 + (x_2-3)^2 \geq 0 = f(x^*).$$

Therefore $x^* = \begin{bmatrix}2 & 3\end{bmatrix}^T$ *is the optimal solution of (3.5). Furthermore, one can also show that at the optimal solution* $x^* = \begin{bmatrix}2 & 3\end{bmatrix}^T$*, the following is satisfied.*

$$\frac{\partial f(x)}{\partial x_1} \mid_{x=x^*} = 2(x_1-2) \mid_{x=x^*} = 0$$
$$\frac{\partial f(x)}{\partial x_2} \mid_{x=x^*} = 2(x_2-3) \mid_{x=x^*} = 0$$

or equivalently

$$\nabla f(x^*) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \mid_{x=x^*} \\ \frac{\partial f(x)}{\partial x_2} \mid_{x=x^*} \end{bmatrix} = 0$$

In previous two examples, the gradient of the objective function at the optimal solution is 0. Let's now look at constrained case. Let's revisit (3.4).

Example 3.2.4 (Scalar Constrained Optimization)**.** *Consider the following optimization problem.*

$$\min_x f(x) = (x-2)^2 \qquad \textit{subject to} \quad c(x) = x+1 \leq 0 \tag{3.6}$$

Then it is obvious that $x^* = -1$ *is a solution to (3.6), as illustrated in Fig. 3.8. Looking closely at Fig. 3.8, at* $x = -2$*, the gradient of the objective function is point to the left, meaning one can reduce the value of objective function by increasing* x*. Meanwhile, the gradient of the constraint is pointing to the right, meaning increasing* x *will also increase the value of the constraint function* $c(x) = x+1$*. Since the point* $x = -2$ *is well within the feasible region, this is fine. Now look at the optimal solution* $x^* = -1$*, where the gradient of the objective function is point to the left, meaning one can reduce the value of objective function by moving to the right. However, this will increase the value of the constraint function* $c(x) = x+1$*. Since the point* $x^* = -1$ *is already on the boundary of the feasible region, this will violate the constraint.*

In this case, the gradient of the objective function at the optimal solution x^* *is not 0. In other words,*

$$\frac{df(x)}{dx} \mid_{x=x^*} = 2(-1-2) \mid_{x=x^*} = -6 \neq 0.$$

To have similar property, introduce the Lagrangian function

$$\mathcal{L}(x,\lambda) = f(x) + \lambda c(x) = (x-2)^2 + \lambda(x+1)$$

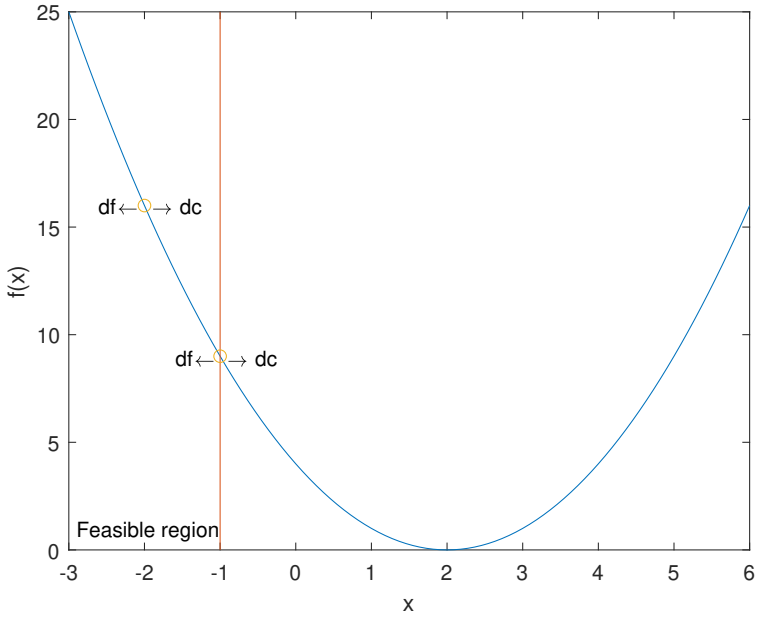

Figure 3.8: Illustration of problem (3.6).

Note that we have $\nabla_x \mathcal{L}(x,\lambda) = \nabla f(x) + \lambda \nabla c(x)$. *Then there exists a scalar* $\lambda^* = 6$ *such that*

$$\nabla_x \mathcal{L}(x^*,\lambda^*) = \nabla f(x^*) + \lambda^* \nabla c(x^*) = 2(x^*-2) + \lambda^* = 2(-1-2) + 6 = 0$$

Now let's look at the following problem.

$$\min_x f(x) = (x-2)^2 \qquad \textit{subject to} \quad c(x) = x - 5 \le 0 \tag{3.7}$$

Compare to problem (3.6), the difference here is the constraint function $c(x)$. *Then it is obvious that* $x^* = 2$ *is a solution to (3.7), as illustrated in Fig. 3.9.*

Looking closely at Fig. 3.9, at boundary point $x = 5$, *the gradient of the objective function and the gradient of the constraints both point to the right, meaning one can decrease the objective function value without violating the constraints. In fact, the optimal solution is at* $x^* = 2$, *at which the gradient of the objective function is 0. In other words, there again exists* $\lambda^* = 0$ *such that*

$$\nabla_x \mathcal{L}(x^*,\lambda^*) = \nabla f(x^*) + \lambda^* \nabla c(x^*) = 2(x^*-2) = 0.$$

Now let's revisit the optimization problem (3.5).

Example 3.2.5 (Multi-variable Constrained Optimization)**.** *Consider the following optimization problem.*

$$\min_{x_1,x_2} f(x_1,x_2) = (x_1-2)^2 + (x_2-3)^2 \qquad \textit{Subject to} \quad \begin{cases} x_1 + 1 \le 0 \\ x_2 - 5 \le 0 \end{cases} \tag{3.8}$$

Then it is obvious that $x^* = \begin{bmatrix} -1 & 3 \end{bmatrix}^T$ *is a solution to (3.8). Similar to the scalar constrained optimization case, here the gradient of objective function at optimal solution is not zero. In particular* $\frac{\partial f(x)}{\partial x_1} \mid_{x=x^*} = 2(x_1^* - 2) = -6 \neq 0$. *However, we can introduce the Lagrangian function similar to the scalar case as*

$$\mathcal{L}(x,\lambda) = f(x) + \lambda^T c(x) = (x_1-2)^2 + (x_2-3)^2 + \lambda_1(x_1+1) + \lambda_2(x_2-5),$$

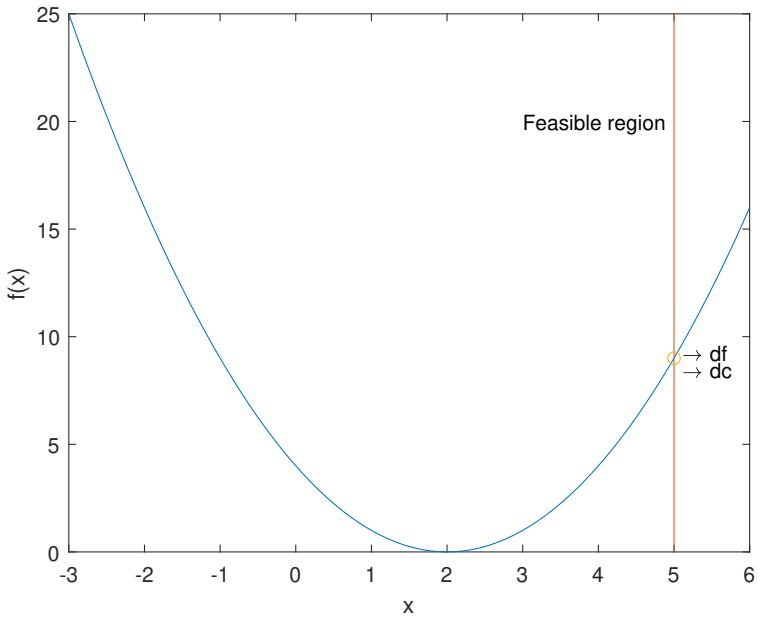

Figure 3.9: Illustration of problem (3.7).

where $\lambda = \begin{bmatrix}\lambda_1 & \lambda_2\end{bmatrix}^T$. *Then there exists* $\lambda^* = \begin{bmatrix}6 & 0\end{bmatrix}^T$ *such that*

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{i=1}^{2} \lambda_i^* \nabla c_i(x^*) = \begin{bmatrix}2(x_1^* - 2)\\ 2(x_2^* - 3)\end{bmatrix} + 6\begin{bmatrix}1\\0\end{bmatrix} + 0\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

3.2.3 KKT Conditions

In all the above examples, we are able to demonstrate that the first order gradient of the Langrangian function with respect to x is 0 at the optimal solution x^*, given certain value for λ. In the remaining of this section, we will formalize this condition, commonly known as Karush–Kuhn–Tucker (KKT) conditions[1]. We start by definition the notion of active set.

Definition 3.2.2 (Active Set)**.** *At any feasible point* x*, the set of constraints such that equality holds is called active set* $A(x)$*. In orhter words,*

$$A(x) = \{i \in \mathcal{I} \cup \mathcal{E} \mid c_i(x) = 0\} = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\}$$

Example 3.2.6 (Active Set)**.** *For optimization problem (3.8), at optimal solution* $x^* = \begin{bmatrix}-1 & 3\end{bmatrix}^T$*, the constraint* c_1 *is satisfied with equality, i.e.,* $c_1(x) = x_1 + 1 = 0$*. Therefore,* c_1 *is called active constraint. On the other hand,* c_2 *is satisfied with inequality, i.e.,* $c_2(x) = x_2 - 6 = -2 < 0$*. Therefore,* c_2 *is called inactive constraint, and the active set at* x^* *is* $A(x^*) = \{1\}$*.*

Consider another feasible point $\bar{x} = \begin{bmatrix}-1 & 5\end{bmatrix}^T$*. Then it is obvious that both constraints are satisfied with equality. Therefore, the active set at* $\bar{x}$ *is* $A(\bar{x}) = \{1, 2\}$*.*

Definition 3.2.3 (LICQ)**.** *Given a feasible point* x *and the active set* $A(x)$*, the linear independence constraint qualification (LICQ) is said to hold if the set of active constraint gradient* $\{\nabla c_i(x), i \in A(x)\}$ *is linearly independent.*

[1] `https://en.wikipedia.org/wiki/Karush-Kuhn-Tucker_conditions`

Theorem 3.2.1 (KKT Conditions; [5]). *Given an optimization problem (3.1) such that the objective function f and constraints c_i are all continuously differentiable. Define Lagrange multiplier λ, with components λ_i, $i \in \mathcal{E} \cup \mathcal{I}$, and Lagrangian function*

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x) = f(x) + \lambda^T c(x). \tag{3.9}$$

Given a local solution x^* such that LICQ holds at x^*, then there exists a Lagrange multiplier λ^* such that the following conditions are satisfied at (x^*, λ^*):

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \qquad \text{(Stationary condition)} \tag{3.10a}$$
$$c_i(x^*) = 0, \quad i \in \mathcal{E}, \qquad \text{(Primary feasibility)} \tag{3.10b}$$
$$c_i(x^*) \leq 0, \quad i \in \mathcal{I}, \qquad \text{(Primary feasibility)} \tag{3.10c}$$
$$\lambda_i^* \geq 0, \quad i \in \mathcal{I}, \qquad \text{(Dual feasibility)} \tag{3.10d}$$
$$\lambda_i^* c_i(x^*) = 0, \quad i \in \mathcal{E} \cup \mathcal{I}, \qquad \text{(Complementary condition)} \tag{3.10e}$$

Note that (3.10a) is called stationary condition, and it characterizes the condition that the gradient of objective function and those of active constraints reach stationarity. (3.10b) and (3.10c) are called primary feasibility conditions and it requires that x^* to be a feasible point. (3.10d) is simply asking that λ_i^* to be nonnegative, which is also called dual feasibility conditions. Finally, (3.10e) is the complementary conditions, which implies that for any constraint i, it is either active (so $c_i(x) = 0$) or $\lambda_i^* = 0$, or possibly both. It further implies that for any inactive constraint, its corresponding Lagrange multiplier is 0, i.e., λ_i^* for all $i \notin A(x^*)$. Therefore, the stationary condition (3.10a) can also be rewritten as

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{i \in A(x^*)} \lambda_i^* \nabla c_i(x^*) \tag{3.11}$$

Example 3.2.7 (KKT Conditions). *Consider the optimization problem (3.8) discussed in Example 3.2.5. The Lagrangian function is given by*

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T c(x) = (x_1 - 2)^2 + (x_2 - 3)^2 + \lambda_1(x_1 + 1) + \lambda_2(x_2 - 5)),$$

Next we show that the optimal solution $x^* = \begin{bmatrix} -1 & 3 \end{bmatrix}^T$ with active set $A(x^*) = \{1\}$ (see Example 3.2.6), and $\lambda^* = \begin{bmatrix} 6 & 0 \end{bmatrix}^T$ satisfy the KKT condition.

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{i=1}^{2} \lambda_i^* \nabla c_i(x^*) = \begin{bmatrix} 2(x_1^* - 2) \\ 2(x_2^* - 3) \end{bmatrix} + 6 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$c_1(x^*) = -1 + 1 = 0 \leq 0$$
$$c_2(x^*) = 3 - 5 = -2 \leq 0$$
$$\lambda_1 = 6 > 0$$
$$\lambda_2^* = 0 \geq 0$$
$$\lambda_1 c_1(x^*) = 6 \times 0 = 0$$
$$\lambda_2 c_2(x^*) = 0 \times (-1) = 0$$

Remark 3.2.1. *KKT conditions characterize the first-order necessary conditions that an optimal solution need to satisfy. A lot of numerical optimization algorithms aim to solve the optimization problem by finding a pair of primal-dual solution (x^*, λ^*) that satisfy the KKT condition.*

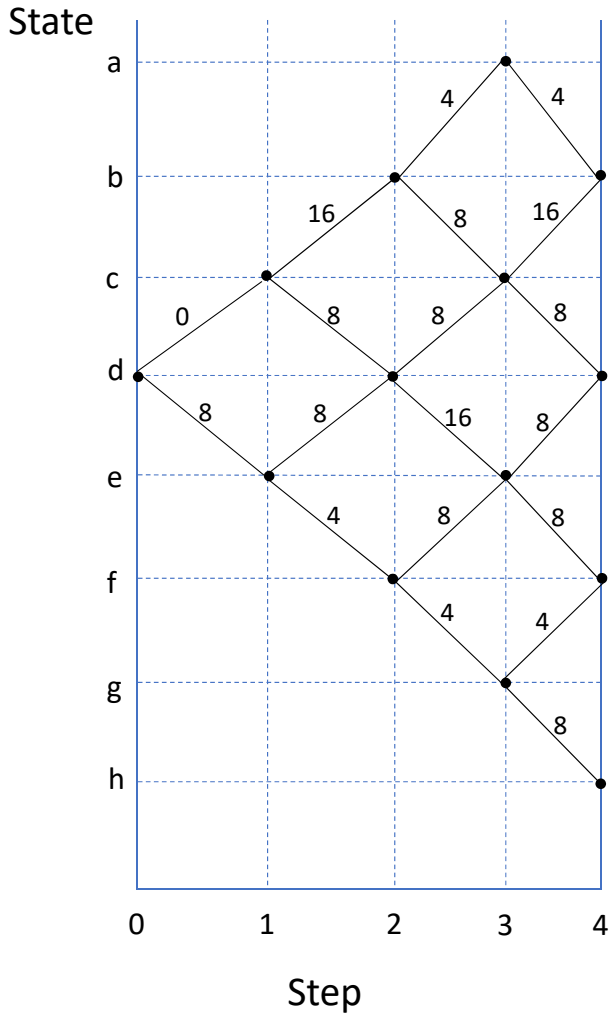

Figure 3.10: Routing problem, modified from [6].

Remark 3.2.2. *It is worth noting that KKT condition is not sufficient in general, meaning a solution that satisfies KKT conditions may not be an optimal solution (neither global nor local). When the problem is convex, however, KKT become both necessary and sufficient.*

3.3 Dynamic Programming

3.3.1 Routing Problem

Before we introduce dynamic programming mathematically, let's look at a simple routing problem as illustrated in Fig. 3.10, which is taken and modified from [6]. Denote the state space in this example as $X = \{a, b, c, d, e, f, g, h\}$, and the goal is to travel from the starting point $x_0 = d$ to any of the states at $t = 4$, where the control input at each step can be either U (denoting step-up) or D (denoting step-down). The number along the line between two points denote the cost associated with the transitions. The control problem is then to decide a sequence of control actions that minimizes the total cost.

Since in this problem there are a total number of 15 possible control sequences, one can actually

Control	-	UUUD	UUDU	UUDD	UDUU	UDUD	UDDU	UDDD
Cost	-	24	40	32	32	24	32	32
Control	DUUU	DUUD	DUDU	DUDD	DDUU	DDUD	**DDDU**	DDDD
Cost	40	32	40	40	28	28	**20**	24

Table 3.1: Control cost for all possible sequences.

Step	0	1		2			3			
State	d	c	e	b	d	f	a	c	e	g
Optimal control	D	U/D	D	U	U	D	D	D	U/D	U
Optimal cost	20	24	12	8	16	8	4	8	8	4

Table 3.2: Control cost for all possible sequences.

enumerate the cost associated with each sequence to find the optimal one. Table 3.1 list the costs for all the 15 control sequences. It is then straightford to see that the control sequence "DDDU" results in the minimum cost of 20.

A more systematic way to solve the above problem, which is the backbone of the dynamic programming, is to divide the whole N-stage optimization problem by N 1-stage problems, where $N = 4$ in this example. The first problem start at step $N - 1 = 3$ and tries to find the optimal control for each state at this step, which is listed in Table 3.2. Then at step $N - 2 = 2$, there are three possible states, b, d, and f. For each state, its cost is the one step cost plus the minimum cost of the resulted state. For example, at state b, the action U will incur a cost of 4 and reach state a for which the minimum cost is 4. Therefore, the action U at state b at step 2 will result in a cost of 8. Similar argument applies to d and f at step 3. Then we move one step back to step 1, where there are two possible states, c and e. Finally we move to step 0 at state d, where one can choose U to move up to state c with cost 0+24=24, or choose D to move down to state e with cost 8+12=20. Obviously the optimal control is D at this point.

3.3.2 Optimal Control Problem

This section presents the optimal control problem (OCP) that has many applications, and then in the next section we will discuss how dynamic programming can be used to solve the optimal control problem.

Consider a discrete time system

$$x_{k+1} = f(x_k, u_k) \tag{3.12}$$

Note that here we use f to denote the system dynamics, and the DT LTI system (2.16) is a special case of (3.12). Consider the following optimal control problem over a finite time step $k = 0, 1, \ldots, K$.

$$\min_{\mathbf{u},\mathbf{x}} \quad V_f(x_K) + \sum_{k=0}^{K-1} \ell(x_k, u_k) \qquad \text{subject to } \begin{cases} x_{k+1} = f(x_k, u_k), & k = 0, 1, \ldots, K-1 \\ x_k \in \mathcal{X}, u_k \in \mathcal{U}, & k = 0, 1, \ldots, K-1 \\ x_K \in \mathcal{X}_f \end{cases} \tag{3.13}$$

where the initial state x_0 is given, $\mathbf{u} = \begin{bmatrix} u_0^T & u_1^T & \cdots & u_{K-1}^T \end{bmatrix}^T$, $\mathbf{x} = \begin{bmatrix} x_1^T & x_2^T & \cdots & x_K^T \end{bmatrix}^T$, $\ell(x, u)$ is called stage cost function since it defines the cost of each time step, $V_f(\cdot)$ is called terminal cost function which is applied to the terminal state only, $\mathcal{X}$ and $\mathcal{U}$ are feasible regions for state and input, and $\mathcal{X}_f$ is the terminal constraint.

Now for the simplicity of notation, let's drop the input and state constraints, and focus on unconstrained optimal control problem[2]

$$\min_{\mathbf{u},\mathbf{x}} \quad V_f(x_K) + \sum_{k=0}^{K-1} \ell(x_k, u_k) \qquad \text{subject to } x_{k+1} = f(x_k, u_k), \quad k = 0, 1, \ldots, K-1 \tag{3.14}$$

For $0 \le i < K$, define $\mathbf{u}^i = \begin{bmatrix} u_i^T & u_{i+1}^T & \cdots & u_{K-1}^T \end{bmatrix}^T$ and

$$V(\mathbf{u}^i, x, i) = V_f(x_K) + \sum_{k=i}^{K-1} \ell(x_k, u_k) \quad \text{such that } \begin{cases} x_{k+1} = f(x_k, u_k), & k = i, 1, \ldots, K-1 \\ x_i = x \end{cases}$$

Then it is obvious that the cost function in (3.14) can be denoted as $V(\mathbf{u}, x_0, 0) = V_f(x_K) + \sum_{k=0}^{K-1} \ell(x_k, u_k)$. Finally, denote $\mathbf{u}^*$ as the optimal solution of (3.14), and $V^*(x_0, 0)$ as the optimal cost associated with $\mathbf{u}^*$, i.e., $V^*(x_0, 0) = V(\mathbf{u}^*, x_0, 0)$. We further define $V^*(x, i) = V(\mathbf{u}^{i,*}, x, i)$.

Example 3.3.1 (Optimal Control Problem)**.** *Consider steering control problem in Example 3.1.3, where the OCP is formulated as*

$$\min_{u_k, x_k} \quad \sum_{k=1}^{K} \|x_k\|_Q^2 \qquad \textit{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, & k = 1, \ldots, K-1 \\ u_k(1) = 0, & k = 1, \ldots, K-1 \\ u_l \le u_k(2) \le u_u, & k = 1, \ldots, K-1 \end{cases} \tag{3.15}$$

Here given x_0, we have the following formula for various cost functions and constraints discussed above.

$$\begin{aligned} \ell(x_k, u_k) &= \|x_k\|_Q^2 = x_k^T Q x_k \\ V_f(x_K) &= \|x_K\|_Q^2 = x_K^T Q x_K \\ \mathcal{X} &= \mathbb{R}^3 \\ \mathcal{X}_f &= \mathbb{R}^3 \\ \mathcal{U} &= \{u \in \mathbb{R}^{n_2} \mid u(1) = 0, u_l \le u(2) \le u_u\} \\ V(\boldsymbol{u}^i, x, i) &= \sum_{k=i}^{K} \|x_k\|_Q^2 \quad \textit{such that} \quad x_{k+1} = Ax_k + Bu_k, \quad x_i = x \\ V(\boldsymbol{u}, x_0, 0) &= \sum_{k=0}^{K} \|x_k\|_Q^2 \quad \textit{such that} \quad x_{k+1} = Ax_k + Bu_k, \quad x_0 = x \end{aligned}$$

Similar to the routing problem, DP breaks down the OCP (3.14) into multiple 1-stage problem. We will utilize the following observation.

$$\begin{aligned} V^*(x, i) &= \min_{\mathbf{u}^i} \left(V_f(x_K) + \sum_{k=i}^{K-1} \ell(x_k, u_k) \right) \\ &= \min_{\mathbf{u}^i} \left(\ell(x, u_i) + \underbrace{V_f(x_K) + \sum_{k=i+1}^{K-1} \ell(x_k, u_k)}_{V(\mathbf{u}^{i+1}, x_{i+1}, i+1)} \right) \end{aligned}$$

[2]Refer to [6, Appendix C] for treatment of DP-based constrained OCP. We will return to constrained OCP when we discuss model predictive control.

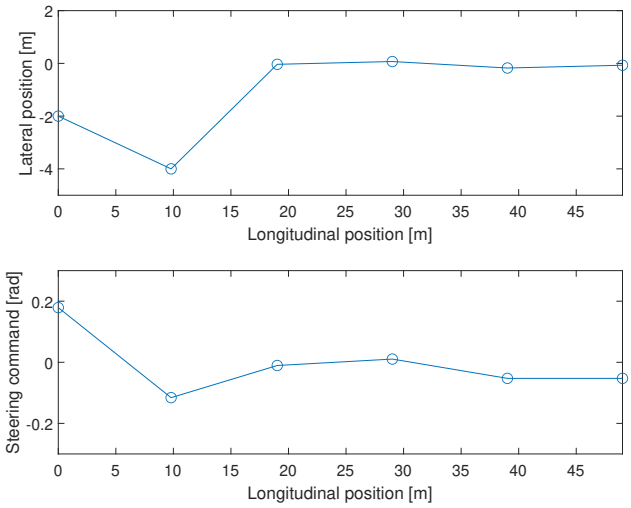

Figure 3.11: Optimal solution for Example 3.3.2.

$$
\begin{aligned}
&= \min_{u_i} \left(\ell(x, u_i) + \min_{\mathbf{u}^{i+1}} \left(V(\mathbf{u}^{i+1}, f(x, u_i), i+1) \right) \right) \\
&= \min_{u_i} \left(\ell(x, u_i) + V^*(f(x, u_i), i+1) \right)
\end{aligned} \tag{3.16}
$$

Therefore, at each step of DP, one needs to find the control u_i for current step such that the sum of 1-stage cost and the value function after the state transition is minimized.

Example 3.3.2 (DP Approach for Steering Control)**.** *Consider steering control problem in Example 3.1.3. Instead of using Genetic Algorithm, which can be time consuming, we now use DP to find the solution of (3.15). The results is shown in Fig. 3.11, where the sampling time T_s is increased to 1 second to alleviate the computational burden. Compare the the GA results, DP solution presents similar patterns. Note that the Matlab Code Listing 3.4 is a naive implementation of DP. We will talk about a nicer way to "implement" DP when we discuss LQR.*

Remark 3.3.1. *The Bellman equation (3.16) involves a minimization in the bracket. In other words, the optimization is done recursively, as can be seen from the Matlab Code Listing 3.4 for Example 3.3.2. If one can find a way to parameterize the value function $V^*(f(x, u_i), i+1)$, then the minimization involved in the right hand side of (3.16) becomes truly 1-stage optimization. This is essentially the idea of reinforcement learning, where the "learning" part is to learn the optimal value function through function approximator such as polynomial and neural nets. Once the value function is learned, the control is simply to find the optimizer that minimizes the sum of one step cost and value function of next step (whose calculation does not involve optimization anymore).*

3.4 Quadratic Programming

An optimization problem with a quadratic objective function and linear constraints is called a quadratic programming (QP) problem. This section will focus on generic QP problems and their solutions. It will become clear how QP can be utilized to solve the OCP (3.13) when we discuss model predictive control in Chapter 5.

The general QP has the following form

$$\min_{\mathbf{x}} \quad J(x) = \frac{1}{2}x^T Q x + h^T x \quad \text{subject to } \begin{cases} c_i^T x = b_i, & i \in \mathcal{E}, \\ c_i^T x \le b_i, & i \in \mathcal{I}, \end{cases} \tag{3.17}$$

where Q is a symmetric $n \times n$ matrix, h, c_i and x are vectors with size n.

Remark 3.4.1. *When Q is a positive semi-definite matrix, then QP (3.17) is a convex QP. When Q is a positive definite matrix, then then QP (3.17) is a strictly convex QP.*

Remark 3.4.2. *QP can always be solved in a finite amount of time, and convex QP is even easier to solve. Therefore, QP has a lot of application in embedded control systems, where the control problems are cast into QP for the ease of computation.*

Example 3.4.1 (Portfolio Optimization[3])**.** *Suppose there are a collection of n possible investments (stocks, ETF, etc). The return for each investment is denoted as r_i, $i = 1, 2, \ldots, n$. Assume r_i is a random variable with mean $\mu_i = E[r_i]$ and variance $\sigma_i^2 = E[(r_i - \mu_i)^2]$. Furthermore, the correlations between two returns are given as follows*

$$\rho_{ij} = \frac{E[(r_i - \mu_j)(r_j - \mu_j)]}{\sigma_i \sigma_j}, \qquad \forall i, j = 1, 2, \ldots, n$$

An investor distributes a fraction x_i of available funds into investment i, $i = 1, 2, \ldots, n$. Assume that the investor distributes all available funds, and short-selling are not allowed. Therefore, we have the following constraints on the decision variable x_i

$$\sum_{i=1}^{n} x_i = 1 \qquad \textit{and} \qquad x \ge 0.$$

The return can be modeled by

$$R = \sum_{i=1}^{n} x_i r_i.$$

Suppose the objective of the investor is to maximize the expected return while at the same time minimize the risk (as measured by the variance of R). The expected return is given by

$$E[R] = E\left[\sum_{i=1}^{n} x_i r_i\right] = \sum_{i=1}^{n} E\left[x_i r_i\right] = \sum_{i=1}^{n} x_i E\left[r_i\right] = \sum_{i=1}^{n} x_i \mu_i = \mu^T x$$

and the variance is given by

$$\begin{aligned} Var[R] &= E\left[(R - E[R])^2\right] = E\left[\left(\sum_{i=1}^{n} x_i r_i - \sum_{i=1}^{n} x_i \mu_i\right)^2\right] \\ &= E\left[\left(\sum_{i=1}^{n} x_i (r_i - \mu_i)\right)^2\right] = E\left[\sum_{i=1}^{n}\sum_{j=1}^{n} x_i (r_i - \mu_i)(r_j - \mu_j) x_j\right] \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} E\left[x_i (r_i - \mu_i)(r_j - \mu_j) x_j\right] = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i E\left[(r_i - \mu_i)(r_j - \mu_j)\right] x_j \end{aligned}$$

[3] Adopted from [5, Example 16.1].

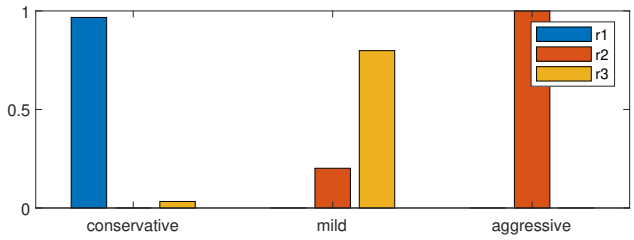

Figure 3.12: Optimal solution for Example 3.4.1.

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i \rho_{ij} \sigma_i \sigma_j x_j$$

Define a $n \times n$ symmetric positive semi-definite covariance matrix Q such that the ijth element of Q is given by $\rho_{ij}\sigma_i\sigma_j$. Then we have

$$Var[R] = x^T Q x$$

To combine the maximization of expected return and the minimization of risk, one can solve the following QP problem to find the optimal portfolio:

$$\min_{\boldsymbol{x}} \quad J(x) = \frac{1}{2} x^T Q x - \kappa \mu^T x \quad \textit{subject to} \begin{cases} c^T x = 1 \\ -x \le 0, \end{cases}$$

where $c = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$ and κ is a hyperparameter that balances expected return and risks.

Let $n = 3$, $\mu_1 = 0.3$, $\mu_2 = 2$, $\mu_3 = 1$, $\sigma_1 = 0.2$, $\sigma_2 = 2$, $\sigma_3 = 0.7$, $\rho_{12} = 0.2$, $\rho_{13} = -0.1$, and $\rho_{23} = 0.7$. Figs. 3.12 and 3.13 plots results for $\kappa = 0$ (conservative), $\kappa = 1$ (mild), and $\kappa = 10$ (aggressive). As can be seen, the expected return is higher if more aggressive portfolio is selected, while at the same time, the worst case loss is higher.

3.4.1 Equality-constrained QP

Consider the following equality-constrained QP

$$\min_{\mathbf{x}} \quad J(x) = \frac{1}{2} x^T Q x + h^T x \quad \text{subject to } Ax = b, \tag{3.18}$$

where the constraint matrix $A \in \mathbb{R}^{m \times n}$ is assumed to has full row rank.

According to Theorem 3.2.1, for x^* to be the solution of (3.18), there exists a Lagrange multiplier λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) = Qx^* + h + A^T \lambda^* = 0 \\ Ax^* = b. \end{aligned}$$

Putting it into matrix form, we have

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix} \tag{3.19}$$

The matrix in (3.19) is called KKT matrix, and it is the basis for many numerical solver, as it converts the optimization (3.18) into a set of linear equations (often referred to as KKT equations or KKT system), for which a lot of methods can be used.

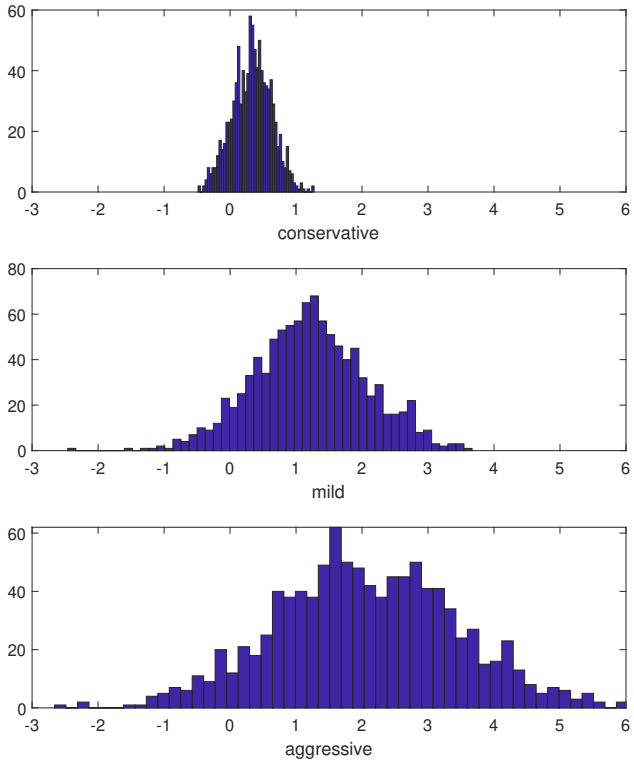

Figure 3.13: Simulated returns for Example 3.4.1.

Example 3.4.2 (Solving equality-constrained QP)**.** *Consider an equality-constrained QP of (3.18) with*

$$Q = \begin{bmatrix} 0.04 & 0.08 & -0.014 \\ 0.08 & 4 & 0.98 \\ -0.014 & 0.98 & 0.49 \end{bmatrix}, \quad h = \begin{bmatrix} -0.1 \\ -2 \\ -1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad b = 1$$

Utilizing Matlab's quadprog command, the optimal solution can be found to be

$$x^* = \begin{bmatrix} -0.6374 \\ 0.1018 \\ 1.5356 \end{bmatrix}$$

The KKT equations in this example is given by

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} 0.04 & 0.08 & -0.014 & 1 \\ 0.08 & 4 & 0.98 & 1 \\ -0.014 & 0.98 & 0.49 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix} = \begin{bmatrix} -0.1 \\ -2 \\ -1 \\ 1 \end{bmatrix}$$

solving which yields

$$\begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -0.6374 \\ 0.1018 \\ 1.5356 \\ 0.1389 \end{bmatrix}.$$

In many cases, the KKT system (3.19) is solved iteratively. Given a current iterate x, the goal is to find a step p such that $x^* = x + p$ satisfies (3.19). Therefore,

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x+p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} p \\ \lambda^* \end{bmatrix} + \begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix}$$

and

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix} - \begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} -h - Qx \\ b - Ax \end{bmatrix} = \begin{bmatrix} -g \\ \bar{b} \end{bmatrix}$$

where $g = Qx + h$ and $\bar{b} = b - Ax$.

3.4.2 Inequality-constrained QP

Now, let's turn our attention to inequality-constrained QP (3.17), rewritten in matrix form as

$$\min_{\mathbf{x}} \quad J(x) = \frac{1}{2} x^T Q x + h^T x \quad \text{subject to } Ax \le b \quad \& \quad A_{eq} x = b_{eq}. \tag{3.20}$$

Recall that the notion of active set $A(x)$ is the set of constraints for which equality holds at x. If we know $A(x^*)$ in advance, then solving (3.20) is same as solving the KKT system (3.19) for equality-constrained QP with equality constraints $A(x^*)$, i.e.,

$$\begin{bmatrix} Q & (A(x^*))^T \\ (A(x^*)) & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix}$$

However, in most cases, $A(x^*)$ is not known. Fortunately there is only a finite number of candidates for $A(x^*)$. The *active-set method* utilizes this fact, and search for an active set $A(x^*)$ such that the solution of (3.19) satisfies all the other KKT conditions in (3.10).

Example 3.4.3 (Solving inequality-constrained QP through brute force search)**.** *Consider an inequality-constrained QP of (3.20) with*

$$Q = \begin{bmatrix} 0.04 & 0.08 & -0.014 \\ 0.08 & 4 & 0.98 \\ -0.014 & 0.98 & 0.49 \end{bmatrix}, \quad h = \begin{bmatrix} -0.1 \\ -2 \\ -1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Furthermore, $A_{eq} = b_{eq} = \emptyset$. Utilizing Matlab's quadprog command, the optimal solution can be found to be

$$x^* = \begin{bmatrix} 3.2468 \\ 0 \\ 2.1336 \end{bmatrix}$$

Since there are 3 constraints, there are a total number of $2^3 = 8$ candidates for active set $A(x^*)$. Let's assume $A(x^*) = \{1\}$. Then the KKT equations are given by

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} 0.04 & 0.08 & -0.014 & 1 \\ 0.08 & 4 & 0.98 & 0 \\ -0.014 & 0.98 & 0.49 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix} = \begin{bmatrix} -0.1 \\ -2 \\ -1 \\ 0 \end{bmatrix}$$

solving which yields

$$\begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2.0408 \\ -0.1286 \end{bmatrix}.$$

Since other KKT conditions are not satisfied, i.e., $\lambda^* \geq 0$, $\{1\}$ is not the optimal active set. Now, let's repeat for $A(x^*) = \{1, 2\}$, in this case we have

$$\begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2.0408 \\ -0.1286 \\ 0 \end{bmatrix}.$$

Again we have the conditions $\lambda^* \geq 0$ violated. Therefore $\{1, 2\}$ is not the optimal active set. Now let's try $A(x^*) = \{2\}$, the KKT equations in this case are given by

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} 0.04 & 0.08 & -0.014 & 0 \\ 0.08 & 4 & 0.98 & 1 \\ -0.014 & 0.98 & 0.49 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -h \\ b \end{bmatrix} = \begin{bmatrix} -0.1 \\ -2 \\ -1 \\ 0 \end{bmatrix}$$

solving which yields

$$\begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} 3.2468 \\ 0 \\ 2.1336 \\ 0.3506 \end{bmatrix},$$

which satisfies all the other KKT conditions. Therefore, $\{2\}$ is the optimal active set and the solution found.

