PREDICTIVE CONTROL
WITH CONSTRAINTS
SOLUTIONS MANUAL

J.M.Maciejowski
Cambridge University Engineering Department

3 December 2001

MATLAB and *Simulink* are registered trademarks of *The MathWorks, Inc.*

World-Wide Web site for this book: `http://www.booksites.net/maciejowski/`

Contents

Chapter 1

Introduction

1.1 Assuming a perfect model and no disturbances, $y(k+1) = 2.2864$. Hence using $\hat{y}(k+2|k+1) = 0.7y(k+1) + 2u(k+1)$, the free response is given by assuming that $u(k+1) = u(k+2) = 0.4432$:

$$\hat{y}_f(k+2|k+1) = 0.7 \times 2.2864 + 2 \times 0.4432 = 2.4869 \quad (1)$$

$$\hat{y}_f(k+3|k+1) = 0.7 \times 2.4869 + 2 \times 0.4432 = 2.6272 \quad (2)$$

We have $\epsilon(k+1) = s(k+1) - y(k+1) = 3 - 2.2864 = 0.7136$, hence

$$r(k+3|k+1) = s(k+3) - \lambda^2\epsilon(k+1) = 3 - 0.7165^2 \times 0.7136 = 2.6337 \quad (3)$$

Hence

$$\Delta\hat{u}(k+1|k+1) = \frac{2.6337 - 2.6272}{3.4} = 0.0019 \quad (4)$$

$$u(k+1) = \hat{u}(k+1|k+1) = u(k) + \Delta\hat{u}(k+1|k+1) \quad (5)$$

$$= 0.4432 + 0.0019 = 0.4451 \quad (6)$$

That is one step done.

Now the second step:

$$y(k+2) = 0.7 \times 2.2864 + 2 \times 0.4451 = 2.4907 \quad (7)$$

$$\hat{y}_f(k+3|k+2) = 0.7 \times 2.4907 + 2 \times 0.4451 = 2.6337 \quad (8)$$

$$\hat{y}_f(k+4|k+2) = 0.7 \times 2.6337 + 2 \times 0.4451 = 2.7338 \quad (9)$$

$$\epsilon(k+2) = s(k+2) - y(k+2) = 3 - 2.4907 = 0.5093 \quad (10)$$

$$r(k+4|k+2) = 3 - 0.7165^2 \times 0.5093 = 2.7385 \quad (11)$$

$$\Delta\hat{u}(k+2|k+2) = \frac{2.7385 - 2.7338}{3.4} = 0.0014 \quad (12)$$

$$u(k+2) = 0.4451 + 0.0014 = 0.4469 \quad (13)$$

Verification: $y(k+2) = 2.4907 \neq 2.2864 = 0.7 \times 2.0 + 2 \times 0.4432 = \hat{y}(k+2|k)$.

1.2 Here is the solution as a *MATLAB* file:

```
% Solution to Exercise 1.2:
setpoint = 3;
Tref = 9;  % Reference trajectory time constant
Ts = 3;    % sample time
lambda = exp(-Ts/Tref);
P = [1;2]; % coincidence points

% Initial conditions:
yk = 2;      % output y(k), from Example 1.3 (note y(k)=y(k-1)).
uk = 0.4429; % last applied input signal u(k) (from Example 1.4)
ykplus1 = 0.7*yk + 2*uk; % output y(k+1)

% Step response vector [S(P(1));S(P(2))] as in (1.21):
S = [2.0 ; 3.4]; % from Example 1.4

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Next step:
% Reference trajectory:
error = setpoint-ykplus1;
% Form reference (or target) vector [r(k+2|k+1) ; r(k+3|k+1)]:
T = [setpoint;setpoint]-[lambda^P(1);lambda^P(2)]*error;

% Free response vector:
yf1 = 0.7*ykplus1 + 2*uk;  % yf(k+2|k+1)
yf2 = 0.7*yf1 + 2*uk;      % yf(k+3|k+1)
Yf = [yf1 ; yf2]; % vector of free responses - as in (1.21)

% New optimal control signal:
Deltau = S\(T-Yf);  % as in (1.22)
ukplus1 = uk + Deltau(1) % use first element only of Deltau

% Resulting output:
ykplus2 = 0.7*ykplus1 + 2*ukplus1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% One more step:
% Reference trajectory:
error = setpoint-ykplus2;
% Form reference (or target) vector [r(k+3|k+2) ; r(k+4|k+2)]:
T = [setpoint;setpoint]-[lambda^P(1);lambda^P(2)]*error;

% Free response vector:
yf1 = 0.7*ykplus2 + 2*ukplus1;  % yf(k+3|k+2)
yf2 = 0.7*yf1 + 2*ukplus1;      % yf(k+4|k+2)
```

```
Yf = [yf1 ; yf2]; % vector of free responses - as in (1.21)

% New optimal control signal:
Deltau = S\(T-Yf);  % as in (1.22)
ukplus2 = ukplus1 + Deltau(1) % use first element only of Deltau

% Resulting output:
ykplus3 = 0.7*ykplus2 + 2*ukplus2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

This gives the following results:

$$u(k+1) = 0.4448 \qquad y(k+2) = 2.4897$$
$$u(k+2) = 0.4463 \qquad y(k+3) = 2.6354$$

In this *MATLAB* program the computations for each step have been set out separately. In practice one would set them out once inside a repeating loop — see program `basicmpc`.

1.3 **Repeat of Example 1.3:**
As in Example 1.3 we have $\epsilon(k) = 1$. Hence

$$\epsilon(k+2) = \left(1 - \frac{2T_s}{T_{ref}}\right)\epsilon(k) = \frac{1}{3} \tag{14}$$

Hence $r(k+2|k) = 3 - \frac{1}{3} = 2.6667$. So proceeding as in Example 1.3,

$$\Delta\hat{u}(k|k) = \frac{2.6667 - 2.0}{3.4} = 0.1961 \tag{15}$$
$$\hat{u}(k|k) = u(k-1) + \Delta\hat{u}(k|k) = 0.4961 \tag{16}$$

This should result in the next plant output value being $y(k+1) = 0.7 \times 2 + 2 \times 0.4961 = 2.3922$.

Repeat of Example 1.4:
The vector of reference trajectory values (or 'target' vector) at the coincidence points is now

$$\mathcal{T} = \begin{bmatrix} 3 - \frac{2}{3} \\ 3 - \frac{1}{3} \end{bmatrix} = \begin{bmatrix} 2.3333 \\ 2.6667 \end{bmatrix} \tag{17}$$

while the vectors $\mathcal{Y}_f$ and $\mathcal{S}$ remain unchanged. Hence

$$\Delta\hat{u}(k|k) = \mathcal{S}\backslash(\mathcal{T} - \mathcal{Y}_f) = 0.1885 \tag{18}$$
$$\hat{u}(k|k) = u(k-1) + \Delta\hat{u}(k|k) = 0.4885 \tag{19}$$

This should result in the next plant output value being $y(k+1) = 0.7 \times 2 + 2 \times 0.4885 = 2.3770$.

1.4 *There is an error in equation (1.23) in the book.* Since the free response $\hat{y}_f(k+P_i|k)$ is defined to be the response when the input remains at its last value, namely $u(k-1)$, each term in (1.23) involving an input value $\hat{u}(k+j|k)$ should in fact involve the difference $\hat{u}(k+j|k)-u(k-1)$. Thus the correct expression for (1.23) is:

$$\begin{aligned}\hat{y}(k+P_i|k) = \hat{y}_f(k+P_i|k) + H(P_i)[\hat{u}(k|k) - u(k-1)] + \\ H(P_i-1)[\hat{u}(k+1|k) - u(k-1)] + \cdots \\ H(P_i-H_u+2)[\hat{u}(k+H_u-2|k) - u(k-1)] + \\ S(P_i-H_u+1)[\hat{u}(k+H_u-1|k) - u(k-1)] \end{aligned} \quad (20)$$

This can be written as

$$\begin{aligned}\hat{y}(k+P_i|k) = \hat{y}_f(k+P_i|k) \ + [S(P_i) - S(P_i-1)][\hat{u}(k|k) - u(k-1)] + \\ [S(P_i-1) - S(P_i-2)]\hat{u}(k+1|k) + \cdots \\ [S(P_i-H_u+2) - S(P_i-H_u+1)]\hat{u}(k+H_u-2|k) + S(P_i-H_u+1)\hat{u}(k+H_u-1|k)\end{aligned} \quad (21)$$

The terms can be regrouped as:

$$\begin{aligned}\hat{y}(k+P_i|k) &= \hat{y}_f(k+P_i|k) + S(P_i)[\hat{u}(k|k) - u(k-1)] + \\ &\quad S(P_i-1)[\hat{u}(k+1|k) - \hat{u}(k|k)] + \cdots \\ &\quad S(P_i-H_u+1)[\hat{u}(k+H_u-1|k) - \hat{u}(k+H_u-2|k)] \\ &= \hat{y}_f(k+P_i|k) + S(P_i)\Delta\hat{u}(k|k) + \\ &\quad S(P_i-1)\Delta\hat{u}(k+1|k) + \cdots + \\ &\quad S(P_i-H_u+1)\Delta\hat{u}(k+H_u-1|k)\end{aligned} \quad (22)$$

which verifies (1.24).

1.5 The reference trajectory values at the two coincidence points are the same whichever model is used, so we calculate these first. Let t denote the current time. The initial error is $\epsilon(t) = 3 - 1 = 2$, so the reference trajectory 6 sec ahead is $r(t+6|t) = 3 - \exp(-6/5) \times 2 = 2.3976$, and 10 sec ahead is $r(t+10|t) = 3 - \exp(-10/5) \times 2 = 2.7293$. Thus the target vector needed in (1.22) is

$$\mathcal{T} = \begin{bmatrix} 2.3976 \\ 2.7293 \end{bmatrix} \quad (23)$$

Since the output is constant at 1, the input must be constant. The steady-state gain of the model is 2, so this constant value of the input must be 0.5. The free response, with $u(t+\tau) = 0.5$ for $\tau > 0$, is $y_f(t+\tau) = 1$, since the output would remain at its equilibrium value if the input value did not change.

(a). The only information needed from the continuous-time model is the step response values at the coincidence points. The transfer function corresponds to the differential equation

$$y(t) + 7\dot{y}(t) = 2u(t - 1.5) \tag{24}$$

so the step response is the solution to

$$y_f(t) + 7\dot{y}_f(t) = \begin{cases} 0 & (t < 1.5) \\ 2 & (t \geq 1.5) \end{cases} \tag{25}$$

with initial condition $y(0) = 0$, which is $y(t) = 2(1 - e^{-(t-1.5)/7})$ for $t \geq 1.5$ (from the elementary theory of differential equations). Hence the vector of step responses at the coincidence points is

$$\mathcal{S} = \begin{bmatrix} 0.9484 \\ 1.4062 \end{bmatrix} \tag{26}$$

(This can also be obtained by using the function `step` in *MATLAB*'s *Control System Toolbox*.) So now we can apply (1.22) to get

$$\Delta u(\hat{k}|k) = \mathcal{S}\backslash(\mathcal{T} - \mathcal{Y}_f) = 1.3060 \tag{27}$$

so the optimal input is

$$u(k) = 0.5 + 1.3060 = 1.8060 \tag{28}$$

(b). An equivalent discrete-time model is obtained most easily using *MATLAB*'s *Control System Toolbox* function `c2d` on the original transfer function without the delay:

