Chapter 2

A basic formulation of predictive control

2.1 State-space models

2.1.1 Form of the model

In most of this book we will assume a linearized, discrete-time, state-space model of the plant, in the form

$$x(k+1) = Ax(k) + Bu(k) \tag{2.1}$$
$$y(k) = C_y x(k) \tag{2.2}$$
$$z(k) = C_z x(k) \tag{2.3}$$

where x is an n-dimensional state vector, u is an ℓ-dimensional input vector, y is an m_y-dimensional vector of measured outputs, and z is an m_z-dimensional vector of outputs which are to be controlled, either to particular set-points, or to satisfy some constraints, or both. The variables in y and z will usually overlap to a large extent, and frequently they will be the same — that is, all the controlled outputs will frequently be measured.

We will often assume that $y \equiv z$, and we will then use C to denote both C_y and C_z, and m to denote both m_y and m_z. The index k counts 'time steps'.

The reason for using this standard form is mainly that it connects well with the standard theory of linear systems and control. We will often generalize this model by including the effects of measured or unmeasured disturbances, and of measurement noise.

We are going to assume that the sequence of actions at time step k is the following:

1. Obtain measurements $y(k)$.
2. Compute the required plant input $u(k)$.
3. Apply $u(k)$ to the plant.

This implies that there is always some delay between measuring $y(k)$ and applying $u(k)$. For this reason there is no 'direct feed-through' from $u(k)$ to $y(k)$ in the measured output equation (2.2), so that the model is 'strictly proper'. Taking computational delay into account is considered in more detail in Section 2.5.

The controlled outputs $z(k)$ could, in principle, depend on $u(k)$. That is, the description

$$z(k) = C_z x(k) + D_z u(k) \tag{2.4}$$

with $D_z \neq 0$ could be appropriate and useful in some cases. This would, however, complicate the computation of the optimal $u(k)$ slightly. This complication can be avoided, without losing anything, by defining a new vector of controlled outputs

$$\tilde{z}(k) = z(k) - D_z u(k) \tag{2.5}$$

which clearly depends on $x(k)$ only, without any direct feed-through from $u(k)$: $\tilde{z}(k) = C_z x(k)$. The corresponding changes to the cost function and constraints are easily made — see Exercises 2.12 and 2.13. We shall therefore assume that the controlled outputs are defined by (2.3).

2.1.2 Linear model, nonlinear plant

The relationship between the linear model (2.1)–(2.3) and the real plant needs careful consideration for predictive control. In most control methodologies the linear model is used off-line, as an aid to analysis and design. In predictive control it is used as part of the control algorithm, and the resulting signals are applied directly to the plant. Therefore careful attention must be paid to appropriate treatment of measurements before using them in the control computation algorithm, and of the computed control signal.

The plant in reality behaves in some complicated nonlinear fashion. Suppose that its state vector X evolves according to some nonlinear differential equation

$$\frac{dX}{dt} = f(X, U, t) \tag{2.6}$$

where U is the vector of inputs. (This is a simplification. Many processes are naturally described by implicit equations of the form

$$f(X, dX/dt, U, t) = 0 \tag{2.7}$$

and the extraction of the explicit form (2.6) may not be easy — but that is not important here.) Suppose the process is in some state $X = X_0$, with input $U = U_0$, and consider the effects of small perturbations $X = X_0 + x$, $U = U_0 + u$, with $\|x\|$ and $\|u\|$ both small:

$$\frac{dX}{dt} = f(X_0 + x, U_0 + u, t) \tag{2.8}$$

$$\approx f(X_0, U_0, t) + \left.\frac{\partial f}{\partial X}\right|_{(X_0, U_0, t)} x + \left.\frac{\partial f}{\partial U}\right|_{(X_0, U_0, t)} u \tag{2.9}$$

where quadratic and higher-order terms in x and u have been neglected. The expressions $\partial f/\partial X|_{(X_0,U_0,t)}$ and $\partial f/\partial U|_{(X_0,U_0,t)}$ denote matrices of partial derivatives, evaluated at (X_0, U_0, t). Denote these ('Jacobian') matrices by A_c and B_c, respectively. Since $X = X_0 + x$ and X_0 is a particular value of X, we have $dX/dt = dx/dt$. Hence we have the linearized model

$$\frac{dx}{dt} = A_c x + B_c u + f(X_0, U_0, t) \tag{2.10}$$

If (X_0, U_0) is an *equilibrium point* (that is, a possible steady state) at time t, namely if $f(X_0, U_0, t) = 0$, then clearly this model simplifies further, to the familiar form of continuous-time linear state-space model used very widely in systems and control theory, namely

$$\frac{dx}{dt} = A_c x + B_c u \tag{2.11}$$

This is the most common case: linearized models are usually found for the neighbourhood of an equilibrium point.

Sometimes it is useful to linearize at a point (X_0, U_0) which is *not* an equilibrium point. In this case we can define $(\dot{X})_0 = f(X_0, U_0, t)$[1] and

$$\dot{x} = \frac{dx}{dt} - (\dot{X})_0 \tag{2.12}$$

to obtain

$$\dot{x} = A_c x + B_c u \tag{2.13}$$

The form of this equation is the same as the one obtained when linearizing about an equilibrium point, and it can be used in much the same way. But it must be remembered that $\dot{x}$ is now a perturbation of the time derivative dx/dt, not the derivative itself. This affects the formation of predictions, for instance. The exact linearization about a non-equilibrium trajectory of a system is a time-varying linear model, since the state X does not remain constant at X_0. When using such models for predictive control, one does not necessarily re-linearize at each time step, however. Often the same linear model is retained for a number of time steps before re-linearizing, even if the plant is being transferred from one state to another.

[1] Note that this is different from dX_0/dt, which is zero. Non-standard notation is being used here; also our use of $\dot{x}$ is non-standard.

For predictive control we need the difference equation model (2.1), since we use a discrete-time setting. This can be obtained from the linearized differential equation by standard techniques, usually assuming that the input u is constant between sampling intervals [FPEN94]. Section 2.5 considers what to do if the computational delay is too large to allow the standard conversion formulae to be used.

Conceptually, it is also possible to arrive at the linear, discrete-time model (2.1) by linearizing an assumed discrete-time nonlinear model of the general form

$$x(k+1) = \phi\left(x(k), u(k), k\right) \tag{2.14}$$

But the function ϕ is usually not one that can be written down as a set of equations; it is the function implemented by solving the continuous-time differential equations between sampling intervals, assuming that the inputs are constant. In other words, it is necessary to run a simulation in order to discover what the value of the function ϕ is for a particular triple $(x(k), u(k), k)$.

The outputs of a real plant are also determined from the state in a nonlinear way:

$$Y = g(X, t) \tag{2.15}$$

where g is some nonlinear function, and we have assumed no explicit dependence on the input U. Proceeding as before, suppose that $Y_0 = g(X_0, t)$ and $Y = Y_0 + y$. Then

$$\begin{aligned} Y &= g(X_0 + x, t) && (2.16)\\ &\approx g(X_0, t) + \left.\frac{\partial g}{\partial X}\right|_{(X_0, t)} x && (2.17)\\ &= Y_0 + C_y x && (2.18) \end{aligned}$$

which leads to (2.2). The linearized equation (2.3) for the controlled outputs z can obviously be obtained in the same way. Since we assume that the output equations are static equations, their linearizations are the same in both the continuous-time and the discrete-time models. When obtaining measurements from the plant, the 'baseline' values Y_0 must be subtracted before using them in the linearized model. When making predictions, and when expressing constraints and objectives, care must be taken to either re-insert Y_0 at the appropriate points, or to express everything relative to Y_0. Conceptually this is very straightforward, but it requires care and a systematic approach to get it right on a complex plant.

Similar considerations apply when applying the control signal to the plant. The predictive control calculations usually give a value of u, and the 'baseline' value U_0 must be added to this before applying it to the plant. However, as we saw in the previous chapter, in predictive control we most often compute the required changes in the control signal from one time step to the next, $\Delta u(k) = u(k) - u(k-1)$. Since $\Delta u = \Delta U$, the required change can be applied directly. But the correct expression of constraints again requires attention to consideration of U_0, as does the expression of objectives in those cases in which the input values, as well as their changes, are of concern.

2.1.3 First-principles models and system identification

The linearized model (2.1)–(2.3) used in predictive control can be obtained in two ways, essentially. Most commonly, it is obtained by performing tests on the plant, which involve injecting known signals, such as steps, multi-sines, pseudo-random, or others, at the plant inputs, and recording the resulting plant outputs. Linearized models can then be obtained by using the techniques of *system identification*, which range from simple curve-fitting to sophisticated statistically based methods; there is a large literature on this, and we refer the reader to [Lju89, Nor86, vOdM96] as good starting points. It is sometimes claimed that predictive control has special requirements with regard to system identification. This is probably inaccurate, however; but it is certainly true that most applications of predictive control are in the process industries, and these industries have special requirements, largely because of the multivariable cross-couplings between inputs and outputs, and because of the slow dynamics that are typical. Some material on system identification specifically for the process industries can be found in [Ric91, RPG92, SMS92, ZB93, Zhu98].

Models obtained in this way are 'black box' models, which represent only the input–output behaviour of the plant, and carry no information about its internal structure. In some cases special-purpose tests may be unnecessary or impossible, but it may be possible to identify a linearized model on the basis of normal operating data; in particular, this is the situation in adaptive control [ÅW89, SR96a, Mos95].

It should be emphasized that the form of linear model that is adopted does not constrain the kind of test that should be applied to the plant. There appears to be quite a widespread misconception that step inputs are required in order to identify a model in step-response form, or that pseudo-random inputs are needed to obtain a transfer-function model form, and so on. This is not true; if one is lucky enough to have some choice of the input signal to use, that choice should depend primarily on the experimental conditions — how much testing is possible, how noisy are the measurements, etc. — and on the inherent requirements of the model — at which operating point should it be valid, what range of signal amplitudes are expected, over what range of frequencies should it be accurate, etc. Once a linear model is obtained in any of the standard forms, it is very straightforward to transform it to any other form. This point will be emphasized again in Chapter 4, and some specific transformation procedures will be discussed there.

Sometimes a *first-principles* nonlinear model of the plant is available. That is, a model in which the equations are obtained from a knowledge of the underlying physical, chemical and thermodynamic processes. For flight control, for example, the equations which accurately describe the nonlinear dynamics of an aircraft are well known, and not very complicated. For complex industrial processes such first-principles dynamic models are much more complex and expensive to develop, but this is being done increasingly commonly. (Typically they are developed for the purposes of operator training or safety certification, but once they have been developed, there is no reason why they should not be used for the purposes of control.) The availability of such a model corresponds to having (2.7) available, though one should be aware that the innocuous-looking notation $f(X, dX/dt, U, t) = 0$ can represent a very large collection of differential-algebraic equations, containing non-smooth elements such as switches (*case* statements) and look-up tables.

Linearized models can be obtained from a first-principles nonlinear model. In relatively simple cases, such as aircraft, this can be done largely 'by hand', but in complex cases it has to be done automatically. The standard way of doing this is by applying perturbations to the nonlinear model, and estimating the Jacobian matrices (A_c and B_c) numerically. Another very effective procedure is to use the nonlinear model to generate simulation data for particular conditions, and then to apply system identification techniques to this data, as if it had been obtained from the plant itself — see Chapter 4 for an example.

2.2 A basic formulation

For the basic formulation of predictive control we shall assume that the plant model is linear, that the cost function is quadratic (see Mini-Tutorial 1), and that constraints are in the form of *linear inequalities*. We shall also assume that everything is time-invariant. Furthermore, we shall assume that the cost function does not penalize particular values of the input vector $u(k)$, but only changes of the input vector, $\Delta u(k)$, which are defined as before. This formulation coincides with that used in the majority of the predictive control literature.

To make the formulation useful in the real world, we shall not assume that the state variables can be measured, but that we can obtain an estimate $\hat{x}(k|k)$ of the state $x(k)$, the notation indicating that this estimate is based on measurements up to time k — that is, on measurements of the outputs up to $y(k)$, and on knowledge of the inputs only up to $u(k-1)$, since the next input $u(k)$ has not yet been determined. $\hat{u}(k+i|k)$ will denote a future value (at time $k+i$) of the input u, which is assumed at time k. $\hat{x}(k+i|k)$, $\hat{y}(k+i|k)$ and $\hat{z}(k+i|k)$ will denote the predictions, made at time k, of the variables x, y and z at time $k+i$, on the assumption that some sequence of inputs $\hat{u}(k+j|k)$ $(j = 0, 1, \ldots, i-1)$ will have occurred. These predictions will be made consistently with the assumed linearized model (2.1)–(2.3). We will usually assume that the real plant is governed by the same equations as the model, although this is not really true. Only in Chapter 8 will we explicitly consider the fact that the model is inevitably inaccurate.

Cost function

The cost function V penalizes deviations of the predicted controlled outputs $\hat{z}(k+i|k)$ from a (vector) reference trajectory $r(k+i|k)$. Again the notation indicates that this reference trajectory may depend on measurements made up to time k; in particular, its initial point may be the output measurement $y(k)$, as in Chapter 1. But it may also be a fixed set-point, or some other predetermined trajectory. We define the cost function to be

$$V(k) = \sum_{i=H_w}^{H_p} \|\hat{z}(k+i|k) - r(k+i|k)\|^2_{Q(i)} + \sum_{i=0}^{H_u-1} \|\Delta\hat{u}(k+i|k)\|^2_{R(i)} \tag{2.19}$$

There are several points to note here. The prediction horizon has length H_p, but we do not necessarily start penalizing deviations of z from r immediately (if $H_w > 1$), Expressions like $x^T Qx$ and $u^T Ru$, where x, u are vectors and Q, R are symmetric matrices, are called *quadratic forms*, and are often written as $\|x\|_Q^2$ and $\|u\|_R^2$, respectively. They are just compact representations of certain quadratic functions in several variables. This is best illustrated by an example.

Example 2.1

Suppose that

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

and

$$Q = \begin{bmatrix} 10 & 4 \\ 4 & 3 \end{bmatrix}.$$

Then

$$\begin{aligned} x^T Qx &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 10 & 4 \\ 4 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 10x_1^2 + 8x_1x_2 + 3x_2^2. \end{aligned}$$

If $x^T Qx > 0$ for every x except $x = 0$, then this quadratic form is called *positive definite*. It can be shown that a quadratic form is positive definite if and only if all the eigenvalues of the matrix involved in it are positive, in which case we write $Q > 0$. If $x^T Qx \geq 0$ then the quadratic form is called *semi-positive definite* and we write $Q \geq 0$.

The matrix Q in the example has both eigenvalues positive. So the quadratic form is positive-definite. You can try substituting any real values for x_1 and x_2 (even negative ones), and you will see that the value is always positive. Note that this happens even if $x_1 = 0$ or $x_2 = 0$, provided that they are not both zero.

If $Q = I$ then $x^T Qx = x^T x = ||x||^2$, the squared 'length' of the vector, or squared *Euclidean norm*. Even if $Q \neq I$ (but $Q \geq 0$), it is possible to think of a quadratic form as the square of a 'weighted norm', since $x^T Qx = \|Q^{1/2}x\|^2$. Hence the notation $\|x\|_Q^2$.

The only other thing we will need to know about a quadratic form is how to find its gradient. Let $V = x^T Qx$, but now consider the general case where x has n elements, $x = [x_1, x_2, \ldots, x_n]^T$, and Q is an $n \times n$ matrix. Then the gradient of V is given by:

$$\begin{aligned} \nabla V &= \left[\frac{\partial V}{\partial x_1}, \frac{\partial V}{\partial x_2}, \ldots, \frac{\partial V}{\partial x_n} \right] \\ &= 2x^T Q \end{aligned}$$

(We are not going to prove this here.) Note that we have defined the gradient to be a row vector. Sometimes it is defined to be a column vector, in which case it is given by $2Qx$.

Mini-Tutorial 1 Quadratic forms.

because there may be some delay between applying an input and seeing any effect. H_u is the *control horizon*. We will always assume that $H_u \leq H_p$, and that $\Delta\hat{u}(k+i|k) = 0$ for $i \geq H_u$, so that $\hat{u}(k+i|k) = \hat{u}(k+H_u-1|k)$ for all $i \geq H_u$. (Recall Figure 1.4.)

The form of the cost function (2.19) implies that the error vector $\hat{z}(k+i|k) - r(k+i|k)$ is penalized at every point in the prediction horizon, in the range $H_w \leq i \leq H_p$. This is indeed the most common situation in predictive control. But it is possible to penalize the error at only a few coincidence points, as was done in Chapter 1, by setting $Q(i) = 0$ for most values of i. It is also possible to have different coincidence points for different components of the error vector by setting appropriate elements of the weighting matrices $Q(i)$ to 0. To allow for these possibilities, we do not insist on $Q(i) > 0$, but allow the weaker condition $Q(i) \geq 0$.[2] (At least this condition is required, to ensure that $V(k) \geq 0$.)

Example 2.2

Suppose that there are two controlled outputs. The prediction horizon is $H_p = 3$. There is only one coincidence point for the first output, at $i = 2$. For the second output there are two coincidence points, at $i = 2$ and $i = 3$. Errors in the second output are penalized more heavily than errors in the first output. This could be represented by

$$Q(1) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad Q(2) = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad Q(3) = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix} \tag{2.20}$$

with $H_w = 1$. It could also be represented by taking $H_w = 2$, in which case $Q(1)$ would not be defined, while $Q(2)$ and $Q(3)$ would remain as above. Most commonly, the weighting matrices $Q(i)$ and $R(i)$ are diagonal, as in this example.

We also need $R(i) \geq 0$ to ensure that $V(k) \geq 0$. Again, we do not insist on the stronger condition that $R(i) > 0$, because there are cases in which the changes in the control signal are not penalized — for instance, all the examples considered in Chapter 1. The weighting matrices $R(i)$ are sometimes called *move supression factors*, since increasing them penalizes changes in the input vector more heavily.

The cost function (2.19) only penalizes changes in the input vector, not its value. In some cases an additional term of the form $\sum \|\hat{u}(k+i|k) - u_0\|_S^2$ is added, which penalizes deviations of the input vector from some *ideal resting value*. Usually this is done only if there are more inputs than variables which are to be controlled to set-points. We shall not include such a term in our basic formulation, but it will be discussed in more detail in Section 10.1.

The prediction and control horizons H_p and H_u, the 'window' parameter H_w, the weights $Q(i)$ and $R(i)$, and the reference trajectory $r(k+i)$, all affect the behaviour of the closed-loop combination of plant and predictive controller. Some of these parameters, particularly the weights, may be dictated by the economic objectives of the control system, but usually they are in effect *tuning parameters* which are adjusted to give satisfactory dynamic performance. Chapters 6, 7 and 8 all discuss the influence of these parameters (and others).

Constraints

In the following we use $vec(0)$ to denote a column vector, each element of which is 0. We also use $vec(a, b, c)$ to denote the vector $[a, b, c]^T$, etc. We assume that constraints

[2] The notation $Q \geq 0$ usually excludes the extreme possibility that $Q = 0$. But we use it here to include this possibility.

of the following form hold over the control and prediction horizons:

$$E\ vec(\Delta\hat{u}(k|k), \dots, \Delta\hat{u}(k+H_u-1|k), 1) \le vec(0) \tag{2.21}$$

$$F\ vec(\hat{u}(k|k), \dots, \hat{u}(k+H_u-1|k), 1) \le vec(0) \tag{2.22}$$

$$G\ vec(\hat{z}(k+H_w|k), \dots, \hat{z}(k+H_p|k), 1) \le vec(0) \tag{2.23}$$

Here E, F and G are matrices of suitable dimensions. We can use constraints of this form to represent possible actuator slew rates (2.21), actuator ranges (2.22), and constraints on the controlled variables (2.23).

Example 2.3

Suppose we have a plant with 2 input variables and 2 controlled variables. The control horizon is $H_u = 1$ and the prediction horizon is $H_p = 2$ (with $H_w = 1$). We have the following constraints, which are to hold at each time:

$$-2 \le \Delta u_1 \le 2 \tag{2.24}$$

$$0 \le u_2 \le 3 \tag{2.25}$$

$$z_1 \ge 0 \tag{2.26}$$

$$z_2 \ge 0 \tag{2.27}$$

$$3z_1 + 5z_2 \le 15 \tag{2.28}$$

To put these into our standard form, we rewrite the Δu_1 constraints as follows.

$$-2 \le \Delta u_1 \Leftrightarrow -\Delta u_1 \le 2 \Leftrightarrow -\frac{1}{2}\Delta u_1 - 1 \le 0 \Leftrightarrow [-1/2, 0, -1]\begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ 1 \end{bmatrix} \le 0$$

$$\Delta u_1 \le 2 \Leftrightarrow \frac{1}{2}\Delta u_1 - 1 \le 0 \Leftrightarrow [1/2, 0, -1]\begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ 1 \end{bmatrix} \le 0$$

Now we write these two inequalities together, and replace Δu by $\Delta\hat{u}$:

$$\begin{bmatrix} -1/2 & 0 & -1 \\ 1/2 & 0 & -1 \end{bmatrix}\begin{bmatrix} \Delta\hat{u}_1(k|k) \\ \Delta\hat{u}_2(k|k) \\ 1 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

so that we have

$$E = \begin{bmatrix} -1/2 & 0 & -1 \\ 1/2 & 0 & -1 \end{bmatrix}$$

We leave finding the matrix F for this example to the reader (Exercise 2.4), and go on to find G. We proceed in the same way, the main difference being that we have to express the inequalities across the prediction horizon, which is greater than 1.

$$z_1 \geq 0 \Leftrightarrow -z_1 \leq 0$$

$$z_2 \geq 0 \Leftrightarrow -z_2 \leq 0$$

$$3z_1 + 5z_2 \leq 15 \Leftrightarrow \frac{1}{5}z_1 + \frac{1}{3}z_2 - 1 \leq 0 \Leftrightarrow [1/5, 1/3, -1]\begin{bmatrix} z_1 \\ z_2 \\ 1 \end{bmatrix} \leq 0$$

from which we get

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 1/5 & 1/3 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1/5 & 1/3 & -1 \end{bmatrix}}_{G} \begin{bmatrix} \hat{z}_1(k+1|k) \\ \hat{z}_2(k+1|k) \\ \hat{z}_1(k+2|k) \\ \hat{z}_2(k+2|k) \\ 1 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

It will be seen that the dimensions of the matrices E, F and G can become very large very quickly (H_p may be 10 or 20 or more). However, assembling them from the given constraints is simple, and this job can be automated. In fact, the software which we will use — the *Model Predictive Control Toolbox* for *MATLAB* — does this for the user.

The important thing about the constraints is that they are all in the form of *linear inequalities*. When we come to solve the predictive control optimization problem, all these inequalities will need to be translated into inequalities concerning $\Delta\hat{u}(k+i|k)$. Since we assume a linear model, we will see that these inequalities remain linear, even after this translation (Exercise 2.5). This is going to be crucial for being able to solve the optimization problem reliably and efficiently.

Notice that in the example there are variables (u_1, Δu_2) which are not constrained. It is also possible to have the converse situation, of variables which are constrained, but do not appear in the cost function (*zone* objectives). This is only likely to occur with the z variables, and is represented in our standard formulation by having appropriate zero entries in the weighting matrices $Q(i)$.

Example 2.4

Paper machine headbox

This example is based on an example in the *Model Predictive Control Toolbox* User's Guide [MR95], and some of the software-based problems will refer to it.

Part of a paper-making machine is shown in Figure 2.1. The following variables are involved:

Inputs (u)	**States (x)**	**Measured outputs (y)**	**Controlled variables (z)**
Stock flowrate G_p	Feed tank level H_1		
WW flowrate G_w	Headbox level H_2	H_2	H_2
	Feed tank consistency N_1	N_1	
	Headbox consistency N_2	N_2	N_2

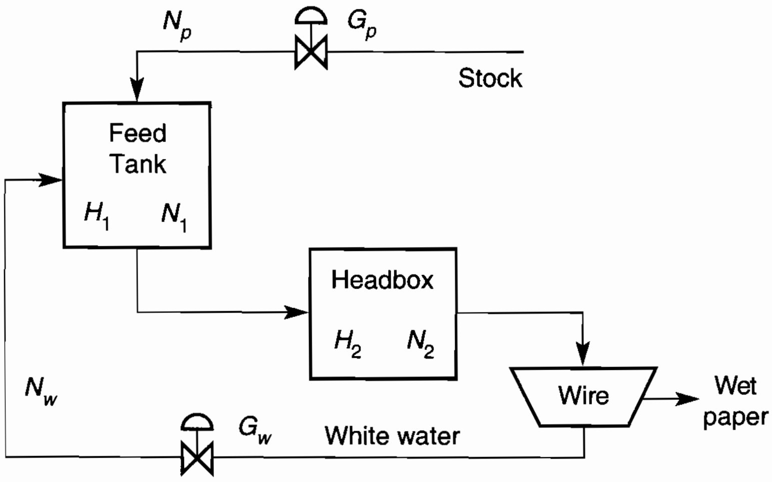

Figure 2.1 Paper machine with headbox.

A linearized model of this machine has the (continuous-time) state-space matrices

$$A_c = \begin{bmatrix} -1.93 & 0 & 0 & 0 \\ 0.394 & -0.426 & 0 & 0 \\ 0 & 0 & -0.63 & 0 \\ 0.82 & -0.784 & 0.413 & -0.426 \end{bmatrix} B_c = \begin{bmatrix} 1.274 & 1.274 \\ 0 & 0 \\ 1.34 & -0.65 \\ 0 & 0 \end{bmatrix}$$

$$C_y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad C_z = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where time has been measured in minutes. Predictive control is to be applied to this plant, with an update interval of $T_s = 2$ minutes. Discretizing the model with this sample interval (using the *MATLAB* function c2d for example) gives

$$A = \begin{bmatrix} 0.0211 & 0 & 0 & 0 \\ 0.1062 & 0.4266 & 0 & 0 \\ 0 & 0 & 0.2837 & 0 \\ 0.1012 & -0.6688 & 0.2893 & 0.4266 \end{bmatrix} \qquad B = \begin{bmatrix} 0.6462 & 0.6462 \\ 0.2800 & 0.2800 \\ 1.5237 & -0.7391 \\ 0.9929 & 0.1507 \end{bmatrix}$$

Note that the matrices C_y and C_z remain the same in the discrete-time model.

The prediction horizon is to be $H_p = 10$ (that is, 20 minutes) and the control horizon is to be $H_u = 3$. Since control of consistency is more important than control of level, errors in consistency (N_2) are penalized more than errors in level (H_2):

$$Q(i) = \begin{bmatrix} 0.2 & 0 \\ 0 & 1 \end{bmatrix}$$

Changes of the two valves are penalized equally, and the same penalties are applied at each step of the prediction and control horizons (that is, for each i):

$$R(i) = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

Since there is no delay in the plant (beyond the usual 1-step delay — $u(k)$ does not affect $x(k)$), errors are penalized from the first step, namely $H_w = 1$.

The valves are scaled so that their range of operation is:

$$-10 \leq u_1(k) \leq 10 \qquad -10 \leq u_2(k) \leq 10$$

(0 is usually the point at which the plant was linearized, so that the range is not always symmetric about 0), and their slew rate (maximum rate of change) is such that they can move through 10% of the range in one step (2 minutes):

$$-2 \leq \Delta u_1(k) \leq 2 \qquad -2 \leq \Delta u_2(k) \leq 2$$

The headbox level (H_2) is constrained, so that the headbox never runs dry or overfills:

$$-3 \leq z_1 \leq 5$$

Note that if an operating point is represented by $u = 0$, then with a linear model we must also have $y = 0$ at that point (unless there is integration in the plant).

From these constraints we can assemble the matrices E, F and G which appear in our standard formulation of the predictive control problem. These matrices will have the following dimensions:

$$E: \quad 4 \times 7 \qquad F: \quad 4 \times 7 \qquad G: \quad 20 \times 21$$

2.3 General features of constrained predictive control

Since predictive control problems usually include constraints, the resulting control law is usually *nonlinear*. It is easy to understand why this is so. Suppose we have a gas-turbine engine with a 'hard' state constraint of the form $x_j(t) < X_j$ — such as 'Turbine temperature lower than 1000 °C'. If the turbine is running at 995 °C and a disturbance d occurs which has the effect of moving x_j away from X_j — the cooling oil flow rate increases for some reason, say — then the control system will react in a fairly 'relaxed' way, and will coordinate bringing x_j back to its set-point with maintaining other objectives — correct turbine speed, power output, fuel economy, etc. Now if a disturbance $-d$ occurs instead, then by definition a linear controller would react in a similarly relaxed way, replacing a control signal $u(t)$ by $-u(t)$.

In contrast, a controller which is aware of the constraint will react in 'panic mode'. It 'knows' that there is no point in worrying about other objectives, because if the constraint is exceeded catastrophic failure is likely to result. It therefore reacts in a very different way to the disturbance $-d$ than it would to disturbance $+d$ — in our example, perhaps it will cut the fuel flow rate sharply and change the inlet vane angle to get maximum air flow. In a multi-engine aircraft it might even shut the engine down. Such behaviour is of course nonlinear.

We will see later that 'standard' predictive controllers in fact behave linearly so long as the plant is operating safely away from constraints, but nonlinearly when constraints are approached too closely. On the other hand, a standard predictive controller would not go so far as to shut down an engine, and it does not have the authority to do so. That kind of decision is usually handled by a separate higher-level 'supervisor', which may be a human operator or pilot.

Although constrained predictive controllers are nonlinear, they are usually *time-invariant*. This means that there is a function h, such that the control signal can be expressed as $u = h(x)$, namely it depends only on the current state, and not explicitly on time (that is, it is not necessary to write $u = h(x, t)$). Of course this function 'exists' only in a mathematical sense; it would be impossibly difficult to write it down in 'closed-form' (as a formula). In order to see what u is, you have to feed x into an optimization algorithm and see what comes out. This time-invariance occurs so long as there is nothing in the problem formulation that depends explicitly on time: the model of the plant, the cost function and the constraints must all be independent of time.

We have to be rather careful about what this means exactly. First of all, our model (2.1)–(2.3) does not depend on the time k, except through x and u. Secondly, the cost function (2.19) *can* depend on i — the weights $Q(i)$ and $R(i)$ can vary over the prediction and control horizons — but *not* on k. So a different weight can be attached to an error predicted to be 3 steps in the future than to an error predicted 1 step ahead. But when the problem is solved at the next k value, it must have the same pattern of weights. As time moves on, so k increases to $k + 1$, but exactly the same problem is solved as was solved at time k. So the cost function does not depend on time.

Thirdly, to get a time-invariant controller we can have constraints depending on i but not on k. So $|\hat{x}_j(k+i|k)| < X_j(i)$ is OK, but $|\hat{x}_j(k+i|k)| < X_j(k)$ is not. More generally,

$$\hat{x}(k+i|k) \in X(i) \tag{2.29}$$

$$\hat{u}(k+i|k) \in U(i) \tag{2.30}$$

is OK, whereas

$$\hat{x}(k+i|k) \in X(k) \tag{2.31}$$

$$\hat{u}(k+i|k) \in U(k) \tag{2.32}$$

is not. This can be a restriction if we want a time-invariant controller. For instance, when landing an aircraft, one may want constraints to become tighter as the aircraft approaches the runway. If one assumes a certain speed profile, the constraint tightening can be 'scheduled' as a function of time. This would give a time-varying predictive control law. However, it may make more sense to schedule the constraint tightening as a function of distance from the runway, and make this one of the states. In this case the control law will be time-invariant (though not necessarily simpler!).

There is no necessary advantage to having a time-invariant control law. Having one may make analysis of its behaviour easier in some cases, but in practice changes to the model, the cost and the constraints are all made from time to time.

2.4 Alternative state variable choices

As usual, our plant model (2.1) expresses the plant state x in terms of the values of the input u. But the cost function penalizes changes in the input, Δu, rather than the input values themselves. We shall see in the next chapter that the predictive control algorithm will in fact produce the changes Δu rather than u. It is therefore convenient for many purposes to regard the 'controller' as producing the signal Δu, and the 'plant' as having this signal as its input. That is, it is often convenient to regard the 'discrete-time integration' from Δu to u as being included in the plant dynamics. Figure 2.2 shows the real controller producing the signal u, which is passed to the real plant. It also shows the MPC 'controller' producing the signal Δu, which is passed to the MPC 'plant'.

