MODEL PREDICTIVE CONTROL

Manfred Morari
Jay H. Lee
Carlos E. García

February 8, 2002

Chapter 2

Modeling and Identification

Throughout the book we will assume that the control task is carried out by a computer as depicted in Fig. 2.1. The controlled output signal $y(t)$ is sampled at intervals T resulting in a sequence of measurements $\{y(0), y(1), \ldots, y(k), \ldots\}$ where the arguments $0, 1, \ldots, k, \ldots$ denote time measured in multiples of the sampling interval T. On the basis of this sequence the control algorithm determines a sequence of inputs $\{u(0), u(1), \ldots, u(k), \ldots\}$. The input time function $u(t)$ is obtained by holding the input constant over a sampling interval:

$$u(t) = u(k) \quad kT < t \leq (k+1)T$$

The appropriate control action depends on the dynamic characteristics of the plant to be controlled. Therefore, the control algorithm is usually designed based on a plant's *model* which describes its dynamic characteristics (at least approximately). In the context of Fig. 2.1 we are looking for a plant model which allows us to determine the sequence of outputs $\{y(0), y(1), \ldots, y(k), \ldots\}$ resulting from a sequence of inputs $\{u(0), u(1), \ldots, u(k), \ldots\}$.

In general, we are not only interested in the response of the plant to the manipulated variable (MV) u but also in its response to other inputs, for example, a disturbance variable (DV) d. We will use the symbol v to denote a "general" input which can be manipulated (u), or a disturbance (d). Thus the general plant model will relate a sequence of inputs $\{v(0), v(1), \ldots, v(k), \ldots\}$ to a sequence of outputs $\{y(0), y(1), \ldots, y(k), \ldots\}$.

Initially we will assume for our derivations that the system has only a single input v and a single output y (Single Input - Single Ouput (SISO) system). Later we will generalize the modeling concepts to describe multi-input multi-output (MIMO) systems.

In this chapter we will discuss some basic modeling assumptions and introduce *convolution models*. We will explain how these models can be used to predict the response of the output to an input sequence. Finally, we will give some advice on how to obtain the models from experiments.

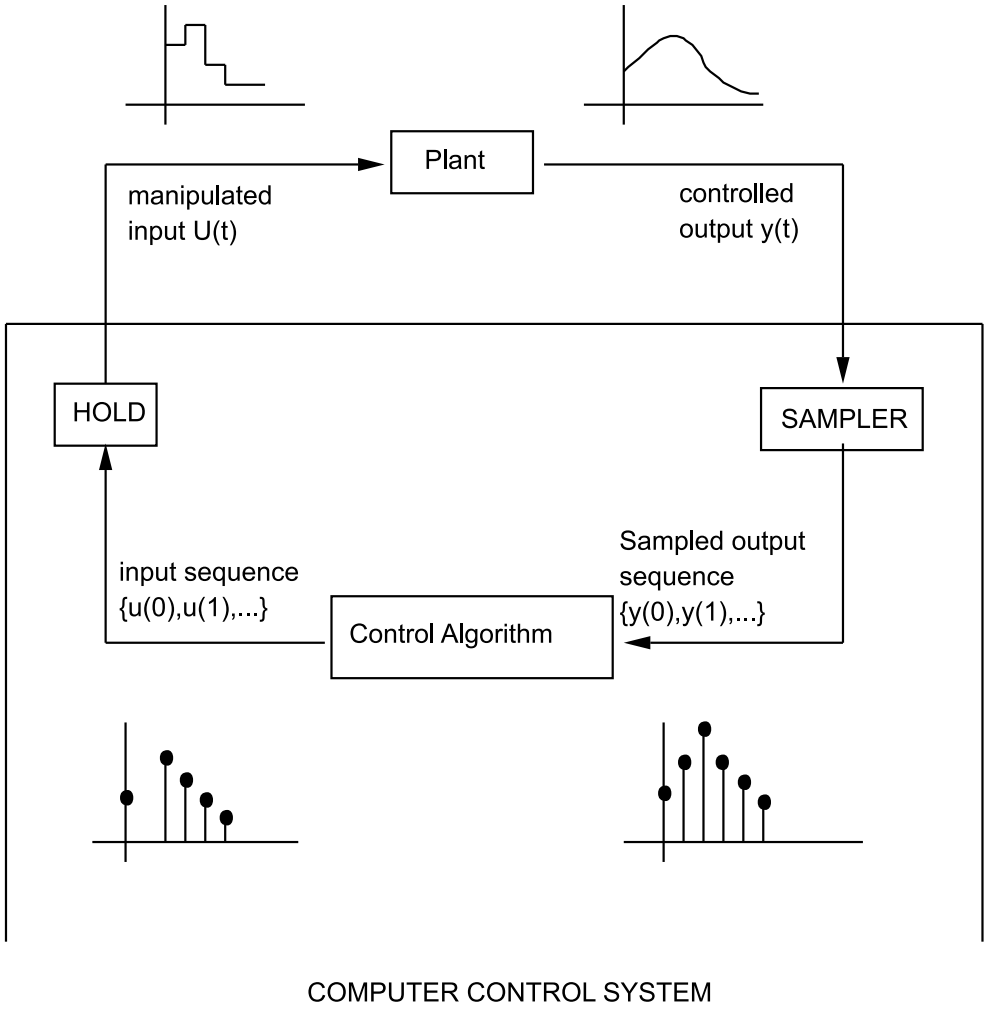

Figure 2.1: Sampled-data control system

2.1 Linear Time Invariant Systems

Let us consider two specific input sequences[1]

$$\mathbf{v}^1(0) = \left\{v^1(0), v^1(1), \ldots, v^1(k), \ldots\right\}$$

$$\mathbf{v}^2(0) = \left\{v^2(0), v^2(1), \ldots, v^2(k), \ldots\right\}$$

For example, $\mathbf{v}^1(0)$ might represent a step

$$\mathbf{v}^1(0) = \{1, 1, \ldots, 1, \ldots\}$$

[1]Throughout this book, we will adopt the convention that the input sequences are zero for negative time, i.e., for the example here $v(k) = 0,$ for $k < 0$.

and $\mathbf{v}^2(0)$ a pulse.

$$\mathbf{v}^2(0) = \{1, 0, \ldots, 0, \ldots\}$$

Let us assume that for a particular system P these two input sequences give rise to the output sequences

$$\mathbf{y}^1(0) = \left\{y^1(0), y^1(1), \ldots, y^1(k), \ldots\right\}$$

$$\mathbf{y}^2(0) = \left\{y^2(0), y^2(1), \ldots, y^2(k), \ldots\right\}$$

respectively. Symbolically, we can also write

$$\mathbf{v}^1(0) \quad \xrightarrow{P} \quad \mathbf{y}^1(0)$$

and

$$\mathbf{v}^2(0) \quad \xrightarrow{P} \quad \mathbf{y}^2(0)$$

Let us consider now the two input sequences

$$\mathbf{v}^3(0) \quad \stackrel{\Delta}{=} \quad \alpha\mathbf{v}^1(0) = \left\{\alpha v^1(0), \alpha v^1(1), \ldots, \alpha v^1(k), \ldots\right\}$$

where α is a real constant, and

$$\mathbf{v}^4(0) \quad \stackrel{\Delta}{=} \quad \mathbf{v}^1(0)+\mathbf{v}^2(0) = \left\{v^1(0) + v^2(0), v^1(1) + v^2(1), \ldots, v^1(k) + v^2(k), \ldots\right\}$$

If it is true that changing the input magnitude by a factor α changes the output magnitude by the same factor

$$\alpha\mathbf{v}^1(0) = \mathbf{v}^3(0) \quad \xrightarrow{P} \quad \mathbf{y}^3(0) = \alpha\mathbf{y}^1(0)$$

and that the response to a sum of two input sequences is the sum of the two output sequences

$$\mathbf{v}^1(0) + \mathbf{v}^2(0) = \mathbf{v}^4(0) \quad \xrightarrow{P} \quad \mathbf{y}^4(0) = \mathbf{y}^1(0) + \mathbf{y}^2(0)$$

for arbitrary choices of $\mathbf{v}^1(0), \mathbf{v}^2(0)$ and $-\infty < \alpha < +\infty$, then the system P is called *linear*.

Linearity is a very useful property. It has the following implication: If we know that the input sequences $\mathbf{v}^1(0), \mathbf{v}^2(0), \mathbf{v}^3(0)$, etc. yield the output sequences $\mathbf{y}^1(0), \mathbf{y}^2(0), \mathbf{y}^3(0)$ etc., respectively, then the response to any linear combination of input sequences is simply a linear combination of the output sequences. We also say that the process output can be obtained by *superposition*.

$$\alpha_1\mathbf{v}^1(0)+\alpha_2\mathbf{v}^2(0)+\ldots+\alpha_k\mathbf{v}^k(0)+\ldots \quad \xrightarrow{P} \quad \alpha_1\mathbf{y}^1(0)+\alpha_2\mathbf{y}^2(0)+\ldots+\alpha_k\mathbf{y}^k(0)+\ldots$$

where $-\infty < \alpha_i < \infty$.

In practice there are no truly linear systems. Linearity is a mathematical idealization which allows us to utilize a wealth of very powerful theory. In a small neighborhood around a fixed operating point the behavior of many systems is approximately linear. Because the purpose of control is usually to keep the system operation close to a fixed point, the setpoint, linearity is often a very good assumption.

Let us define now the shifted input sequence

$$\mathbf{v}^1(-\ell) = \left\{ \underbrace{0, 0, \ldots, 0}_{\ell}, v^1(1), v^1(2), \ldots, v^1(k), \ldots \right\}$$

where the first ℓ elements are zero. (Here we have simply moved the signal $\mathbf{v}^1(0)$ to the right by ℓ time steps). If the resulting system output sequence is shifted in the same manner

$$\mathbf{y}^1(-\ell) = \left\{ \underbrace{0, 0, \ldots, 0}_{\ell}, y^1(1), y^1(2), \ldots, y^1(k), \ldots \right\}$$

and if this system property

$$\mathbf{v}(-\ell) \quad \stackrel{P}{\longrightarrow} \quad \mathbf{y}(-\ell)$$

holds for arbitrary input sequences $\mathbf{v}(0)$ and arbitrary integers ℓ then the system is referred to as *time-invariant*. Time invariance implies, for example, that if tomorrow the same input sequence is applied to the plant as today, then the same output sequence will result. Throughout this part of the book we will assume that *linear time-invariant* models are adequate for the systems we are trying to describe and control.

2.2 System Stability

Let us perturb the input v to a system in a step-wise fashion and observe the output y. We can distinguish the three types of system behavior depicted in Fig. 2.2.

A *stable system* settles to a steady state after an input perturbation. Asymptotically the output does not change. An *unstable system* does not settle, its output continues to change. The output of an *integrating system* approaches a ramp asymptotically; it changes in a linear fashion. The output of an *exponentially unstable system* changes in a exponential manner.

DO!

Figure 2.2: Output responses to a step input change for stable, integrating and unstable systems.

Most processes encountered in the process industries are stable. An important example of an integrating process is shown in Fig. 2.3B.

At the outlet of the tank a controller is installed which delivers a constant flow. If a step change occurs in the inflow, the level will rise in a ramp-like fashion. (Or in the case of a pulse change shown, the level changes to a different value on a permanent basis.)

Some chemical reactors are exponentially unstable when left uncontrolled. However, for safety reasons, reactors are usually designed to be stable so that there is no danger of runaway when the control system fails. The control concepts discussed in this part of the book are not applicable to exponentially unstable systems. This system type will be dealt with in the advanced part.

2.3 Impulse Response Models

Let us inject a unit pulse

$$\mathbf{v}(0) = \{1, 0, \ldots, 0, \ldots\}$$

into a system at rest and observe the response

$$\mathbf{y}(0) = \{h_0, h_1, \ldots, h_n, h_{n+1} \ldots\}\,.$$

We will assume that the system has the following characteristics:

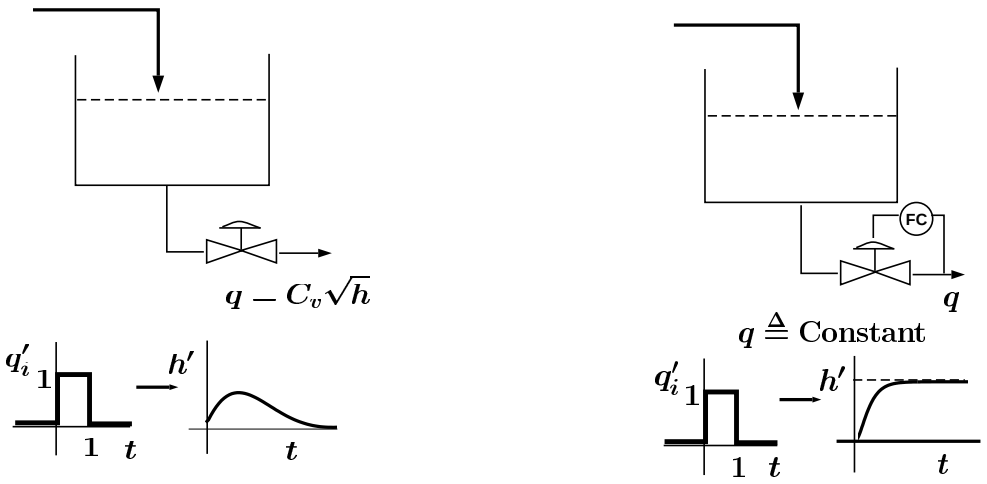

Figure 2.3: Water tank

- $h_0 = 0$, i.e. the systems does not react immediately to the input
- $h_k = 0$ for $k > n$, i.e. the system settles after n time steps

Such a system is called a Finite Impulse Response (FIR) system with the matrix of FIR coefficients

$$H = [h_1, h_2 \ldots, h_n]^T. \tag{2.1}$$

Many practical systems can be approximated well by FIR systems. Consider the simple examples in Fig. 2.3. In one case the outflow of the tank occurs through a fixed opening. Therefore it depends on the liquid height in the tank. After the pulse inflow perturbation the level returns to its original value. The system can be approximated by an FIR system. In the other case the outflow is determined by a pump and is constant. The pulse adds liquid volume which permanently changes the level in the tank. This *integrating system* cannot be approximated by an FIR model.

For linear time-invariant systems, shifting the input pulse

$$\mathbf{v}(0) = \{0, 1, 0, \ldots, 0, \ldots\}$$

will simply shift the impulse response

$$\mathbf{y}(0) = \{0, h_0, h_1, \ldots, h_n, h_{n+1} \ldots\}.$$

The dynamical behavior of an FIR system is completely characterized by the set of FIR coefficients. The system response can be computed by superposition as follows. Any arbitrary input

$$\mathbf{v}(0) = \{v(0), v(1), v(2) \ldots\}$$

can be represented as a sum of impulses

$$\begin{aligned}\mathbf{v}(0) &= \{1,0,0,\ldots\}\, v(0) \\ &\quad + \{0,1,0,\ldots\}\, v(1) \\ &\quad + \{0,0,1,0\ldots\}\, v(2) \\ &\quad + \ldots\end{aligned}$$

The system output is obtained by summing the impulse responses weighted by their respective impulse strength $v(i)$

$$\begin{aligned}\mathbf{y}(0) &= \{0,h_1,h_2\ldots,h_n,0,0,\ldots\}\, v(0) \\ &\quad + \{0,0,h_1,h_2,\ldots,h_n,0,0,\ldots\}\cdot v(1) \\ &\quad + \{0,0,0,h_1,h_2,\ldots,h_n,0,0,\ldots\}\cdot v(2) \\ &\quad + \ldots \\ &= \{0,h_1v(0),h_2v(0)+h_1v(1),h_3v(0)+h_2v(1)+h_1v(2),\ldots\}\end{aligned}$$

Directly by inspection we see from this derivation that at a particular time the output is given by

$$y(k) = \sum_{i=1}^{n} h_i v(k-i). \tag{2.2}$$

The coefficient h_i expresses the effect of an input, which occurred i intervals in the past, on the present output $y(k)$. In order to compute this output we need to keep the last n inputs $v(k-1), v(k-2), \ldots, v(k-n)$ in memory.

2.4 Step Response Models

For a unit input step

$$\mathbf{v}(0) = \{1,1,\ldots,1,\ldots\}$$

the system output will be

$$\begin{aligned}\mathbf{y}(0) &= \{0,h_1,h_1+h_2,h_1+h_2+h_3,\ldots\} &(2.3)\\ &\stackrel{\Delta}{=} \{0,s_1,s_2,s_3,\ldots\} &(2.4)\end{aligned}$$

Here we have defined the step response coefficients $s_1, s_2, s_3, \ldots$. For linear time-invariant systems the shifted step

$$\mathbf{v}(0) = \{0,1,1,\ldots,1,\ldots\}$$

will give rise to a shifted step response

$$\mathbf{y}(0) = \{0,0,s_1,s_2,s_3,\ldots\}\,.$$

The matrix of step response coefficients

$$S = [s_1, s_2, \ldots, s_n]^T \tag{2.5}$$

is a *complete* description of how a particular input affects the system output when the system is at rest. Any arbitrary input

$$\mathbf{v}(0) = \{v(0), v(1), v(2), \ldots\}$$

can be represented as a sum of steps

$$\begin{aligned} \mathbf{v}(0) \quad = \quad & \{1,1,1,1,\ldots\}v(0) \\ & +\{0,1,1,1,\ldots\}(v(1)-v(0)) \\ & +\{0,0,1,1,\ldots\}(v(2)-v(1)) \\ & +\ldots \end{aligned}$$

where we will define $\Delta v(i)$ to denote the input changes

$$\Delta v(i) = v(i) - v(i-1) \tag{2.6}$$

from one time step to the next. The system output is obtained by summing the step responses weighted by their respective step heights $\Delta v(i)$

$$\begin{aligned} \mathbf{y}(0) \quad = \quad & \{0, s_1, \ldots, s_n, s_n, s_n, \ldots\}\, v(0) \\ & +\{0, 0, s_1, \ldots, s_n, s_n, s_n, \ldots\}\, \Delta v(1) \\ & +\{0, 0, 0, s_1, \ldots, s_n, s_n, s_n, \ldots\}\, \Delta v(2) \\ & +\ldots \\ = \quad & \{0, s_1 v(0), s_2 v(0) + s_1 \Delta v(1), s_3 v(0) + s_2 \Delta v(1) + s_1 \Delta v(2), \ldots \\ & \ldots, s_n v(0) + s_{n-1}\Delta v(1) + s_{n-2}\Delta v(2) + \ldots + s_1 \Delta v(n-1), \\ & \left. s_n \underbrace{(v(0) + \Delta v(1))}_{v(1)} + s_{n-1}\Delta v(2) + s_{n-2}\Delta v(3) + \ldots + s_1 \Delta v(n), \ldots \right\} \end{aligned}$$

We see that at a particular time the output is given by

$$y(k) = \sum_{i=1}^{n-1} s_i \Delta v(k-i) + s_n v(k-n). \tag{2.7}$$

The coefficient s_i expresses the effect of an input change, which occurred i intervals in the past, on the present output $y(k)$. In order to compute this output we need to keep the last n inputs in memory.

The step response coefficients are directly calculable from the impulse response coefficients and vice versa.

$$s_k = \sum_{i=1}^{k} h_i \tag{2.8}$$

$$h_k = s_k - s_{k-1} \tag{2.9}$$

2.5 Multi-Step Prediction

The objective of the controller is to determine the control action such that a desirable output behavior results in the future. Thus we need to be able to predict efficiently the future output behavior of the system. This future behavior will be a function of past inputs to the process and future inputs, i.e., inputs which we are considering to take in the future. We will separate the effects of the past inputs and the future inputs on the output. All past input information will be summarized in the *dynamic state* of the system. Thus the future output behavior will be determined by the present system state and the present and future inputs to the system.

Usually the representation of the state is not unique. For example, for an FIR system we could choose the past n inputs as the state x.

$$x(k) = [v(k-1),\ v(k-2),\ \ldots,\ v(k-n)]^T \tag{2.10}$$

Clearly this state summarizes all relevant past input information for an FIR system and allows us to compute the future evolution of the system when we are given the present input $v(k)$ and the future inputs $v(k+1), v(k+2), \ldots$.

There are other possible choices. For example, instead of the n past inputs we could choose the *effect* of the past inputs on the future outputs at the next n steps as the state. In other words, we define as the state the present output and the $n-1$ future outputs – assuming that the present and future inputs are zero. The two states are equivalent in that they both have dimension n and are related uniquely by a linear map.