Remark 3.4.3. *Example 3.4.3 is to illustrate the basic idea of active set method only. Since the number candidates for optimal active set is exponential to the number of inequality constraints, brute force search is of minimal practical usage. In fact, many numerical solvers utilize the constraints violation from one iteration to search for the next "working" active set to try, instead of naively search one by one. See [5, Chapter 16] for more detailed treatment of inequality-constrained QP.*

3.5 Matlab Codes

```
close all; clear variables; clc

l = 3;
xbar = [10;0;0]; x0 = [0;0;0];
Ts = 1; u = [0.5;0.01];
t = 0:Ts:5; N = numel(t);

A = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; B = [Ts 0;0 0;0 xbar(1)/l*Ts];
xd = zeros(N,3);
for n = 2:N
    xd(n,:) = (A*xd(n-1,:)'+B*u)'+randn(1,3)./[10,10,100];
end
xd = xd + repmat(x0',N,1);

lh = fmincon(@(l)se(l,xd,xbar,x0,u,Ts,N),2,[],[],[],[],1,4);

Ah = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; Bh = [Ts 0;0 0;0 xbar(1)/lh*Ts];
xh = zeros(N,3);
for n = 2:N
    xh(n,:) = (Ah*xh(n-1,:)'+Bh*u)';
end
xh = xh + repmat(x0',N,1);

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*2; f.Position = pos;

subplot(3,1,1), plot(t,xd(:,1),'o-',t,xh(:,1),'^-'),
xlabel('Time [s]'), ylabel('v [m/s]'), axis tight,
legend('Data','Estimated')
subplot(3,1,2), plot(t,xd(:,2),'o-',t,xh(:,2),'^-'),
xlabel('Time [s]'), ylabel('p_y [m]'), axis tight,
legend('Data','Estimated')
subplot(3,1,3), plot(t,xd(:,3),'o-',t,xh(:,3),'^-'),
xlabel('Time [s]'), ylabel('\phi [rad]'), axis tight,
legend('Data','Estimated')

function v = se(l,xd,xbar,x0,u,Ts,N)
v = 0;
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; B = [Ts 0;0 0;0 xbar(1)/l*Ts];
x = x0;
for n = 2 : N
    x = A*x+B*u; v = v + norm(x-xd(n,:)')^2;
end

end
```

Listing 3.1: MATLAB Commands for Example 3.1.2.

```
close all; clear variables; clc

l = 3;
xbar = [10;0;0]; ubar = [0;0];
x0 = [0;-2;-0.2];
Ts = 0.2; t = 0:Ts:5; N = numel(t);
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1];
B = [Ts 0;0 0;0 xbar(1)/l*Ts];
lb = -0.1*ones(N-1,1); ub = 0.1*ones(N-1,1);

uopt = ga(@(u)lateral_error(u,A,B,x0,N),N-1,[],[],[],[],lb,ub);

x = zeros(N,3);
x(1,:) = x0';
for n = 2:N
    x(n,:) = (A*x(n-1,:)'+B*[0;uopt(n-1)])';
end
x = x + repmat(xbar',N,1);

px = cumsum(x(:,1).*cos(x(:,3))*Ts); px=[0;px(1:end-1)];
py = x(:,2);

f=figure;
subplot(2,1,1), plot(px,py,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-3,2])

subplot(2,1,2), plot(px,[uopt,uopt(end)],'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.3,0.3])

function v = lateral_error(u,A,B,x0,N)
v = 0;
x = x0;
for n = 2 : N
    x = A*x+B*[0;u(n-1)];
    v = v + norm(x)^2;
end
end
```

Listing 3.2: MATLAB Commands for Example 3.1.3.

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; x0 = [0;-2;-0.2];
Ts = 0.2; t = 0:Ts:5; N = numel(t);
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; B = [Ts 0;0 0;0 xbar(1)/l*Ts];
lb = -0.2*ones(N-1,1); ub = 0.2*ones(N-1,1);

uopt = ga(@(u)lateral_error(u,A,B,x0,N),N-1,[],[],[],[],lb,ub);
x = zeros(N,3); x(1,:) = x0';
u1 = uopt+(rand(1,N-1)-0.5)/10;
for n = 2:N
    x(n,:) = (A*x(n-1,:)'+B*[0;u1(n-1)])';
end
x = x + repmat(xbar',N,1);

lb=lb+0.05; ub=ub-0.05;
uopt1 = ga(@(u)lateral_error(u,A,B,x0,N),N-1,[],[],[],[],lb,ub);
x1 = zeros(N,3); x1(1,:) = x0';
u1 = uopt1+(rand(1,N-1)-0.5)/50;
for n = 2:N
    x1(n,:) = (A*x1(n-1,:)'+B*[0;u1(n-1)])';
end
x1 = x1 + repmat(xbar',N,1);

px = cumsum(x(:,1).*cos(x(:,3))*Ts); px=[0;px(1:end-1)];
py = x(:,2);
px1 = cumsum(x1(:,1).*cos(x1(:,3))*Ts); px1=[0;px1(1:end-1)];
py1 = x1(:,2);

f=figure;

subplot(2,1,1), plot(px,py,'*-',px1,py1,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-3,2]), legend('Expected','Robust')

subplot(2,1,2), plot(px,[uopt,uopt(end)],'*-',px1,[uopt1,uopt1(end)],'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.3,0.3]), legend('Expected','Robust')

function v = lateral_error(u,A,B,x0,N)
v = 0;
x = x0;
for n = 2 : N
    x = A*x+B*[0;u(n-1)];
    v = v + norm(x)^2;
end
end
```

Listing 3.3: MATLAB Commands for Example 3.1.4.

```
l = 3; xbar = [10;0;0]; ubar = [0;0]; x0 = [0;-2;-0.2];
Ts = 1; t = 0:Ts:5; N = numel(t);
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; B = [Ts 0;0 0;0 xbar(1)/l*Ts];
lb = -0.2; ub = 0.2; U = linspace(lb,ub,20);

tic, [uopt,vopt] = dp(x0,0,N-1,U,A,B); toc

x = zeros(N,3); x(1,:) = x0';
for n = 2:N
    x(n,:) = (A*x(n-1,:)'+B*[0;uopt(n-1)])';
end
x = x + repmat(xbar',N,1);

f=figure;
px = cumsum(x(:,1).*cos(x(:,3))*Ts); px=[0;px(1:end-1)]; py = x(:,2);

subplot(2,1,1), plot(px,py,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-5,2])

subplot(2,1,2), plot(px,[uopt,uopt(end)],'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.3,0.3])

function [uopt,vopt] = dp(x0,n,N,U,A,B)
uopt = 0; vopt = inf;
if n == N-1
    for u = U
        x = A*x0+B*u;
        if norm(x)^2 < vopt
            vopt = norm(x)^2; uopt = u;
        end
    end
else
    for u = U
        x = A*x0+B*u;
        [u1,v1] = dp(x,n+1,N,U,A,B);
        if norm(x)^2+v1 < vopt
            vopt = norm(x)^2+v1; u1opt = u1; uopt = u;
        end
    end
    uopt = [uopt,u1opt(:)'];
end
end

Elapsed time is 5.407224 seconds.
```

Listing 3.4: MATLAB Commands for Example 3.3.2. The implementation here is naive. We will talk about a nicer way to "implement" DP when we discuss LQR.

```
mu = [0.3 2 1];
sig = [0.2 2 0.7];
rho = [1 0.2 -0.1;0.2 1 0.7;-0.1 0.7 1];

Q = zeros(3,3);
for i = 1 : 3
    for j = 1 : 3
        Q(i,j) = rho(i,j)*sig(i)*sig(j);
    end
end
A = -eye(3);
b = zeros(3,1);
Aeq = ones(1,3);
beq = 1;

kapm = [0 1 10];
N = numel(kapm);
R = zeros(N,1);
X = zeros(N,3);
for i = 1 : N
    kap = kapm(i); h = -kap*mu;
    x = quadprog(Q,h,A,b,Aeq,beq);
    X(:,i) = x'; R(i) = mu*x;
end

f=figure;
subplot(2,1,1), bar(X'),
set(gca,'xticklabel',{'conservative','mild','aggressive'})
legend('r1','r2','r3')

S=rho-eye(3)+diag(sig); r = mvnrnd(mu,S,300);

R = zeros(300,N);
for i = 1 : N
    R(:,i) = r*X(:,i);
end

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos;


subplot(N,1,1), hist(R(:,1),30);xlim([-3 6]); xlabel('conservative');
subplot(N,1,2), hist(R(:,2),30);xlim([-3 6]); xlabel('mild');
subplot(N,1,3), hist(R(:,3),30);xlim([-3 6]); xlabel('aggressive');
```

Listing 3.5: MATLAB Commands for Example 3.4.1.

```
close all; clear variables; clc

Q = [0.0400    0.0800   -0.0140;
    0.0800    4.0000    0.9800;
   -0.0140    0.9800    0.4900];

A = ones(1,3);
b = 1;

h = [-0.1; -2; -1];
[x,~,~,~,lambda] = quadprog(Q,h,[],[],A,b);

x1 = [Q A';A 0]\[-h;b]


x =

   -0.6374
    0.1018
    1.5356


lambda =

  struct with fields:

    ineqlin: [0×1 double]
      eqlin: 0.1389
      lower: [3×1 double]
      upper: [3×1 double]


x1 =

   -0.6374
    0.1018
    1.5356
    0.1389
```

Listing 3.6: MATLAB Commands for Example 3.4.2.

```
Q = [0.0400    0.0800   -0.0140;
     0.0800    4.0000    0.9800;
    -0.0140    0.9800    0.4900];

A = -eye(3); b = zeros(3,1); h = [-0.1; -2; -1];
[x,~,~,~,lambda] = quadprog(Q,h,A,b)

idx = 1; As = A(idx,:);
x1 = [Q As';As zeros(size(As,1),size(As,1))]\[-h;b(idx)]

idx = 1:2; As = A(idx,:);
x1 = [Q As';As zeros(size(As,1),size(As,1))]\[-h;b(idx)]

idx = 2; As = A(idx,:);
x1 = [Q As';As zeros(size(As,1),size(As,1))]\[-h;b(idx)]

x =
    3.2468
    0.0000
    2.1336

lambda =
  struct with fields:
    ineqlin: [3×1 double]
      eqlin: [0×1 double]
      lower: [3×1 double]
      upper: [3×1 double]

x1 =
         0
         0
    2.0408
   -0.1286

x2 =
         0
         0
    2.0408
   -0.1286
         0

x3 =
    3.2468
    0.0000
    2.1336
    0.3506
```

Listing 3.7: MATLAB Commands for Example 3.4.3.

Chapter 4

Linear Quadratic Regulator

Consider the following discrete-time LTI system.

$$x_{k+1} = Ax_k + Bu_k \tag{4.1}$$

with initial value x_0. The goal of optimal control is to find a feedback control law $u_k = g(x_k)$ so that a certain performance index is optimized. In the case of linear quadratic regulator (LQR), a quadratic cost function will be used. Note that here we assume state feedback is available, and the goal is to regulate the states instead of output. The topic of state estimation from output measurement will be handled in Chapter 7 and the topic of output regulator/control will be discussed in Chapter 5.

Different from MPC in Chapter 5, LQR does not handle input and state constraints. In fact, this is a key difference between LQR and MPC. Though LQR does not handle constraints, a very useful feature in practice, LQR is easy to implement and slightly easier to tune compared to MPC.

4.1 Finite Horizon LQR

In finite horizon LQR, consider the following optimal control problem (OCP) over N time steps into the future:

$$\min_{\mathbf{u}} \quad \frac{1}{2}\|x_N\|_{Q_N}^2 + \sum_{k=0}^{N-1}\left(\frac{1}{2}\|x_k\|_Q^2 + \frac{1}{2}\|u_k\|_R^2\right) \qquad \text{subject to} \quad x_{k+1} = Ax_k + Bu_k, \tag{4.2}$$

where the input sequence is defined as

$$\mathbf{u} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}.$$

The weight matrices Q, R, Q_N are real and symmetric, with Q and Q_N being positive semidefinite and R being positive definite. Furthermore, to be consistent with previous chapters, we denote the stage and terminal cost functions as

$$\ell(x_k, u_k) = \frac{1}{2}\left(x_k^T Q x_k + u_k^T R u_k\right) = \frac{1}{2}\|x_k\|_Q^2 + \frac{1}{2}\|u_k\|_R^2$$

$$V_f(x_N) = \frac{1}{2} x_N^T Q_N x_N = \frac{1}{2} \|x_N\|_{Q_N}^2$$

Before we derive a solution to (4.2) using dynamic programming, we first derive the following formula for the sum of two quadratic functions.

Lemma 4.1.1 (Sum of Two Quadratics)**.** *Let $f_1(x) = \frac{1}{2}(x-a)^T A(x-a)$ and $f_2(x) = \frac{1}{2}(Cx-b)^T B(Cx-b)$, where A and B are positive definite matrices, then $f(x) = f_1(x) + f_2(x) = \frac{1}{2}(x-v)^T H(x-v) + \frac{1}{2}d$, where*

$$\begin{aligned} H &= A + C^T BC \\ v &= H^{-1}(Aa + C^T Bb) \\ d &= -(Aa + C^T Bb)^T H^{-1}(Aa + C^T Bb) + a^T Aa + b^T Bb \end{aligned}$$

Proof. According to the setup

$$\begin{aligned} 2f(x) =& 2f_1(x) + 2f_2(x) = (x-a)^T A(x-a) + (Cx-b)^T B(Cx-b) \\ =& x^T Ax - 2x^T Aa + a^T Aa + x^T C^T BCx - 2x^T C^T Bb + b^T Bb \\ =& x^T \underbrace{(A + C^T BC)}_{H} x - 2x^T (Aa + C^T Bb) + a^T Aa + b^T Bb \\ =& x^T Hx - 2x^T H \underbrace{H^{-1}(Aa + C^T Bb)}_{v} + a^T Aa + b^T Bb \\ =& (x-v)^T H(x-v) - v^T Hv + a^T Aa + b^T Bb \\ =& (x-v)^T H(x-v) - (Aa + C^T Bb)^T H^{-T} H H^{-1}(Aa + C^T Bb) + a^T Aa + b^T Bb \\ =& (x-v)^T H(x-v) \underbrace{- (Aa + C^T Bb)^T H^{-1}(Aa + C^T Bb) + a^T Aa + b^T Bb}_{d} \end{aligned}$$

Therefore

$$\begin{aligned} f(x) =& \frac{1}{2}(x-v)^T H(x-v) + \frac{1}{2}d \\ H =& A + C^T BC \\ v =& H^{-1}(Aa + C^T Bb) \\ d =& -(Aa + C^T Bb)^T H^{-1}(Aa + C^T Bb) + a^T Aa + b^T Bb \end{aligned}$$

Proof completes. □

Next we apply dynamic programm to solve (4.2), which can be written as follows.

$$\min_{u_0, u_1, \ldots, u_{N-2}} \quad \sum_{k=0}^{N-2} \ell(x_k, u_k) + \min_{u_{N-1}} V_f(x_N) + \ell(x_{N-1}, u_{N-1}) \qquad \text{subject to} \quad x_{k+1} = Ax_k + Bu_k. \tag{4.3}$$

Therefore, the problem to be solved at the last stage at time $k = N-1$ is

$$\min_{u_{N-1}} V_f(x_N) + \ell(x_{N-1}, u_{N-1}) \qquad \text{subject to} \quad x_N = Ax_{N-1} + Bu_{N-1},$$

or equivalently

$$\min_{u_{N-1}} \quad V_f(Ax_{N-1} + Bu_{N-1}) + \ell(x_{N-1}, u_{N-1}).$$

Note that at this stage, x_{N-1} is a parameter of the above optimization problem, as u_{N-1} has no impact on its value. Next we expand the objective function of the last stage optimization problem and get

$$\begin{aligned}
&V_f(Ax_{N-1}+Bu_{N-1})+\ell(x_{N-1},u_{N-1})\\
&\quad=\frac{1}{2}\left(Ax_{N-1}+Bu_{N-1}\right)^T Q_N\left(Ax_{N-1}+Bu_{N-1}\right)+\frac{1}{2}x_{N-1}^T Q x_{N-1}+\frac{1}{2}u_{N-1}Ru_{N-1}\\
&\quad=\underbrace{\frac{1}{2}\left(Ax_{N-1}+Bu_{N-1}\right)^T Q_N\left(Ax_{N-1}+Bu_{N-1}\right)+\frac{1}{2}u_{N-1}Ru_{N-1}}_{\text{Using formula derived in Lemma 4.1.1}}+\frac{1}{2}x_{N-1}^T Q x_{N-1}\\
&\quad=\frac{1}{2}(u_{N-1}-v)^T H(u_{N-1}-v)+\frac{1}{2}d+\frac{1}{2}x_{N-1}^T Q x_{N-1},
\end{aligned}$$

where

$$\begin{aligned}
H=&R+B^TQ_NB\\
v=&-H^{-1}B^TQ_NAx_{N-1}\\
d=&-(B^TQ_NAx_{N-1})^TH^{-1}(B^TQ_NAx_{N-1})+(Ax_{N-1})^TQ_NAx_{N-1}\\
=&-x_{N-1}^TA^TQ_NBH^{-1}B^TQ_NAx_{N-1}+x_{N-1}^TA^TQ_NAx_{N-1}\\
=&x_{N-1}^T\left(A^TQ_NA-A^TQ_NBH^{-1}B^TQ_NA\right)x_{N-1}
\end{aligned}$$

By inspecting the cost function, we have the optimal solution $u_{N-1}^*=v$, which is linear on x_{N-1}, and the optimal cost $\frac{1}{2}x_{N-1}^TQx_{N-1}+d$, which is quadratic on x_{N-1}. Therefore, all for x, when $x_{N-1}=x$, the solution to the last stage optimization problem is given as

$$\begin{aligned}
u_{N-1}^*(x)&=-H^{-1}B^TQ_NAx=K_{N-1}x\\
x_N^*(x)&=(A+BK_{N-1})x\\
V_{N-1}^*(x)&=\frac{1}{2}x^T\Pi_{N-1}x
\end{aligned}$$

where

$$\begin{aligned}
K_{N-1}&=-\left(R+B^TQ_NB\right)^{-1}B^TQ_NA\\
\Pi_{N-1}&=A^TQ_NA-A^TQ_NB\left(R+B^TQ_NB\right)^{-1}B^TQ_NA+Q
\end{aligned}$$

Note that $V_{N-1}^*(x)$ defines the *cost to go* from state x at $k=N-1$ under the optimal control law. Now, lets turn to the optimization problem (4.3), which can be represented as

$$\min_{u_0,u_1,\ldots,u_{N-2}}\quad\sum_{k=0}^{N-2}\ell(x_k,u_k)+\frac{1}{2}x_{N-1}^T\Pi_{N-1}x_{N-1}\qquad\text{subject to}\quad x_{k+1}=Ax_k+Bu_k.$$

Note that this problem is exact same as the LQR problem (4.2) with horizon $N-1$ and terminal weights Π_{N-1}, whose last stage problem - or second last stage of (4.3) - is given as follows.

$$\min_{u_{N-2}}V_{N-1}(x_{N-1})+\ell(x_{N-2},u_{N-2})\qquad\text{subject to}\quad x_{N-1}=Ax_{N-2}+Bu_{N-2}.$$

Repeat the above analysis, and get

$$u_{N-2}^*(x)=K_{N-2}x$$

$$x_N^*(x) = (A + BK_{N-2})x$$
$$V_{N-2}^*(x) = \frac{1}{2}x^T\Pi_{N-2}x$$

where

$$K_{N-2} = -\left(R + B^T\Pi_{N-1}B\right)^{-1}B^T\Pi_{N-1}A$$
$$\Pi_{N-2} = A^T\Pi_{N-1}A - A^T\Pi_{N-1}B\left(R + B^T\Pi_{N-1}B\right)^{-1}B^T\Pi_{N-1}A + Q$$

To summarize, to solve LQR problem (4.2), one needs to solve the following backward Riccati iteration

$$\Pi_{k-1} = A^T\Pi_k A - A^T\Pi_k B\left(R + B^T\Pi_k B\right)^{-1}B^T\Pi_k A + Q, \qquad , k = N, N-1, \ldots, 1 \tag{4.4a}$$

with terminal condition

$$\Pi_N = Q_N \tag{4.4b}$$

The optimal control policy at each time step is dependent on the Riccati matrix at time step $k+1$ and is given by

$$u_k^* = K_k x \tag{4.4c}$$
$$K_k = -\left(R + B^T\Pi_{k+1}B\right)^{-1}B^T\Pi_{k+1}A. \tag{4.4d}$$

And the optimal cost to go from time k to time N is

$$V_k^*(x) = \frac{1}{2}x^T\Pi_k x. \tag{4.4e}$$

Example 4.1.1 (LQR Steering Control)**.** *Consider the simplified vehicle dynamic model discussed in Example 3.1.3, where the DT LTI model is given as follows.*

$$x_{k+1} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 2\\ 0 & 0 & 1\end{bmatrix}x_k + \begin{bmatrix}0.2 & 0\\ 0 & 0\\ 0 & \frac{2}{3}\end{bmatrix}u_k = Ax_k + Bu_k$$

Now suppose the initial conditions are specified as $v_0 = 1$, $p_{y,0} = -2$, and $\phi_0 = -0.2$.

To regulate the vehicle speed, lateral position, and heading angle back to their nominal value, the following LQR problem is formulated.

$$\min_{\boldsymbol{u}} \quad \frac{1}{2}\|x_N\|_Q^2 + \frac{1}{2}\sum_{k=0}^{N-1}\|x_k\|_Q^2 + \|u_k\|_R^2 \qquad \textit{subject to} \quad x_{k+1} = Ax_k + Bu_k, \quad k = 1, \ldots, K-1$$

When

$$Q_N = Q = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}, \qquad R = \begin{bmatrix}1 & 0\\ 0 & 10\end{bmatrix}$$

the results are plotted in Fig. 4.1. As can be seen, the states are effectively converging back to 0*. However, since there is no constraints enforcement, the steering command can be large initially, i.e., 0.6 rad (or 34 degree). Now, let's decrease the penalty on acceleration, and increase the weight on steering efforts. In other words, now we have*

$$Q_N = Q = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}, \qquad R = \begin{bmatrix}0.1 & 0\\ 0 & 1000\end{bmatrix}$$

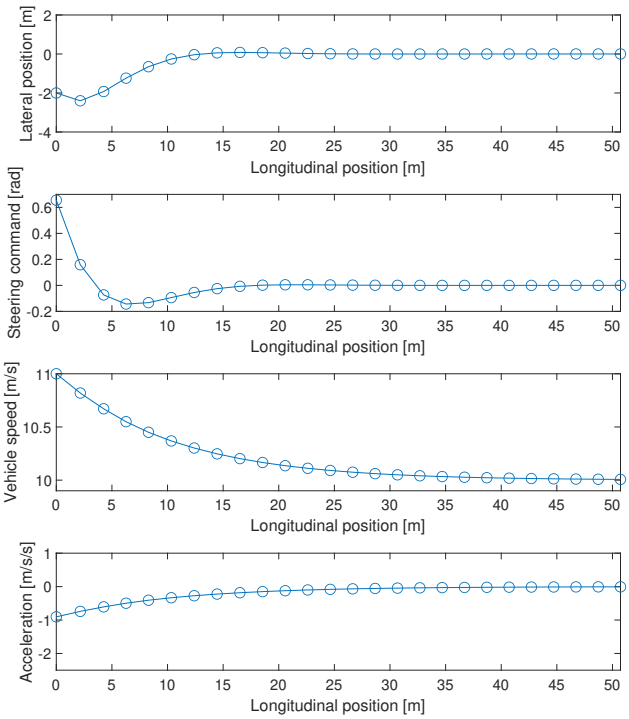

Figure 4.1: Optimal solution for Example 4.1.1.

The results are plotted in Fig. 4.2. As can be seen, now the deceleration is larger with better speed control. At the same time, the steering command is much smaller, with a price of worse lateral control performance.

4.2 Infinite Horizon LQR

When $N \to \infty$, problem (4.2) becomes infinite horizon LQR problem, rewritten as follows.

$$\min_{\mathbf{u}} \quad \sum_{k=0}^{\infty} \left(\frac{1}{2} \|x_k\|_Q^2 + \frac{1}{2} \|u_k\|_R^2 \right) \qquad \text{subject to} \quad x_{k+1} = Ax_k + Bu_k, \tag{4.5}$$

where the input sequence is defined as

$$\mathbf{u} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \end{bmatrix}.$$

Similar to the case with finite horizon, we assume that both Q and R are real and symmetric, with Q being positive semidefinite and R being positive definite.

Note that this is infinite dimensional problem and the optimal cost can be infinity.

Example 4.2.1. *Let*

$$x_{k+1} = 3x_k + 0u_k, \qquad x_0 = 2$$

Then for any control sequence $\boldsymbol{u}$*, we have*

$$\sum_{k=0}^{\infty} \left(\frac{1}{2} \|x_k\|_Q^2 + \frac{1}{2} \|u_k\|_R^2 \right) \geq \sum_{k=0}^{\infty} \left(\frac{1}{2} \|x_k\|_Q^2 \right) \geq \sum_{k=0}^{\infty} \left(\frac{1}{2} \|x_0\|_Q^2 \right) = \sum_{k=0}^{\infty} 2Q = \infty$$

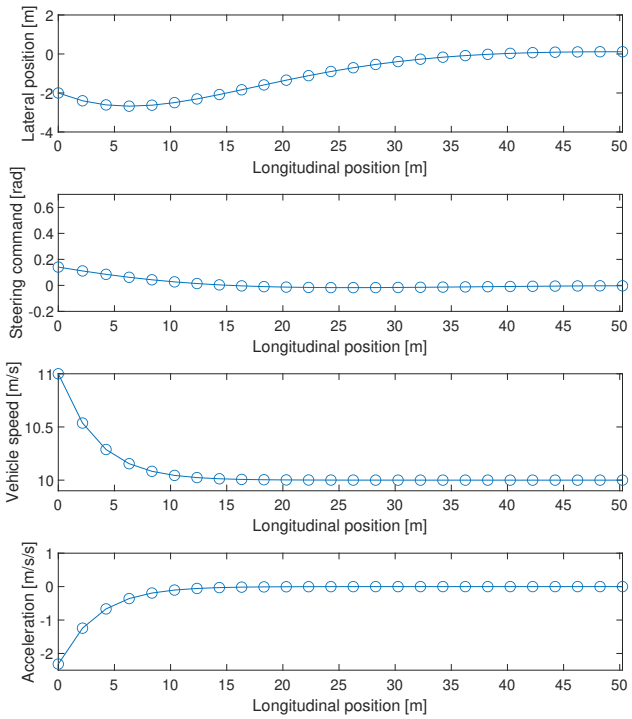

Figure 4.2: Optimal solution for Example 4.1.1 with different gains.

Remark 4.2.1. *In the above example, the system under control is not controllable. In general, if* (A, B) *is controllable, then there exists a finite control sequence* $u_0, u_1, \ldots, u_K$ *such that the states are steered to* 0*. Afterwards, applying the control* $u = 0$ *will keep the states at* 0*. Therefore, in this case the optimal cost is finite, for any initial state condition.*

Let us assume that $\mathbf{u}^*$ is the solution to (4.5) with initial condition x, and denote the optimal cost as $V^*(x)$. Then it can be argued that $V^*(x)$ is a quadratic function of x, denoted as $V^*(x) = \frac{1}{2}x^T\Pi x$ with $\Pi = \Pi^T$ being positive semidefinite. Next, we apply forward DP to (4.5) as follows.

$$
\begin{aligned}
V^*(x) = \min_{\mathbf{u}} \quad & \sum_{k=0}^{\infty}\left(\frac{1}{2}\|x_k\|_Q^2 + \frac{1}{2}\|u_k\|_R^2\right) \\
= \min_{u_0} \quad & \frac{1}{2}x^TQx + \frac{1}{2}u_0^TRu_0 + \underbrace{\min_{u_1,u_2,\ldots}\sum_{k=1}^{\infty}\left(\frac{1}{2}\|x_k\|_Q^2 + \frac{1}{2}\|u_k\|_R^2\right)}_{V^*(Ax+Bu_0)} \\
= \min_{u_0} \quad & \frac{1}{2}x^TQx + \frac{1}{2}u_0^TRu_0 + V^*(Ax+Bu_0) \\
= \min_{u_0} \quad & \frac{1}{2}x^TQx + \frac{1}{2}u_0^TRu_0 + \frac{1}{2}(Ax+Bu_0)^T\Pi(Ax+Bu_0) \\
= \min_{u_0} \quad & \frac{1}{2}x^TQx + \frac{1}{2}(u_0 + H^{-1}B^T\Pi Ax)^TH(u_0 + H^{-1}B^T\Pi Ax) + \frac{1}{2}d
\end{aligned} \tag{4.6}
$$

with

$$H = R + B^T\Pi B$$

$$d = -x^T A^T \Pi B H^{-1} B^T \Pi A x + x^T A^T \Pi A x$$

Examining the right-hand side of (4.6), we find the optimal u_0 to be

$$u_0^* = -H^{-1} B^T \Pi A x = -\left(R + B^T \Pi B\right)^{-1} B^T \Pi A x = Kx.$$

Therefore, the optimal state feedback gain is a constant matrix, whose value depends on Π. To find Π, we substitute u_0^* back to (4.6), and get

$$V^*(x) = \frac{1}{2} x^T Q x - \frac{1}{2} x^T A^T \Pi B \left(R + B^T \Pi B\right)^{-1} B^T \Pi A x + \frac{1}{2} x^T A^T \Pi A x = \frac{1}{2} x^T \Pi x$$

Therefore Π satisfies,

$$\Pi = Q - A^T \Pi B \left(R + B^T \Pi B\right)^{-1} B^T \Pi A + A^T \Pi A \tag{4.7}$$

(4.7) is often referred to as algebraic Riccatic equation (ARE).

Remark 4.2.2. *ARE (4.7) has one unique solution if Π is symmetric and positive semidefinite. To solve (4.7), one can either use matrix factorization method (beyond the scope of this course), or iterating the following equation until converge*

$$\begin{aligned} \Pi_0 &= Q \\ \Pi_{k+1} &= Q - A^T \Pi_k B \left(R + B^T \Pi_k B\right)^{-1} B^T \Pi_k A + A^T \Pi_k A \end{aligned}$$

Example 4.2.2 (Infinite LQR Steering Control)**.** *Consider the steering control problem discussed in Example 4.1.1. To regulate the vehicle speed, lateral position, and heading angle back to their nominal value, the following infinite LQR problem is formulated.*

$$\min_{\boldsymbol{u}} \quad \frac{1}{2} \sum_{k=1}^{\infty} \|x_k\|_Q^2 + \frac{1}{2} \sum_{k=0}^{\infty} \|u_k\|_R^2 \qquad \textit{subject to} \quad x_{k+1} = Ax_k + Bu_k, \quad \forall k$$

When

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad R = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$$

Π *can be computed using Matlab's command icare, which is given as*

$$\Pi = \begin{bmatrix} 5.5249 & 0 & 0 \\ 0 & 3.355 & 7.6508 \\ 0 & 7.6508 & 36.0346 \end{bmatrix}$$

Using iterative methods, with 16 iterations, we can have

$$\Pi = \begin{bmatrix} 5.5111 & 0 & 0 \\ 0 & 3.355 & 7.6508 \\ 0 & 7.6508 & 36.0345 \end{bmatrix}$$

which is fairly close to its final value. Furthermore, the control performance is plotted in Fig. 4.3. As can be seen, the states are effectively converging back to 0*, as is in the finite horizon LQR case.*

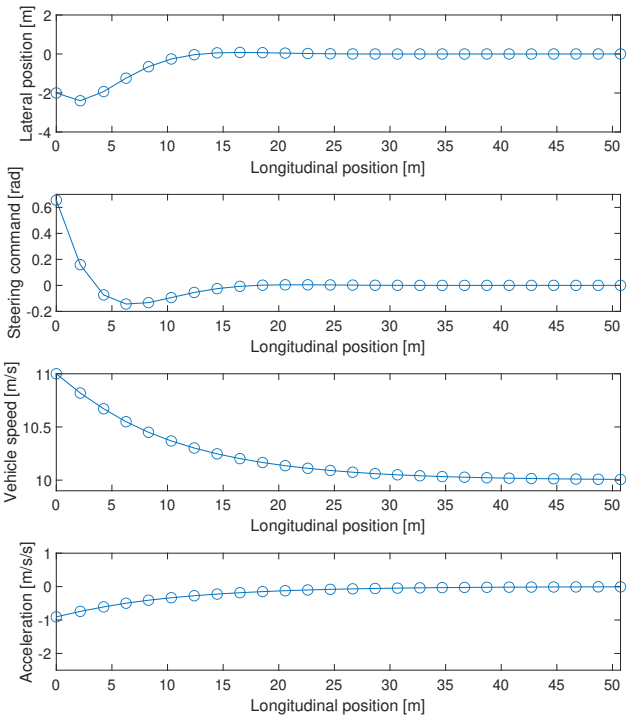

Figure 4.3: Optimal solution for Example 4.2.2.

4.3 Receding Horizon LQR

One of the drawback of the finite horizon LQR discussed earlier is that one needs to calculate the backward Riccatti equations for N times, and stores N gain matrices $K_0, K_1, \ldots, K_{N-1}$ for real time control. This poses significant challenges for online computation and storage. A similar challenge is also present for infinite horizon LQR, where finding Π can be difficult. To address these challenges, the receding horizon LQR can be used. The central idea of receding horizon LQR is that, instead of solving (4.2) for the entire length N, we solve the following smaller problem for each time step t:

$$\min_{\mathbf{u}} \quad \sum_{k=t}^{t+p} \left(\frac{1}{2} \|x_k\|_Q^2 + \frac{1}{2} \|u_k\|_R^2 \right) \qquad \text{subject to} \quad x_{k+1} = Ax_k + Bu_k, \tag{4.8}$$

where $p < N$ and the input sequence is defined as

$$\mathbf{u} = \begin{bmatrix} u_t \\ u_{t+1} \\ \vdots \\ u_{t+p-1} \end{bmatrix}.$$

Note that there is no terminal cost in (4.8). In other words, the receding horizon LQR minimizes p-step-ahead LQR cost, and the optimal solution u_t^* of (4.8) is called *optimal receding horizon control*. Furthermore, at each time step, only u_t^* is implemented by actuator, while at next time step $t+1$, a new LQR of (4.8) is formulated and solved. Fig. 4.4 illustrates the idea of receding horizon LQR control.

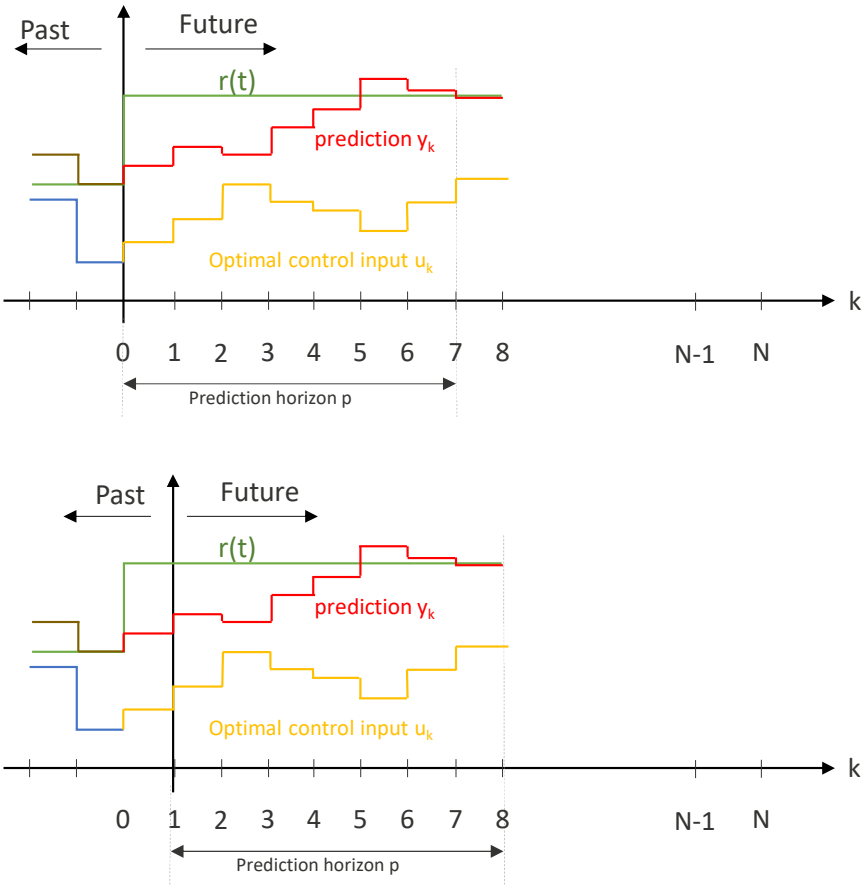

Figure 4.4: Illustration of receding horizon LQR.

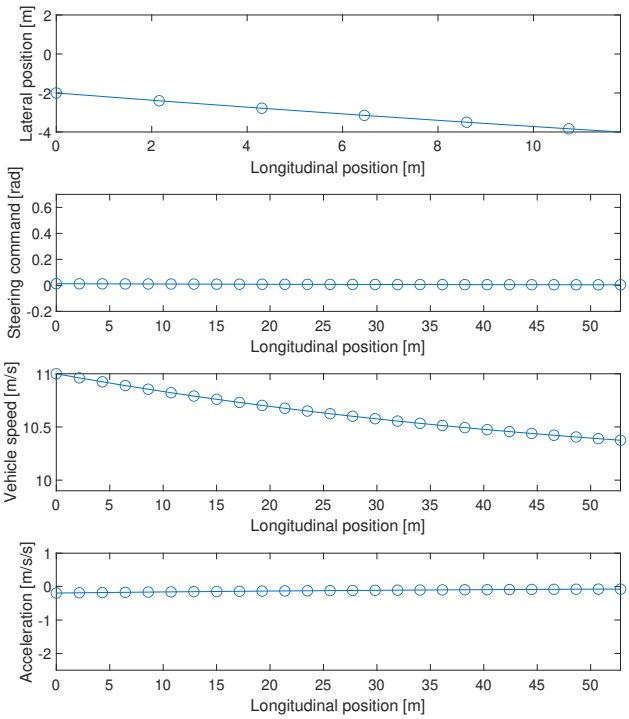

Figure 4.5: Optimal solution for Example 4.3.1, with $p = 2$.

To solve (4.2), the exact recursion (4.4) for finite horizon LQR can be used, with $Q_N = Q$. Since for each time step, the same LQR is solved, then the feedback gain is a constant matrix.

Remark 4.3.1. *Compared to infinite horizon LQR, receding horizon LQR may be easier to solve (depending on p). Furthermore, when p is large, state feedback gain converges to infinite horizon optimal, if the system is controllable.*

Example 4.3.1 (Receding LQR Steering Control)**.** *Consider the steering control problem discussed in Example 4.2.2. Instead of the infinite horizon LQR approach, we use the receding horizon approach. Figures 4.5, 4.6 and 4.7 plot the results for different horizon p. When $p = 2$, control is short-sighted, and is reluctant to make efforts, since in this case it does not see many benefits to make control actions. When p increases, the benefit of making control efforts is obvious to the controller, and the receding LQR can achieve similar performance to that of infinite horizon LQR.*

Furthermroe, Figures 4.8, 4.9 and 4.10 compare control performance of different LQR, namely, finite horizon LQR (f-LQR), infinite horizon LQR (i-LQR), and receding horizon LQR (r-LQR). As can be seen, the horizon p plays a critical role in control performance, and is challenging to tune in practice.

4.4 Matlab Codes

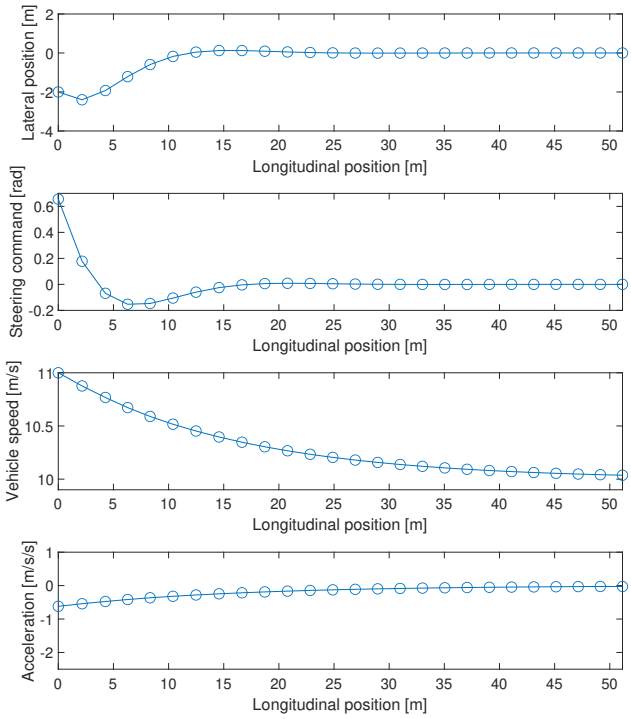

Figure 4.6: Optimal solution for Example 4.3.1 with $p = 5$.

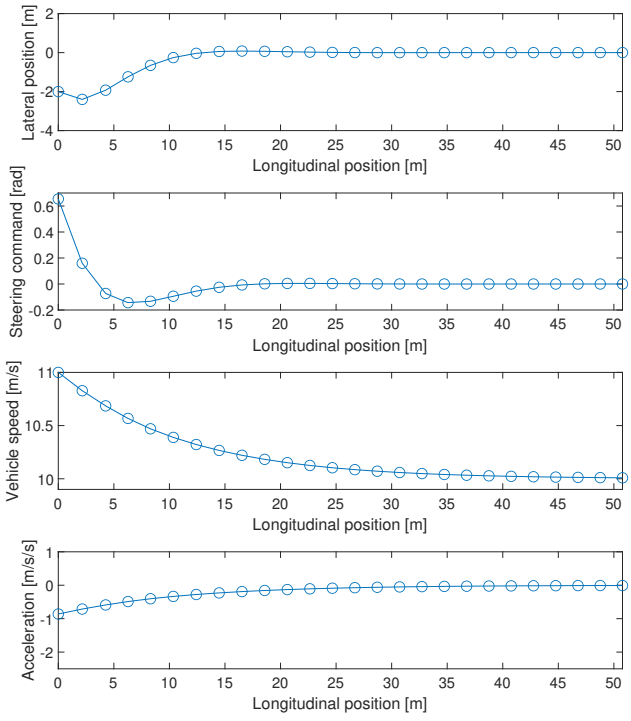

Figure 4.7: Optimal solution for Example 4.3.1 with $p = 10$.

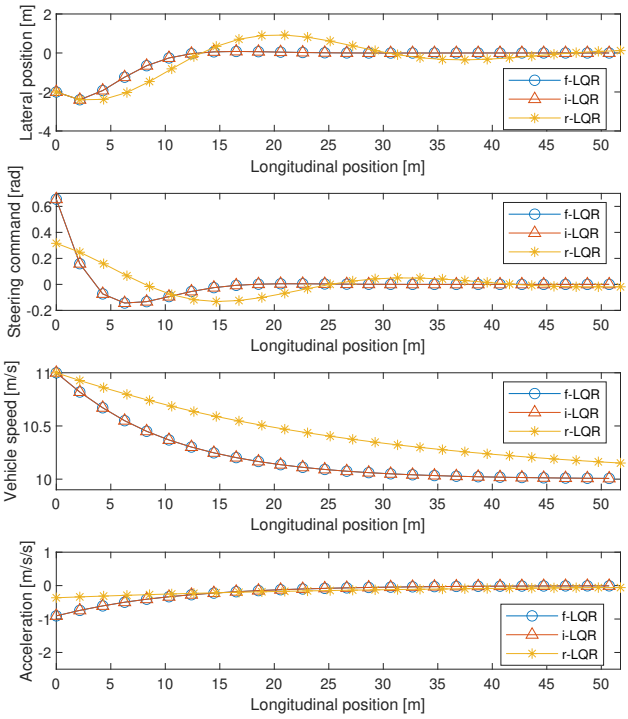

Figure 4.8: Comparison of different LQR, with $p = 3$.

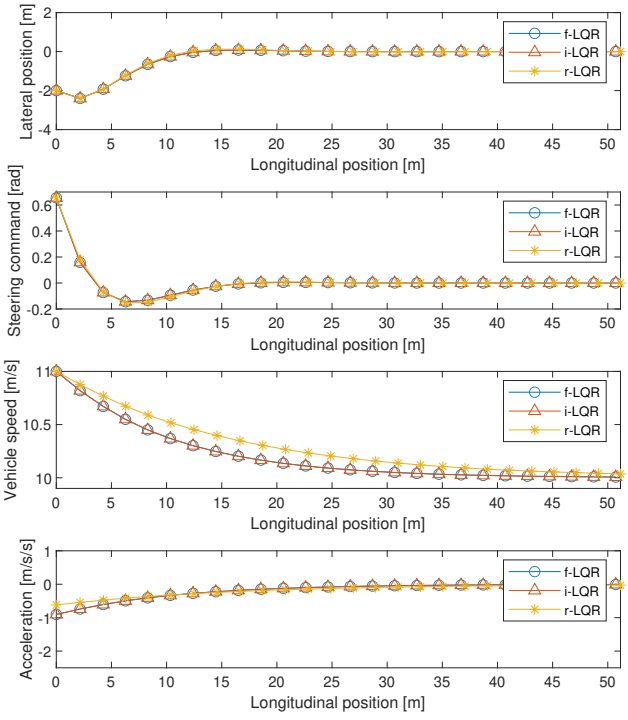

Figure 4.9: Comparison of different LQR, with $p = 5$.

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; x0 = [1;-2;-0.2];
Ts = 0.2; t = 0:Ts:5; N = numel(t);
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; B = [Ts 0;0 0;0 xbar(1)/l*Ts];

Q = diag([1,1,1]); R = diag([0.1 1000]);  QN = Q;

Kopt = zeros(2,3,N);
PI = QN;

for n = N : -1 : 2
    Kopt(:,:,n-1) = Fill here;
    PI = Fill here;
end

uopt = zeros(2,N); x = zeros(3,N); x(:,1) = x0;
for n = 2:N
    uopt(:,n-1) = Fill here;
    x(:,n) = A*x(:,n-1)+B*uopt(:,n-1);
end
uopt(:,n) = uopt(:,n-1); x = x + repmat(xbar,1,N);

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos;

px = cumsum(x(1,:).*cos(x(3,:))*Ts); px=[0 px(1:end-1)]; py = x(2,:);

subplot(4,1,1), plot(px,py,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-4,2])

subplot(4,1,2), plot(px,uopt(2,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.2,0.7])

subplot(4,1,3), plot(px,x(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Vehicle speed [m/s]'),
axis tight, ylim([9.9,11])

subplot(4,1,4), plot(px,uopt(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Acceleration [m/s/s]'),
axis tight, ylim([-2.5,1])
```