There are several ways of including this 'integration' in a state-space model of the MPC 'plant'. All of them involve augmenting the state vector. The first way is to define the state vector[3]

$$\xi(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}. \tag{2.33}$$

Then, assuming the linear model (2.1) holds for the real plant state, we have

$$\begin{bmatrix} x(k+1) \\ u(k) \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} \Delta u(k) \tag{2.34}$$

$$y(k) = [C_y, 0] \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix} \tag{2.35}$$

$$z(k) = [C_z, 0] \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix} \tag{2.36}$$

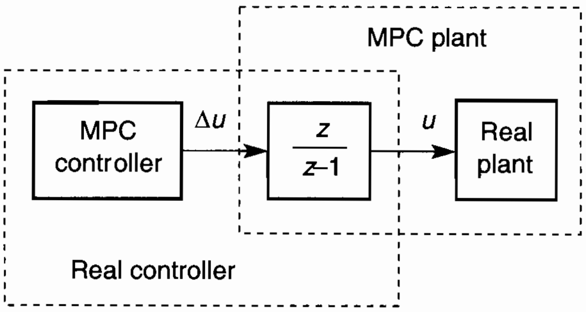

Figure 2.2 The boundary between controller and plant — a matter of convenience.

[3] We shall often use ξ as an augmented state, with associated matrices $\tilde{A}$ and $\tilde{B}$, etc. The meaning will be different, depending on the context, but will always be defined 'locally'.

A less obvious way is to define the state vector

$$\xi(k) = \begin{bmatrix} \Delta x(k) \\ y(k) \\ z(k) \end{bmatrix} \tag{2.37}$$

where $\Delta x(k) = x(k) - x(k-1)$ is the change in the state at time k. From (2.1) we have

$$x(k+1) = Ax(k) + Bu(k) \tag{2.38}$$

$$x(k) = Ax(k-1) + Bu(k-1) \tag{2.39}$$

Subtracting these gives

$$\Delta x(k+1) = A\Delta x(k) + B\Delta u(k) \tag{2.40}$$

Furthermore, from (2.2) and (2.3) we have

$$\begin{aligned} y(k+1) &= C_y x(k+1) \\ &= C_y[\Delta x(k+1) + x(k)] \\ &= C_y[A\Delta x(k) + B\Delta u(k)] + y(k) \end{aligned} \tag{2.41}$$

$$\begin{aligned} z(k+1) &= C_z x(k+1) \\ &= C_z[\Delta x(k+1) + x(k)] \\ &= C_z[A\Delta x(k) + B\Delta u(k)] + z(k) \end{aligned} \tag{2.42}$$

so that we have the state-space representation

$$\begin{bmatrix} \Delta x(k+1) \\ y(k+1) \\ z(k+1) \end{bmatrix} = \begin{bmatrix} A & 0 & 0 \\ C_y A & I & 0 \\ C_z A & 0 & I \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ y(k) \\ z(k) \end{bmatrix} + \begin{bmatrix} B \\ C_y B \\ C_z B \end{bmatrix} \Delta u(k) \tag{2.43}$$

$$\begin{bmatrix} y(k) \\ z(k) \end{bmatrix} = \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ y(k) \\ z(k) \end{bmatrix} \tag{2.44}$$

Both of these state-space representations have been used in the literature of predictive control. For example, the first representation (2.33) was used in [Ric90], while the second one (2.37) was used in [PG88] and many other publications. The second representation is also used in the *Model Predictive Control Toolbox* for some purposes. These two seem to be the most useful, although other representations are possible — see Exercise 2.8. Note that if the number of inputs is not the same as the number of measured and controlled outputs, then at least one of these representations is not minimal — that is, it must be either uncontrollable or unobservable — since the two representations will then have different state dimensions, but both give the same input–output transfer function.

2.5 Allowing for computational delay

Since predictive control involves on-line optimization, a considerable computational delay may be involved, and this should be taken account of. Figure 2.3 shows the assumptions we shall make about the timing of measurements made on the plant being controlled, and the resulting control signals being applied.

The measurement interval and control update interval are assumed to be the same, with length T_s. The plant output vector is measured at time kT_s, and this measurement is labelled $y(k)$. If there is a measured disturbance this is assumed to be measured at the same time, and this is labelled $d_m(k)$. There is then a delay of τ, which is the time taken by the predictive controller to complete its computations, after which a new control vector is produced and applied as the plant input signal. This input signal is labelled $u(k)$. The input signal is assumed to be held constant until it is recomputed T_s time units later. This sequence is repeated at time $(k+1)T_s$, and regularly thereafter.

In practice, process plants may have hundreds of measurements which may be taken, and/or made available, at various times during the measurement interval. If accurate modelling is required, this may have to be taken into account. Also, the computation delay τ may vary in practice, in which case the decision must be taken whether to apply the new control signal to the plant as soon as it becomes available — which probably improves the control but complicates the modelling and analysis — or whether the result should be held up until a standard interval has elapsed before applying it to the

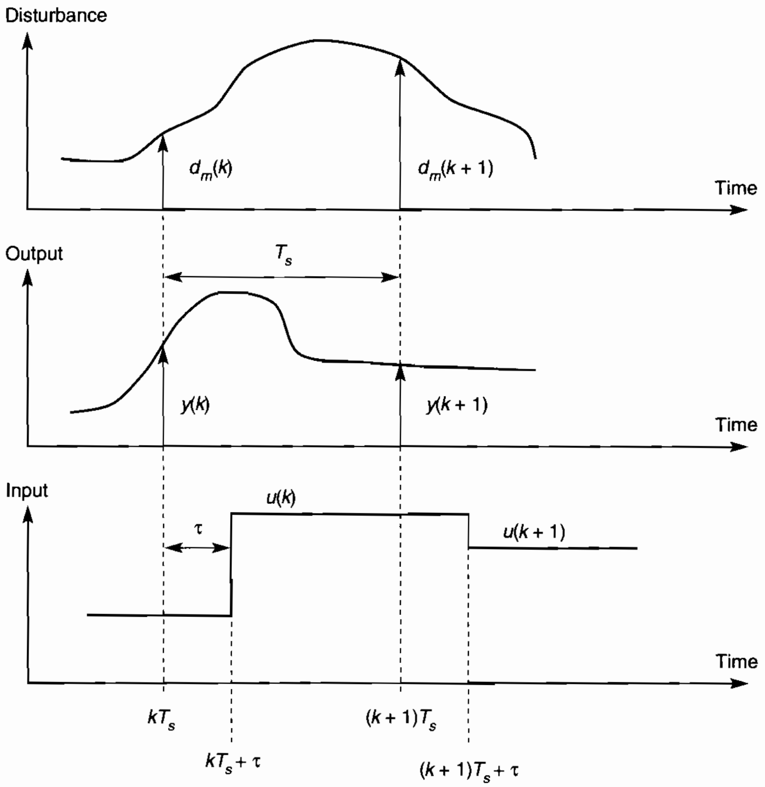

Figure 2.3 Assumed timing of measuring and applying signals.

plant. It would be impossible to deal with all such eventualities here, so we will assume that all the measurements are taken synchronously, as shown in Figure 2.3, and that the computational delay τ is the same at each step.

Now we suppose that we have a linearized model for the continuous-time plant in the state-space form:

$$\dot{x}_c(t) = A_c x_c(t) + B_c u_c(t) \tag{2.45}$$

$$y_c(t) = C_c x_c(t) \tag{2.46}$$

(It is enough to consider the measured outputs y. Controlled outputs z are treated in exactly the same way.) The assumptions shown in Figure 2.3 imply that $y(k) = y_c(kT_s)$, and that

$$u_c(t) = u(k) \quad \text{for} \quad kT_s + \tau \le t < (k+1)T_s + \tau \tag{2.47}$$

so that we have

$$\dot{x}_c(t) = \begin{cases} A_c x_c(t) + B_c u(k-1) & \text{for} \quad kT_s \le t < kT_s + \tau \\ A_c x_c(t) + B_c u(k) & \text{for} \quad kT_s + \tau \le t < (k+1)T_s \end{cases} \tag{2.48}$$

Now the solution of (2.45) is well known [FPEN94, Kai80] to be given by

$$x_c(t) = e^{A_c(t-t_0)} x_c(t_0) + \int_{t_0}^{t} e^{A_c(t-\theta)} B u_c(\theta) d\theta \tag{2.49}$$

where $t_0 < t$ is some initial time, and $x_c(t_0)$ is the initial state at that time. So, applying this solution over the interval $kT_s \le t < kT_s + \tau$, and defining $x(k) = x_c(kT_s)$, gives

$$x_c(kT_s + \tau) = e^{A_c \tau} x(k) + \left(\int_{kT_s}^{kT_s+\tau} e^{A_c(kT_s+\tau-\theta)} d\theta \right) B_c u(k-1) \tag{2.50}$$

Now using the change of variable $\eta = kT_s + \tau - \theta$ it is easy to show that

$$\int_{kT_s}^{kT_s+\tau} e^{A_c(kT_s+\tau-\theta)} d\theta = \int_0^{\tau} e^{A_c \eta} d\eta = \Gamma_1 \tag{2.51}$$

which is a constant matrix, so we have

$$x_c(kT_s + \tau) = e^{A_c \tau} x(k) + \Gamma_1 B_c u(k-1) \tag{2.52}$$

Similarly, applying (2.49) over the interval $kT_s + \tau \le t < (k+1)T_s$ gives

$$x(k+1) = e^{A_c(T_s-\tau)} x_c(kT_s + \tau) + \left(\int_{kT_s+\tau}^{(k+1)T_s} e^{A_c([k+1]T_s-\theta)} d\theta \right) B_c u(k) \tag{2.53}$$

$$= e^{A_c(T_s-\tau)} [e^{A_c \tau} x(k) + \Gamma_1 B_c u(k-1)] + \left(\int_0^{T_s-\tau} e^{A_c \eta} d\eta \right) B_c u(k) \tag{2.54}$$

$$= Ax(k) + B_1 u(k-1) + B_2 u(k) \tag{2.55}$$

where

$$A = e^{A_c T_s} \qquad B_1 = e^{A_c(T_s-\tau)}\Gamma_1 B_c \qquad B_2 = \int_0^{T_s-\tau} e^{A_c\eta}d\eta B_c \tag{2.56}$$

Now this is not in the standard form of a discrete-time state-space model, because of the dependence of $x(k+1)$ on $u(k-1)$. But we can easily bring it into the standard form by introducing the state

$$\xi(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix} \tag{2.57}$$

which gives the state equation in standard form:

$$\xi(k+1) = \tilde{A}\xi(k) + \tilde{B}u(k) \tag{2.58}$$

where

$$\tilde{A} = \begin{bmatrix} A & B_1 \\ 0 & 0 \end{bmatrix} \qquad \tilde{B} = \begin{bmatrix} B_2 \\ I \end{bmatrix} \tag{2.59}$$

From (2.46) the associated equation for the measured outputs is

$$y(k) = C_c x(k) = \tilde{C}\xi(k) \tag{2.60}$$

where $\tilde{C} = [C_c, 0]$.

Because of this possibility, we can continue to assume that the linearized plant model has the form (2.1)–(2.3), providing that an appropriate definition of the state vector is used. The reason for using this standard form is mainly that it connects well with the standard theory of linear systems and control. It is not necessarily the best form for implementation — for example it may be more efficient to use the form (2.55) in some circumstances.

The reader is warned that most standard software for finding discrete-time equivalents of continuous-time systems, such as the *Control System Toolbox* for *MATLAB* function c2d, does not take account of computation delay. It is, however, fairly easy to obtain the required model using such software — see Exercise 2.9.

Note that if the computational delay is negligible ($\tau = 0$) then $B_1 = 0$, and in the other extreme case, namely $\tau = T_s$, we have $B_2 = 0$. Most examples which appear in the predictive control literature neglect the computational delay.

2.6 Prediction

In order to solve the predictive control problem, we must have a way of computing the predicted values of the controlled variables, $\hat{z}(k+i|k)$, from our best estimate of the current state, $\hat{x}(k|k)$, and the assumed future inputs, or equivalently, the latest input, $u(k-1)$, and the assumed future input changes, $\Delta\hat{u}(k+i|k)$.

It may seem that this is really part of the solution, not of the problem formulation, so that it belongs in the next chapter. But it turns out that the way the predictions are made has a great effect on the performance of the closed-loop system running under predictive control. So the choice of prediction strategy is another 'tuning parameter' for predictive control, just as choices of horizons and cost functions are. Furthermore, the prediction strategy follows in a rather systematic way from assumptions made about disturbances acting on the system and measurement errors such as noise. So we can say that, rather than choosing a prediction strategy, we are specifying a model of the environment in which the plant is operating. And that properly belongs here, as part of the problem formulation.

However, prediction can get very complicated. So to avoid being too distracted by it at this stage, we will deal with a few simple cases here — which will already cover much of industrial practice — and come back to more general cases later.

2.6.1 No disturbances, full state measurement

We will start with the simplest situation. Assume that the whole state vector is measured, so that $\hat{x}(k|k) = x(k) = y(k)$ (so $C_y = I$). Also assume that we know nothing about any disturbances or measurement noise. Then all we can do is to predict by iterating the model (2.1)–(2.3). So we get

$$\begin{aligned}
\hat{x}(k+1|k) &= Ax(k) + B\hat{u}(k|k) && (2.61)\\
\hat{x}(k+2|k) &= A\hat{x}(k+1|k) + B\hat{u}(k+1|k) && (2.62)\\
&= A^2x(k) + AB\hat{u}(k|k) + B\hat{u}(k+1|k) && (2.63)\\
&\vdots\\
\hat{x}(k+H_p|k) &= A\hat{x}(k+H_p-1|k) + B\hat{u}(k+H_p-1|k) && (2.64)\\
&= A^{H_p}x(k) + A^{H_p-1}B\hat{u}(k|k) + \ldots + B\hat{u}(k+H_p-1|k) && (2.65)
\end{aligned}$$

In the first line we have used $\hat{u}(k|k)$ rather than $u(k)$, because at the time when we need to compute the predictions we do not yet know what $u(k)$ will be.

Now recall that we have assumed that the input will only change at times $k, k+1, \ldots, k+H_u-1$, and will remain constant after that. So we have $\hat{u}(k+i|k) = \hat{u}(k+H_u-1)$ for $H_u \leq i \leq H_p-1$. In fact, we will later want to have the predictions expressed in terms of $\Delta\hat{u}(k+i|k)$ rather than $\hat{u}(k+i|k)$, so let us do that now. Recall that $\Delta\hat{u}(k+i|k) = \hat{u}(k+i|k) - \hat{u}(k+i-1|k)$, and that at time k we already know $u(k-1)$. So we have

$$\begin{aligned}
\hat{u}(k|k) &= \Delta\hat{u}(k|k) + u(k-1)\\
\hat{u}(k+1|k) &= \Delta\hat{u}(k+1|k) + \Delta\hat{u}(k|k) + u(k-1)\\
&\vdots\\
\hat{u}(k+H_u-1|k) &= \Delta\hat{u}(k+H_u-1|k) + \ldots + \Delta\hat{u}(k|k) + u(k-1)
\end{aligned}$$

and hence we get

$$\begin{aligned}
\hat{x}(k+1|k) &= Ax(k) + B[\Delta\hat{u}(k|k) + u(k-1)] \\
\hat{x}(k+2|k) &= A^2x(k) + AB[\Delta\hat{u}(k|k) + u(k-1)] \\
&\quad + B\underbrace{[\Delta\hat{u}(k+1|k) + \Delta\hat{u}(k|k) + u(k-1)]}_{\hat{u}(k+1|k)} \\
&= A^2x(k) + (A+I)B\Delta\hat{u}(k|k) + B\Delta\hat{u}(k+1|k) + (A+I)Bu(k-1) \\
&\vdots \\
\hat{x}(k+H_u|k) &= A^{H_u}x(k) + (A^{H_u-1} + \ldots + A + I)B\Delta\hat{u}(k|k) \\
&\quad \ldots + B\Delta\hat{u}(k+H_u-1|k) + (A^{H_u-1} + \ldots + A + I)Bu(k-1)
\end{aligned}$$

(Notice the change at this point)

$$\begin{aligned}
\hat{x}(k+H_u+1|k) &= A^{H_u+1}x(k) + (A^{H_u} + \ldots + A + I)B\Delta\hat{u}(k|k) \\
&\quad \ldots + (A+I)B\Delta\hat{u}(k+H_u-1|k) \\
&\quad + (A^{H_u} + \ldots + A + I)Bu(k-1) \\
&\vdots \\
\hat{x}(k+H_p|k) &= A^{H_p}x(k) + (A^{H_p-1} + \ldots + A + I)B\Delta\hat{u}(k|k) \\
&\quad \ldots + (A^{H_p-H_u} + \ldots + A + I)B\Delta\hat{u}(k+H_u-1|k) \\
&\quad + (A^{H_p-1} + \ldots + A + I)Bu(k-1)
\end{aligned}$$

Finally we can write this in matrix-vector form:

$$\begin{bmatrix} \hat{x}(k+1|k) \\ \vdots \\ \hat{x}(k+H_u|k) \\ \hat{x}(k+H_u+1|k) \\ \vdots \\ \hat{x}(k+H_p|k) \end{bmatrix} = \underbrace{\begin{bmatrix} A \\ \vdots \\ A^{H_u} \\ A^{H_u+1} \\ \vdots \\ A^{H_p} \end{bmatrix} x(k) + \begin{bmatrix} B \\ \vdots \\ \sum_{i=0}^{H_u-1} A^i B \\ \sum_{i=0}^{H_u} A^i B \\ \vdots \\ \sum_{i=0}^{H_p-1} A^i B \end{bmatrix} u(k-1)}_{\text{past}} +$$

$$\underbrace{\begin{bmatrix} B & \cdots & 0 \\ AB+B & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \sum_{i=0}^{H_u-1} A^i B & \cdots & B \\ \sum_{i=0}^{H_u} A^i B & \cdots & AB+B \\ \vdots & \vdots & \vdots \\ \sum_{i=0}^{H_p-1} A^i B & \cdots & \sum_{i=0}^{H_p-H_u} A^i B \end{bmatrix} \begin{bmatrix} \Delta\hat{u}(k|k) \\ \vdots \\ \Delta\hat{u}(k+H_u-1|k) \end{bmatrix}}_{\text{future}} \qquad (2.66)$$

The predictions of z are now obtained simply as

$$\hat{z}(k+1|k) = C_z\hat{x}(k+1|k) \tag{2.67}$$
$$\hat{z}(k+2|k) = C_z\hat{x}(k+2|k) \tag{2.68}$$
$$\vdots$$
$$\hat{z}(k+H_p|k) = C_z\hat{x}(k+H_p|k) \tag{2.69}$$

or

$$\begin{bmatrix} \hat{z}(k+1|k) \\ \vdots \\ \hat{z}(k+H_p|k) \end{bmatrix} = \begin{bmatrix} C_z & 0 & \cdots & 0 \\ 0 & C_z & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C_z \end{bmatrix} \begin{bmatrix} \hat{x}(k+1|k) \\ \vdots \\ \hat{x}(k+H_p|k) \end{bmatrix} \tag{2.70}$$

A warning is in order about the prediction equation (2.66). It involves computing A^i, possibly for quite large values of i. This can lead to numerical problems. If the plant is unstable, then some elements in A^i may become extremely large relative to others, and relative to elements in lower powers of A. Since computers work with finite-precision arithmetic, this can sometimes lead to wrong results. Similar problems can occur if the plant is stable; in this case some elements of A^i may become extremely small relative to others. Again wrong answers may result. (Using 'IEEE Standard' arithmetic, computers cannot distinguish between 1 and $1+\epsilon$ if $|\epsilon| < 10^{-16}$, approximately.)

The safest way of computing the predictions is probably to iterate one step at a time. This is in effect what is done if the predictions are not computed explicitly when computing the optimal control signal, but if

$$\hat{x}(k+i|k) = A\hat{x}(k+i-1|k) + B\hat{u}(k+i-1|k) \tag{2.71}$$

is included as an equality constraint during the optimization, which also leads to the most efficient computation [RWR98] — see Section 3.2 for details. Alternatively, problems of predicting with an unstable plant can be alleviated by pre-stabilizing the plant, as described in Section 5.2. Even if explicit predictions are not made for the purposes of optimization, they may be needed for the operator interface. One of the problems associated with running an advanced controller, such as a predictive controller, is that plant operators may lose confidence in the correctness of the actions taken by the controller, if they do not understand what it is doing. Displaying predictions can contribute very effectively to retaining their confidence.

2.6.2 Constant output disturbance

Now we will assume that there are disturbances acting on the plant. The simplest assumption is that the measured and controlled outputs are the same ($z = y$), and that there is an 'output disturbance' — see Figure 2.4. At time k we do not know what the disturbance $d(k)$ is, but we can form an estimate of it, $\hat{d}(k|k)$, by comparing the measured output with the predicted one:

$$y(k) = \hat{y}(k|k-1) + \hat{d}(k|k) \tag{2.72}$$
$$= C_y\hat{x}(k|k-1) + \hat{d}(k|k) \tag{2.73}$$

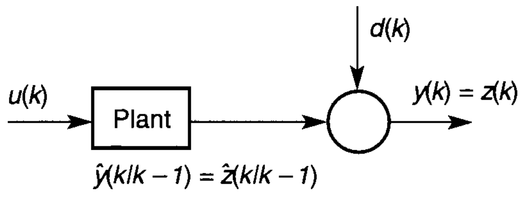

Figure 2.4 Output disturbance.

Another simple assumption is that this disturbance will continue unchanged during the prediction horizon. This means that, although we keep (2.2) and (2.3), we predict y and z using

$$\hat{z}(k+i|k) = \hat{y}(k+i|k) = C_y\hat{x}(k+i|k) + \hat{d}(k+i|k) \tag{2.74}$$

where

$$\hat{d}(k+i|k) = \hat{d}(k|k) \tag{2.75}$$

So at time step k we do the following:

1. Measure the actual plant output $y(k)$.
2. Estimate the disturbance as the difference between the actual and the estimated output.
3. Use that estimate to predict outputs over the prediction horizon.

We will generally follow this scheme, but Step 2 will become more elaborate when we make more elaborate assumptions about the nature of the disturbance. In Step 3 we need to assume future input movements, in order to use (2.1) or (2.66) to predict $\hat{x}(k+i|k)$.

The assumption of a constant output disturbance, and the simplest disturbance estimation scheme of (2.74)–(2.75) is sometimes called the 'DMC scheme', because that is what was used in one of the original proprietary predictive control products, *DMC* (= *Dynamic Matrix Control*) [CR80, PRC82]. The same scheme is used in most proprietary predictive control products — see Appendix A — and we have already used it in Chapter 1.

Notice that, even if $C_y = I$, we now have $\hat{y}(k+i|k) \neq \hat{x}(k+i|k)$. In general, this means that we need an *observer* to estimate $\hat{x}$.

2.6.3 Using an observer

If we cannot measure the full state vector, or if the measured outputs consist of some linear combinations of the states, so that the states cannot be measured directly, then an observer can be used to estimate the state vector. It is instructive to see how the 'constant output disturbance' assumption we made in the previous subsection can be handled using observer theory. We can do this by augmenting the model of the plant, The general structure of a *state observer* is shown in Figure 2.5, for a plant described by the equations

$$x(k+1) = Ax(k) + Bu(k), \qquad y(k) = Cx(k) \tag{2.76}$$

It is a copy of the plant, with feedback from the measured plant output, through the gain matrix L, to correct the state estimate $\hat{x}$.

The equations of the observer are:

$$\hat{x}(k|k) = \hat{x}(k|k-1) + L'[y(k) - \hat{y}(k|k-1)] \tag{2.77}$$

$$\hat{x}(k+1|k) = A\hat{x}(k|k) + Bu(k) \tag{2.78}$$

$$\hat{y}(k|k-1) = C\hat{x}(k|k-1) \tag{2.79}$$

Substituting the third equation into the first, and eliminating $\hat{x}(k|k)$, we get

$$\hat{x}(k+1|k) = A(I - L'C)\hat{x}(k|k-1) + Bu(k) + AL'y(k) \tag{2.80}$$

$$= (A - LC)\hat{x}(k|k-1) + Bu(k) + Ly(k) \tag{2.81}$$

where $L = AL'$. This is a stable system if the eigenvalues of $A - LC$ lie inside the unit disk. Furthermore if we define the state estimation error as $e(k) = x(k) - \hat{x}(k|k-1)$, then using (2.76) we have

$$e(k+1) = (A - LC)e(k)$$

which shows that the state estimation error converges to zero if the observer is stable, at a rate determined by the eigenvalues of $A - LC$.

If the pair (A, C) is observable, then given an arbitrary set of locations in the complex plane, a gain matrix L exists which places the observer eigenvalues at these locations. The problem of finding L is the dual of the state-feedback pole-placement problem, and can be solved using the same algorithm. (See functions `place`, `acker`, or `kalman` in *MATLAB*'s *Control System Toolbox*.)

If the state and output equations of the plant are assumed to be subjected to white noise disturbances with known covariance matrices, then L can be chosen such that the mean square state estimation error is the smallest possible. The observer is then known as a *Kalman Filter*.

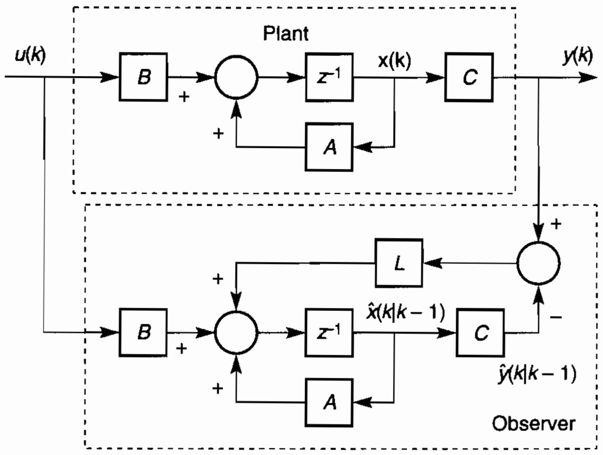

Figure 2.5 A state observer.

Mini-Tutorial 2 State observers.

so that it also includes a model of the output disturbance. We do this by defining a new augmented state:

$$\xi(k) = \begin{bmatrix} x(k) \\ d(k) \end{bmatrix}$$

Since we assume that the output disturbance is constant, the new state and output equations are

$$\begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) \tag{2.82}$$

$$y(k) = [C_y \quad I] \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} \tag{2.83}$$

Now if we partition the observer gain matrix L:

$$L = \begin{bmatrix} L_x \\ L_d \end{bmatrix}$$

then applying the standard observer equation (see Mini-Tutorial 2) gives the following estimates:

$$\begin{bmatrix} \hat{x}(k+1|k) \\ \hat{d}(k+1|k) \end{bmatrix} = \left(\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} - \begin{bmatrix} L_x \\ L_d \end{bmatrix} [C_y \quad I] \right) \begin{bmatrix} \hat{x}(k|k-1) \\ \hat{d}(k|k-1) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} L_x \\ L_d \end{bmatrix} y(k) \tag{2.84}$$

But in the previous subsection we had

$$\hat{d}(k|k) = -C_y \hat{x}(k|k-1) + y(k)$$

and since we assume that $\hat{d}(k+1|k) = \hat{d}(k|k)$ we have

$$\hat{d}(k+1|k) = -C_y \hat{x}(k|k-1) + y(k) \tag{2.85}$$

We can see that the estimate obtained from the observer is the same as this if the observer gain matrix L is

$$L = \begin{bmatrix} L_x \\ L_d \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

This gives

$$\left(\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} - \begin{bmatrix} L_x \\ L_d \end{bmatrix} [C_y \quad I] \right) = \begin{bmatrix} A & 0 \\ -C_y & 0 \end{bmatrix}$$

The block-triangular structure of this matrix shows that the observer's eigenvalues are those of the plant (the matrix A) and the remaining ones are zeros. The zeros indicate that 'deadbeat' estimation of the disturbance is obtained, namely that the disturbance estimate is exact after a finite number of steps — and we can see from (2.85) that it is in fact exact after only 1 step (if the real disturbance behaves exactly according to our model).

The fact that the observer's eigenvalues include those of A shows that this simple disturbance estimation scheme can be used only with stable plants — otherwise the estimate $\hat{x}(k+1|k)$ will get worse and worse as k increases. So we see that, although the 'DMC scheme' for estimating disturbances is simple and intuitive, it has some limitations. We also see that it is easy to overcome these limitations: we can keep the same disturbance model, but use a different observer gain matrix L. Even with unstable plants, it is possible to find L such that all the observer's eigenvalues lie inside the unit disk (providing that the pair (A, C_y) is observable).

Note that if an observer is used, then the form of the state prediction equation (2.66) remains the same, but $x(k)$ has to be replaced by $\hat{x}(k|k)$. We will look at the details of this later.

Suppose now that we use the alternative state-space representation (2.37), and assume again that y and z are the same. Since we are now assuming the presence of a disturbance on the output, we must distinguish between the actual measured output $y(k)$, and the output $\eta(k)$ that would have been obtained in the absence of any disturbance. We now take the state vector to be

$$\xi(k) = \begin{bmatrix} \Delta x(k) \\ \eta(k) \end{bmatrix} \tag{2.86}$$

which gives the plant model:

$$\xi(k+1) = \begin{bmatrix} A & 0 \\ C_yA & I \end{bmatrix} \xi(k) + \begin{bmatrix} B \\ C_yB \end{bmatrix} \Delta u(k) \tag{2.87}$$

$$y(k) = [0 \quad I]\xi(k) + d(k) \tag{2.88}$$

so the standard observer equation is

$$\begin{aligned}\hat{\xi}(k+1) = {} & \left(\begin{bmatrix} A & 0 \\ C_yA & I \end{bmatrix} - \begin{bmatrix} L_{\Delta x} \\ L_\eta \end{bmatrix} [0 \quad I] \right) \hat{\xi}(k) \\ & + \begin{bmatrix} B \\ C_yB \end{bmatrix} \Delta u(k) + \begin{bmatrix} L_{\Delta x} \\ L_\eta \end{bmatrix} y(k) \end{aligned} \tag{2.89}$$

Reading along the second row of this equation gives

$$\begin{aligned} \hat{\eta}(k+1) &= C_yA\Delta\hat{x}(k) + (I - L_\eta)\hat{\eta}(k) + C_yB\Delta u(k) + L_\eta y(k) \\ &= [C_yA\Delta\hat{x}(k) + \hat{\eta}(k) + C_yB\Delta u(k)] + L_\eta[y(k) - \hat{\eta}(k)] \end{aligned} \tag{2.90}$$

The first bracketed term is what we would expect the output to be at time $k+1$ in the absence of any disturbance at time $k+1$, and this is corrected by the second term, which depends on the error in the previous prediction.

Now if we have $L_\eta = I$, and if we write $\hat{d}(k|k) = y(k) - \hat{\eta}(k)$, then we get precisely the DMC prediction of the next plant output. That is, the difference between the actual and the predicted outputs is taken as the estimate of an output disturbance, and the same estimate is assumed to affect the next output. So we see that, in both state-space representations, the DMC estimate is obtained by the same observer gain matrix:

$$\begin{bmatrix} L_x \\ L_d \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix} = \begin{bmatrix} L_{\Delta x} \\ L_\eta \end{bmatrix} \tag{2.91}$$

(Note that the partitions in these two observer gain matrices have the same dimensions, since d and η are vectors with the same number of elements.) From (2.89) we see that in this case the observer's state transition matrix is

$$\left(\begin{bmatrix} A & 0 \\ C_y A & I \end{bmatrix} - \begin{bmatrix} L_{\Delta x} \\ L_\eta \end{bmatrix} [0 \quad I] \right) = \begin{bmatrix} A & 0 \\ C_y A & 0 \end{bmatrix} \tag{2.92}$$

so again the observer's eigenvalues are located at the plant's pole locations and at zero.

An important observation is that, with either representation, if we have $L_d = L_\eta = I$ then at each step the prediction is corrected by the prediction error at the previous step, whatever the value of L_x or $L_{\Delta x}$ — that is, *even if* $L_x \neq 0$ or $L_{\Delta x} \neq 0$. The significance of this is that most often it is not possible to obtain truly 'optimal' estimates of the state, because not enough is known about the statistical characteristics of disturbances and measurement noises, and because one has limited confidence in the accuracy of one's model. In these circumstances the observer gain becomes, in effect, another 'tuning parameter' which affects the behaviour of the predictive controller — particularly its response to disturbances, its stability margins, etc.: see Sections 7.4.2, 8.2 and 8.3 for further details. The state estimates produced by the observer can therefore bear little relation to any physical variables. But the key feature of predictive control is the ability to respect constraints. Clearly, it will only be possible to keep accurately to output constraints if accurate estimates of the outputs are available. The choice $L_d = I$ or $L_\eta = I$ ensures that the predicted output never diverges too far from the actual output — prediction errors are not allowed to accumulate over more than one step. This will therefore often be a constraint upon the observer design.