The latter choice of state will prove more convenient for predictive control computations. It shows explicitly how the system will evolve when there is no control action and therefore allows us to determine easily what control action should be taken to achieve a specified behavior of the outputs in the future. In the next sections we will discuss how to do multi-step prediction for FIR and step-response models based on this state representation.

2.5.1 Multi-Step Prediction Based on FIR Model

Let us define the state at time k as

$$\bar{Y}(k) = \left[\ \bar{y}_0(k),\ \ \bar{y}_1(k),\ \ \ldots,\ \ \bar{y}_{n-1}(k)\ \right]^T \tag{2.11}$$

where

$$\bar{y}_i(k) \stackrel{\Delta}{=} y(k+i) \text{ for } v(k+j) = 0; j \geq 0 \tag{2.12}$$

Thus we have defined the i^{th} system state $\bar{y}_i(k)$ as the system output at time $k+i$ under the assumption the system inputs are zero from time k into the future ($v(k+j) = 0; \quad j \geq 0$). This state completely characterizes the evolution of the system output under the assumption that the present and future

inputs are zero. In order to determine the future output we simply add to the state the effect of the present and future inputs using (2.2).

$$\begin{aligned} y(k+1) &= \bar{y}_1(k) + h_1 v(k) && (2.13)\\ y(k+2) &= \bar{y}_2(k) + h_1 v(k+1) + h_2 v(k) && (2.14)\\ y(k+3) &= \bar{y}_3(k) + h_1 v(k+2) + h_2 v(k+1) + h_3 v(k) && (2.15)\\ y(k+4) &= \dots && (2.16) \end{aligned}$$

We can put these equations in matrix form

$$\begin{bmatrix} y(k+1) \\ y(k+2) \\ \vdots \\ \vdots \\ y(k+p) \end{bmatrix} = \underbrace{\begin{bmatrix} \bar{y}_1(k) \\ \bar{y}_2(k) \\ \vdots \\ \vdots \\ \bar{y}_p(k) \end{bmatrix}}_{\text{effect of past inputs from } Y^0(k)} + \underbrace{\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ \vdots \\ h_p \end{bmatrix} v(k) + \begin{bmatrix} 0 \\ h_1 \\ \vdots \\ \vdots \\ h_{p-1} \end{bmatrix} v(k+1) + \dots\dots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ h_1 \end{bmatrix} v(k+p-1)}_{\text{effect of future inputs (yet to be determined)}}$$

and note that the first term is a part of the state and reflects the effect of the past inputs. The other terms express the effect of the hypothesized future inputs. They are simply the responses to impulses occurring at the future time steps.

In order to obtain the state at $k+1$, which according to the definition is

$$\bar{Y}(k+1) = \left[\ \bar{y}_0(k+1),\ \ \bar{y}_1(k+1),\ \ \dots,\ \ \bar{y}_{n-1}(k+1)\ \right]^T \quad \text{with} \qquad (2.17)$$

$$\bar{y}_i(k+1) \stackrel{\Delta}{=} y(k+1+i) \text{ for } v(k+1+j) = 0; j \geq 0, \qquad (2.18)$$

we need to add the effect of the input $v(k)$ at time k to the state $\bar{Y}(k)$.

$$\begin{aligned} \bar{y}_0(k+1) &= \bar{y}_1(k) + h_1 v(k) && (2.19)\\ \bar{y}_1(k+1) &= \bar{y}_2(k) + h_2 v(k) && (2.20)\\ &\dots && (2.21)\\ \bar{y}_{n-1}(k+1) &= \bar{y}_n(k) + h_n v(k) && (2.22) \end{aligned}$$

We note that $\bar{y}_n(k)$ was not a part of the state at time k, but we know it to

be 0 because of the FIR assumption. By defining the matrix

$$M = \left.\begin{bmatrix} 0 & 1 & 0 & \dots & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \dots & \dots & \dots & 0 & 1 \\ 0 & 0 & \dots & \dots & \dots & 0 & 0 \end{bmatrix}\right\} \quad n \tag{2.23}$$

we can express the state update compactly as

$$\bar{Y}(k+1) = M \cdot \bar{Y}(k) + Hv(k). \tag{2.24}$$

Multiplication with the matrix M in the above represents the simple operation of shifting the vector $\bar{Y}(k)$ and setting the last element of the resulting vector to zero.

2.5.2 Recursive Multi-Step Prediction Based on Step-Response Model

Let us define the state at time k as

$$\tilde{Y}(k) = \left[\ \tilde{y}_0(k), \quad \tilde{y}_1(k), \quad \dots, \quad \tilde{y}_{n-1}(k)\ \right]^T \tag{2.25}$$

where

$$\tilde{y}_i(k) \stackrel{\Delta}{=} y(k+i) \text{ for } \Delta v(k+j) = 0; j \geq 0 \tag{2.26}$$

Thus, in this case, we have defined the state $\tilde{y}_i(k)$ as the system output at time $k+i$ under the assumption the input *changes* are zero from time k into the future $\Delta(v(k+i) = 0; \quad i \geq 0)$. Note that because of the FIR assumption the step response settles after n steps, i.e., $\tilde{y}_{k+n-1}(k) = \tilde{y}_{k+n}(k) = \ldots = \tilde{y}_\infty(k)$. Hence, the choice of state $\tilde{Y}(k)$ completely characterizes the evolution of the system output under the assumption that the present and future input changes are zero. In order to determine the future output we simply add to the state the effect of the present and future input changes. From (2.7) we find

$$\begin{aligned} y(k+1) &= \sum_{i=1}^{n-1} s_{i+1}\Delta v(k-i) + s_n v(k-n) + s_1 \Delta v(k) && (2.27) \\ &= \tilde{y}_1(k) + s_1 \Delta v(k). && (2.28) \end{aligned}$$

Continuing for $k+2, k+3, \ldots$ we find

$$\begin{aligned} y(k+2) &= \tilde{y}_2(k) + s_1\Delta v(k+1) + s_2 \Delta v(k) && (2.29) \\ y(k+3) &= \tilde{y}_3(k) + s_1\Delta v(k+2) + s_2 \Delta v(k-1) + s_3 \Delta v(k) && (2.30) \\ y(k+4) &= \ldots && (2.31) \end{aligned}$$

We can put these equations in matrix form

$$\begin{bmatrix} y(k+1) \\ y(k+2) \\ \vdots \\ \vdots \\ y(k+p) \end{bmatrix} = \underbrace{\begin{bmatrix} \tilde{y}_1(k) \\ \tilde{y}_2(k) \\ \vdots \\ \vdots \\ \tilde{y}_p(k) \end{bmatrix}}_{\text{effect of past inputs + current bias from } \tilde{Y}(k)} + \underbrace{\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ \vdots \\ s_p \end{bmatrix} \Delta v(k) + \begin{bmatrix} 0 \\ s_1 \\ \vdots \\ \vdots \\ s_{p-1} \end{bmatrix} \Delta v(k+1) + \ldots\ldots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ s_1 \end{bmatrix} \Delta v(k+p-1)}_{\text{effect of future input changes (yet to be determined)}} \tag{2.32}$$

and note that the first term is a part of the state and reflects the effect of the past inputs. The other terms express the effect of the hypothesized future input changes. They are simply the reponses to steps occurring at the future time steps.

In order to obtain the state at $k+1$, which according to the definition is

$$\tilde{Y}(k+1) = \left[\; \tilde{y}_0(k+1), \quad \tilde{y}_1(k+1), \quad \ldots, \quad \tilde{y}_{n-1}(k+1) \; \right]^T \quad \text{with} \tag{2.33}$$

$$\tilde{y}_i(k+1) \stackrel{\Delta}{=} y(k+1+i) \text{ for } \Delta v(k+1+j) = 0; j \geq 0, \tag{2.34}$$

we need to add the effect of the input change $\Delta v(k)$ at time k to the state $\tilde{Y}(k)$.

$$\begin{aligned} \tilde{y}_0(k+1) &= \tilde{y}_1(k) + s_1 \Delta v(k) && (2.35) \\ \tilde{y}_1(k+1) &= \tilde{y}_2(k) + s_2 \Delta v(k) && (2.36) \\ &\ldots && (2.37) \\ \tilde{y}_2(k+1) &= \tilde{y}_n(k) + s_n \Delta v(k) && (2.38) \end{aligned}$$

We note that $\tilde{y}_n(k) =$
$tildey_{n-1}(k)$ because of the FIR assumption. By defining the matrix

$$M = \left.\begin{bmatrix} 0 & 1 & 0 & \ldots & \ldots & 0 & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \ldots & \ldots & \ldots & 0 & 1 \\ 0 & 0 & \ldots & \ldots & \ldots & 0 & 1 \end{bmatrix}\right\} \; n \tag{2.39}$$

we can express the state update compactly as

$$\tilde{Y}(k+1) = M \cdot \tilde{Y}(k) + S \Delta v(k). \tag{2.40}$$

Multiplication with the matrix M denotes the operation of shifting the vector $\tilde{Y}(k)$ and repeating the last element. The recursive relation (2.40) is referred to as the *step response model* of the system.

As is apparent from the derivation, the FIR and the step response models are very similar. The definition of the state is slightly different. In the FIR model the future inputs are assumed to be zero, in the step response model the future input *changes* are kept zero. Also the input representation is different. For the FIR model the future inputs are given in terms of pulses, for the step response model the future inputs are steps. Because the step response model expresses the future inputs in terms of changes Δv, it will be very convenient for incorporating integral action in the controller as we will show.

2.5.3 Multivariable Generalization

The model equation (2.40) generalizes readily to the case when the system has n_y outputs $y\ell, \ell = 1, \ldots, n_y$ and n_v inputs $vj, j = 1, \ldots, n_v$. We define the output *vector*

$$y(k-1) = [y1(k-1), \ldots, yn_y(k-1)]^T,$$

input *vector* $\Delta v(k-1) = [\Delta v1(k-1), \ldots, \Delta vn_v(k-1)]^T$ and step response coefficient matrix

$$S_i \quad = \quad \begin{bmatrix} s_{1,1,i} & s_{1,2,i} & \ldots & s_{1,n_v,i} \\ s_{2,1,i} & & & \\ \vdots & & & \\ s_{n_y,1,i} & s_{n_y,2,i} & \ldots & s_{n_y,n_v,i} \end{bmatrix}$$

where $s_{\ell,m,i}$ is the i^{th} step response coefficient relating the m^{th} input to the ℓ^{th} output. The $(n_y \cdot n) \times n_u$ step response matrix is obtained by stacking up the step response coefficient matrices

$$S = \left[S_1^T, S_2^T, S_3^T, \ldots, S_n^T\right]^T.$$

The state of the multiple output system at time k is defined as

$$\begin{array}{rcll} \tilde{Y}(k) & = & \left[\ \tilde{y}_0(k), \quad \tilde{y}_1(k), \quad \ldots, \quad \tilde{y}_{n-1}(k)\ \right]^T & \text{with} \\ \tilde{y}_i(k) & \stackrel{\Delta}{=} & y(k+i) \text{ for } \Delta v(k+j) = 0; j \geq 0 & \end{array} \tag{2.41}$$

With these definitions an equivalent update equation results as in the single variable case (2.40)

$$\tilde{Y}(k+1) = M \cdot \tilde{Y}(k) + S\Delta v(k). \tag{2.42}$$

Here M is defined as

$$M = \left.\begin{bmatrix} 0 & I & 0 & \dots & \dots & 0 & 0 \\ 0 & 0 & I & 0 & \dots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \dots & \dots & \dots & 0 & I \\ 0 & 0 & \dots & \dots & \dots & 0 & I \end{bmatrix}\right\} n$$

where the identity matrix I is of dimension $n_y \times n_y$. The recursive relation (2.42) is referred to as the *step response model* of the multivariable system. It was proposed in the original formulation of Dynamic Matrix Control.

2.6 Examples

The following examples illustrate different types of step response models.

Example 1 *SISO process with deadtime and disturbance. The step responses in Fig. 2.4 show the effect of "side draw" (manipulated variable) and "intermediate reflux duty" (disturbance) on "side endpoint" in the heavy crude fractionator example problem. The sampling time is 7 minutes and 35 step response coefficients were generated. Note the deadtimes of approximately 14 minutes in both responses. (The transfer functions used for the simulation were* $g_{sd} = 5.72e^{-14s}/(60s+1)$ *and* $g_{ird} = 1.52e^{-15s}/(25s+1)$*).*

Example 2 *SISO process with inverse response. The step response in Fig. 2.5 shows the experimentally observed effect ref?? of a change in steam flowrate on product concentration in a multieffect evaporator system. The long term effect of the steam rate increase is to increase evaporation and thus the concentration. The initial decrease of concentration is caused by an increased (diluting) flow from adjacent vaporizers. (The transfer function used for the simulation was* $g = \frac{2.69(-6s+1)e^{-1.5s}}{(20s+1)(5s+1)}$*).*

The control of systems with "inverse response" is a special challenge: the controller must not be mislead by the inverse-response effect and its action must be cautious.

Example 3 *MIMO process. Figure 2.6 shows the response of overhead* (y_1) *and bottom* (y_2) *compositions of a high purity distillation column to changes in reflux* (u_1) *and boilup* (u_2)*. Note that all the responses are very similar which will cause control problems as we will see later. (The transfer matrix used for the simulation was* $G = \frac{1}{75s+1}\begin{bmatrix} 0.878 & 0.864 \\ 1.082 & 1.096 \end{bmatrix}$ *).*

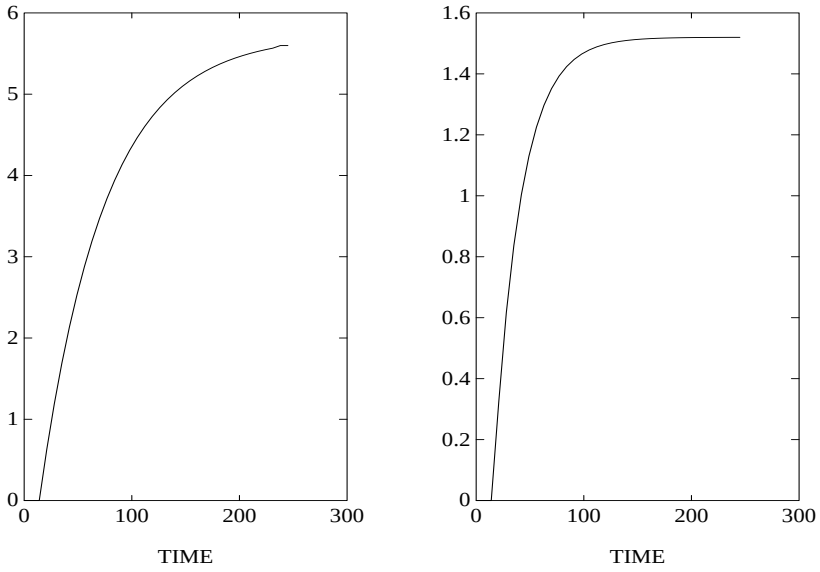

Figure 2.4: Responses of side endpoint for step change in side draw (left) and intermediate reflux duty (right). ($n = 35, T = 7$)

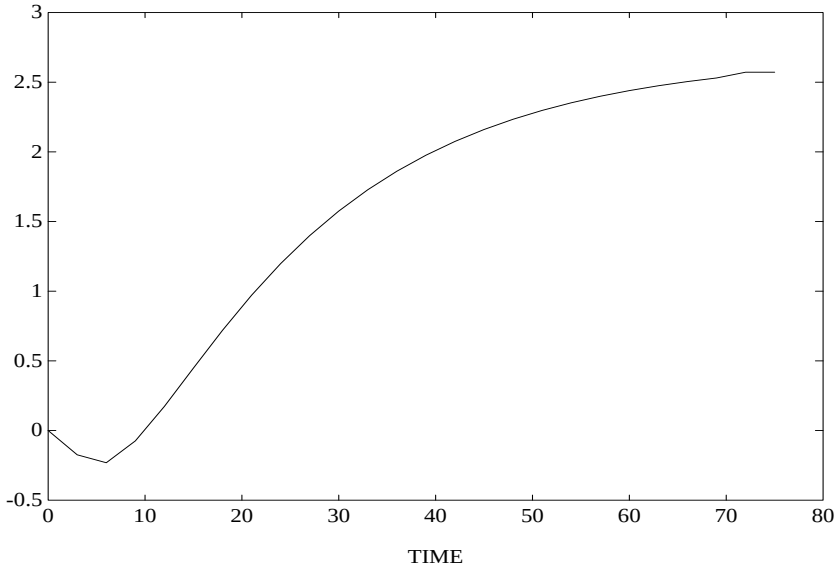

Figure 2.5: Response of evaporator product concentration to steam flowrate ($n = 25, T = 3$).

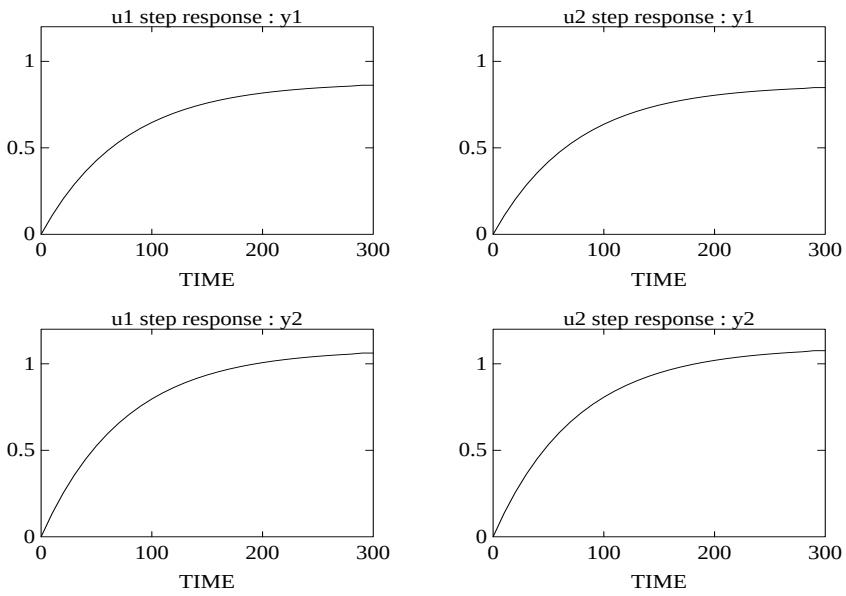

Figure 2.6: Response of bottom and top composition of a high purity distillation column to step changes in boilup and reflux ($n = 30, T = 10$).

2.7 Identification

Two approaches are available for obtaining the models needed for prediction and control move computation. One can derive the differential equations representing the various material, energy and momentum balances and solve these equations numerically. Thus one can simulate the response of the system to a step input change and obtain a step response model. This "fundamental" modeling approach is quite complex and requires much engineering effort, because the physicochemical phenomena have to be understood and all the process parameters have to be known or determined from experiments. The main advantage of the procedure is that the resulting differential–equation models are usually valid over wide ranges of operating conditions.

The second approach to modeling is to perturb the inputs of the real process and record the output responses. By relating the inputs and outputs a process model can be derived. This approach is referred to as *process identification*. Especially in situations when the process is complex and the fundamental phenomena are not well understood, experimental process identification is the preferred modeling approach in industry.

In this section, we will discuss the direct identification of an FIR model. The primary advantage of fitting an FIR model is that the only parameters to be specified by the user are the sampling time and the response length n. This is in contrast with other techniques which use more structured models and thus require that a structure identification step be performed first.

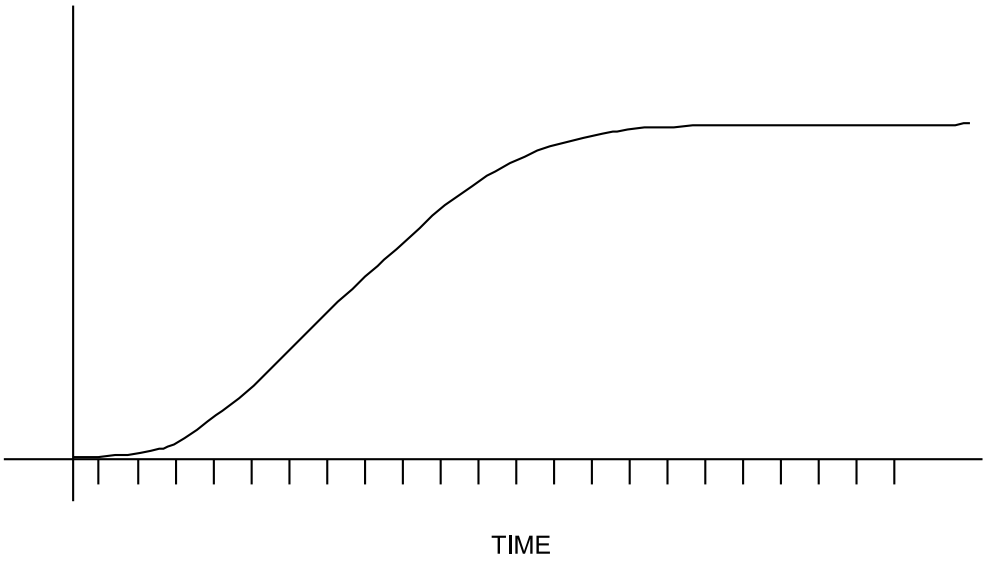

Figure 2.7: Step response

One of the simplest ways to obtain the step response coefficients is to step up or step down one of the inputs. The main drawbacks are that it is not always easy to carry out the experiments and even if they are successful the model is usually valid only in an operating region close to where the experiment was performed. In addition, measurement noises and disturbances can significantly corrupt the result, especially during initial transient periods.