Listing 4.1: MATLAB Commands for Example 4.1.1.

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; x0 = [1;-2;-0.2];
Ts = 0.2; t = 0:Ts:5; N = numel(t);
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1];
B = [Ts 0;0 0;0 xbar(1)/l*Ts];

Q = diag([1,1,1]); R = diag([1 10]);

PI = Q;
for n = 1:100
    PI_old = PI; PI = Fill here;
    if norm(PI(:)-PI_old(:))<0.01
        break
    end
end
n

[Pi,Kopt,~] = idare(A,B,Q,R); % can use lqr command as well.

uopt = zeros(2,N); x = zeros(3,N); x(:,1) = x0;
for n = 2:N
    uopt(:,n-1) = -Kopt*x(:,n-1); x(:,n) = A*x(:,n-1)+B*uopt(:,n-1);
end
uopt(:,n) = uopt(:,n-1); x = x + repmat(xbar,1,N);

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos;

px = cumsum(x(1,:).*cos(x(3,:))*Ts); px=[0 px(1:end-1)];
py = x(2,:);

subplot(4,1,1), plot(px,py,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-4,2])

subplot(4,1,2), plot(px,uopt(2,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.2,0.7])

subplot(4,1,3), plot(px,x(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Vehicle speed [m/s]'),
axis tight, ylim([9.9,11])