On the other hand, if some measurements are known to be very noisy, then there is no point in having the estimate of the real (i.e. meaningful) output jump about, following the noise at every step, which will not only cause unnecessary moves of the plant inputs, but might also cause random activation and de-activation of output constraints. In that case, at least we should be sure that the estimated output follows the real output 'on average'. In particular, if the measured output is constant, the estimated output should converge to the same constant value. Since the observer should be designed to be asymptotically stable, we know that, if the plant input and output measurements are constant ($\Delta u(k) = 0$ and $\Delta x(k) = 0$), then the state estimate will converge to a constant value. Suppose that $\hat{\eta}(k)$ converges to η_0. From (2.90) we see that, if the measured plant output has constant value $y(k) = y_0$, then

$$\eta_0 = \eta_0 + L_\eta(y_0 - \eta_0) \tag{2.93}$$

and hence $\eta_0 = y_0$ *whatever the value of* L_η (providing that $\det L_\eta \neq 0$). So, 'on average' the observer will always estimate the measured plant output correctly, so long as it is asymptotically stable. If there is no bias (no systematic error) in the measurement, then it will also estimate the real plant output correctly 'on average'. The same conclusion follows if we use the representation (2.82)–(2.83) — see Exercise 2.16.

2.6.4 Independent and realigned models

In Chapter 1 we noted the idea of having an *independent model*, namely one whose predictions are influenced only by the inputs which are applied to the plant, but not by the actual plant response. We can now see that this corresponds precisely to choosing an observer gain such that $L_x = 0$ (or $L_{\Delta x} = 0$ if the alternative representation is used). That part of the model's state vector which is supposed to correspond to the plant's state vector is not affected by output measurements in this case, as can be seen from equation (2.84) or (2.89).

As was already stated in Chapter 1, an independent model can only be used if it is stable. Attempting to use an independent unstable model would lead to rapid divergence of the state estimates from the plant states, which in turn would lead to very inaccurate predictions. With an unstable model, it is necessary to use $L_x \neq 0$ (or $L_{\Delta x} \neq 0$) in order to stabilize the observer.

One widely used procedure with difference equation (transfer function) models is to *realign* the model on the measurements, namely to use actual measurements of past outputs instead of past model outputs, when predicting future outputs. We have already introduced this in Section 1.6. We shall now show that this can be interpreted as using an observer, with a particular way of stabilizing it.

Consider the case when no attempt is made to model any disturbance. This corresponds to the use of equation (1.39). Suppose that a difference equation model is given in the form

$$y(k) + \sum_{i=1}^{r} A_i y(k-i) = \sum_{i=1}^{r} B_i u(k-i) \tag{2.94}$$

Several equivalent state-space forms can be obtained [Kai80]. One simple way of getting a state-space form of this model is to define the state to be made up of past inputs and outputs:

$$\begin{aligned} x(k) = [&y(k)^T, y(k-1)^T, \dots, y(k-r+1)^T, u(k-1)^T, u(k-2)^T, \dots, \\ &u(k-r+1)^T]^T \end{aligned} \tag{2.95}$$

Then a state-space model equivalent to (2.94) has its (A, B, C) matrices defined as:

$$A = \left[\begin{array}{cccc|cccc} -A_1 & -A_2 & \dots & -A_{r-1} & -A_r & B_2 & \dots & B_{r-1} & B_r \\ I_m & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & I_m & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & I_m & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 & I_\ell & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & I_\ell & 0 \end{array}\right] \tag{2.96}$$

$$B = \left[\begin{array}{cccccccccc} B_1^T & 0 & 0 & \dots & 0 & I_\ell & 0 & \dots & 0 \end{array}\right]^T \tag{2.97}$$

$$C = \left[\begin{array}{cccccccccc} I_m & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \end{array}\right] \tag{2.98}$$

Now suppose that an observer is used with gain matrix

$$L' = \begin{bmatrix} I_m & 0 & \dots & 0 \end{bmatrix}^T \tag{2.99}$$

Then

$$I - L'C = \begin{bmatrix} 0_{m,m} & 0 \\ 0 & I \end{bmatrix} \tag{2.100}$$

so the first element of $\hat{x}(k|k)$ becomes $y(k)$. Furthermore, the form of A ensures that $y(k)$ will become the second (vector) element of $\hat{x}(k+1|k)$ and of $\hat{x}(k+1|k+1)$, the third element of $\hat{x}(k+2|k+1)$ and of $\hat{x}(k+2|k+2)$, and so on. Also $u(k)$ will enter $\hat{x}(k+1|k)$ and $\hat{x}(k+1|k+1)$, and again the form of A will lead to it propagating 'downwards' in $\hat{x}(k+2|k+1), \hat{x}(k+3|k+2)$, etc. Thus, after at most r steps of running the observer, it will be true that $\hat{x}(k|k) = x(k)$. That is, the state estimate obtained from the observer will contain present and past values of the input and output, and predictions obtained on the basis of this estimate will be precisely the same as predictions obtained using the 'realigned' difference equation model.

The dynamics of the observer are determined by the eigenvalues of $A(I - L'C) = A - LC$, which has the block-triangular form:

$$A(I - L'C) = A - LC = \begin{bmatrix} 0_{m,m} & X \\ 0 & J \end{bmatrix} \tag{2.101}$$

where $X = [-A_2, -A_3, \dots, B_r]$, and

$$J = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ I_m & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I_m & 0 \end{bmatrix} \tag{2.102}$$

Since this matrix is block-triangular, the top-left block is zero, and all the eigenvalues of J are zero, it is clear that all the eigenvalues of $A - LC$ are zero. That is, this observer is always stable, and in fact 'deadbeat'; note that no assumptions have been made about the stability of the plant.

This section, then, has shown several things:

1. 'Realigning' the prediction model on past data is a way of stabilizing the observer, and explains why it can be used with unstable plant models.

2. It is not the only way of dealing with unstable plant models. There are many other observer gain matrices which give a stable observer, and may sometimes be more suitable.

3. However, the advantage of 'realigning'the model, namely of choosing the state as in (2.95), is that the past inputs and outputs are already known, and there is no real need for an observer — the analysis of this section is just an interpretation in terms of observers.

2.7 Example: Citation aircraft model

The following is a constant-speed approximation of some of the linearized dynamics of a Cessna Citation 500 aircraft, when it is cruising at an altitude of 5000 m and a speed of 128.2 m/sec. The elevator angle (rad) is the only input, and the pitch angle (rad), altitude (m), and altitude rate (m/s) are outputs.

$$\dot{x} = Ax + Bu \qquad y = Cx + Du$$

where

$$A = \begin{bmatrix} -1.2822 & 0 & 0.98 & 0 \\ 0 & 0 & 1 & 0 \\ -5.4293 & 0 & -1.8366 & 0 \\ -128.2 & 128.2 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} -0.3 \\ 0 \\ -17 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -128.2 & 128.2 & 0 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

For the purposes of this example we will pretend that this is an accurate representation of the aircraft dynamics.

The elevator angle is limited to $\pm 15°$ (± 0.262 rad), and the elevator slew rate is limited to $\pm 30°/\text{sec}$ (± 0.524 rad/sec). These are limits imposed by the equipment design, and cannot be exceeded. For passenger comfort the pitch angle is limited to $\pm 20°$ (± 0.349 rad).

Figure 2.6 shows the response to a step change of 40 m in the altitude set-point at time 0, with set-points for pitch angle and altitude rate held at 0, and the reference trajectory equal to the set-point: $r(k+i|k) = [0, 40, 0]^T$ for $k \geq 0$. The sampling interval is $T_s = 0.5$ sec, the prediction horizon is $H_p = 10$ (5 sec), and the control horizon is $H_u = 3$ (1.5 sec). Tracking errors are penalized over the whole prediction horizon ($H_w = 1$). The weights on tracking errors are $Q(i) = I_3$ (the same at each point in the prediction horizon) and the weights on control moves are $R(i) = 1$ (the same at each point in the control horizon). For this magnitude of change none of the constraints are active at any point of the transient, and a linear time-invariant controller results — as will be shown in detail in Chapter 3.

Note that the altitude tracking error dominates the pitch angle and altitude rate errors during most of the transient, so that the pitch angle and altitude rate depart from their set-points in order to allow the altitude error to be reduced. As the required altitude is acquired, all three outputs settle rapidly to their set-points. This behaviour is entirely a result of the numerical values of the errors which arise, the altitude error being sufficiently larger, numerically, than the other errors for its contribution to the cost function to be the dominant one. This will not be the case in general, and — as in all multivariable control problems — problem-dependent scaling must be employed to obtain the desired performance. (For example, if the pitch angle had been represented in degrees rather than radians in the predictive control algorithm then the pitch set-point would have been much more significant, and would have impeded the acquisition of the required altitude.)

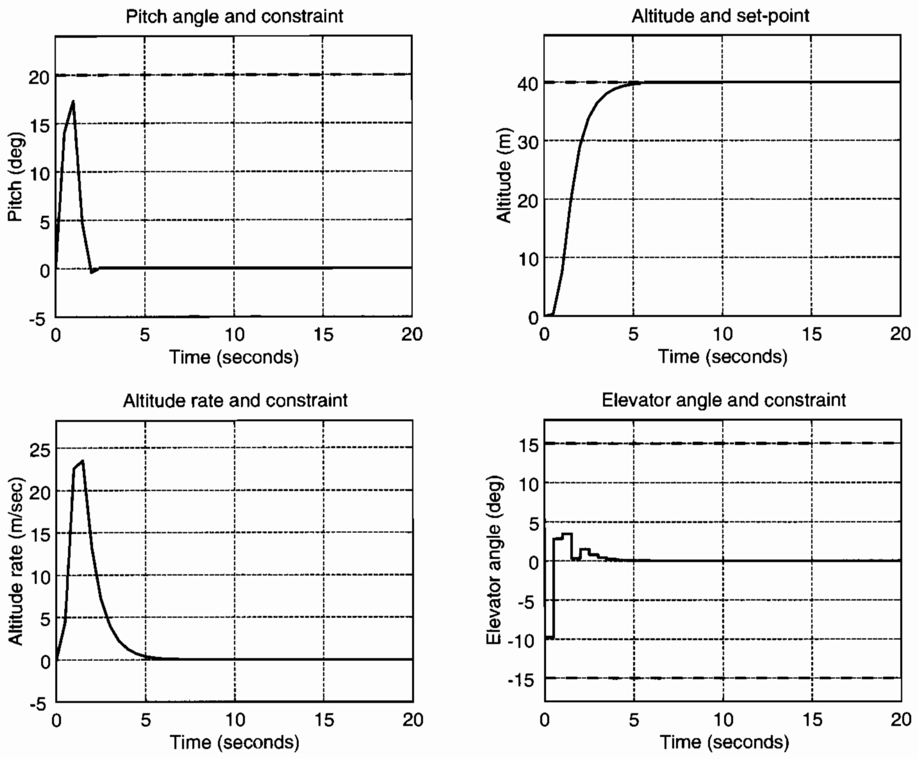

Figure 2.6 Response of Citation aircraft to 40 m step change in altitude set-point.

Other specifications are possible for this manoeuvre. One obvious possibility is to allow the pitch angle and altitude rate to be free until the altitude error has been reduced to some small value. This could be done by reducing the weights on the corresponding errors, possibly for a fixed time, based on the anticipated time required to complete the manoeuvre:

$$Q_1(k+i|k) = Q_3(k+i|k) = \begin{cases} 0 & \text{if} \quad k+i < 8 \\ 1 & \text{if} \quad k+i \geq 8 \end{cases} \tag{2.103}$$

But this would need to be adjusted for each amplitude of required altitude change, and would be prone to error if the manoeuvre was not performed exactly as planned; a much better alternative would be to make the weights state-dependent:[4]

$$Q_1(k+i|k) = Q_3(k+i|k) = \begin{cases} 0 & \text{if} \quad |r_2(k+i|k) - \hat{y}_2(k+i|k)| > 5 \\ 1 & \text{if} \quad |r_2(k+i|k) - \hat{y}_2(k+i|k)| \leq 5 \end{cases} \tag{2.104}$$

Suppose that a larger change in altitude is required: 400 m instead of 40 m. Figure 2.7 shows the response in this case. It can be seen that the pitch angle constraint becomes

[4] Simulation with state-dependent weights is not possible in the *Model Predictive Control Toolbox*. However, the modified functions `scmpc2` and `scmpcnl2` available on this book's web site make this possible, albeit inefficiently.

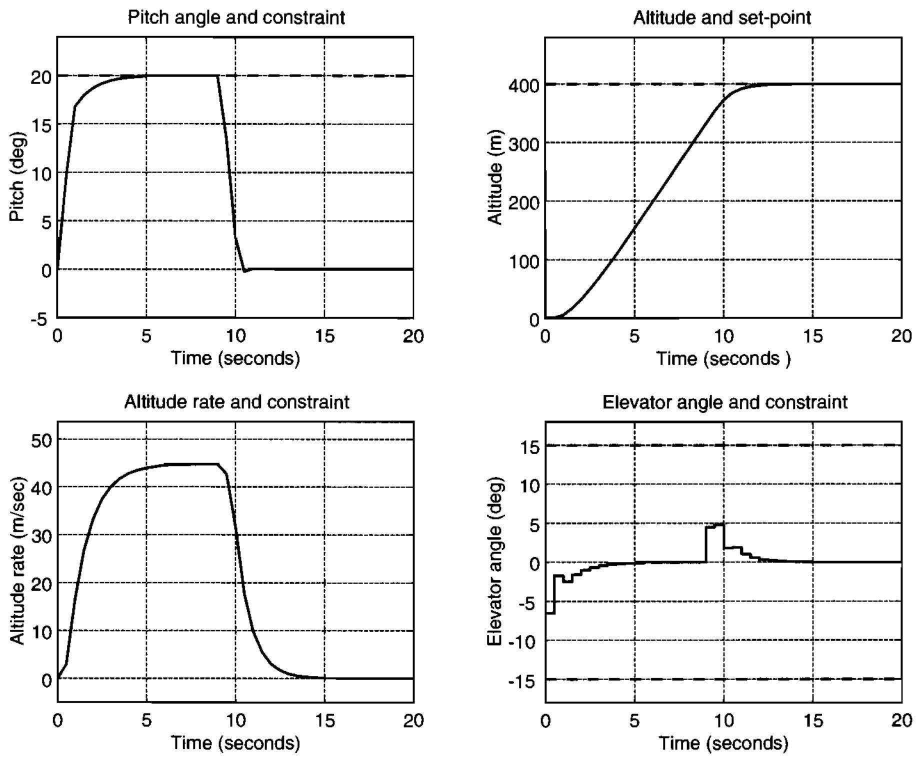

Figure 2.7 Response of Citation aircraft to 400 m step change in altitude set-point.

active for much of the transient, with the result that the altitude rate becomes constant for part of the transient. The control action is now nonlinear.

In Figures 2.6 and 2.7 the rate of change of altitude was unconstrained. Suppose now that it is constrained to 30 m/sec, and an altitude change of 400 m is again required. Figure 2.8 shows the resulting response. It can be seen that the pitch angle constraint is briefly active near the beginning of the transient, but that the altitude rate constraint is active for most of the time. The resulting (nonlinear) behaviour is close to that obtained from a conventional autopilot during an altitude change: a required rate of climb is held until the required altitude is nearly acquired, whereupon the required altitude is acquired and held.

This example demonstrates some of the flexibility that results from the ability to specify and respect constraints. Quite complicated behaviours can be obtained by relatively simple specifications, without any need to introduce mode-switching logic explicitly.[5] It should be emphasized that the intention here is to demonstrate just how

[5] There is an analogy here with the difference between 'procedural' programming, in which software is used to tell a computer *how* to perform a required computation, and 'declarative' programming, in which it is used to define *what* is to be computed. Mode-switching logic specifies *how* the controller is to achieve the required behaviour, whereas the use of constraints specifies *what* behaviour it must achieve.

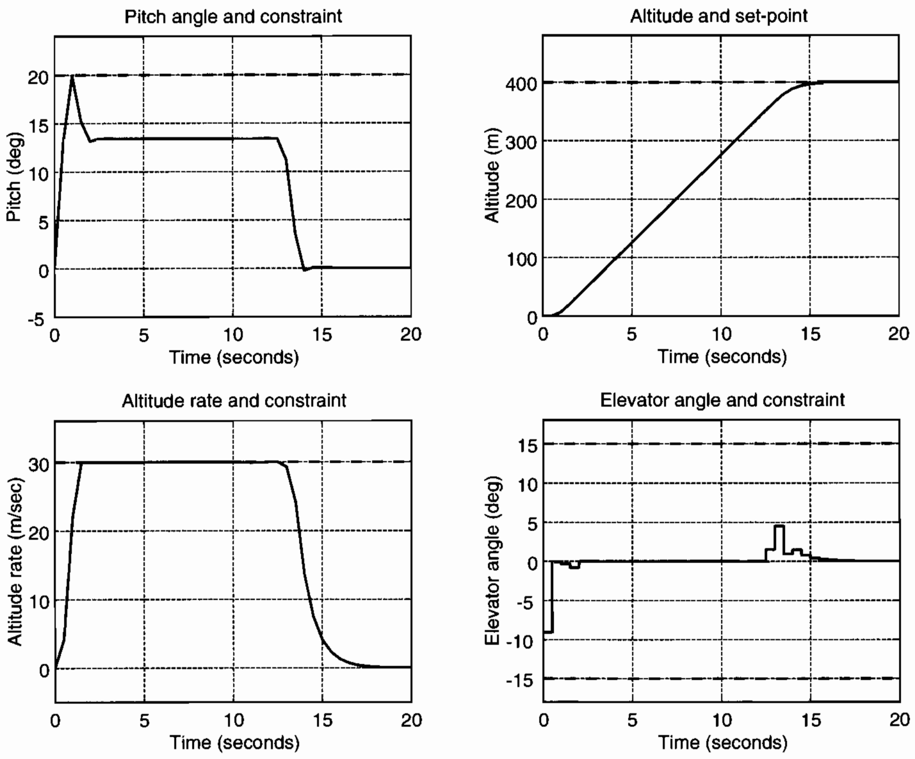

Figure 2.8 Response of Citation aircraft to 400 m step change in altitude set-point, with altitude rate constraint.

much flexibility there is. It is *not* being proposed that constraints are usually a good substitute for set-points. In this example it may well be preferable to have a set-point for the altitude rate, either simultaneously with a constant set-point for the altitude (which may require changing the weights during the maneouvre, as in (2.103) or (2.104)), or to ramp the altitude reference trajectory from the latest altitude measurement:

$$r(k+i|k) = \begin{bmatrix} 0 \\ y_2(k) + 30T_s i \\ 30 \end{bmatrix} \quad \text{if} \quad \hat{y}_2(k+i|k) < 395 \tag{2.105}$$

$$r(k+i|k) = \begin{bmatrix} 0 \\ 400 \\ 0 \end{bmatrix} \quad \text{if} \quad \hat{y}_2(k+i|k) \geq 395 \tag{2.106}$$

Different ways of specifying such manoeuvres will result in different behaviours in response to disturbances. For example, turbulence which results in the altitude rate exceeding 30 m/sec will result in more agressive control action to reduce this rate if it is specified by a constraint rather than as a set-point. Figures 2.9 and 2.10 show the responses when turbulence appears as a disturbance pulse on the altitude rate of 5 m/sec, lasting for 5 sec, which starts 5 sec into the transient, in the two cases. Figure 2.9

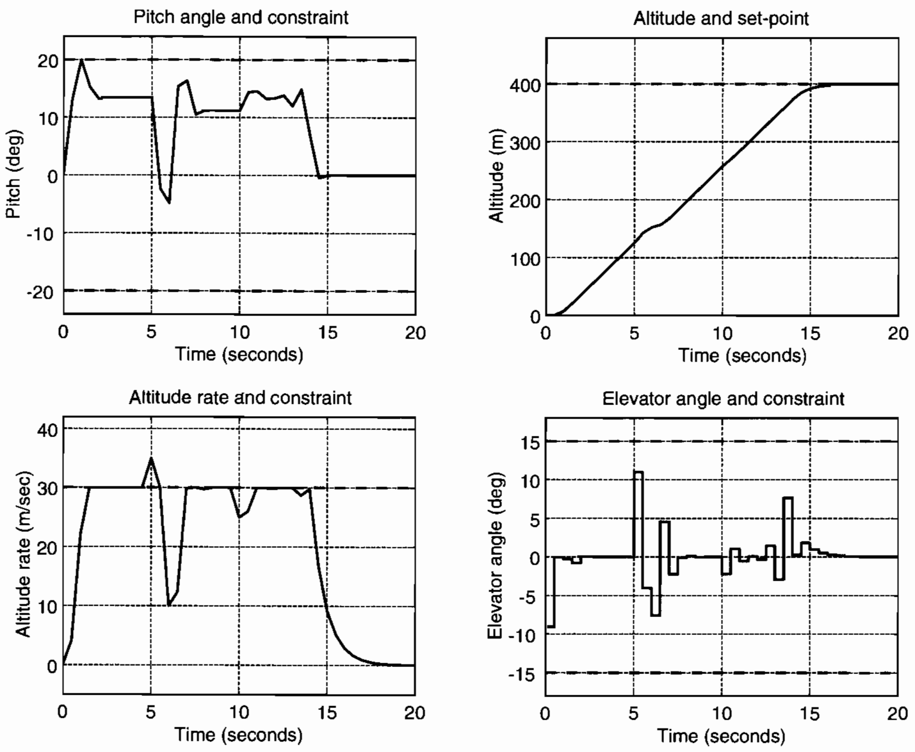

Figure 2.9 Response to disturbance with altitude rate constraint.

corresponds to the constraint of 30 m/sec being imposed on the altitude rate. This constraint is active when the disturbance occurs, causing a violation of the constraint. The controller immediately brings the altitude rate down below the constraint by moving the elevator to $+11°$ for one sample period, which causes the pitch angle to change suddenly from about 13° to $-5°$, before recovering to a steady value of $+11°$. This value of the pitch angle holds the altitude rate at its constraint while the disturbance is present. The disturbance ceases suddenly 10 seconds into the manoeuvre, which has the immediate effect of reducing the altitude rate. Since there is now no danger of the altitude rate constraint being violated, the controller reacts to this in a much less agressive manner than it did to the onset of the disturbance. This nicely illustrates the nonlinear behaviour of the controller.

Figure 2.10, on the other hand, shows what happens when there is a set-point of 30 m/sec for the altitude rate instead of a constraint. The set-points for the pitch angle and altitude are constant, as before, at 0° and 400 m, respectively. In order to give the altitude rate set-point sufficient weight, compared with the altitude set-point, the tracking error weights $Q(i)$ were chosen to be $Q(i) = \text{diag}[1, 1, 100]$ for $i = 1, \ldots, 5$, and $Q(i) = I_3$ for $i = 6, \ldots, 10$ — recall that $H_p = 10$. It can be seen that the response to the disturbance is now much milder. Although the altitude rate remains above 30 m/sec for a little longer than in the previous case, overall it remains closer

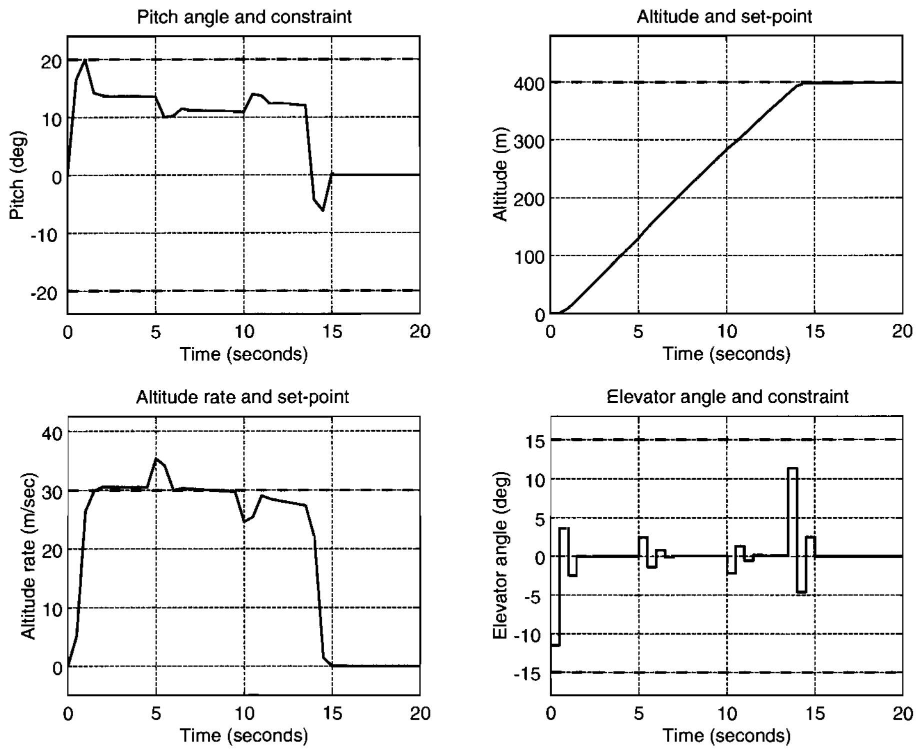

Figure 2.10 Response to disturbance with altitude rate set-point.

to this value than in the previous case, because there is no big dip below 30 m/sec, since the elevator action is now much smaller. It can also be seen that the responses to the start and end of the disturbance pulse are now much more symmetric, as a result of the fact that the controller is now operating in linear mode. The fact that at the end of the manoeuvre there is a more violent transient than in the previous case is an unimportant artefact of the choice of weights and set-point specifications, and could be removed by more careful tuning.

For further discussion of possible specifications of closed-loop behaviours see Section 5.5 and Chapter 7.

Now suppose that the gain from the elevator angle to the altitude rate is mis-modelled, so that the model used by the predictive controller underestimates this gain by 10%. (That is, elements $A(4, 1)$, $A(4, 2)$, $C(3, 1)$, $C(3, 2)$ are 10% smaller in the model than in the plant.) Figure 2.11 shows the response to an altitude set-point change of 400 m, when the altitude rate is constrained to 30 m/sec, and when the constant output disturbance assumption is made on each output. Comparing the response with Figure 2.8 shows that the response is now a little more erratic, with more elevator activity. The altitude rate constraint is violated briefly about 1.5 sec into the transient (by about 0.5 m/sec), but after that it is not violated again. The required altitude is acquired and held without any error in the steady state.

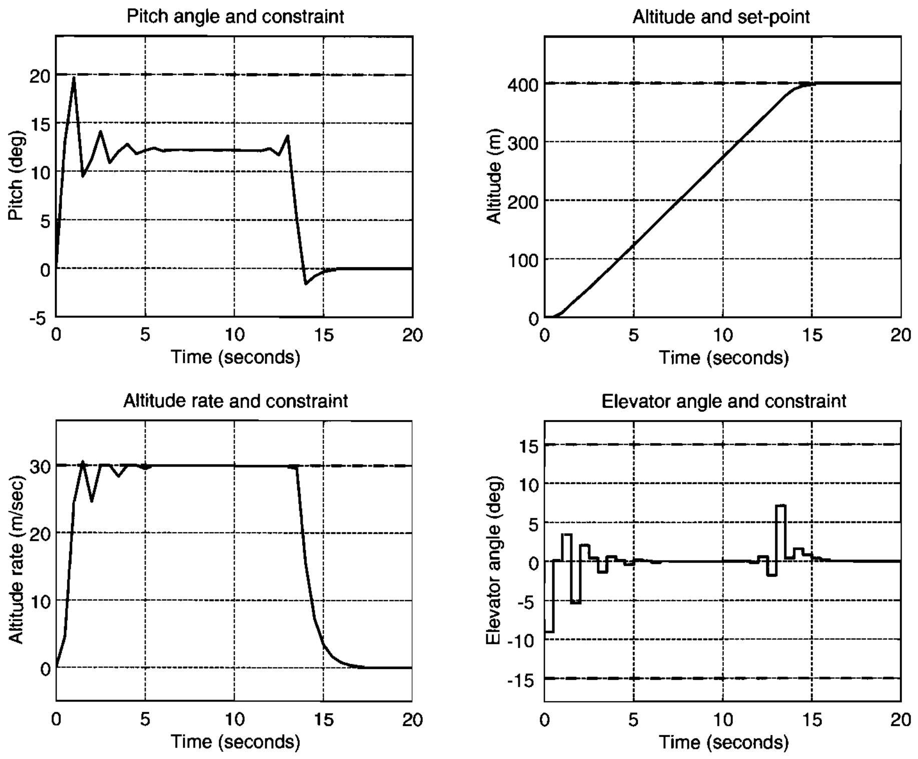

Figure 2.11 Response of Citation aircraft to 400 m step change in altitude set-point, with altitude rate constraint and plant–model error.

Exercises

The first three exercises can be skipped by those familiar with quadratic forms, ∇V, etc.

2.1 Use *MATLAB* to find the eigenvalues of the matrix Q which appears in Example 2.1. Check that they are both positive.

2.2 (a) Suppose $V(x) = 9x_1^2 + 25x_2^2 + 16x_3^2$. Find Q such that $V(x) = x^T Qx$.

(b) Suppose $V(x,u) = (5x_1^2 + 2x_2^2 + x_3^2) + (100u_1^2 + 4u_2^2)$. Find Q and R such that $V(x) = x^T Qx + u^T Ru$.

(c) Are the functions V which appear in this problem positive-definite?

2.3 Check that the formula (2.19) for the gradient is correct for Example 2.1, by working out the partial derivatives in the elementary way, and comparing the results.

2.4 Find the matrix F corresponding to the constraints given in Example 2.4.

2.5 Suppose that we have a 1-state, 1-input model with 1 controlled variable:

$$\hat{x}(k+1|k) = 2\hat{x}(k|k-1) + u(k)$$
$$\hat{z}(k|k) = 3\hat{x}(k|k)$$

and we have the constraint

$$-1 \leq z(k) \leq 2 \quad \text{for all } k$$

If $\hat{x}(k|k) = 3$ and $u(k-1) = -1$, show that the corresponding constraint on $\Delta u(k)$ is

$$-\frac{16}{3} \leq \Delta u(k) \leq -\frac{13}{3}$$

2.6 As practice at using *MATLAB*, and a little revision of discrete-time systems, check the discretization done in Example 2.4. You can create a state-space system object by a command of the form: `cont_sys=ss(Ac,Bc,Cc,Dc);` where `Ac` etc. are the continuous time state-space matrices. You can discretize it using `disc_sys=c2d(cont_sys,Ts);` where `Ts` is the sample interval. Matrices can be extracted from these system objects by commands of the form `Ad=get(disc_sys,'a')`. (Use `help lti` for more information.)

Is the system stable? With the default discretization method used by `c2d` each eigenvalue λ of A_c should be mapped to $\exp(\lambda T_s)$. Check this (using the functions `eig` and `exp`).

2.7 Suppose that in Example 2.4 the feed tank level H_1 is constrained, to ensure that the feed tank neither runs dry nor overfills (in addition to the other constraints).

(a) Explain why the matrix C_z in the basic formulation must be changed to represent this. What is the new value of this matrix?

(b) What are the new dimensions of the matrices E, F and G in this case?

(c) Do you anticipate any additional difficulty with this case? (*Hint:* Compare the matrices C_y and C_z.)

2.8 Referring to Section 2.4, show that a third possible state-space representation is obtained by taking the augmented state vector to be

$$\xi(k) = \begin{bmatrix} \Delta x(k) \\ y(k-1) \\ z(k-1) \end{bmatrix} \tag{2.107}$$

in which case the plant–model equations become

$$\xi(k+1) = \begin{bmatrix} A & 0 & 0 \\ C_y & I & 0 \\ C_z & 0 & I \end{bmatrix} \xi(k) + \begin{bmatrix} B \\ 0 \\ 0 \end{bmatrix} \Delta u(k) \tag{2.108}$$

$$\begin{bmatrix} y(k) \\ z(k) \end{bmatrix} = \begin{bmatrix} C_y & I & 0 \\ C_z & 0 & I \end{bmatrix} \xi(k) \tag{2.109}$$

2.9 Write a *MATLAB* function 'c2dd' to obtain the discrete-time model in the form (2.58) from a continuous-time model in the form (2.45) when there is a computational delay, using the *Control System Toolbox* function c2d.

2.10 For the linearized model of the Citation aircraft defined in Section 2.7:

(a) Obtain discrete-time models of the aircraft, assuming a sampling interval of 0.1 second, with

(i) no computational delay,

(ii) a computational delay of 0.02 second (using the function c2dd referred to in Exercise 2.9).

(b) Compare the step responses from the elevator angle to the three outputs for the two models.

(c) Compare the frequency responses from the elevator angle to the three outputs for the two models. Comment on the implications, for control, of neglecting the computational delay.

(The function `makecita`, available on this book's web site, creates the continuous-time Citation model.)

2.11 Using equations (2.67)–(2.69), or (2.70), write the predictions $\hat{z}(k+i|k)$ in matrix-vector form (analogously to (2.66)).