For the experimental identification approaches, a number of issues need to be addressed.

2.7.1 Settling Time

Very few physical processes are actually FIR but the step response coefficients become "essentially" constant after some time. Thus, we have to determine by inspection a time beyond which the step response coefficients do not change appreciably and define an appropriate *truncated* step response model which is FIR. If we choose this time too short the model is in error and the performance of a controller based on this model will suffer. On the other hand, if we choose this time too long the model will include many step response coefficients, which makes the prediction and control computation unwieldy.

2.7.2 Sampling Time

Usually the minimum sampling interval is dictated by the control computer. However, often this minimum sampling interval is smaller than necessary for

effective control. For example, if a process has a deadtime of ten minutes it is unnecessary to sample the output and recompute the control action every 30 sec. Usually faster sampling leads to better closed–loop performance, but the closed–loop performance of a process with a ten–minute deadtime is severely limited by the deadtime (no reaction for ten minutes) and fast sampling does not yield any performance gains. The following rule of thumb covers most practical cases:

$$\text{Sampling time} = \max\{0.03 \times \text{settling time}, 0.3 \times \text{deadtime}\}$$

For large deadtimes the sampling time is determined by the deadtime, for small ones by the settling time. Typically step response models with about 30 step response coefficients are employed in practice (Settling Time/Sampling Time ≈ 30). If the number of coefficients necessary to describe the response adequately is significantly smaller or larger, then the sampling time should be decreased or increased, respectively.

2.7.3 Choice of the Input Signal for Experimental Identification

The simplest test is to step up or step down the inputs to obtain the step response coefficients. When one tries to determine the impulse response directly from step experiments one is faced with several experimental problems.

At the beginning of the experiment the plant has to be at *steady state* which is often difficult to accomplish because of disturbances over which the experimenter has no control. A typical response starting from a non–zero initial condition is shown in Fig. 2.8 B. Disturbances can also occur during the experiment leading to responses like that in Fig. 2.8 C.

Finally, there might be so much noise that the step response is difficult to recognize Fig. 2.8 D. It is then necessary to increase the magnitude of the input step and thus to increase the signal–noise ratio of the output signal. Large steps, however, are frowned upon by the operating personnel, who are worried about, for example, off–spec products when the operating conditions are disturbed too much.

In the presence of significant noise, disturbances and non–steady state initial conditions, it is better to choose input signals that are more "random" in nature. This idea of "randomness" of signals will be made more precise in the advanced section of the book. Input signals that are more "random" should help reduce the adverse effects due to non–steady state initial conditions, disturbances and noise. However, we recommend any engineer to try to perform a step test first since it is the easiest experiment.

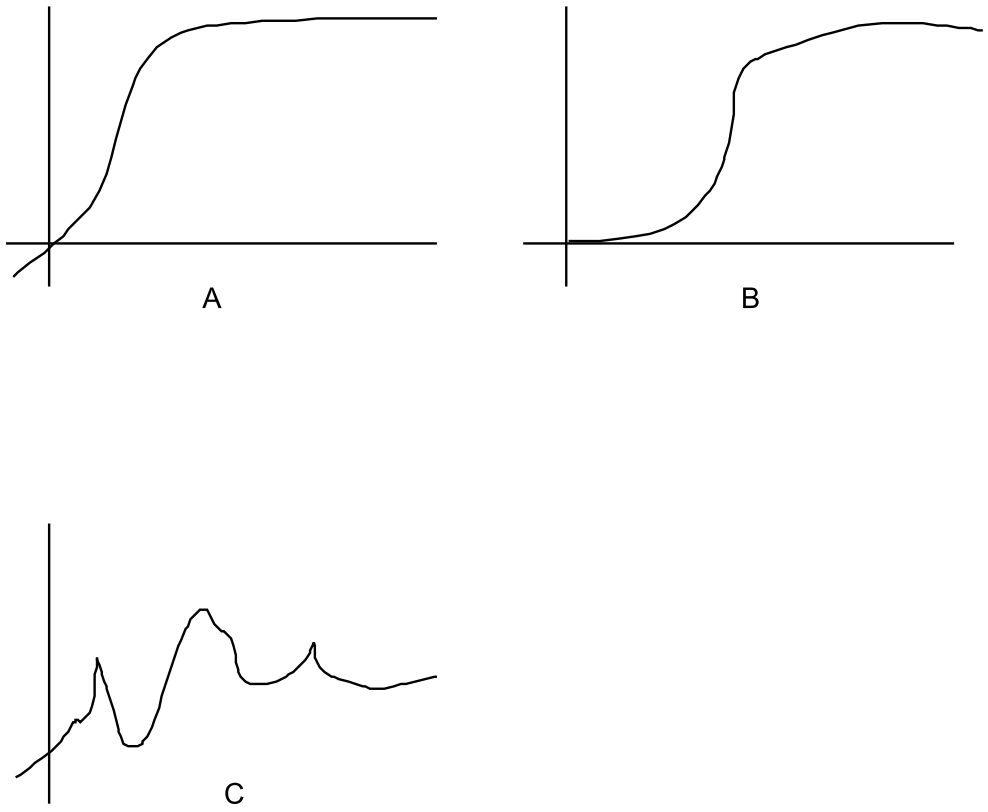

Figure 2.8: (A) "True" step response. (B) Erroneous step responses caused by non–steady state initial conditions and (C) unmeasured disturbances, (D) measurement noise.

2.7.4 The Linear Least Squares Problem

In order to explain the basic technique to identify an FIR model for arbitrary input signals, we need to introduce some basic concepts of linear algebra. Let us assume that the following system of linear equations is given:

$$b = Ax \tag{2.43}$$

where the matrix A has more rows than columns. That is, there exist more equations than there are unknowns, the linear system is *overspecified*. Let us also assume that A has *full column rank*. This means that the rank of A is equal to the dimension of the solution vector x.

This system of equations has no exact solution because there are not enough degrees of freedom to satisfy all equations simultaneously. However, one can find a solution that makes the left hand and right hand sides "close" to each other. One way is to find a vector x that minimizes the sum of squares of the equation *residuals*. Let us denote the vector of residuals as

$$\rho = b - Ax \tag{2.44}$$

The optimization problem that finds this solution is formulated as follows:

$$\min_x \ \rho^T \rho = \min_x (b - Ax)^T (b - Ax) \tag{2.45}$$

The necessary condition for optimality yields the equations[2]

$$\frac{d(\rho^T \rho)}{dx} = -2A^T (b - Ax) = 0 \tag{2.47}$$

Note that the second order derivative

$$\frac{d^2(\rho^T \rho)}{dx^2} = 2A^T A \tag{2.48}$$

is positive definite because of the rank condition of A, and thus the solution of (2.47) for x

$$x = (A^T A)^{-1} A^T b \tag{2.49}$$

minimizes the objective function. This solution of a set of overspecified linear equations is denoted as a *least squares* solution. The matrix $(A^T A)^{-1} A^T$ is denoted as the *generalized inverse* of the matrix A. Note that in case A is square, its generalized inverse is A^{-1}, the exact inverse.

In some cases, it may be desirable to weigh each component of the residual vector ρ differently in finding x. The solution that minimizes the 2-norm of the weighted residual vector $\Lambda\rho$ is

$$x = (A^T \Lambda^T \Lambda A)^{-1} A^T \Lambda^T \Lambda b \tag{2.50}$$

Note that formulas (2.49) and (2.50) are algebraically correct, but are never used for numerical computations. Most available software employs the QR algorithm for determining the least squares solution. The reader is referred to the appropriate numerical analysis literature for more information.

2.7.5 Linear Least Squares Identification

The standard parameter fitting algorithm employed in identification is the least squares algorithm or one of its many variants. In the most general case it fits

[2]The derivative of the quadratic expression $f^T f$ where $f = f(x)$ is defined as follows:

$$\frac{d(f^T f)}{dx} = 2\frac{df^T}{dx} f \tag{2.46}$$

a model relating one output variable of a process to multiple inputs (the same algorithm is used for all outputs from the same test data). When many inputs vary simultaneously they must be uncorrelated so that the individual models can be obtained. Directly fitting an FIR model provides many advantages to an inexperienced user. For example, apart from the settling time of the process, it requires essentially no other *a priori* information about the process. Here we give a brief derivation of the least squares identification technique and some modifications to overcome the drawbacks of directly fitting an FIR model.

Let us assume the process has a single output y and a set of inputs $vj, j = 1, \ldots, n_v$. Let us also assume that the process is described by an impulse response model as follows:

$$y(k) = \sum_{j=1}^{n_v} \sum_{i=1}^{n} h_{j,i} vj(k-i) + \nu(k-1) \tag{2.51}$$

where $\nu(k-1)$ collects any effects on y not described by the model.

We assume that n is selected sufficiently large so that ν only has to account for unmeasured input effects such as noise, disturbances, and bias in the steady-state values.

In practice, it is difficult to define the steady states for inputs and outputs because they tend to change with time according to process disturbances. Hence, for the purpose of identification, it is common to re-express the model in terms of incremental changes in the inputs and outputs from sample time to sample time:

$$\Delta y(k) = \sum_{j=1}^{n_v} \sum_{i=1}^{n} h_{j,i} \Delta vj(k-i) + \Delta\nu(k-1) \tag{2.52}$$

(This model can be obtained directly by writing (2.51) for the times k and $k-1$ and differencing.) We assume that $\Delta\nu$ is independent of, or uncorrelated to Δv.

If we collect the values of the output and inputs over $N+n$ intervals of time from an on-line experiment we can estimate the impulse response coefficients of the model as follows. We can write (2.52) over the past N intervals of time up to the current time $N+n$:

$$\left[\underbrace{\begin{matrix} \Delta y(n+1) \\ \Delta y(n+2) \\ \vdots \\ \Delta y(n+N) \end{matrix}}_{Y_N} \right] =$$

$$\underbrace{\begin{bmatrix} \Delta v_1(n) & \cdots & \Delta v1(1) & \Delta v2(n) & \cdots & \Delta vn_v(1) \\ \Delta v_1(n+1) & \cdots & \Delta v1(2) & \Delta v2(n-1) & \cdots & \Delta vn_v(2) \\ & \vdots & & & \vdots & \\ \Delta v_1(n+N-1) & \cdots & \Delta v1(N) & \Delta v2(n+N-1) & \cdots & \Delta vn_v(N) \end{bmatrix}}_{\Phi_N}$$

$$\cdot \underbrace{\begin{bmatrix} h_{1,1} \\ \vdots \\ h_{1,n} \\ h_{2,1} \\ \vdots \\ h_{n_v,n} \end{bmatrix}}_{\theta} + \underbrace{\begin{bmatrix} \Delta\nu(n+1) \\ \Delta\nu(n+2) \\ \vdots \\ \Delta\nu(n+N) \end{bmatrix}}_{V_N} \tag{2.53}$$

(2.54)

A natural objective for selecting θ is to minimize the square norm of the residual vector V_N. With this objective, the parameter estimation problem is reduced to the least-squares problem discussed in the previous section:

$$Y_N = \Phi_N \theta \tag{2.55}$$

where Y_N is the vector of past output measurements, Φ_N is the matrix containing all past input measurements, θ is the vector of parameters to be identified. If the number of data points N is larger than the total number of parameters in θ $(n_v \cdot n)$, the following formula for the least squares estimate of θ for N data points, that is the choice of θ that minimizes the quantity $(Y_N - \Phi_N\theta)^T(Y_N - \Phi_N\theta)$, can be found from (2.49):

$$\hat{\theta}_N = [{\Phi_N}^T\Phi_N]^{-1}{\Phi_N}^T Y_N(k). \tag{2.56}$$

In practical identification experiments, the expected variance of the error $\nu(k)$ for each data point may vary. For example, the engineer may know that the data points obtained during a certain time interval are corrupted by more severe noise than others. It is logical that the errors for the data points that are likely to be inaccurate are weighed less in calculating θ than others. This can be easily done by finding θ that minimizes the weighted square of the residuals. Higher weights are assigned to data points that are believed to be more accurate. Once again, this is formulated into a least squares problem whose objective is to minimize

$$(Y_N - \Phi_N\theta)^T\Lambda_\nu^T\Lambda_\nu(Y_N - \Phi_N\theta) \tag{2.57}$$

The solution follows from:

$$\hat{\theta}_N = [{\Phi_N}^T \Lambda_\nu^T \Lambda_\nu \Phi_N]^{-1} {\Phi_N}^T \Lambda_\nu^T \Lambda_\nu Y_N(k). \tag{2.58}$$

It can be shown that if the underlying system is truly linear and n is large enough so that $\Delta\nu$ and Δv are uncorrelated, this estimate is *unbiased* (*i.e.*, it is "expected" to be right or right on the average). Also, the estimate converges to the true value as the number of data points N becomes large, under some mild condition. Reliable software exists to obtain the least squares estimates of the parameters θ.

A major drawback of this approach is that a large number of data points needs to be collected because of the many parameters to be fitted. This is required because in the presence of noise the variance of the parameters could be so large as to render the fit useless. Often, the resulting step response will be non-smooth with many sharp peaks. One simple approach to alleviate this difficulty is to add a penalty term on the magnitude of the *changes* of the step response coefficients, i.e. the impulse response coefficients, to be identified. In other words, θ is found such that the quantity $(Y_N - \Phi_N\theta)^T \Lambda_\nu^T \Lambda_\nu (Y_N - \Phi_N\theta) + \theta^T \Lambda_\theta^T \Lambda_\theta \theta$ is minimized where Λ_θ is a weighting matrix penalizing the magnitudes of the impulse response coefficients. In other words, the weight penalizes sharp changes in the step response coefficients. As before, this can be formulated as a least-squares problem

$$\begin{bmatrix} \Lambda_\nu Y_N \\ 0 \end{bmatrix} = \begin{bmatrix} \Lambda_\nu \Phi_N \\ \Lambda_\theta \end{bmatrix} \theta \tag{2.59}$$

yielding the solution

$$\hat{\theta}_N = [{\Phi_N}^T \Lambda_\nu^T \Lambda_\nu \Phi_N + \Lambda_\theta^T \Lambda_\theta]^{-1} {\Phi_N}^T \Lambda_\nu^T \Lambda_\nu Y_N \tag{2.60}$$

This simple modification to the standard least-squares identification algorithm should result in smoother step responses. One drawback of the method is that the optimal choice of the weighting matrix Λ_θ is often unclear. Choosing too large a Λ_θ can lead to severely biased estimates even with large data sets. On the other hand, too small a choice of Λ_θ may not smooth the step response sufficiently. Other more sophisticated statistical methods to reduce the error variance of the parameters will be discussed in the advanced part of this book.

The procedure above can also be used to fit measured disturbance models to be used for feedforward compensation in MPC. Designing the manipulated inputs such that they are uncorrelated with the disturbance should minimize problems when fitting disturbance models. Of course, the measured disturbance must also have enough natural excitation, which may be hard to guarantee.

We stress that process identification is the most time consuming step in the implementation of MPC. We have presented in this section a rudimentary discussion of the most basic identification technique. A more rigorous discussion of the technique as well as the discussion of other more advanced identification algorithms will be given in the advanced part of this book.

Chapter 3

Dynamic Matrix Control - The Basic Algorithm

Dynamic Matrix Control (DMC) was one of the first commercial implementations of Model Predictive Control (MPC). In this chapter we describe the basic ideas of the algorithm.

3.1 The Idea of Moving Horizon Control

Consider the diagram in Fig. 3.1. At the present time k the behavior of the process over a horizon p is considered. Using the model, the response of the process output to changes in the manipulated variable is predicted. Current and future moves of the manipulated variables are selected such that the predicted response has certain desirable (or *optimal*) characteristics. For instance, a commonly used objective is to minimize the sum of squares of the future errors, i.e., the deviations of the controlled variable from a desired target (setpoint). This minimization can also take into account constraints which may be present on the manipulated variables and the outputs.

The idea is appealing but would not work very well in practice if the moves of the manipulated variable determined at time k were applied blindly over the future horizon. Disturbances and modelling errors may lead to deviations between the predicted behavior and the actual observed behavior, so that the computed manipulated variable moves may not be appropriate any more. Therefore only the first one of the computed moves is actually implemented. At the next time step $k+1$ a measurement is taken, the horizon is shifted forward by one step, and the optimization is done again over this shifted horizon based on the current system information. Therefore this control strategy is also referred to as *moving horizon control*.

A similar strategy is used in many other non-technical situations. One

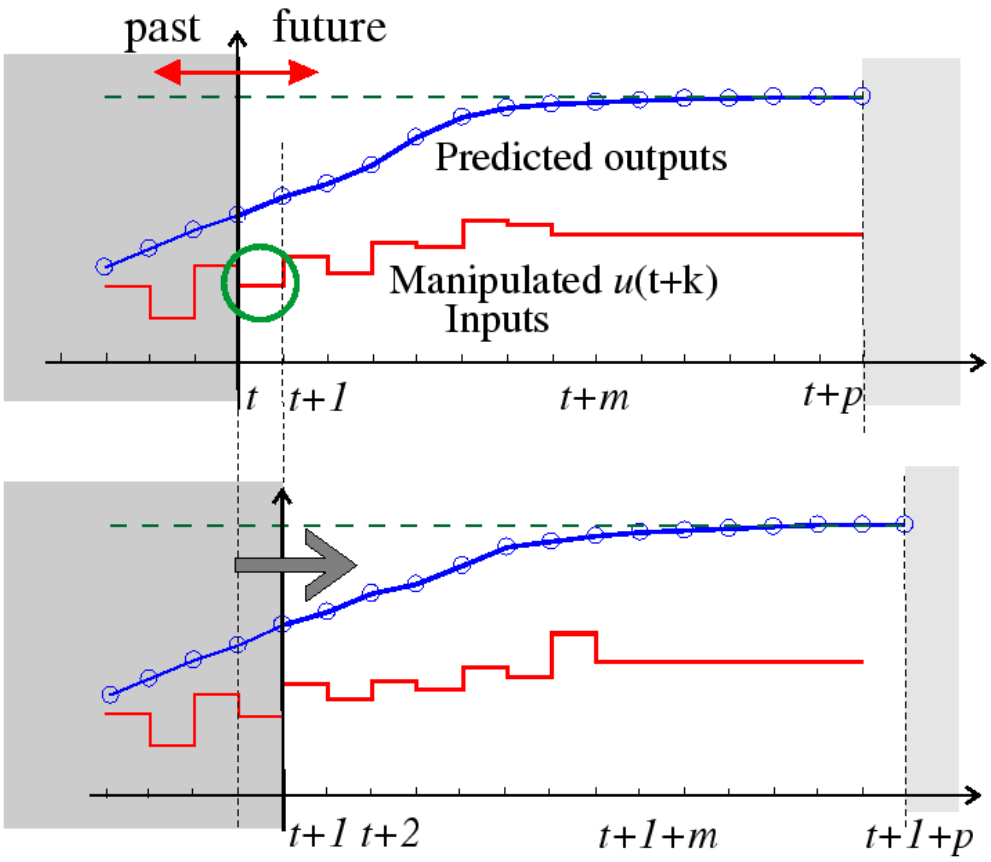

Figure 3.1: Moving Horizon Control

example is computer chess where the computer moves after evaluating all possible moves over a specified "depth" (the horizon). At the next turn the evaluation is repeated based on the current board situation. Another example would be investment planning. A five-year plan is established to maximize the return. Periodically a new five year plan is put together over a shifted horizon to take into account changes which have occurred in the economy.

The DMC algorithm includes as one of its major components a technique to predict the future output of the system as a function of the inputs and disturbances. This prediction capability is necessary to determine the optimal future control inputs and was outlined in the previous chapter. Afterwards we will state the objective function, formulate the optimization problem and comment on its solution. Finally we will discuss the various tuning parameters which are available to the user to affect the performance of the controller.