subplot(4,1,4), plot(px,uopt(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Acceleration [m/s/s]'),
axis tight, ylim([-2.5,1])
```

Listing 4.2: MATLAB Commands for Example 4.2.2.

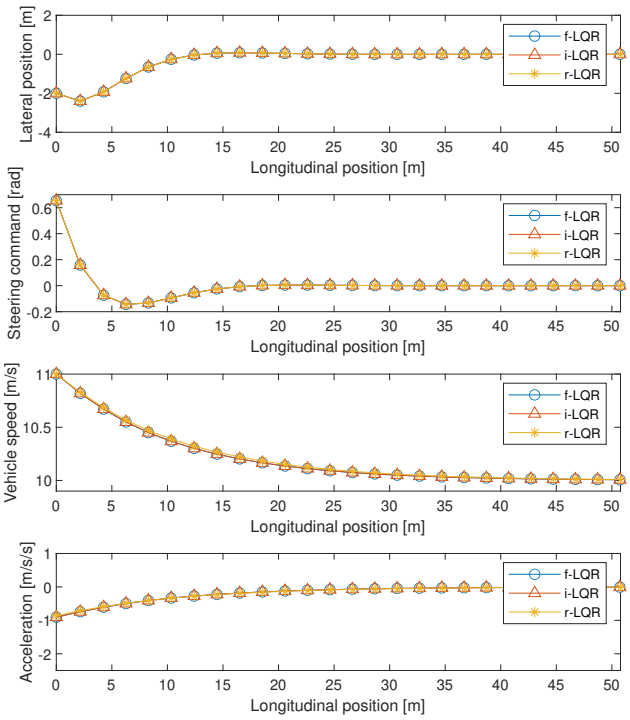

Figure 4.10: Comparison of different LQR, with $p = 10$.

Chapter 5

Linear Model Predictive Control

Similar to receding horizon LQR problem, model predictive control (MPC) solves a *parametric* optimization problem at each time step, formulated as follows:

$$\min_{\mathbf{u}} \quad \sum_{k=t}^{t+p-1} \ell(x_k, u_k) + V_f(x_p) \qquad \text{subject to} \quad \begin{cases} x_{k+1} = f(x_k, u_k), \\ \mathbf{x} \in \mathcal{X}, \quad \mathbf{u} \in \mathcal{U} \\ x(t) = x_0 \end{cases} \tag{5.1}$$

where p is called prediction horizon, and the input sequence and state sequence are defined as

$$\mathbf{u} = \begin{bmatrix} u_t \\ u_{t+1} \\ \vdots \\ u_{t+p-1} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_{t+1} \\ x_{t+2} \\ \vdots \\ x_{t+p} \end{bmatrix}.$$

Problem (5.1) is solved for each time step t, and only the first element of $\mathbf{u}$, i.e., u_t, is implemented by the actuators. Then time advances by 1 time step, a new initial state x_0 is obtained, and MPC solves a new optimization problem (5.1) with updated x_0. Note that a key difference between MPC and LQR is that, (5.1) enforces the input and state constraints $x_k \in \mathcal{X}$ and $u_k \in \mathcal{U}$, which is not treated at all in the case of LQR.

In general, stage and terminal cost $\ell(x, u)$ and $V_f(x)$ can be any arbitrary functions. For this chapter, we limit both $\ell(x, u)$ and $V_f(x)$ to quadratic functions, and restrict the system dynamics $f(x, u)$ and all constraints to be linear. In other words, linear MPC solves the following constrained optimal control problem (OCP) at each time step:

$$\min_{\mathbf{u}} \quad \frac{1}{2} \sum_{k=1}^{p} \|x_k\|_{Q_x}^2 + \frac{1}{2} \sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 \qquad \text{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \text{given } x_0 \\ x_{\min} \le x_k \le x_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \end{cases} \tag{5.2}$$

Note that in (5.2) we drop t from the subscript. Compared (5.2) to (5.1), we have

$$\begin{aligned} \ell(x, u) &= \frac{1}{2} \|x\|_{Q_x}^2 + \frac{1}{2} \|u\|_{Q_u}^2 = \frac{1}{2} x^T Q_x x + \frac{1}{2} u^T Q_u u \\ V_f(x) &= \frac{1}{2} \|x\|_{Q_x}^2 = \frac{1}{2} x^T Q_x x \\ \mathcal{X} &= \{\mathbf{x} \mid x_{\min} \le x_k \le x_{\max}, \forall k\} \\ \mathcal{X} &= \{\mathbf{u} \mid u_{\min} \le u_k \le u_{\max} \quad \& \quad \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max}\} \end{aligned}$$

Example 5.0.1 (MPC Steering Control)**.** *Let's revisit the vehicle steering control problem discussed in Example 4.1.1, where the system dynamics are described by*

$$x_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0.2 & 0 \\ 0 & 0 \\ 0 & \frac{2}{3} \end{bmatrix} u_k = Ax_k + Bu_k$$

Now suppose the initial conditions are specified as $v_0 = 1$, $p_{y,0} = -2$, *and* $\phi_0 = -0.2$, *i.e.,* $x_0 = \begin{bmatrix} 1 & -2 & -0.2 \end{bmatrix}^T$. *To regulate the vehicle speed, lateral position, and heading angle back to their nominal value, the following MPC problem is formulated.*

$$\min_{\boldsymbol{u}} \quad \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2 + \frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 \qquad \textit{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, & \textit{given } x_0 \\ x_{min} \le x_k \le x_{max} \\ u_{min} \le u_k \le u_{max} \end{cases}$$

When

$$Q_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad Q_u = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$$

and

$$x_{min} = \begin{bmatrix} 9 \\ -4 \\ -\infty \end{bmatrix}, \quad x_{max} = \begin{bmatrix} 11 \\ 2 \\ \infty \end{bmatrix}, \quad u_{min} = \begin{bmatrix} -0.2 \\ -2 \end{bmatrix}, \quad u_{max} = \begin{bmatrix} 0.2 \\ 1 \end{bmatrix},$$

the results are plotted in Fig. 5.1. As can be seen, the states are effectively converging back to 0. *Compared to Fig. 4.1 of Example 4.1.1, since there is constraints enforcement by MPC, the steering command are strictly within the limits (note the large initial value, i.e., 0.6 rad or 34 degree in Fig. 4.1.*

5.1 Numerical Solver

This chapter will discuss how the OCP (5.2) is solved in real-time. In particular, (5.2) will first be reformulated into a quadratic programming (QP) problem, which can then be solved by various QP solvers discussed in Chapter 3.4. Two ways can be used to formulate QP problem, dense formulation, and sparse formulation.

5.1.1 Sparse Formulation

Consider OCP (5.2), which can be equivalently written as

$$\min_{\mathbf{u},\mathbf{x}} \quad \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2 + \frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 \qquad \text{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, & \text{given } x_0 \\ x_{\min} \le x_k \le x_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \end{cases} \tag{5.3}$$

where the only difference compared to (5.2) is that $\mathbf{x}$ is also treated as an optimization variable. This is simply a difference in representation, and OCP (5.3) is in fact mathematically equivalent

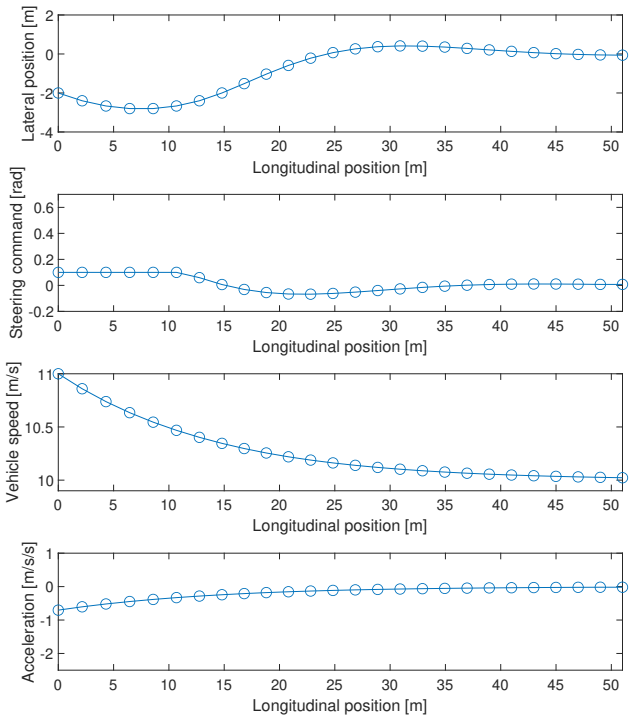

Figure 5.1: Control performance for Example 5.0.1.

to (5.2). Define the optimization variable to be

$$\mathbf{z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} u_0 \\ \vdots \\ u_{p-1} \\ x_1 \\ \vdots \\ x_p \end{bmatrix}.$$

Denote the cost function of (5.3) as

$$J = \underbrace{\frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2}_{J_x} + \underbrace{\frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2}_{J_u}$$

Then we have

$$J_x = \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2 = \frac{1}{2}\sum_{k=1}^{p} x_k^T Q_x x_k = \frac{1}{2}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}^T \underbrace{\begin{bmatrix} Q_x & 0 & 0 & \cdots & 0 \\ 0 & Q_x & 0 & \cdots & 0 \\ 0 & 0 & Q_x & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q_x \end{bmatrix}}_{M_x} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \frac{1}{2}\mathbf{x}^T M_x \mathbf{x}$$

$$J_u = \frac{1}{2}\sum_{k=0}^{p-1}\|u_k\|_{Q_u}^2 = \frac{1}{2}\sum_{k=0}^{p-1}u_k^T Q_u u_k = \frac{1}{2}\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{p-1}\end{bmatrix}^T \underbrace{\begin{bmatrix} Q_u & 0 & 0 & \cdots & 0 \\ 0 & Q_u & 0 & \cdots & 0 \\ 0 & 0 & Q_u & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q_u \end{bmatrix}}_{M_u} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{p-1}\end{bmatrix} = \frac{1}{2}\mathbf{u}^T M_u \mathbf{u}$$

$$J = J_x + J_u = \frac{1}{2}\begin{bmatrix}\mathbf{u} \\ \mathbf{x}\end{bmatrix}^T \begin{bmatrix} M_u & 0 \\ 0 & M_x\end{bmatrix}\begin{bmatrix}\mathbf{u} \\ \mathbf{x}\end{bmatrix} \tag{5.4}$$

The system dynamic $x_{k+1} = Ax_k + Bu_k$ can be expressed in matrix form as:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_p\end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ A & 0 & 0 & \cdots & 0 \\ 0 & A & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}}_{M_A}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_p\end{bmatrix} + \underbrace{\begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ 0 & B & 0 & \cdots & 0 \\ 0 & 0 & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & B \end{bmatrix}}_{M_B}\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{p-1}\end{bmatrix} + \underbrace{\begin{bmatrix} Ax_0 \\ 0 \\ 0 \\ \vdots \\ 0\end{bmatrix}}_{r_0}$$

which is equivalent to

$$(I - M_A)\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_p\end{bmatrix} - M_B\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{p-1}\end{bmatrix} = r_0$$

or

$$\begin{aligned}(I - M_A)\mathbf{x} - M_B\mathbf{u} &= r_0 \\ \begin{bmatrix} -M_B & I - M_A\end{bmatrix}\begin{bmatrix}\mathbf{u} \\ \mathbf{x}\end{bmatrix} &= r_0\end{aligned} \tag{5.5}$$

The state constraints $x_{\min} \le x_k \le x_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix} x_{\min} \\ x_{\min} \\ \vdots \\ x_{\min}\end{bmatrix} \le \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p\end{bmatrix} \le \begin{bmatrix} x_{\max} \\ x_{\max} \\ \vdots \\ x_{\max}\end{bmatrix}$$

or equivalently

$$\mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{x}_{\max} \tag{5.6}$$

Similarly the input constraints $u_{\min} \le u_k \le u_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix} u_{\min} \\ u_{\min} \\ \vdots \\ u_{\min}\end{bmatrix} \le \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{p-1}\end{bmatrix} \le \begin{bmatrix} u_{\max} \\ u_{\max} \\ \vdots \\ u_{\max}\end{bmatrix}$$

or equivalently

$$\mathbf{u}_{\min} \le \mathbf{u} \le \mathbf{u}_{\max} \tag{5.7}$$

Lastly, the input rate constraint $\Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix}\Delta_{\min}\\ \Delta_{\min}\\ \Delta_{\min}\\ \vdots\\ \Delta_{\min}\end{bmatrix} \le \underbrace{\begin{bmatrix}1 & 0 & 0 & \cdots & 0 & 0\\ -1 & 1 & 0 & \cdots & 0 & 0\\ 0 & -1 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & \cdots & -1 & 1\end{bmatrix}}_{D} \begin{bmatrix}u_0\\ u_1\\ u_2\\ \vdots\\ u_{p-1}\end{bmatrix} + \underbrace{\begin{bmatrix}-u_{-1}\\ 0\\ 0\\ \vdots\\ 0\end{bmatrix}}_{r_u} \le \begin{bmatrix}\Delta_{\max}\\ \Delta_{\max}\\ \Delta_{\max}\\ \vdots\\ \Delta_{\max}\end{bmatrix}$$

Therefore,

$$\boldsymbol{\Delta}_{\min} - r_u \le D\mathbf{u} \le \boldsymbol{\Delta}_{\max} - r_u \tag{5.8}$$

Putting (5.4), (5.5), (5.6), (5.7) and (5.8) together, we have the following QP formulation for (5.3):

$$\min_{\mathbf{z}} \quad \frac{1}{2}\mathbf{z}^T \begin{bmatrix}M_u & 0\\ 0 & M_x\end{bmatrix}\mathbf{z} \qquad \text{subject to} \quad \begin{cases}\begin{bmatrix}-M_B & I - M_A\end{bmatrix}\mathbf{z} = r_0\\ \mathbf{z} \le \begin{bmatrix}\mathbf{u}_{\max}\\ \mathbf{x}_{\max}\end{bmatrix}\\ -\mathbf{z} \le -\begin{bmatrix}\mathbf{u}_{\min}\\ \mathbf{x}_{\min}\end{bmatrix}\\ \begin{bmatrix}D & 0\end{bmatrix}\mathbf{z} \le \boldsymbol{\Delta}_{\max} - r_u\\ -\begin{bmatrix}D & 0\end{bmatrix}\mathbf{z} \le -\boldsymbol{\Delta}_{\min} + r_u\end{cases} \tag{5.9}$$

Example 5.1.1 (Sparse QP Formulation for MPC)**.** *Consider the following linear system*

$$x_{k+1} = \begin{bmatrix}0.7 & 0.1\\ 0 & 0.1\end{bmatrix} x_k + \begin{bmatrix}1\\ 0\end{bmatrix} u_k$$

Consider the MPC formulation of (5.3) with $p = 2$, $Q_x = \begin{bmatrix}2 & 0\\ 0 & 1\end{bmatrix}$, $Q_u = 3$, $x_{min} = \begin{bmatrix}-1 & -1\end{bmatrix}^T$, $x_{max} = \begin{bmatrix}5 & 5\end{bmatrix}^T$, $u_{min} = -2$, $u_{max} = 3$, $\Delta_{min} = -0.1$ *and* $\Delta_{max} = 0.1$. *At a given time step, assume* $x_0 = \begin{bmatrix}0.2\\ -0.1\end{bmatrix}$ *and* $u_{-1} = 2$. *Find its equivalent sparse QP formulation.*

According to (5.4), (5.5), (5.6), (5.7) and (5.8), we have

$$M_u = \begin{bmatrix}Q_u & 0\\ 0 & Q_u\end{bmatrix} = \begin{bmatrix}3 & 0\\ 0 & 3\end{bmatrix}$$

$$M_x = \begin{bmatrix}Q_x & 0\\ 0 & Q_x\end{bmatrix} = \begin{bmatrix}2 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 2 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$M_A = \begin{bmatrix}0 & 0\\ A & 0\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0.7 & 0.1 & 0 & 0\\ 0 & 0.1 & 0 & 0\end{bmatrix}$$

$$M_B = \begin{bmatrix}B & 0\\ 0 & B\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix}$$

$$r_0 = \begin{bmatrix} Ax_0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.13 \\ -0.01 \\ 0 \\ 0 \end{bmatrix}$$
$$r_u = \begin{bmatrix} -u_{-1} \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$
$$D = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}.$$

Denote

$$\boldsymbol{z} = \begin{bmatrix} u_0 \\ u_1 \\ x_1 \\ x_2. \end{bmatrix}$$

Therefore the corresponding sparse QP formulation is given as

$$\min_{\boldsymbol{z}} \quad \frac{1}{2}\boldsymbol{z}^T \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{z}$$

$$\textit{subject to} \quad \begin{cases} \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & -0.7 & -0.1 & 1 & 0 \\ 0 & 0 & 0 & -0.1 & 0 & 1 \end{bmatrix} \boldsymbol{z} = \begin{bmatrix} 0.13 \\ -0.01 \\ 0 \\ 0 \end{bmatrix} \\ \boldsymbol{z} \le \begin{bmatrix} 3 & 3 & 5 & 5 & 5 & 5 \end{bmatrix}^T \\ -\boldsymbol{z} \le \begin{bmatrix} 2 & 2 & 1 & 1 & 1 & 1 \end{bmatrix}^T \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{z} \le \begin{bmatrix} 2.1 \\ 0.1 \end{bmatrix} \\ \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{z} \le \begin{bmatrix} -1.9 \\ 0.1 \end{bmatrix} \end{cases}$$

Or equivalently

$$\min_{\boldsymbol{z}} \quad \frac{1}{2}\boldsymbol{z}^T \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{z}$$

$$
\textit{subject to} \quad \begin{cases} \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & -0.7 & -0.1 & 1 & 0 \\ 0 & 0 & 0 & -0.1 & 0 & 1 \end{bmatrix} \boldsymbol{z} = \begin{bmatrix} 0.13 \\ -0.01 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{z} \leq \begin{bmatrix} 3 \\ 3 \\ 5 \\ 5 \\ 5 \\ 5 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 2.1 \\ 0.1 \\ -1.9 \\ 0.1 \end{bmatrix} \end{cases}
$$

As shown in Matlab Code Listing 5.2, solving the above QP yields the same solution as using Matlab's MPC toolbox.

5.1.2 Dense Formulation

In the dense formulation, the optimization variable is $\mathbf{u} = \begin{bmatrix} u_0 & u_1 & \cdots & u_{p-1} \end{bmatrix}^T$, and the relationship between $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_p \end{bmatrix}^T$ and $\mathbf{u}$ is explicitly expressed in the QP matrices. To derive a relationship between x_k and the control inputs u_0, u_1, $\cdots$, u_{k-1}, we have,

$$
\begin{aligned}
x_1 &= Ax_0 + Bu_0 \\
x_2 &= Ax_1 + Bu_1 = A(Ax_0 + Bu_0) + Bu_1 = A^2x_0 + ABu_0 + Bu_1 \\
x_3 &= Ax_2 + Bu_2 = A(Ax_1 + Bu_1) + Bu_2 = A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \\
&\vdots \\
x_k &= Ax_{k-1} + Bu_{k-1} = A(Ax_{k-2} + Bu_{k-2}) + Bu_{k-1} \\
&= A^2x_{k-2} + ABu_{k-2} + Bu_{k-1} = A^2(Ax_{k-3} + Bu_{k-3}) + ABu_{k-2} + Bu_{k-1} \\
&= A^3x_{k-3} + A^2Bu_{k-3} + ABu_{k-2} + Bu_{k-1} \\
&\vdots \\
&= A^kx_0 + \sum_{n=1}^{k} A^{n-1}Bu_{k-n}
\end{aligned}
$$

Putting it into matrix form, we have

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_p \end{bmatrix} = \underbrace{\begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ A^2B & AB & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{p-1}B & A^{p-2}B & A^{p-3}B & \cdots & B \end{bmatrix}}_{M_{AB}} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{p-1} \end{bmatrix} + \underbrace{\begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^p \end{bmatrix}}_{M_{Ak}} x_0 \tag{5.10}$$

Or equivalently

$$\mathbf{x} = M_{AB}\mathbf{u} + M_{Ak}x_0 \tag{5.11}$$

Consider the OCP (5.2), which can now be equivalently represented as

$$\min_{\mathbf{u}} \quad \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2 + \frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 \qquad \text{subject to} \quad \begin{cases} \mathbf{x} = M_{AB}\mathbf{u} + M_{Ak}x_0 \\ x_{\min} \le x_k \le x_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \end{cases} \tag{5.12}$$

Denote the cost function of (5.12) as

$$J = \underbrace{\frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2}_{J_x} + \underbrace{\frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2}_{J_u}$$

Then we have

$$J_x = \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2 = \frac{1}{2}\sum_{k=1}^{p} x_k^T Q_x x_k = \frac{1}{2}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}^T \underbrace{\begin{bmatrix} Q_x & 0 & 0 & \cdots & 0 \\ 0 & Q_x & 0 & \cdots & 0 \\ 0 & 0 & Q_x & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q_x \end{bmatrix}}_{M_x} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \frac{1}{2}\mathbf{x}^T M_x \mathbf{x}$$

$$= \frac{1}{2}(M_{AB}\mathbf{u} + M_{Ak}x_0)^T M_x (M_{AB}\mathbf{u} + M_{Ak}x_0)$$

$$= \frac{1}{2}\mathbf{u}^T M_{AB}^T M_x M_{AB}\mathbf{u} + x_0^T M_{Ak}^T M_x M_{AB}\mathbf{u} + \underbrace{\frac{1}{2}x_0^T M_{Ak}^T M_x M_{Ak} x_0}_{\text{Independent of } \mathbf{u}}$$

Ignoring the term independent of $\mathbf{u}$, we have

$$J_x = \frac{1}{2}\mathbf{u}^T \underbrace{M_{AB}^T M_x M_{AB}}_{H_x}\mathbf{u} + \underbrace{x_0^T M_{Ak}^T M_x M_{AB}}_{f_x}\mathbf{u} \tag{5.13}$$

where

$$H_x = M_{AB}^T M_x M_{AB}$$

$$= \begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ A^2B & AB & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{p-1}B & A^{p-2}B & A^{p-3}B & \cdots & B \end{bmatrix}^T \begin{bmatrix} Q_xB & 0 & 0 & \cdots & 0 \\ Q_xAB & Q_xB & 0 & \cdots & 0 \\ Q_xA^2B & Q_xAB & Q_xB & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Q_xA^{p-1}B & Q_xA^{p-2}B & Q_xA^{p-3}B & \cdots & Q_xB \end{bmatrix}$$

$$f_x = x_0^T M_{Ak}^T M_x M_{AB} = x_0^T M_{Ak}^T \begin{bmatrix} Q_xB & 0 & 0 & \cdots & 0 \\ Q_xAB & Q_xB & 0 & \cdots & 0 \\ Q_xA^2B & Q_xAB & Q_xB & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Q_xA^{p-1}B & Q_xA^{p-2}B & Q_xA^{p-3}B & \cdots & Q_xB \end{bmatrix}$$

Similarly to the case of sparse formulation, we have

$$J_u = \frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 = \frac{1}{2}\sum_{k=0}^{p-1} u_k^T Q_u u_k = \frac{1}{2} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{p-1} \end{bmatrix}^T \underbrace{\begin{bmatrix} Q_u & 0 & 0 & \cdots & 0 \\ 0 & Q_u & 0 & \cdots & 0 \\ 0 & 0 & Q_u & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q_u \end{bmatrix}}_{M_u} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{p-1} \end{bmatrix} = \frac{1}{2}\mathbf{u}^T M_u \mathbf{u} \tag{5.14}$$

Putting (5.13) and (5.14) together, we have

$$J = J_x + J_u = \frac{1}{2}\mathbf{u}^T \left(H_x + M_u\right)\mathbf{u} + f_x\mathbf{u} \tag{5.15}$$

The state constraints $x_{\min} \le x_k \le x_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix} x_{\min} \\ x_{\min} \\ \vdots \\ x_{\min} \end{bmatrix} \le \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \le \begin{bmatrix} x_{\max} \\ x_{\max} \\ \vdots \\ x_{\max} \end{bmatrix}$$

or equivalently

$$\mathbf{x}_{\min} \le M_{AB}\mathbf{u} + M_{Ak}x_0 \le \mathbf{x}_{\max}$$

Therefore we have

$$\mathbf{x}_{\min} - M_{Ak}x_0 \le M_{AB}\mathbf{u} \le \mathbf{x}_{\max} - M_{Ak}x_0 \tag{5.16}$$

Similarly the input constraints $u_{\min} \le u_k \le u_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix} u_{\min} \\ u_{\min} \\ \vdots \\ u_{\min} \end{bmatrix} \le \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{p-1} \end{bmatrix} \le \begin{bmatrix} u_{\max} \\ u_{\max} \\ \vdots \\ u_{\max} \end{bmatrix}$$

or equivalently

$$\mathbf{u}_{\min} \le \mathbf{u} \le \mathbf{u}_{\max} \tag{5.17}$$

Lastly, the input rate constraint $\Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix}\Delta_{\min}\\ \Delta_{\min}\\ \Delta_{\min}\\ \vdots\\ \Delta_{\min}\end{bmatrix} \le \underbrace{\begin{bmatrix}1 & 0 & 0 & \cdots & 0 & 0\\ -1 & 1 & 0 & \cdots & 0 & 0\\ 0 & -1 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & \cdots & -1 & 1\end{bmatrix}}_{D} \begin{bmatrix}u_0\\ u_1\\ u_2\\ \vdots\\ u_{p-1}\end{bmatrix} + \underbrace{\begin{bmatrix}-u_{-1}\\ 0\\ 0\\ \vdots\\ 0\end{bmatrix}}_{r_u} \le \begin{bmatrix}\Delta_{\max}\\ \Delta_{\max}\\ \Delta_{\max}\\ \vdots\\ \Delta_{\max}\end{bmatrix}$$

Therefore,

$$\boldsymbol{\Delta}_{\min} - r_u \le D\mathbf{u} \le \boldsymbol{\Delta}_{\max} - r_u \tag{5.18}$$

Putting (5.15), (5.16), (5.17) and (5.18) together, we have the following QP formulation for (5.12):

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T\left(H_x + M_u\right)\mathbf{u} + f_x\mathbf{u} \qquad \text{subject to} \quad \begin{cases} M_{AB}\mathbf{u} \le \mathbf{x}_{\max} - M_{Ak}x_0\\ -M_{AB}\mathbf{u} \le -\mathbf{x}_{\min} + M_{Ak}x_0\\ \mathbf{u} \le \mathbf{u}_{\max}\\ -\mathbf{u} \le -\mathbf{u}_{\min}\\ D\mathbf{u} \le \boldsymbol{\Delta}_{\max} - r_u\\ -D\mathbf{u} \le -\boldsymbol{\Delta}_{\min} + r_u \end{cases} \tag{5.19}$$

Example 5.1.2 (Dense QP Formulation for MPC)**.** *Consider the MPC problem studied in Example 5.1.1. Find its equivalent dense QP formulation.*

According to (5.15), (5.16), (5.17) and (5.18), we have

$$M_u = \begin{bmatrix}Q_u & 0\\ 0 & Q_u\end{bmatrix} = \begin{bmatrix}3 & 0\\ 0 & 3\end{bmatrix}$$

$$M_x = \begin{bmatrix}Q_x & 0\\ 0 & Q_x\end{bmatrix} = \begin{bmatrix}2 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 2 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$M_{AB} = \begin{bmatrix}B & 0\\ AB & B\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 0\\ 0.7 & 1\\ 0 & 0\end{bmatrix}$$

$$M_xM_{AB} = \begin{bmatrix}2 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 2 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\\ 0.7 & 1\\ 0 & 0\end{bmatrix} = \begin{bmatrix}2 & 0\\ 0 & 0\\ 1.4 & 2\\ 0 & 0\end{bmatrix}$$

$$H_x = M_{AB}^TM_xM_{AB} = \begin{bmatrix}1 & 0 & 0.7 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}2 & 0\\ 0 & 0\\ 1.4 & 2\\ 0 & 0\end{bmatrix} = \begin{bmatrix}2.98 & 1.4\\ 1.4 & 2\end{bmatrix}$$

$$M_{Ak}x_0 = \begin{bmatrix}A\\ A^2\end{bmatrix}x_0 = \begin{bmatrix}0.7 & 0.1\\ 0 & 0.1\\ 0.49 & 0.08\\ 0 & 0.01\end{bmatrix}\begin{bmatrix}0.2\\ -0.1\end{bmatrix} = \begin{bmatrix}0.13\\ -0.01\\ 0.09\\ -0.001\end{bmatrix}$$

$$f_x = x_0^T M_{Ak}^T M_x M_{AB} = \begin{bmatrix} 0.13 & -0.01 & 0.09 & -0.001 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 1.4 & 2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.386 & 0.18 \end{bmatrix}$$

$$r_u = \begin{bmatrix} -u_{-1} \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}.$$

Therefore the corresponding dense QP formulation is given as

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \frac{1}{2}\boldsymbol{u}^T \begin{bmatrix} 5.98 & 1.4 \\ 1.4 & 5 \end{bmatrix} \boldsymbol{u} + \begin{bmatrix} 0.386 & 0.18 \end{bmatrix} \boldsymbol{u} \\ \textit{subject to} \quad & \begin{cases} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0.7 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{u} \leq \begin{bmatrix} 4.87 & 5.01 & 4.91 & 5.001 \end{bmatrix}^T \\ \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ -0.7 & -1 \\ 0 & 0 \end{bmatrix} \boldsymbol{u} \leq \begin{bmatrix} 1.13 & 0.99 & 1.09 & 0.999 \end{bmatrix}^T \\ \boldsymbol{u} \leq \begin{bmatrix} 3 \\ 3 \end{bmatrix} \\ -\boldsymbol{u} \leq \begin{bmatrix} 2 \\ 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \boldsymbol{u} \leq \begin{bmatrix} 2.1 \\ 0.1 \end{bmatrix} \\ \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \boldsymbol{u} \leq \begin{bmatrix} -1.9 \\ 0.1 \end{bmatrix} \end{cases} \end{aligned}$$

Or equivalently

$$\min_{\boldsymbol{z}} \quad \frac{1}{2}\boldsymbol{u}^T \begin{bmatrix} 5.98 & 1.4 \\ 1.4 & 5 \end{bmatrix} \boldsymbol{u} + \begin{bmatrix} 0.386 & 0.18 \end{bmatrix} \boldsymbol{u}$$

$$\textit{subject to} \quad \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0.7 & 1 \\ 0 & 0 \\ -1 & 0 \\ 0 & 0 \\ -0.7 & -1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ -1 & 1 \\ -1 & 0 \\ 1 & -1 \end{bmatrix} \boldsymbol{u} \leq \begin{bmatrix} 4.87 \\ 5.01 \\ 4.91 \\ 5.001 \\ 1.13 \\ 0.99 \\ 1.09 \\ 0.999 \\ 3 \\ 3 \\ 2 \\ 2 \\ 2.1 \\ 0.1 \\ -1.9 \\ 0.1 \end{bmatrix}$$

As shown in Matlab Code Listing 5.3, solving the above QP yields the same solution as using Matlab's MPC toolbox.

Compare (5.9) and (5.19), or compare Example 5.1.1 and Example 5.1.2, it is not hard to see why (5.3) is called sparse formulation and (5.12) dense.

5.1.3 Real-time Implementation

For real-time implementation, the following considerations are important when deciding whether sparse or dense formulation should be used.

Let n_x and n_u denote the number of states and inputs, respectively. Then the sparse formulation has $(n_x + n_u)p$ optimization variables with a $(n_x + n_u)p \times (n_x + n_u)p$ Hessian matrix. Furthermore, sparse formulation has $n_x * p$ equality constraints and $2n_xp + 4n_up$ inequality constraints, out of which $2(n_x + n_u)p + 2$ are bound constraints. On the other hand, the dense formulation has much smaller QP problem, with only n_up optimization variables and $2n_xp + 4n_up$ inequality constraints, out of which $2n_up + 2$ are bound constraints.

In other words, sparse formulation has more optimization variables, but slightly less complicated constraints. Therefore, solving sparse MPC QP requires more time. However, converting OCP (5.3) to (5.9) is a much simpler process than that of converting (5.12) to (5.19). Lastly, sparse formulation not only yield optimal control sequence $\mathbf{u}^*$, but also the optimal state sequence $\mathbf{x}^*$, while dense formulation only generate $\mathbf{u}^*$.

For real-time implementation, it is also very important to ensure the feasibility of OCP (5.2). Since the input constraints in (5.2) are all bound constraints, they can be carefully calibrated to ensure feasibility. However, the state constraints may not always be feasible. To see this, consider Example 5.1.1 but with $x_{\max} = \begin{bmatrix} 2 & 2 \end{bmatrix}^T$. In this case, the resulting QP is as follows.

$$\min_{\mathbf{z}} \quad \frac{1}{2}\mathbf{u}^T \begin{bmatrix} 5.98 & 1.4 \\ 1.4 & 5 \end{bmatrix} \mathbf{u} + \begin{bmatrix} 0.386 & 0.18 \end{bmatrix} \mathbf{u}$$

$$\text{subject to} \quad \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0.7 & 1 \\ 0 & 0 \\ -1 & 0 \\ 0 & 0 \\ -0.7 & -1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ -1 & 1 \\ -1 & 0 \\ 1 & -1 \end{bmatrix} \mathbf{u} \leq \begin{bmatrix} 1.87 \\ 2.01 \\ 1.91 \\ 2.001 \\ 1.13 \\ 0.99 \\ 1.09 \\ 0.999 \\ 3 \\ 3 \\ 2 \\ 2 \\ 2.1 \\ 0.1 \\ -1.9 \\ 0.1 \end{bmatrix}$$

Then one can verify that the resulting QP is in fact infeasible. In fact, the first constraint requires $u_1 \leq 1.87$ while the second from last constraint requires $-u_1 \leq -1.9$ which is equivalent to $u_1 \geq 1.9$, hence creating conflicts. Such infeasibility can cause issue in real-time, as there may not be any solution for actuators to implement. To address this, one trick is to introduce a slack variable and make all state constraints soft. Let ϵ be an slack variable, and revise (5.2) as follows.

$$\min_{\mathbf{u},\epsilon} \quad \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x}^2 + \frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 + \frac{1}{2}\rho_\epsilon \epsilon^2 \quad \text{subject to} \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \text{given } x_0 \\ x_{\min} - \epsilon \leq x_k \leq x_{\max} + \epsilon \\ u_{\min} \leq u_k \leq u_{\max} \\ \Delta_{\min} \leq u_k - u_{k-1} \leq \Delta_{\max} \end{cases} \tag{5.20}$$

When (5.2) is feasible, then (5.20) is equivalent to (5.2), since $\epsilon = 0$ will be the optimal solution of (5.20). When (5.2) is infeasible, then (5.20) would still be feasible, outputting a solution that violates the hard state constraints, where the balancing between constraint violations and optimality (as measured by the first two terms in cost function) is achieved by ρ_ϵ. The state constraints in (5.20) are often referred to as "soft constraints", while those in (5.2) referred to as "hard constraints".

Example 5.1.3 (Infeasible OCP)**.** *Let's revisit Example 5.1.1, but with* $x_{max} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$*. In this case the OCP with hard state constraints is infeasible, but that with soft state constraints is feasible. Using the code in Listing 5.3, but change xmax to 2, it will return a valid solution but violate certain constraint.*

5.2 MPC Examples

The OCP (5.2) is a simplest case of the generic OCP (5.1). We have use it thus far to illustrate the MPC solver due to its simplicity of notation. In this subsection, we will discuss a few more cases of (5.1) that are slightly more complicated than (5.2), and of course, are more practical.

5.2.1 Output Tracking MPC

Recall that (5.2) is also called a regulator, since it regulates both input and state to origin. In many practical use cases, it is rather desirable to track (instead of regulate) the output (instead of input) to certain reference signals. This leads to the following OCP for each time step:

$$\min_{\mathbf{u}} \quad \frac{1}{2}\sum_{k=1}^{p}\|y_k - y_k^r\|_{Q_y}^2 + \frac{1}{2}\sum_{k=0}^{p-1}\|u_k - u_k^r\|_{Q_u}^2 \qquad \text{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \text{given } x_0 \\ y_k = Cx_k \\ y_{\min} \le y_k \le y_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \end{cases} \tag{5.21}$$

Note that here y_k^r is called output reference and u_k^r is called input reference. In many cases, u_k^r is not available, and one can simply set $u_k^r = 0$. Define the optimization variable to be

$$\mathbf{z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} u_0 \\ \vdots \\ u_{p-1} \\ x_1 \\ \vdots \\ x_p \end{bmatrix}.$$

We first notice that

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} = \underbrace{\begin{bmatrix} C & 0 & \cdots & 0 \\ 0 & C & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C \end{bmatrix}}_{M_C} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = M_c\mathbf{x}$$

Denote the cost function of (5.21) as

$$J = \underbrace{\frac{1}{2}\sum_{k=1}^{p}\|y_k - y_k^r\|_{Q_y}^2}_{J_y} + \underbrace{\frac{1}{2}\sum_{k=0}^{p-1}\|u_k - u_k^r\|_{Q_u}^2}_{J_u}$$

Then we have

$$\begin{aligned} J_y =& \frac{1}{2}\sum_{k=1}^{p}\|y_k - y_k^r\|_{Q_y}^2 = \frac{1}{2}\sum_{k=1}^{p}(y_k - y_k^r)^T Q_x (y_k - y_k^r) \\ =& \frac{1}{2}\begin{bmatrix} y_1 - y_1^r \\ y_2 - y_2^r \\ \vdots \\ y_p - y_p^r \end{bmatrix}^T \underbrace{\begin{bmatrix} Q_y & 0 & 0 & \cdots & 0 \\ 0 & Q_y & 0 & \cdots & 0 \\ 0 & 0 & Q_y & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q_y \end{bmatrix}}_{M_y} \begin{bmatrix} y_1 - y_1^r \\ y_2 - y_2^r \\ \vdots \\ y_p - y_p^r \end{bmatrix} = \frac{1}{2}\left(\mathbf{y} - \mathbf{y}^r\right)^T M_y \left(\mathbf{y} - \mathbf{y}^r\right) \end{aligned}$$

$$=\frac{1}{2}\mathbf{y}^T M_y \mathbf{y} - (\mathbf{y}^r)^T M_y \mathbf{y} + \underbrace{\frac{1}{2}(\mathbf{y}^r)^T M_y \mathbf{y}^r}_{\text{Independent of } \mathbf{u}}$$

Therefore, we have

$$J_y = \frac{1}{2}\mathbf{y}^T M_y \mathbf{y} - (\mathbf{y}^r)^T M_y \mathbf{y} = \frac{1}{2}\mathbf{x}^T M_c^T M_y M_c \mathbf{x} - (\mathbf{y}^r)^T M_y M_c \mathbf{x}$$

The second term in the cost function can be calculated as

$$\begin{aligned} J_u =& \frac{1}{2}\sum_{k=0}^{p-1} \|u_k - u_k^r\|_{Q_u}^2 = \frac{1}{2}\sum_{k=0}^{p-1}(u_k - u_k^r)^T Q_u (u_k - u_k^r) \\ =& \frac{1}{2}\begin{bmatrix} u_0 - u_0^r \\ u_1 - u_1^r \\ \vdots \\ u_{p-1} - u_{p-1}^r \end{bmatrix}^T \underbrace{\begin{bmatrix} Q_u & 0 & 0 & \cdots & 0 \\ 0 & Q_u & 0 & \cdots & 0 \\ 0 & 0 & Q_u & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q_u \end{bmatrix}}_{M_u} \begin{bmatrix} u_0 - u_0^r \\ u_1 - u_1^r \\ \vdots \\ u_{p-1} - u_{p-1}^r \end{bmatrix} = \frac{1}{2}\left(\mathbf{u} - \mathbf{u}^r\right)^T M_u \left(\mathbf{u} - \mathbf{u}^r\right) \\ =& \frac{1}{2}\mathbf{u}^T M_u \mathbf{u} - (\mathbf{u}^r)^T M_u \mathbf{u} + \underbrace{\frac{1}{2}(\mathbf{u}^r)^T M_u \mathbf{u}^r}_{\text{Independent of } \mathbf{u}} \end{aligned}$$

Therefore, we have

$$J_u = \frac{1}{2}\mathbf{u}^T M_u \mathbf{u} - (\mathbf{u}^r)^T M_u \mathbf{u}$$

Putting J_y and J_u together, we have

$$J = J_x + J_u = \frac{1}{2}\begin{bmatrix}\mathbf{u} \\ \mathbf{x}\end{bmatrix}^T \begin{bmatrix} M_u & 0 \\ 0 & M_c^T M_y M_c \end{bmatrix} \begin{bmatrix}\mathbf{u} \\ \mathbf{x}\end{bmatrix} - \begin{bmatrix}(\mathbf{u}^r)^T M_u & (\mathbf{y}^r)^T M_y M_c\end{bmatrix} \begin{bmatrix}\mathbf{u} \\ \mathbf{x}\end{bmatrix} \tag{5.22}$$

Note that when $\mathbf{y}^r$ and $\mathbf{u}^r$ are both 0, and C equals identity matrix, (5.22) would reduce to exactly (5.4).

The output constraints $y_{\min} \le y_k \le y_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix} y_{\min} \\ y_{\min} \\ \vdots \\ y_{\min} \end{bmatrix} \le \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix} \le \begin{bmatrix} y_{\max} \\ y_{\max} \\ \vdots \\ y_{\max} \end{bmatrix}$$

or equivalently

$$\mathbf{y}_{\min} \le M_C \mathbf{x} \le \mathbf{y}_{\max} \tag{5.23}$$

Putting everything together, we have the following QP formulation for (5.21):

$$\min_{\mathbf{z}} \quad \frac{1}{2}\mathbf{z}^T \begin{bmatrix} M_u & 0 \\ 0 & M_c^T M_y M_c \end{bmatrix} \mathbf{z} - \begin{bmatrix}(\mathbf{u}^r)^T M_u & (\mathbf{y}^r)^T M_y M_c\end{bmatrix} \mathbf{z} \tag{5.24a}$$

$$\text{subject to} \quad \begin{cases} \begin{bmatrix} -M_B & I - M_A \end{bmatrix} \mathbf{z} = r_0 \\ \begin{bmatrix} I & 0 \\ 0 & M_C \end{bmatrix} \mathbf{z} \le \begin{bmatrix} \mathbf{u}_{\max} \\ \mathbf{y}_{\max} \end{bmatrix} \\ -\begin{bmatrix} I & 0 \\ 0 & M_C \end{bmatrix} \mathbf{z} \le -\begin{bmatrix} \mathbf{u}_{\min} \\ \mathbf{y}_{\min} \end{bmatrix} \\ \begin{bmatrix} D & 0 \end{bmatrix} \mathbf{z} \le \boldsymbol{\Delta}_{\max} - r_u \\ -\begin{bmatrix} D & 0 \end{bmatrix} \mathbf{z} \le -\boldsymbol{\Delta}_{\min} + r_u \end{cases} \tag{5.24b}$$

Example 5.2.1 (MPC Steering Control with Output Tracking)**.** *Let's revisit the vehicle steering control problem discussed in Example 5.0.1, where the system dynamics are described by*

$$x_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0.2 & 0 \\ 0 & 0 \\ 0 & \frac{2}{3} \end{bmatrix} u_k = Ax_k + Bu_k, \qquad y_k = x_k$$

Now suppose the initial conditions are specified as $v_0 = 1$, $p_{y,0} = -2$, *and* $\phi_0 = -0.2$, *i.e.,* $x_0 = \begin{bmatrix} 1 & -2 & -0.2 \end{bmatrix}^T$. *To regulate the vehicle speed and heading angle back to their nominal value while maintaining a constant lateral position of* 1, *the following MPC problem is formulated.*

$$\min_{\boldsymbol{u}} \quad \frac{1}{2} \sum_{k=1}^{p} \|y_k - y^r\|_{Q_y}^2 + \frac{1}{2} \sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 \qquad \textit{subject to} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, & \textit{given } x_0 \\ y_{min} \le y_k \le y_{max} \\ u_{min} \le u_k \le u_{max} \end{cases}$$

When

$$y^r = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad Q_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad Q_u = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$$

and

$$y_{min} = \begin{bmatrix} 9 \\ -4 \\ -\infty \end{bmatrix}, \quad y_{max} = \begin{bmatrix} 11 \\ 2 \\ \infty \end{bmatrix}, \quad u_{min} = \begin{bmatrix} -0.2 \\ -2 \end{bmatrix}, \quad u_{max} = \begin{bmatrix} 0.2 \\ 1 \end{bmatrix},$$

the results are plotted in Fig. 5.2. As can be seen, both longitudinal position p_x *and yaw angle* ϕ *are effectively converging back to* 0, *while lateral position* p_y *converge to the reference of* $y^r = 1$.

5.2.2 Penalization of Input Rate

In practice, it is often desired to limit the actuator change rate, this can be done through constraint $\Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max}$, but it can also be achieved by adding an additional term in the cost function, as follows.

$$\min_{\mathbf{u}} \quad \frac{1}{2} \sum_{k=1}^{p} \|x_k\|_{Q_x}^2 + \frac{1}{2} \sum_{k=0}^{p-1} \|u_k\|_{Q_u}^2 + \frac{1}{2} \sum_{k=0}^{p-1} \|u_k - u_{k-1}\|_{Q_d}^2 \qquad \text{s.t.} \quad \begin{cases} x_{k+1} = Ax_k + Bu_k, & \text{given } x_0 \\ x_{\min} \le x_k \le x_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \end{cases} \tag{5.25}$$

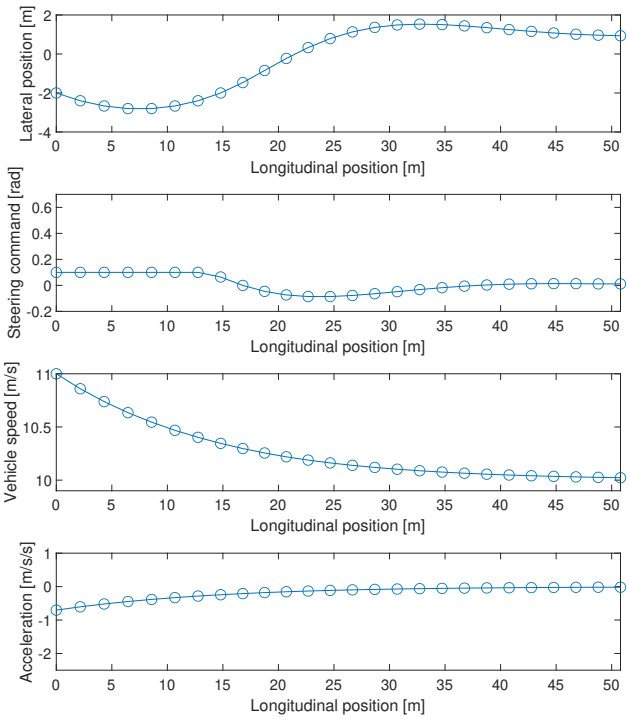

Figure 5.2: Control performance for Example 5.2.1.

The new term, denoted as $J_d = \frac{1}{2}\sum_{k=0}^{p-1}\|u_k - u_{k-1}\|_{Q_d}^2$, can be calculated as

$$J_d = \frac{1}{2}\sum_{k=0}^{p-1}\|u_k - u_{k-1}\|_{Q_d}^2 = \frac{1}{2}\begin{bmatrix} u_0 - u_{-1} \\ u_1 - u_0 \\ \vdots \\ u_{p-1} - u_{p-2} \end{bmatrix}^T \underbrace{\begin{bmatrix} Q_d & 0 & \cdots & 0 \\ 0 & Q_d & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q_d \end{bmatrix}}_{M_D} \begin{bmatrix} u_0 - u_{-1} \\ u_1 - u_0 \\ \vdots \\ u_{p-1} - u_{p-2} \end{bmatrix}$$

Since

$$\begin{bmatrix} u_0 - u_{-1} \\ u_1 - u_0 \\ \vdots \\ u_{p-1} - u_{p-2} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{bmatrix}}_{D} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{p-1} \end{bmatrix} + \underbrace{\begin{bmatrix} -u_{-1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{r_u},$$

we have

$$J_d = \frac{1}{2}(D\mathbf{u} + r_u)^T M_D (D\mathbf{u} + r_u) = \frac{1}{2}\mathbf{u}^T \underbrace{D^T M_D D}_{H_D}\mathbf{u} + \underbrace{r_u^T M_D D}_{f_d}\mathbf{u} + \underbrace{\frac{1}{2} r_u^T M_D r_u}_{Independent of \mathbf{u}}$$

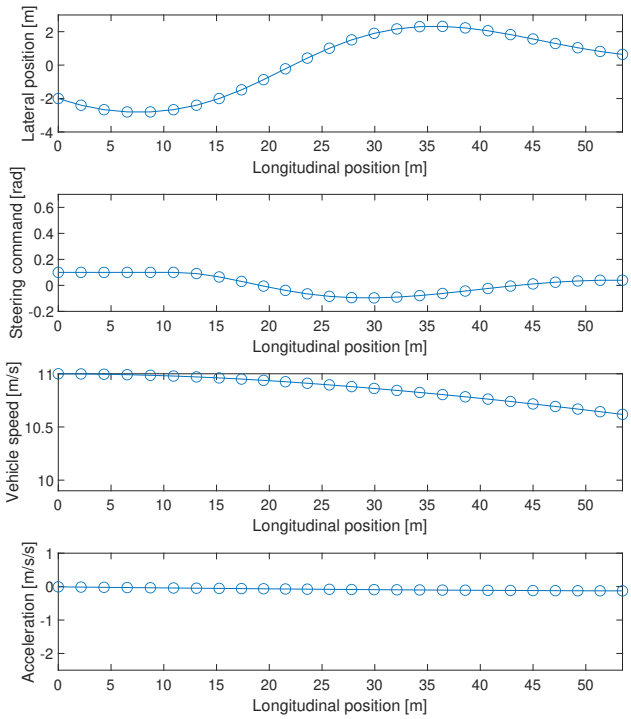

Figure 5.3: Control performance for Example 5.2.2.

Therefore, OCP (5.25) can be reformulated into the following (dense) QP

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T\left(H_x + M_u + H_D\right)\mathbf{u} + (f_x + f_d)\mathbf{u} \qquad \text{subject to} \quad \begin{cases} M_{AB}\mathbf{u} \le \mathbf{x}_{\max} - M_{Ak}x_0 \\ -M_{AB}\mathbf{u} \le -\mathbf{x}_{\min} + M_{Ak}x_0 \\ \mathbf{u} \le \mathbf{u}_{\max} \\ -\mathbf{u} \le -\mathbf{u}_{\min} \\ D\mathbf{u} \le \boldsymbol{\Delta}_{\max} - r_u \\ -D\mathbf{u} \le -\boldsymbol{\Delta}_{\min} + r_u \end{cases} \tag{5.26}$$

Example 5.2.2 (MPC Steering Control with Rate Weights)**.** *Let's revisit the vehicle steering control problem discussed in Example 5.2.1, but with rate weights. In other words, we set*

Weights.ManipulatedVariablesRate = [20 20];

The result is plotted in Fig. 5.3. Comparing to Fig. 5.2, it can be seen that with positive input rate weights, the controller becomes slower.

5.2.3 Move Blocking

Another practically useful variation of MPC is called "move blocking", which solves the following OCP

$$\min_{\mathbf{u}} \quad \frac{1}{2}\sum_{k=1}^{p}\|x_k\|_{Q_x}^2+\frac{1}{2}\sum_{k=0}^{m-1}\|u_k\|_{Q_u}^2 \qquad \text{s.t.} \quad \begin{cases} x_{k+1}=Ax_k+Bu_k, \quad \text{given } x_0 \\ x_{\min}\le x_k\le x_{\max} \\ u_{\min}\le u_k\le u_{\max}, k=0,\dots,m-1 \\ \Delta_{\min}\le u_k-u_{k-1}\le \Delta_{\max}, k=0,\dots,m-1 \\ u_k=u_{k-1}, k=m,\dots,p-1 \end{cases} \tag{5.27}$$

In other words, the control sequence can vary up to $m \le p$ time step into the horizon, and after which it is kept constant at u_{m-1} from time step m until $p-1$. Note that this blocking is only for the prediction horizon. Due to the receding horizon structure, the control action will be time varying all the time.

The parameter m is often called "control horizon", as it specifies the horizon during which control sequence can vary. OCP (5.27) can be equivalently expressed as

$$\min_{\mathbf{u}} \quad \frac{1}{2}\sum_{k=1}^{p}\|x_k\|_{Q_x}^2+\sum_{k=0}^{m-1}\frac{1}{2}\|u_k\|_{Q_u}^2 \qquad \text{s.t.} \quad \begin{cases} x_{k+1}=Ax_k+Bu_k, \quad \text{given } x_0, k\le m-1 \\ x_{k+1}=Ax_k+Bu_{m-1}, \quad \text{given } x_0, k\ge m \\ x_{\min}\le x_k\le x_{\max} \\ u_{\min}\le u_k\le u_{\max} \\ \Delta_{\min}\le u_k-u_{k-1}\le \Delta_{\max} \end{cases} \tag{5.28}$$

with an abuse of notation that $\mathbf{u}$ is defined as:

$$\mathbf{u}=\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{m-1} \end{bmatrix} \tag{5.29}$$

5.2.4 Linear Cost MPC

The MPC cost function can also be 1-norm or ∞-norm, or a combination of 1-norm, 2-norm, and ∞-norm. For the case of 1-norm, the OCP is given as

$$\min_{\mathbf{u}} \quad \sum_{k=1}^{p}\|Q_xx_k\|_1+\sum_{k=0}^{p-1}\|Q_uu_k\|_1 \qquad \text{subject to} \quad \begin{cases} x_{k+1}=Ax_k+Bu_k, \quad \text{given } x_0 \\ x_{\min}\le x_k\le x_{\max} \\ u_{\min}\le u_k\le u_{\max} \\ \Delta_{\min}\le u_k-u_{k-1}\le \Delta_{\max} \end{cases} \tag{5.30}$$

where the cost function can be equivalently represented as

$$\begin{aligned} J&=\sum_{k=1}^{p}\|Q_xx_k\|_1+\sum_{k=0}^{p-1}\|Q_uu_k\|_1 \\ &=\sum_{k=1}^{p}|Q_x^1x_k|+|Q_x^2x_k|+\cdots|Q_x^{n_x}x_k|+\sum_{k=0}^{p-1}|Q_u^1u_k|+|Q_u^2u_k|+\cdots|Q_u^{n_u}u_k| \end{aligned}$$

where the notation Q^i denotes the ith row of Q matrix. OCP (5.30) can be reformulated into a "linear programming" (LP) problem, by introducing slack variables ϵ^x_{ik} for $k = 1, \ldots, p$, $i = 1, \ldots, n_x$, and ϵ^u_{ik} for $k = 0, \ldots, p-1$, $i = 1, \ldots, n_u$. The unconstrained minimization of J is equivalent to

$$\begin{aligned} \min_{\mathbf{u},\epsilon} \quad & \sum_{k=1}^{p} \epsilon^x_{1k} + \epsilon^x_{2k} + \cdots \epsilon^x_{n_x k} + \sum_{k=0}^{p-1} \epsilon^u_{1k} + \epsilon^u_{2k} + \cdots \epsilon^u_{n_u k} \\ \text{subject to} \quad & \epsilon^x_{1k} \geq |Q^1_x x_k|, \ldots, \epsilon^x_{n_x k} \geq |Q^{n_x}_x x_k|, k = 1, \ldots, p \\ & \epsilon^u_{1k} \geq |Q^1_u u_k|, \ldots, \epsilon^u_{n_u k} \geq |Q^{n_u}_u u_k|, k = 0, \ldots, p-1 \end{aligned}$$

or equivalently

$$\begin{aligned} \min_{\mathbf{u},\epsilon} \quad & \sum_{k=1}^{p} \sum_{i=1}^{n_x} \epsilon^x_{ik} + + \sum_{k=0}^{p-1} \sum_{i=1}^{n_u} \epsilon^u_{ik} \\ \text{subject to} \quad & \epsilon^x_{ik} \geq |Q^i_x x_k|, k = 1, \ldots, p, i = 1, \ldots, n_x \\ & \epsilon^u_{ik} \geq |Q^i_u u_k|, k = 0, \ldots, p-1, i = 1, \ldots, n_u \end{aligned}$$

The above formulation is not a standard LP problem due to the constraints $\epsilon^x_{ik} \geq |Q^i_x x_k|$. However, due to the following equivalence

$$\begin{aligned} \epsilon^x_{ik} \geq |Q^i_x x_k| &\Longleftrightarrow \epsilon^x_{ik} \geq Q^i_x x_k \text{ and } \epsilon^x_{ik} \geq -Q^i_x x_k \\ \epsilon^u_{ik} \geq |Q^i_u x_k| &\Longleftrightarrow \epsilon^u_{ik} \geq Q^i_u u_k \text{ and } \epsilon^u_{ik} \geq -Q^i_u x_k \end{aligned}$$

Then the unconstrained minimization of J is equivalent to

$$\begin{aligned} \min_{\mathbf{u},\epsilon} \quad & \sum_{k=1}^{p} \sum_{i=1}^{n_x} \epsilon^x_{ik} + + \sum_{k=0}^{p-1} \sum_{i=1}^{n_u} \epsilon^u_{ik} \\ \text{subject to} \quad & \epsilon^x_{ik} \geq Q^i_x x_k, k = 1, \ldots, p, i = 1, \ldots, n_x \\ & \epsilon^x_{ik} \geq -Q^i_x x_k, k = 1, \ldots, p, i = 1, \ldots, n_x \\ & \epsilon^u_{ik} \geq Q^i_u u_k, k = 0, \ldots, p-1, i = 1, \ldots, n_u \\ & \epsilon^u_{ik} \geq -Q^i_u u_k, k = 0, \ldots, p-1, i = 1, \ldots, n_u \end{aligned}$$

Putting everything together, the OCP (5.30) is equivalent to the following LP problem

$$\min_{\mathbf{u},\epsilon} \quad \sum_{k=1}^{p} \sum_{i=1}^{n_x} \epsilon^x_{ik} + + \sum_{k=0}^{p-1} \sum_{i=1}^{n_u} \epsilon^u_{ik} \quad \text{subject to} \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \text{given } x_0 \\ x_{\min} \leq x_k \leq x_{\max} \\ u_{\min} \leq u_k \leq u_{\max} \\ \Delta_{\min} \leq u_k - u_{k-1} \leq \Delta_{\max} \\ \epsilon^x_{ik} \geq Q^i_x x_k \\ \epsilon^x_{ik} \geq -Q^i_x x_k \\ \epsilon^u_{ik} \geq Q^i_u u_k \\ \epsilon^u_{ik} \geq -Q^i_u u_k \end{cases} \tag{5.31}$$

For the case of ∞-norm, the OCP is given as

$$\min_{\mathbf{u}} \quad \sum_{k=1}^{p} \|Q_x x_k\|_\infty + \sum_{k=0}^{p-1} \|Q_u u_k\|_\infty \qquad \text{subject to } \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \text{given } x_0 \\ x_{\min} \le x_k \le x_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \end{cases} \tag{5.32}$$

where the cost function can be equivalently represented a

$$J = \sum_{k=1}^{p} \|Q_x x_k\|_\infty + \sum_{k=0}^{p-1} \|Q_u u_k\|_\infty = \sum_{k=1}^{p} \max_i |Q_x^i x_k| + \sum_{k=0}^{p-1} \max_i |Q_u^i u_k|$$

Similarly we introduce slack variables ϵ_k^x for $k = 1, \ldots, p$, and ϵ_k^u for $k = 0, \ldots, p-1$. The unconstrained minimization of J is equivalent to

$$\min_{\mathbf{u},\epsilon} \quad \sum_{k=1}^{p} \epsilon_k^x + \sum_{k=0}^{p-1} \epsilon_k^u \qquad \text{subject to } \begin{cases} \epsilon_k^x \ge \max_i |Q_x^i x_k|, k = 1, \ldots, p \\ \epsilon_k^u \ge \max_i |Q_u^i u_k|, k = 0, \ldots, p-1 \end{cases}$$

or equivalently

$$\min_{\mathbf{u},\epsilon} \quad \sum_{k=1}^{p} \epsilon_k^x + \sum_{k=0}^{p-1} \epsilon_k^u \qquad \text{subject to } \begin{cases} \epsilon_k^x \ge |Q_x^i x_k|, k = 1, \ldots, p, i = 1, \ldots, n_x \\ \epsilon_k^u \ge |Q_u^i u_k|, k = 0, \ldots, p-1, i = 1, \ldots, n_u \end{cases}$$

The above formulation is not a standard LP problem due to the constraints $\epsilon_k^x \ge |Q_x^i x_k|$. However, due to the following equivalence

$$\epsilon_k^x \ge |Q_x^i x_k| \Longleftrightarrow \epsilon_k^x \ge Q_x^i x_k \text{ and } \epsilon_k^x \ge -Q_x^i x_k$$
$$\epsilon_k^u \ge |Q_u^i x_k| \Longleftrightarrow \epsilon_k^u \ge Q_u^i u_k \text{ and } \epsilon_k^u \ge -Q_u^i x_k$$

Then the unconstrained minimization of J is equivalent to

$$\min_{\mathbf{u},\epsilon} \quad \sum_{k=1}^{p} \epsilon_k^x + \sum_{k=0}^{p-1} \epsilon_k^u \qquad \text{subject to } \begin{cases} \epsilon_k^x \ge Q_x^i x_k, k = 1, \ldots, p, i = 1, \ldots, n_x \\ \epsilon_k^x \ge -Q_x^i x_k, k = 1, \ldots, p, i = 1, \ldots, n_x \\ \epsilon_k^u \ge Q_u^i u_k, k = 0, \ldots, p-1, i = 1, \ldots, n_u \\ \epsilon_k^u \ge -Q_u^i u_k, k = 0, \ldots, p-1, i = 1, \ldots, n_u \end{cases}$$

Putting everything together, the OCP (5.32) is equivalent to the following LP problem

$$\min_{\mathbf{u},\epsilon} \quad \sum_{k=1}^{p} \epsilon_k^x + + \sum_{k=0}^{p-1} \epsilon_k^u \quad \text{subject to } \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \text{given } x_0 \\ x_{\min} \le x_k \le x_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \\ \epsilon_k^x \ge Q_x^i x_k \\ \epsilon_k^x \ge -Q_x^i x_k \\ \epsilon_k^u \ge Q_u^i u_k \\ \epsilon_k^u \ge -Q_u^i u_k \end{cases} \tag{5.33}$$

Example 5.2.3 (Linear Cost MPC)**.** *Let's revisit Example 5.1.1, where the linear system is given as*

$$x_{k+1} = \begin{bmatrix} 0.7 & 0.1 \\ 0 & 0.1 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k$$

Consider the 1-norm cost MPC formulation of (5.30) with $p = 2$, $Q_x = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$, $Q_u = 3$. *Let's only consider input constraint with* $u_{min} = -2$ *and* $u_{max} = 3$. *At a given time step, assume* $x_0 = \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$. *Find its equivalent sparse LP formulation.*

The 1-norm cost MPC in this case is written as

$$\min_{\boldsymbol{u}} \quad \sum_{k=1}^{p} \|Q_x x_k\|_1 + \sum_{k=0}^{p-1} \|Q_u u_k\|_1 \qquad \textit{subject to} \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \textit{given } x_0 \\ u_{min} \le u_k \le u_{max} \end{cases}$$

where the cost function is equivalent to

$$J = 2|x_{11}| + |x_{21}| + 2|x_{12}| + |x_{22}| + 3|u_{00}| + 3|u_{01}|$$

Introduce slack variables ϵ_{11}^x, ϵ_{21}^x, ϵ_{12}^x, ϵ_{22}^x, ϵ_{00}^u *and* ϵ_{01}^u. *Then the OCP is equivalently expressed as*

$$\min_{\boldsymbol{u},\epsilon} \quad \epsilon_{11}^x + \epsilon_{21}^x + \epsilon_{12}^x + \epsilon_{22}^x + \epsilon_{00}^u + \epsilon_{01}^u \qquad \textit{subject to} \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \textit{given } x_0 \\ u_{min} \le u_k \le u_{max} \\ \epsilon_{11}^x \ge 2x_{11}, \epsilon_{11}^x \ge -2x_{11} \\ \epsilon_{21}^x \ge x_{21}, \epsilon_{21}^x \ge -x_{21} \\ \epsilon_{12}^x \ge 2x_{12}, \epsilon_{12}^x \ge -2x_{12} \\ \epsilon_{22}^x \ge x_{22}, \epsilon_{22}^x \ge -x_{22} \\ \epsilon_{00}^u \ge 3u_{00}, \epsilon_{00}^u \ge -3u_{00} \\ \epsilon_{01}^u \ge 3u_{01}, \epsilon_{01}^u \ge -3u_{01} \end{cases}$$

Consider the ∞*-norm cost MPC formulation of (5.32) with* $p = 2$, $Q_x = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$, $Q_u = 3$. *Let's only consider input constraint with* $u_{min} = -2$ *and* $u_{max} = 3$. *At a given time step, assume* $x_0 = \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$. *Find its equivalent sparse LP formulation.*

The ∞*-norm cost MPC in this case is written as*

$$\min_{\boldsymbol{u}} \quad \sum_{k=1}^{p} \|Q_x x_k\|_\infty + \sum_{k=0}^{p-1} \|Q_u u_k\|_\infty \qquad \textit{subject to} \begin{cases} x_{k+1} = Ax_k + Bu_k, \quad \textit{given } x_0 \\ u_{min} \le u_k \le u_{max} \end{cases}$$

where the cost function is equivalent to

$$J = \max(2|x_{11}|, |x_{21}|) + \max(2|x_{12}|, |x_{22}|) + 3|u_{00}| + 3|u_{01}|$$

Introduce slack variables ϵ_1^x, ϵ_2^x, ϵ_0^u and ϵ_1^u. Then the OCP is equivalently expressed as

$$\min_{\boldsymbol{u},\boldsymbol{\epsilon}} \quad \epsilon_1^x+\epsilon_2^x+\epsilon_0^u+\epsilon_1^u \qquad \textit{subject to} \begin{cases} x_{k+1}=Ax_k+Bu_k, \quad \textit{given } x_0 \\ u_{min}\le u_k\le u_{max} \\ \epsilon_1^x\ge 2x_{11}, \epsilon_1^x\ge -2x_{11} \\ \epsilon_1^x\ge x_{21}, \epsilon_1^x\ge -x_{21} \\ \epsilon_2^x\ge 2x_{12}, \epsilon_2^x\ge -2x_{12} \\ \epsilon_2^x\ge x_{22}, \epsilon_2^x\ge -x_{22} \\ \epsilon_0^u\ge 3u_{00}, \epsilon_0^u\ge -3u_{00} \\ \epsilon_1^u\ge 3u_{01}, \epsilon_1^u\ge -3u_{01} \end{cases}$$

5.2.5 Robust MPC and Stochastic MPC

So far, we have considered the system dynamic to be deterministic. Many practical systems, however, are stochastic, and can be modeled as, for example,

$$x_{k+1}=Ax_k+Bu_k+E\omega_k \tag{5.34}$$

where ω_k is a random variable with known distribution and satisfies $\omega_{\min}\le\omega_k\le\omega_{\max}$. Note that ω_k is often called "unknown disturbance" in literature. In this case the OCP (5.2) may not be robust since it does not take into consideration the random variable ω_k. One popular approach to deal with ω_k is to formulate an OCP as follows.

$$\min_{\mathbf{u}} \quad \max_{\omega} \quad \frac{1}{2}\sum_{k=1}^{p}\|x_k\|_{Q_x}^2+\frac{1}{2}\sum_{k=0}^{p-1}\|u_k\|_{Q_u}^2 \qquad \text{s.t.} \quad \begin{cases} x_{k+1}=Ax_k+Bu_k+E\omega_k, \quad \text{given } x_0 \\ x_{\min}\le x_k\le x_{\max} \\ u_{\min}\le u_k\le u_{\max} \\ \Delta_{\min}\le u_k-u_{k-1}\le\Delta_{\max} \end{cases} \tag{5.35}$$

In other words, the goal here is to find optimal control sequence $\mathbf{u}^*$ such that the worst-case loss function is minimized, and is often termed as min-max MPC. The OCP (5.35) can be reformulated into a QP problem. See [7] for details.

However, the min-max MPC discussed above can be over-conservative, since it aims to minimize the worst-case loss function. In practice, it can be desirable to minimize the "average" (instead of the worst-case) loss function. To implement this, one can randomly sample N instances of the random variables ω_k, denoted as $\bar{\omega}_k^1,\bar{\omega}_k^2,\ldots,\bar{\omega}_k^N$, and solve the following MPC [8].

$$\min_{\mathbf{u}} \quad \sum_{i=1}^{N}\left\{\frac{1}{2}\sum_{k=1}^{p}\|x_k^n\|_{Q_x}^2+\frac{1}{2}\sum_{k=0}^{p-1}\|u_k^n\|_{Q_u}^2\right\} \qquad \text{s.t.} \quad \begin{cases} x_{k+1}^n=Ax_k^n+Bu_k^n+E\bar{\omega}_k^n, \quad \text{given } x_0 \\ x_{\min}\le x_k^n\le x_{\max} \\ u_{\min}\le u_k^n\le u_{\max} \\ \Delta_{\min}\le u_k^n-u_{k-1}^n\le\Delta_{\max} \\ u_0^1=u_0^2=u_0^3=\cdots=u_0^N \end{cases} \tag{5.36}$$

This approach is called "stochastic MPC" in literature. In other words, stochastic MPC samples N trajectory for the unknown disturbance, and for each trajectory i, an optimal control sequence $(\mathbf{u}^*)^i$ is found so that the cost function as of (5.2) is minimized if the disturbance is exactly $\bar{\omega}_k$. Furthermore, the constraint $u_0^1=u_0^2=u_0^3=\cdots=u_0^N$ is used to ensure that the optimal control

sequence for different trajectory start at the same point, which will be implemented by the actuator. It is trivial to see that when N is large, the solution of (5.36) will minimize the "expected loss" function, which computation burden increases as N increases.

5.3 Explicit MPC

The OCP (5.2) requires solving a QP problem in real-time, which can be time consuming. Fortunately, this can be addressed using explicit MPC approach [9]. Recall that (5.2) can be cast into (5.19), as follows.

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T\left(H_x+M_u\right)\mathbf{u}+f_x\mathbf{u} \qquad \text{subject to} \quad \begin{cases} M_{AB}\mathbf{u}\le \mathbf{x}_{\max}-M_{Ak}x_0 \\ -M_{AB}\mathbf{u}\le -\mathbf{x}_{\min}+M_{Ak}x_0 \\ \mathbf{u}\le \mathbf{u}_{\max} \\ -\mathbf{u}\le -\mathbf{u}_{\min} \\ D\mathbf{u}\le \boldsymbol{\Delta}_{\max}-r_u \\ -D\mathbf{u}\le -\boldsymbol{\Delta}_{\min}+r_u \end{cases}$$

Note that here the vectors that needs real-time update are marked in red, while the remaining matrices/vectors are constant in real-time and can be calculated offline. Also recall that

$$f_x = x_0^T \underbrace{M_{Ak}^T M_x M_{AB}}_{F} = x_0^T F$$

For the simplicity of notation, let's consider a special case without rate constraint $\Delta_{\min}\le u_k-u_{k-1}\le\Delta_{\max}$[1]. In this case, the MPC problem can be cast into the following QP.

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T\left(H_x+M_u\right)\mathbf{u}+x_0^T F\mathbf{u} \qquad \text{subject to} \quad \begin{cases} M_{AB}\mathbf{u}\le \mathbf{x}_{\max}-M_{Ak}x_0 \\ -M_{AB}\mathbf{u}\le -\mathbf{x}_{\min}+M_{Ak}x_0 \\ \mathbf{u}\le \mathbf{u}_{\max} \\ -\mathbf{u}\le -\mathbf{u}_{\min} \end{cases} \tag{5.37}$$

Further, denote

$$\begin{aligned} H &= H_x+M_u \\ G &= \begin{bmatrix} M_{AB} \\ -M_{AB} \\ I \\ -I \end{bmatrix} \\ W &= \begin{bmatrix} \mathbf{x}_{\max} \\ -\mathbf{x}_{\min} \\ \mathbf{u}_{\max} \\ -\mathbf{u}_{\min} \end{bmatrix} \\ S &= \begin{bmatrix} -M_{Ak} \\ M_{Ak} \\ 0 \\ 0 \end{bmatrix} \end{aligned}$$

[1]The case with rate constraints, output reference and/or input reference can be treated similarly.

Then (5.37) can be compactly represented as

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T H\mathbf{u} + x_0^T F\mathbf{u} \qquad \text{subject to} \quad G\mathbf{u} \le W + Sx_0 \tag{5.38}$$

Note that (5.38) is also called "multi-parametric" QP, or mpQP, as it is parameterized by x_0.

The primal and dual solution of (5.38) should satisfy the KKT condition. Therefore,

$$H\mathbf{u}^* + F^T x_0 + G^T \lambda^* = 0 \tag{5.39a}$$
$$\lambda_i^*(G^i\mathbf{u} - W^i - S^i x_0) = 0, \qquad i \in \mathcal{A} \tag{5.39b}$$
$$\lambda^* \ge 0 \tag{5.39c}$$
$$G\mathbf{u} \le W + Sx_0 \tag{5.39d}$$

where $\mathcal{A}$ is the set of active constraints, i.e., constraints that the equality holds, corresponding to the optimal solution $\mathbf{u}^*$. Then the KKT condition above gives rise to the following linear system

$$\begin{bmatrix} H & \left(G^{\mathcal{A}}\right)^T \\ G^{\mathcal{A}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -F^T x_0 \\ W^{\mathcal{A}} + S^{\mathcal{A}} x_0 \end{bmatrix} \tag{5.40}$$

Since H is positive definitive, we have

$$\mathbf{u}^* = -H^{-1}\left(\left(G^{\mathcal{A}}\right)^T \lambda^* + F^T x_0\right). \tag{5.41}$$

Assuming that LICQ holds, we have

$$\begin{aligned} G^{\mathcal{A}}\mathbf{u}^* &= W^{\mathcal{A}} + S^{\mathcal{A}} x_0 \\ -G^{\mathcal{A}} H^{-1}\left(\left(G^{\mathcal{A}}\right)^T \lambda^* + F^T x_0\right) &= W^{\mathcal{A}} + S^{\mathcal{A}} x_0 \\ -G^{\mathcal{A}} H^{-1}\left(G^{\mathcal{A}}\right)^T \lambda^* - G^{\mathcal{A}} H^{-1} F^T x_0 &= W^{\mathcal{A}} + S^{\mathcal{A}} x_0 \\ -G^{\mathcal{A}} H^{-1}\left(G^{\mathcal{A}}\right)^T \lambda^* &= W^{\mathcal{A}} + S^{\mathcal{A}} x_0 + G^{\mathcal{A}} H^{-1} F^T x_0 \\ \lambda^* &= -\left(G^{\mathcal{A}} H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(W^{\mathcal{A}} + S^{\mathcal{A}} x_0 + G^{\mathcal{A}} H^{-1} F^T x_0\right) \end{aligned}$$

Plug in back to (5.41), we have

$$\begin{aligned} \mathbf{u}^* =& -H^{-1}\left(-\left(G^{\mathcal{A}}\right)^T\left(G^{\mathcal{A}} H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(W^{\mathcal{A}} + S^{\mathcal{A}} x_0 + G^{\mathcal{A}} H^{-1} F^T x_0\right) + F^T x_0\right) \\ =& \underbrace{H^{-1}\left[\left(G^{\mathcal{A}}\right)^T\left(G^{\mathcal{A}} H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(S^{\mathcal{A}} + G^{\mathcal{A}} H^{-1} F^T\right) - F^T\right]}_{K(\mathcal{A})} x_0 \\ &+ \underbrace{H^{-1}\left(G^{\mathcal{A}}\right)^T\left(G^{\mathcal{A}} H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(W^{\mathcal{A}}\right)}_{g(\mathcal{A})} \end{aligned}$$

In other words, we have

$$\mathbf{u}^* = K(\mathcal{A})x_0 + g(\mathcal{A}) \tag{5.42}$$

Recall that the optimal control to be implemented is given by

$$u^* = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix}\mathbf{u}^* = \underbrace{\begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} K(\mathcal{A})}_{K_{\mathcal{A}}} x_0 + \underbrace{\begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} g(\mathcal{A})}_{g_{\mathcal{A}}} \tag{5.43}$$

where $K_{\mathcal{A}}$ is given by the first n_u row of $K(\mathcal{A})$ and $g_{\mathcal{A}}$ is given by the first n_u rows of $g(\mathcal{A})$. Then we have the following affine feedback control law

$$u^* = K_{\mathcal{A}} x_0 + g_{\mathcal{A}}, \tag{5.44}$$

which is optimal as long as $\mathcal{A}$ remains active. In other words, the KKT condition continues to hold. According to (5.39d) and (5.39c), we have

$$GH^{-1}\left(\left(G^{\mathcal{A}}\right)^T\left(G^{\mathcal{A}}H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(W^{\mathcal{A}}+S^{\mathcal{A}}x_0+G^{\mathcal{A}}H^{-1}F^Tx_0\right)-F^Tx_0\right)\leq W+Sx_0 \tag{5.45a}$$

$$\lambda^* = -\left(G^{\mathcal{A}}H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(W^{\mathcal{A}}+S^{\mathcal{A}}x_0+G^{\mathcal{A}}H^{-1}F^Tx_0\right)\geq 0 \tag{5.45b}$$

or equivalently

$$\left[GH^{-1}\left(\left(G^{\mathcal{A}}\right)^T\left(G^{\mathcal{A}}H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(S^{\mathcal{A}}+G^{\mathcal{A}}H^{-1}F^T\right)-F^T\right)-S\right]x_0 \leq W-GH^{-1}\left(\left(G^{\mathcal{A}}\right)^T\left(G^{\mathcal{A}}H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}W^{\mathcal{A}}\right) \tag{5.46a}$$

$$\left(G^{\mathcal{A}}H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}\left(S^{\mathcal{A}}+G^{\mathcal{A}}H^{-1}F^T\right)x_0 \leq -\left(G^{\mathcal{A}}H^{-1}\left(G^{\mathcal{A}}\right)^T\right)^{-1}W^{\mathcal{A}} \tag{5.46b}$$

Therefore, (5.46) defines the polyhedron that in which $\mathcal{A}$ is active and feedback law (5.44) can be used.

Now the remaining goal is to find, for each possible active set $\mathcal{A}$, the subspace, or critical region $\mathrm{CR}(\mathcal{A})$, such that for all $x_0 \in \mathrm{CR}(\mathcal{A})$, the inequalities (5.46) hold. In other words, the offline portion of explicit MPC divides the space of $\mathbb{R}^{n_x}$ into multiple CRs so that in each CR, the active set remains the same. Furthermore, for each CR, the feedback gain $K_{\mathcal{A}}$ and $g_{\mathcal{A}}$ are calculated and stored in memory. Then the online portion of explicit MPC simply locates the CR based on current state feedback x_0, and uses (5.44) to compute the optimal control action, without numerically solving any optimization problem in real-time.

Example 5.3.1 (Visualize Explicit MPC Control Law)**.** *Consider the MPC problem discussed in Example 5.1.1. Fig. 5.4 plots the CRs for explicit MPC when $u_{-1} = 2$. As can be seen from Matlab Code Listing 5.6, there are a total number of 36 regions, and several of them are not visuable in Fig. 5.4 due to the small size. Therefore, in practice, one can combine regions to reduce the storage requirements.*

Furthermore, as shown in Matlab Code Listing 5.6, the explicit MPC and conventional MPC yield the same solution for real time.

Example 5.3.2 (Explicit MPC Approach for Steering Control)**.** *Consider the MPC problem discussed in Example 5.2.1. Fig. 5.5 plots closed-loop simulation results, which are identical to the conventional MPC approach plotted in Fig. 5.2.*

Explicit MPC approach can significantly reduce online computation, with an expense of using extra memory to store $K_{\mathcal{A}}$ and $g_{\mathcal{A}}$. However, the number of potential active set is exponential to the number of constraints. In the case of (5.37), this equals to $2^{2(n_x+n_u)p}$. This poses three challenges. First, for offline calculation, it may be impossible to find all CRs. Second, the required storage can be too high for practical use. Third, for a given x_0 the computation required to locate the corresponding CR may be too long to be beneficial. However, for small MPC problem, explicit MPC can still outperform the conventional approach.

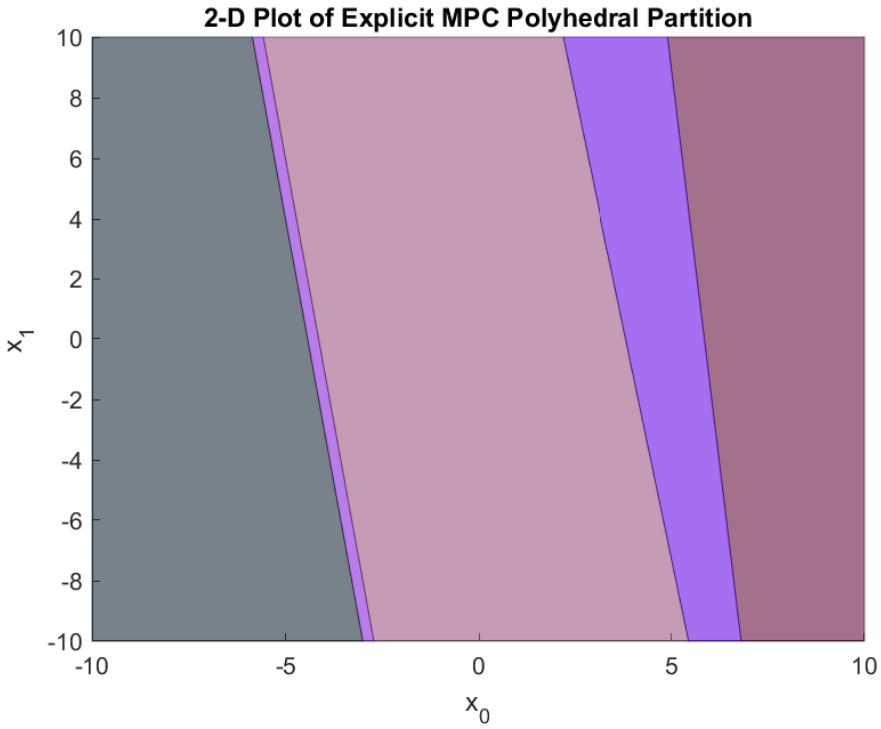

Figure 5.4: Feedback control law for Example 5.3.1.

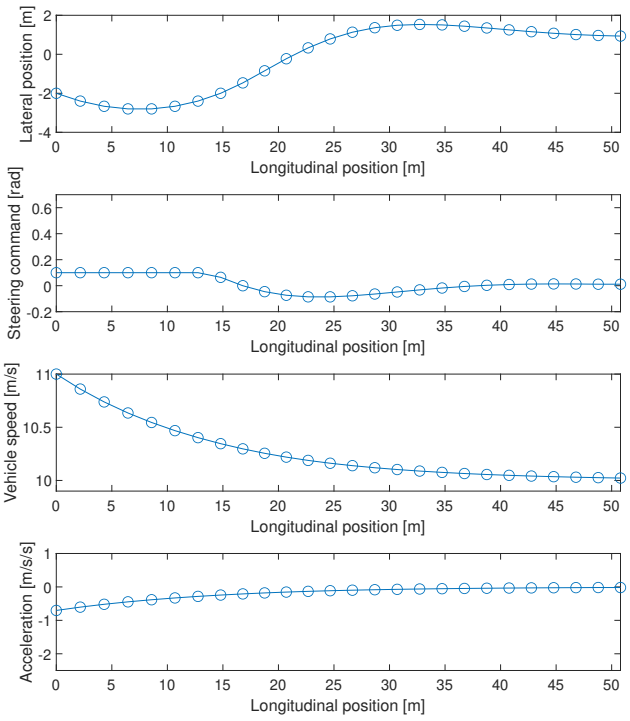

Figure 5.5: Explicit MPC control performance for Example 5.3.2.

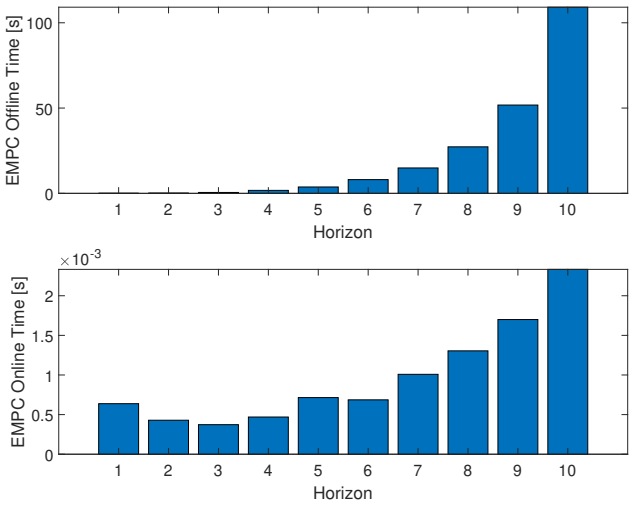

Figure 5.6: Feedback control law for Example 5.3.3.

Example 5.3.3 (Explicit MPC Computation Time)**.** *Consider the MPC problem discussed in Example 5.1.1. Fig. 5.6 the offline computation time required to build explicit MPC and the online computation time required to located CR and output optimal control, both as function of prediction horizon. As can be seen, the computation time for explicit MPC increases exponentially with respect to prediction horizon p, making it only practical for smaller control problem.*

5.4 Matlab Codes

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; x0 = [1;-2;-0.2];
Ts = 0.2; t = 0:Ts:5; N = numel(t);
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; B = [Ts 0;0 0;0 xbar(1)/l*Ts];
C = eye(3);

plant = ss(A,B,C,[],Ts); p = 5; m = p;

Weights.ManipulatedVariables = [1 10];
Weights.ManipulatedVariablesRate = [0 0];
Weights.OutputVariables = [1 1 1];
mpcobj = mpc(plant,[],p,m,Weights);
setEstimator(mpcobj,'custom');
setoutdist(mpcobj,'model',tf(zeros(3,1)));
mpcobj.MV(2).Min = -0.1; mpcobj.MV(2).Max = 0.1;

xc = mpcstate(mpcobj); xc.Plant=x0;

uopt = zeros(2,N); x = zeros(3,N); x(:,1) = x0;
for n = 2:N
    xc.Plant=x(:,n-1);
    uopt(:,n-1) = mpcmove(mpcobj,xc,[],[0;0;0]);
    x(:,n) = A*x(:,n-1)+B*uopt(:,n-1);
end
uopt(:,n) = uopt(:,n-1); x = x + repmat(xbar,1,N);

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos;

px = cumsum(x(1,:).*cos(x(3,:))*Ts); px=[0 px(1:end-1)]; py = x(2,:);

subplot(4,1,1), plot(px,py,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-4,2])

subplot(4,1,2), plot(px,uopt(2,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.2,0.7])

subplot(4,1,3), plot(px,x(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Vehicle speed [m/s]'),
axis tight, ylim([9.9,11])

subplot(4,1,4), plot(px,uopt(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Acceleration [m/s/s]'),
axis tight, ylim([-2.5,1])
```