2.12 Suppose that $D_z \neq 0$ in (2.4). Introduce the new vector of controlled variables $\tilde{z}(k) = z(k) - D_z u(k)$. Show that if the cost function and linear inequality constraints are rewritten in terms of $\tilde{z}$ instead of z then the predictive control problem remains in the standard form, namely the cost function remains quadratic in $\Delta\hat{u}$ and the constraints remain as linear inequalities involving $\Delta\hat{u}$.

2.13 Suppose that $D_z \neq 0$ in (2.4), but that the change of variable $\tilde{z}(k) = z(k) - D_z u(k)$ is not made. How does the computation of $\hat{z}(k+i|k)$ need to be modified from (2.67)–(2.69)? Can you find an expression for the vector $[\hat{z}(k+1|k), \dots, \hat{z}(k+H_p|k)]^T$ in matrix-vector form? (Note that the predictions $\hat{z}$ remain linear in the $\hat{x}$ and $\Delta\hat{u}$ variables.)

2.14 Show that the pair

$$\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}, \qquad [C_y, \quad I]$$

is observable if and only if the pair (A, C_y) is observable.

(The significance is that if this pair is observable then an observer gain matrix L can always be found which places the eigenvalues of the 'constant output' observer (2.84) at arbitrary locations.)

2.15 A plant with 1 input, 1 output and 1 state has the model

$$x(k+1) = 0.9x(k) + 0.5u(k), \qquad y(k) = x(k)$$

Predictive control is to be used with $Q(i) = 1$, $R(i) = 0$, $H_p = 30$, and $H_u = 2$. A constant step disturbance *at the plant input* is assumed (that is, $u(k)$ is replaced by $u(k) + d(k)$).

(a) Show how the plant model can be augmented to incorporate this disturbance model.

(b) Design a state observer with a 'deadbeat' response for the plant with this disturbance model. (The easy way to do this is to use the *Model Predictive Control Toolbox* function `smpcest(imod,Q,R)`, where `Q` and `R` are fictitious state and output noise covariances, respectively, with `R` very small. The observer approaches a deadbeat observer as `R` approaches 0.)

(c) Simulate the response of the predictive controller to an input step disturbance when your observer is used. Compare the response to that obtained when the default 'DMC' observer is used. (Assume there are no constraints.)

(d) How does the observer design affect the set-point response?

(e) Work out the details of how to use the *Control System Toolbox* function `place` to design observers for use with the *Model Predictive Control Toolbox*. (You will need to pay attention to the model representations used by the *Model Predictive Control Toolbox*.)

2.16 Using the representation (2.82)–(2.83) show that the plant output estimated by any asymptotically stable observer converges to the measured plant output, if that is constant (assuming that the plant input and state are both constant).

Chapter 3

Solving predictive control problems

This chapter explains how the standard predictive control problem can be solved. It also discusses the structure of the controller which results. *Active set* and *interior point* methods of solving the optimization problems which arise in predictive control are introduced, and the very important topic of *constraint softening* is treated.

3.1 Unconstrained problems

3.1.1 Measured state, no disturbances

Recall that the cost function which we must minimize is

$$V(k) = \sum_{i=H_w}^{H_p} ||\hat{z}(k+i|k) - r(k+i)||^2_{Q(i)} + \sum_{i=0}^{H_u-1} ||\Delta\hat{u}(k+i|k)||^2_{R(i)} \tag{3.1}$$

We can rewrite this as

$$V(k) = ||\mathcal{Z}(k) - \mathcal{T}(k)||^2_{\mathcal{Q}} + ||\Delta\mathcal{U}(k)||^2_{\mathcal{R}} \tag{3.2}$$

where

$$\mathcal{Z}(k) = \begin{bmatrix} \hat{z}(k+H_w|k) \\ \vdots \\ \hat{z}(k+H_p|k) \end{bmatrix} \quad \mathcal{T}(k) = \begin{bmatrix} \hat{r}(k+H_w|k) \\ \vdots \\ \hat{r}(k+H_p|k) \end{bmatrix}$$

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

and the weighting matrices $\mathcal{Q}$ and $\mathcal{R}$ are given by

$$\mathcal{Q} = \begin{bmatrix} Q(H_w) & 0 & \cdots & 0 \\ 0 & Q(H_w+1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q(H_p) \end{bmatrix} \tag{3.3}$$

$$\mathcal{R} = \begin{bmatrix} R(0) & 0 & \cdots & 0 \\ 0 & R(1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R(H_u-1) \end{bmatrix} \tag{3.4}$$

Also recall from (2.66) and (2.70) — and from Exercises 2.11 and 2.13 of Chapter 2 — that $\mathcal{Z}$ has the form

$$\mathcal{Z}(k) = \Psi x(k) + \Upsilon u(k-1) + \Theta\Delta\mathcal{U}(k) \tag{3.5}$$

for suitable matrices Ψ, Υ and Θ. Define

$$\mathcal{E}(k) = \mathcal{T}(k) - \Psi x(k) - \Upsilon u(k-1) \tag{3.6}$$

This can be thought of as a 'tracking error', in the sense that it is the difference between the future target trajectory and the 'free response' of the system, namely the response that would occur over the prediction horizon if no input changes were made — that is, if we set $\Delta\mathcal{U}(k) = 0$. And if $\mathcal{E}(k)$ really were 0, then it would indeed be correct to set $\Delta\mathcal{U}(k) = 0$. Now we can write

$$V(k) = ||\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)||^2_{\mathcal{Q}} + ||\Delta\mathcal{U}(k)||^2_{\mathcal{R}} \tag{3.7}$$

$$= [\Delta\mathcal{U}(k)^T\Theta^T - \mathcal{E}(k)^T]\mathcal{Q}[\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)] + \Delta\mathcal{U}(k)^T\mathcal{R}\Delta\mathcal{U}(k) \tag{3.8}$$

$$= \mathcal{E}(k)^T\mathcal{Q}\mathcal{E}(k) - 2\Delta\mathcal{U}(k)^T\Theta^T\mathcal{Q}\mathcal{E}(k) + \Delta\mathcal{U}(k)^T[\Theta^T\mathcal{Q}\Theta + \mathcal{R}]\Delta\mathcal{U}(k) \tag{3.9}$$

But this has the form

$$V(k) = const - \Delta\mathcal{U}(k)^T\mathcal{G} + \Delta\mathcal{U}(k)^T\mathcal{H}\Delta\mathcal{U}(k) \tag{3.10}$$

where

$$\mathcal{G} = 2\Theta^T\mathcal{Q}\mathcal{E}(k) \tag{3.11}$$

and

$$\mathcal{H} = \Theta^T \mathcal{Q} \Theta + \mathcal{R}, \tag{3.12}$$

and neither $\mathcal{G}$ nor $\mathcal{H}$ depends on $\Delta\mathcal{U}(k)$.

To find the optimal $\Delta\mathcal{U}(k)$ we can now find the gradient of $V(k)$ and set it to zero. From (3.10) we find

$$\nabla_{\Delta\mathcal{U}(k)} V = -\mathcal{G} + 2\mathcal{H}\Delta\mathcal{U}(k) \tag{3.13}$$

so the optimal set of future input moves is

$$\boxed{\Delta\mathcal{U}(k)_{opt} = \frac{1}{2}\mathcal{H}^{-1}\mathcal{G}} \tag{3.14}$$

Remember that we use only the part of this solution corresponding to the first step, in accordance with the receding horizon strategy. So if the number of plant inputs is ℓ then we just use the first ℓ rows of the vector $\Delta\mathcal{U}(k)_{opt}$. We can represent this as

$$\Delta u(k)_{opt} = [I_\ell, \underbrace{0_\ell, \ldots, 0_\ell}_{(H_u-1)\text{ times}}]\Delta\mathcal{U}(k)_{opt} \tag{3.15}$$

where I_ℓ is the $\ell \times \ell$ identity matrix, and 0_ℓ is the $\ell \times \ell$ zero matrix.

Note that we can write $\Delta u(k)_{opt}$ here, rather than $\hat{u}(k|k)_{opt}$, because we have now found the solution and this is the input that is *really* applied to the plant at time k.

Does $\Delta\mathcal{U}(k)_{opt}$ really give a minimum of the cost function V? It certainly gives a stationary point, but that is not enough to guarantee a minimum. Differentiating the gradient $\nabla_{\Delta\mathcal{U}(k)} V$ (3.13) again with respect to $\Delta\mathcal{U}(k)$ gives the matrix of second derivatives, or *Hessian*, of V:

$$\frac{\partial^2 V}{\partial \Delta\mathcal{U}(k)^2} = 2\mathcal{H} = 2(\Theta^T \mathcal{Q}\Theta + \mathcal{R}). \tag{3.16}$$

We have assumed that $Q(i) \geq 0$ for each i, and this ensures that $\Theta^T \mathcal{Q}\Theta \geq 0$. So if $\mathcal{R} > 0$ then the Hessian is certainly positive-definite, which is enough to guarantee that we have a minimum. And this will be the case if $R(i) > 0$ for each i.

But sometimes we may want to have no penalty on the input moves, which would lead to $\mathcal{R} = 0$. Or we may want to leave the moves of some inputs unpenalized, or the moves at some points in the control horizon unpenalized. In these cases we will have $\mathcal{R} \geq 0$, but not $\mathcal{R} > 0$. When $\mathcal{R} = 0$ we need $\Theta^T \mathcal{Q}\Theta > 0$ in order to have a minimum — and of course, to ensure that $\mathcal{H}^{-1}$ exists. In intermediate cases, when $\mathcal{R} \geq 0$, we need to ensure that $\Theta^T \mathcal{Q}\Theta + \mathcal{R} > 0$.

Let us check the dimensions of the various matrices introduced in this section. Θ has as many rows as there are elements in $\mathcal{Z}$. If we have p controlled outputs, and we predict between steps H_w and H_p, then there are $p(H_p - H_w + 1)$ of these elements. The number of columns in Θ is the same as the number of elements in $\Delta\mathcal{U}(k)$, which is ℓH_u. $Q(i)$ has m rows and columns, so $\mathcal{Q}$ has $m(H_p - H_w + 1)$ of each. Thus $\mathcal{H} = \Theta^T \mathcal{Q}\Theta$ is square, with ℓH_u rows and columns. This agrees (as it must!) with the number of rows and columns in $\mathcal{R}$. Similar checks on the remaining matrices lead to Table 3.1.

Table 3.1 Dimensions of matrices and vectors involved in computing the optimal input moves. (The plant has ℓ inputs, n states, and m controlled outputs.)

Matrix	Dimensions
$\mathcal{Q}$	$m(H_p - H_w + 1) \times m(H_p - H_w + 1)$
$\mathcal{R}$	$\ell H_u \times \ell H_u$
Ψ	$m(H_p - H_w + 1) \times n$
Υ	$m(H_p - H_w + 1) \times \ell$
Θ	$m(H_p - H_w + 1) \times \ell H_u$
$\mathcal{E}$	$m(H_p - H_w + 1) \times 1$
$\mathcal{G}$	$\ell H_u \times 1$
$\mathcal{H}$	$\ell H_u \times \ell H_u$

3.1.2 Formulation as a least-squares problem

The optimal solution, as expressed in (3.14), should *never* be obtained by computing the inverse of $\mathcal{H}$. The matrix Θ is often ill-conditioned, which can result in $\mathcal{H}$ being ill-conditioned. It is therefore imperative to pay attention to the numerical algorithms involved in finding the optimal solution.

The best way of computing the solution is by solving it as a 'least-squares' problem. It is also a way which gives some additional insight.

Since $\mathcal{Q} \geq 0$ and $\mathcal{R} \geq 0$, we can find matrices $S_\mathcal{Q}$ and $S_\mathcal{R}$ which are their 'square-roots':

$$S_\mathcal{Q}^T S_\mathcal{Q} = \mathcal{Q} \qquad\qquad S_\mathcal{R}^T S_\mathcal{R} = \mathcal{R}$$

If $\mathcal{Q}$ and $\mathcal{R}$ are diagonal, it is trivial to do this — just take the square-root of each diagonal element. If they are not diagonal, square-roots can be obtained by using the 'Cholesky' algorithm (function `chol` in *MATLAB*) for positive-definite matrices, or by using other algorithms, such as singular value decomposition (`svd`) for semi-definite matrices.

Now consider the vector

$$\begin{bmatrix} S_\mathcal{Q}\{\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)\} \\ S_\mathcal{R}\Delta\mathcal{U}(k) \end{bmatrix}$$

We shall show that the squared 'length' of this vector, or equivalently, the sum of squares of its elements, is the same as the cost function $V(k)$, so that $\Delta\mathcal{U}(k)_{opt}$ is the value of $\Delta\mathcal{U}(k)$ which minimizes this length.

$$\left\| \begin{bmatrix} S_\mathcal{Q}\{\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)\} \\ S_\mathcal{R}\Delta\mathcal{U}(k) \end{bmatrix} \right\|^2 = \left\| \begin{bmatrix} S_\mathcal{Q}\{\mathcal{Z}(k) - \mathcal{T}(k)\} \\ S_\mathcal{R}\Delta\mathcal{U}(k) \end{bmatrix} \right\|^2 \tag{3.17}$$

$$= [\mathcal{Z}(k) - \mathcal{T}(k)]^T S_\mathcal{Q}^T S_\mathcal{Q}[\mathcal{Z}(k) - \mathcal{T}(k)] + \Delta\mathcal{U}(k)^T S_\mathcal{R}^T S_\mathcal{R}\Delta\mathcal{U}(k) \tag{3.18}$$

$$= ||\mathcal{Z}(k) - \mathcal{T}(k)||_\mathcal{Q}^2 + ||\Delta\mathcal{U}(k)||_\mathcal{R}^2 \tag{3.19}$$

$$= V(k) \tag{3.20}$$

So $\Delta\mathcal{U}(k)_{opt}$ is the 'least-squares' solution of the equation

$$\begin{bmatrix} \mathcal{S}_Q\{\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)\} \\ \mathcal{S}_\mathcal{R}\Delta\mathcal{U}(k) \end{bmatrix} = 0 \tag{3.21}$$

or, equivalently, of:

$$\begin{bmatrix} \mathcal{S}_Q\Theta \\ \mathcal{S}_\mathcal{R} \end{bmatrix} \Delta\mathcal{U}(k) = \begin{bmatrix} \mathcal{S}_Q\mathcal{E}(k) \\ 0 \end{bmatrix} \tag{3.22}$$

Equations of the form $A\theta = b$ can be solved in a least-squares sense using the '*QR*' algorithm. In *MATLAB* this solution is obtained as $\theta_{opt} = A\backslash b$. Although formally this solution is the same as $\theta_{opt} = (A^T A)^{-1} A^T b$ (which gives (3.14)), this algorithm avoids 'squaring up' A, and never forms the product $A^T A$, or computes its inverse. If A is ill-conditioned and/or large this is crucial, since it avoids unnecessary loss of precision. For more details see any book on numerical linear algebra, such as [GL89], or Chapter 8 of [Mac89] (or the *MATLAB* documentation).

Hence we have, using *MATLAB* notation:

$$\Delta\mathcal{U}(k)_{opt} = \begin{bmatrix} \mathcal{S}_Q\Theta \\ \mathcal{S}_\mathcal{R} \end{bmatrix} \Big\backslash \begin{bmatrix} \mathcal{S}_Q\mathcal{E}(k) \\ 0 \end{bmatrix} \tag{3.23}$$

Equation (3.22) is almost always over-determined: there are not enough degrees of freedom to get an exact solution. It contains $m(H_p - H_w + 1) + \ell H_u$ scalar equations, but there are only ℓH_u variables to be solved for. Even in the special case of $\mathcal{R} = 0$, when there is no penalty on input moves, and the equation simplifies to

$$\mathcal{S}_Q\Theta\Delta\mathcal{U}(k) = \mathcal{S}_Q\mathcal{E}(k) \tag{3.24}$$

there are usually more scalar equations than variables. A unique exact solution exists only if the matrix $\mathcal{S}_Q\Theta$ is square and non-singular, which requires $m(H_p - H_w + 1) = \ell H_u$. However, we usually have $m(H_p - H_w + 1) > \ell H_u$. Note that in the case of $\mathcal{R} = 0$ we cannot remove $\mathcal{S}_Q$ from each side of the equation — doing so would give a different solution, since it would change the weighting used when minimizing the sum of squares of the error.

Why do we expect ill-conditioning in the predictive control problem? Typically it arises in the following way. Suppose we had two controlled variables which behaved exactly the same as each other. Then the corresponding pairs of rows of Ψ would be identical. Consequently the rank of Θ (the number of linearly independent rows) would be smaller than expected from its dimensions by $H_p - H_w + 1$ — that is how many pairs of identical rows there would be. If it were sufficiently smaller, the rank could become smaller than ℓH_u, at which point $\Theta^T Q\Theta$ would become singular. If we also had $\mathcal{R} = 0$, then $\mathcal{H}$ would be singular. In practice variables do not behave exactly the same as each other. But sometimes they behave very similarly. Consider a distillation column with 40 trays, for example. Adjacent trays do not behave identically, but obviously if you can manipulate only a few temperatures in the column, adjacent trays are going to react very similarly. In such cases one can have many rows of Θ being 'nearly linearly dependent' on each other, and then $\mathcal{H}$ can be nearly singular. Even without this cause,

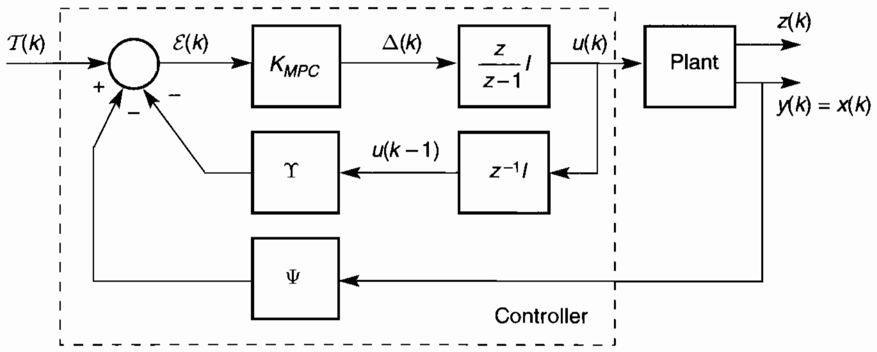

Figure 3.1 Structure of controller with no constraints and full state measurement.

the rows near the bottom of Θ are often 'nearly linearly dependent' if $H_p - H_w \gg H_u$. One remedy for this is to increase the diagonal elements of $\mathcal{R}$ — but that may distort the problem away from the real one [QB96].

3.1.3 Structure of the unconstrained controller

Recall from (3.15) and (3.6) that

$$\Delta u(k)_{opt} = [I_\ell, 0_\ell, \dots, 0_\ell]\mathcal{H}^{-1}\Theta^T\mathcal{Q}\mathcal{E}(k) \tag{3.25}$$

and

$$\mathcal{E}(k) = \mathcal{T}(k) - \Psi x(k) - \Upsilon u(k-1). \tag{3.26}$$

The only part of this solution which changes from step to step is the 'tracking error' $\mathcal{E}(k)$. Consequently the predictive controller, for the unconstrained problem and with full state measurement, can be drawn as in Figure 3.1.

The block labelled K_{MPC} is defined by

$$K_{MPC} = [I_\ell, 0_\ell, \dots, 0_\ell]\mathcal{H}^{-1}\Theta^T\mathcal{Q} \tag{3.27}$$

We point out that the 'correct' way of computing K_{MPC} is (again using *MATLAB* notation, including the ' : ' operator to pick out the first ℓ rows of the solution):

$$K_{full} = \begin{bmatrix} S_\mathcal{Q}\Theta \\ S_\mathcal{R} \end{bmatrix} \Big\backslash \begin{bmatrix} S_\mathcal{Q} \\ 0 \end{bmatrix} \tag{3.28}$$

$$K_{MPC} = K_{full}(1 : \ell, :) \tag{3.29}$$

(This works for the following reason: If we had $\mathcal{E}(k) = [1, 0, \dots, 0]^T$ then we would effectively have only the first column of $S_\mathcal{Q}$ on the right-hand side of (3.23). $\mathcal{E}(k) = [0, 1, 0, \dots, 0]^T$ would effectively give the second column, etc. Since $\mathcal{E}(k)$ enters the solution linearly, it is only necessary to solve (3.23) with these columns on the right-hand side. The '\' operator in *MATLAB* is smart enough to solve for all the columns simultaneously — and this is very efficient, since most of the work involved is in computing the QR decomposition of the left-hand side, which needs to be done only once.)

It is clear from the figure that the controller is a linear time-invariant system in this case. So it is possible to compute its frequency response, stability margins etc. We shall do that kind of analysis in a later chapter. Note that the controller is, in general, dynamic. It is not just 'state feedback', except under very special circumstances.

The matrix K_s computed by the *Model Predictive Control Toolbox* function smpccon is related to K_{MPC} as follows:

$$K_s = K_{MPC} \left[\begin{bmatrix} I \\ I \\ \vdots \\ I \end{bmatrix}, -\Psi, -\Upsilon \right] \tag{3.30}$$

so that

$$\Delta u(k)_{opt} = K_s \begin{bmatrix} \mathcal{T}(k) \\ x(k) \\ u(k-1) \end{bmatrix} \tag{3.31}$$

3.1.4 Estimated state

Now we need to address the more realistic case, when we do not have measurements of the whole state vector, and must use an observer. We will see that the solution is very similar to the solution in the previous case. In fact, we will see that all the gain matrices involved are exactly the same as in the previous case. The only difference is that we need to use an observer, and use the state estimate $\hat{x}(k|k)$ to replace the measured state $x(k)$. The controller structure in this case is shown in Figure 3.2. This is again a linear time-invariant system, but now with more dynamic complexity, because its state vector includes the observer state.

To obtain the vector of predicted controlled outputs, $\mathcal{Z}(k)$, it is reasonable to go back to equation (3.5), and simply replace $x(k)$ by the best estimate of it available to us, namely $\hat{x}(k|k)$. We say 'reasonable' because it is not self-evident that this is the 'optimal' thing to do in any sense. The *Separation Principle* or *Certainty Equivalence Principle* says that it *is* the optimal thing to do if one is solving a stochastic linear quadratic problem, and one has Gaussian noises acting on the states and outputs, and the observer gain L' is obtained using Kalman filtering theory [AM79, BH75, BGW90]. But in general it is just a heuristic, albeit one which is very widely used in control engineering. After all, no other obvious alternative presents itself.

So now we define

$$\mathcal{Z}(k) = \Psi\hat{x}(k|k) + \Upsilon u(k-1) + \Theta\Delta\mathcal{U}(k) \tag{3.32}$$

and we make the corresponding change in the definition of the 'tracking error' $\mathcal{E}(k)$ (compare with equation (3.6)):

$$\mathcal{E}(k) = \mathcal{T}(k) - \Psi\hat{x}(k|k) - \Upsilon u(k-1) \tag{3.33}$$

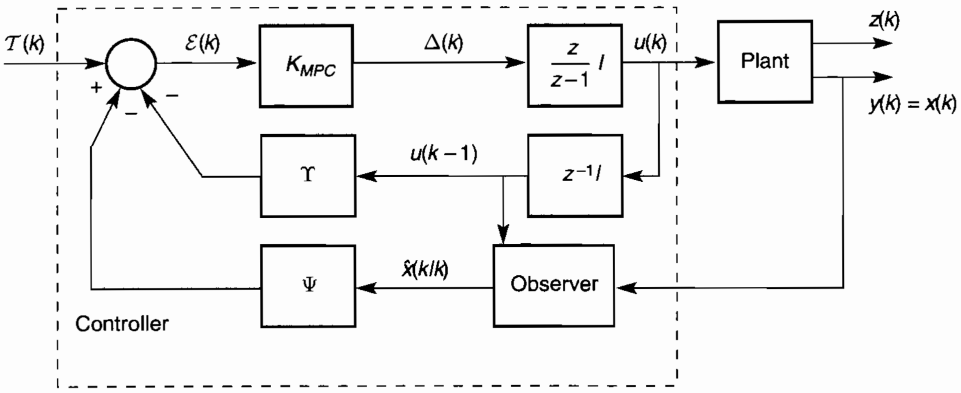

Figure 3.2 Structure of controller with no constraints and state observer.

Once these changes have been made, the derivation of the optimal control $\Delta u(k)_{opt}$ is exactly the same as in earlier sections. Hence we arrive at the controller structure shown in Figure 3.2.

3.2 Constrained problems

3.2.1 Formulation as a QP problem

Now we deal with the case when constraints are present. Recall that these are in the form:

$$E\begin{bmatrix}\Delta\mathcal{U}(k)\\1\end{bmatrix} \le 0 \tag{3.34}$$

$$F\begin{bmatrix}\mathcal{U}(k)\\1\end{bmatrix} \le 0 \tag{3.35}$$

$$G\begin{bmatrix}\mathcal{Z}(k)\\1\end{bmatrix} \le 0 \tag{3.36}$$

where $\mathcal{U}(k) = [\hat{u}(k|k)^T, \dots, \hat{u}(k+H_u-1|k)^T]^T$ is defined analogously to $\Delta\mathcal{U}(k)$. We have to express all of these as constraints on $\Delta\mathcal{U}(k)$.

Suppose F has the form

$$F = [F_1, F_2, \cdots, F_{H_u}, f]$$

where each F_i is of size $q \times m$, and f has size $q \times 1$, so that (3.35) can be written as

$$\sum_{i=1}^{H_u} F_i\hat{u}(k+i-1|k) + f \le 0.$$

Since

$$\hat{u}(k+i-1|k) = u(k-1) + \sum_{j=0}^{i-1} \Delta\hat{u}(k+j|k)$$

we can write (3.35) as

$$\sum_{j=1}^{H_u} F_j \Delta\hat{u}(k|k) + \sum_{j=2}^{H_u} F_j \Delta\hat{u}(k+1|k) + \ldots + F_{H_u}\Delta\hat{u}(k+H_u-1|k)$$
$$+ \sum_{j=1}^{H_u} F_j u(k-1) + f \leq 0$$

Now define $\mathcal{F}_i = \sum_{j=i}^{H_u} F_j$ and $\mathcal{F} = [\mathcal{F}_1, \ldots, \mathcal{F}_{H_u}]$. Then (3.35) can be written as

$$\mathcal{F}\Delta\mathcal{U}(k) \leq -\mathcal{F}_1 u(k-1) - f \tag{3.37}$$

where the right-hand side of the inequality is a vector, which is known at time k. So we have converted (3.35) into a linear inequality constraint on $\Delta\mathcal{U}(k)$.

Note that if we have simple range constraints on the inputs, of the form

$$u_{low}(k+i) \leq \hat{u}(k+i|k) \leq u_{high}(k+i) \tag{3.38}$$

then this inequality takes quite a simple form. (See Exercise 3.6.)

Now we have to do a similar thing for (3.36). Fortunately we have already done most of the work needed in this case. Assuming full state measurements, we can use (3.5) to write (3.36) as

$$G\begin{bmatrix} \Psi x(k) + \Upsilon u(k-1) + \Theta\Delta\mathcal{U}(k) \\ 1 \end{bmatrix} \leq 0$$

Now letting $G = [\Gamma, g]$, where g is the last column of G, this is the same as

$$\Gamma[\Psi x(k) + \Upsilon u(k-1)] + \Gamma\Theta\Delta\mathcal{U}(k) + g \leq 0$$

or

$$\Gamma\Theta\Delta\mathcal{U}(k) \leq -\Gamma[\Psi x(k) + \Upsilon u(k-1)] - g \tag{3.39}$$

which is in the required form.

If we have only state estimates available, then we replace $x(k)$ by $\hat{x}(k|k)$, just as we did in Section 3.1.4.

It only remains to put inequality (3.34) into the form

$$W\Delta\mathcal{U}(k) \leq w \tag{3.40}$$

(see Exercise 3.6). Then we can assemble inequalities (3.37), (3.39), and (3.40) into the single inequality

$$\begin{bmatrix} \mathcal{F} \\ \Gamma\Theta \\ W \end{bmatrix} \Delta\mathcal{U}(k) \leq \begin{bmatrix} -\mathcal{F}_1 u(k-1) - f \\ -\Gamma[\Psi x(k) + \Upsilon u(k-1)] - g \\ w \end{bmatrix} \tag{3.41}$$

(Replace $x(k)$ by $\hat{x}(k|k)$ if an observer is used.)

Now the cost function $V(k)$ which we have to minimize is still the same as in the unconstrained case. So, from (3.10), we see that we have to solve the following constrained optimization problem:

$$\text{minimize } \Delta\mathcal{U}(k)^T \mathcal{H} \Delta\mathcal{U}(k) - \mathcal{G}^T \Delta\mathcal{U}(k) \tag{3.42}$$

subject to the inequality constraint (3.41). But this has the form

$$\min_{\theta} \frac{1}{2}\theta^T \Phi \theta + \phi^T \theta \tag{3.43}$$

subject to

$$\Omega\theta \leq \omega \tag{3.44}$$

which is a standard optimization problem known as the *Quadratic Programming* (or *QP*) problem, and standard algorithms are available for its solution.

Similarly to solving the unconstrained problem, it is better to pass the *QP* problem to a solution algorithm in 'square root' form, namely in the form

$$\min_{\Delta\mathcal{U}(k)} \left\| \begin{bmatrix} S_{\mathcal{Q}}\{\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)\} \\ S_{\mathcal{R}}\Delta\mathcal{U}(k) \end{bmatrix} \right\|^2 \quad \text{subject to (3.41).} \tag{3.45}$$

Since $\mathcal{H} \geq 0$, the *QP* problem which we have to solve is *convex*. This is extremely good news — see Mini-Tutorial 3. Because of the convexity we can guarantee termination of the optimization problem, and because of the additional structure of the *QP* problem, we can estimate how long it will take to solve. This is an extremely desirable property for an algorithm which has to be used on-line, and to keep up with the real-time operation of the plant.

A major problem which can occur with constrained optimization is that the problem may be infeasible. Standard *QP* solvers just stop in such cases, their only output being a message such as `Problem Infeasible`, or perhaps some diagnostics in certain cases. This is obviously unacceptable as a substitute for a control signal which must be provided to the plant. So when implementing predictive control it is essential to take steps either to avoid posing an infeasible problem, or to have a 'back-up' method of computing the control signal. Various approaches to this have been suggested, including:

- Avoid 'hard' constraints on z.
- Actively manage the constraint definition at each k.
- Actively manage the horizons at each k.
- Use non-standard solution algorithms.

We shall examine these more closely later in the book.

In general optimization problems are solved numerically by 'going downhill' — assuming a minimization problem — until one reaches a minimum. The big problem with this is that in general problems there are many 'local' minima, and the algorithm is very likely to be stuck in such a local minimum, unaware that the true 'global' minmum is elsewhere.

A *convex* optimization problem is one in which this problem does not occur. Because of the convexity of the objective function, there is only one minimum — or possibly a connected set of equally good minima, as on a flat valley floor. When solving a convex problem, one is guaranteed that a global minimum will eventually be reached if one keeps 'going downhill'.

For a 'smooth' problem the property of convexity can be established from the Hessian of the objective function — it has to be positive semi-definite everywhere. But constrained problems are usually not smooth, or at least not everywhere. In this case convexity can be characterized without using derivatives, as follows. A function $V(\theta)$ is convex if, for every pair of points θ_1 and θ_2, and any λ such that $0 \leq \lambda \leq 1$, it is always true that

$$\lambda V(\theta_1) + (1-\lambda)V(\theta_2) \geq V(\lambda\theta_1 + [1-\lambda]\theta_2) \tag{3.46}$$

The straight line joining any two points on the cost surface is never below the surface.

A *Quadratic Program* is an optimization problem of the form

$$\min_\theta \frac{1}{2}\theta^T\Phi\theta + \phi^T\theta \qquad \text{subject to} \qquad \Omega\theta \leq \omega$$

Here the objective function is $V(\theta) = \frac{1}{2}\theta^T\Phi\theta + \phi^T\theta$ and its Hessian is Φ. If there are no constraints this is clearly convex if $\Phi \geq 0$. (Without this condition there might be no minimum, since arbitrarily large negative values of V might be attainable.) Since the constraints are linear inequalities, the surfaces on which they are active are hyperplanes. So the constrained objective function can be visualized as a convex quadratic surface, parts of which have been cut off by a number of flat 'faces'. It is intuitively clear that this constrained surface remains convex, although we shall not give a formal proof of this.

A *Linear Program* (or *LP*) is the special case of a *QP* when $\Phi = 0$, so that the objective function is linear rather than quadratic. It is also convex when Ω and ϕ are such that a minimum exists; in this case the minimum always occurs at a vertex (or possibly an edge). The constrained objective surface can be visualized as a convex object with flat faces. Such an object is called a *simplex*. There are standard algorithms for solving large LP problems, including the famous 'simplex method'.