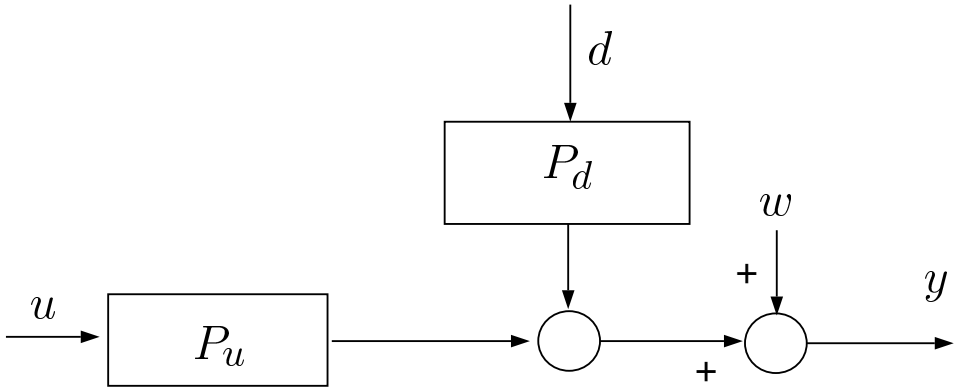

Figure 3.2: Basic Problem Setup

3.2 Multi-Step Prediction

We consider the setup depicted in Fig. 3.2 where we have three different types of external inputs: the manipulated variable (MV) u, whose effect on the output, usually a controlled variable (CV), is described by P_u; the measured disturbance variable (DV) d whose effect on the output is described by P_d; and finally the unmeasured and unmodeled disturbances w_y which add a bias to the system output. The overall system can be described by

$$y(k) = \begin{bmatrix} P^u & P^d \end{bmatrix} \begin{bmatrix} u(k) \\ d(k) \end{bmatrix} + w_y(k) \tag{3.1}$$

We assume that step response models S^u, S^d are available for the system dynamics P_u and P_d, respectively. We can define the overall multivariable step response model

$$S = \begin{bmatrix} S^u & S^d \end{bmatrix} \tag{3.2}$$

which is driven by the known overall input

$$\Delta v(k) = \begin{bmatrix} \Delta u(k) \\ \Delta d(k) \end{bmatrix}. \tag{3.3}$$

Let us adopt (2.41) as the system state

$$\tilde{Y}(k) = \begin{bmatrix} \tilde{y}_0(k), & \tilde{y}_1(k), & \ldots, & \tilde{y}_{n-1}(k) \end{bmatrix}^T \tag{3.4}$$

By definition the state consists of the future system outputs

$$\tilde{Y}(k) = \begin{bmatrix} y(k) \\ y(k+1) \\ \vdots \\ y(k+n-1) \end{bmatrix} \tag{3.5}$$

obtained under the assumption that the system inputs do not change from the previous values, *i.e.*,

$$\begin{array}{rcl} \Delta u(k) & = & \Delta u(k+1) = \cdots = 0 \\ \Delta d(k) & = & \Delta d(k+1) = \cdots = 0. \end{array} \tag{3.6}$$

Also, the state does not include any unmeasureed disturbance information and hence it is assumed in the definition that

$$w_y(k) = w_y(k+1) = \cdots = 0 \tag{3.7}$$

The state is updated according to (2.42)

$$\tilde{Y}(k) = M \cdot \tilde{Y}(k-1) + S\Delta v(k-1). \tag{3.8}$$

The equation reflects the effect of the input change $\Delta v(k-1)$ on the future evolution of the system assuming that there are no further input changes. The influence of the input change manifests itself through the step response matrix S. The effect of any future input changes is described as well by the appropriate step response matrix. Let us consider the predicted output over

the next p time steps

$$
\begin{aligned}
\begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ \vdots \\ \vdots \\ y(k+p|k) \end{bmatrix} &= \begin{bmatrix} \tilde{y}_1(k) \\ \tilde{y}_2(k) \\ \vdots \\ \vdots \\ \tilde{y}_p(k) \end{bmatrix} + \begin{bmatrix} S_1^u \\ S_2^u \\ \vdots \\ \vdots \\ S_p^u \end{bmatrix} \Delta u(k|k) \\
&+ \begin{bmatrix} 0 \\ S_1^u \\ S_2^u \\ \vdots \\ S_{p-1}^u \end{bmatrix} \Delta u(k+1|k) + \cdots + \cdots\cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ S_1^u \end{bmatrix} \Delta u(k+p-1|k) \\
&+ \begin{bmatrix} S_1^d \\ S_2^d \\ \vdots \\ \vdots \\ S_p^d \end{bmatrix} \Delta d(k) + \begin{bmatrix} 0 \\ S_1^d \\ S_2^d \\ \vdots \\ S_{p-1}^d \end{bmatrix} \Delta d(k+1|k) + \cdots \\
&\cdots\cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ S_1^d \end{bmatrix} \Delta d(k+p-1|k) \\
&+ \begin{bmatrix} w_y(k+1|k) \\ w_y(k+2|k) \\ \vdots \\ \vdots \\ w_y(k+p|k) \end{bmatrix}
\end{aligned}
\tag{3.9}
$$

Here the first term on the right hand side, the first p elements of the state, describes the future evolution of the system when all the future input changes are zero. The remaining terms describe the effect of the present and future changes of the manipulated inputs $\Delta u(k+i|k)$, the measured disturbances $\Delta d(k+i|k)$, and the unmeasured and unmodeled disturbances $w_y(k+i|k)$. The notation $y(k+i|k)$ represents the prediction of $y(k+i)$ made based on the information available at time k. The same notation applies to Δd and w_y.

The values of most of these variables are not available at time k and have to be *predicted* in a rational fashion. From the measurement at time k $d(k)$ is known and therefore $\Delta d(k) = d(k) - d(k-1)$. Unless some additional process information or "upstream" measurements are available to conclude about the future disturbance behavior, the disturbances are assumed not to change in

the future for the derivation of the DMC algorithm.

$$\Delta d(k+1|k) = \Delta d(k+2|k) = \cdots = \Delta d(k+p-1|k) = 0 \tag{3.10}$$

This assumption is reasonable when the disturbances are varying only infrequently. Similarly, we will assume that the future unmodeled disturbances $w_y(k+i|k)$ do not change.

$$w_y(k|k) = w_y(k+1|k) = w_y(k+2|k) = \cdots = w_y(k+p|k) \tag{3.11}$$

We can get an estimate of the present unmodeled disturbance from (3.1)

$$w_y(k|k) \approx y_m(k) - \tilde{y}_0(k). \tag{3.12}$$

where $y_m(k)$ represents the value of the output as actually measured in the plant. Here $\tilde{y}_0(k)$, the first component of the state $\tilde{Y}(k)$, is the model prediction of the output at time k (assuming $w_y(k) = 0$) based on the information up to this time. The difference between this predicted output and the measurement provides a good estimate of the unmodeled disturbance.

For generality we want to consider the case where the manipulated inputs are not varied over the whole horizon p but only over the next m steps $(\Delta u(k|k), \Delta u(k+1|k), \cdots, \Delta u(k+m-1|k))$ and that the input changes are set to zero after that.

$$\Delta u(k+m|k) = \Delta u(k+m+1|k) = \cdots = \Delta u(k+p-1)|k) = 0 \tag{3.13}$$

With these assumptions (3.9) becomes

$$\begin{aligned}\mathcal{Y}(k+1|k) = & \underbrace{\begin{bmatrix} \tilde{y}_1(k) \\ \tilde{y}_2(k) \\ \vdots \\ \vdots \\ \tilde{y}_p(k) \end{bmatrix}}_{\mathcal{M}\tilde{Y}(k)} \\ & \text{from the memory} \\ + & \underbrace{\begin{bmatrix} S_1^d \\ S_2^d \\ \vdots \\ \vdots \\ S_p^d \end{bmatrix} \Delta d(k)}_{\mathcal{S}^d \Delta d(k)} + \underbrace{\begin{bmatrix} y_m(k) - \tilde{y}_0(k) \\ y_m(k) - \tilde{y}_0(k) \\ \vdots \\ \vdots \\ y_m(k) - \tilde{y}_0(k) \end{bmatrix}}_{\mathcal{I}_p(y_m(k) - \tilde{y}_0(k))} \\ & \text{feedforward term} \qquad \text{feedback term} \\ + & \underbrace{\begin{bmatrix} S_1^u & 0 & \cdots & \cdots & 0 \\ S_2^u & S_1^u & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ S_m^u & S_{m-1}^u & \cdots & \cdots & S_1^u \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ S_p^u & S_{p-1}^u & \cdots & \cdots & S_{p-m+1}^u \end{bmatrix}}_{\mathcal{S}^u} \underbrace{\begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \vdots \\ \Delta u(k+m-1|k) \end{bmatrix}}_{\Delta\mathcal{U}(k)} \\ & \text{dynamic matrix} \qquad \text{future input moves} \end{aligned} \tag{3.14}$$

Here we have introduced the new symbols

$$\mathcal{Y}(k+1|k) = \begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ \vdots \\ y(k+p|k) \end{bmatrix} \tag{3.15}$$

$$\mathcal{S}^u = \begin{bmatrix} S_1^u & 0 & \dots & 0 \\ S_2^u & S_1^u & \dots & 0 \\ \vdots & \vdots & & \vdots \\ S_m^u & S_{m-1}^u & \dots & S_1^u \\ \vdots & \vdots & & \vdots \\ S_p^u & S_{p-1}^u & \dots & S_{p-m+1}^u \end{bmatrix}$$

$$\mathcal{S}^d = \begin{bmatrix} S_1^d \\ S_2^d \\ \vdots \\ S_p^d \end{bmatrix} \tag{3.16}$$

$$\mathcal{I}_p \quad = \quad \left.\begin{bmatrix} I \\ I \\ \vdots \\ I \end{bmatrix}\right\} p \tag{3.17}$$

$$\Delta\mathcal{U}(k) = \begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \Delta u(k+m-1|k) \end{bmatrix} \tag{3.18}$$

$$\mathcal{M} = \begin{cases} \left.\begin{bmatrix} 0 & I & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 0 & I & 0 & \dots & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & I & 0 & \cdots & 0 \end{bmatrix}\right\} p \quad \text{for p ¡ n} \\ \left.\begin{bmatrix} 0 & I & 0 & \dots & 0 \\ 0 & 0 & I & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & I \\ \vdots & \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 & I \end{bmatrix}\right\} p \quad \text{for } p \geq n \end{cases} \tag{3.19}$$

With this new notation the p-step ahead prediction becomes

$$\mathcal{Y}(k+1|k) = \mathcal{M}\tilde{Y}(k) + S^d\Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k)) + \mathcal{S}^u\Delta\mathcal{U}(k). \tag{3.20}$$

where the first three terms are completely defined by past control actions ($\tilde{Y}(k)$, $\tilde{y}_0(k)$) and present measurements ($y_m(k)$, $\Delta d(k)$) and the last term describes the effect of future manipulated variable moves $\Delta\mathcal{U}(k)$.

This prediction equation can be easily adjusted if different assumptions are made on the future behavior of the measured and unmeasured disturbances. For instance, if the disturbances are expected to evolve in a ramp-like fashion then we would set

$$\Delta d(k) = \Delta d(k+1|k) = \cdots = \Delta d(k+p-1|k) \tag{3.21}$$

and

$$w_y(k+\ell|k) = w_y(k|k) + \ell(w_y(k|k) - w_y(k-1|k-1)) \tag{3.22}$$

3.3 Objective Function

Plant operation requirements determine the performance criteria of the control system. These criteria must be expressed in mathematical terms so that a control law can be obtained in algorithmic form. In DMC a *quadratic* objective function is used which can be stated in its simplest form as[1]

$$\min_{\Delta u(k|k)\ldots\Delta u(k+m-1|k)} \sum_{\ell=1}^{p} \|y(k+\ell|k) - r(k+\ell)\|_{\cdot}^{2} \tag{3.23}$$

This criterion minimizes the sum of squared deviations of the predicted CV values from a time–varying reference trajectory or setpoint $r(k+\ell)$ over p future time steps. The quadratic criterion penalizes large deviations proportionally more than smaller ones so that on the average the output remains close to its reference trajectory and large excursions are avoided.

Note that the manipulated variables are assumed to be constant after m intervals of time into the future, or equivalently,

$$\Delta u(k+m|k) = \Delta u(k+m+1|k) = \cdots = \Delta u(k+p-1|k) = 0,$$

where $m \leq p$ always. This means that DMC determines the next m moves, only. The choices of m and p affect the closed–loop behavior. Moreover, m, the number of degrees of freedom, has a dominant influence on the computational effort. Also, it does not make sense to make the horizon longer than $m+n$ ($p \leq m+n$), because for an FIR system of order n the system reaches a steady state after $m+n$ steps. Increasing the horizon beyond $m+n$ would simply add identical constant terms to the objective function(3.23).

Due to inherent process interactions, it is generally not possible to keep all outputs close to their corresponding reference trajectories simultaneously. Therefore, in practice only a subset of the outputs is controlled well at the expense of larger excursions in others. This can be influenced transparently by including weights in the objective function as follows

$$\min_{\Delta u(k|k)\ldots\Delta u(k+m-1|k)} \sum_{\ell=1}^{p} \|\Gamma_{\ell}^{y}[y(k+\ell|k) - r(k+\ell)]\|_{\cdot}^{2} \tag{3.24}$$

For example, for a system with two outputs y_1 and y_2, and constant diagonal weight matrices of the form

$$\Gamma_{\ell}^{y} = \begin{bmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \end{bmatrix} \quad ;\forall \ell \tag{3.25}$$

[1] $\|x\|$ denotes the norm $(x^T x)^{\frac{1}{2}}$ of the vector x.

the objective becomes

$$\min_{\Delta u(k|k)\ldots\Delta u(k+m-1|k)} \quad \{\ {\gamma_1}^2 \sum_{\ell=1}^{p} [y_1(k+\ell|k) - r_1(k+\ell)]^2 + {\gamma_2}^2 \sum_{\ell=1}^{p} [y_2(k+\ell|k) - r_2(k+\ell)]^2\ \}. \tag{3.26}$$

Thus, the larger the weight is for a particular output, the larger is the contribution of its sum of squared deviations to the objective. This will make the controller bring the corresponding output closer to its reference trajectory.

Finally, the manipulated variable moves that make the output follow a given trajectory could be too severe to be acceptable in practice. This can be corrected by adding a penalty term for the manipulated variable moves to the objective as follows:

$$\min_{\Delta\mathcal{U}(k)} \sum_{\ell=1}^{p} \|\Gamma_\ell^y [y(k+\ell|k) - r(k+\ell)]\|^2 + \sum_{\ell=1}^{m} \|\Gamma_\ell^u [\Delta u(k+\ell-1)]\|^2. \tag{3.27}$$

Note that the larger the elements of the matrix Γ_ℓ^u are, the smaller the resulting moves will be, and consequently, the output trajectories will not be followed as closely. Thus, the relative magnitudes of Γ_ℓ^y and Γ_ℓ^u will determine the trade-off between following the trajectory closely and reducing the action of the manipulated variables.

Of course, not every practical performance criterion is faithfully represented by this quadratic objective. However, many control problems can be formulated as trajectory tracking problems and therefore this formulation is very useful. Most importantly this formulation leads to an optimization problem for which there exist effective solution techniques.

3.4 Constraints

In many control applications the desired performance cannot be expressed solely as a trajectory following problem. Many practical requirements are more naturally expressed as constraints on process variables.

There are three types of process constraints

Manipulated Variable Constraints: these are hard limits on inputs $u(k)$ to take care of, for example, valve saturation constraints;

Manipulated Variable Rate Constraints: these are hard limits on the size of the manipulated variable moves $\Delta u(k)$ to directly influence the rate of change of the manipulated variables;

Output Variable Constraints: hard or soft limits on the outputs of the system are imposed to, for example, avoid overshoots and undershoots. These can be of two kinds:

- *Controlled Variables:* limits for these variables are specified even though deviations from their setpoints are minimized in the objective function
- *Associated Variables:* no setpoints exist for these output variables but they must be kept within bounds (i.e. corresponding rows of Γ_ℓ^y are zero for the projections of these variables in the objective function given in (3.27).

The three types of constraints in DMC are enforced by formulating them as linear inequalities. In the following we explicitly formulate these inequalities.

3.4.1 Manipulated Variable Constraints

The solution vector of DMC contains not only the current moves to be implemented but also the moves for the future m intervals of time. Although violations can be avoided by constraining only the move to be implemented, constraints on future moves can be used to allow the algorithm to anticipate and prevent future violations thus producing a better overall response. The manipulated variable value at a future time $k+\ell$ is constrained to be

$$u_{low}(\ell) \leq \sum_{j=0}^{\ell} \Delta u(k+j|k) + u(k-1) \leq u_{high}(\ell);\ \ell = 0, 1, \ldots m-1$$

where $u(k-1)$ is the implemented previous value of the manipulated variable. For generality, we allowed the limits $u_{low}(\ell), u_{high}(\ell)$ to vary over the horizon. These constraints are expressed in matrix form for all projections as

$$\begin{bmatrix} -I_L \\ I_L \end{bmatrix} \Delta\mathcal{U}(k) \geq \begin{bmatrix} u(k-1) - u_{high}(0) \\ \vdots \\ u(k-1) - u_{high}(m-1) \\ u_{low}(0) - u(k-1) \\ \vdots \\ u_{low}(m-1) - u(k-1) \end{bmatrix} \tag{3.28}$$

where

$$I_L = \begin{bmatrix} I & 0 & \cdots & 0 \\ I & I & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ I & I & \cdots & I \end{bmatrix}. \tag{3.29}$$

3.4.2 Manipulated Variable Rate Constraints

Often MPC is used in a supervisory mode where there are limitations on the rate at which lower level controller setpoints are moved. These are enforced by adding constraints on the manipulated variable move sizes:

$$\begin{bmatrix} -I \\ I \end{bmatrix} \Delta\mathcal{U}(k) \geq \begin{bmatrix} -\Delta u_{max}(0) \\ \vdots \\ -\Delta u_{max}(m-1) \\ -\Delta u_{max}(0) \\ \vdots \\ -\Delta u_{max}(m-1) \end{bmatrix} \tag{3.30}$$

where $\Delta u_{max}(\ell) > 0$ is the possibly time varying bound on the magnitude of the moves.

3.4.3 Output Variable Constraints

The algorithm can make use of the output predictions (3.20) to anticipate future constraint violations.

$$\mathcal{Y}_{low} \leq \mathcal{Y}(k+1|k) \leq \mathcal{Y}_{high} \tag{3.31}$$

Substituting from (3.20) we obtain constraints on $\Delta\mathcal{U}(k)$

$$\begin{bmatrix} -\mathcal{S}^u \\ \mathcal{S}^u \end{bmatrix} \Delta\mathcal{U}(k) \geq \begin{bmatrix} \mathcal{M}\tilde{Y}(k) + S^d\Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k)) - \mathcal{Y}_{high} \\ -(\mathcal{M}\tilde{Y}(k) + S^d\Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k))) + \mathcal{Y}_{low} \end{bmatrix} \tag{3.32}$$

where

$$\mathcal{Y}_{low} = \begin{bmatrix} y_{low}(1) \\ y_{low}(2) \\ \vdots \\ y_{low}(p) \end{bmatrix}; \quad \mathcal{Y}_{high} = \begin{bmatrix} y_{high}(1) \\ y_{high}(2) \\ \vdots \\ y_{high}(p) \end{bmatrix}$$

are vectors of output constraint trajectories $y_{low}(\ell), y_{high}(\ell)$ over the horizon length p.