Listing 5.1: MATLAB Commands for Example 5.0.1.

```
close all; clear variables; clc

A = [0.7, 0.1; 0 0.1]; B = [1;0]; C = eye(2); Ts = 1;
plant = ss(A,B,C,[],Ts);

p = 2; m = p;  Qu = 3; Qx = [2 1];

xmax = 5;% try 2 instead of 5;

Weights.ManipulatedVariables = Qu;
Weights.ManipulatedVariablesRate = 0;
Weights.OutputVariables = Qx;
mpcobj = mpc(plant,[],p,m,Weights);
setEstimator(mpcobj,'custom');
setoutdist(mpcobj,'model',tf(zeros(2,1)));
mpcobj.MV.Min = -2; mpcobj.MV.Max = 3;
mpcobj.MV.RateMin = -0.1; mpcobj.MV.RateMax = 0.1;
mpcobj.OV(1).Min = -1; mpcobj.OV(1).Max = xmax; % try 2 instead of 5
mpcobj.OV(2).Min = -1; mpcobj.OV(2).Max = xmax; % try 2 instead of 5

xc = mpcstate(mpcobj); xc.Plant=[0.2;-0.1]; xc.LastMove = 2;

[u,opt] = mpcmove(mpcobj,xc,[],[0;0]);
Uopt=opt.Uopt(1:2);
Xopt=opt.Xopt(2:3,:)';

z1 = [Uopt(:);Xopt(:)];

H = diag([3,3,2,1,2,1]);
f=zeros(6,1);

Aeq = [-1 0 1 0 0 0; 0 0 0 1 0 0; 0 -1 -0.7 -0.1 1 0;0 0 0 -0.1 0 1];
beq = [0.13; -0.01; 0; 0];
Ac=[eye(6);-eye(6);1 0 0 0 0 0;-1 1 0 0 0 0;-1 0 0 0 0 0;1 -1 0 0 0 0];
bc=[3;3;xmax;xmax;xmax;xmax; 2; 2; 1; 1; 1; 1; 2.1; 0.1; -1.9; 0.1];
z2 = quadprog(H,f,Ac,bc,Aeq,beq);

>> norm(z1-z2)

ans =

   2.4924e-15
```

Listing 5.2: MATLAB Commands for Example 5.1.1.

```
close all; clear variables; clc

A = [0.7, 0.1; 0 0.1]; B = [1;0]; C = eye(2); Ts = 1;
plant = ss(A,B,C,[],Ts);

p = 2; m = p;  Qu = 3; Qx = [2 1];

xmax = 5;% try 2 instead of 5;

Weights.ManipulatedVariables = Qu;
Weights.ManipulatedVariablesRate = 0;
Weights.OutputVariables = Qx;
mpcobj = mpc(plant,[],p,m,Weights);
setEstimator(mpcobj,'custom');
setoutdist(mpcobj,'model',tf(zeros(2,1)));
mpcobj.MV.Min = -2; mpcobj.MV.Max = 3;
mpcobj.MV.RateMin = -0.1; mpcobj.MV.RateMax = 0.1;
mpcobj.OV(1).Min = -1; mpcobj.OV(1).Max = xmax; % try 2 instead of 5
mpcobj.OV(2).Min = -1; mpcobj.OV(2).Max = xmax; % try 2 instead of 5

xc = mpcstate(mpcobj); xc.Plant=[0.2;-0.1]; xc.LastMove = 2;

[u,opt] = mpcmove(mpcobj,xc,[],[0;0]);
Uopt=opt.Uopt(1:2);
Xopt=opt.Xopt(2:3,:)';

z1 = [Uopt(:);Xopt(:)];

H = [5.98 1.4; 1.4 5];
f = [0.386 0.18];

Ac = [1 0;0.7 1; -1 0;-0.7 -1; 1 0;0 1;-1 0;0 -1;1 0;-1 1;-1 0;1 -1];
bc = [xmax-0.13; xmax-0.09; 1.13;1.09;3; 3; 2; 2; 2.1; 0.1; -1.9; 0.1];
z2 = quadprog(H,f,Ac,bc);

x0 = [0.2;-0.1]; x1 = A*x0+B*z2(1); x2 = A*x1+B*z2(2);
z2 = [z2;x1;x2];

>> norm(z1-z2)

ans =

   8.3081e-16
```

Listing 5.3: MATLAB Commands for Example 5.1.2.

```
>> [u,opt] = mpcmove(mpcobj,xc,[],[0;0]);
>> opt

opt =

  struct with fields:

          Uopt: [3×1 double]
          Yopt: [3×2 double]
          Xopt: [3×2 double]
          Topt: [3×1 double]
         Slack: 1.0500
    Iterations: 1
        QPCode: 'feasible'
          Cost: 3.3086e+05

>> u

u =

    1.8000

>> opt.Xopt

ans =

    0.2000   -0.1000
    1.9300   -0.0100
    3.0500   -0.0010
```

Listing 5.4: MATLAB Commands for Example 5.1.3.

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; x0 = [1;-2;-0.2];
Ts = 0.2; t = 0:Ts:5; N = numel(t);
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1]; B = [Ts 0;0 0;0 xbar(1)/l*Ts];
C = eye(3);

plant = ss(A,B,C,[],Ts); p = 5; m = p;

Weights.ManipulatedVariables = [1 10];
Weights.ManipulatedVariablesRate = [0 0];
Weights.OutputVariables = [1 1 1];
mpcobj = mpc(plant,[],p,m,Weights);
setEstimator(mpcobj,'custom');
setoutdist(mpcobj,'model',tf(zeros(3,1)));
mpcobj.MV(2).Min = -0.1; mpcobj.MV(2).Max = 0.1;

xc = mpcstate(mpcobj); xc.Plant=x0;

uopt = zeros(2,N); x = zeros(3,N); x(:,1) = x0;
for n = 2:N
    xc.Plant=x(:,n-1);
    uopt(:,n-1) = mpcmove(mpcobj,xc,[],[0;1;0]);
    x(:,n) = A*x(:,n-1)+B*uopt(:,n-1);
end
uopt(:,n) = uopt(:,n-1); x = x + repmat(xbar,1,N);

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos;

px = cumsum(x(1,:).*cos(x(3,:))*Ts); px=[0 px(1:end-1)]; py = x(2,:);

subplot(4,1,1), plot(px,py,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-4,2])

subplot(4,1,2), plot(px,uopt(2,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.2,0.7])

subplot(4,1,3), plot(px,x(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Vehicle speed [m/s]'),
axis tight, ylim([9.9,11])

subplot(4,1,4), plot(px,uopt(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Acceleration [m/s/s]'),
axis tight, ylim([-2.5,1])
```

Listing 5.5: MATLAB Commands for Example 5.2.1.

```
close all; clear variables; clc
A = [0.7, 0.1; 0 0.1]; B = [1;0]; C = eye(2); Ts = 1;
plant = ss(A,B,C,[],Ts);
p = 2; m = p;  Qu = 3; Qx = [2 1]; xmax = 5;
Weights.ManipulatedVariables = Qu;
Weights.ManipulatedVariablesRate = 0;
Weights.OutputVariables = Qx;
mpcobj = mpc(plant,[],p,m,Weights);
setEstimator(mpcobj,'custom');
setoutdist(mpcobj,'model',tf(zeros(2,1)));
mpcobj.MV.Min = -2; mpcobj.MV.Max = 3;
mpcobj.MV.RateMin = -0.1; mpcobj.MV.RateMax = 0.1;
mpcobj.OV(1).Min = -1; mpcobj.OV(1).Max = xmax;
mpcobj.OV(2).Min = -1; mpcobj.OV(2).Max = xmax;

range = generateExplicitRange(mpcobj);
range.State.Min(:) = [-10;-10]; range.State.Max(:) = [10;10];
range.Reference.Min(:) = [-10;-10]; range.Reference.Max(:) = [10;10];
range.ManipulatedVariable.Min(:)=-1; range.ManipulatedVariable.Max(:)=2;
empcobj = generateExplicitMPC(mpcobj,range);

plotParams = generatePlotParameters(empcobj);
plotParams.State.Index = []; plotParams.State.Value = [];
plotParams.ManipulatedVariable.Index(1) = 1;
plotParams.ManipulatedVariable.Value(1) = 2;
plotParams.MeasuredDisturbance.Index(1) = 1;
plotParams.MeasuredDisturbance.Value(1) = 0;
plotParams.Reference.Index(1) = 1;
plotParams.Reference.Value(1) = 0;
plotParams.Reference.Index(2) = 2;
plotParams.Reference.Value(2) = 0;

plotSection(empcobj,plotParams);
xlabel('x_0'), ylabel('x_1');

xe = mpcstate(empcobj); xe.Plant=[0.2;-0.1]; xe.LastMove = 2;
u_empc = mpcmoveExplicit(empcobj,xe,[],[0;0])
xc = mpcstate(mpcobj); xc.Plant=[0.2;-0.1]; xc.LastMove = 2;
[u_mpc,~] = mpcmove(mpcobj,xc,[],[0;0])

>>
Regions found / unexplored:        36/        0

u_empc =
    1.9000

u_mpc =
    1.9000
```

Listing 5.6: MATLAB Commands for Example 5.3.1.

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; x0 = [1;-2;-0.2]; Ts = 0.2;
t = 0:Ts:5; N = numel(t); A = [1 0 0;0 1 xbar(1)*Ts;0 0 1];
B = [Ts 0;0 0;0 xbar(1)/l*Ts]; C=eye(3); plant=ss(A,B,C,[],Ts); p=5;m=p;

Weights.ManipulatedVariables = [1 10];
Weights.ManipulatedVariablesRate = [0 0];
Weights.OutputVariables = [1 1 1];
mpcobj = mpc(plant,[],p,m,Weights);
setEstimator(mpcobj,'custom');
setoutdist(mpcobj,'model',tf(zeros(3,1)));
mpcobj.MV(2).Min = -0.1; mpcobj.MV(2).Max = 0.1;

range = generateExplicitRange(mpcobj);
range.State.Min(:) = [-3;-3;-3];
range.State.Max(:) = [5;5;5];
range.Reference.Min(:) = [-1;-1;-1];
range.Reference.Max(:) = [2;2;2];
range.ManipulatedVariable.Min(:) = [-1;-1];
range.ManipulatedVariable.Max(:) = [1;1];
empcobj = generateExplicitMPC(mpcobj,range);
xc = mpcstate(empcobj); xc.Plant=x0; uopt = zeros(2,N);
x = zeros(3,N); x(:,1) = x0;
for n = 2:N
    xc.Plant=x(:,n-1);
    uopt(:,n-1) = mpcmoveExplicit(empcobj,xc,[],[0;1;0]);
    x(:,n) = A*x(:,n-1)+B*uopt(:,n-1);
end
uopt(:,n) = uopt(:,n-1); x = x + repmat(xbar,1,N);

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos;
px = cumsum(x(1,:).*cos(x(3,:))*Ts); px=[0 px(1:end-1)]; py = x(2,:);

subplot(4,1,1), plot(px,py,'o-'),
xlabel('Longitudinal position [m]'), ylabel('Lateral position [m]'),
axis tight, ylim([-4,2])
subplot(4,1,2), plot(px,uopt(2,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Steering command [rad]'),
axis tight, ylim([-0.2,0.7])
subplot(4,1,3), plot(px,x(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Vehicle speed [m/s]'),
axis tight, ylim([9.9,11])
subplot(4,1,4), plot(px,uopt(1,:),'o-'),
xlabel('Longitudinal position [m]'), ylabel('Acceleration [m/s/s]'),
axis tight, ylim([-2.5,1])
```

Listing 5.7: MATLAB Commands for Example 5.3.2.

```
close all; clear variables; clc

A = [0.7, 0.1; 0 0.1]; B = [1;0]; C = eye(2); Ts = 1;
plant = ss(A,B,C,[],Ts);

N = 10;
p = 5; m = p;  Qu = 3; Qx = [2 1];

Weights.ManipulatedVariables = Qu;
Weights.ManipulatedVariablesRate = 0;
Weights.OutputVariables = Qx;
mpcobj = mpc(plant,[],p,m,Weights);
setEstimator(mpcobj,'custom');
setoutdist(mpcobj,'model',tf(zeros(2,1)));
mpcobj.MV.Min = -2; mpcobj.MV.Max = 3;
mpcobj.MV.RateMin = -0.1; mpcobj.MV.RateMax = 0.1;
mpcobj.OV(1).Min = -1; mpcobj.OV(1).Max = 5;
mpcobj.OV(2).Min = -1; mpcobj.OV(2).Max = 5;

t_gen = zeros(N,1); t_move = zeros(N,1);

for n = 1 : N
    p = n;
    mpcobj.PredictionHorizon = p; mpcobj.ControlHorizon = p;
    range = generateExplicitRange(mpcobj);
    range.State.Min(:) = [-10;-10];
    range.State.Max(:) = [10;10];
    range.Reference.Min(:) = [-10;-10];
    range.Reference.Max(:) = [10;10];
    range.ManipulatedVariable.Min(:) = -1;
    range.ManipulatedVariable.Max(:) = 2;

    tic, empcobj = generateExplicitMPC(mpcobj,range); t_gen(n) = toc;

    xe = mpcstate(empcobj); xe.Plant=[0.2;-0.1]; xe.LastMove = 2;

    tic
    for k = 1 : 3
        u_empc = mpcmoveExplicit(empcobj,xe,[],[0;0]);
    end
    t_move(n) = toc/3;
end

subplot(211),bar(t_gen),
xlabel('Horizon'),ylabel('EMPC Offline Time [s]')
subplot(212),bar(t_move),
xlabel('Horizon'),ylabel('EMPC Online Time [s]')
```

Listing 5.8: MATLAB Commands for Example 5.3.3.

Chapter 6

Nonlinear Model Predictive Control

Consider again the parametric optimization problem (5.1), which is rephrased as follows.

$$\min_{\mathbf{u}} \quad \sum_{k=t}^{t+p-1} \ell(x_k, u_k) + V_f(x_p) \qquad \text{subject to} \quad \begin{cases} x_{k+1} = f(x_k, u_k), \\ \mathbf{x} \in \mathcal{X}, \quad \mathbf{u} \in \mathcal{U} \\ x(t) = x_0 \end{cases} \tag{6.1}$$

where p is called prediction horizon, and the input sequence and state sequence are defined as

$$\mathbf{u} = \begin{bmatrix} u_t \\ u_{t+1} \\ \vdots \\ u_{t+p-1} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_{t+1} \\ x_{t+2} \\ \vdots \\ x_{t+p} \end{bmatrix}.$$

Problem (6.1) is solved for each time step t, and only the first element of $\mathbf{u}$, i.e., u_t, is implemented by the actuators. Then time advances by 1 time step, a new initial state x_0 is obtained, and MPC solves a new optimization problem (6.1) with updated x_0.

In Chapter 5, we have treated (6.1) by restricting both $\ell(x, u)$ and $V_f(x)$ to quadratic functions, and restricting the system dynamics $f(x, u)$ and all constraints to be linear. Such assumption is necessary for MPC to be implementable in embedded systems, but it can often cause performance degradation. In this chapter, we deal with the case without such assumptions, which if often referred to as nonlinear MPC (or NMPC). Note that for NMPC, either the dynamics or constraints are nonlinear, or the cost function is not linear/quadratic. We start this chapter by discussing sequential quadratic programming, or SQP, a technique to solve nonlinear optimization problem, followed by the techniques for NMPC. This chapter will be concluded by discussing linear time-varying MPC, a technique that balances between linear MPC and nonlinear MPC.

6.1 Sequential Quadratic Programming

Consider the following generic nonlinear optimization problem

$$\min_{x} \quad f(x) \qquad \text{subject to} \begin{cases} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \le 0, & i \in \mathcal{I}, \end{cases} \tag{6.2}$$

where the objective function f and/or constraints c_i can be nonlinear. Such nonlinear optimization problem can be solved by sequential quadratic programming, or SQP.

Given a current estimate x_k of the solution x^*, the objective function can be approximated through Taylor expansion, as follows

$$\begin{aligned} f(x) =& f(x_k) + \left(\frac{d}{dx}f(x)\Big|_{x=x_k}\right)^T (x-x_k) + \frac{1}{2}(x-x_k)^T \frac{d^2}{dx^2}f(x)\Big|_{x=x_k} (x-x_k) + \text{ higher order terms} \\ \approx& f(x_k) + \left(\frac{d}{dx}f(x)\Big|_{x=x_k}\right)^T (x-x_k) + \frac{1}{2}(x-x_k)^T \frac{d^2}{dx^2}f(x)\Big|_{x=x_k} (x-x_k) \\ =& f(x_k) + \left(\frac{d}{dx}f(x)\Big|_{x=x_k}\right)^T p + \frac{1}{2}p^T \frac{d^2}{dx^2}f(x)\Big|_{x=x_k} p \end{aligned}$$

where we define $p = x - x_k$. In other words, using the above approximation, optimizing $f(x)$ is equivalent to optimize its quadratic representation over p. Similarly, the constraints can be approximated through first order Taylor expansion, as follows

$$\begin{aligned} c_i(x) =& c_i(x_k) + \left(\frac{d}{dx}c_i(x)\Big|_{x=x_k}\right)^T (x-x_k) + \text{ higher order terms} \\ \approx& c_i(x_k) + \left(\frac{d}{dx}c_i(x)\Big|_{x=x_k}\right)^T (x-x_k) = c_i(x_k) + \left(\frac{d}{dx}c_i(x)\Big|_{x=x_k}\right)^T p \end{aligned}$$

Denote $f(x_k)$ as f_k, $\frac{d}{dx}f(x)\big|_{x=x_k}$ as f_k', $\frac{d^2}{dx^2}f(x)\Big|_{x=x_k}$ as f_k'', $c_i(x_k)$ as c_{ik}, $\frac{d}{dx}c_i(x)\big|_{x=x_k}$ as c_{ik}'. Then (6.2) is approximately equivalent to

$$\min_p \quad f_k + f_k'^T p + \frac{1}{2}p^T f_k'' p \qquad \text{subject to } \begin{cases} c_{ik} + c_{ik}'^T p = 0, & i \in \mathcal{E}, \\ c_{ik} + c_{ik}'^T p \le 0, & i \in \mathcal{I}, \end{cases} \tag{6.3}$$

which is a standard QP problem that can be solved as discussed in Section 3.4. However, since (6.3) is only an approximation of (6.2), the optimal solution p^* to (6.3) does not necessarily lead us to the optimal solution of (6.2), but at least closer. Therefore, once p^* is obtained, we set $x_{k+1} = x_k + p$, and repeat the above process to obtain a new QP approximation. The whole process repeats until $p^* \approx 0$, i.e., when x_k converges.