The literature of convex optimization, LP and QP problems is enormous. Good general books on optimization which include relevant material are [Fle87, GMW81]. A whole book on using convex optimization for control design (but which does not deal with predictive control) is [BB91].

Mini-Tutorial 3 Convex optimization, QP and LP problems.

3.2.2 Controller structure

So long as all the constraints are inactive, the solution of the predictive controller is exactly the same as in the unconstrained case. But if constraints become active then the controller becomes nonlinear and the structure shown in Figures 3.1 and 3.2 is lost. Figure 3.3 shows the controller structure in this case. The controller is nonlinear because the box labelled 'Optimizer' computes a nonlinear function of its inputs.

We can say a little more about the controller structure. Suppose a particular set of constraints is active. That is, in the QP problem (3.43)–(3.44), suppose that

$$\Omega_a\theta = \omega_a \tag{3.47}$$

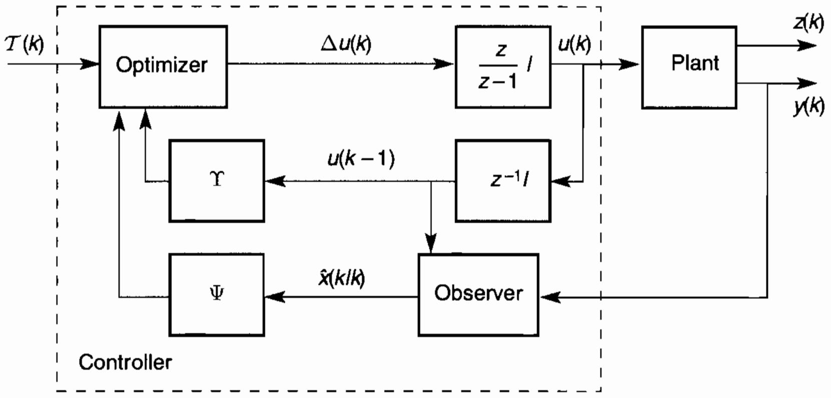

Figure 3.3 Structure of controller with constraints and state observer.

where Ω_a is made up of those rows of Ω which relate to the active constraints, and ω_a is made up of the corresponding elements of ω. *If we knew* before solving the problem that these would be the active constraints, and were therefore equality rather than inequality constraints, then we could (only in principle!) pose the optimization problem

$$\min_{\theta} \frac{1}{2}\theta^T\Phi\theta + \phi^T\theta \qquad \text{subject to} \qquad \Omega_a\theta = \omega_a \tag{3.48}$$

which could, by the theory of Lagrange multipliers, be solved by solving the problem

$$\min_{\theta,\lambda} L(\theta,\lambda) \tag{3.49}$$

where

$$L(\theta,\lambda) = \frac{1}{2}\theta^T\Phi\theta + \phi^T\theta + \lambda(\Omega_a\theta - \omega_a) \tag{3.50}$$

Now

$$\nabla_\theta L(\theta,\lambda) = \Phi\theta + \phi + \Omega_a^T\lambda \tag{3.51}$$

$$\nabla_\lambda L(\theta,\lambda) = \Omega_a\theta - \omega_a \tag{3.52}$$

or

$$\nabla L(\theta,\lambda) = \begin{bmatrix} \Phi & \Omega_a^T \\ \Omega_a & 0 \end{bmatrix}\begin{bmatrix} \theta \\ \lambda \end{bmatrix} - \begin{bmatrix} -\phi \\ \omega_a \end{bmatrix} \tag{3.53}$$

Consequently the optimal solution would be obtained, by setting $\nabla L(\theta,\lambda) = 0$, as

$$\begin{bmatrix} \theta \\ \lambda \end{bmatrix}_{opt} = \begin{bmatrix} \Phi & \Omega_a^T \\ \Omega_a & 0 \end{bmatrix}^{-1}\begin{bmatrix} -\phi \\ \omega_a \end{bmatrix} \tag{3.54}$$

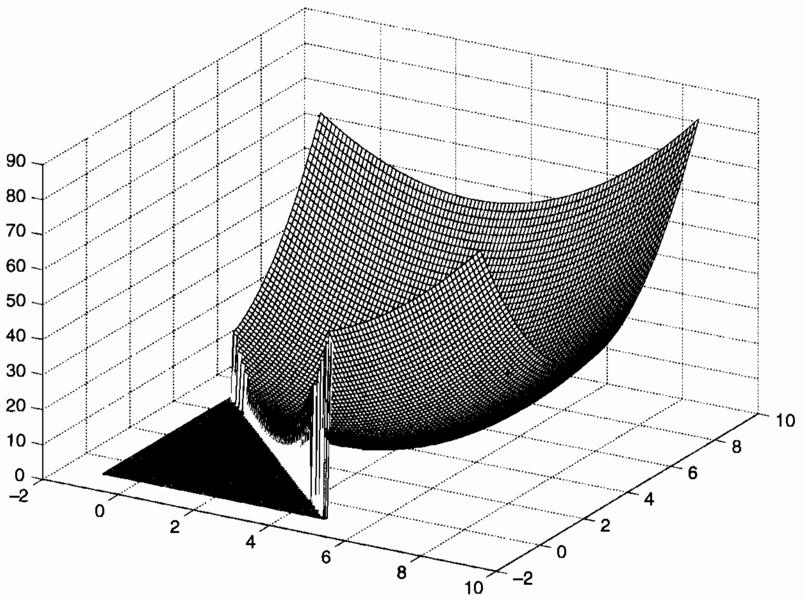

Figure 3.4 A quadratic cost surface and a linear inequality constraint: constraint inactive.

Now recall that

$$\Omega = \begin{bmatrix} \mathcal{F} \\ \Gamma\Theta \\ W \end{bmatrix}$$

and $\Phi = \mathcal{H}/2$. Looking back at the definitions of $\mathcal{F}$, Γ, Θ, W and $\mathcal{H}$, it will be seen that none of these depend on signals at time k. So the matrix which is being inverted here is fixed, so long as a fixed set of constraints is active. On the other hand, $\phi = -\mathcal{G} = -2\Theta^T \mathcal{Q}\mathcal{E}(k)$ clearly does depend on the signals present at time k, and so does ω_a.

We can therefore conclude that the constrained predictive control law is a linear time-invariant control law, *so long as the set of active constraints is fixed.* A more intuitive explanation can be given on the basis of Figures 3.4 and 3.5. These show a quadratic cost function as a surface in the case when there are only two decision variables. In Figure 3.4 a linear inequality constraint restricts the decision variables so that only part of the cost surface is 'attainable', but the unconstrained minimum is in the feasible region, so the optimal solution is at the unconstrained minimum. In Figure 3.5 the constraint is active, so that the optimal solution is on the intersection of the cost surface and the inequality constraint. It can be seen that the optimal solution is still the global minimum of a quadratic surface, but one of lower dimension — in this case, the minimum of a quadratic curve. It is the fact that a quadratic problem is still being solved that leads to the controller being linear.

In practice the set of active constraints changes, so we have the picture of the control law as consisting of a number of linear controllers, each with a structure similar to that

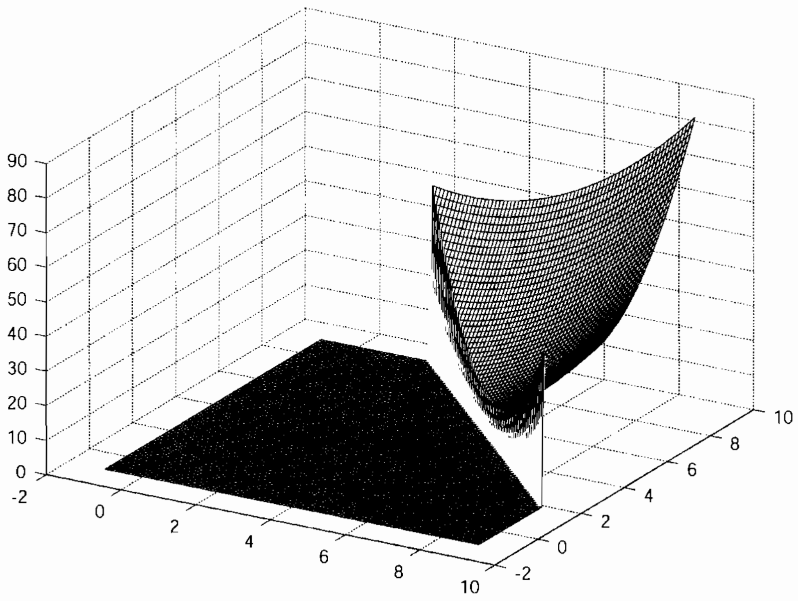

Figure 3.5 A quadratic cost surface and a linear inequality constraint: constraint active.

shown in Figure 3.2, and switching between them. If we could be sure that changes of active constraint sets occurred rarely enough — and there is no reason to suppose that this would be the case in general — we might be able to exploit this structure to perform some analysis of the constrained controller.

In [BMDP, BMDP99] a promising attempt is made not only to use the piecewise-linear nature of the controller as a basis for analysis, but also as a way of computing the optimal solution efficiently for small problems. In this context 'small' means that the number of constraints is such that the number of possible active constraint sets is manageable, since a separate solution is pre-computed explicitly for each such set. Note that the number of possible active constraint sets can be extremely large. If there are q constraints in the problem formulation (Ω has q rows) then there are 2^q possible sets of active constraints. Analysis of specific problems can often reduce the size of this set to some extent — for example, lower and upper limits on variable values cannot be active simultaneously.

The approach taken by Bemporad *et al* in [BMDP] and [BMDP99] is based on the observation that in the MPC problem, the inequality constraint (3.44) takes the form:

$$\Omega\theta \leq \omega(\hat{x}) \tag{3.55}$$

The vector on the right-hand side of the inequality depends on the (estimate of the) current state, which can be thought of as a set of parameters of the QP problem. Techniques of *multiparametric programming* are then employed to analyze the set of solutions. One new result which is obtained by this means is that the control law is *continuous* in $\hat{x}$. Algorithms are also given for determining efficiently the boundaries (in the state-space) at which changes from one piecewise-linear control law to another occur. The idea, then, is that for small problems, in which the state-space is divided up into a manageably small number of (convex) pieces, one could pre-compute (off-line) the control law that should be applied in each piece, and then the MPC algorithm would consist simply of reading the appropriate gain matrix from a table look-up, depending on the current state estimate. This should be particularly useful in high-bandwidth applications, when high control update rates are required.

Although this idea is not feasible for applications in which the number of constraints is large ($q > 10$, say), it may be possible even in large problems to compute the solution by conventional means (as described in the next subsection) when a region of the state space is entered for the first time, but then to store it in a growing database, as the state space is explored, so that if the region is revisited later the solution does not need to be recomputed. Many variations of this can be easily imagined.

3.3 Solving QP problems

In this section we shall consider the general QP problem:

$$\min_{\theta} \frac{1}{2}\theta^T \Phi\theta + \phi^T\theta \qquad (\Phi = \Phi^T \geq 0) \tag{3.56}$$

subject to

$$H\theta = h \tag{3.57}$$

and

$$\Omega\theta \leq \omega \tag{3.58}$$

This is the same as the QP problem introduced earlier, except that the equality constraints $H\theta = h$ have been identified explicitly. We shall assume that it is not feasible to pre-compute the solutions in the manner described at the end of the previous subsection.

Two approaches to solving the QP problem appear to offer the best performance: *Active Set* methods [Fle87], and the more recent *Interior Point* methods [NN94, Wri97, RTV97]. It is tempting to assume that the more recent methods should offer the best performance, but Wright [Wri96] argues that for predictive control this issue is still open. He points out that the predictive control problem has a lot of special structure, and that the main performance gains are to be obtained by exploiting this structure, whichever approach is used. The special structure is primarily due to two features:

1. With a particular ordering of the variables, the QP problem which arises in predictive control is 'sparse'.

2. Most of the time — in the absence of large disturbances, or of set-point changes, etc. — a very good initial guess of the solution is available, from the solution computed at the previous step.

In this section, both the Active Set and the Interior Point methods will be described very briefly, and some of the points at which the structure of the predictive control problem can be exploited will be identified. The presentation is based on [Fle87]and [Wri96].

As a result of the QP problem being convex, the necessary and sufficient conditions for θ to be the global optimum are given by the *Karush–Kuhn–Tucker* (or *KKT*) conditions [Fle87, Wri97]: there must exist vectors (Lagrange multipliers) $\lambda \geq 0$ and ζ, and a vector $t \geq 0$, such that

$$\Phi\theta + H^T\zeta + \Omega^T\lambda = -\phi \tag{3.59}$$
$$-H\theta = -h \tag{3.60}$$
$$-\Omega\theta - t = -\omega \tag{3.61}$$
$$t^T\lambda = 0 \tag{3.62}$$

3.3.1 Active Set methods

The *Active Set* method assumes that a feasible solution is available. (We shall consider later how that might be found.) For such a solution, the equality constraints (3.57) are of course satisfied, and some subset of the inequality constraints (3.58) is active, namely the constraints hold with equality for this subset. This subset is called the *active set*; it is possible for it to be empty, if none of the inequality constraints is active at the current feasible solution. Using the same notation as earlier, we use the subscript a to denote those rows of (3.58) which are in the active set, so that

$$\Omega_a\theta = \omega_a$$

Suppose that at the rth iteration we have the feasible solution θ_r. The Active Set method then finds an improved solution $\theta_r + \Delta\theta$ which minimizes the cost (3.56) while satisfying the equality constraints $H\theta = h$ and $\Omega_a\theta_r = \omega_a$, without worrying about the inactive inequality constraints. If this new solution is feasible, that is if $\Omega(\theta_r + \Delta\theta) \leq \omega$, then it is accepted as the next iteration: $\theta_{r+1} = \theta_r + \Delta\theta$. If it is not feasible, then a line-search is made in the direction $\Delta\theta$ to locate the point at which feasibility is lost — namely the point at which one of the inactive inequality constraints becomes active. The solution at this point is accepted as the next iteration: $\theta_{r+1} = \theta_r + \alpha_r\Delta\theta$, where $0 < \alpha_r < 1$, and the newly active constraint is added to the active set.

It remains to decide whether this new solution is already the global optimum of the QP problem, or whether further improvement can be obtained. This decision can be made by checking whether the *KKT* conditions (3.59)–(3.62) are satisfied at θ_{r+1}. Note that the *complementarity condition* (3.62) implies that those elements of λ corresponding to inactive constraints are zero, since the corresponding elements of t are positive. So only those elements of λ corresponding to active constraints need to be evaluated; if they are all nonnegative then the global solution has been found, because of the condition $\lambda \geq 0$; otherwise further iteration is needed. So suppose that $\lambda_q < 0$. Since λ_q is a Lagrange multiplier corresponding to the qth inequality constraint, which is active (by assumption), its negative value indicates that the cost function could be reduced by making that constraint inactive, namely moving to a solution θ for which $\Omega_q\theta < \omega_q$. Thus the qth constraint is removed from the active set. (If more than one element of

λ is negative, then the constraint corresponding to the most negative one is removed from the active set.) So now a new active set has been selected, and the whole process is repeated, replacing θ_r by θ_{r+1}.

Note that $V(\theta_{r+1}) < V(\theta_r)$, where $V(\theta) = \frac{1}{2}\theta^T\Phi\theta + \phi^T\theta$ is the cost function, so that this iterative process is guaranteed to terminate at the global optimum, because of the convexity of the QP problem. A potential advantage of the Active Set method over other methods, for predictive control, is that the iterations remain feasible once an initial feasible solution has been found. In many predictive control problems it is the feasibility of the solution, rather than exact optimality, that is most important. So if the QP solver has failed to terminate by the time the next control move is required, using the latest available iteration may be an adequate solution, and in many cases it is likely to be nearly as effective as the true optimal solution. This may arise, for example, when an unusually large disturbance occurs, so that the most important thing is to keep the plant within a safe operating region — that is, find a feasible solution if the constraints delineate the boundary of this region; in these circumstances, a good initial guess of the solution is not likely to be available from earlier steps.

Now we consider in more detail the minimization of the cost function at the rth iteration. The new cost is

$$V(\theta_r + \Delta\theta) = \frac{1}{2}(\theta_r + \Delta\theta)^T\Phi(\theta_r + \Delta\theta) + \phi^T(\theta_r + \Delta\theta) \tag{3.63}$$

$$= V(\theta_r) + \frac{1}{2}\Delta\theta^T\Phi\Delta\theta + (\phi^T + \theta_r^T\Phi)\Delta\theta \tag{3.64}$$

so the minimization problem to be solved can also be stated as

$$\min_{\Delta\theta} \frac{1}{2}\Delta\theta^T\Phi\Delta\theta + \phi_r^T\Delta\theta \tag{3.65}$$

subject to

$$H\Delta\theta = 0 \qquad \text{and} \qquad \Omega_a\Delta\theta = 0 \tag{3.66}$$

where $\phi_r = \phi + \Phi\theta_r$. This is a convex QP problem, but with equality constraints only.

One way of solving such an equality-constrained problem is by the method of Lagrange multipliers — that is, by using the *KKT* conditions for this sub-problem — but since there are only equality constraints, only (3.59) and (3.60) are needed from the *KKT* conditions. However, it is convenient to separate out the vector ζ into those components corresponding to $H\Delta\theta = 0$ and those corresponding to $\Omega_a\Delta\theta = 0$. We shall call these $\Delta\zeta$ and $\Delta\lambda$, respectively. Thus the *KKT* conditions for this sub-problem are: there must exist vectors $\Delta\zeta$ and $\Delta\lambda$ (unrestricted as to sign), such that

$$\Phi\Delta\theta + H^T\Delta\zeta + \Omega_a^T\Delta\lambda = -\phi_r \tag{3.67}$$

$$-H\Delta\theta = 0 \tag{3.68}$$

$$-\Omega_a\Delta\theta = 0 \tag{3.69}$$

These equations can be assembled into the matrix equation:

$$\begin{bmatrix} \Phi & H^T & \Omega_a^T \\ H & 0 & 0 \\ \Omega_a & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\zeta \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -\phi_r \\ 0 \\ 0 \end{bmatrix} \tag{3.70}$$

This equation is solved by some version of Gaussian elimination, which involves factorizing the matrix on the left-hand side into the product of a lower-triangular and an upper-triangular matrix:

$$\begin{bmatrix} \Phi & H^T & \Omega_a^T \\ H & 0 & 0 \\ \Omega_a & 0 & 0 \end{bmatrix} = LU \tag{3.71}$$

Once this factorization has been obtained, the solution is obtained very quickly, since it requires the solution (for η) of

$$L\eta = \begin{bmatrix} -\phi_r \\ 0 \\ 0 \end{bmatrix}$$

which is done by forward substitution, followed by the solution of

$$U \begin{bmatrix} \Delta\theta \\ \Delta\zeta \\ \Delta\lambda \end{bmatrix} = \eta$$

which is done by back-substitution. So the speed of the Active Set method is dominated by the speed with which this factorization can be performed. At first sight it seems obvious that the symmetric structure of the matrix should be exploited, and there are algorithms for efficient LU factorization of symmetric matrices [GL89]. But as pointed out in [Wri96], more advantage may be obtained by rearranging the variables in $[\Delta\theta^T, \Delta\zeta^T, \Delta\lambda^T]^T$, so that the matrix becomes *banded*, if that is possible. The structure of the predictive control problem allows this to be done, though at the cost of introducing many more variables into the problem.

Taking the predictive control QP problem to be defined by (3.41) and (3.42), we see that the corresponding values for Φ and Ω are

$$\Phi = \mathcal{H} = \Theta^T \mathcal{Q} \Theta \tag{3.72}$$

$$\Omega = \begin{bmatrix} F \\ \Gamma\Theta \\ W \end{bmatrix} \tag{3.73}$$

and H does not exist. Although $\mathcal{Q}$ is block-diagonal, Θ is nearly full, so that Φ does not have any particular structure which can be exploited, apart from symmetry.

But an alternative formulation is obtained by not eliminating the predicted states $\hat{x}(k+j|k)$ from the problem, and leaving them as variables to be found by the QP solver. That is, we still minimize the cost function (3.2), but instead of replacing $\mathcal{Z}(k)$ by using (3.5), we use (2.70) and impose the *equality* constraints

$$\hat{x}(k+j+1|k) = A\hat{x}(k+j|k) + B\hat{u}(k+j|k) \qquad \text{for} \qquad j = 0, \dots, H_p - 1 \tag{3.74}$$

and

$$\hat{u}(k+j+1|k) = \hat{u}(k+j|k) + \Delta\hat{u}(k+j+1|k) \qquad \text{for} \qquad j = 0, \dots, H_u - 1 \tag{3.75}$$

(with $\hat{u}(k-1|k) = u(k-1)$). The minimization is then performed over the variables $\Delta\hat{u}(k+j|k)$ and $\hat{u}(k+j|k)$ for $(j = 0, \dots, H_u - 1)$, and $\hat{x}(k+j|k)$ for $j = 1, \dots, H_p$. This introduces $\ell H_u + nH_p$ additional variables into the problem (where ℓ is the number of inputs and n is the state dimension), which looks like a bad idea. But the potential benefit is that now the matrix which has to be factorized in the Active Set method can be banded. If the variables are ordered in the following way:

$$\theta = \begin{bmatrix} \hat{u}(k|k) \\ \Delta\hat{u}(k|k) \\ \hat{x}(k+1|k) \\ \hat{u}(k+1|k) \\ \Delta\hat{u}(k+1|k) \\ \hat{x}(k+2|k) \\ \vdots \\ \hat{u}(k+H_u-1|k) \\ \Delta\hat{u}(k+H_u-1|k) \\ \hat{x}(k+H_u|k) \\ \hat{x}(k+H_u+1|k) \\ \vdots \\ \hat{x}(k+H_p|k) \end{bmatrix} \tag{3.76}$$

then we have (assuming $H_w = 1$ and $H_u < H_p$ for simplicity):

$$\Phi = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & R(0) & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \bar{Q}(1) & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & R(1) & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \bar{Q}(2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bar{Q}(H_p-1) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bar{Q}(H_p) \end{bmatrix} \tag{3.77}$$

$$H = \begin{bmatrix} B & 0 & -I & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ I & -I & 0 & -I & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & A & B & 0 & -I & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & I & -I & 0 & -I & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & A & -I \end{bmatrix} \tag{3.78}$$

where $\bar{Q}(i) = C_z^T Q(i) C_z$. The point here is that both Φ and H have a banded structure (i.e. all their non-zero entries are relatively close to the principal diagonal), as does the matrix Ω associated with the inequality constraints. Now this does not itself make the matrix to be factorized in (3.71) banded, but this can also be achieved by reordering the order of the variables. The key is to keep all the variables associated with predicted time

$k+j$ grouped together, including the relevant elements of $\Delta\zeta$ and $\Delta\lambda$. The bandedness can be exploited to speed up the factorization. Furthermore, the matrix to be factorized at the $(r+1)$th iteration of the active set method is only a little different from the one factorized at the rth iteration. Wright [Wri96] proposes a method which takes advantage of both of these features.

Is it worthwhile obtaining bandedness in this way, since it involves introducing additional variables into the QP problem? According to [Wri96], the time required to factor a banded matrix with half-bandwidth b and total dimension $N(b+1)$ is $O(N(b+1)^3)$, compared with $O(N^3(b+1)^3)$ for a dense matrix of the same size. If we assume that $H_u = H_p$ (so that the bandwidth is the same along the whole diagonal of the matrix) and that half of the inequality constraints are active, then the half-bandwidth in the scheme just outlined is approximately $2n + 3\ell + \nu/2$, where ν is the number of inequality constraints (on inputs, input changes, and outputs), and the total size of the matrix to be factorized is $H_p(2n+3\ell+\nu/2)$. On the other hand, for the original scheme, using (3.41) and (3.42), the matrix is dense, and its size is approximately $(\ell+\nu/2)H_p$. So, roughly speaking, it is worth considering the banded scheme if

$$H_p^2\left(\ell+\frac{\nu}{2}\right)^3 > \left(2n+3\ell+\frac{\nu}{2}\right)^3 \tag{3.79}$$

For some typical numbers this works out as follows:

Typical process application $n = 200$, $\ell = 20$, $\nu = 40$
$\ell + \nu/2 = 40$, $2n + 3\ell + \nu/2 = 480$
'Banded' scheme looks worthwhile if $H_p > 41$.

Typical aerospace application $n = 12$, $\ell = 3$, $\nu = 4$
$\ell + \nu/2 = 5$, $2n + 3\ell + \nu/2 = 35$
'Banded' scheme looks worthwhile if $H_p > 18$.

These examples suggest that the banded scheme is likely to be worthwhile in many cases. As we will see in Chapters 6 and 8, long prediction horizons help to ensure closed-loop stability, and to minimize the risk of driving the plant into 'dead-ends' from which no feasible solution is possible; thus there are good reasons for making H_p as large as computation speed will allow. Note that if $H_u \ll H_p$ then the original 'dense' scheme will be more favoured over the 'banded' scheme than inequality (3.79) suggests. The comparison should be done more carefully for any particular application, since it will be affected by factors such as how many constraints are likely to be active typically (which it may be possible to predict for particular applications). Also, the comparison as made above disregards the difference in the difficulty of finding an initial feasible solution. In practice, actual performance comparisons of the two schemes may be required in order to make the correct decision.

One more point remains to be dealt with: how to find an initial feasible solution. This is needed in order to get the iterations of the active set method started. The basic idea is the following. Suppose that we have an infeasible solution θ_0, which satisfies the equality constraints $H\theta_0 = h$, and that a subset of the inequality constraints is satisfied, so that $\Omega_s\theta_0 \le \omega_s$, and the remaining inequality constraints are not satisfied:

$\Omega_u\theta_0 > \omega_u$. Then a solution θ is sought for the *linear programming* (LP) problem:

$$\min_{\theta} \sum_j (\omega_u^j - \Omega_u^j\theta) \tag{3.80}$$

subject to

$$H\theta = h \tag{3.81}$$

and

$$\Omega_s\theta \leq \omega_s \tag{3.82}$$

where Ω_u^j denotes the jth row of Ω_u and ω_u^j denotes the jth element of ω_u. The usual simplex algorithm for solving LP problems can be used, which searches among the vertices of the feasible region. However, the problem formulation is changed at every new vertex visited, because the selection of rows in Ω_s and Ω_u (and of corresponding elements in ω_s and ω_u) changes at every vertex. Eventually either all the inequality constraints are satisfied, in which case a feasible solution has been found, or the cost (3.80) is still positive and the Lagrange multipliers indicate that it cannot be improved, in which case the problem is infeasible. Problems with this basic idea are:

1. The feasible solution which is found is arbitrary, and can be very far from the optimal solution of the QP problem.

2. The simplex algorithm always requires as many active constraints as the dimension of θ. So it may not be possible to use the solution of the QP problem solved at the previous step as an initial guess *(hot starting)*, because that may have a different number of active constraints.

These problems can be overcome by adding a small multiple of the QP cost to the cost function (3.80). This 'biases' the initial feasible solution towards the optimal solution of the QP problem, hopefully leading to fewer iterations of the Active Set method, and it also changes the initial feasibility problem from an LP problem into a QP problem, which can be started with any number of active constraints.

3.3.2 Interior point methods

In the last twenty years or so, a rival family of algorithms for solving convex optimization problems has emerged. These are the so-called *interior point* methods [NN94, Wri97, RTV97]. They first came to prominence with Kamarkar's algorithm for solving LP problems, the first serious rival to the simplex algorithm, and one which was potentially dramatically faster for large problems. Much development since then has led to versions of interior point methods which are particularly effective for solving QP problems. The attraction of these new methods is that their computational complexity (number of iterations, time to complete, ...) is no worse than some polynomial function of parameters such as the number of constraints or the number of variables, whereas the complexity of other known approaches, including Active Set methods, can be exponential in these parameters in the worst case. We remark that the availability of interior point methods for more general convex optimization problems has led to the intense current interest in *linear matrix inequalities* (*LMI*s) for solving control and other 'systems' problems [BGFB94] — we shall make use of them in Section 8.4.

In this subsection we shall give a brief indication of how interior point methods work. As with active set methods, one can apply interior point methods to the predictive control problem 'naively', that is regarding it as just a standard QP problem, or one can exploit the particular structure inherent in predictive control problems. The second alternative has been investigated in detail by Wright *et al* [Wri96, RWR98].

Whereas the main iterations of the active set methods search among points on the boundary of the feasible region, early interior point methods searched among points in the 'interior' of this region; in other words, the iterates were always feasible solutions of the optimization problem being solved. It turned out that this was not a particularly efficient strategy, among other things requiring an initial feasible point to be found. The iterates in more recent versions are not feasible — until the end of the search, typically — but they are still away from the boundary of the feasible region, and in that sense 'interior' (and they are in the interior of a certain positive orthant).

Suppose that a function $V(x)$ is to be minimized over x, subject to the constraint $Ax \leq b$. One way of looking at interior point methods is to consider the minimization of the function

$$f(x,\gamma) = \gamma V(x) \underbrace{- \sum_i log(b_i - a_i^T x)}_{barrierfunction} \tag{3.83}$$

where a_i^T is the ith row of A, b_i is the ith element of the vector b, and γ is some positive scalar. If the minimum of $f(x,\gamma)$ is being sought by some search strategy, starting in the feasible region, then the logarithmic barrier function prevents the search leaving the feasible region, since it becomes infinite on the boundary of the region.[1] Let x_γ be the minimizer of $f(x,\gamma)$, and let x^* be the solution of the original problem, namely the minimizer of $V(x)$ which satisfies the constraints. x_0 is known as the *analytic centre* of the constraints; if the feasible region is not empty then x_0 lies in its interior, and does not depend on the objective function $V(x)$ at all. On the other hand, $x_\gamma \to x^*$ as $\gamma \to \infty$. The underlying idea is that if an initial (feasible) solution is found in the vicinity of x_0, then it can be continuously improved by increasing γ and minimizing $f(x,\gamma)$, until $\|x^* - x_\gamma\|$ is sufficiently small. The path traced out by x_γ is known as the *central path*.

Example 3.1

Consider the following QP problem, taken from Fletcher [Fle87, Problem 10.4]:

$$\min_\theta V(\theta) = \theta_1^2 - \theta_1\theta_2 + \theta_2^2 - 3\theta_1$$

[1] Other barrier functions are possible.

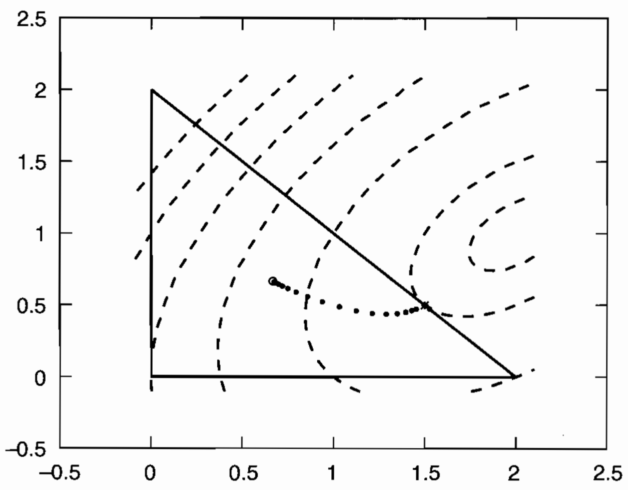

Figure 3.6 Basic interior point method for Example 3.1.

subject to

$$\theta_1 \geq 0$$
$$\theta_2 \geq 0$$
$$\theta_1 + \theta_2 \leq 2$$

Figure 3.6 shows the feasible region — the interior of the triangle — and some contours of $V(\theta)$. It is clear that the optimal solution is at $(3/2, 1/2)$, where the $V(\theta) = -11/4$ contour is tangential to the constraint $\theta_1 + \theta_2 \leq 2$. The small circle in the middle of the triangle shows the analytic centre, which is at $(2/3, 2/3)$, and the dots show the central path traced out as γ increases from 0 to 100. To get the sequence of solutions shown, the γ values were spaced logarithmically between 0.01 and 100, using 20 values (*MATLAB* command: `logspace(-2,2,20)`).

Examining the neighbourhood of the optimal solution in Figure 3.6 reveals a drawback of naive implementations of the interior point approach: one of the solutions seems to be at the constraint, but a little way away from the optimum — in fact, it is at $(1.5266, 0.4742)$, which is just infeasible. This is the solution obtained with the last few values of γ, and it is clearly incorrect. The reason is that with these values of γ the solution is very close to the constraint boundary, where the logarithmic barrier function is increasing extremely quickly, and as a result the optimization problem becomes very ill-conditioned. The solution which is obtained then depends strongly on the particular algorithm used; the solution shown in the figure was obtained using the very unsophisticated 'direct-search' approach implemented by *MATLAB*'s *Optimization Toolbox* function `fmins`. The major reason why good interior point algorithms are relatively complicated is that they take elaborate precautions to deal with this ill-conditioning.