```
sysc=tf(2,[7,1])
sysd=c2d(sysc,0.5)
```

which gives the z-transform transfer function $0.1379/(z - 0.9311)$. Now the delay of 1.5 sec, namely three sample periods, can be incorporated by multiplying by z^{-3}: $0.1379/z^3(z - 0.9311)$. Now using the function `step` gives the step response at the coincidence points as 0.9487 and 1.4068, respectively. Proceeding as in (a), we get $\Delta u(\hat{k}|k) = 1.3055$ and hence $u(k) = 1.8055$.

The two points made by this exercise are: (1) Continuous-time models can be used directly. (2) Whether a continuous-time or a discrete-time model is used makes little difference, if the sampling interval is sufficiently small.

1.6 Since the plant has a delay of 1.5 sec, the predicted output is not affected by the input within that time. So choosing a coincidence point nearer than 1.5 sec into the future would have no effect on the solution.

1.7 With only one coincidence point (assumed to be P steps ahead) and $H_u = 1$ we have $\mathcal{T} = r(k+P|k)$, $\mathcal{Y}_f = y_f(k+P|k)$, and $\Theta = S(P)$, so (1.31) becomes

$$\Delta\hat{u}(k|k) = \frac{r(k+P|k) - d(k) - y_f(k+P|k)}{S(P)} \tag{29}$$

In steady state (1.33) and (1.34) become

$$\Delta\hat{u}(k|k) = \frac{r_P - d_\infty - y_{fP}}{S(P)} \tag{30}$$

$$= \frac{r_P - y_{p\infty} + y_{m\infty} - y_{fP}}{S(P)} \tag{31}$$

(1.35) becomes $y_{m\infty} - y_{fP} = 0$, and (1.36) becomes

$$\Delta\hat{u}(k|k) = \frac{r_P - y_{p\infty}}{S(P)} \tag{32}$$

But in the steady state $\Delta\hat{u}(k|k) = 0$ and hence $r_P = y_{p\infty}$. But $r_P = s_\infty - \lambda^P(s_\infty - y_{p\infty}) = s_\infty - \lambda^P(s_\infty - r_P)$. Hence $y_{p\infty} = r_P = s_\infty$.

1.8 To do this exercise, simply change files `basicmpc.m` and `trackmpc.m` as follows. First define the plant by replacing the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW PLANT %%%%%
nump=1;
denp=[1,-1.4,0.45];
plant=tf(nump,denp,Ts);
%%%%% CHANGE UP TO HERE TO DEFINE NEW PLANT %%%%%
```

by the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW PLANT %%%%%
nump=1;
denp=[1,-1.5,0.5];   % (z-0.5)(z-1)
plant=tf(nump,denp,Ts);
%%%%% CHANGE UP TO HERE TO DEFINE NEW PLANT %%%%%
```

then define the model by replacing the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW MODEL %%%%%
model = plant;
%%%%% CHANGE UP TO HERE TO DEFINE NEW MODEL %%%%%
```

by the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW MODEL %%%%%
numm=1.1;
denm=[1,-1.5,0.5];
model = tf(numm,denm,Ts);
%%%%% CHANGE UP TO HERE TO DEFINE NEW MODEL %%%%%
```

1.9 With the independent model implementation, the measured plant output does not directly influence the model output. When $T_{ref} = 0$ the reference trajectory does not depend on the measured plant output, because it is equal to the future set-point trajectory. (Whereas with $T_{ref} \neq 0$ it does depend on the measured output.) Thus for each coincidence point the controller is concerned only to move the output from the present model output value $y_m(k)$ to the required value $s(k + P_i)$. Neither of these values, nor the free response of the model, is affected by the measurement noise. Hence the control signal is not affected by the noise, either.

When offset-free tracking is provided, the disturbance estimate $d(k) = y_p(k) - \hat{y}(k|k-1)$ is affected by the noise, through the measurement $y_p(k)$, and this directly affects the control signal.

Comment: This shows how essential it is to tie the model output to the plant output in some way. Otherwise the model and plant could drift arbitrarily far apart. The way it is done in Section 1.5 is a very easy way of doing it, but not the only possible one.

1.10 This exercise can be solved just by editing the files `basicmpc.m` and `trackmpc.m`, to define the variables `plant`, `model`, `Tref`, `Ts`, `P` and `M` appropriately.

1.11 Just edit the file `unstampc.m` in the obvious way — change the definition of variable `model`.

1.12 The text following equation (1.31) shows what has to be done: in file `unstampc.m` replace the lines:

```
% Compute input signal uu(k):
  if k>1,
    dutraj = theta\(reftraj-ymfree(P)');
    uu(k) = dutraj(1) + uu(k-1);
  else
    dutraj = theta\(reftraj-ymfree(P)');
    uu(k) = dutraj(1) + umpast(1);
  end
```

by:

```
% Compute input signal uu(k):
  d = yp(k) - ym(k);
  if k>1,
    dutraj = theta\(reftraj-d-ymfree(P)');
    uu(k) = dutraj(1) + uu(k-1);
  else
    dutraj = theta\(reftraj-d-ymfree(P)');
    uu(k) = dutraj(1) + umpast(1);
  end
```

1.13 In the existing file `unstampc.m` the restriction that the plant and model should have the same order arises in the lines

```
% Simulate model:
  % Update past model inputs:
  umpast = [uu(k);umpast(1:length(umpast)-1)];
  % Simulation:
  ym(k+1) = -denm(2:ndenm+1)*ympast+numm(2:nnumm+1)*umpast;
  % Update past model outputs:
  ympast = yppast; %%% RE-ALIGNED MODEL
```

In particular the product `denm(2:ndenm+1)*ympast` assumes that the length of the vector `denm(2:ndenm+1)`, which depends on the model order, is the same as the length of the vector `yppast` (since `ympast` is set equal to `yppast`), which depends on the order of the plant.

To handle different orders of plant and model it is necessary to store the appropriate number of past values of the plant output in the vector `ympast`. If the model order is smaller than the plant order this is very easy: just truncate the vector `yppast` (since the most recent plant output values are stored in the initial locations of `yppast`):

```
% Simulate model:
  % Update past model inputs:
  umpast = [uu(k);umpast(1:length(umpast)-1)];
  % Simulation:
  ym(k+1) = -denm(2:ndenm+1)*ympast+numm(2:nnumm+1)*umpast;
  % Update past model outputs:
  if ndenm <= ndenp,          %%% MODEL ORDER <= PLANT ORDER
    ympast = yppast(1:ndenm); %%% RE-ALIGNED MODEL
  end
```

If the model order is larger than the plant order then there are not enough past values in `yppast`, as currently defined. One solution is to leave `yppast`

unchanged, but to hold the additional past values in `ympast`, by 'shuffling' them up to the tail of the vector:

```
if ndenm > ndenp,            %%% MODEL ORDER > PLANT ORDER
  ympast(ndenp+1:ndenm) = ympast(ndenp:ndenm-1);
  ympast(1:ndenp) = yppast;
end
```

1.14 The vector $\Delta\mathcal{U}$ should contain only the changes of the control signal:

$$\Delta\mathcal{U} = \begin{bmatrix} \Delta\hat{u}(k+M_1|k) \\ \Delta\hat{u}(k+M_2|k) \\ \vdots \\ \Delta\hat{u}(k+M_{H_u}|k) \end{bmatrix} \tag{33}$$

The matrix Θ should contain the columns corresponding to these elements of $\Delta\mathcal{U}$, so that (1.27) is replaced by:

$$\Theta = \begin{bmatrix} S(P_1-M_1) & S(P_1-M_2) & \cdots & S(P_1-M_{H_u}) \\ S(P_2-M_1) & S(P_2-M_2) & \cdots & S(P_2-M_{H_u}) \\ \vdots & \vdots & \vdots & \vdots \\ S(P_c-M_1) & S(P_c-M_2) & \cdots & S(P_c-M_{H_u}) \end{bmatrix} \tag{34}$$

(remember that $S(i) = 0$ if $i < 0$). Note that the vector $\mathcal{Y}$ remains the same as in (1.26).

1.15 Exercise 1.15 is done by editing the files `basicmpc`, `trackmpc` and `noisympc`. The main point to be made here is that the effects of changing various parameters are difficult to predict precisely, and can occasionally be counter-intuitive. This provides some motivation for Chapter 7, which discusses tuning MPC controllers.

Obvious things to check are:

(a). Increasing T_{ref} slows down the response.

(b). Increasing the prediction horizon (the location of the last coincidence point) improves closed-loop stability (because the controller pays more attention to long-term consequences of its actions).

(c). Increasing H_u increases the computation time. (Use *MATLAB* functions `tic` and `toc` to check this.)

Chapter 2

A basic formulation of predictive control

2.1 The following *MATLAB* code does the job:

```
Q = [10 4 ; 4 3]
eig(Q)
```

This shows the eigenvalues are 11.815 and 1.185, which are both positive.

2.2 (a).

$$V(x) = 9x_1^2 + 25x_2^2 + 16x_3^2 = [x_1, x_2, x_3] \begin{bmatrix} 9 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (1)$$

so $Q = \text{diag}[9, 25, 16]$.

(b). With $x^T = [x_1, x_2, x_3]$ and $u^T = [u_1, u_2]$ we get $V(x, u) = x^T Q x + u^T R u$ if

$$Q = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} 100 & 0 \\ 0 & 4 \end{bmatrix} \quad (2)$$

(c). Yes. If $x_i \neq 0$ for any i then both $V(x)$ in (a) and $V(x, u)$ in (b) are positive, by inspection. Similarly, if $u_i \neq 0$ for any i then $V(x, u)$ in (b) is positive. If $x = 0$ then $V(x) = 0$. If $x = 0$ and $u = 0$ then $V(x, u) = 0$. So both functions V are positive-definite.

2.3 *Error in question:* The formula for the gradient is not (2.19), but the formula for ∇V which appears at the end of Mini-Tutorial 1.

If $V(x) = x^T Q x = 10x_1^2 + 8x_1x_2 + 3x_2^2$, as in Example 2.1, then

$$\frac{\partial V}{\partial x_1} = 20x_1 + 8x_2 \qquad \text{and} \qquad \frac{\partial V}{\partial x_2} = 8x_1 + 6x_2 \tag{3}$$

so $\nabla V = [20x_1 + 8x_2, 8x_1 + 6x_2]$ from the definition. Applying the formula we get

$$\nabla V = 2x^T Q = 2[x_1, x_2] \begin{bmatrix} 10 & 4 \\ 4 & 3 \end{bmatrix} = 2[10x_1 + 4x_2, 4x_1 + 3x_2] \tag{4}$$

so the two agree.

2.4 *Error in question:* The question should refer to Example 2.3, not Example 2.4. (But the reader could answer the question for Example 2.4 too — see below.)

Solution for Example 2.3:
In Example 2.3 the constraints on the inputs are $0 \le u_2 \le 3$. The control horizon is $H_u = 1$, so this translates into the constraint $0 \le \hat{u}_2(k|k) \le 3$, which can be written as

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -3 \end{bmatrix} \begin{bmatrix} \hat{u}_1(k|k) \\ \hat{u}_2(k|k) \\ 1 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5}$$

so that

$$F = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -3 \end{bmatrix} \tag{6}$$

Here is the corresponding solution for Example 2.4:
In Example 2.4 the constraints on the inputs are $-10 \le u_j(k) \le 10$ for $j = 1, 2$. Since these must be respected across the whole control horizon, we have $-10 \le \hat{u}_j(k+i|k) \le 10$ for $i = 0, 1, 2, 3$ (since $H_u = 3$) and $j = 1, 2$. For each i and j this constraint can be expressed as

$$\begin{bmatrix} -1 & -10 \\ 1 & -10 \end{bmatrix} \begin{bmatrix} \hat{u}_j(k+i|k) \\ 1 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{7}$$

So all the constraints can be represented using a 16×9 matrix F (not 4×7, as written in the book):

$$\begin{bmatrix} -1 & 0 & \cdots & 0 & -10 \\ 1 & 0 & \cdots & 0 & -10 \\ 0 & -1 & \cdots & 0 & -10 \\ 0 & 1 & \cdots & 0 & -10 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & -10 \\ 0 & 0 & \cdots & 1 & -10 \end{bmatrix} \begin{bmatrix} \hat{u}_1(k|k) \\ \hat{u}_2(k|k) \\ \hat{u}_1(k+1|k) \\ \hat{u}_2(k+1|k) \\ \hat{u}_1(k+2|k) \\ \hat{u}_2(k+2|k) \\ \hat{u}_1(k+3|k) \\ \hat{u}_2(k+3|k) \\ 1 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

2.5 The choice of $\Delta u(k)$ will affect $x(k+1)$, and hence $z(k+1)$. It will not affect $z(k)$, since the model has no direct 'feed-through' from $u(k)$ to $z(k)$. We have measurements up to time k, so our best estimate of $z(k+1)$ is

$$\begin{aligned} \hat{z}(k+1|k) &= 3\hat{x}(k+1|k) = 3[2\hat{x}(k|k) + u(k)] && (9)\\ &= 3[2\hat{x}(k|k) + u(k-1) + \Delta u(k)] && (10)\\ &= 3[2\times 3 - 1 + \Delta u(k)] = 15 + 3\Delta u(k)] && (11) \end{aligned}$$

But we must respect the constraint $-1 \le \hat{z}(k+1|k) \le 2$, so $-1 \le 15+3\Delta u(k) \le 2$, hence

$$-\frac{16}{3} \le \Delta u(k) \le -\frac{13}{3} \qquad (12)$$

2.6 Checking the discretization using *MATLAB*, assuming the continuous-time system matrices `Ac,Bc,Cy` have already been defined to correspond to A_c, B_c, C_y in Example 2.4:

```
D = zeros(3,2);  % Direct 'feed-through' matrix is 0
cont_sys=ss(Ac,Bc,Cy,D);    % Continuous-time system
Ts = 2;                          % Sampling interval
disc_sys=c2d(cont_sys,Ts);    % Discrete-time system
A = get(disc_sys,'a'); % Should agree with A in Example 2.4
B = get(disc_sys,'b'); % Should agree with B in Example 2.4
```

Checking stability:

```
eig(A)           % Eigenvalues of A
```

This shows that the eigenvalues of the discrete-time system are 0.4266, 0.4266, 0.2837 and 0.0211. These all lie within the unit circle, so the dicrete-time system is stable.

To check that each of these eigenvalues corresponds to $\exp(\lambda T_s)$, where λ is an eigenvalue of A_c:

```
exp(eig(Ac)*Ts)
```

2.7 (a). Since the feed-tank level H_1 is to be constrained, it becomes a controlled output — even though it is not to be controlled to a set-point. So C_z must have another row, to represent this. Assuming that H_1 will be the first element in the vector of controlled outputs, C_z becomes:

$$C_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (13)$$

(b). Note that the dimensions given for the matrices E and F in Example 2.4 are wrong. They should be 16×9 (not 4×7, as stated). The dimensions given for G are correct. Since E and F are concerned with constraints on the input amplitudes and input increments, they are not affected by adding constraints on H_1, which is an output.

There will be two constraints on H_1 for each step in the prediction horizon — one minimum value and one maximum value. So two rows need to be added to G for each step. The prediction horizon is $H_p = 10$, so G will need 20 additional rows. That is, the new dimensions of G will be 40×21.

(c). The additional difficulty is that although H_1 is to be controlled within constraints, it is not measured. So its value has to be inferred from the measured variables somehow. If the system behaviour is sufficiently linear then the standard way of doing this is to use a state observer (see Mini-Tutorial 2). But if this does not give sufficiently accurate estimates of H_1 then some nonlinear observer may have to be used, or (preferably) a sensor to measure H_1 should be installed, if feasible.

2.8 Equation (2.40) states that $\Delta x(k+1) = A\Delta x(k) + B\Delta u(k)$. This gives the first rows of the matrices in (2.108). From (2.2) we have

$$\begin{aligned} y(k) &= C_y x(k) && (14)\\ &= C_y[\Delta x(k) + x(k-1)] && (15)\\ &= C_y \Delta x(k) + y(k-1) && (16)\\ &= [C_y, I, 0]\xi(k) && (17) \end{aligned}$$

which gives the second row of (2.108) and the first row of (2.109). The third row of (2.108) and the second row of (2.109) follow similarly from (2.3).

2.9 We have $B_1 = e^{A_c(T_s-\tau)} \int_0^\tau e^{A_c\eta} d\eta B_c$ and $B_2 = \int_0^{T_s-\tau} e^{A_c\eta} d\eta B_c$.

Now the function `c2d` computes the discretised system with the assumption that $\tau = 0$, so the call `[Ad,Bd]=c2d(Ac,Bc,Ts)` gives $A_d = e^{A_c T_s}$ and $B_d = \int_0^{T_s} e^{A_c\eta} d\eta B_c$. Thus the required realization can be obtained using the following function `c2dd`:

```
function [Ad,Bd,Cd] = c2dd(Ac,Bc,Cc,Ts,tau)
%C2DD Computes discrete-time model with computational delay
%
% Usage:  [Ad,Bd,Cd] = c2dd(Ac,Bc,Cc,Ts,tau)
%
% Computes discrete-time state space realisation for system with
% computation delay.
% Inputs: Ac,Bc,Cc: 'A','B','C' matrices of continuous-time model
%                Ts: Sampling interval
```

```
%              tau: Computational delay
% Outputs: Ad,Bd,Bd: 'A','B','C' matrices of discrete-time model
%
% 'D' matrix assumed to be zero.

% Written by J.M.Maciejowski, 25.9.98.
% Reference: 'Predictive control with constraints', section
%            on 'Computational delays'.

[nstates,ninputs] = size(Bc);
noutputs = size(Cc,1);

if Ts < tau,
  error('tau greater than Ts')
end

[Ad,dummy] = c2d(Ac,Bc,Ts);  % (1,1) block of Ad computed

[A1,B1] = c2d(Ac,Bc,tau);
[A2,B2] = c2d(Ac,Bc,Ts-tau); % B2 computed
B1 = A2*B1;                  % B1 computed

Ad = [Ad, B1; zeros(ninputs,nstates+ninputs)];
Bd = [B2; eye(ninputs)];
Cd = [Cc,zeros(noutputs,ninputs)];
```

2.10 Using the function `c2d` for the case $T_s = 0.1, \tau = 0$ gives

$$
\begin{aligned}
A &= \begin{bmatrix} 0.85679 & 0 & 0.08312 & 0 \\ -0.02438 & 1 & 0.09057 & 0 \\ -0.46049 & 0 & 0.80976 & 0 \\ -12.03761 & 12.8200 & 0.03680 & 1 \end{bmatrix} \\
B &= \begin{bmatrix} -0.10272 \\ -0.07944 \\ -1.53242 \\ 0.16674 \end{bmatrix} \\
C &= \begin{bmatrix} -128.2 & 128.2 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

Using the function `c2dd` written for the previous problem for the case $T_s =$

$0.1, \tau = 0.02$ gives

$$A = \begin{bmatrix} 0.85679 & 0 & 0.08312 & 0 & -0.03109 \\ -0.02438 & 1 & 0.09057 & 0 & -0.02788 \\ -0.46049 & 0 & 0.80976 & 0 & -0.27938 \\ -12.03761 & 12.82000 & 0.03680 & 1 & 0.05578 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} -0.07163 \\ -0.05156 \\ -1.25304 \\ 0.11096 \\ 1 \end{bmatrix}$$

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

As is to be expected, the step (or impulse) responses of the model with the computational delay lag behind the other ones by 0.05 sec.

The frequency responses are almost identical up to a frequency of about 1 rad/sec. At about 6 rad/sec (about 1 Hz) there are phase differences of about 10° in each channel. This means that if the closed-loop bandwidth is about 1 rad/sec, or lower, the computational delay can be neglected. If it is above 1 rad/sec then it should be taken into account. But note that the Nyquist frequency is $\pi/T_s = 10\pi \approx 30$ rad/sec, so one is unlikely to try to obtain a bandwidth as high as 5 rad/sec, say, with this value of T_s.

2.11

$$\begin{bmatrix} \hat{z}(k+1|k) \\ \vdots \\ \hat{z}(k+H_u|k) \\ \hat{z}(k+H_u+1|k) \\ \vdots \hat{z}(k+H_p|k) \end{bmatrix} = \begin{bmatrix} C_zA \\ \vdots \\ C_zA^{H_u} \\ C_zA^{H_u+1} \\ \vdots \\ C_zA^{H_p} \end{bmatrix} x(k) + \begin{bmatrix} C_zB \\ \vdots \\ C_z\sum_{i=0}^{H_u-1} A^iB \\ C_z\sum_{i=0}^{H_u} A^iB \\ \vdots \\ C_z\sum_{i=0}^{H_p-1} A^iB \end{bmatrix} u(k-1) +$$

$$\begin{bmatrix} C_zB & \cdots & 0 \\ C_z(AB+B) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ C_z\sum_{i=0}^{H_u-1} A^iB & \cdots & C_zB \\ C_z\sum_{i=0}^{H_u} A^iB & \cdots & C_z(AB+B) \\ \vdots & \ddots & \vdots \\ C_z\sum_{i=0}^{H_p-1} A^iB & \cdots & C_z\sum_{i=0}^{H_p-H_u} A^iB \end{bmatrix} \begin{bmatrix} \Delta\hat{u}(k|k) \\ \vdots \\ \Delta\hat{u}(k+H_u-1|k) \end{bmatrix} \quad (18)$$

2.12 *Note: The solution to this exercise may be clearer after reading Sections 3.1 and 3.2.*

From (2.19) the cost function is

$$V(k) = \sum_{i=H_w}^{H_p} \|\hat{\tilde{z}}(k+i|k) - D_z\hat{u}(k+i|k) - r(k+i|k)\|^2_{Q(i)} + \sum_{i=0}^{H_u-1} \|\Delta\hat{u}(k+i|k)\|^2_{R(i)} \tag{19}$$

But for vectors a and b we have

$$\begin{aligned} \|a-b\|^2_M &= (a-b)^T M(a-b) && (20)\\ &= a^T Ma - 2a^T Mb + b^T Mb && (21)\\ &= \|a\|^2_M + \|b\|^2_M - 2a^T Mb && (22) \end{aligned}$$

Hence, taking $a = \hat{\tilde{z}}(k+i|k) - r(k+i|k)$ and $b = D_z\hat{u}(k+i|k)$, the first term in $V(k)$ is

$$\begin{aligned} \sum_{i=H_w}^{H_p} \Big\{ &\|\hat{\tilde{z}}(k+i|k) - r(k+i|k)\|^2_{Q(i)} + \|D_z\hat{u}(k+i|k)\|^2_{R(i)} - \\ &2\hat{\tilde{z}}(k+i|k)^T Q(i) D_z\hat{u}(k+i|k) + 2r(k+i|k)^T Q(i) D_z\hat{u}(k+i|k) \Big\} \end{aligned} \tag{23}$$

Since $\tilde{z}(k) = z(k) - D_z u(k) = C_z x(k)$ we have $\hat{\tilde{z}}(k+i|k) = C_z\hat{x}(k+i|k)$, which is linear in $\Delta\hat{u}$ if a linear observer is used, so the first and third terms in this expression are quadratic in $\Delta\hat{u}$ (since $\hat{u}(k+i|k) = u(k-1) + \sum_i \Delta\hat{u}(k+i|k)$ is also linear in $\Delta\hat{u}$). The second term is clearly quadratic in $\Delta\hat{u}$, and the fourth term is linear in $\Delta\hat{u}$. Hence the whole expression is quadratic in $\Delta\hat{u}$.

Since $\hat{\tilde{z}}$ is linear in both $\hat{z}$ and $\hat{u}$ (and hence in $\Delta\hat{u}$), and $\hat{z}$ is itself linear in $\Delta\hat{u}$, linear constraints on $\hat{z}$ can be expressesd as linear constraints on $\hat{\tilde{z}}$ and $\Delta\hat{u}$, and hence on the $\Delta\hat{u}$ variables only.

2.13 (2.66) still holds when $D_z \neq 0$. But (2.67)–(2.69) have to be changed, since now

$$\begin{aligned} \hat{z}(k+i|k) &= C_z\hat{x}(k+i|k) + D(z)\hat{u}(k+i|k) && (24)\\ &= C_z\hat{x}(k+i|k) + D(z)[u(k-1) + \sum_{j=0}^{i} \Delta\hat{u}(k+j|k)] && (25) \end{aligned}$$

So (2.70) should be replaced by

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

$$\begin{bmatrix} D_z \\ \vdots \\ D_z \end{bmatrix} u(k-1) + \begin{bmatrix} D_z & D_z & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_z & D_z & D_z & \cdots & D_z \end{bmatrix} \begin{bmatrix} \Delta\hat{u}(k|k) \\ \Delta\hat{u}(k+1|k) \\ \Delta\hat{u}(k+2|k) \\ \vdots \\ \Delta\hat{u}(k+H_p|k) \end{bmatrix} \tag{26}$$

Note, however, that if $D_z \neq 0$ then one should consider including $\hat{z}(k|k)$ in the cost function, since it is influenced by $\Delta\hat{u}(k)$ (unlike in the case $D_z = 0$).

2.14 Applying the standard observability test, the pair

$$\left(\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}, \begin{bmatrix} C_y & I \end{bmatrix} \right)$$

is observable if and only if the matrix

$$\begin{bmatrix} \begin{bmatrix} C_y & I \end{bmatrix} \\ \begin{bmatrix} C_y & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \\ \begin{bmatrix} C_y & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}^2 \\ \vdots \end{bmatrix} = \begin{bmatrix} C_y & I \\ C_yA & I \\ C_yA^2 & I \\ \vdots & \vdots \end{bmatrix} \tag{27}$$

has full column rank. It is easy to see that this is the case if and only if the matrix

$$\begin{bmatrix} C_y \\ C_yA \\ C_yA^2 \\ \vdots \end{bmatrix}$$

has full column rank, which is precisely the condition that the pair (A, C_y) should be observable.

2.15 *Note: This exercise requires the use of the* Model Predictive Control Toolbox *in parts (c)–(e). It may be better done after Exercise 3.1. Even then it will require some reading of the* Model Predictive Control Toolbox *documentation. The advice given in part (b) of the question is seriously misleading; it is easier to do it by hand than to use the* Model Predictive Control Toolbox.

(a). The model becomes

$$\begin{aligned} x(k+1) &= 0.9x(k) + 0.5[u(k) + d(k)] && (28)\\ d(k+1) &= d(k) && (29)\\ y(k) &= x(k) && (30) \end{aligned}$$

which can be written in state-space form as

$$\begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} = \begin{bmatrix} 0.9 & 0.5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} u(k) \qquad (31)$$

$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} \qquad (32)$$

(b). Using this model in the observer equation (2.81), with the same notation as used in (2.84), gives the observer's state transition matrix as

$$A - LC = \begin{bmatrix} 0.9 & 0.5 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} L_x \\ L_d \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.9 - L_x & 0.5 \\ -L_d & 1 \end{bmatrix} \qquad (33)$$

For a deadbeat observer both eigenvalues of this matrix should be zero, so both its determinant and its trace should be zero. Hence we need

$$0.9 - L_x + 0.5L_d = 0 \qquad \text{and} \qquad 1.9 - L_x = 0 \qquad (34)$$

from which $L_x = 1.9$ and $L_d = 2$.

(c). To do this using the *Model Predictive Control Toolbox* is complicated by the fact that the *Model Predictive Control Toolbox* uses the augmented state vector (2.37) — without distinguishing between y and z. See Appendix C and the *Model Predictive Control Toolbox* documentation.
From (2.43) the 'A' matrix of the plant with the state vector as in (2.37) is

$$\tilde{A} = \begin{bmatrix} \begin{bmatrix} 0.9 & 0.5 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0.9 & 0.5 \end{bmatrix} & 1 \end{bmatrix} \qquad (35)$$

and the corresponding 'C' matrix is $\tilde{C} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. The observer gain matrix 'L' now has 3 elements and the closed-loop state transition matrix of the observer '$A - LC$' is

$$\begin{aligned} \tilde{A} - \tilde{L}\tilde{C} &= \begin{bmatrix} \begin{bmatrix} 0.9 & 0.5 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0.9 & 0.5 \end{bmatrix} & 1 \end{bmatrix} - \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} && (36)\\ &= \begin{bmatrix} 0.9 & 0.5 & -L_1 \\ 0 & 1 & -L_2 \\ 0.9 & 0.5 & 1 - L_3 \end{bmatrix} && (37) \end{aligned}$$

The characteristic polynomial of this is

$$\det\left[\lambda I - (\tilde{A} - \tilde{L}\tilde{C})\right] =$$
$$\lambda^3 + (L_3 - 2.9)\lambda^2 + (2.8 + 0.9L_1 + 0.5L_2 - 1.9L_3)\lambda - 0.9(L_1 - L_3 + 1) \qquad (38)$$

For a deadbeat observer all the roots of this should be at zero, so this characteristic polynomial should be just λ^3; hence setting the coefficients of λ^2, λ and λ^0 all to zero gives

$$L_1 = 1.9, \quad L_2 = 2, \quad L_3 = 2.9 \tag{39}$$

(Compare this with the observer gain obtained in part (b).)

One more computation is required before the simulation can proceed: the observer gain matrix that must be applied to the *Model Predictive Control Toolbox* function `smpcsim` is $\tilde{L}'$, where $\tilde{A}\tilde{L}' = \tilde{L}$ — see Mini-Tutorial 2, on state observers. Since $\tilde{A}$ is invertible in this case, we obtain

$$\tilde{L}' = \tilde{A}^{-1}\tilde{L} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \tag{40}$$

Now the response can be simulated using the *Model Predictive Control Toolbox* functions `smpccon` and `smpcsim` as follows:

```
%%%%%%   Define plant and internal model:
A = [0.9 .5 ; 0 1]; B = [0.5 ; 0]; % Define the plant with
C = [1 0]; D = 0;                  % disturbance as in part (a)
pmod = ss2mod(A,B,C,D); % Put plant model into MOD format
imod = pmod;            % Internal model same as plant model

%%%%%%   Compute predictive controller:
Q = 1; % penalty on plant output errors
R = 0; % penalty on input moves
Hp = 30; Hu = 2; % Prediction and control horizons,
                 % respectively
Ks = smpccon(imod,Q,R,Hu,Hp)  % Compute controller
                              % gain matrix
%%% CHECK: I get Ks = [2, -1.8, 56.6163, -2]

%%%%%%   Simulate with deadbeat observer:
Lprime = [1 ; 2 ; 1]; % Observer gain L' as calculated above
tend = 50; % End time for simulation
r = 1; % Set-point = 1 (constant)
wu=[zeros(1,20),1]'; % Unmeasured step disturbance on the
                     % input after 20 time steps
[y1,u1] = smpcsim(pmod,imod,Ks,tend,r,[],Lprime,[],[],[],wu);
plotall(y1,u1)

%%%%%%   Simulate with default observer:
[y2,u2] = smpcsim(pmod,imod,Ks,tend,r,[],[],[],[],[],wu);
figure  % Open new figure window
plotall(y2,u2)
```

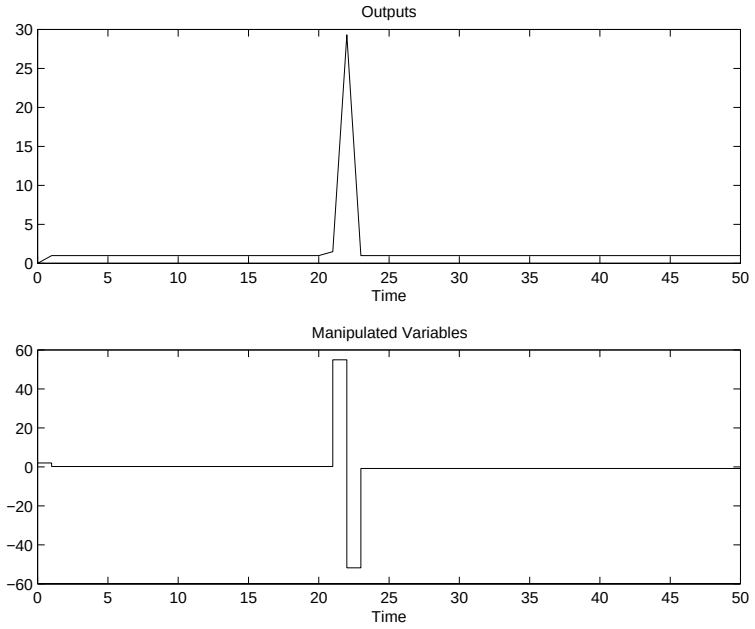

Figure 1: Exercise 2.15(c): simulation with deadbeat observer.

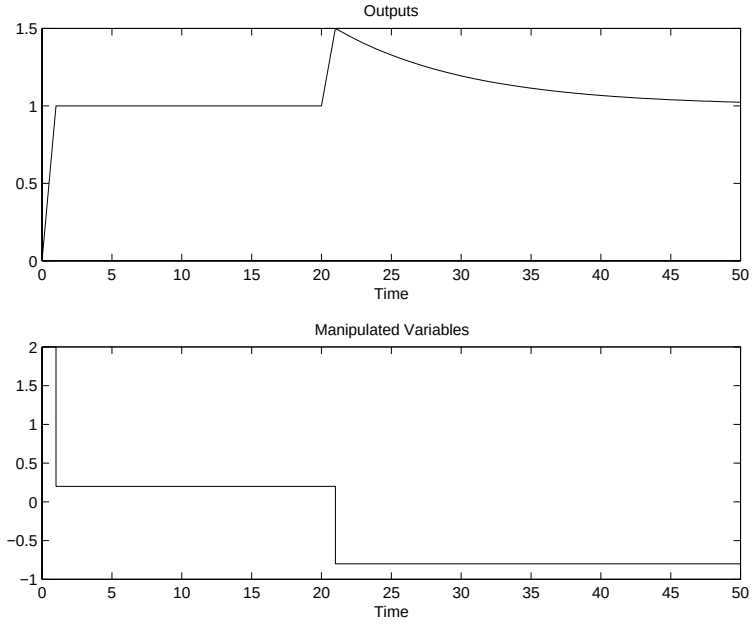

Figure 2: Exercise 2.15(c): simulation with default observer.

Figures 1 and 2 show the results using the deadbeat and default observer, respectively. As expected, with the deadbeat observer the controller completely compensates for the disturbance after two steps (because the plant with disturbance model has two states), but large input changes occur and the peak deviation of the output from its set-point is large. On the other hand the controller with the default (DMC) observer takes longer to correct for the disturbance, but the peak deviation of the output is much smaller, and much less violent input changes are made.

(d). The observer makes no difference to the set-point response, as can be seen from the two sets of plots, which are identical until the disturbance occurs. This can be checked further by defining various trajectories for the set-point vector `r` in the simulations.

(e). To use the *Control System Toolbox* function `place` it is first necessary to put the plant model into the 'augmented' form used by the *Model Predictive Control Toolbox*, as in part (c) above. The *Model Predictive Control Toolbox* function `mpcaugss` does this:

```
%%%% Assume we are given A,B,C,D %%%%
[Atilde,Btilde,Ctilde] = mpcaugss(A,B,C);
```

Placing the eigenvalues of the observer closed-loop transition matrix $\tilde{A} - L\tilde{C}$ is the dual of the problem of placing the eigenvalues of the closed-loop state-feedback matrix $\tilde{A} - \tilde{B}K$, which is solved by the function `place`:

```
%%%% Assume we have vector P of desired closed-loop
%%%% observer pole locations, and solve
%%%% dual pole-placement problem:
L = place(Atilde',Ctilde',P);  % Atilde and Ctilde
                               % must be transposed
L = L';   % and L itself must be transposed too to get its
          % dimensions right.
```

Note that there is a restriction, that no pole in `P` can have multiplicity greater than the number of outputs of the plant (beause of the particular algorithm used in `place`). So you can design a deadbeat observer only approximately in this way, because you have to specify poles close to 0, but not all exactly at 0.

Finally, as remarked in (c) above, the observer gain matrix required for *Model Predictive Control Toolbox* functions such as `smpcsim` is L' rather than L, where $L = \tilde{A}L'$:

```
Lprime = Atilde\L;
```

This can now be used with *Model Predictive Control Toolbox* functions, for example as shown in part (c) above.

2.16 Suppose that the (measured) plant output vector has constant value $y(k) = y_0$. If the plant input is also constant and if the observer (2.86) is asymptotically stable then the estimated state will converge to a constant value $\hat{\xi}_0 = [\hat{x}_0^T, \hat{d}_0^T]^T$. From the second 'row' of (2.86) we see that

$$\hat{d}_0 = \hat{d}_0 + L_d(y_0 - C_y\hat{x}_0 - \hat{d}_0) \tag{41}$$

But from (2.79) we have the output estimate

$$\hat{y}_0 = C_y\hat{x}_0 + \hat{d}_0 \tag{42}$$

and hence $\hat{y}_0 = y_0$.

Chapter 3

Solving predictive control problems

3.1 No solution required here. Follow *Model Predictive Control Toolbox* User's Guide.

3.2 No solution required here. Follow *Model Predictive Control Toolbox* User's Guide.

3.3 (a). The continuous-time state-space model of the swimming pool is

$$\dot{\theta} = -\frac{1}{T}\theta + \left[\frac{k}{T}, \frac{1}{T}\right] \begin{bmatrix} q \\ \theta_a \end{bmatrix} \tag{1}$$

where θ is taken as the state variable. The following *MATLAB* code fragment gets the discrete-time model:

```
kmod = 0.2; Tmod = 1; % assumed model parameters
amod = -1/Tmod; bmod = [kmod/Tmod, 1/Tmod]; % A and B
cmod = 1; dmod = [0, 0];                    % C and D
modc = ss(amod,bmod,cmod,dmod); % Makes LTI system object
Ts = 0.25;  % Sampling interval
modd = c2d(modc,Ts);    % Convert to discrete-time model
[ad,bd] = ssdata(modd); % A and B matrices of discrete-time
                        % model. C and D stay unchanged.
```

Now you should have `ad = 0.7788` and `bd = [0.0442, 0.2212]`, which gives (3.94). This could also have been obtained by using transfer functions, using *Control System Toolbox* function `tf` instead of `ss`, or possibly `tf2ss`.

To form the model in the *Model Predictive Control Toolbox*'s MOD format, first define the `minfo` vector, which specifies the sampling interval, and that there is 1 state, 1 control input, 1 measured output and 1 unmeasured disturbance:

```
minfo = [Ts,1,1,0,1,1,0];
```

Now create the model in MOD format:

```
imod = ss2mod(ad,bd,c,d,minfo);
```

(b). Define the parameters (using the *Model Predictive Control Toolbox* notation), then compute K_s:

```
ywt = 1;  uwt = 0; % Q and R, respectively
P = 10;   M = 3;   % Hp and Hu respectively
Ks = smpccon(imod,ywt,uwt,M,P); % Should give
                % [22.6041,  -17.6041, -22.6041]
```

Closed-loop stability can be verified either by simulating the closed loop using `smpcsim` — see below — or by forming the closed-loop system and checking the closed-loop poles (that is, the eigenvalues of its state-transition matrix):

```
clmod = smpccl(imod,imod,Ks); % Form closed-loop system
smpcpole(clmod)               % Compute closed-loop poles
```

This shows 2 poles at the origin, and 1 at 0.7788. These are all inside the unit disk, so the closed-loop is stable. (The poles at 0 indicate that 'deadbeat' closed-loop behaviour has been obtained, because of using $R = 0$. The pole at 0.7788 is the open-loop plant pole, which arises from the use of the default 'DMC' observer; it does not appear in the set-point → output transfer function.)

(c). To simulate the behaviour with a perfect model over 25 hours use `smpcsim` as follows:

```
tend = 25; % End time
setpoint = 20;
airtemp = 15;
[wtemp,power] = smpcsim(imod,imod,Ks,tend,setpoint,...
                        [],[],[],[],airtemp,[]);
                % wtemp is water temperature.
```

To see what happens when the plant parameters change, create a new model of the plant `pmod`:

```
kplant = 0.3; Tplant = 1.25;
% Now proceed as in part (a) above, get discrete-time model,
% and convert to MOD format, eventually getting:
pmod = ss2mod(aplantd,bplantd,c,d,minfo);
```

Now simulate (see Figure 1):

```
[wtemp,power] = smpcsim(pmod,imod,Ks,tend,setpoint,...
                        [],[],[],[],airtemp,[]);
timevector = (0:0.25:120)'; % for plotting purposes
plotall(wtemp,power,timevector(1:length(wtemp)) % Plot results
```

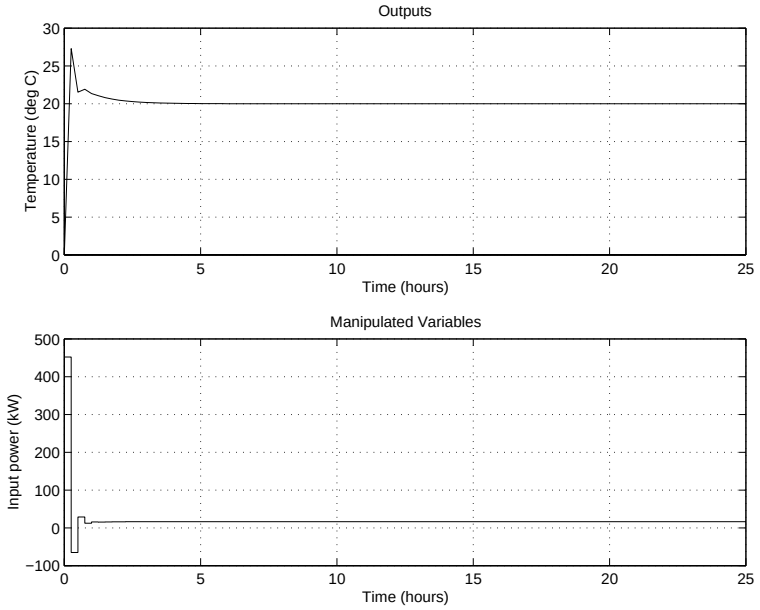

Figure 1: Solution to Exercise 3.3(c).

(d). The only change required here is to redefine the air temperature variation. Here we run the simlation over 120 hours (Figure 2):

```
airtemp = 15+10*sin(2*pi*timevector/24); % Diurnal variation
tend = 120;
[wtemp,power] = smpcsim(pmod,imod,Ks,tend,setpoint,...
                        [],[],[],[],airtemp,[]);
plotall(wtemp,power,timevector)
```

(e). Now the limits on q must be specified. In order to solve the QP problem, *Model Predictive Control Toolbox* function `scmpc` is needed (Figure 3):

```
ulim = [ 0, 40, 1e6]; % Finite limit on slew rate required.
[wtemp,power] = scmpc(pmod,imod,ywt,uwt,M,P,tend,setpoint,...
                      ulim,[],[],[],[],airtemp,[]);
```

3.4 *Error in question: The question should make reference to equation (2.66), not to (2.23).*

As in Exercise 1.14, the vector $\Delta\mathcal{U}(k)$ should contain only the changes of the control signal (using the definition of $\Delta\mathcal{U}(k)$ introduced in Section 3.1, equation (3.2)):

$$\Delta\mathcal{U}(k) = \begin{bmatrix} \Delta\hat{u}(k+M_1|k) \\ \Delta\hat{u}(k+M_2|k) \\ \vdots \\ \Delta\hat{u}(k+M_{H_u}|k) \end{bmatrix} \tag{2}$$

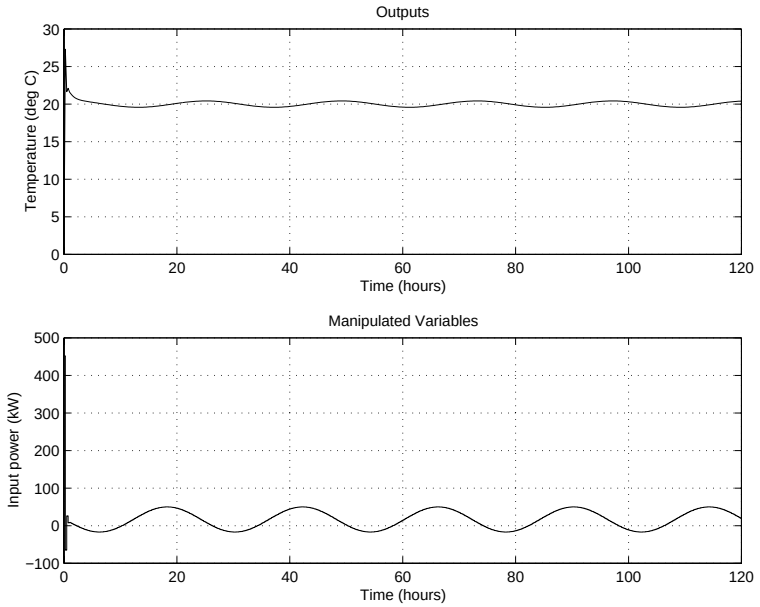

Figure 2: Solution to Exercise 3.3(d). The water and air temperatures are shown by the solid and dotted lines, respectively.

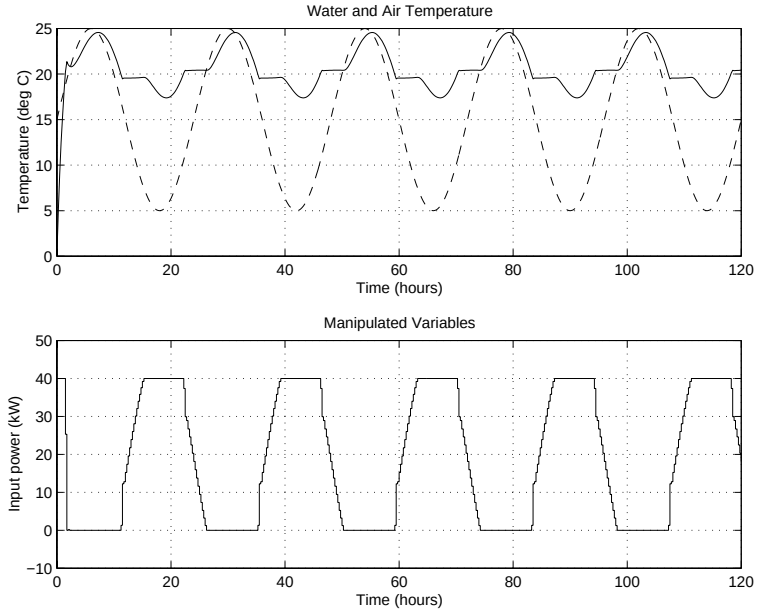

Figure 3: Solution to Exercise 3.3(e). The water and air temperatures are shown by the solid and dotted lines, respectively.

and the matrix which multiplies this vector in (2.66) should contain only the corresponding columns. In the following expression it should be understood that *negative powers of A are to be replaced by the zero matrix*, so that a prediction $\hat{x}(k+j|k)$ does not depend on $\Delta\hat{u}(k+\ell|k)$ if $\ell \geq j$:

$$\begin{bmatrix} \sum_{i=0}^{-M_1} A^i B & 0 & \cdots & 0 \\ \sum_{i=0}^{1-M_1} A^i B & \sum_{i=0}^{1-M_2} A^i B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{H_u-1-M_1} A^i B & \sum_{i=0}^{H-u-1-M_2} A^i B & \cdots & \sum_{i=0}^{H_u-1-M_{H_u}} A^i B \\ \sum_{i=0}^{H_u-M_1} A^i B & \sum_{i=0}^{H-u-M_2} A^i B & \cdots & \sum_{i=0}^{H_u-M_{H_u}} A^i B \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{H_p-M_1} A^i B & \sum_{i=0}^{H-p-M_2} A^i B & \cdots & \sum_{i=0}^{H_p-M_{H_u}} A^i B \end{bmatrix} \tag{3}$$

The first two terms in (2.66) remain unchanged.

These changes propagate through to the terms Θ and $\Delta\mathcal{U}(k)$ in equation (3.5), the other terms remaining unchanged.

3.5 (a). The only difference as regards prediction is that the values of $\hat{u}(k+i|k)$ $(i = 1, \ldots, H_u - 1)$ must be obtained, either explicitly or implicitly, in terms of $\Delta\mathcal{U}(k)$ and $u(k-1)$. Explicitly, these are given by

$$\mathcal{U}(k) = \begin{bmatrix} \hat{u}(k|k) \\ \hat{u}(k+1|k) \\ \vdots \\ \hat{u}(k+H_u-1|k) \end{bmatrix} = \begin{bmatrix} I \\ I \\ \vdots \\ I \end{bmatrix} u(k-1) + \begin{bmatrix} I & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ I & I & \cdots & I \\ \vdots & \vdots & \ddots & \vdots \\ I & I & \cdots & I \end{bmatrix} \Delta\mathcal{U}(k) \tag{4}$$

Now the cost function can be expressed as

$$V(k) = ||\mathcal{Z}(k) - \mathcal{T}(k)||^2_{\mathcal{Q}} + ||\mathcal{U}(k) - \mathcal{U}_{ref}(k)||^2_{\mathcal{S}} + ||\Delta\mathcal{U}(k)||^2_{\mathcal{R}} \tag{5}$$

which replaces (3.2) in the book, where $\mathcal{U}_{ref}(k)$ is a vector containing the prescribed future trajectory of the inputs and $\mathcal{U}(k)$ is defined above. $\mathcal{S}$ is a block-diagonal matrix with the penalty weight $S(i)$ in the ith block. But we saw above that we can express $\mathcal{U}(k)$ in the form

$$\mathcal{U}(k) = \mathcal{M}u(k-1) + \mathcal{N}\Delta\mathcal{U}(k) \tag{6}$$

with suitable matrices $\mathcal{M}$ and $\mathcal{N}$. So instead of equations (3.7)–(3.9) we

have

$$\begin{aligned} V(k) &= ||\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)||^2_{\mathcal{Q}} + ||\mathcal{M}u(k-1) + \mathcal{N}\Delta\mathcal{U}(k) - \mathcal{U}_{ref}(k)||^2_{\mathcal{S}} \\ &\qquad + ||\Delta\mathcal{U}(k)||^2_{\mathcal{R}} \\ &= \underbrace{\mathcal{E}(k)^T\mathcal{Q}\mathcal{E}(k) - 2\Delta\mathcal{U}(k)^T\Theta^T\mathcal{Q}\mathcal{E}(k) + \Delta\mathcal{U}(k)^T[\Theta^T\mathcal{Q}\Theta + \mathcal{R}]\Delta\mathcal{U}(k)}_{\text{as before}} \\ &\qquad + (\mathcal{M}u(k-1) - \mathcal{U}_{ref}(k))^T\mathcal{S}(\mathcal{M}u(k-1) - \mathcal{U}_{ref}(k)) \\ &\qquad - 2\Delta\mathcal{U}(k)^T\mathcal{N}^T\mathcal{S}(\mathcal{M}u(k-1) - \mathcal{U}_{ref}(k)) + \Delta\mathcal{U}(k)^T\mathcal{N}^T\mathcal{S}\mathcal{N}\Delta\mathcal{U}(k) \end{aligned} \quad (7)$$

which is of the form

$$V(k) = const - \Delta\mathcal{U}(k)^T\mathcal{G}' + \Delta\mathcal{U}(k)^T\mathcal{H}'\Delta\mathcal{U}(k) \quad (8)$$

where $\mathcal{G}' = \mathcal{G} + \mathcal{M}u(k-1) - \mathcal{U}_{ref}(k)$ and $\mathcal{H}' = \mathcal{H} + \mathcal{N}^T\mathcal{S}\mathcal{N}$ (compare these with (3.10) and (3.11)).

So $V(k)$ is a quadratic function of $\Delta\mathcal{U}(k)$ again, as in the standard formulation in the book. With the usual linear constraints on the inputs and states, computation of the optimal $\Delta\mathcal{U}(k)$ again requires the solution of a QP problem.

(b). The main reason for specifying trajectories for the inputs as well as for outputs is usually economic. Some inputs may be associated with higher costs than others, so if a choice of input trajectories exists it may be important to 'steer' the solution towards the lowest possible cost. Most commonly this relates to steady-state values, the optimal combination being found from a steady-state economic optimization. (See also Section 10.1.1.) The main reason against doing this is that if steady-state gains are not known accurately, then pushing the inputs towards particular values will prevent the controller from achieving offset-free tracking, because those values are likely to be inconsistent with offset-free tracking.

3.6 In this case the constraints are of the form

$$\begin{aligned} \hat{u}(k+i|k) - u_{high}(k+i) &\le 0 \\ \hat{u}(k+i|k) + u_{low}(k+i) &\le 0 \end{aligned}$$

so that F has the form

$$F = \begin{bmatrix} I & 0 & \cdots & 0 & -u_{high}(k) \\ -I & 0 & \cdots & 0 & +u_{low}(k) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I & -u_{high}(k+H_u-1) \\ 0 & 0 & \cdots & -I & +u_{low}(k+H_u-1) \end{bmatrix}$$

Now F_1 is obtained by summing block-columns 1 to H_u, F_2 is obtained by summing block-columns 2 to H_u, etc., so that F has the form indicated in the question.

The right hand side is $-\mathcal{F}_1 u(k-1) - f$, which is in this case:

$$-\begin{bmatrix} I \\ -I \\ \vdots \\ I \\ -I \end{bmatrix} u(k-1) - \begin{bmatrix} -u_{high}(k) \\ +u_{low}(k) \\ \vdots \\ -u_{high}(k+H_u-1) \\ +u_{low}(k+H_u-1) \end{bmatrix} = \begin{bmatrix} -u(k-1)+u_{high}(k) \\ +u(k-1)-u_{low}(k) \\ \vdots \\ -u(k-1)+u_{high}(k+H_u-1) \\ +u(k-1)-u_{low}(k+H_u-1) \end{bmatrix} \quad (9)$$

3.7 *Function modification:*

At line 252 of function `smpccon` we find:

```
X = [diag(ywts)*Su;diag(uwts)]\[diag(ywts);zeros(mnu,pny)];
                                              % This is X=A\B.
```

It will be seen that this implements the unconstrained solution to the MPC problem in the form given in equation (3.28) in the book. `diag(ywts)` corresponds to $\mathcal{S}_\mathcal{Q}$, where $\mathcal{S}_\mathcal{Q}^T\mathcal{S}_\mathcal{Q} = \mathcal{Q}$, and `diag(uwts)` corresponds to $\mathcal{S}_\mathcal{R}$, where $\mathcal{S}_\mathcal{R}^T\mathcal{S}_\mathcal{R} = \mathcal{R}$. Here `ywts` is a vector made up of the diagonal elements of the square roots of the output weights, $[\sqrt{Q_{11}(1)}, \sqrt{Q_{22}(1)}, \ldots, \sqrt{Q_{mm}(H_p)}]$ and `uwts` is a similar vector made up of square roots of the input weights $[\sqrt{R_{11}(0)}, \sqrt{R_{22}(1)}, \ldots, \sqrt{R_{\ell\ell}(H_u-1)}]$. The '`diag`' function converts them into diagonal matrices, so that `diag(ywts)` becomes a square matrix of dimension mH_p and `diag(uwts)` becomes a square matrix of dimension ℓH_u.

All that needs to be done is to replace these diagonal matrices by block-diagonal matrices corresponding to the matrices $\mathcal{S}_\mathcal{Q}$ and $\mathcal{S}_\mathcal{R}$, respectively. One way of doing this is to allow the input arguments `ywts` and `uwts` to the function `smpccon` to be three-dimensional arrays with dimensions $m \times m \times H_p$ and $\ell \times \ell \times H_u$, respectively, so that `ywts(i,j,k)` represents $Q_{ij}^{1/2}(k)$, etc., then form them into block-diagonal matrices as follows:

```
Qroot = [];  Rroot = [];
for k=1:P,  %%% P is prediction horizon in MPC Toolbox
  Qroot = blkdiag(Qroot,ywts(:,:,k));
end
for k=1:M,  %%% M is control horizon in MPC Toolbox
  Rroot = blkdiag(Rroot,uwts(:,:,k));
end
```

These can then be used in the formation of `X`:

```
X = [Qroot*Su;Rroot]\[Qroot;zeros(mnu,pny)]; % This is X=A\B.
```

For this to work some dimension-checking code must be commented out in the function `smpccon`.

It is also possible to have the input arguments `ywts` and `uwts` corresponding to $\mathcal{Q}$ and $\mathcal{R}$ rather than $\mathcal{S}_\mathcal{Q}$ and $\mathcal{S}_\mathcal{R}$, but then matrix square roots must be computed when forming the variables `Qroot` and `Rroot`, for instance by finding Cholesky factors:

```
for k=1:P,
  Qroot = blkdiag(Qroot,chol(ywts(:,:,k)));
end
for k=1:M,
  Rroot = blkdiag(Rroot,chol(uwts(:,:,k)));
end
```

Solution of Example 6.1: Using the function `mysmpccon`, modified as described above, Example 6.1 can be solved as follows:

```
a = [0,0 ; 1,0];  b=[1 ; 0];
c = eye(2); d = zeros(2,1);  % Both states are controlled outputs
imod = ss2mod(a,b,c,d); % Put model into MPC Toolbox MOD format.

Hp = 1; % Prediction horizon
Hu = 1; % Control horizon
ywt(:,:,1) = [1,2 ; 2,6]; % Only 1 matrix needed since Hp=1.
uwt = 1e-10; % Not zero because 'chol' needs
             % positive definite weight.

Ks = mysmpccon(imod,ywt,uwt,Hu,Hp)
%%% CHECK: I get Ks = [1, 2, -2, 0, -1, -2]

% Simulation of the resulting closed loop:
[y,u] = smpcsim(imod,imod,Ks,10,[1,1]);
plotall(y,u)
```

Figure 4 shows the resulting closed-loop behaviour. It can be seen to be unstable, as predicted in Example 6.1.

Solution of Exercise 6.2:
For part (a) only `Hp` has to be changed from 1 to 2, and `ywt(:,:,2)` needs to be defined:

```
Hp = 2;
ywt(:,:,2)=ywt(:,:,1);
Ks = mysmpccon(imod,ywt,uwt,Hu,Hp)
%%% CHECK: I get Ks = [0.3333 0.8333 -0.8333 0 -0.3333 -0.8333]
```

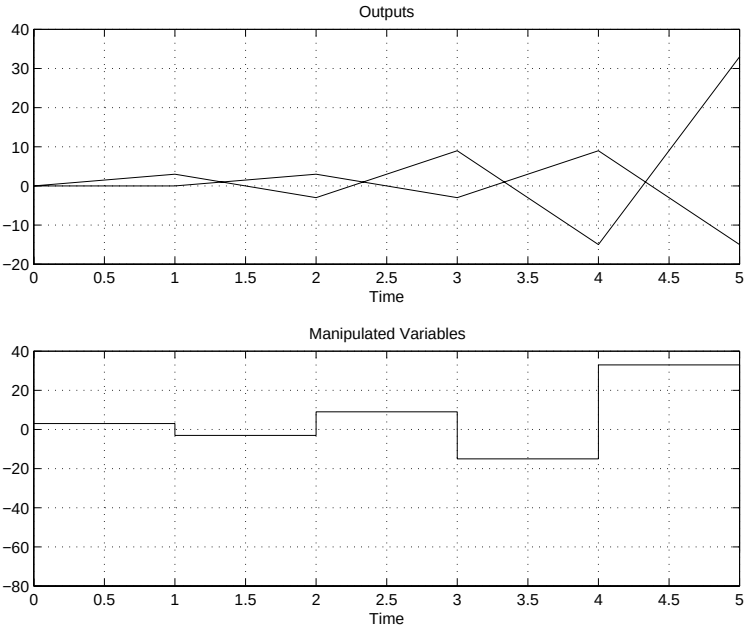

Figure 4: Exercise 3.7: simulation of Example 6.1.

```
% Simulate and plot:
[y,u] = smpcsim(imod,imod,Ks,10,[1,1]);
plotall(y,u)
```

Figure 5 shows the resulting closed-loop behaviour. It can be seen to be stable, as predicted in Exercise 6.2(a).

For part (b) `Hu` must be changed from 1 to 2, and `uwt` changed correspondingly:

```
Hu = 2;
uwt(1,1,2)=uwt(1,1,1);
Ks = mysmpccon(imod,ywt,uwt,Hu,Hp)
%%% CHECK: I get Ks = [0.3333 1.3333 -1.3333 0 -0.3333 -1.3333]
% Simulate and plot:
[y,u] = smpcsim(imod,imod,Ks,10,[1,1]);
plotall(y,u)
```

Figure 6 shows the resulting closed-loop behaviour. It can be seen to be stable, as predicted in Exercise 6.2(b).

3.8 The objective function can be written as

$$\frac{1}{2}\theta^T \underbrace{\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}}_{\Phi} \theta + \underbrace{\begin{bmatrix} 0 \\ -3 \end{bmatrix}^T}_{\phi} \theta$$

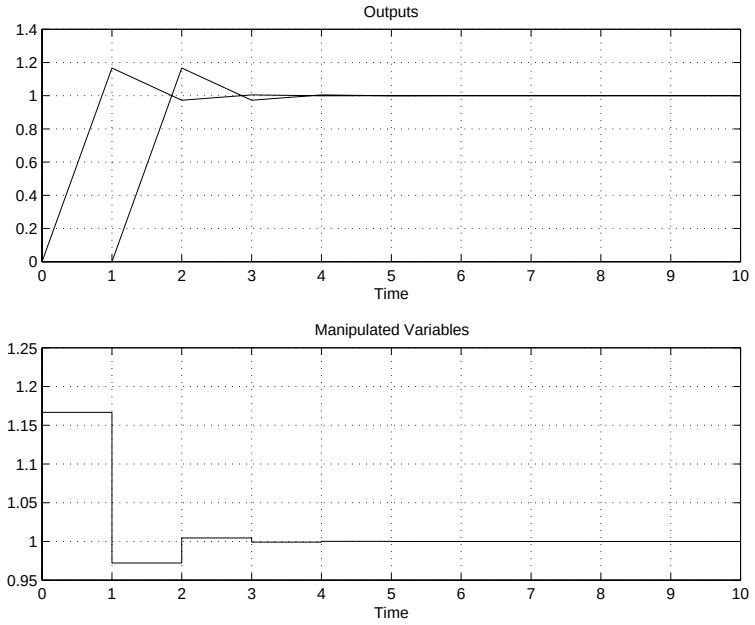

Figure 5: Exercise 3.7: simulation of Exercise 6.2(a).

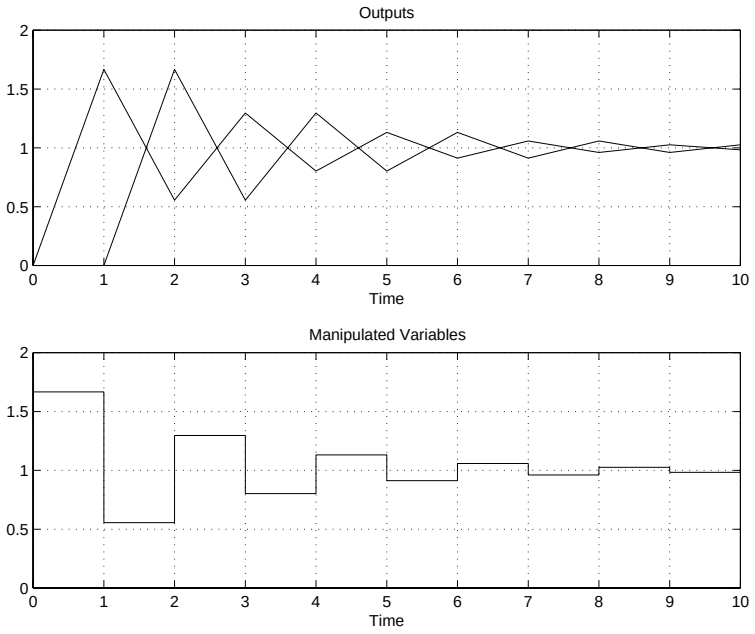

Figure 6: Exercise 3.7: simulation of Exercise 6.2(b).

and the constraints can be written as

$$\underbrace{\begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix}}_{\Omega} \theta \leq \underbrace{\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}}_{\omega}$$

The eigenvalues of Φ are 3 and 1, so $\Phi > 0$. Hence the QP problem is convex.

KKT conditions:
There are no equality constraints in this problem, so H and h are empty in this case. Condition (3.61) gives $t = [3/2, 1/2, 0]^T$ if $\theta = [3/2, 1/2]^T$. Hence, from the complementarity condition $t^T\lambda = 0$ (3.62) and the fact that $\lambda \geq 0$ and $t \geq 0$, the first two elements of λ must be zero, so that $\lambda = [0, 0, \lambda_3]^T$. So finally we have to check the condition (3.59):

$$\begin{aligned} \Omega^T\lambda &= -\Phi\theta - \phi \\ \begin{bmatrix} -1 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \lambda_3 \end{bmatrix} &= \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \end{aligned}$$

which is satisfied by

$$\lambda_3 = \frac{1}{2}$$

Thus the *KKT* conditions are satisfied, and $\theta = [3/2, 1/2]^T$ is the optimal solution.

3.9 (a). The optimization problem can be written as

$$\min_{\theta,\epsilon} \frac{1}{2}[\theta^T \quad \epsilon^T] \begin{bmatrix} \Phi & 0 \\ 0 & \rho I \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} + [\phi^T \quad 0] \begin{bmatrix} \theta \\ \epsilon \end{bmatrix}$$

subject to

$$\begin{bmatrix} \Omega & -I \\ 0 & -I \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} \leq \begin{bmatrix} \omega \\ 0 \end{bmatrix}$$

which is in the standard form of a QP problem.

(b). The expression (3.86) stays unchanged, except that the dimension of ϵ is now the same as that of ω_2. Inequalities (3.87) should be replaced by

$$\begin{aligned} \Omega_1\theta &\leq \omega_1 \\ \Omega_2\theta &\leq \omega_2 + \epsilon \\ \epsilon &\geq 0 \end{aligned}$$

So the optimization problem can be written as

$$\min_{\theta,\epsilon} \frac{1}{2}[\theta^T \quad \epsilon^T] \begin{bmatrix} \Phi & 0 \\ 0 & \rho I \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} + [\phi^T \quad 0] \begin{bmatrix} \theta \\ \epsilon \end{bmatrix}$$

subject to

$$\begin{bmatrix} \Omega_1 & 0 \\ \Omega_2 & -I \\ 0 & -I \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} \leq \begin{bmatrix} \omega_1 \\ \omega_2 \\ 0 \end{bmatrix}$$

which is again in the standard form of a QP problem.

3.10 *Error in question:* The reference to (3.4) is spurious.

1-norm problem, (3.88):
The cost function can be written as

$$\frac{1}{2}[\theta^T \quad \epsilon^T] \begin{bmatrix} \Phi & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} + [\phi^T \quad \rho \mathbf{1}^T] \begin{bmatrix} \theta \\ \epsilon \end{bmatrix}$$

subject to

$$\begin{bmatrix} \Omega & -I \\ 0 & -I \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} \leq \begin{bmatrix} \omega \\ 0 \end{bmatrix}$$

where $\mathbf{1}$ is a vector of the same dimension as ϵ, each element of which is 1. This is in the standard form of a QP problem.

∞-norm problem, (3.89)–(3.91).
Take the second formulation, namely that of (3.90)–(3.91). This can be written as

$$\frac{1}{2}[\theta^T \quad \epsilon^T] \begin{bmatrix} \Phi & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} + [\phi^T \quad \rho] \begin{bmatrix} \theta \\ \epsilon \end{bmatrix}$$

subject to

$$\begin{bmatrix} \Omega & -\mathbf{1} \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \theta \\ \epsilon \end{bmatrix} \leq \begin{bmatrix} \omega \\ 0 \end{bmatrix}$$

Again this is in the standard form of a QP problem.

3.11 No solution required. Use function `mismatch2` as suggested in the question.

3.12 No solution required. Use function `disturb2` as suggested in the question.

Chapter 4

Step response and transfer function formulations

4.1 $S(k) = H(0) + H(1) + \ldots + H(k)$. Hence $H(k) = S(k) - S(k-1)$. So we can find $H(N), H(N-1), \ldots, H(1)$ without any problems. For $H(0)$ it seems that we need $H(-1)$; but recall that for any causal system we must have $H(k) = 0$ for $k < 0$. Hence we can also get $H(0) = S(0)$.

4.2 (a). This is probably the simplest, though not the most efficient, implementation. Note that you get $S(1), \ldots, S(N)$, even if $M > 1$. With the given interface specification you can't get $S(0)$, since that would have to be stored in `S(:,:,0)`, but a zero index to an array is not allowed in *MATLAB*.