Example 6.1.1 (Sequential QP)**.** *Consider the following nonlinear optimization problem*

$$\min_{x_1,x_2,x_3} \quad x_1^2 + x_2^3 + 6x_3^3 \qquad \textit{subject to } \begin{cases} x_2^3 = 5 \\ x_1^2 \ge 2 \\ x_3^3 \ge 1, \end{cases}$$

By inspection, it is clear that $x_1^* = \pm\sqrt{2}$, $x_2 = \sqrt[3]{5}$, and $x_3 = 1$ is the optimal solution. To solve it, we assume the current iterate is $x_k = \begin{bmatrix} x_{1k} & x_{2k} & x_{3k} \end{bmatrix}^T$. Then we have

$$\begin{aligned} f_k =& x_{1k}^2 + x_{2k}^3 + 6x_{3k}^3 \\ f_k' =& \begin{bmatrix} 2x_{1k} \\ 3x_{2k}^2 \\ 18x_{3k}^2 \end{bmatrix} \end{aligned}$$

$$
\begin{aligned}
f_k'' &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 6x_{2k} & 0 \\ 0 & 0 & 36x_{3k} \end{bmatrix} \\
c_{1k} &= x_{2k}^3 \\
c_{1k}' &= \begin{bmatrix} 0 \\ 3x_{2k}^2 \\ 0 \end{bmatrix} \\
c_{2k} &= x_{1k}^2 \\
c_{2k}' &= \begin{bmatrix} 2x_{1k} \\ 0 \\ 0 \end{bmatrix} \\
c_{3k} &= x_{3k}^3 \\
c_{3k}' &= \begin{bmatrix} 0 \\ 0 \\ 3x_{3k}^2 \end{bmatrix}
\end{aligned}
$$

Therefore, given x_k, one can solve the following QP on p to find an approximation of the optimal solution.

$$
\begin{aligned}
\min_{p} \quad & \begin{bmatrix} 2x_{1k} & 3x_{2k}^2 & 18x_{3k}^2 \end{bmatrix} p + \frac{1}{2} p^T \begin{bmatrix} 2 & 0 & 0 \\ 0 & 6x_{2k} & 0 \\ 0 & 0 & 36x_{3k} \end{bmatrix} p \\
\text{subject to} \quad & \begin{cases} x_{2k}^3 + \begin{bmatrix} 0 & 3x_{2k}^2 & 0 \end{bmatrix} p = 5 \\ x_{1k}^2 + \begin{bmatrix} 2x_{1k} & 0 & 0 \end{bmatrix} p \geq 2 \\ x_{3k}^3 + \begin{bmatrix} 0 & 0 & 3x_{3k}^2 \end{bmatrix} p \geq 1 \end{cases}
\end{aligned}
$$

Let's start with $x^0 = \begin{bmatrix} 2 & 2 & 2 \end{bmatrix}^T$[1]. In this case, the QP at hand is give as

$$
\begin{aligned}
\min_{p} \quad & \begin{bmatrix} 4 & 12 & 72 \end{bmatrix} p + \frac{1}{2} p^T \begin{bmatrix} 2 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 72 \end{bmatrix} p \\
\text{subject to} \quad & \begin{cases} 8 + \begin{bmatrix} 0 & 12 & 0 \end{bmatrix} p = 5 \\ 4 + \begin{bmatrix} 4 & 0 & 0 \end{bmatrix} p \geq 2 \\ 8 + \begin{bmatrix} 0 & 0 & 12 \end{bmatrix} p \geq 1 \end{cases}
\end{aligned}
$$

Solving the above QP yields $p^{0*} = \begin{bmatrix} -0.5 & -0.25 & -0.5833 \end{bmatrix}^T$. Now we move to the next iteration and get $x^1 = x^0 + p^{0*} = \begin{bmatrix} 1.5 & 1.75 & 1.4167 \end{bmatrix}^T$, which is used to form another QP. In fact, the

[1] Here we use superscript to denote the iteration to avoid confusion with the decision variables.

solutions to the sequential QPs are listed as follows.

$$p^{1*} = \begin{bmatrix} -0.0833 \\ -0.0391 \\ -0.3061 \end{bmatrix}, \quad x^2 = \begin{bmatrix} 1.4167 \\ 1.7109 \\ 1.1105 \end{bmatrix}$$
$$p^{2*} = \begin{bmatrix} -0.0025 \\ -0.0009 \\ -0.0999 \end{bmatrix}, \quad x^3 = \begin{bmatrix} 1.4142 \\ 1.71 \\ 1.0106 \end{bmatrix}$$
$$p^{3*} = \begin{bmatrix} 0 \\ 0 \\ -0.0105 \end{bmatrix}, \quad x^4 = \begin{bmatrix} 1.4142 \\ 1.71 \\ 1.0001 \end{bmatrix}$$
$$p^{4*} = \begin{bmatrix} 0 \\ 0 \\ -0.0001115 \end{bmatrix}, \quad x^5 = \begin{bmatrix} 1.4142 \\ 1.71 \\ 1 \end{bmatrix}$$

In other words, the SQP converges to the optimal solution within 5 iterations.

Note that SQP can sometimes lead to degraded QP. For example, let $x^k = \begin{bmatrix} 2 & 2 & 0 \end{bmatrix}^T$. Then the approximation of the last constraint becomes

$$0 + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} p \geq 1,$$

which is infeasible.

Example 6.1.2 (SQP may breaks down)**.** *Consider the following optimization problem*

$$\min_{x_1, x_2} \quad -x_1 - \frac{1}{2}x_2^2 \qquad \textit{subject to} \quad x_1^2 + x_2^2 = 1.$$

It is not difficult to see that $\begin{bmatrix} 1 & 0 \end{bmatrix}^T$ *is the optimal solution. Assume that the* $x^k = \begin{bmatrix} 1+\epsilon & 0 \end{bmatrix}^T$*. Then the QP subproblem is given as*

$$\min_{p} \quad \begin{bmatrix} -1 & 0 \end{bmatrix} p + \frac{1}{2} p^T \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} p = -p_1 - \frac{1}{2}p_2^2 \qquad \textit{subject to } (1+\epsilon)^2 + 2(1+\epsilon)p_1 = -1$$

Since there is no constraint on p_2*, it is clear that the above QP is unbounded.*

6.2 Nonlinear MPC

As mentioned earlier, the nonlinear MPC solves a nonlinear optimization problem of (6.1), which is rephrased as follows.

$$\min_{\mathbf{u},\mathbf{x}} \quad \sum_{k=t}^{t+p-1} \ell(x_k, u_k) + V_f(x_p) \qquad \text{subject to} \quad \begin{cases} x_{k+1} = f(x_k, u_k), \\ h(x_k, u_k) \leq 0 \\ x(t) = x_0 \end{cases} \tag{6.4}$$

Given a current iterate (x^k, u^k), then the QP subproblem is given as follows.

$$\min_{\mathbf{u},\mathbf{x}} \quad \sum_{k=t}^{t+p-1} \left[\frac{\partial \ell(x_k^k, u_k^k)}{\partial x_k} \right]^T (x_k - x_k^k) + \sum_{k=t}^{t+p-1} (x_k - x_k^k)^T \left[\frac{\partial^2 \ell(x_k^k, u_k^k)}{\partial x_k^2} \right] (x_k - x_k^k)$$

$$
\begin{aligned}
&+\sum_{k=t}^{t+p-1}\left[\frac{\partial \ell(x_k^k,u_k^k)}{\partial u_k}\right]^T(u_k-u_k^k)+\sum_{k=t}^{t+p-1}(u_k-u_k^k)^T\left[\frac{\partial^2 \ell(x_k^k,u_k^k)}{\partial u_k^2}\right](u_k-u_k^k)\\
&+\left[\frac{\partial V(x_p^k)}{\partial x_p}\right]^T(x_p-x_p^k)+(x_p-x_p^k)^T\left[\frac{\partial^2 V(x_p^k)}{\partial x_p^2}\right](x_p-x_p^k)
\end{aligned}
$$

$$
\text{subject to}\quad
\begin{cases}
x_{k+1}=f(x_k^k,u_k^k)+A^k(x_k-x_k^k)+B^k(u_k-u_k^k),\\
h(x_k^k,u_k^k)+\left[\frac{\partial h(x_k^k,u_k^k)}{\partial x_k^k}\right]^T(x_k-x_k^k)+\left[\frac{\partial h(x_k^k,u_k^k)}{\partial u_k^k}\right]^T(u_k-u_k^k)\le 0\\
x(t)=x_0
\end{cases}
$$

Note that the above problem is QP as it can be cast into a standard QP formulation as discussed in Chapter 5.

Example 6.2.1 (Nonlinear MPC)**.** *Consider the following dynamical system*

$$x_{k+1}=-x_k^2+x_ku_k$$

and suppose we want to regulate the system back to its origin. Therefore, the following NMPC is formulated.

$$
\min_{\boldsymbol{u},\boldsymbol{x}}\quad \frac{1}{2}\sum_{k=1}^{3}x_k^2+\frac{1}{2}\sum_{k=0}^{2}u_k^2 \qquad \textit{subject to}\quad
\begin{cases}
x_{k+1}=-x_k^2+x_ku_k,\\
-1\le u_k\le 1\\
x(t)=x_0
\end{cases}
$$

Given current iterate (x^k,u^k)*, define* $\boldsymbol{p}=\begin{bmatrix}x_1-x_1^k & x_2-x_2^k & x_3-x_3^k & u_0-u_0^k & u_1-u_1^k & u_2-u_2^k\end{bmatrix}^T$*. Then the following QP subproblem is constructed.*

$$
\min_{\boldsymbol{p}}\quad \begin{bmatrix}x_1^k & x_2^k & x_3^k & u_0^k & u_1^k & u_2^k\end{bmatrix}\boldsymbol{p}+\boldsymbol{p}^T\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{bmatrix}\boldsymbol{p}
$$

$$
\textit{subject to}\quad
\begin{cases}
\begin{bmatrix}x_1^k-x_0u_0^k\\ x_2^k+\left(x_1^k\right)^2-x_1^ku_1^k\\ x_3^k+\left(x_2^k\right)^2-x_2^ku_2^k\end{bmatrix}+\begin{bmatrix}1&0&0&-x_0&0&0\\2x_1^k&1&0&0&-x_1^k&0\\0&2x_2^k&1&0&0&-x_2^k\end{bmatrix}\boldsymbol{p}=\begin{bmatrix}-x_0^2\\0\\0\end{bmatrix}\\
\begin{bmatrix}-1\\-1\\-1\end{bmatrix}\le\begin{bmatrix}u_0^k\\u_1^k\\u_2^k\end{bmatrix}+\begin{bmatrix}0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{bmatrix}\boldsymbol{p}\le\begin{bmatrix}1\\1\\1\end{bmatrix}
\end{cases}
$$

or equivalently

$$
\min_{\boldsymbol{p}}\quad \begin{bmatrix}x_1^k & x_2^k & x_3^k & u_0^k & u_1^k & u_2^k\end{bmatrix}\boldsymbol{p}+\boldsymbol{p}^T\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{bmatrix}\boldsymbol{p}
$$

$$
\textit{subject to} \quad \begin{cases} \begin{bmatrix} 1 & 0 & 0 & -x_0 & 0 & 0 \\ 2x_1^k & 1 & 0 & 0 & -x_1^k & 0 \\ 0 & 2x_2^k & 1 & 0 & 0 & -x_2^k \end{bmatrix} \boldsymbol{p} = \begin{bmatrix} -x_0^2 - x_1^k + x_0 u_0^k \\ -x_2^k - \left(x_1^k\right)^2 + x_1^k u_1^k \\ -x_3^k - \left(x_2^k\right)^2 + x_2^k u_2^k \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{p} \leq \begin{bmatrix} 1 - u_0^k \\ 1 - u_1^k \\ 1 - u_2^k \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \boldsymbol{p} \leq \begin{bmatrix} 1 + u_0^k \\ 1 + u_1^k \\ 1 + u_2^k \end{bmatrix} \end{cases}
$$

Using the Matlab Code Listing 6.2, the SQP returns the following solution when $x_0 = -1$.

$$
x^* = \begin{bmatrix} -0.4302 \\ -0.155 \\ -0.0235 \end{bmatrix} \qquad u^* = \begin{bmatrix} -0.5698 \\ -0.0698 \\ -0.0036 \end{bmatrix}
$$

Matlab Code Listing 6.2 also lists the use of Matlab's built-in nonlinear MPC function, which returns the following solutions

$$
x^* = \begin{bmatrix} -0.4343 \\ -0.1575 \\ -0.0242 \end{bmatrix} \qquad u^* = \begin{bmatrix} -0.5657 \\ -0.0717 \\ -0.0038 \end{bmatrix}
$$

The closeness of the solutions returned by the two methods can be clearly observed.

6.3 Linear Time-Varying MPC

Consider a special nonlinear MPC where the system dynamics is nonlinear, while the cost function is quadratic and the constraints are linear, as follows.

$$
\min_{\mathbf{u},\mathbf{x}} \quad \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x} + \frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u} \qquad \text{subject to} \quad \begin{cases} x_{k+1} = f(x_k, u_k), \\ x_{\min} \leq x_k \leq x_{\max} \\ u_{\min} \leq u_k \leq u_{\max} \\ \Delta_{\min} \leq u_k - u_{k-1} \leq \Delta_{\max} \\ x(t) = x_0 \end{cases} \tag{6.5}
$$

Then the linear time-varying MPC (LTV MPC), or sometimes referred to as real-time iteration, performs the linearization of (6.5) **once** for each time step. The primary goal here is to reduce online computation required by the SQP approach. In other words, given current state estimate x_0 and last control move u_{-1}, LTV MPC solves the following OCP.

$$
\min_{\mathbf{u},\mathbf{x}} \quad \frac{1}{2}\sum_{k=1}^{p} \|x_k\|_{Q_x} + \frac{1}{2}\sum_{k=0}^{p-1} \|u_k\|_{Q_u} \tag{6.6a}
$$

$$
\text{subject to} \quad \begin{cases} x_{k+1} = f(x_0, u_0) + A_t(x_k - x_0) + B_t(u_k - u_0), \\ x_{\min} \leq x_k \leq x_{\max} \\ u_{\min} \leq u_k \leq u_{\max} \\ \Delta_{\min} \leq u_k - u_{k-1} \leq \Delta_{\max} \\ x(t) = x_0 \end{cases} \tag{6.6b}
$$

Denote $f_t = f(x_0, u_0) - A_0 x_0 - B_0 u_0$. Then the linearized system dynamics is given by

$$x_{k+1} = f_t + A_t x_k + B_t u_k \tag{6.7}$$

The above OCP (6.6) can be cast into a standard (dense) QP problem as follows.

To derive a relationship between x_k and the control inputs u_0, u_1, $\cdots$, u_{k-1}, we have,

$$\begin{aligned}
x_1 &= A_t x_0 + B_t u_0 + f_t \\
x_2 &= A_t x_1 + B_t u_1 + f_t = A_t(A_t x_0 + B_t u_0 + f_t) + B_t u_1 + f_t = A_t^2 x_0 + A_t B_t u_0 + B_t u_1 + A_t f_t + f_t \\
x_3 &= A_t x_2 + B_t u_2 + f_t = A_t(A_t x_1 + B_t u_1 + f_t) + B_t u_2 + f_t \\
&= A_t^3 x_0 + A_t^2 B_t u_0 + A_t B_t u_1 + B_t u_2 + A_t^2 f_t + A_t f_t + f_t \\
&\vdots \\
x_k &= A_t x_{k-1} + B_t u_{k-1} + f_t = A_t(A_t x_{k-2} + B_t u_{k-2} + f_t) + B_t u_{k-1} + f_t \\
&= A_t^2 x_{k-2} + A_t B_t u_{k-2} + B_t u_{k-1} + A_t f_t + f_t \\
&= A_t^2(A_t x_{k-3} + B_t u_{k-3} + f_t) + A_t B_t u_{k-2} + B_t u_{k-1} + A_t f_t + f_t \\
&= A_t^3 x_{k-3} + A_t^2 B_t u_{k-3} + A_t B_t u_{k-2} + B_t u_{k-1} + A^2 f_t + A_t f_t + f_t \\
&\vdots \\
&= A_t^k x_0 + \sum_{n=1}^{k} A_t^{n-1} B_t u_{k-n} + \sum_{n=1}^{k} A_t^{n-1} f_t
\end{aligned}$$

Putting it into matrix form, we have

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_p \end{bmatrix} = \underbrace{\begin{bmatrix} B_t & 0 & 0 & \cdots & 0 \\ A_t B_t & B_t & 0 & \cdots & 0 \\ A_t^2 B_t & A_t B_t & B_t & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t^{p-1} B_t & A_t^{p-2} B_t & A_t^{p-3} B_t & \cdots & B_t \end{bmatrix}}_{M_{AB}} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{p-1} \end{bmatrix} + \underbrace{\begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^p \end{bmatrix}}_{M_{Ak}} x_0$$
$$+ \underbrace{\begin{bmatrix} I & 0 & 0 & \cdots & 0 \\ A_t & I & 0 & \cdots & 0 \\ A_t^2 & A_t & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_t^{p-1} & A_t^{p-2} & A_t^{p-3} & \cdots & I \end{bmatrix} \begin{bmatrix} f_t \\ f_t \\ f_t \\ \vdots \\ f_t \end{bmatrix}}_{\mathbf{f}}$$

Or equivalently

$$\mathbf{x} = M_{AB}\mathbf{u} + M_{Ak} x_0 + \mathbf{f}$$

The OCP (6.7) can now be equivalently represented as

$$\min_{\mathbf{u}} \quad \sum_{k=1}^{p} \frac{1}{2} \|x_k\|_{Q_x}^2 + \sum_{k=0}^{p-1} \frac{1}{2} \|u_k\|_{Q_u}^2 \qquad \text{subject to} \quad \begin{cases} \mathbf{x} = M_{AB}\mathbf{u} + M_{Ak} x_0 + \mathbf{f} \\ x_{\min} \le x_k \le x_{\max} \\ u_{\min} \le u_k \le u_{\max} \\ \Delta_{\min} \le u_k - u_{k-1} \le \Delta_{\max} \end{cases} \tag{6.8}$$

Denote the cost function of (6.8) as

$$J=\underbrace{\sum_{k=1}^{p}\frac{1}{2}\left\|x_k\right\|_{Q_x}^2}_{J_x}+\underbrace{\sum_{k=0}^{p-1}\frac{1}{2}\left\|u_k\right\|_{Q_u}^2}_{J_u}$$

Then we have

$$\begin{aligned}J_x=&\sum_{k=1}^{p}\frac{1}{2}\left\|x_k\right\|_{Q_x}^2=\frac{1}{2}\sum_{k=1}^{p}x_k^TQ_xx_k=\frac{1}{2}\begin{bmatrix}x_1\\x_2\\\vdots\\x_p\end{bmatrix}^T\underbrace{\begin{bmatrix}Q_x&0&0&\cdots&0\\0&Q_x&0&\cdots&0\\0&0&Q_x&\cdots&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&\cdots&Q_x\end{bmatrix}}_{M_x}\begin{bmatrix}x_1\\x_2\\\vdots\\x_p\end{bmatrix}=\frac{1}{2}\mathbf{x}^TM_x\mathbf{x}\\=&\frac{1}{2}(M_{AB}\mathbf{u}+M_{Ak}x_0+\mathbf{f})^TM_x(M_{AB}\mathbf{u}+M_{Ak}x_0+\mathbf{f})\\=&\frac{1}{2}\mathbf{u}^TM_{AB}^TM_xM_{AB}\mathbf{u}+\left(x_0^TM_{Ak}^T+\mathbf{f}^T\right)M_xM_{AB}\mathbf{u}+\underbrace{\frac{1}{2}\left(x_0^TM_{Ak}^T+\mathbf{f}^T\right)M_x(M_{Ak}x_0+\mathbf{f})}_{\text{Independent of }\mathbf{u}}\end{aligned}$$

Ignoring the term independent of $\mathbf{u}$, we have

$$J_x=\frac{1}{2}\mathbf{u}^T\underbrace{M_{AB}^TM_xM_{AB}}_{H_x}\mathbf{u}+\underbrace{\left(x_0^TM_{Ak}^T+\mathbf{f}^T\right)M_xM_{AB}}_{f_x}\mathbf{u}. \tag{6.9}$$

Similarly to the case of linear MPC, we have

$$J_u=\sum_{k=0}^{p-1}\frac{1}{2}\left\|u_k\right\|_{Q_u}^2=\frac{1}{2}\sum_{k=0}^{p-1}u_k^TQ_uu_k=\frac{1}{2}\begin{bmatrix}u_0\\u_1\\\vdots\\u_{p-1}\end{bmatrix}^T\underbrace{\begin{bmatrix}Q_u&0&0&\cdots&0\\0&Q_u&0&\cdots&0\\0&0&Q_u&\cdots&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&\cdots&Q_u\end{bmatrix}}_{M_u}\begin{bmatrix}u_0\\u_1\\\vdots\\u_{p-1}\end{bmatrix}=\frac{1}{2}\mathbf{u}^TM_u\mathbf{u} \tag{6.10}$$

Putting (6.9) and (6.10) together, we have

$$J=J_x+J_u=\frac{1}{2}\mathbf{u}^T\left(H_x+M_u\right)\mathbf{u}+f_x\mathbf{u} \tag{6.11}$$

The state constraints $x_{\min}\le x_k\le x_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix}x_{\min}\\x_{\min}\\\vdots\\x_{\min}\end{bmatrix}\le\begin{bmatrix}x_1\\x_2\\\vdots\\x_p\end{bmatrix}\le\begin{bmatrix}x_{\max}\\x_{\max}\\\vdots\\x_{\max}\end{bmatrix}$$

or equivalently

$$\mathbf{x}_{\min}\le M_{AB}\mathbf{u}+M_{Ak}x_0+\mathbf{f}\le\mathbf{x}_{\max}$$

Therefore we have

$$\mathbf{x}_{\min} - M_{Ak}x_0 - \mathbf{f} \leq M_{AB}\mathbf{u} \leq \mathbf{x}_{\max} - M_{Ak}x_0 - \mathbf{f} \tag{6.12}$$

Similarly the input constraints $u_{\min} \leq u_k \leq u_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix} u_{\min} \\ u_{\min} \\ \vdots \\ u_{\min} \end{bmatrix} \leq \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{p-1} \end{bmatrix} \leq \begin{bmatrix} u_{\max} \\ u_{\max} \\ \vdots \\ u_{\max} \end{bmatrix}$$

or equivalently

$$\mathbf{u}_{\min} \leq \mathbf{u} \leq \mathbf{u}_{\max} \tag{6.13}$$

Lastly, the input rate constraint $\Delta_{\min} \leq u_k - u_{k-1} \leq \Delta_{\max}$ can be expressed in matrix form as:

$$\begin{bmatrix} \Delta_{\min} \\ \Delta_{\min} \\ \Delta_{\min} \\ \vdots \\ \Delta_{\min} \end{bmatrix} \leq \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{bmatrix}}_{D} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{p-1} \end{bmatrix} + \underbrace{\begin{bmatrix} -u_{-1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{r_u} \leq \begin{bmatrix} \Delta_{\max} \\ \Delta_{\max} \\ \Delta_{\max} \\ \vdots \\ \Delta_{\max} \end{bmatrix}$$

Therefore,

$$\boldsymbol{\Delta}_{\min} - r_u \leq D\mathbf{u} \leq \boldsymbol{\Delta}_{\max} - r_u \tag{6.14}$$

Putting (6.11), (6.12), (6.13) and (6.14) together, we have the following QP formulation for (6.6):

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T\left(H_x + M_u\right)\mathbf{u} + f_x\mathbf{u} \qquad \text{subject to} \quad \begin{cases} M_{AB}\mathbf{u} \leq \mathbf{x}_{\max} - M_{Ak}x_0 - \mathbf{f} \\ -M_{AB}\mathbf{u} \leq -\mathbf{x}_{\min} + M_{Ak}x_0 + \mathbf{f} \\ \mathbf{u} \leq \mathbf{u}_{\max} \\ -\mathbf{u} \leq -\mathbf{u}_{\min} \\ D\mathbf{u} \leq \boldsymbol{\Delta}_{\max} - r_u \\ -D\mathbf{u} \leq -\boldsymbol{\Delta}_{\min} + r_u \end{cases} \tag{6.15}$$

Remark 6.3.1. *The LTV MPC can be considered as a good compromise between linear MPC and nonlinear MPC. Instead of using a fixed linearized model, LTV MPC linearizes the dynamic in real-time to improve the accuracy, with an increase of online computation. Compared to nonlinear MPC that requires sequential linearization and QP solving, which can be very time-consuming, the LTV MPC only linearizes the system once every time step. In other words, the intermediate solution of an SQP approach will actually be sent out to implement at the actuators.*

6.4 Matlab Codes

```
close all; clear variables; clc

x0 = [2;2;0];

H = eye(3); H(1,1) = 2;
f = zeros(3,1); A = zeros(2,3); b  = zeros(2,1); Aeq = zeros(1,3);
xk = x0;
for n = 1 : 10
    H(2,2) = 6*xk(2);
    H(3,3) = 36*xk(3);
    f(1) = 2*xk(1);
    f(2) = 3*xk(2)^2;
    f(3) = 18*xk(3)^2;
    A(1,3) = -3*xk(3)^2;
    A(2,1) = -2*xk(1);
    b(1) = xk(3)^3-1;
    b(2) = xk(1)^2-2;
    Aeq(2) = 3*xk(2)^2;
    beq = 5-xk(2)^3;
    p = quadprog(H,f,A,b,Aeq,beq)
    xk = xk + p

end
```

Listing 6.1: MATLAB Commands for Example 6.1.1.

```
close all; clear variables; clc

x0 = -1; um1 = 0.5;
xk = [x0;x0;x0]; uk = [um1;um1;um1];

H = eye(6);
f = zeros(6,1); Aeq = zeros(3,6); beq  = zeros(3,1);
for n = 1 : 10
    f = [xk;uk];
    Aeq(1,1)=1; Aeq(1,4) = -x0;
    Aeq(2,1)=2*xk(1); Aeq(2,2)=1; Aeq(2,5)=-xk(1);
    Aeq(3,2)=2*xk(2); Aeq(3,3)=1; Aeq(3,6)=-xk(2);
    beq(1) = -x0^2-xk(1)+uk(1)*x0;
    beq(2) = -xk(2)-xk(1)^2+uk(2)*xk(1);
    beq(3) = -xk(3)-xk(2)^2+uk(3)*xk(2);

    ub = [inf;inf;inf;1-uk];
    lb = [-inf;-inf;-inf;-1-uk];

    p = quadprog(H,f,[],[],Aeq,beq,lb,ub);
    xk = xk + p(1:3);
    uk = uk + p(4:6);
    [xk;uk]
    1;
end

nlobj = nlmpc(1,1,1);
nlobj.p=3;
nlobj.ControlHorizon=3;
nlobj.Model.StateFcn  = @(x,u) -x^2+x*u;
nlobj.Model.OutputFcn = @(x,u) x;
nlobj.Model.IsContinuousTime = false;
nlobj.Model.NumberOfParameters = 0;
nlobj.Weights.ManipulatedVariables=1;
nlobj.Weights.OutputVariables=1;
nlobj.Weights.ECR=0.001;
nlobj.Weights.ManipulatedVariablesRate=0;
nlobj.ManipulatedVariables.Min=-1;
nlobj.ManipulatedVariables.Max=1;
validateFcns(nlobj, 2, 1);
[mv,~,info] = nlmpcmove(nlobj,x0,0,[0;0;0]);
[info.Xopt(2:end);info.MVopt(1:3)]
```

Listing 6.2: MATLAB Commands for Example 6.2.1.

Chapter 7

State Estimation

Previous chapters have assumed that full state feedback is available. Recall that (5.1), which is reproduced below, requires that x_0 is available to form the OCP.

$$\min_{\mathbf{u}} \quad \sum_{k=t}^{t+p-1} \ell(x_k, u_k) + V_f(x_p) \qquad \text{subject to} \quad \begin{cases} x_{k+1} = f(x_k, u_k), \\ \mathbf{x} \in \mathcal{X}, \quad \mathbf{u} \in \mathcal{U} \\ x(t) = x_0 \end{cases}$$

The assumption that x_0 is available is restrictive and not practical for many reasons. First, not all the states are measurable. Some internal states may be hard to measured, while some other internal states may not have physical meaning at all. Second, though some states are measurable, the cost of sensors may be too high so that it is often economically effective to not measure such states. Third, though measurements for some states are available, there can be measurement noise that render such measurement unreliable. Therefore, for MPC (and even LQR), it is often to design a state estimator to estimate a reasonable value for current state, given the input and output sequence so far. More formally, consider the following dynamic systems,

$$\begin{aligned} x_{k+1} =& f(x_k, u_k) + w_k \\ y_k =& g(x_k) + v_k \end{aligned}$$

and the observation of $\mathbf{u}_t = \begin{bmatrix} u_0 & u_1 & \cdots u_t \end{bmatrix}^T$ and $\mathbf{y}_t = \begin{bmatrix} y_0 & y_1 & \cdots y_t \end{bmatrix}^T$. Then the goal of state estimation is to find $\mathbf{x}_t = \begin{bmatrix} x_0 & x_1 & \cdots & x_t \end{bmatrix}^T$ that best explains $\mathbf{y}_t$. Note that here w_k is the process noise satisfying Gaussian distribution with mean 0 and covariance Q, v_k is the measurement noise satisfying Gaussian distribution with mean 0 and covariance R. In addition, the initial condition $x_0 \sim \mathcal{N}(\bar{x}_0, P_0)$, i,e, the initial condition satisfying Gaussian distribution with mean $\bar{x}_0$ and covariance P_0. Furthermore,

1. Q denotes the variance of the process noise. Q is large when the state is subjected to large disturbance. On the other hand, when the state is subject to small disturbance, Q is small.

2. R denotes the variance of the measurement noise. R is small when the measurements are highly accurate. On the other hand, if the measurements are quite noisy, then R is large.

3. $\bar{x}_0$ and P_0 reflects the guess of initial condition, when no measurement has been observed. In reality, it is quite common that neither $\bar{x}_0$ nor P_0 is known in advance. Therefore in such situation, we often set $\bar{x}_0 = 0$ and P_0 to a very large value.

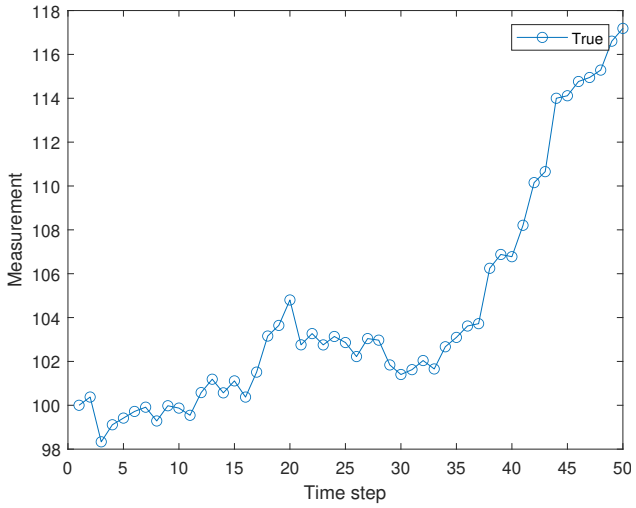

Figure 7.1: Measurement data for Example 7.0.1.

Now, the set of input $\mathbf{u}_t = \begin{bmatrix} u_0 & u_1 & \cdots u_t \end{bmatrix}^T$ and output $\mathbf{y}_t = \begin{bmatrix} y_0 & y_1 & \cdots y_t \end{bmatrix}^T$, we first formulate the following objective function to measure the goodness of $\mathbf{x}_t = \begin{bmatrix} x_0 & x_1 & \cdots & x_t \end{bmatrix}^T$:.

$$V_t(\mathbf{x}_t) = \frac{1}{2}\|x_0 - \bar{x}_0\|^2_{P_0^{-1}} + \frac{1}{2}\sum_{k=0}^{t-1}\|x_{k+1} - f(x_k, u_k)\|^2_{Q^{-1}} + \frac{1}{2}\sum_{k=0}^{t}\|y_k - g(x_k)\|^2_{R^{-1}} \tag{7.1}$$

where

1. the first term measures the distance between estimate $\hat{x}_0$ to our prior knowledge $\bar{x}_0$. When P_0 is large, indicating the prior knowledge is unreliable, the first term has less contribution to the total index functon V_t.

2. the second term measures the distance between $\mathbf{x}_t$ and the (nominal) model (7.3). When the state is subject to large disturbance, satisfying (7.3) becomes less meaningful. In this case Q is large, and so the contribution of second term is small.

3. the third term measures the distance between model predicted output $C\mathbf{x}_t$ and measurement $\mathbf{y}_t$. When the measurement is unreliable, then R is large, making the contribution of the third term small.

Then the goal is to find $\mathbf{x}_t$ by solving the following optimization problem

$$\min_{\mathbf{x}_t} \quad V_t(\mathbf{x}_t) = \frac{1}{2}\|x_0 - \bar{x}_0\|^2_{P_0^{-1}} + \frac{1}{2}\sum_{k=0}^{t-1}\|x_{k+1} - f(x_k, u_k)\|^2_{Q^{-1}} + \frac{1}{2}\sum_{k=0}^{t}\|y_k - g(x_k)\|^2_{R^{-1}}\,. \tag{7.2}$$

Example 7.0.1 (State Estimation Motivation and Clarification)**.** *Consider the measurement data as plotted in Fig. 7.1, which seems to be noisy. To filter it, one can choose various digital filter to smooth data. However, state estimation is different from digital filter, in the sense that a dynamical model is in place to capture the evolution of the states. The purpose of state estimation is therefore to estimate the state, as opposed to filtering the measurement data. However, as can be shown later, one can use various state estimation methods to perform the same task of digital filter.*

7.1 Kalman Filter

For linear system, Kalman filter is a very popular choice for state estimation and has been proven to be robust and valuable, with minimum amount of calibrations needed. Kalman filter assumes the system dynamics to be

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{7.3}$$
$$y_k = Cx_k + v_k \tag{7.4}$$

where w_k is the process noise and v_k is the measurement noise as discussed above. To find state estimate $\mathbf{x}_t$, one needs to solve the optimization problem (7.2), which reduces to following least square problem:

$$\min_{\mathbf{x}_t} \quad V_t(\mathbf{x}_t) = \frac{1}{2}\left\|x_0 - \bar{x}_0\right\|^2_{P_0^{-1}} + \frac{1}{2}\sum_{k=0}^{t-1}\left\|x_{k+1} - (Ax_k + Bu_k)\right\|^2_{Q^{-1}} + \frac{1}{2}\sum_{k=0}^{t}\left\|y_k - Cx_k\right\|^2_{R^{-1}}. \tag{7.5}$$

7.1.1 Derivation of Kalman Filter

Solving (7.5) can be time consuming, and the size of optimization problem grows as more measurements are available, i.e., t increase. Fortunately, (7.5) can be solved recursively. We first introduce the following lemma.

Lemma 7.1.1. *Let* $f_1(x) = \frac{1}{2}(x-a)^T A(x-a)$ *and* $f_2(x) = \frac{1}{2}(Cx-b)^T B(Cx-b)$, *then* $f(x) = f_1(x) + f_2(x) = \frac{1}{2}(x-v)^T H^{-1}(x-v) + \frac{1}{2}d$, *where*

$$\begin{aligned} H &= A^{-1} - A^{-1}C^T(CA^{-1}C^T + B^{-1})^{-1}CA^{-1} \\ v &= a + A^{-1}C^T(CA^{-1}C^T + B^{-1})^{-1}(b - Ca) \\ d &= (b-Ca)^T(CA^{-1}C^T + B^{-1})^{-1}(b-Ca) \end{aligned}$$

Next will solve (7.5) recursively. When $t = 0$, we have u_0 and y_0, and the goal is to find x_0 by solving

$$\min_{x_0} \quad V_0^+(x_0) = \underbrace{\frac{1}{2}\left\|x_0 - \bar{x}_0\right\|^2_{P_0^{-1}}}_{\text{prior}} + \underbrace{\frac{1}{2}\left\|y_0 - Cx_0\right\|^2_{R^{-1}}}_{\text{measurement}}. \tag{7.6}$$

where

$$V_0^+(x_0) = \frac{1}{2}\left\|x_0 - \bar{x}_0\right\|^2_{P_0^{-1}} - \frac{1}{2}\left\|(y_0 - C\bar{x}_0) - C(x_0 - \bar{x}_0)\right\|^2_{R^{-1}}$$

Using Lemma 7.1.1, we have

$$V_0^+(x_0) = \frac{1}{2}(x_0 - \bar{x}_0 - v)^T H^{-1}(x_0 - \bar{x}_0 - v) + \frac{1}{2}d$$

where

$$\begin{aligned} H &= P_0 - P_0C^T(CP_0C^T + R)^{-1}CP_0 \\ v &= P_0C^T(CP_0C^T + R)^{-1}(y_0 - C\bar{x}_0) \\ d_0 &= (y_0 - C\bar{x}_0)^T(CP_0C^T + R)^{-1}(y_0 - C\bar{x}_0) = \left\|y_0 - C\bar{x}_0\right\|^2_{(CP_0C^T+R)^{-1}} \end{aligned}$$

If we define

$$\begin{aligned} L_0 =& P_0C^T(CP_0C^T+R)^{-1} \\ P_0^+ =& H = P_0 - P_0C^T(CP_0C^T+R)^{-1}CP_0 = P_0 - L_0CP_0 = (I-L_0C)P_0 \end{aligned}$$

then we have

$$\hat{x}_0^+ = \arg\min_{x_0} V_0^+(x_0) = \bar{x}_0 + v = \bar{x}_0 + L_0(y_0 - C\bar{x}_0) \tag{7.7}$$

is a minimizer of (7.6), and the value function (7.6) can be expressed as

$$V_0^+(x_0) = \frac{1}{2}\left\|x_0 - \hat{x}_0^+\right\|^2_{(P_0^+)^{-1}} + \frac{1}{2}d_0 \tag{7.8}$$

When $t=1$, but before y_1 is available, then we have u_0 and y_0, and the goal is to find x_0 and x_1 by solving

$$\min_{x_0,x_1} \quad V_1(x_0,x_1) = \underbrace{\frac{1}{2}\|x_0-\bar{x}_0\|^2_{P_0^{-1}} + \frac{1}{2}\|y_0 - Cx_0\|^2_{R^{-1}}}_{V_0^+(x_0)} + \frac{1}{2}\|x_1 - (Ax_0+Bu_0)\|^2_{Q^{-1}} \tag{7.9}$$

where

$$\begin{aligned} V_1(x_0,x_1) =& V_0^+(x_0) + \frac{1}{2}\|x_1-(Ax_0+Bu_0)\|^2_{Q^{-1}} \\ =& \frac{1}{2}\left\|x_0-\hat{x}_0^+\right\|^2_{(P_0^+)^{-1}} + \frac{1}{2}\|x_1-(Ax_0+Bu_0)\|^2_{Q^{-1}} + \frac{1}{2}d_0 \end{aligned}$$

Again, with a little abuse of notation, this can be expressed as

$$V_1(x_0,x_1) = \frac{1}{2}\|x_0 - v\|^2_{H^{-1}} + \frac{1}{2}d_0$$

where

$$\begin{aligned} H &= P_0^+ - P_0^+A^T(AP_0^+A^T+Q)^{-1}AP_0^+ \\ v &= \hat{x}_0^+ + P_0^+C^T(AP_0^+A^T+Q)^{-1}(x_1 - Bu_0 - A\hat{x}_0^+) \\ d &= (x_1-Bu_0-A\hat{x}_0^+)^T(AP_0^+A^T+Q)^{-1}(x_1-Bu_0-A\hat{x}_0^+) + d_0 \end{aligned}$$

By inspection, it is obvious that the solution to (7.9) is $x_0 = \hat{x}_0^+$ and $x_1 = A\hat{x}_0 + Bu_0 =: \hat{x}_1$. Now, we wish to express the value function $V_1(x_0,x_1)$ as a function of x_1 only. Therefore, we define

$$V_1(x_1) = \min_{x_1} \quad V_1(x_0,x_1) = \frac{1}{2}d = \frac{1}{2}\|x_1-\hat{x}_1\|^2_{P_1^{-1}} + \frac{1}{2}d_0$$

where

$$P_1^{-1} = AP_0^+A^T + Q.$$

Now we are ready to deal with the case that $t=1$ and y_1 is available. In this case, we have u_0, y_0, y_1, and the goal is to find x_0 and x_1 by solving

$$\min_{x_0,x_1} \quad \frac{1}{2}\|x_0-\bar{x}_0\|^2_{P_0^{-1}} + \frac{1}{2}\|y_0-Cx_0\|^2_{R^{-1}} + \frac{1}{2}\|x_1-(Ax_0+Bu_0)\|^2_{Q^{-1}} + \frac{1}{2}\|y_1-Cx_1\|^2_{R^{-1}} \tag{7.10}$$

(7.10) is equivalent to

$$\begin{aligned}
&\min_{x_1} \quad \min_{x_0} \left\{ \frac{1}{2}\|x_0 - \bar{x}_0\|^2_{P_0^{-1}} + \frac{1}{2}\|y_0 - Cx_0\|^2_{R^{-1}} + \frac{1}{2}\|x_1 - (Ax_0 + Bu_0)\|^2_{Q^{-1}} \right\} + \frac{1}{2}\|y_1 - Cx_1\|^2_{R^{-1}} \\
=&\min_{x_1} \quad \min_{x_0} V_1(x_0, x_1) + \frac{1}{2}\|y_1 - Cx_1\|^2_{R^{-1}} = \min_{x_1} \quad \underbrace{V_1(x_1)}_{\text{prior}} + \underbrace{\frac{1}{2}\|y_1 - Cx_1\|^2_{R^{-1}}}_{\text{measurement}} \\
=&\min_{x_1} \quad \underbrace{\frac{1}{2}\|y_1 - Cx_1\|^2_{R^{-1}} + \frac{1}{2}\|x_1 - \hat{x}_1\|^2_{P_1^{-1}} + \frac{1}{2}d_0}_{V_1^+(x_1)}
\end{aligned}$$

Similarly, we have

$$V_1^+(x_1) = \frac{1}{2}\|x_1 - v\|^2_{H^{-1}} + \frac{1}{2}d_1$$

where

$$\begin{aligned}
H &= P_1 - P_1C^T(CP_1C^T + R)^{-1}CP_1 \\
v &= \hat{x}_1 + P_1C^T(CP_1C^T + R)^{-1}(y_1 - C\hat{x}_1) \\
d_1 &= (y_1 - C\hat{x}_1)^T(CP_1C^T + R)^{-1}(y_1 - C\hat{x}_1)
\end{aligned}$$

Define

$$\begin{aligned}
L_1 =& P_1C^T(CP_1C^T + R)^{-1} \\
P_1^+ =& (I - -L_1C)P_1
\end{aligned}$$

and we have

$$\hat{x}_1^+ = \arg\min_{x_1} V_1^+(x_1) = v = \hat{x}_1 + L_1(y_1 - C\hat{x}_1) \tag{7.11}$$

is a minimizer of (7.10), and the value function of (7.10) can be expressed as

$$V_1^+(x_1) = \frac{1}{2}\left\|x_1 - \hat{x}_1^+\right\|^2_{(P_1^+)^{-1}} + \frac{1}{2}d_1 \tag{7.12}$$

In fact, one can generalize the above analysis to arbitrary time $t = k$. The Kalman filter can be summarized as follows. Initialize the filter with $x_0 = \bar{x}_0$ and P_0. Then at time step k, perform the following two step updates.