At present the most effective interior-point algorithms are the so-called *primal-dual* methods [Wri97]. These find solutions to convex optimization problems and their dual problems simultaneously. Recall the KKT conditions (3.59)–(3.62) for the QP problem. If the complementarity condition (3.62) is relaxed to

$$t^T \lambda = \frac{1}{\gamma} > 0 \tag{3.84}$$

then the equations (3.59), (3.60), (3.61), (3.84) define a central path which converges to the optimal solution of the QP problem, and simultaneously of its dual, as $\gamma \to \infty$. The basic idea of primal-dual methods is to alternately take steps towards the central path (γ fixed), and 'parallel' to the central path (γ increased). For both kinds of step the search direction is found by solving an equation of the form

$$\begin{bmatrix} \Phi & H^T & \Omega^T \\ H & 0 & 0 \\ \Omega & 0 & \Lambda^{-1}T \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\zeta \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r_\phi \\ r_H \\ r_\Omega \end{bmatrix} \tag{3.85}$$

where $\Lambda = \text{diag}\{\lambda_i\}$ and $T = \text{diag}\{t_i\}$. The vector on the right-hand side of (3.85) varies, depending on which kind of step is being performed, and on which particular variant of primal-dual algorithms is being implemented. It will be seen that the matrix on the left-hand side of this equation has the same form as the matrix which appears in (3.70). As in the Active Set method, the speed of solution is dominated by the speed with which equation (3.85) can be solved. Therefore much the same considerations apply as were discussed in the previous section. Once again, it is advantageous to order the variables so that the matrix in (3.85) is banded. For further details see [RWR98, Wri97].

3.4 Softening the constraints

A serious problem which can occur with the predictive control problem as we have formulated it so far is that the optimizer may be faced with an infeasible problem. This can happen because an unexpectedly large disturbance has occurred, so there is really no way in which the plant can be kept within the specified constraints. Or it can happen because the real plant behaves differently from the internal model; the predictive controller may attribute differences between the plant and the model behaviours to large disturbances, and if these keep growing then it can eventually decide — erroneously — that it does not have enough control authority to keep the plant within the constraints. There are many ways in which the predictive control problem can become infeasible, and most of them are difficult to anticipate.

Because of this, it is essential to have a strategy for dealing with the possibility of infeasibility. Standard algorithms for solving QP problems just give up, and output a message such as '`Infeasible problem`'. Clearly this is unacceptable behaviour for an on-line controller. Various possibilities exist, ranging from *ad hoc* measures such as outputting the same control signal as in the previous step, or (better) the control signal computed as $\hat{u}(k+2|k)$ in the previous step, to sophisticated strategies of 'constraint management', in which one tries to relax the least-important constraints in an attempt to regain feasibility — see Section 10.2.

One systematic strategy for dealing with infeasibility is to 'soften' the constraints. That is, rather than regard the constraints as 'hard' boundaries which can never be crossed, to allow them to be crossed occasionally, but only if really necessary.

There is an important distinction between input and output constraints. Usually input constraints really are 'hard' constraints, and there is no way in which they can be softened; valves, control surfaces, and other actuators have limited ranges of action, and limited slew rates. Once these have been reached there is no way of exceeding them, except for fitting more powerful actuators. Therefore input constraints are usually not softened.

A straightforward way of softening output constraints is to add new variables, so-called 'slack variables', which are defined in such a way that they are non-zero only if the constraints are violated. Then their non-zero values are very heavily penalized in the cost function, so that the optimizer has a strong incentive to keep them at zero if possible.

A first way of doing this is to add a quadratic penalty for constraint violations [OB94]. The optimization problem (3.43)–(3.44) is modified to

$$\min_{\theta,\epsilon} \frac{1}{2}\theta^T\Phi\theta + \phi^T\theta + \rho\|\epsilon\|^2 \tag{3.86}$$

subject to

$$\begin{cases} \Omega\theta \le \omega + \epsilon \\ \epsilon \ge 0 \end{cases} \tag{3.87}$$

where ϵ is a nonnegative vector of the same dimension as ω, and ρ is a nonnegative scalar. This is still a QP problem, though with a larger number of variables. It is clear that a similar modification can be made if some of the constraints are to be retained as hard constraints.

If $\rho = 0$ then one has the completely unconstrained problem, and as $\rho \to \infty$ so one recovers the solution to the hard-constrained problem. But a drawback of a quadratic penalty on constraint violations is that, if the constraints are active, then for all finite values of ρ this formulation will result in them being violated to some extent (decreasing as ρ increases), even if such violation is not necessary.

Alternatives are to penalize the 1-norm (sum of violations) or the ∞-norm (maximum violation) of the constraint violations:

$$\min_{\theta,\epsilon}\left(\frac{1}{2}\theta^T\Phi\theta + \phi^T\theta + \rho\|\epsilon\|_1\right) = \min_{\theta,\epsilon}\left(\frac{1}{2}\theta^T\Phi\theta + \phi^T\theta + \rho\sum_j \epsilon_j\right) \tag{3.88}$$

subject to

$$\begin{cases} \Omega\theta \le \omega + \epsilon \\ \epsilon \ge 0 \end{cases}$$

or

$$\min_{\theta,\epsilon}\left(\frac{1}{2}\theta^T\Phi\theta + \phi^T\theta + \rho\|\epsilon\|_\infty\right) = \min_{\theta,\epsilon}\left(\frac{1}{2}\theta^T\Phi\theta + \phi^T\theta + \rho\max_j \epsilon_j\right) \tag{3.89}$$

subject to

$$\begin{cases} \Omega\theta \le \omega + \epsilon \\ \epsilon \ge 0 \end{cases}$$

Note that the second alternative can be posed more efficiently as

$$\min_{\theta,\epsilon} \left(\frac{1}{2}\theta^T \Phi \theta + \phi^T \theta + \rho\epsilon \right) \tag{3.90}$$

subject to

$$\begin{cases} \Omega\theta \le \omega + \epsilon\mathbf{1} \\ \epsilon \ge 0 \end{cases} \tag{3.91}$$

where ϵ is a scalar, and $\mathbf{1}$ is a vector, each element of which is 1. In this form only one slack variable is introduced, which gives, in general, a much faster algorithm. Penalizing the 1-norm of constraint violations requires a separate slack variable for every constraint, at every point of the prediction horizon for which the constraint is enforced. The number of slack variables can (greatly) exceed the number of decision variables in the original 'hard-constrained' problem. Both of these 'soft' optimization problems are again equivalent to QP problems.

It can be shown that, with ρ large enough, using either the 1-norm or the ∞-norm penalty on constraint violations gives an 'exact penalty' method, which means that constraint violations will not occur unless there is no feasible solution to the original 'hard' problem. That is, the same solution will be obtained as with a 'hard' formulation, if a feasible solution exists. The reason for this difference in behaviour between these formulations and the quadratic penalty formulation is that, starting from the true constrained solution θ^*, there exists a move to an infeasible solution $\theta^* + \varepsilon d$, for some vector d, which gives a reduction in the cost function of $o(\varepsilon)$. Since the penalty for the violation is $o(\varepsilon^2)$ then the violation gives a smaller value of the 'softened' cost function (3.86), for small enough ε. But in the cost functions (3.88) and (3.90) the penalty is also $o(\varepsilon)$, so if the coefficient ρ is large enough then such a move results in an increase in the cost function. 'Large enough' means 'larger than $\|d(\frac{1}{2}\theta^{*T}\Phi\theta^* + \phi^T\theta^*)/d\omega\|_\infty$ at the optimal solution'; this can be related to the Lagrange multipliers of the original 'hard' problem at the optimal solution — for details see [Fle87]. Unfortunately, in the context of predictive control, the Lagrange multipliers will depend on the current state, so it is not easy to decide just how large ρ should be [OB94, SR96b]. A contribution towards solving this problem is made in [KM00b]. In [SR96b] a mixture of linear and quadratic penalties on constraint violations is proposed.

The following example shows that it can be difficult to predict the circumstances in which infeasibility is encountered, and hence the virtual necessity of softening constraints. In this example the infeasibility arises as a result of modelling error.

Example 3.2

Consider again the linearized Citation aircraft, as introduced in Section 2.7. In that section it has already been shown that some error in the model can be tolerated without

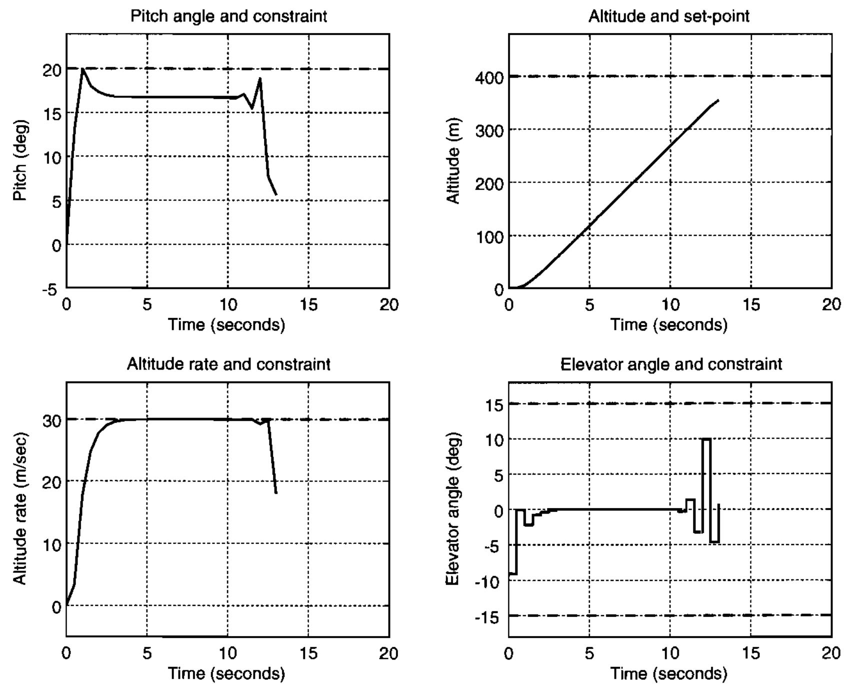

Figure 3.7 Plant–model error and hard constraints result in infeasibility.

serious difficulty. But this is not always true. Suppose that the gain from elevator to altitude rate is *over*-estimated by 20%, and that a constraint is imposed on the altitude, in an attempt to ensure a very small overshoot. If the altitude is to be increased by 400 m, as in Figure 2.8, the constraint

$$\hat{y}_2(k+i|k) < 405 \tag{3.92}$$

is imposed for all k and i, and all other parameters are the same as in Section 2.7, Figure 2.8, then the result is shown in Figure 3.7. Thirteen seconds into the manoeuvre, the predictive controller is faced with an infeasible problem, and stops working.

Figure 3.8 shows the results when the output constraints are softened, using a 1-norm penalty function with weight $\rho = 10^4$. Note that the pitch angle constraint is violated briefly in the early part of the manoeuvre, although this did not happen with hard output constraints — see Figure 3.7; this suggests that the penalty function is not exact, despite the large value of ρ. Another notable feature is that the altitude constraint of 405 m is not violated — in fact there is no overshoot, and the altitude never exceeds 400 m. This may seem inconsistent with the fact that the hard-constrained problem became infeasible after 13 seconds. The explanation is that the figure shows the actual altitude of the aircraft, whereas the infeasibility arose from the controller's predictions of the altitude, which were based on an inaccurate model. This phenomenon, of the controller predicting a constraint violation which does not actually occur, is quite common. The converse phenomenon can be seen in Figure 2.11; there the altitude rate constraint is

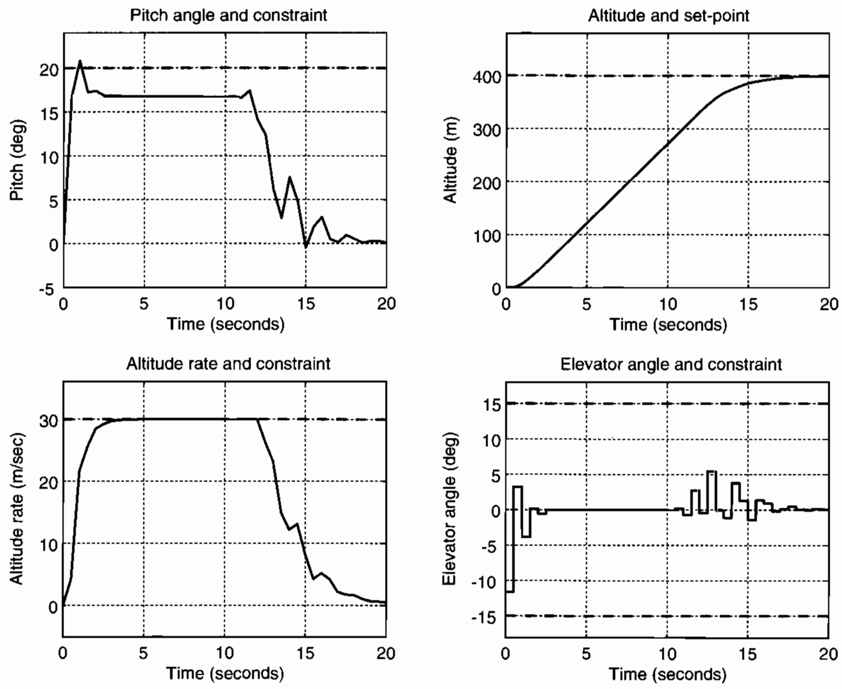

Figure 3.8 Soft output constraints restore feasibility. 1-norm penalty with $\rho = 10^4$.

violated briefly, yet no infeasibility is detected by the controller. Again, the explanation is that the controller's predictions were inaccurate and did not indicate an inevitable constraint violation.

In this example, the 'hard-constrained' QP problem to be solved at each step has three decision variables, namely $\hat{u}(k+i|k)$ for $i = 0, 1, 2$, since $H_u = 3$ and there is only one control input. When the output constraints are softened using a 1-norm penalty, four slack variables are introduced at each point in the prediction horizon — one for each constraint: maximum pitch angle, minimum pitch angle, maximum altitude rate, and maximum altitude. Since the constraints are enforced at each point of the prediction horizon, which has length $H_p = 10$, there are 40 slack variables altogether. So the total number of decision variables has increased from 3 to 43, as a result of softening the constraints in this way. The average time required to solve and simulate one step on a 333 MHz Pentium II processor was 0.81 sec. (Most of this time is taken by the optimization, since simulation of a linear model is very fast.) Solving the same problem using an ∞-norm penalty, again with $\rho = 10^4$, gave virtually identical results, but required only 0.15 sec per step. The average time required per step for the hard-constrained problem was 0.04 sec.[2] These solution times are summarized in Table 3.2.

[2] The simulation was performed using the *Model Predictive Control Toolbox* function `scmpc` for the hard-constrained case, and its modified version `scmpc2` for the soft-constrained cases — see Appendix C. The latter uses a different QP solver, so the solution speeds are not directly comparable. It should also be borne in mind that the quoted solution times are very far from the minimum achievable times.

Table 3.2 Relative solution times for different formulations of altitude change manoeuvre.

Problem formulation	Time
Soft-constrained, 1-norm penalty	0.81 s
Soft-constrained, ∞-norm penalty	0.15 s
Hard-constrained	0.04 s

A more typical reason for encountering infeasibility is the presence of disturbances. The next example illustrates this, and again shows the efficacy of softening constraints. It also shows that using inexact penalty functions can give a useful compromise between the harshness of constraints and the 'sponginess' of set-point specifications.

Example 3.3

In Section 2.7, Figure 2.9, the response was shown to a disturbance of 5 m/sec in the altitude rate, arising from turbulence acting on the Citation aircraft. If, however, the disturbance is slightly greater, 6 m/sec instead of 5 m/sec, then the controller finds that it cannot return the altitude rate to below 30 m/sec within one sample period — because of the elevator angle being limited to 15° — and infeasibility occurs. Note that this is true even if the model is perfect.

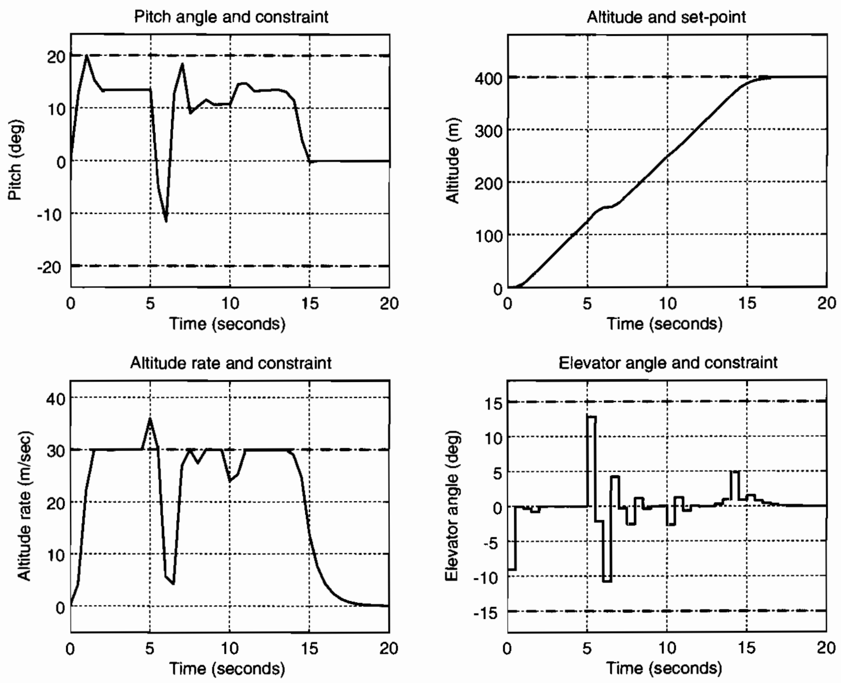

Figure 3.9 Response to disturbance with softened altitude rate constraint, $\rho = 5 \times 10^5$.

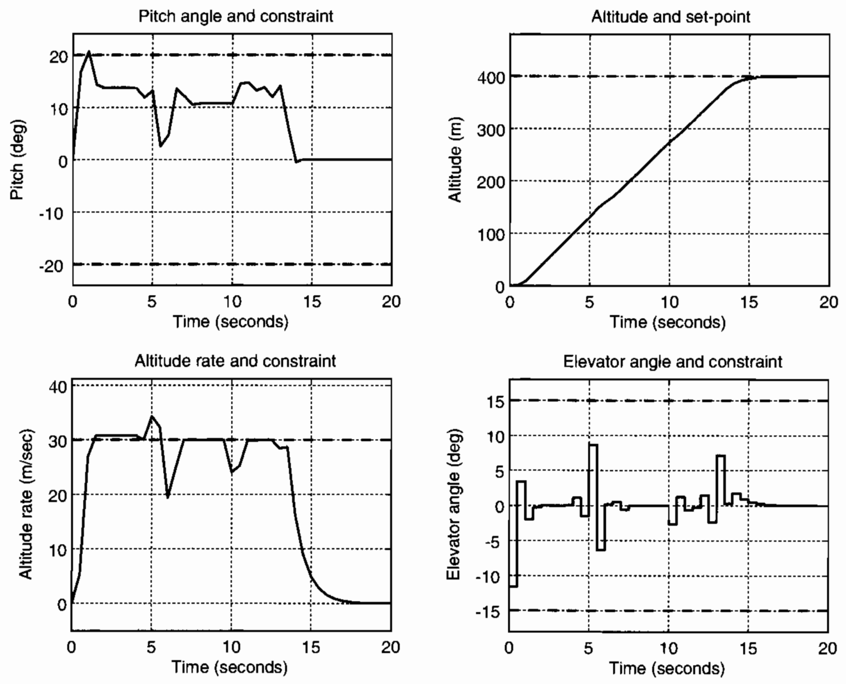

Figure 3.10 Response to disturbance with softened altitude rate constraint, $\rho = 10^4$.

If, however, the constraints are softened, then the disturbance can be dealt with. Figure 3.9 shows the response when an ∞-norm penalty function is used with $\rho = 5 \times 10^5$. Such a large value of ρ is needed to get an exact penalty function in this case; the peak value of all the Lagrange multipliers obtained during the simulation shown in Figure 2.9 was approximately 4×10^5. This response is very similar to that shown in Figure 2.9; thus, although the manoeuvre can be completed successfully despite the large disturbance, the disadvantage of a violent response to avoid violation of the constraints is still present.

If, however, the penalty coefficient is reduced to $\rho = 10^4$, so that the penalty function is no longer exact, then the response shown in Figure 3.10 is obtained. The response to the onset of the disturbance is now significantly less agressive than before, although still more agressive than the response to the disappearance of the disturbance 5 seconds later. The inexactness of the penalty function reveals itself by the slight violation of the altitude rate constraint before the disturbance occurs.

Using a 1-norm penalty function (instead of ∞-norm) gives very similar results for this example, if the same values of the penalty coefficient ρ are used.

In [RK93] an unconventional way of solving the constrained QP problem is introduced. This solves the problem as an iteration of weighted least-squares problems, the weights being changed at each iteration in such a way that the greatest penalty is attached to the most-violated constraint. This can be shown to find the correct solution to the QP problem if it exists, but has the advantage that it produces a solution even if the problem is infeasible, so that it enforces soft constraints, in effect. Furthermore, the solution that is produced is one which tries to minimize the worst-case constraint violation, in the sense of minimizing the peak value of the violation.

It may happen, however, that minimizing the magnitude of constraint violations leads to the violations persisting over an unnecessarily long time, and this may not be the preferred trade-off. In [SR99] several formulations of soft constraints are discussed, which allow various magnitude/duration trade-offs to be imposed.

Exercises

3.1 Work through the unconstrained predictive control example described in the section 'Unconstrained MPC using state-space models' of the *MPC Toolbox User's Guide*. Don't worry about how the *Toolbox* does the conversion from transfer function to state-space form. Also don't worry about how the observer (= 'estimator') gain is designed (p. 55). The emphasis for the moment is on the 'mechanical' skills of running the *Toolbox*.
(You should read the detailed description of the function `smpccon` before/while doing this.)

3.2 Work through the constrained predictive control example described in the section 'State-space MPC with constraints' of the *MPC Toolbox User's Guide*.
(You should read the detailed description of the function `scmpc` before or while doing this.)

3.3 The water temperature in a heated swimming pool, θ, is related to the heater input power, q, and the ambient air temperature, θ_a, according to the equation

$$T\frac{d\theta}{dt} = kq + \theta_a - \theta \tag{3.93}$$

where $T = 1$ hour and $k = 0.2$ °C/kW. (It is assumed that the water is perfectly mixed, so that it has a uniform temperature.) Predictive control is to be applied to keep the water at a desired temperature, and a sampling interval $T_s = 0.25$ hour is to be used. The control update interval is to be the same as T_s.

(a) Use *MATLAB*'s *Control System Toolbox* to show that the corresponding discrete-time model is

$$\theta(k+1) = 0.7788\theta(k) + 0.0442q(k) + 0.2212\theta_a(k) \tag{3.94}$$

Form the corresponding model in the *Model Predictive Control Toolbox*'s 'MOD' format, assuming that q is the control input and that θ_a is an unmeasured disturbance.

(b) Verify that if the weights $Q = 1$ and $R = 0$, and the horizons $H_p = 10$ and $H_u = 3$ (and $H_w = 1$) are used, then

$$K_s = K_{MPC}[1, -\Psi, -\Upsilon] = [22.604, -17.604, -22.604] \tag{3.95}$$

where K_s is the unconstrained controller gain matrix Ks produced by *Model Predictive Control Toolbox* function `smpccon`, and that stable closed-loop behaviour is obtained.

(c) If the air temperature θ_a is constant at 15 °C verify that a set-point for θ of 20 °C is attained without error. Verify that this remains the case even if the pool parameters change to $T = 1.25$ hour, $k = 0.3$ °C/kW but the predictive controller's internal model remains unchanged.

(d) Suppose that the air temperature follows a sinusoidal diurnal variation with amplitude 10 °C:

$$\theta_a(t) = 15 + 10 \sin\left(\frac{2\pi}{24}t\right) \tag{3.96}$$

(where t is measured in hours). Verify that in the steady state the mean water temperature reaches the set-point exactly, but that θ has a small residual oscillation of amplitude approximately 0.5 °C.

(e) Now suppose that the input power is constrained:

$$0 \leq q \leq 40 \quad \text{kW} \tag{3.97}$$

and that the air temperature varies sinusoidally, as above. Investigate the behaviour of the predictive control system — use *Model Predictive Control Toolbox* function `scmpc`.

(You are advised to save any models created, and to keep a record of *MATLAB* commands you use — using `diary` for example — as this swimming pool example will reappear in later examples and problems. See Exercises 5.1 and 5.6, Example 7.4 and Exercise 7.6.)

3.4 If *blocking* is used, as in the *Model Predictive Control Toolbox*, how should the details of equation (2.23) be changed? (Also see Exercise 1.14.)

3.5 Suppose that the cost function used in the predictive control formulation was changed to include the penalty term

$$\sum_{i=0}^{H_u-1} ||\hat{u}(k+i|k) - u_{ref}(k+i)||^2_{S(i)}$$

in addition to the terms penalizing tracking errors and control moves, where u_{ref} denotes some prescribed future trajectory for the plant inputs.

(a) Describe in detail the changes that would need to be made to the computations of predictions and the optimal solution.

(b) Briefly list some reasons for and against making such a modification.

3.6 (a) If inequality (3.35) arises from the simple range constraints (3.38), show that

$\mathcal{F}$ takes the simple form

$$\mathcal{F} = \begin{bmatrix} I & 0 & \cdots & 0 \\ -I & 0 & \cdots & 0 \\ I & I & \cdots & 0 \\ -I & -I & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ I & I & \cdots & 0 \\ -I & -I & \cdots & 0 \\ I & I & \cdots & I \\ -I & -I & \cdots & -I \end{bmatrix}$$

What form does the vector on the right-hand side of inequality (3.37) take in this case? (Note that it is possible to reorder the inequalities so that $\mathcal{F}$ takes the form of two lower-triangular matrices stacked on top of each other in this case, each one being made up of identity matrices. This is commonly done in the literature [PG88].)

(b) Put inequality (3.34) into the form $W\Delta\mathcal{U}(k) \leq w$, where W is a matrix and w is a vector. If (3.34) arises from simple rate constraints of the form $-B(k+i) \leq \Delta\hat{u}(k+i|k) \leq B(k+i)$, what form do W and w take?

3.7 If you look at the code for function `smpccon` in the *MPC Toolbox* (using `type smpccon`), you will see how the weights are forced to be diagonal before solving for the unconstrained gain matrix K_{MPC}. Copy the function to your own directory, change its name, and edit it so that you can specify arbitrary weights. Then solve Example 6.1 and Exercise 6.2 using your new function. Simulate the designs using `smpcsim`.

3.8 Put the QP problem stated in Example 3.1 into the standard form (3.56)–(3.58). Check that $\Phi \geq 0$, and hence that the problem is convex. Check that the point $(\theta_1 = 3/2, \theta_2 = 1/2)$ satisfies the *KKT* conditions (3.59)–(3.62), and hence that it is the optimal solution, as claimed in the example.

3.9 (a) Check that the optimization problem (3.86)–(3.87) is a standard QP problem, as claimed in the text.

(b) Modify the problem (3.86)–(3.87) to deal with the case that a subset of the constraints $\Omega_1\theta \leq \omega_1$ is to be 'hard', while the remainder, $\Omega_2\theta \leq \omega_2$, is to be 'soft', and again check that this is a QP problem.

3.10 Check that the optimization problems (3.88)–(3.4) and (3.90)–(3.91) are standard QP problems.

3.11 Suppose that the gain from the elevator angle to the altitude rate of the Citation aircraft model is underestimated by 20%, rather than overestimated, as in Example 3.2, but everything else is the same as in Example 3.2. Show that infeasibility is not encountered in this case, but the response is relatively oscillatory — as one would expect with conventional control if the plant gain were higher than expected.

(The function `mismatch2`, available from this book's web site, makes it easy to run the simulation. Use parameter values: `percent=-20` to simulate the 20% gain undermodelling, and `pencoeff=inf`, `normtype='inf-norm'` to enforce hard constraints.)

3.12 Verify that similar results to those shown in Example 3.3 are obtained if a 1-norm penalty function is used instead of an ∞-norm penalty function.

(The function `disturb2`, available from this book's web site, makes it easy to run the simulation. The parameter `pencoeff` corresponds to the penalty coefficient ρ, and can either be a scalar (the same coefficient for each constraint violation) or a vector, one element corresponding to each constraint. Set `normtype='1-norm'` to select the 1-norm penalty function.)

Chapter 6

Stability

If one is designing a predictive controller off-line, then nominal stability of the closed loop is hardly an issue. It is quite easy to obtain stability by tuning the parameters in the problem formulation, and very easy to check that the designed system is stable (assuming the correct plant model). This at least is the typical situation with current applications of predictive control. Current applications are invariably implemented on stable plants, and the dynamic performance demanded from predictive controllers is typically not very challenging. As predictive control is applied more widely, with tight dynamic specifications, and especially if it is applied to unstable plants, then this situation may change.

Closed-loop stability is much more of an issue if predictive control is being used in a way which requires any on-line redesign. This includes adaptive control in the traditional set-up, where re-estimation of the plant model and consequent redesign of the controller is assumed to occur continuously. It also includes situations such as occasional re-estimation followed by redesign, for example following a failure or a change in the plant configuration.

In this chapter several ways of guaranteeing nominal stability are investigated. It will be seen that there are now several ways of ensuring that the nominal closed loop, at least — that is, with the assumption that the model is perfect — is stable.

Predictive control, using the receding horizon idea, is a feedback control policy. There is therefore a risk that the resulting closed loop might be unstable. Even though the performance of the plant is being optimized over the prediction horizon, and even though the optimization keeps being repeated, each optimization 'does not care' about what happens beyond the prediction horizon, and so could be putting the plant into such a state that it will eventually be impossible to stabilize. This is particularly likely to occur when there are constraints on the possible control input signals. It is easy to construct an example where this happens, even without constraints:

Example 6.1

Consider a discrete-time system consisting of two delays in series:

$$x_1(k+1) = u(k)$$
$$x_2(k+1) = x_1(k)$$

or, using vector–matrix notation:

$$x(k+1) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(k)$$

Suppose the prediction horizon is $H_p = 1$, and the cost to be minimized is

$$\begin{aligned} V(k) &= \hat{x}_1(k+1|k)^2 + 6\hat{x}_2(k+1|k)^2 + 4\hat{x}_1(k+1|k)\hat{x}_2(k+1|k) \\ &= \hat{x}(k+1|k)^T \begin{bmatrix} 1 & 2 \\ 2 & 6 \end{bmatrix} \hat{x}(k+1|k) \end{aligned}$$

(which is positive-definite). The predictive control problem is to find $u(k)$ which will minimize $V(k)$.

Using the model, the predictions are $\hat{x}_1(k+1|k) = u(k)$ and $\hat{x}_2(k+1|k) = x_1(k)$. Substituting these into $V(k)$ gives

$$V(k) = u(k)^2 + 6x_1(k) + 4u(k)x_1(k).$$

To find the minimum of this, differentiate and set the derivative to zero:

$$\begin{aligned} \frac{\partial V}{\partial u(k)} &= 2u(k) + 4x_1(k) \\ &= 0 \text{ if } u(k) = -2x_1(k). \end{aligned}$$

Substituting this back into the model gives the closed-loop equations:

$$x_1(k+1) = -2x_1(k)$$
$$x_2(k+1) = x_1(k)$$

or

$$x(k+1) = \begin{bmatrix} -2 & 0 \\ 1 & 0 \end{bmatrix} x(k).$$

Clearly this is unstable, since the closed-loop transition matrix has an eigenvalue at -2, outside the unit circle.

One might guess that this problem arises in the example because the prediction horizon is too short, so that the control is too 'short-sighted'. This is correct, and it turns out that stability can usually be ensured by making the prediction horizon long enough, or even infinite. We will see that later in this chapter.

6.1 Terminal constraints ensure stability

Another way of ensuring stability is to have any length of horizon, but to add a 'terminal constraint' which forces the state to take a particular value at the end of the prediction horizon. And it is surprisingly easy to prove this, even in a very general case, by using a Lyapunov function, as was first shown by Keerthi and Gilbert [KG88]. For the purpose of proving stability it is enough to consider the case when the state is to be driven to the origin, from some initial condition.