3.4.4 Combined Constraints

The manipulated variable constraints (3.28), manipulated variable rate constraints (3.30) and output variable constraints (3.32) can be combined into one convenient expression

$$\mathcal{C}^u\Delta\mathcal{U}(k) \geq \mathcal{C}(k+1|k) \tag{3.33}$$

where $\mathcal{C}^u$ combines all the matrices on the left hand side of the inequalities as follows:

$$\mathcal{C}^u = \begin{bmatrix} -I_L \\ I_L \\ -I \\ I \\ -\mathcal{S}^u \\ \mathcal{S}^u \end{bmatrix}. \tag{3.34}$$

The vector $\mathcal{C}(k+1|k)$ on the right hand side collects all the "error" vectors on the constraint equations as follows:

$$\mathcal{C}(k+1|k) = \begin{bmatrix} u(k-1) - u_{high}(0) \\ \vdots \\ u(k-1) - u_{high}(m-1) \\ u_{low}(0) - u(k-1) \\ \vdots \\ u_{low}(m-1) - u(k-1) \\ -\Delta u_{max}(0) \\ \vdots \\ -\Delta u_{max}(m-1) \\ -\Delta u_{max}(0) \\ \vdots \\ -\Delta u_{max}(m-1) \\ \mathcal{M}\tilde{Y}(k) + S^d \Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k)) - \mathcal{Y}_{high} \\ -(\mathcal{M}\tilde{Y}(k) + S^d \Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k))) + \mathcal{Y}_{low} \end{bmatrix} \tag{3.35}$$

3.5 Quadratic Programming Solution of the Control Problem

3.5.1 Quadratic Programs

Before the development of the DMC optimization problem, we introduce some basic concepts of nonlinear programming. In particular, the following formulation of a Quadratic Program (QP) is considered:

$$\begin{aligned} \min_x \quad & x^T H x - g^T x \\ \text{s.t.} \quad & Cx \geq c \end{aligned} \tag{3.36}$$

where

H is a symmetric matrix called the *Hessian* matrix;

g is the gradient vector;

C is the inequality constraint equation matrix; and

c is the inequality constraint equation vector.

This problem minimizes a quadratic objective in the decision variables x subject to a set of linear inequalities. In the absence of any constraints the solution of this optimization problem can be found analytically by computing the necessary conditions for optimality as follows:

$$\frac{d(x^T H x - g^T x)}{dx} = 2Hx - g = 0. \tag{3.37}$$

The second order derivative is

$$\frac{d^2(x^T H x - g^T x)}{dx^2} = 2H$$

which means that for an unconstrained minimum to exist, the Hessian must be positive semi-definite.

Note that in Section 2.7.4 the general Least Squares problem was formulated as an unconstrained minimization of a quadratic objective. In fact, the problem of minimizing the sum of squares of the residual of a set of linear equations

$$\rho = Ax - b$$

can be put in the form of this QP very simply since

$$\begin{aligned} \min_x \rho^T \rho &= \min_x (Ax-b)^T (Ax-b) \\ &= \min_x x^T A^T A x - 2b^T A x + b^T b \end{aligned}$$

Thus the QP is

$$\min_x x^T A^T A x - 2b^T A x$$

yielding

$$\begin{aligned} H &= A^T A \\ g &= 2A^T b. \end{aligned}$$

In order to obtain the unique unconstrained solution

$$x = \frac{1}{2} H^{-1} g$$

H must be positive definite, which is the same condition required in Section 2.7.4.

When the inequality constraints are added, strict positive definiteness of H is not required. For instance, for $H = 0$ the optimization problem becomes

$$\begin{aligned} \min_{x} \quad & -g^T x \\ \text{s.t.} \quad & Cx \geq c \end{aligned} \tag{3.38}$$

which is a *Linear Programming* (LP) problem. The solution of an LP will always lie at a constraint. This is not necessarily true of QP solutions. Although not a requirement, more efficient QP algorithms are available for problems with a positive definite H. For example, *parametric* QP algorithms employ the pre–inverted Hessian in its computations, thus reducing the computational requirements [**?**, **?**].

3.5.2 Formulation of Control Problem as a Quadratic Program

We make use of the prediction equation (3.20) to rewrite the objective

$$\min_{\Delta\mathcal{U}(k)} \sum_{\ell=1}^{p} \|\Gamma_\ell^y[y(k+\ell|k) - r(k+\ell)]\|^2 + \sum_{\ell=1}^{m} \|\Gamma_\ell^u[\Delta u(k+\ell-1)]\|^2. \tag{3.39}$$

and add the constraints (3.33) to obtain the optimization problem

$$\min_{\Delta\mathcal{U}(k)} \quad \left\{ \|\Gamma^y \left[\mathcal{Y}(k+1|k) - \mathcal{R}(k+1)\right]\|^2 + \|\Gamma^u \Delta\mathcal{U}(k)\|^2 \right\} \tag{3.40}$$

$$\begin{aligned} \text{s.t.} \quad & \mathcal{Y}(k+1|k) = \mathcal{M}\tilde{Y}(k) + S^d \Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k)) + \mathcal{S}^u \Delta\mathcal{U}(k) \\ & \mathcal{C}^u \Delta\mathcal{U}(k) \geq \mathcal{C}(k+1|k) \end{aligned} \tag{3.41}$$

where

$$\Gamma^u = \text{diag}\left\{\Gamma_1^u, \cdots, \Gamma_m^u\right\} \tag{3.42}$$

and

$$\Gamma^y = \text{diag}\left\{\Gamma_1^y, \cdots, \Gamma_p^y\right\} \tag{3.43}$$

are the weight matrices in block diagonal form, and

$$\mathcal{R}(k+1) = \begin{bmatrix} r(k+1) \\ r(k+2) \\ \vdots \\ r(k+p) \end{bmatrix} \tag{3.44}$$

is the vector of reference trajectories.

We can substitute the prediction equation into the objective function to obtain

$$\begin{aligned} & \|\Gamma^y \left[\mathcal{Y}(k+1|k) - \mathcal{R}(k+1)\right]\|^2 + \|\Gamma^u \Delta\mathcal{U}(k)\|^2 && (3.45) \\ = \; & \|\Gamma^y \left[\mathcal{S}^u \Delta\mathcal{U}(k) - E_p(k+1|k)\right]\|^2 + \|\Gamma^u \Delta\mathcal{U}(k)\|^2 && (3.46) \\ = \; & \Delta\mathcal{U}^T(k)(\mathcal{S}^{uT}\Gamma^{yT}\Gamma^y\mathcal{S}^u + \Gamma^{uT}\Gamma^u)\Delta\mathcal{U}(k) && (3.47) \\ & -2E_p(k+1|k)^T\Gamma^{yT}\Gamma^y\mathcal{S}^u\Delta\mathcal{U}(k) + E_p^T(k+1|k)\Gamma^{yT}\Gamma^y E_p(k+1|k) \end{aligned}$$

Here we have defined

$$\begin{aligned} E_p(k+1|k) &= \begin{bmatrix} e(k+1|k) \\ e(k+2|k) \\ \vdots \\ e(k+p|k) \end{bmatrix} \\ &\triangleq \mathcal{R}(k+1) - \left[\mathcal{M}\tilde{Y}(k) + S^d \Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k))\right], \end{aligned} \tag{3.48}$$

which is the measurement corrected vector of future output deviations from the reference trajectory (i.e., errors), assuming that all future control moves are zero. Note that this vector includes the effect of the measurable disturbances ($\mathcal{S}^d \Delta d(k)$) on the prediction.

The optimization problem with a quadratic objective and linear inequalities, which we have defined is a Quadratic Program. By converting to the standard QP formulation the DMC problem becomes[2]:

$$\begin{aligned} \min_{\Delta\mathcal{U}(k)} \quad & \Delta\mathcal{U}(k)^T \mathcal{H}^u \Delta\mathcal{U}(k) - \mathcal{G}(k+1|k)^T \Delta\mathcal{U}(k) \\ \text{s.t.} \quad & \mathcal{C}^u \Delta\mathcal{U}(k) \geq \mathcal{C}(k+1|k) \end{aligned} \tag{3.49}$$

where the Hessian of the QP is

$$\mathcal{H}^u = \mathcal{S}^{uT} \Gamma^{yT} \Gamma^y \mathcal{S}^u + \Gamma^{uT} \Gamma^u \tag{3.50}$$

and the gradient vector is

$$\mathcal{G}(k+1|k) = 2\mathcal{S}^{uT} \Gamma^{yT} \Gamma^y E_p(k+1|k). \tag{3.51}$$

3.6 Implementation

As explained in the introduction of this chapter the implementation of DMC is done in a *moving horizon* fashion. This implies that the Quadratic Program derived above will be solved at each controller execution time. Because of this feature, the algorithm can be configured on–line as required to take care of unexpected situations. For example, in case an actuator is lost during the implementation, the high and low constraint limits on that particular manipulated variable can be set to be equal. Then the MPC problem with the remaining manipulated variables is solved. Similarly, the weight parameters in the objective function can also be adjusted on-line, giving the user the ability to tune the control law. In this section we discuss the different implementation issues associated with DMC.

[2]The term $E_p^T(k+1|k)E_p(k+1|k)$ is independent of $\Delta\mathcal{U}(k)$ and can be removed from the objective function.

3.6.1 Moving Horizon Algorithm

The constrained MPC algorithm is implemented on-line as follows.

1. *Preparation.* Do not vary the manipulated variables for at least n time intervals ($\Delta u(-1) = \Delta u(-2) = \ldots = \Delta u(-n) = 0$) and assume the measured disturbances are zero ($\Delta d(-1) = \Delta d(-2) = \ldots = \Delta d(-n) = 0$) during that time. Then the system will be at rest at $k = 0$.

2. *Initialization* ($k = 0$). Measure the output $\hat{y}(0)$ and initialize the model prediction vector as[3]

$$\tilde{Y}(k) = \left[\underbrace{y_m(0)^T, y_m(0)^T, \ldots, y_m(0)^T}_{n}\right]^T \tag{3.52}$$

3. *State Update:* Set $k = k + 1$. Then, update the state according to

$$\tilde{Y}(k) = M \cdot \tilde{Y}(k-1) + S^u \Delta u(k-1) + S^d \Delta d(k-1) \tag{3.53}$$

   where the first element of $\tilde{Y}(k)$, $\tilde{y}(k|k)$, is the model prediction of the output $y_m(k)$ at time k.

4. *Obtain Measurements:* Obtain measurements $(y_m(k), \Delta d(k))$.

5. Compute the reference trajectory error vector

$$E_p(k+1|k) = \mathcal{R}(k+1) - \mathcal{M}\tilde{Y}(k) + S^d \Delta d(k) + \mathcal{I}_p(y_m(k) - \tilde{y}_0(k)) \tag{3.54}$$

6. Compute the QP gradient vector

$$\mathcal{G}(k+1|k) = \mathcal{S}^{uT}(\Gamma^y)^T \Gamma^y E_p(k+1|k). \tag{3.55}$$

7. Compute the constraint equations right hand side vector

[3] If (3.52) is used for intialization and changes in the past n inputs did actually occur, then the initial operation of the algorithm will not be smooth. The transfer from *manual* to *automatic* will introduce a disturbance; it will not be "bumpless".

$$\mathcal{C}(k+1|k) = \begin{bmatrix} u(k-1) - u_{high}(0) \\ \vdots \\ u(k-1) - u_{high}(m-1) \\ u_{low}(0) - u(k-1) \\ \vdots \\ u_{low}(m-1) - u(k-1) \\ -\Delta u_{max}(0) \\ \vdots \\ -\Delta u_{max}(m-1) \\ -\Delta u_{max}(0) \\ \vdots \\ -\Delta u_{max}(m-1) \\ -E_p(k+1|k) + \mathcal{R}(k+1) - \mathcal{Y}_{high} \\ E_p(k+1|k) - \mathcal{R}(k+1) + \mathcal{Y}_{low} \end{bmatrix} \tag{3.56}$$

8. Solve the QP

$$\begin{array}{rcl} \min\limits_{\Delta\mathcal{U}(k)} \ \frac{1}{2}\Delta\mathcal{U}(k)^T\mathcal{H}^u\Delta\mathcal{U}(k) & - & \mathcal{G}(k+1|k)^T\Delta\mathcal{U}(k) \\ \text{s.t. } \mathcal{C}^u\Delta\mathcal{U}(k) & \geq & \mathcal{C}(k+1|k) \end{array} \tag{3.57}$$

   and implement $\Delta u(k|k)$ as $\Delta u(k)$ on the plant.

9. Go to 3.

Note that the sequence of moves produced by the moving horizon implementation of the QP will be different from the sequence of moves $\Delta\mathcal{U}(k)$.

Example (plot first step solution)

3.6.2 Solving the QP

In a moving horizon framework the QP in (3.57) is solved at each controller execution time after a new prediction is obtained. The only time varying elements in this problem are the vectors $E_p(k+1|k)$ (or equivalently $\mathcal{G}(k+1|k)$) and $\mathcal{C}(k+1|k)$. That is, the Hessian $\mathcal{H}^u$ of the QP remains constant for all executions. In that case, as explained above a parametric QP algorithm which employs the pre–inverted Hessian in its computations is preferable in order to reduce on-line computation effort. Note that in the unconstrained case this is equivalent to the off–line computation of K_{MPC}. Of course, in case either Γ^y or Γ^u (or the step response coefficients) need to be updated, or the model's step response coefficients have changed, the Hessian must be recomputed and inverted in background mode in order not to increase the on–line computational requirements.

QP is a *convex* program and therefore is fundamentally tractable, meaning a global optimal solution within a specified tolerance can be assured. Though not extensively as LPs, QPs have been well studied and reliable algorithms have been developed and coded. General-purpose QP solvers like *QPSOL* are readily available but use of tailored algorithms that take advantage of specific problem structures can offer significant computational savings.

The conventional approach for solving QPs is the so called *Active Set* method. In this method, one initiates the search by assuming a set of active constraints. For an assumed active set, one can easily solve the resulting least squares problem (where the active constraints are treated as equality constraints) through the use of Lagrange multiplier. In general, the active set one starts out with will not be the correct one. Through the use of the Karush-Kuhn-Tucker (KKT) condition[4], one can modify the active set iteratively until the correction is found. Most active set algorithms are *feasible path* algorithms, in which the constraints must be met at all times. Hence, the number of constraints can has a significant effect on the computational time.

More recently, a promising new approach called the *Interior Point (IP)* method has been getting a lot of attention. The idea of the IP method is to "trap" the solution within the feasible region by including a so called "barrier" function in the objective function. With the modified objective function, the Newton iteration is applied to find the solution. Though originally developed for LPs, the IP method can be readily generalized to QPs and other more general constrained optimization problems. Even though not formally proven, it has been observed empirically that the Newton iteration converges within 5-50 steps. Significant work has been carried out in using this solution approach for solving QPs that arise in MPC, but details are out of the scope of this book; for interested readers, we give some references at the end of the chapter.

Computational properties of QPs vary with problems. As the number of constraints increase, more iterations are generally required to find the QP solution, and therefore the solution time increases. This may have an impact on the minimum control execution time possible. Also, note that the dimension of the QP (that is, the number of degrees of freedom $m \cdot n_u$) influences the execution time proportionately.

Storage requirements are also affected directly by the number of degrees of freedom and the number of projections $n \cdot n_y$. For example, the Hessian size increases quadratically with the number of degrees of freedom. Also, because of the prediction algorithm, $\tilde{Y}(k)$ must be stored for use in the next controller execution (both $E_p(k+1|k)$ and $\mathcal{C}(k+1|k)$ can be computed from $\tilde{Y}(k)$).

[4]The KKT condition is a necessary condition for the solution to a general constrained optimization problem. For QP, it is a necessary and sufficient condition.

3.6.3 Proper Constraint Formulation

Many engineering control objectives are stated in the form of constraints. Therefore it is very tempting to translate them into linear inequalities and to include them in the QP control problem formulation. In this section we want to demonstrate that constraints make it very difficult to predict the behavior of the control algorithm under real operating conditions. Therefore, they should be used only when necessary and then only with great caution.

First of all constraints tend to greatly increase the time needed to solve the QP. Thus, we should introduce them sparingly. For example, if we wish an output constraint to be satisfied over the whole future horizon, we may want to state it as a linear inequality only at *selected* future sampling times rather than at *all* future sampling times. Unless we are dealing with a highly oscillatory system, a few output constraints at the beginning and one at the end of the horizon should keep the output more or less inside the constraints throughout the horizon. Note that even when constraint violations occur in the prediction this does not imply constraint violations in the actual implementation because of the moving horizon policy. The future constraints serve only to prevent the present control move from being short-sighted.

Output constraints can also lead to an "infeasibility." A QP is *infeasible* if there does not exist any value of the vector of independent variables (the future manipulated variable move $\Delta\mathcal{U}(k)$) which satisfies all the constraints – regardless of the value of the objective function. Physically this situation can arise when there are output constraints to be met but the manipulated variables are not sufficiently effective – either because they are constrained or because there is dead time in the system which delays their effect. Needless to say, provisions must be built into the on-line algorithm such that an infeasibility never occurs.

Mathematically an infeasibility can only occur when the right hand side of the output constraint equations is positive. This implies that a nonzero move *must* be made in order to satisfy the constraint equations. Otherwise, infeasibility is not an issue since $\Delta\mathcal{U}(k) = 0$ is feasible.

A simple example of infeasibility arises is the case of deadtimes in the response. For illustration, assume a SISO system with θ units of deadtime. The output constraint equations for this system will look like:

$$\begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & 0 \\ -S^u_{\theta+1} & 0 & \cdots & 0 \\ -S^u_{\theta+2} & -S^u_{\theta+1} & \cdots & 0 \\ \vdots & & & \vdots \end{bmatrix} \Delta\mathcal{U}(k) \geq \begin{bmatrix} c(k+1|k) \\ \vdots \\ c(k+\theta|k) \\ c(k+\theta+1|k) \\ c(k+\theta+2|k) \\ \vdots \end{bmatrix}$$

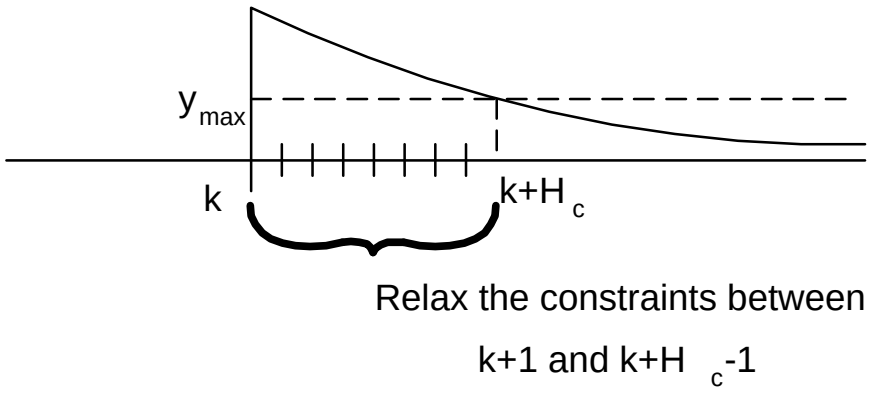

Figure 3.3: Relaxing the constraints

Positive elements $c(k+1|k), \cdots, c(k+\theta|k)$ indicate that a violation is projected unless the manipulated variables are changed ($\Delta\mathcal{U}(k) \neq 0$). Since the corresponding coefficients in the left hand side matrix are zero, the inequalities cannot be satisfied and the QP is infeasible. Of course, this problem can be removed by simply not including these initial θ inequalities in the QP.

Because inequalities are dealt with *exactly* by the QP, the corrective action against a projected violation is equivalent to that generated by a very tightly tuned controller. As a result, the moves produced by the QP to correct for violations may be undesirably severe (even when feasible). Both infeasibilities and severe moves can be dealt with in various ways.

One way is to include is a *constraint window* on the output constraints similar to what we suggested above for computational savings. For each output a time $k+H_c$ in the future is chosen at which constraint violations will start to be checked (Fig. 3.3). For the illustration above, this time should be picked to be at least equal to $\theta+1$. This allows the algorithm to check for violations after the effects of deadtimes and inverse responses have passed. For each situation there is a minimal value of H_c necessary for feasibility. If this minimal value is chosen large, constraint violations may occur over a significant period of time. In many cases, if a larger value of H_c is chosen, smaller constraint violations may occur over a longer time interval. Thus, there is a trade-off between magnitude and duration of constraint violation.

In general, it is difficult to select a value of H_c for each constrained output such that the proper compromise is achieved. Furthermore, in multivariable cases, constraints may need to be relaxed according to the priorities of the constrained variables. The selection of constraint windows is greatly complicated by the fact that appropriate amount and location for relaxation are usually time-dependent due to varying disturbances and occurrences of actuator and sensor failures. Therefore it is usually preferred to soften the constraint by

adding a slack variable ϵ and penalizing this violation through an additional term in the objective function.

$$\min_{\epsilon,\Delta\mathcal{U}(k)}[\text{Usual Objective}] + \lambda\epsilon^2$$

$$y_{\min} - \epsilon \leq y(k+\ell|k) \leq y_{\max} + \epsilon$$
plus other constraints

The optimization seeks a compromise between minimizing the original performance objective and minimizing the constraint violations expressed by ϵ^2. The parameter λ determines the relative importance of the two terms. The degree of constraint violation can be fine tuned arbitrarily by introducing a separate slack variable ϵ for each output and time step, and associating with it a separate penalty parameter λ.