- **Process update or time update**:

Predicted state estimate	$\hat{x}_{k+1} = A\hat{x}_k^+ + Bu_k$	(7.13a)
Predicted covariance estimate	$P_{k+1} = AP_k^+A^T + Q$	(7.13b)

- **Measurements update**: with the arrival of new measurement y_{k+1}

Optimal Kalman gain	$L_{k+1} = P_{k+1}C^T(CP_{k+1}C^T + R)^{-1}$	(7.13c)
Corrected state estimate	$\hat{x}_{k+1}^+ = \hat{x}_{k+1} + L_{k+1}(y_{k+1} - C\hat{x}_{k+1})$	(7.13d)
Corrected covariance estimate	$P_{k+1}^+ = (I - L_{k+1}C)P_{k+1}$	(7.13e)
Cumulative value function	$d_{k+1} = d_k + \|y_{k+1} - C\hat{x}_k\|^2_{(CP_{k+1}C^T+R)^{-1}}$	(7.13f)

where calculation of (7.13f) is completely optional. Note that the process update is also called "prediction" step, as it predicts the state's mean and covariance for next time step, given the system dynamical model (7.3). The measurements update is also called "correction" step, as it uses the latest measurement y_k to correct the previous prediction $\hat{x}_k$ using gain L_k and prediction error $y_k - C\hat{x}_k$.

Remark 7.1.1. *Kalman filter computes the optimal Kalman gain L_k at each time step, which can be equivalently expressed as*

$$L_k = P_k C^T (CP_k C^T + R)^{-1} = (AP_{k-1}^+ A^T + Q)(CP_k C^T + R)^{-1}$$

When R is large, the Kalman gain will be small, resulting in little measurement update in (7.13d) since the measurement is unreliable. On the other hand, when Q is large, the model (7.3) is unreliable. Therefore Kalman gain will be large, resulting in more measurement update in (7.13d). This aligns with our earlier discussion.

Remark 7.1.2. *P is also called covariance matrix, which indicate the covariance of the state estimate. In other words, at each time step, Kalman filter estimate the state to be a Gaussian distribution with mean $\hat{x}$ (or $\hat{x}^k$) and covariance P (or P^+). From this perspective, Kalman filter can be treated as a special case of Bayesian filtering, and the above recursion (7.13) can be derived by maximizing the conditional probability at each time step, given the input $\boldsymbol{u}_t = \begin{bmatrix} u_0 & u_1 & \cdots u_t \end{bmatrix}^T$ and output $\boldsymbol{y}_t = \begin{bmatrix} y_0 & y_1 & \cdots y_t \end{bmatrix}^T$.*

Remark 7.1.3. *Previous derivation assumes that the covariance Q and R do not change over time. In reality, sometimes it may be beneficial to consider those covariance to be time-varying. In this case, one can simply add subscript $_k$ to (7.13).*

Example 7.1.1 (Kalman Filter)**.** *Consider a simple model*

$$\begin{aligned} x_{k+1} &= x_k + u_k + w_k \\ y_k &= x_k + v_k \end{aligned}$$

where both w_k and v_k have standard normal distribution. Let $u_k = 0.5$ and $x_0 = 100$. Furthermore, initialize the Kalman filter with $\bar{x}_0 = 0$ and $P_0 = 100$. Then Kalman filter estimation results are plotted in Figs. 7.2, 7.3, 7.4, and 7.5. Note that though w_k and v_k have standard normal distribution, in reality we do not have such information. In other words, when we tune Kalman filter, we do not know the exact Q and R. Hence they become tunning parameters. From Fig. 7.2, it can be seen that Kalman filter can effectively converge to the true state. Furthermore, Fig. 7.3 plots the evolution of the variance of the state estimation with respect to time. As can be seen, the variation is large initially, and converges to a constant value as more measurements arrive. If we increase R a lot, making Kalman filter rely on the model more, then the estimation results will soly rely on model update without much measurement update, as can be seen from Fig. 7.4. If we increase Q a lot instead, then Kalman filter will rely more on measurements. As can be seen from Fig. 7.5, the estimation converges quicker compared to Fig. 7.2.

Example 7.1.2 (Kalman Filter for Vehicle Dynamics)**.** *Consider the simplified vehicle dynamic model discussed in Example 3.1.2. With $l = 3$, $\bar{v} = 10$ and $T_s = 0.2$, the DT LTI model is given as follows.*

$$x_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0.2 & 0 \\ 0 & 0 \\ 0 & \frac{2}{3} \end{bmatrix} u_k = Ax_k + Bu_k$$

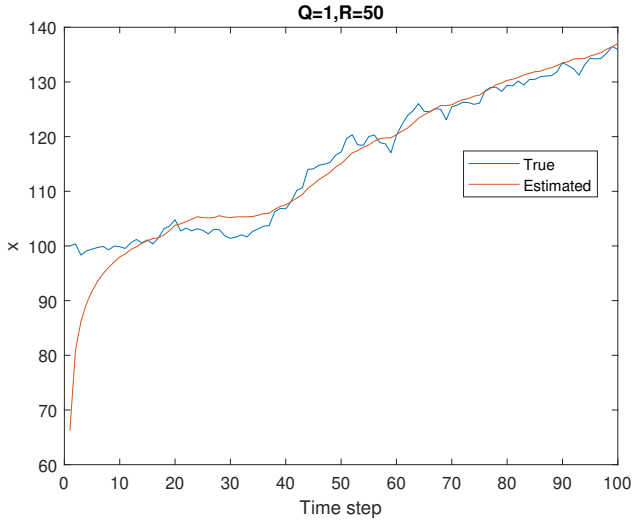

Figure 7.2: Kalman filter state estimation performance for Example 7.1.1.

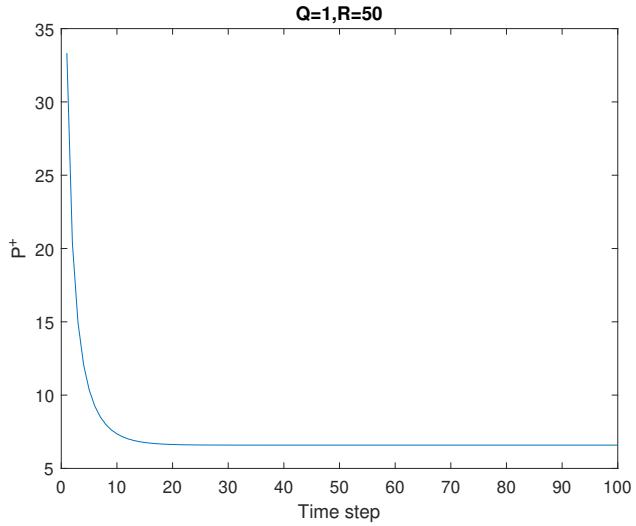

Figure 7.3: Variance with respect to time for Example 7.1.1.

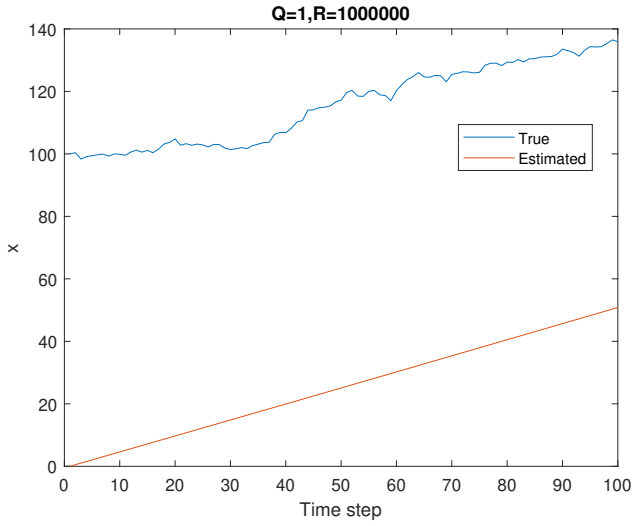

Figure 7.4: Kalman filter state estimation performance for Example 7.1.1. Large R almost disables measurement update, making the estimation mostly relying on model.

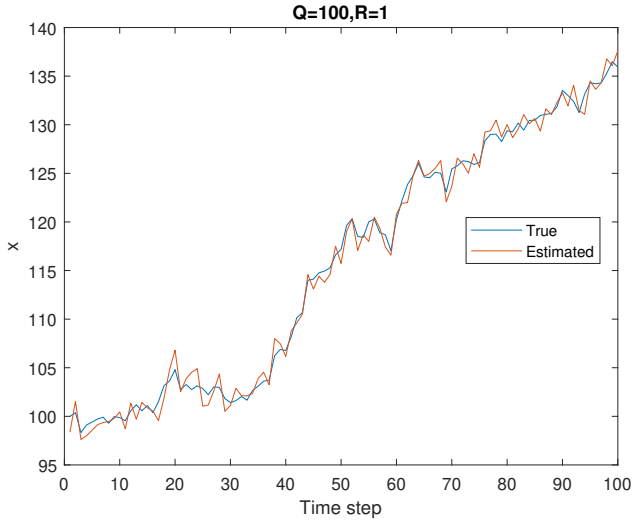

Figure 7.5: Kalman filter state estimation performance for Example 7.1.1. Large Q put more emphasis on measurements, and the estimation converges quicker.

Now suppose only x_1 and x_2 are measured. Recall that x_1 is longitudinal speed and x_2 is the lateral position. Then the output equation is given by

$$y_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} x_k = Cx_k$$

Now suppose the initial conditions are specified as $v_0 = 0$, $p_{y,0} = -2$, and $\phi_0 = -0.2$. And the input u_k is such that the acceleration is constant and the steering has a sinusoidal shape. See Matlab Code Listing 7.2 for more details, where $P_0 = I$. The estimation results are plotted in Fig. 7.6, where all three states can be accurately estimated, even though the yaw angle is not directly measured. Furthermore, the covariance matrix P for each iteration is given below.

$$P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P_1 = \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 5.5 & 2 \\ 0 & 2 & 2 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} 1.6 & 0 & 0 \\ 0 & 8.6154 & 3.0769 \\ 0 & 3.0769 & 2.3846 \end{bmatrix}$$

$$P_3 = \begin{bmatrix} 1.6176 & 0 & 0 \\ 0 & 8.7913 & 3.1277 \\ 0 & 3.1277 & 2.4043 \end{bmatrix}$$

$$P_4 = \begin{bmatrix} 1.618 & 0 & 0 \\ 0 & 8.7963 & 3.1298 \\ 0 & 3.1298 & 2.4052 \end{bmatrix}$$

$$P_5 = \begin{bmatrix} 1.618 & 0 & 0 \\ 0 & 8.7969 & 3.13 \\ 0 & 3.13 & 2.4053 \end{bmatrix}$$

$$P_6 = \begin{bmatrix} 1.618 & 0 & 0 \\ 0 & 8.7969 & 3.13 \\ 0 & 3.13 & 2.4053 \end{bmatrix}$$

$$\vdots$$

As can be seen, not only the state estimation $\hat{x}$ converges, but also its covariance matrix converges.

7.1.2 Stationary Kalman Filter

Kalman filter as presented in (7.13) is effective, but at the same time can be computationally heavy, since it requires several matrix multiplication and matrix inversion in real-time. In practice, a simplified version, called "stationary Kalman filter", or "steady-state Kalman filter", can be used to reduce computational burdern. The steady-state Kalman filter uses the fact that the covariance P_k and P_k^+ converges to certain value (see Fig. 7.3), and so does the Kalman gain L_k. Therefore, instead of computing gain L_k in real-time, we can compute $L = \lim_{k\to\infty} L_k$, and use L for measurement updates. Let $P = \lim_{k\to\infty} P_k$ and $P^+ = \lim_{k\to\infty} P_k^+$. Then according to (7.13)

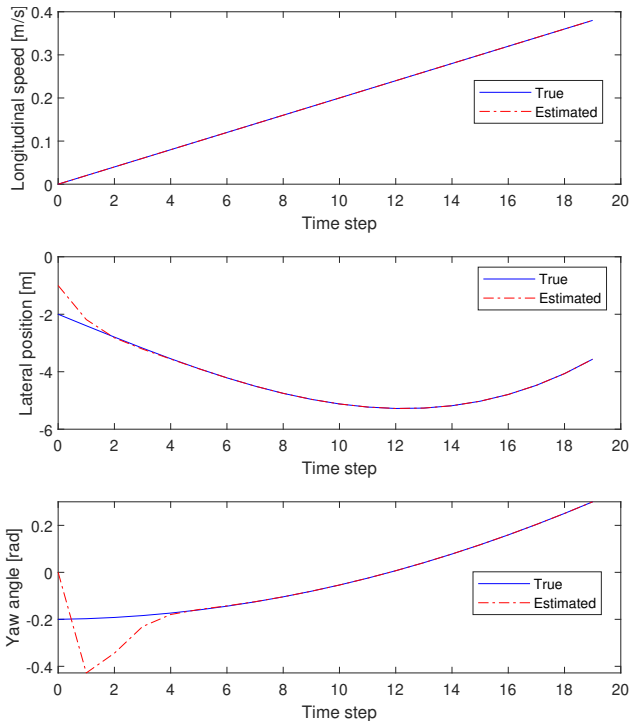

Figure 7.6: Kalman filter state estimation performance for Example 7.1.2.

we have

$$\begin{aligned} P =& AP^+A^T + Q \\ P^+ =& (I - LC)P \\ L =& PC^T(CPC^T + R)^{-1} \end{aligned}$$

Therefore

$$P = A(I - PC^T(CPC^T + R)^{-1}C)PA^T + Q = APA^T - APC^T(CPC^T + R)^{-1}PA^T + Q \quad (7.14)$$

which is a algebraic Riccatti equation. Note the similarity between (4.7). Once the stationary covariance P is found by solving (7.14), the steady-state gain is given by

$$L = PC^T(CPC^T + R)^{-1}$$

Note that these calculations are done offline. Then in real-time, the steady state Kalman filter updates the estimation as follows.

- **Process update or time update**:

$$\text{Predicted state estimate} \qquad \hat{x}_{k+1} = A\hat{x}_k^+ + Bu_k \quad (7.15a)$$

- **Measurements update**: with the arrival of new measurement y_{k+1}

$$\text{Corrected state estimate} \qquad \hat{x}_{k+1}^+ = \hat{x}_{k+1} + L(y_{k+1} - C\hat{x}_{k+1}) \quad (7.15b)$$

Note that compare to (7.13), steady-state Kalman filter (7.15) saves several online calculations, with a price that only mean of the state estimate is available, without covariance matrix.

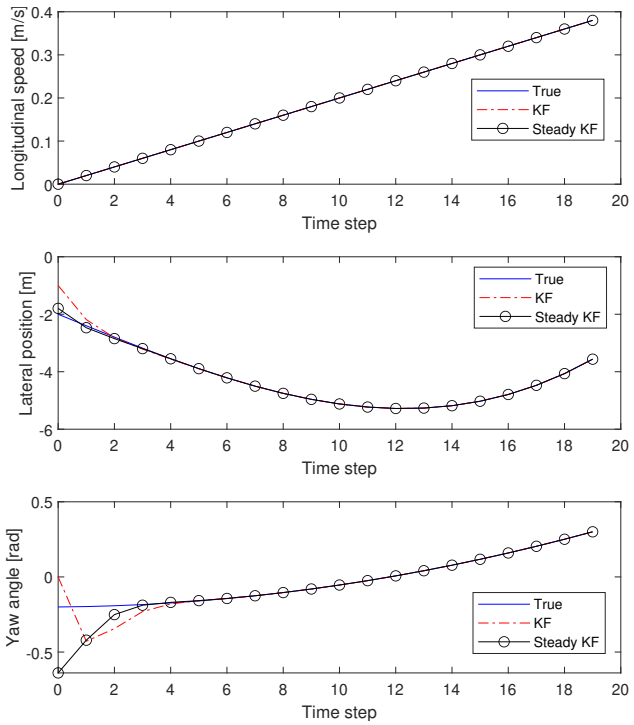

Figure 7.7: Steady-state Kalman filter state estimation performance for Example 7.1.3.

Example 7.1.3 (Steady-state Kalman Filter for Vehicle Dynamics)**.** *Consider state estiamtion problem in Example 7.1.2, with the same initial conditions and input u_k. Instead of calculating optimal gain L_k in real time, we first solve the algrebraic Riccatti equation and obtain*

$$P = \begin{bmatrix} 1.618 & 0 & 0 \\ 0 & 8.7969 & 3.13 \\ 0 & 3.13 & 2.4053 \end{bmatrix}, \qquad L = \begin{bmatrix} 0.618 & 0 \\ 0 & 0.8979 \\ 0 & 0.3195 \end{bmatrix}$$

Then apply (7.15) and get the results as shown in The estimation results are plotted in Fig. 7.7. Comparing the steady-state Kalman filter and standard Kalman filter, we can see that the estimation performance loss is completely acceptable.

7.2 Extended Kalman Filter

When the system is nonlinear, then Kalman filter, which assumes the system has linear dynamic of (7.3), is no longer applicable. When the systems has nonlinear dynamics, such as

$$x_{k+1} = f(x_k, u_k) + w_k \tag{7.16a}$$
$$y_k = g(x_k) + v_k \tag{7.16b}$$

Similar to the case of linear Kalman filter, w_k is the process noise satisfying Gaussian distribution with mean 0 and covariance Q and v_k is the measurement noise satisfying Gaussian distribution with mean 0 and covariance R.

Extended Kalman filter (EKF) estimates the state recursively, with extra step to compute a linearized model for predicting and correcting the covariance matrix. At time step k, given current state estimation is $\hat{x}_k^+$ with covariance matrix P_k^+, perform the following

- **Process update or time update**:

Predicted state estimate	$\hat{x}_{k+1} = f(\hat{x}_k^+, u_k)$	(7.17a)
Linearization	$A_k = \left.\dfrac{\partial f(x,u)}{\partial x}\right\|_{x=\hat{x}_k^+=,u=u_k}$	(7.17b)
Predicted covariance estimate	$P_{k+1} = A_k P_k^+ A_k^T + Q$	(7.17c)

- **Measurements update**: with the arrival of new measurement y_{k+1}

Linearization	$C_{k+1} = \left.\dfrac{\partial g(x)}{\partial x}\right\|_{x=\hat{x}_{k+1}}$	(7.17d)
Near-optimal Kalman gain	$L_{k+1} = P_{k+1}C_{k+1}^T(C_{k+1}P_{k+1}C_{k+1}^T + R)^{-1}$	(7.17e)
Corrected state estimate	$\hat{x}_{k+1}^+ = \hat{x}_{k+1} + L_{k+1}(y_{k+1} - g(\hat{x}_{k+1}))$	(7.17f)
Corrected covariance estimate	$P_{k+1}^+ = (I - L_{k+1}C_{k+1})P_{k+1}$	(7.17g)

Remark 7.2.1. *Note that EKF is not an optimal estimator. Furthermore, in the case of bad initial estimate or poor model, EKF can quickly diverge. However, EKF can give reasonable performance in many applications, and remains as one of the popular state estimation methods for nonlinear systems.*

Example 7.2.1 (EKF for State Estimation)**.** *Consider the state estimation problem similar to the one in Example 7.1.2, but instead let's consider the original nonlinear dynamic. As discussed in Example 2.2.5, a simplified lateral vehicle dynamics model has the following dynamics*

$$\dot{v} = a \tag{7.18a}$$
$$\dot{p}_y = v\sin\phi \tag{7.18b}$$
$$\dot{\phi} = \frac{v}{l}\tan\zeta, \tag{7.18c}$$

where v is the vehicle speed, p_y is the lateral position of the vehicle, and ϕ is the yaw angle of the vehicle, all in global coordinate. l is the wheelbase, a and ζ are the two control inputs, namely vehicle acceleration and steering angle. Denote $x = \begin{bmatrix} v & p_y & \phi \end{bmatrix}^T$, $y = x$, and $u = \begin{bmatrix} a & \zeta \end{bmatrix}^T$. Given a sampling time T_s, one can discretize the continuous time dynamics into discrete time dynamics using Forward Euler, as follows

$$x_{k+1}(1) = x_k(1) + T_s u_k(1) \tag{7.19a}$$
$$x_{k+1}(2) = x_k(2) + T_s x_k(1)\sin x_k(3) \tag{7.19b}$$
$$x_{k+1}(3) = x_k(3) + T_s\frac{x_k(1)}{l}\tan u_k(2). \tag{7.19c}$$

Similar to Example 7.1.2, only the first two states are measure, i.e.,

$$y_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} x_k = Cx_k.$$

To implement EKF, one needs the following two linearizations, given $\hat{x}_k^+$, u_k and $\hat{x}_{k+1}$,

$$A_k = \begin{bmatrix} 1 & 0 & 0 \\ T_s\sin\hat{x}_k^+(3) & 1 & T_s\hat{x}_k^+(1)\cos\hat{x}_k(3) \\ T_s\frac{1}{l}\tan u_k(2) & 0 & 1 \end{bmatrix}$$

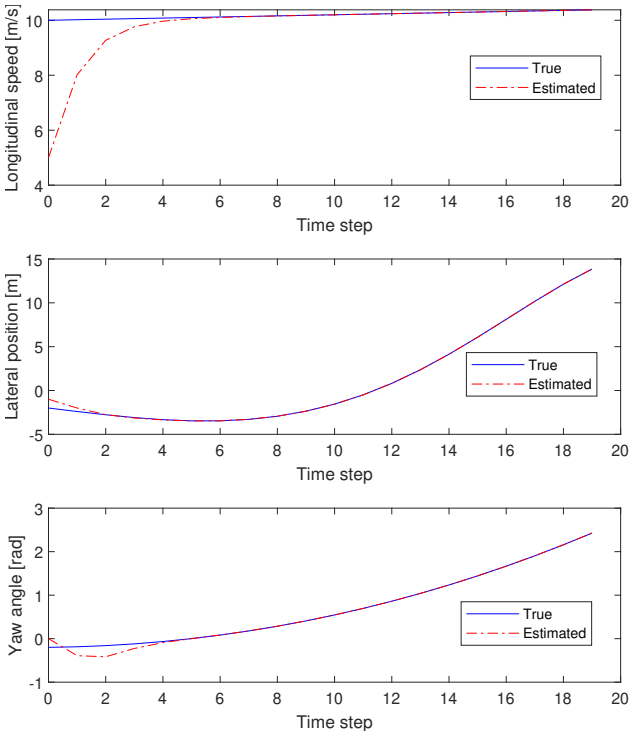

Figure 7.8: EKF state estimation performance for Example 7.2.1, with reasonable initial estimate.

$$C_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The EKF state estimation performance, with $\hat{x}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$*, is plotted in Fig. 7.8, where estimation for all three states converged even if the system dynamics (7.19) are nonlinear. Now if we change the initial estimate to* $\hat{x}_0 = \begin{bmatrix} 0 & 0 & -2 \end{bmatrix}^T$*, the EKF performance is plotted in Fig. 7.9, where the state estimate diverges as EKF is only local optimal.*

Example 7.2.2 (EKF for Parameter Estimation)**.** *Consider the state estimation problem discussed in Example 7.2.1, but instead suppose that the wheelbase* l *is unknown. Then we can use EKF to simultaneously estimate state* x *and parameter* l*. The trick we apply is to treat* l *as an augmented state, whose derivative is 0. In other words, define*

$$\tilde{x} = \begin{bmatrix} x \\ l \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ l \end{bmatrix} = \begin{bmatrix} v \\ p_y \\ \phi \\ l \end{bmatrix}$$

Then the nonlinear system dynamics (7.19) becomes

$$\begin{aligned} \tilde{x}_{k+1}(1) &= \tilde{x}_k(1) + T_s u_k(1) \\ \tilde{x}_{k+1}(2) &= \tilde{x}_k(2) + T_s \tilde{x}_k(1) \sin \tilde{x}_k(3) \\ \tilde{x}_{k+1}(3) &= \tilde{x}_k(3) + T_s \frac{\tilde{x}_k(1)}{\tilde{x}_k(4)} \tan u_k(2) \\ \tilde{x}_{k+1}(4) &= \tilde{x}_k(4). \end{aligned}$$

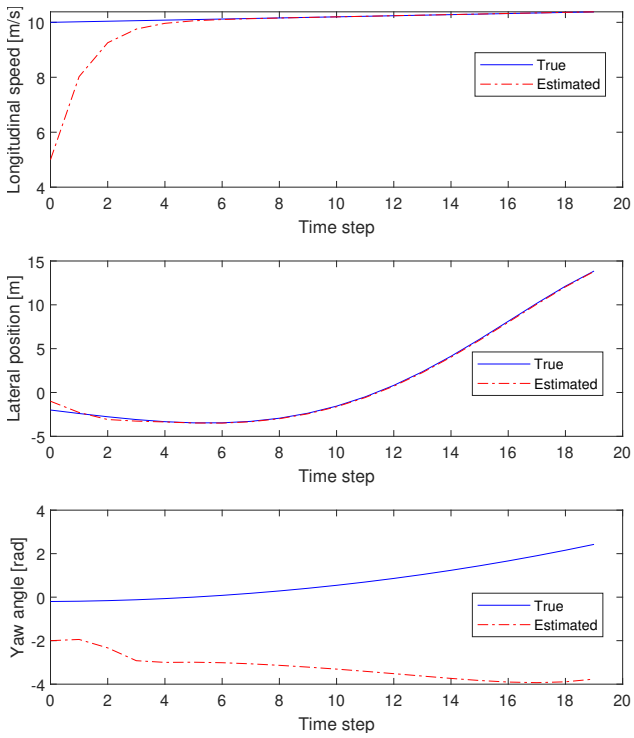

Figure 7.9: EKF state estimation performance for Example 7.2.1, with poor initial estimate.

To implement EKF, one needs the following two linearizations, given $\hat{x}_k^+$, u_k *and* $\hat{x}_{k+1}$,

$$A_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ T_s \sin \hat{x}_k^+(3) & 1 & T_s \hat{x}_k^+(1) \cos \hat{x}_k(3) & 0 \\ T_s \frac{1}{\hat{x}_k^+(4)} \tan u_k(2) & 0 & 1 & -T_s \frac{\hat{x}_k^+(1)}{(\hat{x}_k^+(4))^2} \tan u_k(2) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The EKF state estimation performance is plotted in Fig. 7.10, where estimation for all three states and parameter l *converged to their true value within reasonable amount of time step.*

7.3 Moving Horizon Estimation

Recall the discussion in Section 7.1, given input sequence $\mathbf{u}_t = \begin{bmatrix} u_0 & u_1 & \cdots u_t \end{bmatrix}^T$ and output sequence $\mathbf{y}_t = \begin{bmatrix} y_0 & y_1 & \cdots y_t \end{bmatrix}^T$, the state estimate $\mathbf{x}_t = \begin{bmatrix} x_0 & x_1 & \cdots & x_t \end{bmatrix}^T$ can be found by solving the optimization problem (7.2), which is copied below.

$$\min_{\mathbf{x}_t} \quad V_t(\mathbf{x}_t) = \frac{1}{2} \|x_0 - \bar{x}_0\|^2_{P_0^{-1}} + \frac{1}{2} \sum_{k=0}^{t-1} \|x_{k+1} - f(x_k, u_k)\|^2_{Q^{-1}} + \frac{1}{2} \sum_{k=0}^{t} \|y_k - g(x_k)\|^2_{R^{-1}}. \quad (7.20)$$

When the system dynamic is linear, (7.2) reduces to a linear least square problem (7.5), as follows

$$\min_{\mathbf{x}_t} \quad V_t(\mathbf{x}_t) = \frac{1}{2} \|x_0 - \bar{x}_0\|^2_{P_0^{-1}} + \frac{1}{2} \sum_{k=0}^{t-1} \|x_{k+1} - (Ax_k + Bu_k)\|^2_{Q^{-1}} + \frac{1}{2} \sum_{k=0}^{t} \|y_k - Cx_k\|^2_{R^{-1}}.$$

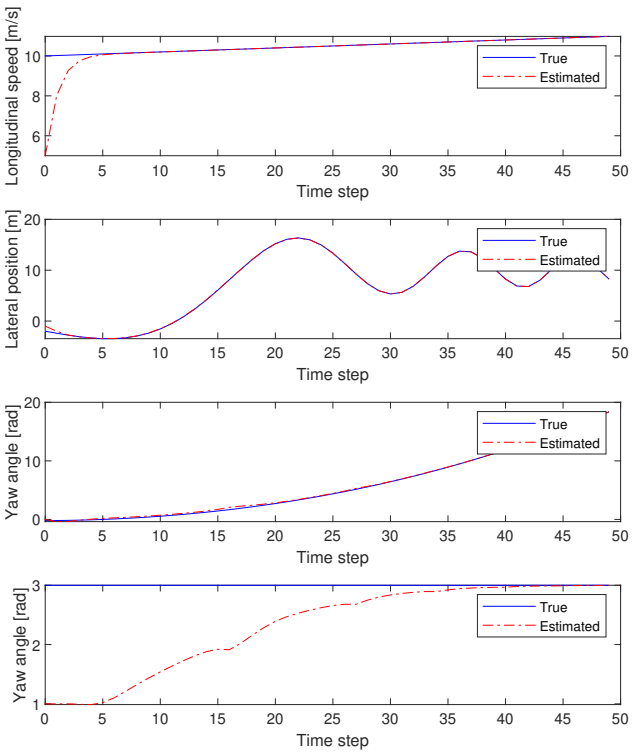

Figure 7.10: EKF state estimation performance for Example 7.2.2.

whose recursive solution is given by (7.13). When the system dynamic is nonlinear, the EKF tries to modify (7.13) to obtain a solution to (7.20), which is effective but not optimal.

Another direction it to directly solve (7.20), which can be computationally heavy with complexity grows with respect to time. To alleviate this, moving horizon estimation (MHE) technique aims to solve a reduced version of (7.20). Given a estimation window p, define

$$\mathbf{x}_t^p = \begin{bmatrix} x_{t-p+1} \\ x_{t-p+2} \\ \vdots \\ x_{t-1} \\ x_t \end{bmatrix} \tag{7.21}$$

and solve the following reduced order optimization problem

$$\min_{\mathbf{x}_t^p} V_t(\mathbf{x}_t^p) = \frac{1}{2}\left\|x_{t-p+1} - \hat{x}_{t-p+1}\right\|_{Q_{x_0}}^2 + \frac{1}{2}\sum_{k=t-p+1}^{t-1}\left\|x_{k+1} - f(x_k, u_k)\right\|_{Q_x}^2 + \frac{1}{2}\sum_{k=t-p+1}^{t}\left\|y_k - g(x_k)\right\|_{Q_y}^2. \tag{7.22}$$

The idea of MHE is illustrated in Fig. 7.11, where instead of using all available data to find the estimate of entire state sequence, MHE uses only a window of p data to find the estimate of a truncated state sequence[1]. When the time advances with new measure, the estimation window move one time step as well, resulting in a new optimization problem with the same dimension. It is obvious that MHE as presented in 7.22 is not optimal.

[1]It is then obvious that during initialization when less than p data are available, then MHE would use all available data to estimate the entire state sequence, which is shorter than p.

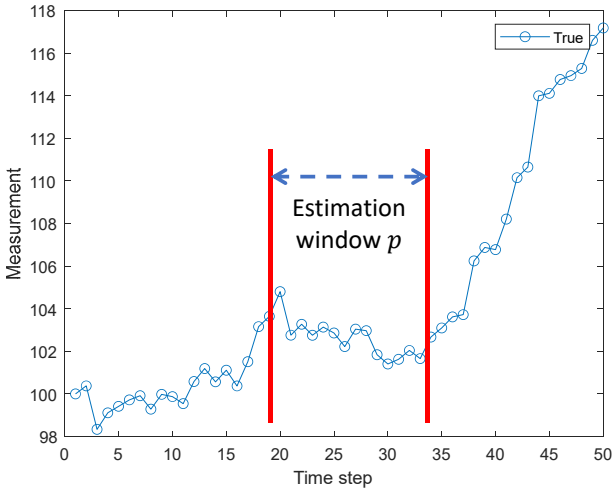

Figure 7.11: Illustration of moving horizon estimation.

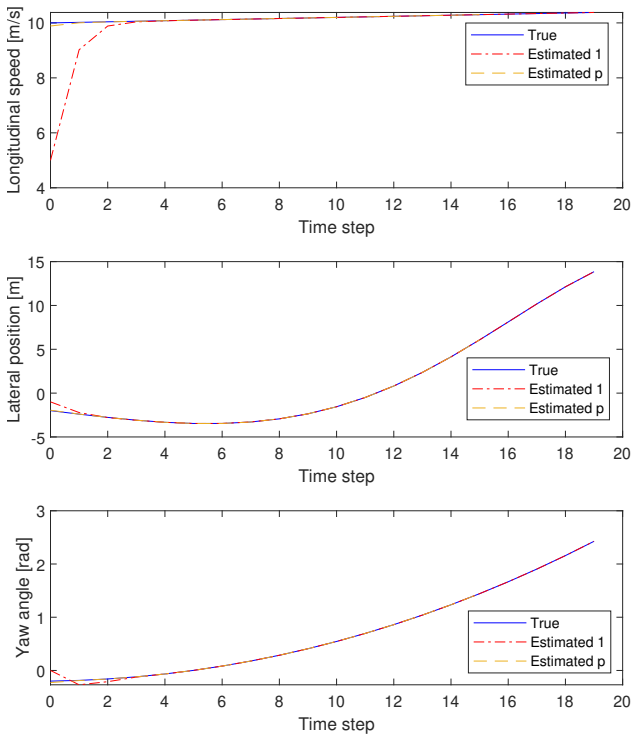

Figure 7.12: MHE state estimation performance for Example 7.3.1, with reasonable initial estimate.

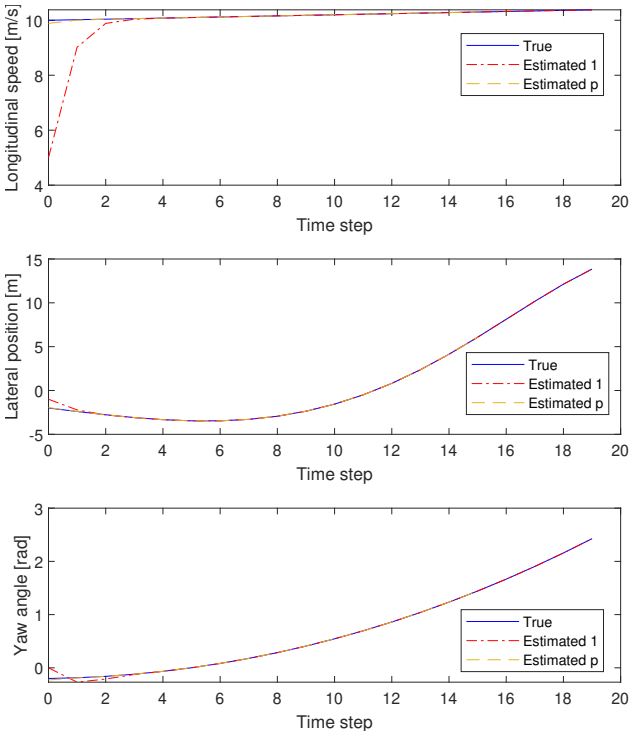

Figure 7.13: MHE state estimation performance for Example 7.3.1, with poor initial estimate.