Theorem 6.1

Suppose that predictive control is obtained for the plant

$$x(k+1) = f(x(k), u(k))$$

by minimizing the cost function

$$V(k) = \sum_{i=1}^{H_p} \ell(\hat{x}(k+i|k), \hat{u}(k+i-1|k))$$

where $\ell(x,u) \geq 0$ and $\ell(x,u) = 0$ only if $x = 0$ and $u = 0$, and ℓ is 'decrescent', subject to the terminal constraint

$$\hat{x}(k+H_p|k) = 0$$

The minimization is over the input signals $\{\hat{u}(k+i|k) : i = 0, 1, \dots, H_u - 1\}$, with $H_u = H_p$ (for simplicity), subject to the constraints $\hat{u}(k+i|k) \in U$ and $\hat{x}(k+i|k) \in X$, where U and X are some sets. We assume that $x = 0$, $u = 0$ is an equilibrium condition for the plant: $0 = f(0,0)$. The receding horizon method is applied, with only $u^0(k|k)$ being used from the optimal solution $\{u^0(k+i|k) : i = 0, \dots, H_u - 1\}$.

Then the equilibrium point $x = 0$, $u = 0$ is stable, providing that the optimization problem is feasible and is solved at each step.

Proof 6.1

The proof makes use of concepts from Mini-Tutorial 6. Let $V^0(t)$ be the optimal value of V which corresponds to the optimal input signal u^0, as evaluated at time t. Clearly $V^0(t) \geq 0$, and $V^0(t) = 0$ only if $x(t) = 0$ — because if $x(t) = 0$ then the optimal thing to do is to set $u(t+i) = 0$ for each i. We are going to show that $V^0(t+1) \leq V^0(t)$, and hence that $V^0(t)$ is a Lyapunov function for the closed-loop system.

As usual in stability proofs, we will assume that the plant model is perfect, so that the predicted and real state trajectories coincide: $x(t+i) = \hat{x}(t+i|t)$ if $u(t+i) = \hat{u}(t+i|t)$.

Equilibrium: The equation $x(k+1) = f(x(k), u(k))$ has an equilibrium (or 'fixed point') at state x_0 and input u_0 if $x_0 = f(x_0, u_0)$.

Note that we can always introduce a change of coordinates $z(k) = x(k) - x_0$, $v(k) = u(k) - u_0$, so that the equilibrium is at $(0, 0)$ in the new coordinates:

$$\begin{aligned} z(k+1) = x(k+1) - x_0 &= f(z(k) + x_0, v(k) + u_0) - f(x_0, u_0) \\ &= g(z(k), v(k)) \\ 0 &= g(0, 0) \end{aligned}$$

So we can always assume that an equilibrium is at $(0, 0)$.

Stability: For nonlinear systems one has to consider the stability of a particular equilibrium point, rather than the system. (Some equilibria may be stable, others unstable.)

An equilibrium point $(0, 0)$ is *stable* (in the sense of Lyapunov) if a small perturbation of the state or input results in a 'continuous' perturbation of the subsequent state and input trajectories. More precisely, given any $\epsilon > 0$, there is a $\delta > 0$ (which depends on $\epsilon > 0$ and the system) such that if $||[x(0)^T, u(0)^T]|| < \epsilon$ then $||[x(k)^T, u(k)^T]|| < \delta$ for all $k > 0$.

It is *asymptotically stable* if in addition $||[x(k)^T, u(k)^T]|| \to 0$ as $k \to \infty$.

For closed-loop systems $u(k)$ itself depends on $x(k)$, so we could have written $x(k+1) = f(x(k))$, where f now represents the closed-loop state equation.

Lyapunov's Theorem: If there is a function $V(x, u)$ which is positive-definite, namely such that $V(x, u) \geq 0$ and $V(x, u) = 0$ only if $(x, u) = (0, 0)$, and has the ('decrescent') property that

$$||[x_1^T, u_1^T]|| > ||[x_2^T, u_2^T]|| \Rightarrow V(x_1, u_1) \geq V(x_2, u_2)$$

and if, along any trajectory of the system $x(k+1) = f(x(k), u(k))$ in some neighbourhood of $(0, 0)$ the property

$$V(x(t+1), u(t+1)) \leq V(x(t), u(t))$$

holds, then $(0, 0)$ is a stable equilibrium point.

If, in addition, $V(x(t), u(t)) \to 0$ as $t \to \infty$ then it is asymptotically stable.

Such a function V is called a *Lyapunov function.*

Mini-Tutorial 6 Stability and Lyapunov functions.

With this assumption we have

$$\begin{aligned} V^0(t+1) &= \min_u \sum_{i=1}^{H_p} \ell(x(t+1+i), u(t+i)) \\ &= \min_u \left\{ \sum_{i=1}^{H_p} \ell(x(t+i), u(t-1+i)) - \ell(x(t+1), u(t)) \right. \\ &\quad \left. + \ell(x(t+1+H_p), u(t+H_p)) \right\} \\ &\leq -\ell(x(t+1), u^0(t)) + V^0(t) \\ &\quad + \min_u \left\{ \ell(x(t+1+H_p), u(t+H_p)) \right\} \end{aligned}$$

since the optimum is certainly no worse than keeping the optimal solution found at time t, which will take us up to time $t + H_p$, then doing the best we can for the final step. But we have assumed that the constraint $x(t + H_p) = 0$ is satisfied, since the optimization problem was assumed to be feasible, so we can make $u(t + H_p) = 0$ and stay at $x = 0$, which gives

$$\min_u \left\{\ell(x(t + 1 + H_p), u(t + H_p))\right\} = 0.$$

Since $\ell(x(t), u^0(t)) \geq 0$, we can conclude that $V^0(t + 1) \leq V^0(t)$. So $V^0(t)$ is a Lyapunov function, and we conclude by Lyapunov's stability theorem that the equilibrium $x = 0, u = 0$ is stable.

This sounds too good to be true. It seems too easy to guarantee stability. The problem is with the assumption that the optimization problem has a solution at each step, and that the global optimum can be found at each step. General constrained optimization problems can be extremely difficult to solve, and just adding a terminal constraint may not be feasible. But we will use the idea of this proof several times later, in more realistic situations.

Example 6.2

If we look again at Example 6.1 and try to get stability by adding the terminal constraint $x(k + 1) = 0$, we get an infeasible problem, because that could only be achieved if $x_1(k) = 0$, which will not be true in general. (In general you need at least two steps to drive a 2-state linear system to the origin.) Exercise 6.2 shows that for this example stability is obtained by increasing the prediction horizon to $H_p = 2$, even without a terminal constraint.

The same basic idea can be applied in various ways. Two particular ways, associated with GPC, have become quite well known and have been incorporated into particular versions of GPC: 'Stable input–output receding horizon control', or *SIORHC* (see [Mos95, Chapter 5.8]), and 'Constrained receding horizon predictive control', or *CRHPC* (see [CC95, Chapter 6.5]).

However, since we shall see in the following sections that stability can be achieved without imposing conditions as severe as single-point terminal constraints, we shall not pursue this approach further.

A very important generalization, and relaxation, of the idea of terminal constraints is to specify a terminal constraint *set* X_0, which contains the origin, rather than a single point. Usually, the idea is to use predictive control to drive the state into such a set, and then to switch to some other control law, which can be guaranteed to stabilize the system for initial conditions within X_0, assuring that the state will be driven to the origin, providing that it has first been driven into X_0. It is assumed that all constraints are inactive within X_0, so that conventional control (usually but not necessarily linear) is adequate when the state is within this terminal set. The idea of using a terminal constraint set was introduced in [MM93], as part of a scheme for obtaining robustly stable predictive control of nonlinear systems. Some more remarks about it can be found in Sections 8.5.2 and 10.3. Schemes which involve first using predictive control to drive the state into a terminal constraint set, then switching to another control law, are often called *dual-mode* predictive control schemes.

It has recently been argued [MRRS00] that all MPC schemes which have been proposed to date and which guarantee closed-loop stability, have a terminal constraint set built into them (possibly implicitly, and possibly consisting of a single point). Furthermore, they contain a control law defined on this terminal set which respects all the constraints, and which keeps the state in the set, once the state trajectory has entered it. Finally, they all contain (possibly implicitly) a terminal *cost*, that is, a cost penalizing non-zero states at the end of the prediction horizon; furthermore, the terminal cost function is such that it is a Lyapunov function when confined to the terminal constraint set. In many of the published proposals one or more of these elements are trivial — for instance, the terminal constraint set may consist of only the origin, the control law on this set (point) may be the prescription $u \equiv 0$, and the terminal cost function may be $F(x) = \infty$ if $x \neq 0$, $F(x) = 0$ if $x = 0$. Nevertheless, the explicit identification of these elements may be valuable for future proposals; in particular the recognition that both a terminal constraint set and a terminal cost can usefully be part of a successful 'prescription' is a surprising revelation.

6.2 Infinite horizons

The specific approach described in this section was originated in [RM93], and elaborated upon in [MR93b, MR93a], although the underlying ideas are similar to those in [KG88], where it was shown that infinite-horizon constrained control problems could be approximated by finite-receding-horizon problems with a terminal constraint. It had been known for some time (e.g. see [BGW90]) that making the horizons infinite in predictive control leads to guaranteed stability, but it was not known how to handle constraints with infinite horizons. The work of Rawlings and Muske made a breakthrough in this respect. The same ideas have also been applied to the transfer-function formulation [Sco97]— not surprisingly, the whole development can be repeated in that setting.

The key idea is to reparametrize the predictive control problem with infinite horizons in terms of a finite number of parameters (instead of the infinite number of control decisions $\Delta u(k|k)$, $\Delta u(k+1|k), \ldots$), so that the optimization can still be performed over a finite-dimensional space — in fact, it remains a QP problem. Since the original work of Rawlings and Muske, several other proposals have been made, which can be interpreted as different reparametrizations of the infinite horizon problem [RK93, SR96b].

In [Mor94] the opinion has been expressed that there is now no longer any reason to use finite horizons — at least with linear models.

Let us begin by examining why the use of an infinite costing horizon guarantees stability.

6.2.1 Infinite horizons give stability

Figure 6.1 shows an essential difference between a finite, receding horizon formulation of predictive control, and an infinite horizon formulation. The top half of the figure

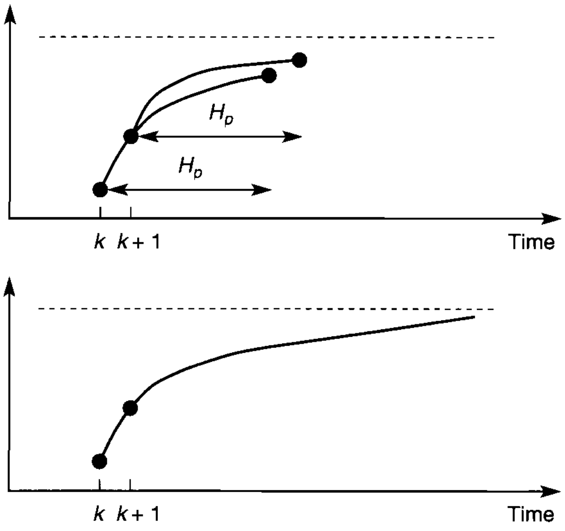

Figure 6.1 Finite and infinite horizons (no disturbances, perfect model).

depicts the finite-horizon formulation, with the plant output heading towards a constant set-point. At time k a particular trajectory is optimal over the prediction horizon of length H_p. In the absence of disturbances, and with a perfect model, the plant at time $k+1$ is precisely in the state which was predicted at the previous time step. One might expect, therefore, that the initial portion of the optimal trajectory over the prediction horizon from time $k+1$ to time $k+1+H_p$ would coincide with the previously computed optimal trajectory. But a new time interval now enters the picture which had not been considered when the earlier optimization had been performed — the interval between time $k+H_p$ and time $k+1+H_p$. The presence of this new time interval may lead to an optimal trajectory completely different from the one computed at the earlier step.

The lower half of Figure 6.1 shows the situation with an infinite horizon. At time k an optimal trajectory over the whole infinite horizon is determined (implicitly). At time $k+1$ no new information enters the optimization problem, so the optimal trajectory from this time on is the same as the 'tail' of the previously computed trajectory. (This is Bellman's celebrated *principle of optimality*, which states that the tail of any optimal trajectory is itself the optimal trajectory from its starting point. It does not apply in the finite-horizon case, because there a different optimization problem arises at each step.) This leads to the cost function $V(k)$ decreasing as k increases (under the no-disturbance, perfect-model assumption), which allows it to be used as a Lyapunov function, hence establishing stability.

Figure 6.2 shows a particular case for which closed-loop stability can be established despite the use of a finite horizon. This is when deadbeat behaviour is enforced, with settling to the set-point occuring within the prediction horizon. In this case the situation is similar to the infinite-horizon case, in that at each step the optimal trajectory is the 'tail' of the previously computed one. (It can also be regarded as a finite-horizon case

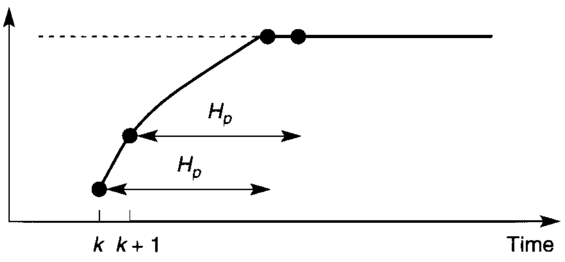

Figure 6.2 Finite horizon and deadbeat response.

with terminal constraint.) So in this case $V(k)$ is also decreasing, and hence stability can be established. This strategy for ensuring closed-loop stability was used in [KR92].

Let us now consider the details of the infinite-horizon formulation. We modify our standard set-up a little. First, we keep the weights constant across the prediction horizon, so that $Q(i) = Q$ and $R(i) = R$. Secondly, we penalize control *levels* as well as moves over an infinite horizon. Thirdly, we assume a 'regulator' formulation, namely that we want to drive the output to zero:

$$V(k) = \sum_{i=1}^{\infty} \left\{ \|\hat{z}(k+i|k)\|_Q^2 + \|\Delta\hat{u}(k+i-1|k)\|_R^2 + \|\hat{u}(k+i-1|k)\|_S^2 \right\} \tag{6.1}$$

although we still assume that only the first H_u control moves are non-zero:

$$\Delta\hat{u}(k+i-1|k) = 0 \qquad \text{for } i > H_u \tag{6.2}$$

We assume that $S > 0$ and $Q \geq 0$, $R \geq 0$. (In [RM93]it was assumed that the control levels were penalized, not the control moves. In [MR93b] both control levels and control moves were penalized. A proof which holds in the general case is given in [MR93a]. Unfortunately the weights R and S in our notation have the opposite meanings to those used in [RM93, MR93b, MR93a].)

Suppose initially that we have full state measurement, so that $x(k) = y(k)$, and that the plant is stable. Then the following argument shows that closed-loop stability is obtained for any $H_u > 0$, providing that the optimization problem is, and remains, feasible, and that the global optimum is found at each step.

Let $V^0(k)$ be the optimal value of the cost function at time k, let u^0 denote the computed optimal input levels, and let z^0 denote values of the controlled output obtained as a result of applying u^0. Note that, since $u^0(k+i) = u^0(k+H_u-1)$ for all $i \geq H_u$, and since the steady-state value of u^0 needed to keep z at 0 is 0 (according to the optimizer's internal model), the optimizer will certainly put $u^0(k+i) = 0$ for $i \geq H_u$, since an infinite cost would otherwise result. Thus we have

$$V^0(k) = \sum_{i=1}^{\infty} \|\hat{z}^0(k+i|k)\|_Q^2 + \sum_{i=1}^{H_u} \left\{ \|\Delta\hat{u}^0(k+i-1|k)\|_R^2 + \|\hat{u}^0(k+i-1|k)\|_S^2 \right\} \tag{6.3}$$

Now if we assume, as usual in proofs of nominal stability, that our model is exact and that there are no disturbances, then $z^0(k+1) = \hat{z}^0(k+1|k)$. Thus the value of V, evaluated at time $k+1$, but maintaining the same control sequence $\hat{u}^0(k+i|k)$ as computed at time k, would be

$$V(k+1) = V^0(k) - \|z^0(k+1)\|_Q^2 - \|\Delta\hat{u}^0(k|k)\|_R^2 - \|\hat{u}^0(k|k)\|_S^2 \tag{6.4}$$

But at time $k+1$ the new optimization problem, with initial condition $z^0(k+1) = C_z x^0(k+1)$ — which is known, since we assume full state measurement — is solved, so that

$$V^0(k+1) \leq V(k+1) \tag{6.5}$$
$$= V^0(k) - \|z^0(k+1)\|_Q^2 - \|\Delta\hat{u}^0(k|k)\|_R^2 - \|\hat{u}^0(k|k)\|_S^2 \tag{6.6}$$
$$< V^0(k) \tag{6.7}$$

the strict inequality being due to the assumption that $S > 0$.

To infer stability we must now show that $V^0(k+1) < V^0(k)$ implies that $||x(k)||$ is decreasing. Since we have assumed that $S > 0$, it is clear that $u^0(k)$ is decreasing (since $u^0(k) = \hat{u}^0(k|k)$), which implies that $x^0(k)$ is eventually decreasing, since we have assumed a stable plant. But we can relax the $S > 0$ condition, if we replace it by something else. For example, assuming that $Q > 0$ and observability of the state from z implies that $||x^0||$ must eventually be decreasing, even if $S = 0$, as in our standard formulation. So there are a number of possible assumptions which ensure that decreasing V^0 implies decreasing $||x^0||$, and that is enough to show that V^0 is a Lyapunov function for the closed loop, which shows, in turn, that the closed-loop system is stable.

Now we must consider the case of an unstable plant. The essential difference in this case is that the unstable modes must be driven to 0 within H_u steps. Otherwise these modes, which are uncontrolled for $i \geq H_u$, would become unbounded, giving an infinite cost value. But it is known that in general it is not possible to drive the states of a system to zero in less than n steps, where n is the state dimension. Since only the unstable modes need to be driven to zero, only n_u steps are actually needed, if n_u is the number of unstable modes. Thus we have to take $H_u \geq n_u$ for an unstable plant. We also need to ensure that the unstable modes *can* be driven to zero by some input. Technically, we have to assume that the pair (A, B) is *stabilizable* — namely that the unstable modes are controllable from the input; the stable modes do not have to be controllable, since they decay to zero anyway.

With these provisos, stability follows by the same argument as for stable plants, providing that feasibility is maintained.

When full state measurement is not available ($y(k) \neq x(k)$) an observer must be used to obtain the state estimate $\hat{x}(k|k)$. Subject to a few technical conditions, and assuming a perfect model as usual, $\hat{x}(k|k) \to x(k)$ as $k \to \infty$ if the observer dynamics are stable. Consequently, as k increases, the value of the optimal cost function $V^0(k)$ will approach the same value as in the case of $y(k) = x(k)$, and stability can again be inferred.

In order to be able to pose problems which are feasible over the whole infinite horizon, it is necessary to introduce two new parameters C_w and C_p, which are analogous to the horizons H_w and H_p, but which refer to the times over which output constraints are enforced. To be precise, suppose that the output constraints are of the form

$$z_{i,min} \le \hat{z}_i(k+j) \le z_{i,max} \qquad \text{for } j = C_w, C_w+1, \dots, C_p. \tag{6.8}$$

In order to apply the stability theorems, both C_w and C_p must be chosen large enough. C_w must be chosen large enough that the problem is feasible at time k. C_p must be chosen large enough that if the problem is feasible over the finite horizon up to time $k + C_p$, then it will remain feasible over the rest of the infinite horizon. In [RM93] it is shown that finite values of C_w and C_p always exist — although for unstable plants they depend on $x(k)$, in general.

6.2.2 Constraints and infinite horizons — stable plant

Now we need to consider how constrained predictive control problems can be solved, when the horizons are infinite. As we have already seen, for the regulator problem the computed input signal $\hat{u}(k+i-1|k)$ is necessarily zero for $i \ge H_u$. The cost function can therefore be written as

$$V(k) = \sum_{i=H_u+1}^{\infty} \|\hat{z}(k+i-1|k)\|_Q^2 + \sum_{i=1}^{H_u} \Big\{ \|\hat{z}(k+i-1|k)\|_Q^2 + \|\Delta\hat{u}(k+i-1|k)\|_R^2 + \|\hat{u}(k+i-1|k)\|_S^2 \Big\} \tag{6.9}$$

Consider the first term in this expression. Since $\hat{u}(k+i-1|k) = 0$ for $i \ge H_u$, we have

$$\hat{z}(k+H_u|k) = C_z\hat{x}(k+H_u|k) \tag{6.10}$$

$$\hat{z}(k+H_u+1|k) = C_zA\hat{x}(k+H_u|k) \tag{6.11}$$

$$\vdots$$

$$\hat{z}(k+H_u+j|k) = C_zA^j\hat{x}(k+H_u|k) \tag{6.12}$$

so that

$$\sum_{i=H_u+1}^{\infty} \|\hat{z}(k+i|k)\|_Q^2 = \hat{x}(k+H_u|k)^T \left[\sum_{i=0}^{\infty}(A^T)^i C_z^T Q C_z A^i\right] \hat{x}(k+H_u|k) \tag{6.13}$$

Now let

$$\bar{Q} = \sum_{i=0}^{\infty}(A^T)^i C_z^T Q C_z A^i \tag{6.14}$$

(the series converges if the plant is stable) then

$$A^T\bar{Q}A = \sum_{i=1}^{\infty}(A^T)^i C_z^T Q C_z A^i \tag{6.15}$$

$$= \bar{Q} - C_z^T Q C_z \tag{6.16}$$

This equation is known as the *matrix Lyapunov equation*, and it can be solved for $\bar{Q}$, given A, C_z and Q. (In *MATLAB* it can be solved using the function `dlyap` from the *Control System Toolbox*.) Furthermore, it is well known that $\bar{Q} \geq 0$ if $Q \geq 0$ and A has all its eigenvalues inside the unit disc, namely if the plant is stable. Thus the cost function can be written as

$$V(k) = \hat{x}(k + H_u|k)^T \bar{Q} \hat{x}(k + H_u|k) + \sum_{i=1}^{H_u} \Big\{ \|\hat{z}(k+i-1|k)\|_Q^2 + \|\Delta\hat{u}(k+i-1|k)\|_R^2 + \|\hat{u}(k+i-1|k)\|_S^2 \Big\} \tag{6.17}$$

This now looks like a predictive control problem formulated over a finite horizon of length H_u, with a terminal cost penalty — which shows, incidentally, that imposing a terminal equality constraint to obtain stability is unnecessarily restrictive.

If the controlled output is to follow a set-point with a non-zero final value, then it is necessary to replace the term $\|\hat{u}(k+i-1|k)\|_S^2$ by $\|\hat{u}(k+i-1|k) - u_s\|_S^2$ in the cost function in order to get a bounded cost, where u_s is a steady input value which is predicted to bring the controlled output to the required steady value r_s:

$$r_s = P(1)u_s = C_z(I - A)^{-1}Bu_s \tag{6.18}$$

But this is unnecessary if $S = 0$, as in our standard formulation. If a future target trajectory is known over a horizon H_p longer than H_u, then the approach outlined above is easily modified to handle this. It is simply necessary to rewrite the problem as a finite-horizon problem with horizon H_p rather than H_u, with a terminal cost of the form $\hat{x}(k + H_p|k)^T \bar{Q} \hat{x}(k + H_p|k)$.

In order to formulate this predictive control problem as a standard QP problem, we proceed as in Chapters 2 and 3. First we form the vector of predicted states $[\hat{x}(k+1|k)^T, \ldots, \hat{x}(k+H_u|k)^T]^T$ exactly as in equation (2.23) — but noting that the number of predictions needed now is H_u. The only other change that is needed is that $\mathcal{Q}$ in (3.3) should be replaced by

$$\mathcal{Q} = \underbrace{\begin{bmatrix} Q & 0 & \cdots & 0 & 0 \\ 0 & Q & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & Q & 0 \\ 0 & 0 & \cdots & 0 & \bar{Q} \end{bmatrix}}_{H_u \text{ blocks}} \tag{6.19}$$

(Note that the last block is different from the others.)

Example 6.3

In [BGW90] the following plant is used as an example of one which is difficult to control:

$$P(z) = \frac{-0.0539z^{-1} + 0.5775z^{-2} + 0.5188z^{-3}}{1 - 0.6543z^{-1} + 0.5013z^{-2} - 0.2865z^{-3}} \tag{6.20}$$

One state-space realization of this has

$$A = \begin{bmatrix} 0.6543 & -0.5013 & 0.2865 \\ 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \end{bmatrix} \qquad C_z = [-0.0536, 0.5775, 0.5188] \tag{6.21}$$

Suppose that $Q = I_3$, then

$$\bar{Q} = \begin{bmatrix} 1.2431 & -0.1011 & 0.1763 \\ -0.1011 & 0.8404 & 0.1716 \\ 0.1763 & 0.1716 & 0.3712 \end{bmatrix} \tag{6.22}$$

6.2.3 Constraints and infinite horizons — unstable plant

With an unstable plant there are two reasons why we cannot proceed exactly as before:

- We need to impose the constraint that the unstable modes must be at zero at the end of the control horizon.
- The infinite series in (6.14) does not converge.

We proceed by decomposing the plant into its stable and unstable parts, by means of an eigenvalue–eigenvector (Jordan) decomposition:[1]

$$A = WJW^{-1} \tag{6.23}$$

$$= [W_u, W_s] \begin{bmatrix} J_u & 0 \\ 0 & J_s \end{bmatrix} \begin{bmatrix} \tilde{W}_u \\ \tilde{W}_s \end{bmatrix} \tag{6.24}$$

The unstable and stable modes can be obtained from this as

$$\begin{bmatrix} \xi_u(k) \\ \xi_s(k) \end{bmatrix} = \begin{bmatrix} \tilde{W}_u \\ \tilde{W}_s \end{bmatrix} x(k) \tag{6.25}$$

and they evolve according to

$$\begin{bmatrix} \xi_u(k+1) \\ \xi_s(k+1) \end{bmatrix} = \begin{bmatrix} J_u & 0 \\ 0 & J_s \end{bmatrix} \begin{bmatrix} \xi_u(k) \\ \xi_s(k) \end{bmatrix} + \begin{bmatrix} \tilde{W}_u \\ \tilde{W}_s \end{bmatrix} Bu(k) \tag{6.26}$$

The terminal equality constraint on the unstable modes becomes

$$\xi_u(k + H_u) = \tilde{W}_u x(k + H_u) = 0 \tag{6.27}$$

[1] There is no numerically stable algorithm for performing the Jordan decomposition. But there are numerically stable algorithms for finding the subspaces of the state space spanned by the unstable and stable modes, respectively [Mac89, Chapter 8]. Since it would be enough to find these spaces, and the corresponding projection matrices, it would be advisable to do this in an implementation.

Because of this constraint, only the stable modes contribute to the infinite sum in (6.9), which therefore has a finite value:

$$\sum_{i=H_u+1}^{\infty} \|\hat{z}(k+i-1|k)\|_Q^2 = \sum_{i=H_u+1}^{\infty} \|C_z \tilde{W}_s \hat{x}(k+i-1|k)\|_Q^2 \tag{6.28}$$

$$= \hat{x}(k+H_u|k)^T \bar{Q} \hat{x}(k+H_u|k) \tag{6.29}$$

where, proceeding as before, we find that

$$\bar{Q} = \tilde{W}_s^T \Pi \tilde{W}_s \tag{6.30}$$

and Π is the solution of the matrix Lyapunov equation:

$$\Pi = W_s^T C_z^T Q C_z W_s + J_s^T \Pi J_s \tag{6.31}$$

Again $\mathcal{Q}$ should be as in (6.19), with $\bar{Q}$ being obtained from (6.30). Predictions are obtained as before.

The significant difference now is the appearance of the equality constraint (6.27). This can be simply added to the QP formulation as an equality constraint. But, as pointed out by Scokaert in the GPC context [Sco97], this equality constraint removes some degrees of freedom from the optimization (as many as there are unstable modes), and it is possible to reparametrize the problem so that the optimization is over a smaller number of decision variables.

In the state-space setting this can be done as follows. From equation (2.66) we see that

$$\hat{\xi}_u(k+H_u|k) = \tilde{W}_u \hat{x}(k+H_u|k) \tag{6.32}$$

$$= \tilde{W}_u \left\{ A^{H_u}\hat{x}(k|k) + \sum_{i=0}^{H_u-1} A^i B u(k-1) + \left[\sum_{i=0}^{H_u-1} A^i B \cdots B \right] \Delta\mathcal{U}(k) \right\} \tag{6.33}$$

This represents n_u equations, if there are n_u unstable modes. Now we can use these equations to express any n_u of the variables in the vector $\Delta\mathcal{U}(k)$ in terms of the remaining ones, and thus eliminate these variables from the optimization. Scokaert [Sco97] considers single-input systems, and expresses the last n_u elements in $\Delta\mathcal{U}(k)$, namely the last n_u elements of the computed input signal, in terms of the other. The same can be done with multi-input systems, but other possibilities may be equally sensible, such as selecting the last n_u elements of the jth input signal $\Delta\hat{u}_j(k+i|k)$ for $i = H_u - n_u, H_u - n_u + 1, \ldots, H_u - 1$. But it is not clear whether it matters how these n_u elements are selected, even in the single-input case. The computational saving is small — typically one has only one or two unstable modes — so it may be as well to let the optimizer decide how to 'use' the equality constraints.[2]

[2] With multivariable unstable models a state-space formulation is almost essential. It is almost impossible to recover the correct number of unstable modes from a transfer function matrix, because of numerical sensitivities.

6.3 Fake algebraic Riccati equations

In [BGW90, Mos95] it is shown that stability can sometimes be guaranteed with finite horizons, even when there is no explicit terminal constraint. The finite horizon predictive control problem is associated with a time-varying Riccati difference equation which is intimately related to the optimal value of the cost function. The Fake Algebraic Riccati Technique replaces this equation with an algebraic (time-invariant) Riccati equation which looks like the one that occurs in infinite-horizon LQ problems. If this equation has properties analogous to those which occur for the (real) algebraic Riccati equation of infinite-horizon control, then stability can be inferred. In order to develop the details we shall need to refer to Mini-Tutorial 7, on linear-quadratic optimal control.

We need to note some differences between the finite-horizon problem treated in the Mini-Tutorial and our standard predictive control problem. First, note that the indices on Q and R 'run backwards' in the Mini-Tutorial, in the sense that $x(t)$ is weighted by Q_{N-1}, whereas $x(t+N-1)$ is weighted by Q_0. Similarly, the state-feedback gain applied at time t is K_{N-1}, while that applied at time $t+N-1$ is K_0.

Secondly, the LQ cost function weights $||x(k)||$ and $||u(k)||$, whereas in our standard problem we weight $||z(k)||$ and $||\Delta u(k)||$. We can put our standard problem into the LQ framework as follows. Introduce the augmented state ξ, and the new matrices $\tilde{A}$ and $\tilde{B}$:

$$\xi(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix} \qquad \tilde{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \qquad \tilde{B} = \begin{bmatrix} B \\ I \end{bmatrix} \tag{6.34}$$

then the two models

$$x(k+1) = Ax(k) + Bu(k) \qquad \text{and} \qquad \xi(k+1) = \tilde{A}\xi(k) + \tilde{B}\Delta u(k) \tag{6.35}$$

are equivalent (with our usual definition $\Delta u(k) = u(k) - u(k-1)$). Furthermore, $||z(k)||_Q = ||\xi(k)||_{\tilde{Q}}$, where

$$\tilde{Q} = \begin{bmatrix} C_z^T Q C_z & 0 \\ 0 & 0 \end{bmatrix} \tag{6.36}$$

(where $z(k) = C_z x(k)$ — recall (2.3)). So our standard predictive control problem is the same as the LQ problem, if we replace A and B in the plant model by $\tilde{A}$ and $\tilde{B}$, and the weight Q_{N-j-1} by $\tilde{Q}_{N-j-1}$ in the cost function. (The weight R_{N-j-1} stays unchanged in the cost function, but of course it is now interpreted as the weight which penalizes $\Delta u(k)$.) The optimal control is obtained as $\Delta u(t+N-j) = -\tilde{K}_{j-1}\xi(t+N-j)$, where $\tilde{K}$ is the state-feedback gain matrix obtained from (6.39) when these substitutions have been made.

Thirdly, a single horizon N is used in the LQ cost function, whereas we use the two horizons H_p and H_u. This is no problem so long as $H_u \le H_p$, because we can always set $R_{N-j-1} = \infty$ for $j \ge H_u$, which will effectively reduce the control horizon to H_u.[3]

[3] Note that the Riccati difference equation (6.38) becomes the Lyapunov equation $P_{j+1} = A^T P_j A + Q_j$ when $R_j = \infty$.