Finally we must realize that while unconstrained MPC is a form of *linear* feedback control, constrained MPC is a *nonlinear* control algorithm. Thus, its behavior for small deviations can be drastically different from that for large deviations. This may be surprising and undesirable and is usually very difficult to analyze a priori.

EX – Example from MPC course. **INCLUDE!**

3.6.4 Choice of Horizon Length

On one hand, the prediction horizon p and the control horizon m should be kept short to reduce the computational effort; on the other hand, they should be made long to prevent short-sighted control policies. Making m short is generally conservative because we are imposing constraints (forcing the control to be constant after m steps) which do not exist in the actual implementation because of the moving horizon policy. Therefore a small m will tend to give rise to a cautious control action.

Choosing p small is "short-sighted" and will generally lead to an aggressive control action. If constraint violations are checked only over a small control horizon p this policy may lead the system into a "dead alley" from which it can escape only with difficulty, i.e., only with large constraint violations and/or large manipulated variable moves.

When p and m are infinity and when there are no disturbance changes and unknown inputs, the sequence of control moves determined at time k is the same sequence which is realized through the moving horizon policy. In this sense our control actions are truly optimal. When the horizon lengths are shortened, then the sequence of moves determined by the optimizer and the sequence of moves actually implemented on the system will become increasingly different. Thus the short time objective which is optimized will have less and less to do with the actual value of the objective realized when the moving horizon control is implemented. This may be undesirable.

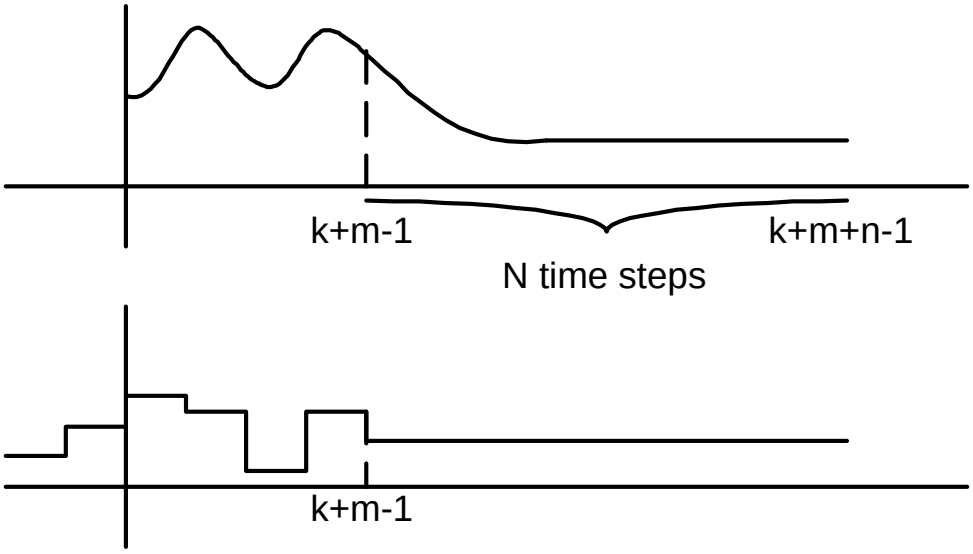

Figure 3.4: Choosing the horizon

In general, we should try to choose a small m to keep the computational effort manageable, but large enough to give us a sufficient number of degrees of freedom. We should choose p as large as possible, possibly ∞, to completely capture the consequences of the control actions. This is possible in several ways. Because an FIR system will settle after $m+n$ steps, choosing a horizon $p = m + n$ is a sensible choice used in many commercial systems (Fig. 3.4). Instead or in addition we can impose a large output penalty at the end of the prediction horizon forcing the system effectively to settle to zero at the end of the horizon. Then, with $p = m+n$, the error after $m+n$ is essentially zero and there is little difference between the finite and the infinite horizon objective.

3.6.5 Input Blocking

As said, use of a large control horizon is generally preferred from the viewpoint of performance but available computational resource may limit its size. One way to relax this limit is through a procedure called *Blocking*, which allows to the user to "block out" the input moves at selected locations from the calculation by setting them to zero *a priori*. Result is a reduction in the number of input moves that need to be computed through the optimization, hopefully without a significant sacrifice in the solution quality. Obviously, judicious selection of blocking locations is critical for achieving the intended effect. The selection is done mostly on an *ad hoc* basis, though there are some qualitative rules like blocking less of the immediate moves and more of the distant ones.

At a more general level, blocking can be expressed as follows:

$$\Delta\mathcal{U} = \mathbf{B}\Delta\mathcal{U}^b \tag{3.58}$$

where $\Delta\mathcal{U}^b$ represent the reduced input parameters to be calculated through the optimization. $\mathbf{B}$ is the blocking matrix that needs to be designed for a good performance. Typically, the rows of $\mathbf{B}$ corresponding to the blocked moves would contain all zeros. In general, columns of $\mathbf{B}$ can be designed to represent different basis in the input space. Note that dimension of $\mathcal{U}^b$, which is less than that of $\mathcal{U}$, must also be determined in the design.

3.6.6 Filtering of the Feedback Signal

In practice, feedback measurements can contain significant noise and other fast-varying disturbances. Since in DMC the effect of unmeasured disturbances is projected as a constant bias in the prediction, the high-frequency contents of a feedback signal must be filtered out in order to obtain a meaningful long-term prediction. For this, one can pass the feedback signal through a low-pass filter of some sort, perhaps a first- or second-order filter, before putting it into the prediction equation. Use of state estimation, discussed in a later chapter of this book, allows one to model the statistical characteristics of disturbances and noise and perform the filtering in an optimal manner.

3.7 Examples

INCLUDE!
Effect of constraints on response – PC examples of slowdown with IMC
Stability-contrast with unconstrained case (EZ)
Example from MPC course demonstrating positive effect of constraint.

3.8 Features Found in Other Algorithms

What we just covered is the basic form of a multivariable control algorithm called *Dynamic Matrix Control* (DMC), which was one of the first MPC algorithms applied to industrial processes with success. The original DMC algorithm did not use QP to handle constraints; instead, it added an extra output to the prediction to drive the input back to the feasible region whenever a predicted future input came close to a constraint. This was somewhat *ad hoc* and it wasn't until the 80s that engineers at Shell Oil proposed the use of QP to handle input and output constraints explicitly and rigorously. They called this modified version QDMC. Currently, this basic form of DMC is still used in a commercial package called *DMC-PLUS*, which is marketed by Aspen Technology.

A figure similar to FIGURE7 in Qin and Badgwell

Figure 3.5: Output Penalties Used in Various Formulations

Besides DMC and QDMC, there are several other MPC algorithms that have seen, and are still seeing, extensive use in practice. These include Model Predictive Heuristic Control (MPHC), which led to popular commercial algorithms like IDCOM and SMC-IDCOM marketed by Setpoint (now Aspen Technology) and Hierarchical Constraint Control (HIECON) and Predictive Functional Control (PFC) marketed by Adersa; Predictive Control Technology (PCT), which was marketed by Profimatics (now Honeywell); and more recent Robust Model Predictive Control Technology (RMPCT), which is currently being marketed by Honeywell. These algorithms share same fundamentals but differ in details of implementation. Rather than elaborating on the details of each algorithm, we will touch upon some popular features not seen in the basic DMC/QDMC method.

3.8.1 Reference Trajectories

In DMC, output deviation from the desired setpoint is penalized in the optimization. Other algorithms like IDCOM, HIECON, and PFC let the user specify not only *where* the output should go but also *how*. For this, a *reference trajectory* is introduced for each controlled variable (CV), which is typically defined as a first-order path from the current output value to the desired setpoint. The time constant of the path can be adjusted according to the speed of the desired closed-loop response. This is displayed in Figure **??**.

Reference trajectories provide an intuitive way to control the aggressiveness of control, which would is adjusted through the weighting matrix for the input move penalty term in DMC. One could argue that the controller's aggressiveness is more conveniently tuned by specifying the speed of output response rather than through input weight parameters, whose effects on the speed of response is highly system-dependent.

3.8.2 Coincidence Points

Some commercial algorithms like IDCOM and PFC allowed the option of penalizing the output error only at a few chosen points in the prediction horizon called *coincidence points*. This is motivated primarily by reduction in computation it brings. When the number of input moves has to be kept small (in order to keep the computational burden low), use of a large prediction horizon, which is sometimes necessary due to large inverse responses, and long dynamics, results in a sluggish control behavior. This problem can be obviated by penalizing output deviation only at a few carefully selected points. At the

extreme, one could ask the output to match the reference trajectory value at a single time point, which can be achieved with a single control move. Such formulation was used, for example, in IDCOM-M, an offspring of the original IDCOM algorithm, marketed by Setpoint.

Clearly, the choice of coincidence points is critical for performance, especially when the number of points used is small. Though some guidelines exist on choosing these points, there is no systematic method for the selection. Because the response time of different outputs can vary significantly, coincidence points are usually defined separately for each output.

3.8.3 The Funnel Approach

The RMPCT algorithm differs from other MPC algorithms in that it attempts to keep each controlled output within a user-specified zone called *funnel*, rather than to keep it on a specific reference trajectory. The typical shape of a funnel is displayed in Figure **??**. The user sets the maximum and minimum limits and also the slope of the funnel through a parameter called 'performance ratio,' which is the desired time to return to the limit zone divided by the open-loop response time. The gap between the maximum and minimum can be closed for exact setpoint control, or left open for range control.

The algorithm solves the following quadratic program at each time:

$$\min_{y^r,u} \sum_{i=1}^{p} \|y(k+i|k) - y^r(k+i|k)\|_Q^2 + \sum_{j=0}^{m-1} \|\Delta u(k+j|k)\|_R^2 \tag{3.59}$$

or

$$\min_{y^r,u} \sum_{i=1}^{p} \|y(k+i|k) - y^r(k+i|k)\|_Q^2 + \sum_{j=0}^{m-1} \|u(k+j|k) - u^r\|_R^2 \tag{3.60}$$

subject to usual constraints plus the funnel constraint

$$y^f_{min}(k+i|k) \le y^r(k+i|k) \le y^f_{max}(k+i|k), \quad 1 \le i \le p \tag{3.61}$$

where $y^f_{min}(k+i|k)$ and $y^f_{min}(k+i|k)$ represent the upper and lower limit values of the funnel for $k+i$ in the prediction horizon as specified at time k. u^r is the desired settling value for the input. Notice that the reference trajectory y^r is a free parameter, which is optimized to lie within the funnel. Typically, $Q \gg R$ in order to keep the outputs within the funnel as much as possible. Then one can think of the above as a multi-objective optimization, in which the primary objective is to minimize the funnel constraint violation by the output and the secondary objective is to minimize the size of input movement (or input deviation from the desired settling value in the case of (3.60)). In this case, as long as there exists an input trajectory that keeps the output within the

INCLUDE!

Table 3.1: Optimization Resulting from Use of Different Spatial and Temporal Norms

funnel, the first penalty term will be made exactly zero. Typically, there will be an infinite number of solutions that achieve this, leading to a 'degenerate' QP. The algorithm thus finds the minimum norm solution, which corresponds to the least amount of input adjustment – hence the name 'Robust' MPCT . However, if there is no input that can keep the output within the funnel, the first term will be the primary factor that determines the input.

The use of funnel is motivated by the fact that, in multivariable systems, the shape of desirable trajectories for outputs are not always clear due to the system interaction. Thus, it is argued that an attractive formulation is to let the user specify an acceptable dynamic zone for each output as a funnel and then find the minimum size input moves (or inputs with minimum deviation from their desired values) that keep the outputs within the zone – or, if not possible, minimize the extent of violation.

3.8.4 Use of Other Norms

In defining the objective function, use of norms other than 2-norm is certainly possible. For example, the possibility of using 1-norm (sum of absolute values) has been explored to a great extent. Use of infinity norm has also been investigated with the aim of minimizing worst-case deviation over time. In both cases, one gets a Linear Program, for which plethora of theories and software exist due to its significance in economics. However, one difficulty with these formulations is in tuning. This is because the solution of a LP lies at the intersection of binding constraints and it can switch abruptly from one vertex to another as one varies tuning parameters (such as the input weight parameters). The solution behavior of a QP is much smoother and therefore it is a preferred formulation for control.

Table 3.1 summarizes the optimization that results from various combinations of spatial and temporal norms.

3.8.5 Input Parameterization

In some algorithms like PFC, the input trajectory can be parameterized using continuous basis functions like polynomials. This can be useful if the objective is to follow *smooth* setpoint trajectories precisely, such as in mechanical servo applications, and the sampling time cannot be made sufficiently small to allow this with piecewise constant inputs.

In other commercial algorithms like HIECON and IDCOM-M, only a single

control move is calculated,which would correspond to $m = 1$ in DMC. With this setting, the calculation is greatly simplified. On the other hand, use of $m = 1$ in DMC would limit the closed-loop performance in general. These algorithms get around this problem by using a single coincidence point, at which the output is asked to match the reference value exactly.

3.8.6 Model Conditioning

In multivariable plants, two or more outputs can behave very similarly in response to all the inputs. This phenomenon is referred to as 'ill-conditioning' and is reflected by a gain matrix that is nearly singular. An implication is that it can be very difficult to control these outputs independently with the inputs, as it will require an excessive amount of input movement in order to move the outputs in certain directions. Using an ill-conditioned process model for control calculation is not recommended as it can lead to numerical problems (*e.g.*, inversion of a nearly singular matrix) and also excessive input movements and/or even an instability.

Even though one would check the conditioning of the model at the design stage, because control structure can change due to constraints and failures of sensors and actuators, one must make sure at each execution time that an ill-conditioned process model is not directly inverted in the input calculation.

In DMC, direct inversion of an ill-conditioned process model can be circumvented by including a substantive input move penalty term, which effectively increases the magnitudes of the diagonal elements of the dynamic matrix that is inverted during the least squares calculation.

In other algorithms that do not include an input move penalty in the objective function, ill-conditioning must be checked at each execution time. In RMPCT, this is done through a method called *Singular Value Threshholding*, where a procedure called 'singular value decomposition' is performed on the gain matrix to determine those CV directions for which the gain is too low for any effective control at all. Those directions with singular values lower than a threshhold value are given up for control and only the remaining 'high-gain' directions are controlled. SMC-IDCOM addresses this based on the user-defined ranking of CVs. Here, whenever an ill-conditioning is detected, CVs are dropped from the control calculation in the order of their ranks, starting from the one with the least assigned priority, until the condition number improves to an acceptable level. When two CVs are seen to behave very similarly, the user can rank the less important CV with a very low priority. Even though the control on the dropped CV is given up, it is hoped that it would be controlled indirectly since it behaves similarly to the other high ranked CV.

3.8.7 Prioritization of CVs and MVs

In most practical control problems, it is not possible to satisfy all constraints and also drive all outputs and inputs to their desired resting values. Hence, priorities need to be assigned to express their relative importance. In DMC, these priorities are determined through weight parameters, which enter into the various quadratic penalty terms in the objective function. For large, complex problems, determining proper weights that lead to an intended behavior can be a daunting task. Even if a set of weights consistent with the control specification is found, the weights can differ vastly in magnitude from one another, causing a numerical conditioning problem.

Algorithms like HIECON and SMC-IDCOM attempt to address this difficulty by letting the user *rank* various objectives in the order of their importance. For example, constraint satisfaction may be the most critical aspect, which must be taken care of before satisfying other objectives. Also driving the CVs to their desired setpoints may be more important than driving the MVs to their most economic values. In these algorithms, an optimization would be solved with the most important objective first and then remaining degrees of freedom would be used to address the other objectives in the order of priority. These algorithms also allow the user to rank each CV and MV according to its priority. Hence, for constraint softening, one may specify the order in which constraints for various CVs must be relaxed. Also, in setpoint tracking, one can prioritize the CVs so that CVs with higher ranks are driven to their setpoints before those with lower ranks are considered.

3.9 Some Possible Enhancements to DMC

3.9.1 Closed-Loop Update of Model State

In the conventional DMC formulation, the model is run in *open-loop* (*i.e.*, by entering known inputs only); the model prediction error is added to the prediction equation, typically as a constant bias term. Hence, the model states do not contain any effect of unmeasured inputs, or more precisely, their estimates based on the feedback measurements. Another possibility for using the feedback measurements is to correct the model state directly at each time so that the model state will also hold information about relevant effects of unmeasured inputs. Since this information will be provided indirectly through feedback measurements, the state can be considered as a holder of relevant past input and measured output information in this context. The difference in the two approaches are displayed graphically in Figure 3.6. Since the model state gets continually corrected by the measurements in the closed-loop update approach, it is not necessary to add another correction term in the prediction stage.

Figure 3.6: Open-Loop vs. Closed-Loop Update of the Model and Corresponding Prediction

For the step response model, we can modify the update equation to add the step for feedback measurement based correction of the model state:

Model Prediction:

$$\tilde{Y}(k|k-1) = M \cdot \tilde{Y}(k-1|k-1) + S\Delta v(k-1) \tag{3.62}$$

where $\tilde{Y}(\cdot|k-1)$ denotes the estimate of $\tilde{Y}(\cdot)$ obtained at $k-1$, taking into account all measurement information up to $k-1$. Recall that this is th esole step we took to update the model state in the previous formulation. Here we postulate to correct the model prediction $\tilde{Y}(k|k-1)$ based on the difference between the measurement $y_m(k)$ at time k and the model prediction $\tilde{y}_0(k|k-1)$, the first output vector appearing in $\tilde{Y}(k|k-1)$, for this time step.

Correction:

$$Y(k|k) = Y(k|k-1) + \mathbf{K}(y_m(k) - \tilde{y}_0(k|k-1)) \tag{3.63}$$

The matrix $\mathbf{K}$ is referred to as *observer gain*. To make the prediction equivalent to the earlier prediction where we added a constant bias term of size $(y_m(k) - \tilde{y}_0(k|k-1))$ to the prediction equation, we should choose the observer gain as

$$\mathbf{K} = \mathbf{I} \quad = \quad \left.\begin{bmatrix} I \\ I \\ \vdots \\ I \end{bmatrix}\right\} \; n \tag{3.64}$$

Element by element this equation is

$$\begin{aligned}
\tilde{y}_0(k|k) &= \tilde{y}_0(k|k-1) + [y_m(k) - \tilde{y}_0] = \hat{y}(k) \\
\tilde{y}_1(k|k) &= \tilde{y}_1(k|k-1) + [y_m(k) - \tilde{y}_0(k|k-1)] \\
\vdots \quad &\quad \vdots \quad \vdots \\
\tilde{y}_{n-1}(k|k) &= \tilde{y}_{n-1}(k|k-1) + [y_m(k) - \tilde{y}_0(k|k-1)]
\end{aligned}$$

Note from the first equation that the estimate of the current output is set equal to the measurement. As before, we have added the same correction term $[\cdot]$ to all future predicted outputs, $\tilde{y}_1(k|k-1), \ldots, \tilde{y}_{n-1}(k|k-1)$ and therefore $y(k+1|k-1), \ldots, y(k+n-1|k-1)$, interpreting it as a bias term which is determined based on the present measurement and remains constant.

Substituting (3.62) into (3.63) we obtain the *state estimator*

$$\begin{aligned}
\tilde{Y}(k|k) &= M \cdot \tilde{Y}(k-1|k-1) + S\Delta v(k-1) \\
&+ \mathbf{I}\left[y_m(k) - N(M \cdot \tilde{Y}(k-1|k-1) + S\Delta v(k-1))\right]
\end{aligned} \tag{3.65}$$

INCLUDE!

Figure 3.7: Step Response of An Integrating System

where

$$N = \left[\underbrace{I\ 0\ 0\ \ldots\ 0}_{n}\right]$$

which allows one to compute the current state estimate $\tilde{Y}(k|k)$ based on the previous estimate $\tilde{Y}(k-1|k-1)$, the previous input move $\Delta v(k-1)$ and the current measurement $y_m(k)$. **I** is referred to as the observer gain.

An advantage of the above formulation is that substantial theories exist that enable us to design **K** optimally so as to account for information we may have about the statistical characteristics of disturbances and measurement noise. Another advantage is that it can be generalized to handle systems with unstable dynamics. We note that running an unstable system model in open loop would lead to an "OVERFLOW" in the computer. The noise filtering issue will be discussed in a simplified way below. The more rigorous general treatment of the design of **K** and its accompanying properties will be given in Chapter **??** in the advanced part of the book.