Example 7.3.1 (MHE for State Estimation)**.** *Consider the state estimation problem as discussed in Example 7.2.1, but instead of EKF let's use MHE.*

The EKF state estimation performance, with $\hat{x}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$*, is plotted in Fig. 7.12, where estimation for all three states converged even if the system dynamics (7.19) are nonlinear. Note that the legend "Estimate 1" refers to the estimation for* x_k *when* y_k *is first available, and "Estimate p" refers to the estimation for* x_k *when it has been estimated* p *times, which actually occurs at time step* $k + p$*. Comparing Fig. 7.12 to Fig. 7.8, the performance of EKF and MHE are not much different.*

Now if we change the initial estimate to $\hat{x}_0 = \begin{bmatrix} 0 & 0 & -2 \end{bmatrix}^T$*, the MHE performance is plotted in Fig. 7.13, where the state estimate can still converge to its true value. Comparing Fig. 7.13 to Fig. 7.9, the robustness of MHE over EKF is rather obvious. However, such advantage comes with a price that at each time step, MHE needs to solve a nonlinear optimization problem (7.22), as illustrated in Matlab Code Listing 7.6.*

7.4 Matlab Codes

```
close all; clear variables; clc

A = 1; B = 1; C = 1;

N = 100; sig_1 = 1; sig_2 = 1;

x0 = 100; u=0.5; y = zeros(N,1); x = zeros(N,1); x(1) = x0;

rng(1); v = randn(N,1); rng(2); w = randn(N,1);

for n = 1 : N
    y(n) = C*x(n)+v(n);
    if n<N
        x(n+1) = A*x(n)+B*u+w(n);
    end
end

xhat = zeros(N,1); xhatPlus = zeros(N,1);
P = zeros(N,1); PPlus = zeros(N,1);

P0 = 100; xbar = 0; xhat(1) = xbar; P(1) = P0;

Q = 100; R = 1;
for n = 1 : N
    L = P(n)*C'/(C*P(n)*C'+R);
    xhatPlus(n) = xhat(n)+L*(y(n)-C*xhat(n));
    PPlus(n) = (1-L*C)*P(n);

    if n < N
        xhat(n+1) = A*xhatPlus(n)+B*u;
        P(n+1) = A*PPlus(n)*A'+Q;
    end
end

f=figure; plot(0:N-1,x,1:N,xhatPlus)
xlabel('Time step'), ylabel('x'), title(sprintf('Q=%d,R=%d',Q,R))
```

Listing 7.1: MATLAB Commands for Example 7.1.1.

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; Ts = 0.2;
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1];
B = [Ts 0;0 0;0 xbar(1)/l*Ts];
C = [1 0 0; 0 1 0];

N = 20; x0 = [0; -2; -0.2]; u=[0.1*ones(1,N);sin((1:N)*Ts/10)/5];
y = zeros(2,N); x = zeros(3,N); x(:,1) = x0;

for n = 1 : N
    y(:,n) = C*x(:,n);
    if n<N
        x(:,n+1) = A*x(:,n)+B*u(:,n);
    end
end

xhat = zeros(3,N); xhatPlus = zeros(3,N);
xbar = zeros(3,1); xhat(:,1) = xbar;

PPlus = zeros(3,3); P0 = eye(3); P = P0;

Q = eye(3); R = eye(2);
for n = 1 : N
    L = P*C'/(C*P*C'+R);
    xhatPlus(:,n) = xhat(:,n)+L*(y(:,n)-C*xhat(:,n));
    PPlus = (eye(3)-L*C)*P;

    if n < N
        xhat(:,n+1) = A*xhatPlus(:,n)+B*u(:,n);
        P = A*PPlus*A'+Q;
    end
end

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos;

subplot(3,1,1), plot(0:N-1,x(1,:),'blue-',0:N-1,xhatPlus(1,:),'r-.'),
xlabel('Time step'), ylabel('Longitudinal speed [m/s]')

subplot(3,1,2), plot(0:N-1,x(2,:),'blue-',0:N-1,xhatPlus(2,:),'r-.'),
xlabel('Time step'), ylabel('Lateral position [m]')

subplot(3,1,3), plot(0:N-1,x(3,:),'blue-',0:N-1,xhatPlus(3,:),'r-.'),
xlabel('Time step'), ylabel('Yaw angle [rad]')
```

Listing 7.2: MATLAB Commands for Example 7.1.2.

```
close all; clear variables; clc

l = 3; xbar = [10;0;0]; ubar = [0;0]; Ts = 0.2;
A = [1 0 0;0 1 xbar(1)*Ts;0 0 1];
B = [Ts 0;0 0;0 xbar(1)/l*Ts]; C = [1 0 0; 0 1 0];

N = 20; x0 = [0; -2; -0.2]; u=[0.1*ones(1,N);sin((1:N)*Ts/10)/5];
y = zeros(2,N); x = zeros(3,N); x(:,1) = x0;

for n = 1 : N
    y(:,n) = C*x(:,n);
    if n<N
        x(:,n+1) = A*x(:,n)+B*u(:,n);
    end
end

xhat = zeros(3,N); xhatPlus = zeros(3,N); xbar = zeros(3,1);
xhat(:,1) = xbar; PPlus = zeros(3,3); P0 = eye(3); P = P0;
Q = eye(3); R = eye(2);
for n = 1 : N
    L = P*C'/(C*P*C'+R);
    xhatPlus(:,n) = xhat(:,n)+L*(y(:,n)-C*xhat(:,n));
    PPlus = (eye(3)-L*C)*P;
    if n < N
        xhat(:,n+1) = A*xhatPlus(:,n)+B*u(:,n); P = A*PPlus*A'+Q
    end
end

xhats = zeros(3,N); xhatPluss = zeros(3,N); xhats(:,1) = xbar;
P = idare(A',C',Q,R); L = P*C'/(C*P*C'+R);
for n = 1 : N
    xhatPluss(:,n) = xhats(:,n)+L*(y(:,n)-C*xhats(:,n));
    if n < N
        xhats(:,n+1) = A*xhatPluss(:,n)+B*u(:,n);
    end
end

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos; t = 0:N-1;
subplot(3,1,1), plot(t,x(1,:),t,xhatPlus(1,:),t,xhatPluss(1,:)),
xlabel('Time step'), ylabel('Longitudinal speed [m/s]'),
legend('True','KF', 'Steady KF')
subplot(3,1,2), plot(t,x(2,:),t,xhatPlus(2,:),t,xhatPluss(2,:)),
xlabel('Time step'), ylabel('Lateral position [m]'),
legend('True','KF', 'Steady KF')
subplot(3,1,3), plot(t,x(3,:),t,xhatPlus(3,:),t,xhatPluss(3,:)),
xlabel('Time step'), ylabel('Yaw angle [rad]'),
legend('True','KF', 'Steady KF')
```

Listing 7.3: MATLAB Commands for Example 7.1.3.

```
close all; clear variables; clc

l = 3; Ts = 0.2; Ak = eye(3); Ckp1 = [1 0 0; 0 1 0];

N = 20; x0 = [10; -2; -0.2]; u=[0.1*ones(1,N);sin((1:N)*Ts/10)];
y = zeros(2,N); x = zeros(3,N); x(:,1) = x0;
for n = 1 : N
    y(:,n) = x(1:2,n);
    if n<N
        x(:,n+1) = f_dynamic(x(:,n),u(:,n),Ts,l);
    end
end

xhat = zeros(3,N); xhatPlus = zeros(3,N); xbar = zeros(3,1);
xbar(3) = -2; xhat(:,1) = xbar;

PPlus = zeros(3,3); P0 = eye(3); P = P0; Q = 1*eye(3); R = 1*eye(2);
for n = 1 : N
    L = P*Ckp1'/(Ckp1*P*Ckp1'+R);
    xhatPlus(:,n) = xhat(:,n)+L*(y(:,n)-xhat(1:2,n));
    PPlus = (eye(3)-L*Ckp1)*P;
    if n < N
        xhat(:,n+1) = f_dynamic(xhatPlus(:,n),u(:,n),Ts,l);
        Ak(2,1) = Ts*sin(xhatPlus(3,n));
        Ak(2,3) = Ts*xhatPlus(1,n)*cos(xhatPlus(3,n));
        Ak(3,1) = Ts/l*tan(u(2,n));
        P = Ak*PPlus*Ak'+Q;
    end
end

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos; t=0:N-1;
subplot(3,1,1), plot(t,x(1,:),'blue-',t,xhatPlus(1,:),'r-.'),
xlabel('Time step'), ylabel('Longitudinal speed [m/s]'),
legend('True','Estimated')
subplot(3,1,2), plot(t,x(2,:),'blue-',t,xhatPlus(2,:),'r-.'),
xlabel('Time step'), ylabel('Lateral position [m]'),
legend('True','Estimated')
subplot(3,1,3), plot(t,x(3,:),'blue-',t,xhatPlus(3,:),'r-.'),
xlabel('Time step'), ylabel('Yaw angle [rad]'),
legend('True','Estimated')

function xkp1 = f_dynamic(xk,uk,Ts,l)
xkp1 = zeros(size(xk));
xkp1(1) = xk(1)+Ts*uk(1);
xkp1(2) = xk(2)+Ts*xk(1)*sin(xk(3));
xkp1(3) = xk(3) + Ts*xk(1)/l*tan(uk(2));
end
```

Listing 7.4: MATLAB Commands for Example 7.2.1.

```
l = 3; Ts = 0.2; Ak = eye(4); Ckp1 = [1 0 0 0; 0 1 0 0];
N = 50; x0 = [10; -2; -0.2]; u=[0.1*ones(1,N);sin((1:N)*Ts/10)];
y = zeros(2,N); x = zeros(3,N); x(:,1) = x0;
for n = 1 : N
    y(:,n) = x(1:2,n);
    if n<N
        x(:,n+1) = f_dynamic(x(:,n),u(:,n),Ts,l);
    end
end

xhat = zeros(4,N); xhatPlus = zeros(4,N);
xbar = [zeros(3,1); 1]; xhat(:,1) = xbar;
PPlus = zeros(4,4); P0 = eye(4); P = P0; Q = 1*eye(4); R = 1*eye(2);
for n = 1 : N
    L = P*Ckp1'/(Ckp1*P*Ckp1'+R);
    xhatPlus(:,n) = xhat(:,n)+L*(y(:,n)-xhat(1:2,n));
    PPlus = (eye(4)-L*Ckp1)*P;
    if n < N
     xhat(1:3,n+1) = f_dynamic(xhatPlus(1:3,n),u(:,n),Ts,xhatPlus(4,n));
     xhat(4,n+1) = xhatPlus(4,n);
     Ak(2,1) = Ts*sin(xhatPlus(3,n));
     Ak(2,3) = Ts*xhatPlus(1,n)*cos(xhatPlus(3,n));
     Ak(3,1) = Ts/xhatPlus(4,n)*tan(u(2,n));
     Ak(3,4) = -Ts/xhatPlus(4,n)*tan(u(2,n))*xhatPlus(1,n);
     P = Ak*PPlus*Ak'+Q;
    end
end

f=figure; pos = get(f, 'Position');
pos(2)=pos(2)/10; pos(4) = pos(4)*1.5; f.Position = pos; t=0:N-1;
subplot(4,1,1), plot(t,x(1,:),'blue-',t,xhatPlus(1,:),'r-.'),
xlabel('Time step'), ylabel('Longitudinal speed [m/s]'),
legend('True','Estimated')
subplot(4,1,2), plot(t,x(2,:),'blue-',t,xhatPlus(2,:),'r-.'),
xlabel('Time step'), ylabel('Lateral position [m]'),
legend('True','Estimated')
subplot(4,1,3), plot(t,x(3,:),'blue-',t,xhatPlus(3,:),'r-.'),
xlabel('Time step'), ylabel('Yaw angle [rad]'),
legend('True','Estimated')
subplot(4,1,4), plot(t,l*ones(size(t)),'blue-',t,xhatPlus(4,:),'r-.'),
xlabel('Time step'), ylabel('Yaw angle [rad]'),
legend('True','Estimated')

function xkp1 = f_dynamic(xk,uk,Ts,l)
xkp1 = zeros(size(xk)); xkp1(1) = xk(1)+Ts*uk(1);
xkp1(2) = xk(2)+Ts*xk(1)*sin(xk(3));
xkp1(3) = xk(3) + Ts*xk(1)/l*tan(uk(2));
end
```

Listing 7.5: MATLAB Commands for Example 7.2.2.

```
l = 3; Ts = 0.2; p = 5; N = 20; x0 = [10; -2; -0.2]; y = zeros(2,N);
x = zeros(3,N); x(:,1) = x0;u=[0.1*ones(1,N);sin((1:N)*Ts/10)];
for n = 1 : N
    y(:,n) = x(1:2,n);
    if n<N
        x(:,n+1) = f_dynamic(x(:,n),u(:,n),Ts,l);
    end
end

xhat1 = zeros(3,N); xhatp = zeros(3,N); xbar = zeros(3,1);
xseq = xbar; xbar(3) = -2; Qx = 1*eye(3); Qy = 1*eye(2);
for n = 1 : N
    if n <=p & n > 1
        xg = [xseq,xseq(:,end)];
    else
        xg = [xseq(:,2:end),xseq(:,end)];
    end
    xbar = xg(:,1); pn = min(n,p);
    xseq = fmincon(@(x)V(x,xbar,y(:,n-pn+1:n),u(:,n-pn+1:n),...
        Qx,Qx,Qy,Ts,l),xg,[],[]);
    xseq = reshape(xseq,3,pn);
    xhat1(:,n) = xseq(:,end);
    if n>=5
        xhatp(:,n-p+1) = xseq(:,1);
    end
end
f=figure; pos = get(f, 'Position'); pos(2)=pos(2)/10; pos(4) = pos(4)*1.5;
f.Position = pos; t=0:N-1; t1=0:N-p; idx = N-p+1;
subplot(3,1,1), plot(t,x(1,:),'blue-',t,xhat1(1,:),'r-.',t1,xhatp(1,1:idx),'--'),
xlabel('Time step'), ylabel('Longitudinal speed [m/s]'),
legend('True','Estimated 1','Estimated p')
subplot(3,1,2), plot(t,x(2,:),'blue-',t,xhat1(2,:),'r-.',t1,xhatp(2,1:idx),'--'),
xlabel('Time step'), ylabel('Lateral position [m]'),
legend('True','Estimated 1','Estimated p')
subplot(3,1,3), plot(t,x(3,:),'blue-',t,xhat1(3,:),'r-.',t1,xhatp(3,1:idx),'--'),
xlabel('Time step'), ylabel('Yaw angle [rad]'),
legend('True','Estimated 1','Estimated p')
function xkp1 = f_dynamic(xk,uk,Ts,l)
xkp1 = zeros(size(xk)); xkp1(1) = xk(1)+Ts*uk(1);
xkp1(2) = xk(2)+Ts*xk(1)*sin(xk(3)); xkp1(3) = xk(3) + Ts*xk(1)/l*tan(uk(2));
end
function r = V(x,x0,y,u,Q0,Qx,Qy,Ts,l)
p = size(y,2); x = reshape(x,3,p); r = 0;
e = x(:,1)-x0; r = r +  e'*Q0*e;
for n = 1 : p-1
    e = x(:,n+1)-f_dynamic(x(:,n),u(:,n),Ts,l); r = r +  e'*Qx*e;
    e = x(1:2,n)-y(:,n);       r = r +  e'*Qy*e;
end
e = x(1:2,p)-y(:,p); r = r +  e'*Qy*e;
end
```


Listing 7.6: MATLAB Commands for Example 7.3.1.

Appendices

Appendix A

Linear Algebra

A.1 Notations

We use $\mathbb{R}$ to denote the set of real numbers, $\mathbb{R}^n$ to denote the set of real column vectors with n elements, and $\mathbb{R}^{n\times m}$ to denote the set of real matrices with n rows and m columns.

Given a matrix A or a vector x, we use superscript ^{T} to denote the transpose, e.g., A^T or x^T.

A.2 Matrices and Vectors

Matrix-vector multiplication. Given a matrix $A \in \mathbb{R}^{n\times m}$ and a vector $b \in \mathbb{R}^m$, the matrix-vector multiplication results in a vector $c \in \mathbb{R}^n$ such that the kth element of c is given by $\sum_{j=1}^m a_{kj}b_j$. In other words,

$$c = Ab = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^m a_{1j}b_j \\ \sum_{j=1}^m a_{2j}b_j \\ \vdots \\ \sum_{j=1}^m a_{nj}b_j \end{bmatrix} \tag{A.1}$$

One can also express the above multiplication as

$$c = Ab = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = b_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \cdots \\ a_{n1} \end{bmatrix} + b_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \cdots \\ a_{n2} \end{bmatrix} + \cdots + b_3 \begin{bmatrix} a_{1m} \\ a_{2m} \\ \cdots \\ a_{nm} \end{bmatrix} \tag{A.2}$$

Example A.2.1 (Matrix-vector multiplication)**.** *Let $A \in \mathbb{R}^{3\times 3}$ and $b \in \mathbb{R}^3$ be*

$$A = \begin{bmatrix} -3 & -1 & 4 \\ 4 & 2 & -4 \\ 5 & 4 & -4 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}.$$

Then we have

$$c = Ab = \begin{bmatrix} -3 & -1 & 4 \\ 4 & 2 & -4 \\ 5 & 4 & -4 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} = \begin{bmatrix} -3\times 2 + (-1)\times(-1) + 4\times 3 \\ 4\times 2 + 2\times(-1) + -4\times 3 \\ 5\times 2 + 4\times(-1) + -4\times 3 \end{bmatrix} = \begin{bmatrix} 7 \\ -6 \\ -6 \end{bmatrix}$$

Alternatively, we have

$$c = Ab = \begin{bmatrix} -3 & -1 & 4 \\ 4 & 2 & -4 \\ 5 & 4 & -4 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} = 2 \begin{bmatrix} -3 \\ 4 \\ 5 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \\ 4 \end{bmatrix} + 3 \begin{bmatrix} 4 \\ -4 \\ -4 \end{bmatrix} = \begin{bmatrix} 7 \\ -6 \\ -6 \end{bmatrix}$$

Matrix-maxtrix multiplication. Given a matrix $A \in \mathbb{R}^{n\times m}$ and a matrix $B \in \mathbb{R}^{m\times k}$, the matrix-matrix multiplication results in a matrix $C \in \mathbb{R}^{n\times k}$ such that the klth element of C is given by $\sum_{j=1}^{m} a_{kj}b_{jl}$. In other words,

$$\begin{aligned} C = AB &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mk} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{j=1}^{m} a_{1j}b_{j1} & \sum_{j=1}^{m} a_{1j}b_{j2} & \cdots & \sum_{j=1}^{m} a_{1j}b_{jk} \\ \sum_{j=1}^{m} a_{2j}b_{j1} & \sum_{j=1}^{m} a_{2j}b_{j2} & \cdots & \sum_{j=1}^{m} a_{2j}b_{jk} \\ \vdots & & & \\ \sum_{j=1}^{m} a_{nj}b_{j1} & \sum_{j=1}^{m} a_{nj}b_{j2} & \cdots & \sum_{j=1}^{m} a_{nj}b_{jk} \end{bmatrix} \end{aligned} \tag{A.3}$$

Example A.2.2 (Matrix-matrix multiplication)**.** *Let $A \in \mathbb{R}^{3\times 3}$ and $B \in \mathbb{R}^{3\times}$ be*

$$A = \begin{bmatrix} -3 & -1 & 4 \\ 4 & 2 & -4 \\ 5 & 4 & -4 \end{bmatrix}, \qquad B = \begin{bmatrix} 3 & -6 & 6 \\ 1 & 8 & -4 \\ 7 & -2 & 8 \end{bmatrix}.$$

Then we have

$$C = AB = \begin{bmatrix} -3 & -1 & 4 \\ 4 & 2 & -4 \\ 5 & 4 & -4 \end{bmatrix} \times \begin{bmatrix} 3 & -6 & 6 \\ 1 & 8 & -4 \\ 7 & -2 & 8 \end{bmatrix} = \begin{bmatrix} 18 & 2 & 18 \\ -14 & 0 & -16 \\ -9 & 10 & -18 \end{bmatrix}$$

Matrix rank. Given a matrix $A \in \mathbb{R}^{n\times m}$, its rank is defined as the number of linearly independent columns or rows, and is denoted as $\text{rank}(A)$. Furthermore we have $\text{rank}(A) \leq n$ and $\text{rank}(A) \leq m$.

Example A.2.3 (Matrix rank)**.** *Let $A \in \mathbb{R}^{3\times 3}$ be*

$$A = \begin{bmatrix} -3 & -1 & -5 \\ 4 & 2 & 8 \\ 5 & 4 & 13 \end{bmatrix} = \begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix}$$

Then we have $A_3 = A_1 + 2A_2$, so A_3 is linearly dependent on A_1 and A_2. It is trivial to verify that A_1 and A_2 are linearly independent. Therefore, rank(A) = 2.

Determinant of a square matrix. Given a square matrix $A \in \mathbb{R}^{n\times n}$, its determinant is recursively defined as follows. For a 1×1 matrix, its determinant is defined as itself. For $n = 2, 3, \ldots,$ the determinant of A is defined as, for any arbitrary $1 \leq j \leq n$,

$$\det A = \sum_{i=1}^{n} a_{ij}(-1)^{i+j} \det\left(\bar{A}_{ij}\right),$$

where a_{ij} is the ijth element of A and $\bar{A}_{ij}$ is the $(n-1)\times(n-1)$ submatrix of A by removing its ith row and jth column.

Remark A.2.1. *When A is a triangular or diagonal matrix, its determinant,* $\det A$ *is given by the product of all diagonal elements.*

Remark A.2.2. *When $n=2$, denote $A\in\mathbb{R}^{2\times 2}$ as*

$$A=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}.$$

Then we have

$$\det A = a_{11}\times(-1)^{1+1}\times a_{22} + a_{21}\times(-1)^{2+1}\times a_{12} = a_{11}a_{22}-a_{21}a_{12}$$

Example A.2.4 (Matrix determinant)**.** *Let $A\in\mathbb{R}^{3\times 3}$ be*

$$A=\begin{bmatrix}-3 & -1 & 4\\ 4 & 2 & -4\\ 5 & 4 & -4\end{bmatrix}.$$

Picking $j=1$, then we have

$$\begin{aligned}\det A &= (-3)\times(-1)^{1+1}\times\det\bar{A}_{11} + 4\times(-1)^{2+1}\times\det\bar{A}_{21} + 5\times(-1)^{3+1}\times\det\bar{A}_{31}\\ &= -3\times\det\bar{A}_{11} - 4\times\det\bar{A}_{21} + 5\times\det\bar{A}_{31}\end{aligned}$$

where

$$\bar{A}_{11}=\begin{bmatrix}2 & -4\\ 4 & -4\end{bmatrix}$$
$$\bar{A}_{21}=\begin{bmatrix}-1 & 4\\ 4 & -4\end{bmatrix}$$
$$\bar{A}_{31}=\begin{bmatrix}-1 & 4\\ 2 & -4\end{bmatrix},$$

and therefore,

$$\begin{aligned}\det\bar{A}_{11} &= -8+16=8\\ \det\bar{A}_{21} &= 4-16=-12\\ \det\bar{A}_{31} &= 4-8=-4.\end{aligned}$$

So we have

$$\det A = -3\times 8 - 4\times(-12) + 5\times(-4) = 4$$

Inverse of a square matrix A square matrix $A\in\mathbb{R}^{n\times n}$ is said to be nonsingular, or have full rank, if $\mathrm{rank}(A)=n$ or $\det(A)\neq 0$. Given $A\in\mathbb{R}^{n\times n}$, its inverse is denoted as A^{-1} and satisfies the following

$$AA^{-1}=A^{-1}A=I. \tag{A.4}$$

When $n=2$, A^{-1} can be computed as follows.

$$A^{-1}=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}^{-1}=\frac{1}{\det A}\begin{bmatrix}a_{22} & -a_{12}\\ -a_{21} & a_{11}\end{bmatrix}=\frac{1}{a_{11}a_{22}-a_{21}a_{12}}\begin{bmatrix}a_{22} & -a_{12}\\ -a_{21} & a_{11}\end{bmatrix}$$

For $n>3$, computing A^{-1} generally involves matrix factorization. The MATLAB command "inv" can be used in this case.

Example A.2.5 (Matrix inversion)**.** *Let* $A \in \mathbb{R}^{2\times 2}$ *be*

$$A = \begin{bmatrix} 2 & -4 \\ 4 & -4 \end{bmatrix}$$

Then we have

$$A^{-1} = \frac{1}{2\times(-4) - 4\times(-4)} \begin{bmatrix} -4 & 4 \\ -4 & 2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

Vector norm. Given a vector $x \in \mathbb{R}^n$, the p-norm of x is denoted as $\|x\|_p$ is defined as

$$\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}. \tag{A.5}$$

When $p = 1$, the above definition reduces to 1-norm and is given by

$$\|x\|_1 := \sum_{i=1}^{n} |x_i|. \tag{A.6}$$

When $p = 2$, the above definition reduces to 2-norm and is given by

$$\|x\|_2 := \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{A.7}$$

Note that 2-norm is also called Euclidean norm and can be equivalently represented as

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{x^T x}. \tag{A.8}$$

When $p = \infty$, the infinity norm (or ∞-norm) is given by

$$\|x\|_\infty := \max_i |x_i|. \tag{A.9}$$

Remark A.2.3. *Note that the concept of norm is a generalization of length or magnitude, and should satisfy the properties of triangle inequality, absolute homogeneity, and positive definiteness. In fact, the definition (A.5) of p-norm satisfies these requirements. In other words, we have* $\|x+y\|_p \le \|x\|_p + \|y\|_p$, $\|ax\|_p = |a|\,\|x\|_p$, *and* $\|x\|_p \ge 0$ *while* $\|x\|_p = 0 \Rightarrow x = 0$.

Remark A.2.4. *Note also that in many case, when the context is clear, we use* $\|x\|$ *without any subscript to denote* 2*-norm.*

Another notation that we would use extensively is the norm of the projected vector. Given a vector $x \in \mathbb{R}^n$ and a positive definite matrix[1] $Q \in \mathbb{R}^n$, we define $\|x\|_Q$ and $\|x\|_Q^2$ as follows

$$\|x\|_Q := \sqrt{x^T Q x} \tag{A.10a}$$

$$\|x\|_Q^2 := x^T Q x \tag{A.10b}$$

[1] A matrix $Q \in \mathbb{R}^n$ is positive definite if and only if $x^T Q x > 0$ for all $x \in \mathbb{R}^n$ such that $\|x\| \neq 0$.

Example A.2.6 (Vector norm)**.** *Given* $x = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix}^T$ *Then we have*

$$\|x\|_1 = \sum_{i=1}^{3} |x_i| = |2| + |-1| + |3| = 6$$
$$\|x\|_2 = \sqrt{\sum_{i=1}^{3} x_i^2} = \sqrt{4+1+9} = \sqrt{14} = 3.7417$$
$$\|x\|_\infty = \max(|2|, |-1|, |3|) = 3$$

A.3 Eigenvalues and Eigenvectors

For a square matrix $A \in \mathbb{R}^{n\times n}$, its eigenvalues $\lambda \in \mathbb{C}$ and eigenvectors $v \in \mathbb{C}^n$ are such that the following equation is satisfied.

$$Av = \lambda v. \tag{A.11}$$

In other words, consider A as a transformation matrix, then the eigenvectors are such that their directions are not changed after the transformation, only their magnitude. Note that 0 vector always satisfies (A.11), and is not of interests here. That is to say, we only consider nonzero v to be eigenvectors. With a slight change of (A.11), we have

$$(A - \lambda I)v = 0 \tag{A.12}$$

Since we only consider nonzero v, this is equivalent to $(A-\lambda I)$ being singular and v lies in the null space of $(A - \lambda I)$. Therefore, λ can be obtained by solving the following characteristic equation of A,

$$\det(A - \lambda I) = 0.$$

Once the eigenvalues are found, the associated eigenvector v of each λ can be found by solving (A.12).

Remark A.3.1. *It is trivial to see that, following the above analysis, for a triangular or diagonal matrix, its eigenvalues are given by the elements on the diagonal.*

Example A.3.1 (Eigenvalues and Eigenvectors)**.** *Let* $A \in \mathbb{R}^{2\times 2}$ *be*

$$A = \begin{bmatrix} 6 & -3 \\ 4 & -1 \end{bmatrix}$$

To find its eigenvalues, we first compute the characteristic equation

$$\det(A - \lambda I) = \det \begin{bmatrix} 6-\lambda & -3 \\ 4 & -1-\lambda \end{bmatrix} = (6-\lambda)(-1-\lambda) + 12 = \lambda^2 - 5\lambda + 6 = (\lambda - 2)(\lambda - 3) = 0.$$

Solving the above equation gives us the two eigenvalues

$$\lambda_1 = 2, \quad \lambda_2 = 3.$$

To find the eigenvector associated with $\lambda_1 = 2$*, we denote* $v_1 = \begin{bmatrix} k_1 & k_2 \end{bmatrix}^T$ *and solve the following equation*

$$(A - \lambda_1 I)v_1 = 0$$

$$\begin{bmatrix} 4 & -3 \\ 4 & -3 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = 0$$
$$4k_1 - 3k_2 = 0$$

Furthermore, we are normally interested in eigenvectors with norm of 1. In other words, $k_1^2 + k_2^2 = 1$. *Therefore we have*

$$\frac{9}{16}k_2^2 + k_2^2 = 1 \Rightarrow k_2 = \frac{4}{5}$$

and

$$v_1 = \begin{bmatrix} 0.6 \\ 0.8 \end{bmatrix}$$

To find the eigenvector associated with $\lambda_2 = 3$, *we denote* $v_2 = \begin{bmatrix} p_1 & p_2 \end{bmatrix}^T$ *and solve the following equation*

$$(A - \lambda_2 I)v_2 = 0$$
$$\begin{bmatrix} 3 & -3 \\ 4 & -4 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = 0$$
$$p_1 - p_2 = 0$$

Together with $p_1^2 + p_2^2 = 1$, *we have* $p = \sqrt{2}/2 = 0.7071$ *and*

$$v_1 = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}.$$

A.4 Linear Algebraic Equations

Consider a set of linear algebraic equations compactly represented as

$$Ax = b \tag{A.13}$$

where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Assume A and b are known and x is the unknown to be solved. When $m > n$, there are more equations then unknowns, and when $m < n$, there are more unknown than equations.

Theorem A.4.1 (Existence of solution of (A.13))**.** *For a given A, a solution x to (A.13) exists for every b, if and only if A has full row rank, i.e., row rank of A equals m.*

Theorem A.4.2 (Uniqueness of solution of (A.13) for square A)**.** *For a given nonsingular square matrix* $A \in \mathbb{R}^{n \times n}$, *then* $Ax = b$ *has a unique solution for every b, which is given by*

$$x = A^{-1}b.$$

Furthermore, the only solution of $Ax = 0$ *is* $x = 0$.

Example A.4.1 (Solution of linear systems)**.** *Let*

$$A = \begin{bmatrix} -3 & -1 & 4 \\ 4 & 2 & -4 \\ 5 & 4 & -4 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

```
>> A = [-3 -1  4; 4  2  -4; 5  4  -4]

A =
    -3    -1     4
     4     2    -4
     5     4    -4

>> b = [2; -1; 3]

b =
     2
    -1
     3

>> c = A*b

c =
     7
    -6
    -6
```

Listing A.1: MATLAB Commands for Example A.2.1.

Find x such that $Ax = b$. To solve for x, we first compute the inverse of A, as follows.

$$A^{-1} = \begin{bmatrix} 2 & 3 & -1 \\ -1 & -2 & 1 \\ 1.5 & 1.75 & -0.5 \end{bmatrix}$$

Then we have

$$x = A^{-1}b = \begin{bmatrix} 2 & 3 & -1 \\ -1 & -2 & 1 \\ 1.5 & 1.75 & -0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 3.5 \end{bmatrix}$$

A.5 Matlab Codes

```
>> A = [-3 -1 4; 4 2 -4; 5 4 -4]

A =
    -3    -1     4
     4     2    -4
     5     4    -4

>> B = [3 -6 6; 1 8 -4; 7 -2 8]

B =
     3    -6     6
     1     8    -4
     7    -2     8

>> C = A*B

C =
    18     2    18
   -14     0   -16
    -9    10   -18
```

Listing A.2: MATLAB Commands for Example A.2.2.

```
>> A = [-3 -1 -5; 4 2 8; 5 4 13]

A =

    -3    -1    -5
     4     2     8
     5     4    13

>> rank(A)

ans =

     2
```

Listing A.3: MATLAB Commands for Example A.2.3.

```
>> A11=[2  -4; 4  -4]

A11 =
     2    -4
     4    -4

>> det(A11)

ans =
     8

>> A = [-3 -1  4; 4  2  -4; 5  4  -4]

A =
    -3    -1     4
     4     2    -4
     5     4    -4

>> det(A)

ans =
    4.0000
```

Listing A.4: MATLAB Commands for Example A.2.4.

```
>> A=[2  -4; 4  -4]

A =
     2    -4
     4    -4

>> inv(A)

ans =
   -0.5000    0.5000
   -0.5000    0.2500
```

Listing A.5: MATLAB Commands for Example A.2.5.

```
>> x = [2; -1 ;3]

x =

     2
    -1
     3

>> n1 = norm(x,1)

n1 =

     6

>> n2 = norm(x,2)

n2 =

    3.7417

>> n2 = norm(x) % norm(x) is same as norm(x,2)

n2 =

    3.7417

>> ninf = norm(x,inf)

ninf =

     3
```

Listing A.6: MATLAB Commands for Example A.2.6.

```
>> A=[6  -3; 4  -1]

A =
      6     -3
      4     -1

>> eig(A)

ans =
      3
      2

>> [v,d]=eig(A)

v =
    0.7071    0.6000
    0.7071    0.8000


d =
      3     0
      0     2
```

Listing A.7: MATLAB Commands for Example A.3.1.

```
>> A=[-3 -1  4; 4  2  -4; 5  4  -4]

A =

    -3    -1     4
     4     2    -4
     5     4    -4

>> b=[1; 2; 3]

b =

     1
     2
     3

>> x=inv(A)*b

x =

    5.0000
   -2.0000
    3.5000

>> A\b

ans =

    5.0000
   -2.0000
    3.5000
```

Listing A.8: MATLAB Commands for Example A.4.1.

Appendix B

Homework

B.1 Chapter 2

Problem 1. Consider the following nonlinear system

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1, x_2, x_3, u) = x_1^2 + x_1 x_2 + x_3^2 u - 3 \\
\dot{x}_2 &= f_2(x_1, x_2, x_3, u) = x_1 + x_1 x_2 + u - 3 \\
\dot{x}_3 &= f_3(x_1, x_2, x_3, u) = x_1 + x_2 u + x_3^2 - 2 \\
y &= h(x_1, x_2, x_3, u) = x_1 + x_3^3 + u
\end{aligned}$$

(1) Given a nominal point that $\bar{u} = 0$, $\bar{x}_1 = 1$, $\bar{x}_2 = 2$, $\bar{x}_3 = -1$ and $\bar{y} = 0$. Find the corresponding linearized representation.

(2) For the linear system found in (1), find the corresponding discrete time representation using forward Euler with sampling time $T_s = 0.1$.

(3) Determine the stability, observability, and controllablity for the discrete time system found in part (2).

B.2 Chapter 3

Problem 1. Consider the following optimization problem:

$$\min_{x_1, x_2} \quad x_1^2 + 3x_2^2 \qquad \text{subject to} \begin{cases} x_1 + 3x_2 = 1, \\ x_1 \leq 4 \\ x_2 \leq 10. \end{cases}$$

Show that $x^* = \begin{bmatrix} 0.25 & 0.25 \end{bmatrix}^T$ and $\lambda^* = \begin{bmatrix} -0.5 & 0 & 0 \end{bmatrix}^T$ satisfy the KKT condition.

Problem 2. Consider routing problem discussed in Section 3.3.1, but with an ending time $t = 3$. In other words, all transition cost from $t = 3$ to $t = 4$ can be ignored. Use dynamic programming to decide a sequence of control actions that minimizes the total cost from $t = 0$ to $t = 3$.

B.3 Chapter 4

Problem 1. Consider the receding horizon LQR (4.8) with prediction horizon p, how many recursive computations are needed to find the feedback gain? Compare it to the finite horizon LQR (4.2) of length N, how many storage space can be saved? Express your answer using p, N, n_x and n_u.

Problem 2. Complete the Matlab code in Listings 4.1 and 4.2.

B.4 Chapter 5

Problem 1. Explain why constraint $\epsilon \geq 0$ is not needed for the QP formulation of (5.20).

Problem 2. What matrices in the dense formulation (5.19) are constant and what matrices need to be updated in real-time?

Problem 3. Derive sparse QP formulation for (5.25).

B.5 Chapter 7

Consider the system in Example 7.2.1. Let

$$\hat{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \qquad P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{bmatrix}, \qquad Q = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}, \qquad R = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

Compute P_0^+.

Bibliography

[1] R. C. Dorf and R. H. Bishop, *Modern control systems*. Pearson Prentice Hall, 2008.

[2] X. Lin, H. E. Perez, S. Mohan, J. B. Siegel, A. G. Stefanopoulou, Y. Ding, and M. P. Castanier, "A lumped-parameter electro-thermal model for cylindrical batteries," *Journal of Power Sources*, vol. 257, pp. 1–11, 2014.

[3] C.-T. Chen, *Linear system theory and design*. Oxford University Press, Inc., 1998.

[4] N. Li, K. Zhang, Z. Li, V. Srivastava, and X. Yin, "Cloud-assisted nonlinear model predictive control for finite-duration tasks," *arXiv preprint arXiv:2106.10604*, 2021.

[5] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[6] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[7] A. Bemporad, F. Borrelli, and M. Morari, "Min-max control of constrained uncertain discrete-time linear systems," *IEEE Transactions on automatic control*, vol. 48, no. 9, pp. 1600–1606, 2003.

[8] D. Bernardini and A. Bemporad, "Stabilizing model predictive control of stochastic constrained linear systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1468–1480, 2011.

[9] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.