Finite horizon:

We consider the problem of having an initial state $x(t)$ at time t, and finding a control sequence which will minimize the finite-horizon cost function

$$V_N(x(t)) = \sum_{j=0}^{N-1}\{||x(t+j)||^2_{Q_{N-j-1}} + ||u(t+j)||^2_{R_{N-j-1}}\} + ||x(t+N)||^2_{P_0} \tag{6.37}$$

where $Q_j \geq 0$, $R_j \geq 0$ and $P_0 \geq 0$. The optimal control is found as follows [BH75, KR72, Mos95, BGW90]: Iterate the *Riccati Difference Equation*

$$P_{j+1} = A^T P_j A - A^T P_j B(B^T P_j B + R_j)^{-1} B^T P_j A + Q_j \tag{6.38}$$

and form the state-feedback gain matrix

$$K_{j-1} = (B^T P_{j-1} B + R_{j-1})^{-1} B^T P_{j-1} A. \tag{6.39}$$

The optimal control sequence is given by

$$u(t+N-j) = -K_{j-1} x(t+N-j) \tag{6.40}$$

and the optimal value of the cost (6.37) obtained in this way is $V_N(x(t))^0 = ||x(t)||^2_{P_N}$.

Infinite horizon:

Now suppose that the horizon becomes infinite ($N \to \infty$), we consider the infinite-horizon cost

$$V_\infty(x(t)) = \lim_{N\to\infty} V_N(x(t)) \tag{6.41}$$

and the weighting matrices are constant: $Q_j = Q$, $R_j = R$. If $R > 0$, the pair (A, B) is stabilizable, and the pair $(A, Q^{1/2})$ is detectable, then $P_j \to P_\infty \geq 0$ (as $j \to \infty$), and (6.38) is replaced by the *Algebraic Riccati Equation*:

$$P_\infty = A^T P_\infty A - A^T P_\infty B(B^T P_\infty B + R)^{-1} B^T P_\infty A + Q. \tag{6.42}$$

Hence

$$K_j \to K_\infty = (B^T P_\infty B + R)^{-1} B^T P_\infty A \tag{6.43}$$

and the optimal control becomes the constant state-feedback law:

$$u(t+j) = -K_\infty x(t+j). \tag{6.44}$$

It can be proved that this feedback law is stabilizing — otherwise the cost $V_\infty(x(t))$ would be infinite — so that all the eigenvalues of the closed-loop state-transition matrix $A - BK_\infty$ lie strictly within the unit circle (if the stated conditions hold). The optimal cost is now $V_\infty(x(t))^0 = ||x(t)||^2_{P_\infty}$.

Mini-Tutorial 7 Linear quadratic optimal control.

When we apply the receding horizon control strategy, we always apply the first part of the finite-horizon control strategy, which means that we apply the constant state-feedback law

$$\Delta u(t) = -\tilde{K}_{N-1}\xi(t) \tag{6.45}$$

(assuming no constraints for the present discussion). The big question now is, when can this law be guaranteed to be stabilizing? That is, when will all the eigenvalues of $\tilde{A} - \tilde{B}\tilde{K}_{N-1}$ be guaranteed to lie inside the unit circle?

We now revert to using the notation which appears in the LQ problem, namely we use A, Q, K etc., with the understanding that these are replaced by $\tilde{A}$, $\tilde{Q}$, $\tilde{K}$ etc. when necessary (see Exercise 6.6.). From the *infinite-horizon* LQ problem, it is apparent (subject to the technical conditions: $Q \geq 0$, (A, B) stabilizable, $(A, Q^{1/2})$ detectable) that so long as the algebraic Riccati equation (6.42) has a positive semi-definite solution ($P \geq 0$), and a constant state feedback gain matrix is obtained from this solution by the formula (6.43), then the resulting feedback law will be stable. In the finite horizon problem we have the Riccati difference equation (6.38) instead of (6.42). Let us suppose that we use constant weighting matrices: $Q_{N-j-1} = Q$ and $R_{N-j-1} = R$, so that (6.38) becomes

$$P_{j+1} = A^T P_j A - A^T P_j B(B^T P_j B + R)^{-1} B^T P_j A + Q. \tag{6.46}$$

Now introduce the matrix

$$\mathcal{Q}_j = Q - (P_{j+1} - P_j) \tag{6.47}$$

then, substituting for P_{j+1} in (6.46) gives

$$P_j = A^T P_j A - A^T P_j B(B^T P_j B + R)^{-1} B^T P_j A + \mathcal{Q}_j. \tag{6.48}$$

But this is an *algebraic* Riccati equation (because the same matrix P_j appears on both left- and right-hand sides). It has been called the *Fake Algebraic Riccati Equation* (or *FARE*) because it does not arise directly from an infinite-horizon LQ problem. If this equation has a solution $P_j \geq 0$, *and if* $\mathcal{Q}_j \geq 0$, then applying the *constant* state-feedback gain K_j (i.e. apply the same gain K_j at each time) will give a stable feedback law, where K_j is obtained from P_j using (6.43). So, one way of ensuring a stable (unconstrained) predictive control law is to ensure that the fake algebraic Riccati equation

$$P_{N-1} = A^T P_{N-1} A - A^T P_{N-1} B(B^T P_{N-1} B + R)^{-1} B^T P_{N-1} A + \mathcal{Q}_{N-1} \tag{6.49}$$

(with A replaced by $\tilde{A}$ etc.) with $\mathcal{Q}_{N-1} \geq 0$ has a solution $P_{N-1} \geq 0$. Note that standard algorithms are available for solving the algebraic Riccati equation; in *MATLAB*'s *Control System Toolbox* it can be solved using the function `dare`.

Since we start with $Q \geq 0$, we will have $\mathcal{Q}_{N-1} \geq Q \geq 0$ providing that $P_N \leq P_{N-1}$. In [BGW90] the following monotonicity property is proved: if $P_{t+1} \leq P_t$, then $P_{t+j+1} \leq P_{t+j}$ for all $j \geq 0$. This shows that stability can be ensured by introducing a suitable P_0 into the predictive control problem, which is the same as introducing a suitable terminal cost (see (6.37)). In fact, $\mathcal{Q}_0 \geq Q$ is sufficient to give stabililty. In [BGW90] it is shown that it is not enough just to make P_0 'very large', although there are some specific ways of making P_0 'large' which have the desired effect. One particular way of doing this is to set $W_0 = P_0^{-1} = 0$, and propagating $W_j = P_j^{-1}$ instead of P_j.[4] Note that this is a particular way of making P_0 infinite, and hence is equivalent to imposing a terminal constraint, which provides an alternative explanation of why this choice of P_0 guarantees stability.

[4] As is done in 'information filter' versions of the Kalman filter. and in recursive least-squares.

There is one further technicality involved: the pair $(A, \mathcal{Q}_{N-1}^{1/2})$ should be detectable. But it is shown in [BGW90] that, if $(A, Q^{1/2})$ is detectable and $\mathcal{Q}_{N-1} \geq Q$, then $(A, \mathcal{Q}_{N-1}^{1/2})$ is detectable. So there is no problem. There are several other related and interesting results in [BGW90]. In particular, the Fake Algebraic Riccati Equation (FARE) approach can be interpreted as producing a predictive control law which is the same as the law which would result from posing a particular infinite-horizon problem, which 'explains' why the approach is effective. Also the monotonicity property mentioned above is related to a similar monotonicity property of the optimal cost, which implies that the optimal cost can serve as a Lyapunov function, and hence gives another explanation of the efficacy of this approach.

It should be emphasized that the FARE approach gives sufficient but not necessary conditions for stability of predictive control laws. In particular, if the GPC formulation of predictive control is used, in which $P(0)$ is fixed as $P(0) = Q = C_y^T C_y$ and $R = \lambda I$, and if the prediction and control horizons are made the same, $H_p = H_u$, then P_j can be shown to be monotonically nondecreasing, rather than nonincreasing [BGW90]. Nevertheless, GPC laws can be stabilizing, even with this choice of horizons.

We have examined three ways of ensuring stability: imposing terminal constraints, using infinite horizons, and using the FARE approach. We have seen that they can all be regarded as modifying the weight on the terminal control error in some way. In that sense the three approaches are not so different from each other. But we can note that the first and most obvious way of achieving stability, namely imposing terminal constraints, is in general unnecessarily severe — one does not have to make the terminal weight infinitely large in order to obtain stability. We also mention again the survey paper [MRRS00], in which alternative common elements are identified for these and other ways of ensuring stability while using predictive control.

The authors of [BGW90] conclude by suggesting the use of infinite horizons, so perhaps the FARE method is only of historical interest now. However, recently the method has been generalized to a *Fake Hamilton–Jacobi–Bellman Equation* method for obtaining stability with nonlinear internal models in unconstrained problems [MS97a].

6.4 Using the Youla parametrization

If the plant is stable, then it is known that the feedback system shown in Figure 6.3 is internally stable if and only if the block labelled *Youla parameter* is stable, providing that the internal model is an exact model of the plant [Mac89, ZDG96]. This holds under rather general conditions; all the blocks shown in the figure can be nonlinear operators, for example. Furthermore, for linear systems it is known that *every* stable feedback system can be represented in this way (if the plant is stable) — so nothing is lost by assuming this form. (The term 'parameter' here comes from the fact that as the Youla parameter ranges over all stable systems, so it generates all possible stabilizing controllers for the given plant. But it is not just a number!)

Let the Youla parameter shown in Figure 6.3 have transfer function (matrix) $Q(z)$, and let both the plant and the internal model have transfer function $P(z)$. We have

$$u(z) = Q(z)[y(z) - P(z)u(z)] \tag{6.50}$$

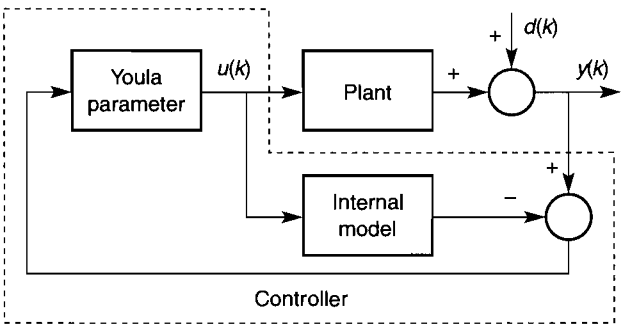

Figure 6.3 Youla parametrization of all stable feedback systems (if the plant is stable).

so that

$$[I + Q(z)P(z)]u(z) = Q(z)y(z) \tag{6.51}$$

and hence

$$u(z) = [I + Q(z)P(z)]^{-1}Q(z)y(z) \tag{6.52}$$

Comparing this with the more conventional representation of a feedback system shown in Figure 7.1, in which we have $u(z) = -K(z)H(z)y(z)$ (if we ignore the reference signal r and the noise n), we see that the following correspondence holds:

$$K(z)H(z) = -[I + Q(z)P(z)]^{-1}Q(z) \tag{6.53}$$

From this, it is very easy to obtain the correspondence in the other direction:

$$Q(z) = -K(z)H(z)[I + P(z)K(z)H(z)]^{-1} \tag{6.54}$$

If the plant is unstable (and linear) then closed-loop stability can still be characterized by the stability of the Youla parameter, but its definition in terms of the controller becomes more complicated [Mac89, ZDG96]. Figure 6.3 has to be replaced by a more complicated block diagram.

Mathematically, the set of all stable Youla parameters is an infinite-dimensional space, so optimizing over it can only be done for particular problems (such as unconstrained *LQG* or H_∞ problems). But it is also possible to parametrize some subset of all stable systems by a finite number of parameters, which converts the optimization into a finite-dimensional one. Furthermore, if the parametrization is linear and the cost function is quadratic then the problem remains convex. This is due to the remarkable property of the Youla parameter, that all the closed-loop transfer functions which are of interest are *affine* in this parameter — that is, they all have the form $XQY + Z$, where Q is the Youla parameter, and X, Y and Z are some (fixed) transfer function matrices.[5] There have been some proposals to solve the constrained predictive control problem with this approach [RK93, vdBdV95].

[5] One consequence of this is that the minimization of any induced norm of a closed-loop transfer function, $||XQY + Z||$, is a convex optimization problem. The book [BB91] is devoted to exploiting this.

Example 6.4

All SISO FIR systems of order n are parametrized linearly by the n terms of their pulse responses $H(0), H(1), \ldots, H(n-1)$.

From Figure 6.3 it is apparent that $u(z) = Q(z)d(z)$, where d is the output disturbance. But if $Q(z)$ is an FIR filter of order n then $Q(z) = H(0) + z^{-1}H(1) + \ldots + z^{-n}H(n)$. Hence the control signal can be computed as

$$\hat{u}(k+j|k) = \sum_{i=0}^{n} H(i)\hat{d}(k+j-i|k) \tag{6.55}$$

or, equivalently,

$$\Delta\hat{u}(k+j|k) = \sum_{i=0}^{n} H(i)[\hat{d}(k+j-i|k) - \hat{d}(k+j-i-1|k)] \tag{6.56}$$

Note that $\Delta\hat{u}(k+j|k)$ is linear in the pulse response coefficients $H(0), \ldots, H(n)$. Furthermore, the disturbance estimates $\hat{d}(k+j-i|k)$ do not depend on future inputs $\hat{u}(k+j|k)$. So substituting (6.56) into (2.66), for instance, makes the state predictions depend linearly on the pulse response coefficients, and hence the cost function (2.19) is quadratic in these coefficients. So the problem remains quadratic.

Note, however, that the constraints on the input levels and the input moves are now related to the 'decision variables', namely to the pulse response coefficients, through transfer functions.

Exercises

6.1 Simulate the model and control law from Example 6.1 using *MATLAB*, and check that the closed loop is unstable.

Recommended procedure: create the state-space matrices `a,b,c,d` corresponding to the *closed-loop* system, then create a Matlab object representing this closed-loop system using:

```
clsys=ss(a,b,c,d,1);
```

(Use `help ss` to see what this does.) Then get the response from initial condition `x0=[1;0]` — or some other initial condition — by typing

```
initial(clsys,x0);
```

6.2 Consider the plant given in Example 6.1 again. But now suppose that the prediction horizon is $H_p = 2$, with the cost being

$$V(k) = \hat{x}(k+1|k)^T \begin{bmatrix} 1 & 2 \\ 2 & 6 \end{bmatrix} \hat{x}(k+1|k) + \hat{x}(k+2|k)^T \begin{bmatrix} 1 & 2 \\ 2 & 6 \end{bmatrix} \hat{x}(k+2|k)$$

(a) Keep $H_u = 1$, so that only $u(k)$ is to be optimized, with the assumption that $u(k+1) = u(k)$. Show that the predictive control law is $u^0(k) = -\frac{1}{6}x_1(k)$, and hence that the closed loop is stable.

(b) Now let $H_u = 2$, so that $u(k)$ and $u(k+1)$ have to be optimized. By setting both derivatives $(\partial V/\partial u(k))$ and $(\partial V/\partial u(k+1))$ to zero (or $\nabla_u V = 0$ if you are happy with that notation) show that the predictive control law is $u^0(k) = -\frac{2}{3}x_1(k)$, and hence that the closed loop is stable.

(c) Simulate the closed-loop behaviours for these two cases, using *MATLAB*.

(Note that the *Model Predictive Control Toolbox* does not allow non-diagonal weights.)

6.3 Consider Example 6.1 again, and Exercise 6.2. Solve and simulate them using the *Model Predictive Control Toolbox*. However, the *Toolbox* only allows diagonal weighting matrices to be specified. If you have solved Exercise 3.7, then you can keep the original non-diagonal weight. Otherwise change the weight to `diag(1,6)` (that is, just keep the diagonal elements). Unfortunately this will not give the same results, but it is still a good exercise to see how the results differ for the cases

- $H_p = H_u = 1$
- $H_p = 2,\ H_u = 1$
- $H_p = H_u = 2$

6.4 Verify that $\bar{Q}$ is given by (6.30) and (6.31) in the case of an unstable plant.

6.5 For the plant

$$x(k+1) = 0.9x(k) + 0.5u(k), \qquad y(k) = x(k)$$

suppose that the infinite-horizon case ($H_p = \infty$) is to be solved.

(a) Formulate an equivalent finite-horizon problem if $Q(i) = 1$, $R(i) = 0$, and $H_u = 2$.

(b) Find the predictive controller and simulate the set-point response.

(c) What feature of the *Model Predictive Control Toolbox* would prevent you from following this procedure in general?

6.6 The infinite-horizon LQ problem requires the assumptions that (A, B) is a stabilizable pair and that $(A, Q^{1/2})$ is a detectable pair. To put the standard predictive control problem into the LQ framework, we need to use the matrices $\tilde{A}$, $\tilde{B}$, and $\tilde{Q}$, as defined in (6.34) and (6.36). Show that, if $(\tilde{A}, \tilde{B})$ is stabilizable then so is the pair (A, B), and that if $(\tilde{A}, \tilde{Q}^{1/2})$ is detectable then so is the pair $(A, Q^{1/2}C)$. (Note that you can take $\tilde{Q}^{1/2} = [Q^{1/2}C, 0]$.)

6.7 Consider the finite-horizon LQ problem for a plant with

$$A = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.57}$$

Verify that if

$$P_{N-1} = \begin{bmatrix} 16 & 0 \\ 0 & 16 \end{bmatrix} \tag{6.58}$$

and $R = 1$ then the corresponding state-feedback gain matrix is $K_{N-1} = [0, 1.8824]$ and that this gives a stable feedback law. Show, nevertheless, that this solution corresponds to a value of Q_{N-1} which is not positive-definite. (This underlines the fact that the FARE approach gives a sufficient, but not necessary, condition for stability.)

9.1 In the design developed in Section 9.1.4, 'blocking' was used, corresponding to the parameter `M=[1,2,4,8]` in *Model Predictive Control Toolbox* function `scmpc` (or `scmpc2`). Investigate use of the simpler alternative of not using blocking, but still retaining five decision times (namely H_u =M=5). Simulate the same scenarios as shown in Figures 9.5–9.9. Show that the responses are not very different from the ones shown in these figures. (Use the *Model Predictive Control Toolbox* and the function `scmpc2` available from this book's web site.)

9.2 Continue exercise 9.1: Show that the singular values of the linear controller obtained in unconstrained operation are not affected much by abandoning 'blocking'. (Use the *Model Predictive Control Toolbox*'s functions `smpccon, smpccl, svdfrsp`.)

9.3 Investigate by simulation the use of longer prediction horizons than used in Section 9.1.4, as regards both resulting performance of the closed-loop system and computation time requirements.

9.4 Investigate alternative choices of the prediction horizon H_p, the control horizon H_u, and the weights Q and R, for the evaporator. Consider how well the controller deals with disturbances on the feed flow rate F_1, the feed concentration X_1, and the cooling water temperature T_{200}. (To simulate disturbances, attach suitable `Source` blocks in place of the input ports on the disturbance variables in the *Simulink* model `evaporator.mdl`.)

Suppose the evaporator is to operate with product concentrations (X_2) of 25%, 15%, and 40%, all at an operating pressure of 50 kPa. Investigate whether you can find a single set of tuning parameters which gives good performance at all these concentrations, including transitions between them.

Is there a significant advantage in re-linearizing the internal model as the product concentration changes?

Note: Use, and modify as necessary, the file `simevap.m`*, which is available on this book's web site.*

9.5 Simulate the same scenario as shown in Figures 9.18–9.20, but with the separator level treated as a *zone* variable — namely with no set-point, but only the constraints $0.5 < L_2 < 1.5$. (See Section 5.5.)

9.6 It makes more sense to re-linearize the internal model when the operating point has changed significantly, rather than at regular intervals of time. Examine the *MATLAB* script file `simevap.m`. It allows the evaporator model to be re-linearized at regular intervals of time. Modify it so that the model is only re-linearized if the state has changed by more than some threshold amount. (You will still need to interrupt the simulation at regular intervals, then test whether the state has changed significantly.)

Using the modified file, simulate the same scenario as shown in Figures 9.18–9.20, re-linearizing the model whenever the product concentration X_2 has changed by more than 5%.

Chapter 10

Perspectives

This chapter introduces a few topics which are potentially very important, but which are mostly still at the research stage.

10.1 Spare degrees of freedom

10.1.1 Ideal resting values

If the plant has more inputs than outputs, then a particular constant set-point y_s can be held by a whole family of constant input vectors u_s. There may be good reasons for preferring one of these, usually because there are different economic costs associated with the different control inputs. (Reducing air flow may be cheaper than burning more fuel, for instance.) The traditional story in optimal control has been to say that such preferences should be reflected in the choice of the $R(i)$ matrices. But in practice that is not so easy as it sounds, since $Q(i)$ and $R(i)$ have complicated effects on the dynamic performance. The desirable constant values of the inputs are usually determined at the same time as the set-points are determined, using static optimization — usually by solving an LP problem.

If for some reason set-points are known, but the control inputs are not determined uniquely, then the solution sometimes adopted is to first do a steady-state least-squares optimization, to find the best combination of steady-state inputs which will give a desired set-point vector. If the numbers of inputs and outputs are equal and the steady-state gain matrix is nonsingular, then this is unnecessary because the solution is unique. If there are more inputs than outputs, then the equation

$$y_s - S(\infty)W_u u_s = 0 \tag{10.1}$$

is solved in a least-squares sense, where $S(\infty)$ is the steady-state gain matrix and W_u is a diagonal weighting matrix. This gives the solution with the smallest value of $||W_u u_s||$. (If there are more outputs than inputs then it is usually impossible to reach the correct set-point, and in this case the equation

$$W_y(y_s - S(\infty)u_s) = 0 \tag{10.2}$$

is solved in a least-squares sense.)

When there are more inputs than outputs the elements of u_s are called the *ideal resting values* of the plant inputs for the given set-points. Once they have been found the cost function used in the predictive control problem formulation is augmented with the term $\sum_{i=0}^{H_u-1} ||\hat{u}(k+i|k) - u_s||^2_{\Sigma(i)}$, to encourage the inputs to converge to these 'ideal' values. A danger of this is that, if the steady-state gains are not known accurately, or if there are constant disturbances, then the computed values of u_s will not be exactly correct for attaining y_s at the output, and offset-free tracking may be lost as a result.

10.1.2 Multiobjective formulations

It is difficult to express all the objectives of control into a single cost function. For example, assigning a high weight to the control of a particular output is not really the same as assigning it a high priority. It does not express the objective 'Keep output j near its set-point only if you can also keep output ℓ at its set-point'. A 'multiobjective' formulation makes it easier to do this. This can be applied also in situations in which it may be impossible to meet all the objectives simultaneously. There are several approaches to multiobjective optimization, but one well-suited to the situation in which there are various priorities for various objectives is to have a hierarchy of optimization problems. The most important one is solved first, then this solution is used to impose equality constraints when performing the second optimization, and so on. For example, the 'ideal resting value' problem treated above may be better solved by making the achievement of set-points the primary objective, and the achievement of ideal resting values a secondary objective to be pursued only when the controlled outputs are close to their set-points, and imposing the equality constraints that the outputs should equal their set-points.

The complexity of the optimization problem to be solved may not increase, and may even decrease, as a result of adopting such a strategy. Predictive control problems solved in this way can keep a QP formulation at each step, and it is generally quicker to solve several small problems than one large equivalent problem. In particular, the first, most important problems to be solved have the smallest number of conflicts and the largest number of degrees of freedom, so have the best prospects of a good solution being found, and of this being done quickly. If one runs out of computing time before the next control signal must be produced, then the still-unsolved problems are the least important ones.

10.1.3 Fault tolerance

Predictive controllers seem to offer very good possibilities for fault-tolerant control. If a sensor of a controlled output fails, for example, it is possible to abandon control of that output by removing the corresponding output from the cost function (by setting the corresponding elements of $Q(i)$ to zero). An actuator jam is easily represented by changing the constraints ($|\Delta u_j| = 0$). If a failure affects the capabilities of the plant, then it is possible to change the objectives, or the constraints, or both, accordingly. This is also possible with other control strategies, but it is particularly clear, in the predictive control formulation, how the required changes should be introduced into the controller. This is mainly due to the fact that the control signal is re-computed at each step by solving an optimization problem, so that *one can make changes in the problem formulation*. This is a very different proposition to changing the gains or time constants in a pre-computed controller.[1]

Note that it may not be an easy problem to know how to change the problem formulation if a failure occurs. All that can be said is that, if an appropriate change is known, then it is easy to introduce it and implement it in the predictive control framework.

In [Mac97] it is argued that predictive control offers the possibility of a generic approach to achieving fault tolerance by reconfiguring the controller in response to a serious failure. This proposal relies on combining predictive control with the increasing use of detailed 'high-fidelity' first-principles models to represent complex processes, and with progress in the area of Fault Detection and Identification (FDI).

But it has also been noticed that predictive control has certain fault-tolerant properties even in the absence of any knowledge of the failure. In [Mac98] it is shown that predictive control, with the standard 'DMC' model of disturbances and with hard constraints on the input levels, can automatically redistribute the control effort among actuators, if there are more actuators than controlled outputs, and one (or more) actuator fails.

Some of these capabilities are illustrated in Figure 10.1, which is taken from [Mac97]. It shows a predictive controller responding to a step demand in the yaw angle of a large civilian aircraft when the rudder is jammed.

Case 1 is the normal response when the rudder is working correctly. Case 2 shows the response when the rudder is jammed, and the predictive control problem formulation has no constraints; even in this situation the set-point is eventually achieved, which indicates that other control strategies could achieve this too. Case 3 shows the response when the controller is aware of the usual $\pm 20^\circ$ rudder angle limit. The response is now faster, because after a few seconds the controller 'thinks' that it has driven the rudder to the limit, and starts to move the other control surfaces; but the controller has no knowledge of the failure in this case. Case 4 shows the response when the rudder failure is communicated to the controller by reducing the 'rudder move' constraint to zero. This makes surprisingly little difference in this case (the curves for Cases 3 and 4 are indistinguishable); the only difference is that the controller knows immediately that it cannot move the rudder, whereas in Case 3 it took about 3 seconds before it realized this. This does not make much difference to the complete manoeuvre. Case 5 shows

[1] However, it might be possible to re-compute on-line controllers for other strategies, which have traditionally been pre-computed off-line, rather like 'classical' adaptive control proposals.

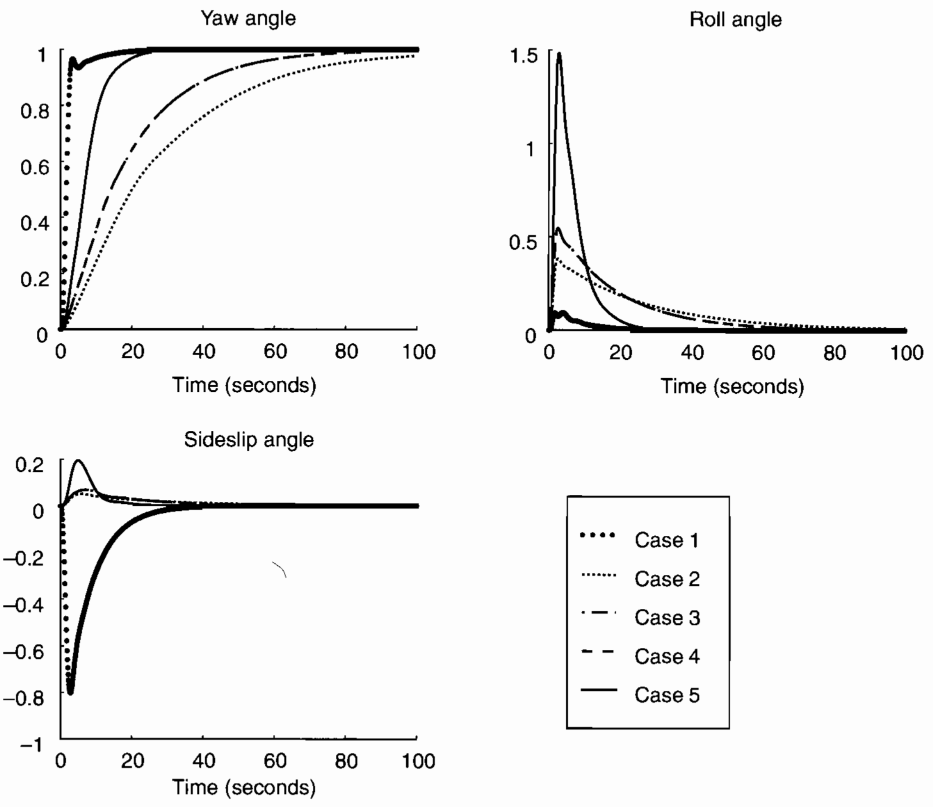

Figure 10.1 Execution of a step change in yaw angle with a jammed rudder.

that better response can be achieved by changing the weights — in this case the weight on keeping the roll angle set-point was reduced, so that the controller made more use of roll angle to reach its objective; this also shows a potential difficulty of this approach to fault-tolerant control, namely that in general some re-tuning may be necessary, and this may not be an easy thing to do on-line.

Another example of the 'daisy-chaining' property arises in connection with a plant for producing liquefied natural gas (LNG), shown in Figure 10.2, which is taken from [Row97]. It has six inputs: the compressor speed, and the positions of the five valves; and five outputs: the levels of liquid in the two separators, and the exit temperatures of the three heat exchangers (of the natural gas). The compressor speed is limited to the range 25 to 50 revs/sec; the valve fully closed to fully open range is represented by −500 to +500. In the example considered the sampling interval is 10 sec, and at this long interval the input rates are effectively unlimited. Each separator level is limited to the range −2 to +2 m (relative to the operating point), and the evaporator exit temperatures are limited to the ranges −6 to +30 K, −13 to +7 K, and −11 to +4 K, relative to their nominal values.

Figure 10.3 shows the simulated response to a step demand of +10 K on the exit

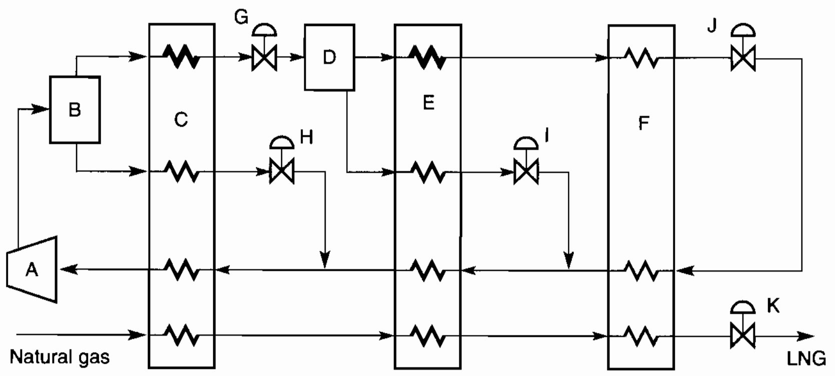

Figure 10.2 LNG liquefaction plant.

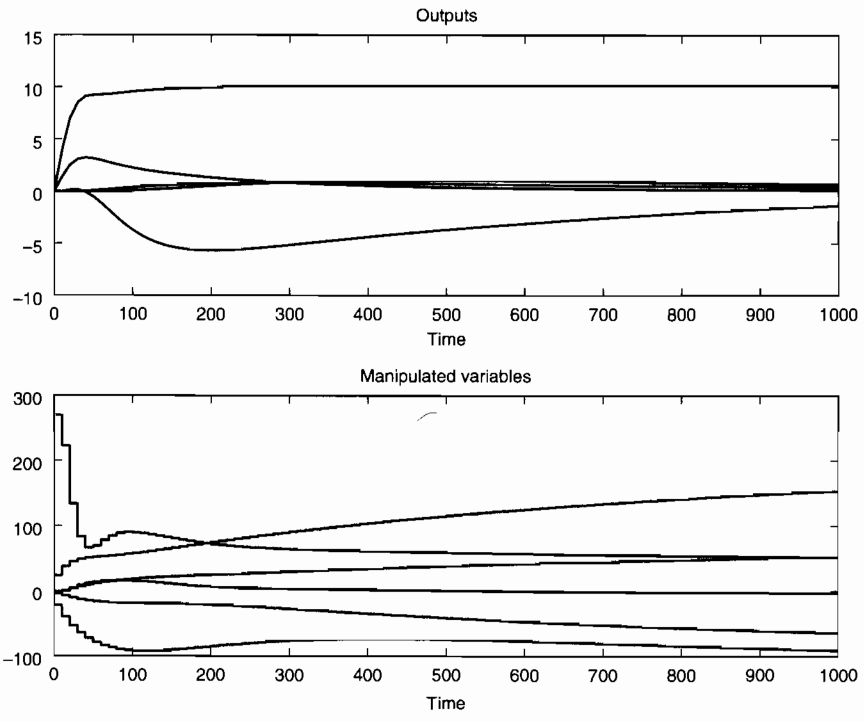

Figure 10.3 LNG plant: response in normal operation.

temperature of heat exchanger C when there is no failure, the set-points for both separators and heat exchanger F are zero, and the exit temperature of heat exchanger E is not controlled, providing it remains within its constraints. The horizons are $H_p = 80$ and $H_u = 5$.

Figure 10.4 shows the response in the same situation, but with valve H stuck at