3.9.2 Integrating System

Integrating dynamics are common in chemical processes. In systems with integrating dynamics, a step response never settles down to a constant value, and therefore, the standard finite step response based model description cannot be used. For these systems, an equivalent assumption to a finite settling time is that, after n steps, all stable dynamics die out and the responses of all the outputs of integrating dynamics becomes pure ramps, as shown in Figure 3.9.2. The step response of such a system can be represented by

$$\{S_1, S_2, \ldots, \ldots, S_n, S_n + (S_n - S_{n-1}), S_n + 2(S_n - S_{n-1}), \ldots, \ldots\} \quad (3.66)$$

We can define the system state as we did for stable systems:

$$\tilde{Y}(k) = \left[\tilde{y}_0(k)^T,\ \tilde{y}_1(k)^T,\ \ldots, \tilde{y}_{n-1}(k)^T\right] k^T \quad (3.67)$$

Then, we can represent the state transition from one sample time to the next as

$$\tilde{Y}(k+1) = M^I \tilde{Y}(k) + S\Delta v(k) \quad (3.68)$$

where $S = \begin{bmatrix} S_1 & \cdots & S_n \end{bmatrix}^T$ as before and

$$M^I = \begin{bmatrix} 0 & 1 & 0 & \dots & \dots & 0 & 0 \\ 0 & 0 & I & 0 & \dots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \dots & \dots & \dots & 0 & I \\ 0 & 0 & \dots & \dots & \dots & -I & 2I \end{bmatrix} \Bigg\} \; n \tag{3.69}$$

where M^I represents essentially the same shift operation as before except the way the last set of elements are constructed. Note that the assumption of pure linear rise after n steps in the step response translates into the transition equation of

$$\tilde{y}_{n-1}(k+1) = \tilde{y}_{n-1}(k) + (\tilde{y}_{n-1}(k) - \tilde{y}_{n-2}(k)) + S_n \Delta v(k) \tag{3.70}$$

One important difference in forming the prediction equation for integrating systems is that the effect of unmeasured disturbances should be extrapolated as a linear rise rather than a constant bias. Hence, one may modify the

prediction equation from before to

$$\mathcal{Y}(k+1|k) = \underbrace{\begin{bmatrix} \tilde{y}_1(k) \\ \tilde{y}_2(k) \\ \vdots \\ \vdots \\ \tilde{y}_p(k) \end{bmatrix}}_{\mathcal{M}^I \tilde{Y}(k)}$$

from the memory

$$+ \underbrace{\begin{bmatrix} S_1^d \\ S_2^d \\ \vdots \\ \vdots \\ S_p^d \end{bmatrix} \Delta d(k)}_{\mathcal{S}^d \Delta d(k)} + \underbrace{\begin{bmatrix} w_y(k|k) + (w_y(k|k) - w_y(k-1|k-1)) \\ w_y(k|k) + 2\,(w_y(k|k) - w_y(k-1|k-1)) \\ \vdots \\ \vdots \\ w_y(k|k) + p\,(w_y(k|k) - w_y(k-1|k-1)) \end{bmatrix}}_{\text{feedback term}}$$

feedforward term

$$+ \underbrace{\begin{bmatrix} S_1^u & 0 & \cdots & \cdots & 0 \\ S_2^u & S_1^u & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ S_m^u & S_{m-1}^u & \cdots & \cdots & S_1^u \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ S_p^u & S_{p-1}^u & \cdots & \cdots & S_{p-m+1}^u \end{bmatrix}}_{\mathcal{S}^u} \underbrace{\begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \vdots \\ \Delta u(k+m-1|k) \end{bmatrix}}_{\Delta \mathcal{U}(k)}$$

dynamic matrix — future input moves

(3.71)

where $w_y(k|k) = y_m(k) - \tilde{y}_0(k)$ representing the model prediction error. Notice that we have used two point linear extrapolation in projecting e into the future. In the above,

$$\mathcal{M}^I = \left.\begin{bmatrix} 0 & I & 0 & \dots & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \dots & \dots & \dots & 0 & I \\ 0 & 0 & \dots & \dots & \dots & -I & 2I \\ 0 & 0 & \dots & \dots & \dots & -2I & 3I \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}\right\} \; p \tag{3.72}$$

Here we assumed $p > n$. If not, one can simply take the first p rows.

However, there is a problem in using the above in practice: The open-loop model prediction of the output (terms like $\tilde{y}_i(k)$) will increase without

bound due to the integrators and eventually lead to an "OVERFLOW" in the control computer. This problem happens not because the physical system is actually going unstable but because the model states are not corrected by measurements to account for various unmeasured inputs. To circumvent this, it is necessary to update the model state directly using the measurements, as in the closed-loop model state update discussed earlier. Let us postulate a closed-loop update equation of the same form (3.62 and 3.63) as for stable systems

Model Prediction:

$$\tilde{Y}(k|k-1) = M^I \cdot \tilde{Y}(k-1|k-1) + S\Delta v(k-1) \tag{3.73}$$

Correction:

$$\tilde{Y}(k|k) = \tilde{Y}(k|k-1) + (\mathbf{I}+\mathbf{I}')(y_m(k) - \tilde{y}_0(k|k-1)) \tag{3.74}$$

where

$$\mathbf{I}' = \begin{bmatrix} 0 \\ I \\ 2I \\ \vdots \\ (n-1)I \end{bmatrix} \tag{3.75}$$

and hence

$$\mathbf{I}+\mathbf{I}' \quad = \quad \begin{bmatrix} I \\ 2I \\ 3I \\ \vdots \\ nI \end{bmatrix} \tag{3.76}$$

Here $\mathbf{I}+\mathbf{I}'$ is referred to as the observer gain. The form of the correction term can be motivated by examining the special situation depicted in Fig. 3.8 (the situation would be caused, for example, by an unknown step disturbance entering the integrator). In this case, estimates $\tilde{y}_\ell(k|k)$ for $\ell \geq k$ will be exactly equal to the true process outputs $y_m(k+\ell)$ if the observer (3.73,3.74) is used.

In general, there will be both stable and integrating outputs and the observer gain has to be chosen either according to (3.64) or (3.76).

3.9.3 Noise Filter

Let us take another look at the correction steps for both stable systems

$$\tilde{Y}(k|k) = \tilde{Y}(k|k-1) + \mathbf{I}(y_m(k) - \tilde{y}_0(k|k-1)) \tag{3.63}$$

and integrating systems

$$\tilde{Y}(k|k) = \tilde{Y}(k|k-1) + (\mathbf{I}+\mathbf{I}')(y_m(k) - \tilde{y}_0(k|k-1)) \tag{3.74}$$

Figure 3.8: Motivation for observer correction term ($\Delta \stackrel{\Delta}{=} y_m(k) - \tilde{y}_0(k|k-1)$).

In both cases, we determine the difference between the model prediction $\tilde{y}_0(k|k-1)$ and the measurement $y_m(k)$ and add it either as a constant bias or a "ramp bias" to the model prediction $\tilde{Y}(k|k-1)$ to obtain the corrected prediction $\tilde{Y}(k|k)$. This is justifiable, for example, if the difference is solely due to a constant disturbance effect. It could, however, be solely due to measurement noise, in which case we would not want to correct the model prediction at all. In general, disturbances, model error, and measurement noise will contribute to the difference, in which case a more cautious correction than implied by (3.63) and (3.74) will be appropriate. This can be achieved by filtering the correction term in (3.63):

$$\tilde{Y}(k|k) = \tilde{Y}(k|k-1) + \mathbf{I}F[y_m(k) - \tilde{y}_0(k|k-1)] \tag{3.77}$$

where F is a diagonal matrix

$$F = \text{diag}\left\{f_1, f_2, \ldots, f_{n_y}\right\} \tag{3.78}$$

and

$$0 < f_\ell \leq 1 \tag{3.79}$$

Thus, rather than correct the model prediction by the full error $[y_m(k) - \tilde{y}_0(k|k-1)]$ one takes a more cautious approach and utilizes only a fraction f_ℓ. The larger the measurement noise associated with output y_ℓ, the smaller f_ℓ should be chosen.

To understand better the effect of this noise filter on control performance, assume that the output suddenly changes to a constant value ("output dis-

turbance") $y_m(0) = \bar{y}$ and that neither the disturbance nor the manipulated variables change ($\Delta d(k) = 0, \Delta u(k) = 0$). Then we find from (3.62)

Model Prediction:

$$\tilde{Y}(k|k-1) = M\tilde{Y}(k-1|k-1) \tag{3.80}$$

and from (3.77)

$$\begin{aligned} \tilde{Y}(k|k) &= \tilde{Y}(k|k-1) + \mathbf{I}F(y_m(k) - y(k|k-1)) \\ &= M\tilde{Y}(k-1|k-1) + \mathbf{I}Fy_m(k) - \mathbf{I}FNM\tilde{Y}(k-1|k-1) \\ &= (I - \mathbf{I}FN)M\tilde{Y}(k-1|k-1) + \mathbf{I}Fy_m(k) \end{aligned} \tag{3.81}$$

where

$$N = \left[\underbrace{I\ 0\ \dots\ 0}_{n}\right] \tag{3.82}$$

The form suggests — and we shall rigorously **prove this in the advanced part** — that $\tilde{Y}(k|k)$ corresponds to $y_m(k)$ passed through a first-order filter. Indeed, the estimate $\tilde{Y}(k|k)$ approaches the true value $\hat{y}$ with the filter time constant:

$$\text{stable system} : -T/\ln(1 - f_\ell) \tag{3.83}$$

where T is the sample time. In principle, for integrating systems, we could also detune the observer gain $\mathbf{I} + \mathbf{I}'$ (3.76) by post-multiplying it by a filter matrix F. However, this choice tends to lead to a highly oscillatory observer response and is therefore undesirable. **As we will show in the advanced part**, it is more desirable to introduce the filter in the following manner into (3.74):

$$\tilde{Y}(k|k) = \tilde{Y}(k|k-1) + (\mathbf{I} \cdot F + \mathbf{I}' \cdot F')(y_m(k) - \tilde{y}_0(k|k-1)) \tag{3.84}$$

where F was defined as above (3.78) and

$$F' = \text{diag}\left\{f'_1, f'_2, \dots, f'_{n_y}\right\} \tag{3.85}$$

and

$$f'_i = \frac{f_i^2}{2 - f_i} \tag{3.86}$$

Thus, for both stable systems (3.77) and integrating systems (3.84), we have a single tuning parameter $0 < f_\ell \leq 1$ for each output. The noise filtering action decreases with increasing f_ℓ. For $f_\ell = 1$, measurement noise is not filtered at all and we recover (3.63) and (3.74).

In general, there will be both stable and integrating outputs and the filter gain is chosen for each output either as suggested in (3.77) or in (3.84). This leads to the following correction expression with the general filter gain K_F:

$$\tilde{Y}(k|k) = \tilde{Y}k|k-1) + K_F(y_m(k) - \tilde{y}_0(k|k-1)) \tag{3.87}$$

3.9.4 Bi-Level Optimization

The MPC calculation can be split into two parts for an added flexibility. First a local *steady-state* optimization can be performed to obtain obtain target values for each input and output. This can be followed by a *dynamic* optimization to determine the most desirable dynamic trajectory to these target values. Even though the local steady-state optimization can be based on an economic index, it does not replace the more comprehensive nonlinear optimization that often runs above the MPC layer – at a much more slower rate – in order to provide an optimal range of inputs and outputs for the plant condition experienced during a particular optimization cycle. The local optimization performed in MPC is based on a linear steady state model, which may be obtained by linearizing a nonlinear model or simply the steady-state version of the step response model used in the dynamic optimization.

The reason for running the local optimization may vary. For example, one may want to perform an economic optimization at a higher frequency to account for local disturbances. Even if there is no economic objective in the given control problem, the steady-state optimization can be helpful to determine best feasible target values for CVs and the corresponding MV settling values.

The two-stage optimization can be formulated as below:

- Step 1: *Steady-State Optimization*
  The general form of a steady-state prediction equation is

$$y(\infty|k) = K_s \underbrace{(u(\infty|k) - u(k-1))}_{\Delta u_s(k)} + b(k) \tag{3.88}$$

  where $y(\infty|k)$ and $u(\infty|k)$ are the steady state values of the output and input projected at time k. With only m input moves considered,

$$\Delta u_s(k) = \Delta u(k) + \Delta u(k+1) + \ldots\ldots + \Delta u(k+m-1) \tag{3.89}$$

  Note that, for the step response model,

$$y(\infty|k) = y(k+m+n-1|k) \tag{3.90}$$

  and $K_s = S_n$. Also,

$$b(k) = \tilde{y}_{n-1}(k) + S_n^d \Delta d(k) + (y_m(k) - \tilde{y}_0(k)) \tag{3.91}$$

This steady-state prediction model can be used to optimize a given economic objective function subject to various input and output constraints.:

$$\min_{\Delta u_s(k)} \ell(u(\infty|k), y(\infty|k)) \tag{3.92}$$

Since an economic objective function is typically linear and the prediction equation is also linear, a Linear Program (LP) results. Alternatively, one can also solve

$$\min_{\Delta u_s(k)} \|\Delta u_s(k)\| \tag{3.93}$$

$$\min_{\Delta u_s(k)} \|r - y(\infty|k))\|_Q \tag{3.94}$$

In the first case, we would be looking for a minimum input change such that all the constraints are satisfied. In the second case, we would seeking a minimum deviation from the setpoint values that are achievable within the given constraints. The solution sets the target settling values for the inputs and outputs.

- Step 2: *Dynamic Optimization*
  The dynamic prediction equation is same as before. A quadratic regulation objective of the following is minimized subject to the give constraints through QP:

$$\left[\sum_{i=1}^{m+n-2} (y(k+i|k) - y^*(\infty|k))^T Q(y(k+i|k) - y^*(\infty|k)) + \sum_{j=0}^{m-1} \Delta u^T(k+j|k) R \Delta u(k+j|k)\right] \tag{3.95}$$

  where $y^*(\infty|k))$ is the solution from the steady-state optimization. An additional constraint may be added to match the settling values of the optimized input trajectories to those computed from the steady-state optimization:

$$\Delta u(k|k) + \Delta u(k+1|k) + \ldots\ldots + \Delta u(k+m-1|k) = \Delta u_s^*(k) \tag{3.96}$$

  This also forces $y(k+m+n-1|k)$ to be at the optimal steady-state value $y^*(k+\infty|k)$.

Note that, this steady-state optimization may be performed as often as at every sample time, that is at the same execution rate as the dynamic optimization. However, it is critical to filter the noise and other high frequency variations from the feedback signal. Otherwise, the solution from the steady-state solution can fluctuate wildly from sample time to sample time, especially in the case of a LP.

3.9.5 Product Property Estimation

Many CVs, such as product compositions and other property variables, cannot be measured at a frequency, speed, accuracy, and/or reliability required for direct feedback control to be effective. Control of these variables may be nevertheless critical and the only recourse may be to develop and use estimates from measurements of other process variables. Since all process variables are driven by a same basic set of disturbances and inputs, their behavior should be strongly correlated. The correlation can be captured, which can be used to build an *inferential estimator* for the property variables.

Typically, linear regression techniques, such as least squares and partial least squares (PLS), are used to build an estimator of the form

$$\hat{y}^p(k) = \mathbf{L} \begin{bmatrix} y_1^s(k-\delta_1) \\ \vdots \\ y_\ell^s(k-\delta_\ell) \end{bmatrix} \tag{3.97}$$

where $\hat{y}^p$ is the estimate of the product property variable in question and y_i^s's are he secondary variables used to estimate the product property. Because different variables can have different response times to various inputs, the regressor may need to include delayed measurement terms as shown above. Determination of appropriate delay amounts would require significant process knowledge or a careful data analysis. In the case that proper values for these delays cannot be determined a priori, one may have to include several delayed terms of a same variable in the regressor.

When direct measurements of the product property variables are available, one may use them in conjunction with the inferential estimates. In practice, the direct measurements are typically used to correct any bias in the inferential estimate. For example, when a measurement of y^p becomes available after a delay of θ_d sample steps, it can be included in the estimator in the following way:

$$\hat{y}^{p*}(k) = \hat{y}^p(k) + (y_m^p(k-\theta_d) - \hat{y}^p(k-\theta_d)) \tag{3.98}$$

where y_m^p is the measured value of y^p and $\hat{y}^{p*}(k)$ is the measurement-corrected estimate.

In the case that the process is highly nonlinear and / or the operation range is wide, nonlinear regression techniques such as Artificial Neural Networks can be used in place of the least squares technique.

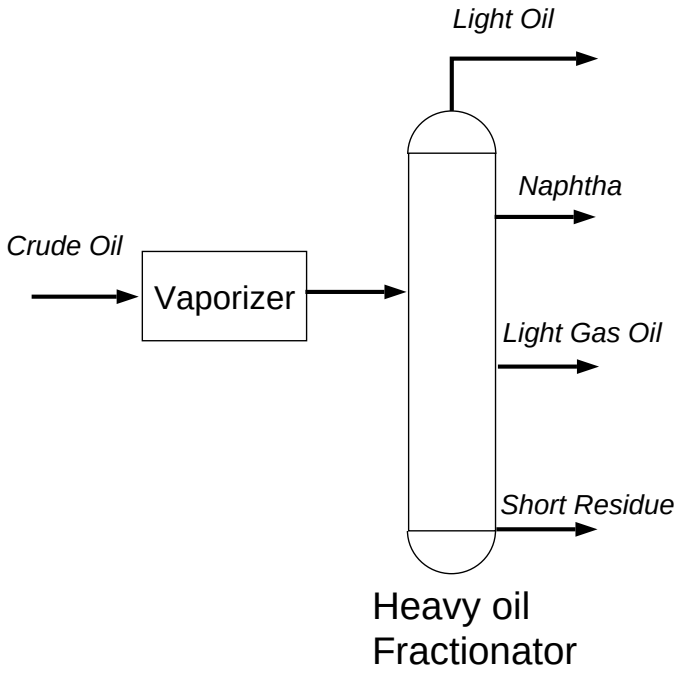

Figure 3.9: Heavy Oil Fractionator

3.10 Application of DMC to the Case Studies

3.10.1 Control of a heavy oil fractionator using *Dynamic Matrix Control*

This case study is intended to show how the MATLAB MPC Toolbox can be used to design a model predictive controller for a fairly complex, practically motivated control problem and test it through simulation. The case study is on a heavy-oil fractionation column and involves multivariable control, constraint handling, and optimization. The interested readers may attempt to solve the problem using the *standard step response based DMC technique* we explained in this chapter. A solution will be provided at the end of the section.

Heavy Oil Fractionator: Background

A heavy oil fractionator (Figure 3.9) splits crude oil into several streams that are further processed downstream. Vaporizing the feed stream consumes much energy and therefore heat integration is of paramount importance. The fractionator shown in the figure has three heat exchangers, which are used to recover energy from the recirculation streams. Product quality needs to be maintained at a desired level and certain constraints have to be met.

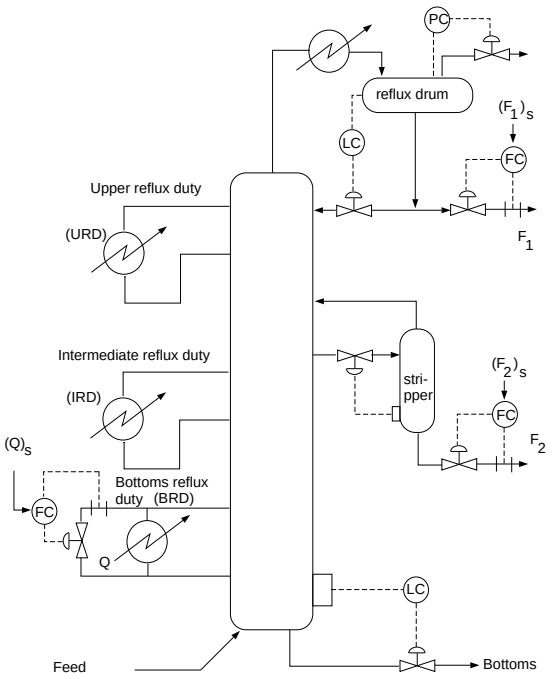

Figure 3.10: Control of Heavy Oil Fractionator — Level 1: Inventory Control

Control Structure Description

The control structure of the fractionator consists of two levels. In the lower level (displayed in Figure **??**), liquid levels and flow rates are controlled. This structure does not need MPC; conventional PD/PID controllers suffice.

In addition to the basic inventory and flow controls, *higher level* of control is needed to assure the quality of the output streams of interest. We may use composition measurements of the product streams obtained through on-line analyzers. Composition analyzers, however, introduce significant delays. Alternatively, we may use other more easily measurable variables such as temperatures as an indicator for compositions of the product streams. In addition to product quality control, this level may handle various constraints and any optimization objective that competes with control requirements. At this level, use of MPC may bring significant benefits. The control structure for this level is shown in Figure 3.11

The following are the control objectives at the MPC level listed in the order of their priority: (1) y_7 should remain above the minimum level of -0.5; (2) y_1, y_2 must be kept at their setpoints; (3) u_3, the opening of the bypass valve, must be minimized to maximize the heat recovery.

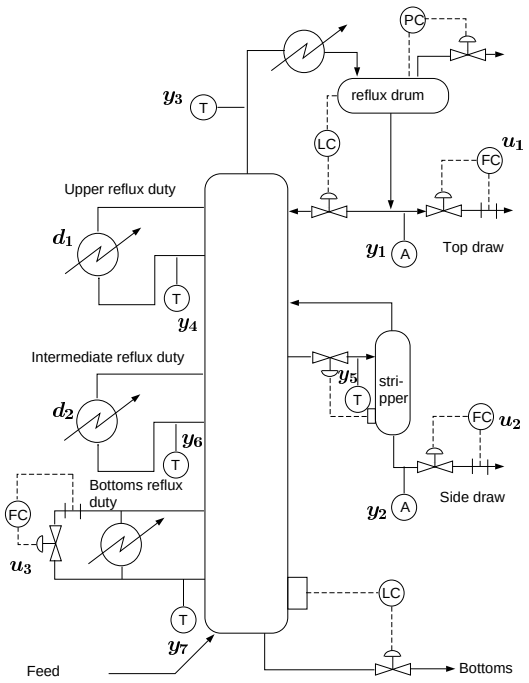

Figure 3.11: Control of Heavy Oil Fractionator — Level 2: Quality Control

Input Output Transfer Functions

Models relating the MVs to the CVs and other outputs are given as below:

	TD	SD	BRD	URD	IRD
	u_1	u_2	u_3	d_1	d_2
TEP y_1	$\frac{4.05e^{-27s}}{50s+1}$	$\frac{1.77e^{-28s}}{60s+1}$	$\frac{5.88e^{-27s}}{50s+1}$	$\frac{1.2e^{-27s}}{45s+1}$	$\frac{1.44e^{-27s}}{40s+1}$
SEP y_2	$\frac{5.39e^{-18s}}{50s+1}$	$\frac{5.72e^{-14s}}{60s+1}$	$\frac{6.9e^{-15s}}{40s+1}$	$\frac{1.52e^{-15s}}{25s+1}$	$\frac{1.83e^{-15s}}{20s+1}$
TT y_3	$\frac{3.66e^{-2s}}{9s+1}$	$\frac{1.65e^{-20s}}{30s+1}$	$\frac{5.53e^{-2s}}{40s+1}$	$\frac{1.16}{11s+1}$	$\frac{1.27}{6s+1}$
URT y_4	$\frac{5.92e^{-11s}}{12s+1}$	$\frac{2.54e^{-12s}}{27s+1}$	$\frac{8.1e^{-2s}}{20s+1}$	$\frac{1.73}{5s+1}$	$\frac{1.79}{19s+1}$
SDT y_5	$\frac{4.13e^{-5s}}{8s+1}$	$\frac{2.38e^{-7s}}{19s+1}$	$\frac{6.23e^{-2s}}{10s+1}$	$\frac{1.31}{2s+1}$	$\frac{1.26}{22s+1}$
IRT y_6	$\frac{4.06e^{-8s}}{13s+1}$	$\frac{4.18e^{-4s}}{33s+1}$	$\frac{6.53e^{-1s}}{9s+1}$	$\frac{1.19}{19s+1}$	$\frac{1.17}{24s+1}$
BRT y_6	$\frac{4.38e^{-20s}}{33s+1}$	$\frac{4.42e^{-22s}}{44s+1}$	$\frac{7.2}{19s+1}$	$\frac{1.14}{27s+1}$	$\frac{1.26}{32s+1}$

The transfer functions for the actual plant are assumed to be the same as the model, except that there can be gain variations. Structure of the variations is shown below. They are to be incorporated into the simulation as a *plant-model mismatch*.

	TD	SD	BRD	URD	IRD
	u_1	u_2	u_3	d_1	d_2
TEP y_1	$4.05 + 2.11\epsilon_1$	$1.77 + 0.39\epsilon_2$	$5.88 + 0.59\epsilon_3$	$1.2 + 0.12\epsilon_4$	$1.44 + 0.16\epsilon_5$
SEP y_2	$5.39 + 3.29\epsilon_1$	$5.72 + 0.57\epsilon_2$	$6.90 + 0.89\epsilon_3$	$1.52 + 0.13\epsilon_4$	$1.83 + 0.13\epsilon_5$
TT y_3	$3.66 + 2.29\epsilon_1$	$1.65 + 0.35\epsilon_2$	$5.53 + 0.67\epsilon_3$	$1.16 + 0.08\epsilon_4$	$1.27 + 0.08\epsilon_5$
URT y_4	$5.92 + 2.34\epsilon_1$	$2.54 + 0.24\epsilon_2$	$8.10 + 0.32\epsilon_3$	$1.73 + 0.02\epsilon_4$	$1.79 + 0.04\epsilon_5$
SDT y_5	$4.13 + 1.71\epsilon_1$	$2.38 + 0.93\epsilon_2$	$6.23 + 0.30\epsilon_3$	$1.31 + 0.03\epsilon_4$	$1.26 + 0.02\epsilon_5$
IRT y_6	$4.06 + 2.39\epsilon_1$	$4.18 + 0.35\epsilon_5$	$6.53 + 0.72\epsilon_3$	$1.19 + 0.08\epsilon_4$	$1.17 + 0.01\epsilon_5$
BRT y_6	$4.38 + 3.11\epsilon_1$	$4.42 + 0.73\epsilon_2$	$7.2 + 1.33\epsilon_3$	$1.14 + 0.18\epsilon_4$	$1.26 + 0.18\epsilon_5$

Problem Statement

The following are the five cases representing different disturbances and gain variations.

	d_1	d_2	ϵ_1	ϵ_2	ϵ_3	ϵ_4	ϵ_5
Case I:	0.5	0.5	0	0	0	0	0
Case II:	- 0.5	-0.5	-1	-1	-1	1	1
Case III:	-0.5	-0.5	1	-1	1	1	1
Case IV:	0.5	-0.5	1	1	1	1	1
Case V:	-0.5	-0.5	-1	1	0	0	0

The following constraints must be met.

- $-0.5 \leq u_1, u_2, u_3 \leq 0.5$
- $|\Delta u_1|, |\Delta u_2|, |\Delta u_3| \leq 0.05$
- $-0.5 \leq y_1 \leq 0.5, y_7 \geq -0.5$
- sampling time ≥ 1 min

The input constraints are *hard* constraints, meaning they must be satisfied. The output constraint can be viewed as a *soft* constraint, which is to be met if possible but may be relaxed in the case that infeasibility problems arise. You should relax the constraint for y_1 before you relax the constraint for y_7.

The objective is to satisfy to regulate y_1 and y_2, the product stream compositions, at their setpoints while satisfying the above constraints, for all five simulation cases. As a secondary objective, we wish to minimize u_3 in order to maximize the heat recovery.

In addition to the standard cases, we want to test some failure scenarios. Specifically, let us simulate the Case I when one of the actuators, u_1 or u_2, is out of service and also when one of the composition sensors goes out of service.

Figure 3.12: Control scenario block diagram

Note that d_1, d_2 are *unmeasured* disturbances. Therefore the model for the MPC calculation should not contain the models d_1, d_2. These models appear only in the "plant" used to do the closed-loop simulation.

Our Solution Strategy

We will use the MPC toolbox in MATLAB to solve this simulation problem. The other toolboxes available with MATLAB include SIMULINK, GUI (Graphical User Interface) etc. The block diagram representation of the problem is shown in Figure 3.12. The system has certain inputs and certain outputs. Out of all the measured outputs, we need to control y_1 and y_2. Since these are composition variables, they are associated with significant time delays. Alternatively, we may use some temperature or such other *inferential* measurements (see the earlier section on *Property Estimation*. However, this possibility is not considered in our solution strategy.

We can identify the following as the output variables of interest:

Controlled variables We need to control y_1 and y_2 at their set points.

Associate variables We need to maintain y_7 above its minimum value.

Optimized variables We need to obtain the required control action with minimal change in u_3. Thus, u_3 "enters" as an output variable.

Note that u_3 is also a manipulated (input) variable. Putting down the foregoing discussion in mathematical terms, we obtain the **model**

$$\begin{bmatrix} y_1 \\ y_2 \\ y_7 \\ u_3 \end{bmatrix} = G^u \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{3.99}$$

where,

$$G^u = \begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{71} & g_{72} & g_{73} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.100}$$

The model for actual **plant** is given by

$$\begin{bmatrix} y_1 \\ y_2 \\ y_7 \\ u_3 \end{bmatrix} = G^u \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} + G^d \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{3.101}$$

We solve the control problem using the `cmpc` algorithm available in the MPC toolbox[5]. The `cmpc` algorithm solves, as the name suggests, **C**onstrained **MPC** problem. The description of `cmpc` could be found in the MPC toolbox manual or by typing `help cmpc` at the MATLAB prompt. Resulting help text displayed is:

CMPC Simulation of the constrained Model Predictive Controller. [y,u,ym] = cmpc(plant, model, ywt, uwt,M,P, tend, r, ulim,ylim,... tfilter, dplant, dmodel, dstep) REQUIRED INPUTS: .....

The first step in solving the problem is to obtain the step response matrices `plant` and `model`. These two models are required to be in step response form. We use the following MATLAB commands [6] to do this.

First, we obtain the models in transfer function form using `poly2tfd` function. You may refer to the MPC toolbox manual for more details. The usage of this function to obtain the transfer function `g11` is shown. The third argument being 0 signifies continuous transfer function form and the final argument is the time delay. *g11 = poly2tfd(4.05, [50 1], 0, 27];* The resulting matrix corresponds to the transfer function $\frac{4.05\ e^{-27s}}{50s+1}$. We then use the `tfd2step` function to get them in the required form. Before that, as described above, we obtain all the necessary transfer functions (see Eq. 3.100). The transfer functions are passed column-wise (and not row-wise). Scalar `Ny` specifies the number of outputs (ie. number of rows).

delt=5; tfin=300; Ny=4;

model = tfd2step(tfin,delt,Ny,g11,g21,g71,gu31,g12,g22,g72,gu32,g13,g23,g73,gu33);

[5]Please refer to the MPC toolbox manual for details. You may also use the GUI (Graphical User Interface) version or SIMULINK, instead

[6]The commands described may be MATLAB commands or MPC Toolbox commands. In this example, we will use the same term in describing both. We assume here that you have MPC Toolbox installed. If not, some of these commands will not work.

You may want to see the validity of the `tfd2step` command by actually computing the step response matrix yourself. One way to do this is to find the inverse *Laplace* transform of a transfer function and obtain the step response matrix. Remember that, to obtain the step response, you need to multiply the transfer function with $1/s$ (output transform for a step input) before taking the inverse Laplace transform. First few elements of `model` are listed below:

$$\gg model =$$

$$\left[\begin{array}{c}\left[\begin{array}{ccc}0.0000 & 0.0000 & 0.0000\\ 0.0000 & 0.0000 & 0.0000\\ 0.0000 & 0.0000 & 1.6659\\ 0.0000 & 0.0000 & 1.0000\end{array}\right]\\ \left[\begin{array}{ccc}0.0000 & 0.0000 & 0.0000\\ 0.0000 & 0.0000 & 0.0000\\ 0.0000 & 0.0000 & 2.9464\\ 0.0000 & 0.0000 & 1.0000\end{array}\right]\\ \left[\begin{array}{ccc}0.0000 & 0.0000 & 0.0000\\ 0.0000 & 0.0945 & 0.0000\\ 0.0000 & 0.0000 & 3.9306\\ 0.0000 & 0.0000 & 1.0000\end{array}\right]\\ \left[\begin{array}{ccc}0.0000 & 0.0000 & 0.0000\\ 0.2113 & 0.5443 & 0.8108\\ \vdots & \vdots & \vdots\end{array}\right]\end{array}\right]$$

We need two step response matrices because there can be a mismatch between the actual `plant` and our `model`. Moreover, the disturbance models are not to be incorporated into the `model`, because the disturbances are not measured. Thus, we introduce the disturbances for the plant only and call them `dplant`, whereas `dmodel` is an empty matrix.

Next step is to specify the weighting matrices (Γ^y and Γ^u). This is done by specifying the vectors `ywt` and `uwt` respectively. Constraint limits are specified through vectors `ulim` and `ylim`. Using these values, we call the function `cmpc`.

Below is a sample program for Case II.

*E1=0; E2=0; E3=0; c11=4.05+2.11*E1; c12=1.77+0.39*E2; c13=5.88+0.59*E3; c21=5.39+3.29*E1; c22=5.72+0.57*E2; c23=6.90+0.89*E3; c71=4.38+3.11*E1; c72=4.42+0.73*E2; c73=7.20+1.33*E3;*

g11=poly2tfd(c11, [50 1], 0, 27); g12=poly2tfd(c12, [60 1], 0, 28); g13=poly2tfd(c13, [50 1], 0, 27);

g21=poly2tfd(c21, [50 1], 0, 18); g22=poly2tfd(c22, [60 1], 0, 14); g23=poly2tfd(c23, [40 1], 0, 15);

g71=poly2tfd(c71, [33 1], 0, 20); g72=poly2tfd(c72, [44 1], 0, 22); g73=poly2tfd(c73,

[19 1]);

gu31=poly2tfd(0, [1]); gu32=poly2tfd(0, [1]); gu33=poly2tfd(1, [1]);

delt=5; tfin=300; Ny=4;

model = tfd2step(tfin,delt,Ny,g11,g21,g71,gu31,g12,g22,g72,gu32,g13,g23,g73,gu33);

E1=-1; E2=-1; E3=-1; E4=1; E5=1; d1=-0.5; d2=-0.5; y7min=-0.5;

*c11=4.05+2.11*E1; c12=1.77+0.39*E2; c13=5.88+0.59*E3; c21=5.39+3.29*E1; c22=5.72+0.57*E2; c23=6.90+0.89*E3; c71=4.38+3.11*E1; c72=4.42+0.73*E2; c73=7.20+1.33*E3;*

g11=poly2tfd(c11, [50 1], 0, 27); g12=poly2tfd(c12, [60 1], 0, 28); g13=poly2tfd(c13, [50 1], 0, 27);

g21=poly2tfd(c21, [50 1], 0, 18); g22=poly2tfd(c22, [60 1], 0, 14); g23=poly2tfd(c23, [40 1], 0, 15);

g71=poly2tfd(c71, [33 1], 0, 20); g72=poly2tfd(c72, [44 1], 0, 22); g73=poly2tfd(c73, [19 1]);

plant = tfd2step(tfin,delt,Ny,g11,g21,g71,gu31,g12,g22,g72,gu32,g13,g23,g73,gu33);

*dc11=1.20+0.12*E4; dc12=1.44+0.16*E5; dc21=1.52+0.13*E4; dc22=1.83+0.13*E5; dc71=1.14+0.18*E4; dc72=1.26+0.18*E5;*

d11=poly2tfd(dc11, [45 1], 0, 27); d12=poly2tfd(dc12, [40 1], 0, 27); d21=poly2tfd(dc21, [25 1], 0, 15); d22=poly2tfd(dc22, [20 1], 0, 15); d71=poly2tfd(dc71, [27 1]); d72=poly2tfd(dc72, [32 1]);

du1=poly2tfd(0, [1]); du2=poly2tfd(0, [1]);

dplant = tfd2step(tfin,delt,Ny,d11,d21,d71,du1,d12,d22,d72,du2);

dmodel=[];

ywt=[1 1 0 1]; uwt=[0.1 0.1 0.1]; M=10; P=20; tend=300; r=[0 0 y7min 0];

ulim=[-0.5 -0.5 -0.5 0.5 0.5 0.5 0.05 0.05 0.05];

ylim=[-0.5 -inf y7min -0.5 0.5 inf inf 0.5];

tfilter=[]; dstep=[d1 d2];

[yp,u] = cmpc(plant,model,ywt,uwt,M,P,tend,r,ulim,ylim,tfilter,dplant,dmodel,dstep);

figure (3); plotall (yp,u,delt); subplot(211); legend('y1','y2','y7','u3',0); subplot(212); legend('u1','u2','u3',0);

Modifying the above for the other cases is trivial. Just the values for E1...E5, D1 and D2 need to be changed. The cases of u_1 actuator being stuck and no feedback for output y_1 are left as an exercise. Helpful hints:

- For a stuck actuator, the respective input should be constrained to a very small region of operation.

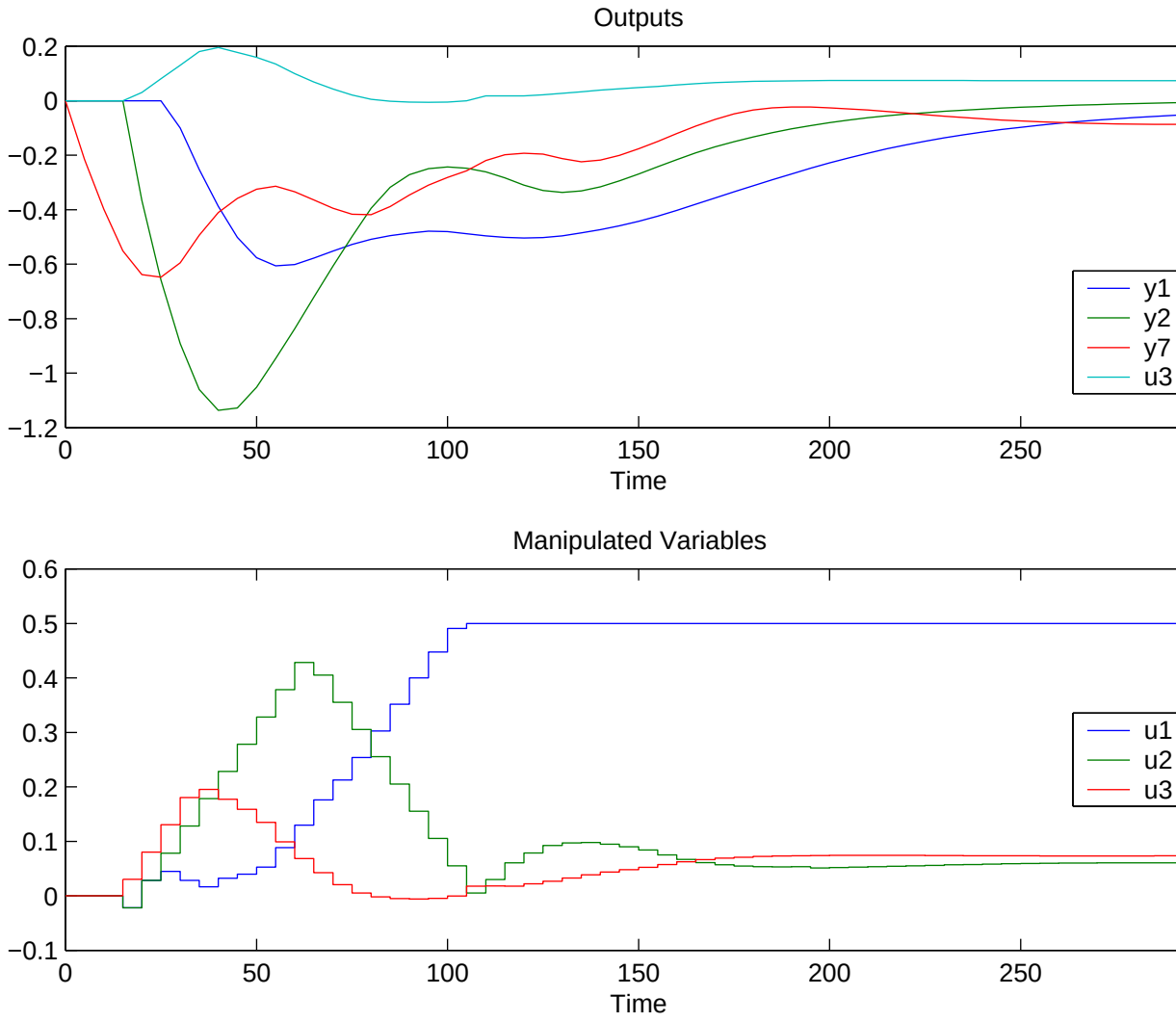

Figure 3.13: Result for Case-II

- If feedback filtering time delay is ∞, we get no feedback.