```
function S = ss2step(A,B,C,D,M,N)
S(:,:,1) = C*B+D;
for t=2:N,
  S(:,:,t) = S(:,:,t-1) + C*A^(t-1)*B; % From (4.24)
end
```

(b).
```
function upsilon = step2ups(S,Hw,Hp)    % Follow (4.27)
[noutputs,ninputs,nsteps] = size(S);
if Hp>nsteps,
  error('Hp exceeds number of responses stored in S')
end
% Reserve space (for speed):
upsilon = zeros((Hp-Hw+1)*noutputs,ninputs);
for t=Hw:Hp,
  upsilon((t-Hw)*noutputs+1:(t-Hw+1)*noutputs,:) = S(:,:,t);
end

function theta = step2the(S,Hw,Hp,Hu)   % Follow (4.28)
[noutputs,ninputs,nsteps] = size(S);
```

```
% Reserve space (for speed):
theta = zeros((Hp-Hw+1)*noutputs,Hu*ninputs);
for j=1:min(Hw,Hu),  % Do blocks of columns at a time
  theta(:,(j-1)*ninputs+1:j*ninputs) = ...
                  step2ups(S,Hw-j+1,Hp-j+1);
end
for j=Hw+1:Hu, % this skipped if Hu <= Hw
  theta((j-Hw)*noutputs+1:(Hp-Hw+1)*noutputs,...
        (j-1)*ninputs+1:j*ninputs) = step2ups(S,1,Hp-j+1);
end
```

4.3 In this case $p(H_p - H_w + 1) = m$, so that Θ and Υ are square. Furthermore from (4.27) and (4.28) we have that $\Upsilon = \Theta = S(H_w)$. But from (3.27) we have $K_{MPC} = \mathcal{H}^{-1}\Theta^T\mathcal{Q}$, since $\mathcal{H}$ has only m rows in this case. Also, we have $\mathcal{H} = \Theta^T\mathcal{Q}\Theta$, since $\mathcal{R} = 0$. So, assuming $S(H_w)^{-1}$ exists, we have $K_{MPC} = \Theta^{-1}\mathcal{Q}^{-1}\Theta^{-T}\Theta^T\mathcal{Q} = \Theta^{-1} = S(H_w)^{-1}$. Therefore $K_{MPC}\Upsilon = I$, and hence $I - K_{MPC}\Upsilon = 0$. Hence there is no feedback from $u(k-1)$ to $u(k)$, and hence the control law (in the full state measurement case) is:

$$u(k) = S(H_w)^{-1}r(k + H_w) - S(H_w)^{-1}C_zA^{H_w}x(k).$$

4.4 The pulse responses are obtained as: $H(0) = S(0) = [0, 0]$ (which tells us that the 'D' matrix of the state-space model will be $D = [0, 0]$), $H(1) = S(1) - S(0) = [0, 2]$, $H(2) = S(2) - S(1) = [1, 1]$, $H(3) = S(3) - S(2) = [0, 0]$ (since $S(k) = S(2)$ for $k > 2$) and similarly $H(k) = [0, 0]$ for $k > 3$. So we have a finite pulse response, hence all the eigenvalues of the 'A' matrix must be at 0. A naive (but adequate) way to proceed is to build two separate state-space models, one for each input, and then put them together.

For the first input, the required pulse response sequence is (omitting $H(0)$): $\{0, 1, 0, 0, \ldots\}$. Hence use two states; as in Example 4.1 try:

$$A_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad B_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

We need to find $C_1 = [c_1, c_2]$ such that

$$\begin{aligned} C_1B_1 &= c_2 = 0 \\ C_1A_1B_1 &= c_1 = 1 \end{aligned}$$

For the second input, the required pulse response sequence is (omitting $H(0)$): $\{2, 1, 0, 0, \ldots\}$. Again use two states; try:

$$A_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad B_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

We need to find $C_2 = [c_3, c_4]$ such that

$$\begin{aligned} C_2 B_2 &= c_4 = 2 \\ C_2 A_2 B_2 &= c_3 = 1 \end{aligned}$$

Now we get a 4-state system with the required pulse response as follows:

$$A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \qquad B = \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix}$$
$$C = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Although this is correct, the state dimension is unnecessarily large. It can be checked that the observability matrix has rank 2 rather than 4, which shows that a *minimal* realization with only two states can be found which gives the same step response. The SVD-based method given in Section 4.1.4 will find such a two-state model (and so will other less naive methods than the one used here). The SVD-based method gives

$$A = \begin{bmatrix} 0.3536 & 0.5493 \\ -0.2275 & -0.3536 \end{bmatrix} \qquad B = \begin{bmatrix} 0.2367 & 1.3798 \\ 0.8880 & 0.1524 \end{bmatrix}$$
$$C = \begin{bmatrix} 1.4935 & -0.3981 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

4.5 If the controllability matrix Γ is 'shifted left' by m columns to form the matrix $\Gamma^{\leftarrow}$ then we have $\Gamma^{\leftarrow} = A\Gamma$. So shift Γ_n left by m columns to form the matrix $\Gamma_n^{\leftarrow}$, and estimate A by solving $\Gamma_n^{\leftarrow} = A\Gamma_n$.

4.6 *Solution of equation (4.110):* We need to find polynomials $E_2'(z^{-1})$ and $F_2'(z^{-1})$, of degrees at most 1 and 0 respectively, such that

$$\frac{1 + 0.5z^{-1}}{1 - z^{-1}} = E_2'(z^{-1}) + z^{-2}\frac{F_2'(z^{-1})}{1 - z^{-1}} \tag{1}$$

Let $E_2'(z^{-1}) = e_0' + e_1' z^{-1}$ and $F_2'(z^{-1}) = f_0'$. Then, multiplying through by $1 - z^{-1}$, we need:

$$\begin{aligned} 1 + 0.5z^{-1} &= (e_0' + e_1' z^{-1})(1 - z^{-1}) + z^{-2} f_0' && (2) \\ &= e_0' + (e_1' - e_0')z^{-1} + (f_0' - e_1')z^{-2} && (3) \end{aligned}$$

Hence, by comparing coefficients, $e_0' = 1$, $e_1' = 1.5$, $f_0' = 1$. So

$$E_2'(z^{-1}) = 1 - 1.5z^{-1}, \qquad F_2'(z^{-1}) = 1 \tag{4}$$

Solution of equation (4.121): We need to find a polynomial $E_2(z^{-1})$ of degree at most 1 (because $i - d = 2 - 1 = 1$), and a polynomial $F_2(z^{-1})$, such that

$$\frac{0.5}{1 - 0.9z^{-1}} = E_2(z^{-1}) + z^{-1}\frac{F_2(z^{-1})}{1 - 0.9z^{-1}} \tag{5}$$

Letting $E_2(z^{-1}) = e_0 + e_1 z^{-1}$ and multiplying through by $1 - 0.9z^{-1}$ gives

$$0.5 = (e_0 + e_1 z^{-1})(1 - 0.9z^{-1}) + z^{-1}F_2(z^{-1}) \tag{6}$$

from which it is clear that $F_2(z^{-1})$ need be only a constant f_0. Hence

$$\begin{aligned} 0.5 &= (e_0 + e_1 z^{-1})(1 - 0.9z^{-1}) + z^{-1} f_0 && (7)\\ &= e_0 + (e_1 - 0.9e_0 + f_0)z^{-1} - 0.9e_1 z^{-2} && (8) \end{aligned}$$

from which $e_0 = 0.5$, $e_1 = 0$, $f_0 = 0.45$. So

$$E_2(z^{-1}) = 0.5, \qquad F_2(z^{-1}) = 0.45 \tag{9}$$

4.7 Let $D(z^{-1}) = (1 - z^{-1})2 = 1 - 2z^{-1} + z^{-2}$, $C(z^{-1}) = 1$, $v(0) = v_0$, $v(1) = v_1$, and $v(k) = 0$ for $k > 1$. The difference equation coresponding to equation (4.100) in the book is

$$d(k) - 2d(k-1) + d(k-2) = v(k) \tag{10}$$

Assuming $v(-1) = v(-2) = 0$ gives

$$\begin{aligned} d(0) &= v_0 \\ d(1) &= v_1 + 2d(0) = v_1 + 2v_0 \\ d(2) &= 2d(1) - d(0) = 2v_1 + 3v_0 \\ d(3) &= 2d(2) - d(1) = 3v_1 + 4v_0 \\ &\vdots \\ d(k) &= k(v_1 + v_0) + v_0 \qquad (11) \end{aligned}$$

which is a ramp disturbance of slope $v_0 + v_1$, initial value v_0.

4.8 Proceeding as in (4.70)–(4.74), the transfer function from $w(k)$ to $y(k)$ is $C_y(zI - A)^{-1}W$. But from (4.74) the transfer function from $u(k)$ to $y(k)$ is $C_y(zI - A)^{-1}B$. But we have

$$(zI - A)^{-1} = \frac{\det(zI - A)}{\text{adj}(zI - A)}$$

where the adjoint matrix $\text{adj}(zI - A)$ contains only polynomial elements. Thus the denominator of each element of both transfer function matrices is the determinant $\det(zI - A)$.

4.9 The distinguishing feature of the GPC model is that the denominator polynomial of the plant's disturbance–output transfer function is $(1 - z^{-1})A(z^{-1})$, where $A(z^{-1})$ is the denominator of the plant's input–output transfer function.

First write the model (4.142)–(4.144) in the form

$$\begin{aligned}\begin{bmatrix} x_p(k+1) \\ x_d(k+1) \end{bmatrix} &= \begin{bmatrix} A_p & I \\ 0 & I \end{bmatrix} \begin{bmatrix} x_p(k) \\ x_d(k) \end{bmatrix} + \begin{bmatrix} B_p \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ B_d \end{bmatrix} v_2(k) \\ y(k) &= [C_p \quad C_d] \begin{bmatrix} x_p(k) \\ x_d(k) \end{bmatrix}\end{aligned}$$

Applying (4.74) to this model, the transfer function from u to y is

$$\begin{aligned}P_{yu}(z) &= [C_p \quad C_d] \begin{bmatrix} zI - A_p & -I \\ 0 & (z-1)I \end{bmatrix}^{-1} \begin{bmatrix} B_p \\ 0 \end{bmatrix} \\ &= C_p(zI - A)^{-1}B_p\end{aligned}$$

from which it follows that the denominator of the plant's input–output transfer function is $\det(zI - A)$. Applying (4.74) again, the transfer function from the disturbance v_2 to y is

$$\begin{aligned}P_{yv}(z) &= [C_p \quad C_d] \begin{bmatrix} zI - A_p & -I \\ 0 & (z-1)I \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ B_d \end{bmatrix} \\ &= \begin{bmatrix} (zI - A_p)^{-1} & \frac{(zI-A_p)^{-1}}{z-1} \\ 0 & \frac{1}{z-1}I \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ B_d \end{bmatrix} \\ &= \frac{C_p(zI - A_p)^{-1}B_d + C_dB_d}{z-1}\end{aligned}$$

from which it follows that the denominator of the plant's disturbance–output transfer function is $\det(zI - A)(z-1)$, which is the same as $\det(zI - A)(1 - z^{-1})z$. The additional z factor does not matter for the disturbance–output transfer function (it affects the phase but not the power spectrum), so this is essentially the same as the GPC model.

4.10 The first p rows of (4.151) give

$$\begin{aligned}y(k+1) &= (I + A_1)y(k) + (A_2 - A_1)y(k-1) + \cdots - A_n y(k-n) \\ &\quad + (B_2 - B_1)u(k-1) + \cdots - B_p u(k-p) \\ &\quad + C_2 v(k-1) + \cdots + C_q v(k-q+1) \\ &\quad + B_1 u(k) + v(k) \qquad (12) \\ &= y(k) + A_1[y(k) - y(k-1)] + \cdots + A_n[y(k-n+1) - y(k-n)] \\ &\quad + B_1[u(k) - u(k-1)] + \cdots + b_p[u(k-p+1) - u(k-p)] \\ &\quad + v(k) + C_2 v(k-1) + \cdots + C_q v(k-q+1) \qquad (13)\end{aligned}$$

which is the same as (4.149) with k replaced by $k+1$. The remaining rows of (4.151) are 'housekeeping' to ensure that identities are satisfied (such as $y(k) = y(k)$) on each side of equation (4.151).

4.11 Equation (4.178) follows by forming the difference $\hat{x}(k|k) - \hat{x}(k-1|k-1)$. From the standard observer update equation (2.81) we have

$$\begin{bmatrix} \Delta\hat{x}(k|k) \\ \hat{\eta}(k|k) \end{bmatrix} = \left\{ I - \begin{bmatrix} L_{\Delta x} \\ L_\eta \end{bmatrix} \begin{bmatrix} 0 & I \end{bmatrix} \right\} \begin{bmatrix} \Delta\hat{x}(k|k-1) \\ \hat{\eta}(k|k-1) \end{bmatrix} + \begin{bmatrix} L_{\Delta x} \\ L_\eta \end{bmatrix} y(k)$$

This gives

$$\Delta\hat{x}(k|k) = \Delta\hat{x}(k|k-1) - L_{\Delta x}\hat{\eta}(k|k-1) + L_{\Delta x}y(k) \tag{14}$$

(for any value of L_η), and $\hat{\eta}(k|k) = y(k)$ if $L_\eta = I$. Now from the observer prediction equation (2.82) we have

$$\begin{bmatrix} \Delta\hat{x}(k|k-1) \\ \hat{\eta}(k|k-1) \end{bmatrix} = \begin{bmatrix} \mathcal{A} & 0 \\ \mathcal{CA} & I \end{bmatrix} \begin{bmatrix} \Delta\hat{x}(k-1|k-1) \\ \hat{\eta}(k-1|k-1) \end{bmatrix} + \begin{bmatrix} \mathcal{B}_u \\ \mathcal{CB}_u \end{bmatrix} \Delta u(k-1) \tag{15}$$

Substituting the value obtained from this for $\hat{\eta}(k|k-1)$ back into (14) gives

$$\Delta\hat{x}(k|k) = (I - L_{\Delta x}\mathcal{C})\Delta\hat{x}(k|k-1) + L_{\Delta x}[y(k) - y(k-1)]$$

since $\hat{\eta}(k-1|k-1) = y(k-1)$ — if $L_\eta = I$. Finally, note that setting $L_{\Delta x} = L'$ gives (4.178).

4.12 The following function performs the simulation and plots the results:

```
%GPC Apply GPC to unstable helicopter.
% Example 4.11 and Exercise 4.12.
%
% Usage: gpcheli(pole)

function gpcheli(pole)

% State-space model of helicopter as in Example 4.11:
A = [3.769 -5.334 3.3423 -0.7773 -8.948 10.27 -7.794 -0.8;
       1      0     0       0       0      0     0     0;
       0      1     0       0       0      0     0     0;
       0      0     1       0       0      0     0     0;
       0      0     0       0       0      0     0     0;
       0      0     0       0       1      0     0     0;
       0      0     0       0       0      1     0     0;
       0      0     0       0       0      0     0     0 ];

Bu = [6.472 0 0 0 1 0 0 0]';
C = [1,0,0,0,0,0,0,0];

imod = ss2mod(A,Bu,C,0,0.6); % Model in MOD format
```

```
% Observer gain - as in (4.166):
Kest = [1,0,0,0,0,0,0,1,1]';

% Define MPC parameters:
Hp = 5; % Prediction horizon
Hu = 2; % Control horizon
ywt = 1; % Tracking error penalty
uwt = 0; % Control move penalty

% Simulation parameters:
tend = 30; % Simulation length
r=1 % Set-point
wu = [zeros(16,1);-0.03;-0.07;-0.1]; % Input disturbance

% Generate response with obs pole at 0.8,
% with disturbance on input:
[y8w,u8w]=scmpc(imod,imod,ywt,uwt,Hu,Hp,tend,r,[],[],Kest,...
          [],[],[],wu);

% Modify model for observer pole at 'pole':
Anew = A;
Anew(1,8) = -pole;
imodnew = ss2mod(Anew,Bu,C,0,0.6);

% Generate response with obs pole at 'pole',
% with disturbance on input:
[ypw,upw]=scmpc(imodnew,imodnew,ywt,uwt,Hu,Hp,tend,r,[],[],...
          Kest,[],[],[],wu);

% Plot results:

tvec = 0:0.6:tend;

% Plots with disturbance:
figure
subplot(211)
plot(tvec,y8w,'-')
hold on
plot(tvec,ypw,'--')
xlabel('Time (seconds)')
ylabel('Output')
grid

subplot(212)
stairs(tvec,u8w,'-')
hold on
```

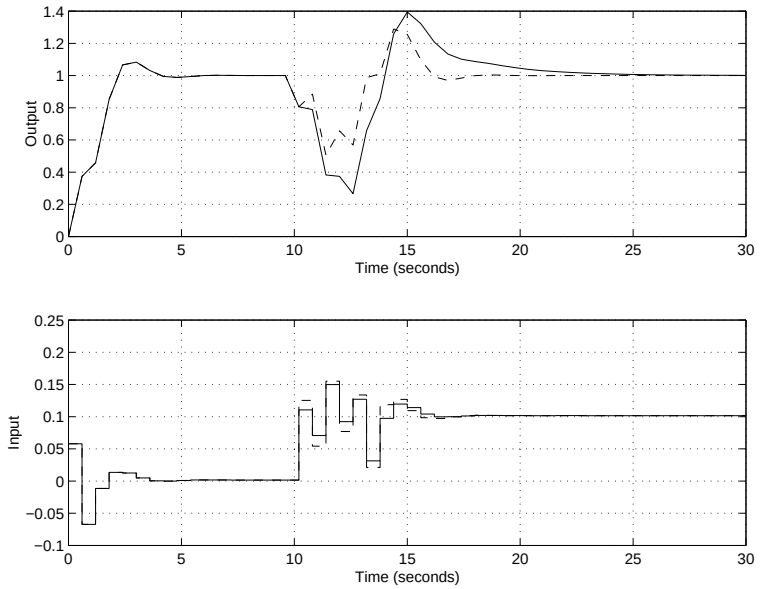

Figure 1: Exercise 4.12: helicopter response with observer pole at 0.8 (solid lines) and at 0.3 (broken lines).

```
stairs(tvec,upw,'--')
xlabel('Time (seconds)')
ylabel('Input')
grid
%%% End of function GPCHELI
```

Figure 1 shows the results with observer poles at 0.8 (solid lines) and 0.3 (broken lines).

Figure 2 shows the results with observer poles at 0.8 (solid lines) and 0.99 (broken lines).

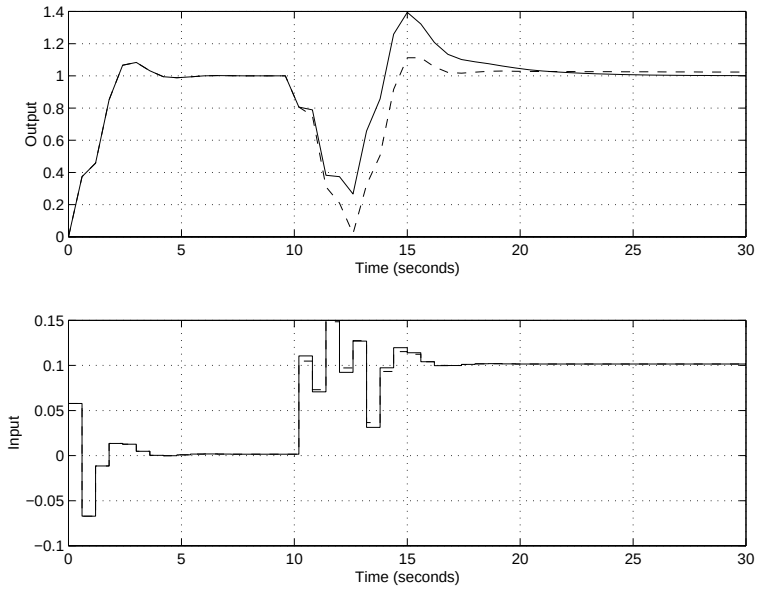

Figure 2: Exercise 4.12: helicopter response with observer pole at 0.8 (solid lines) and at 0.99 (broken lines).

Chapter 5

Other formulations of predictive control

5.1 (a). To represent the fact that the air temperature is a measured disturbance, the vector `minfo` should be defined to be:

```
minfo2 = [Ts, 1, 1, 1, 0,1,0]
```

Then new models of the plant are formed from the state-space matrices, as in Exercise 3.3(a):

```
imod2 = ss2mod(ad,bd,c,d,minfo2);
pmod2 = ss2mod(aplantd,bplantd,c,d,minfo2);
```

To see the effectiveness of feedforward when the model is perfect and there are no input constraints use:

```
[wtemp,power] = smpcsim(imod2,imod2,Ks,tend,setpoint,...
                        [],[],[],airtemp,[],[]);
```

(b). To see the effect when there is a plant/model mismatch use:

```
[wtemp,power] = smpcsim(pmod2,imod2,Ks,tend,setpoint,...
                        [],[],[],airtemp,[],[]);
```

(c). When there are input power constraints then use function `scmpc` to obtain the QP solution. Use `pmod,imod` to simulate unmeasured air temperature, and `pmod2,imod2` to simulate the effect of feedforward from air temperature measurements. (Note that the argument `airtemp` must be supplied in different positions in each of these cases.) Figure 1 shows the behaviour of the water temperature in the two cases, over one period of 24 hours, when the model is perfect (`pmod==imod` etc).

5.2 The cost function can be written as

$$V(k) = \|\mathcal{Q}[\Theta\Delta\mathcal{U}(k) - \mathcal{E}(k)]\|_1 + \|\mathcal{R}\Delta\mathcal{U}(k)\|_1 \qquad (1)$$

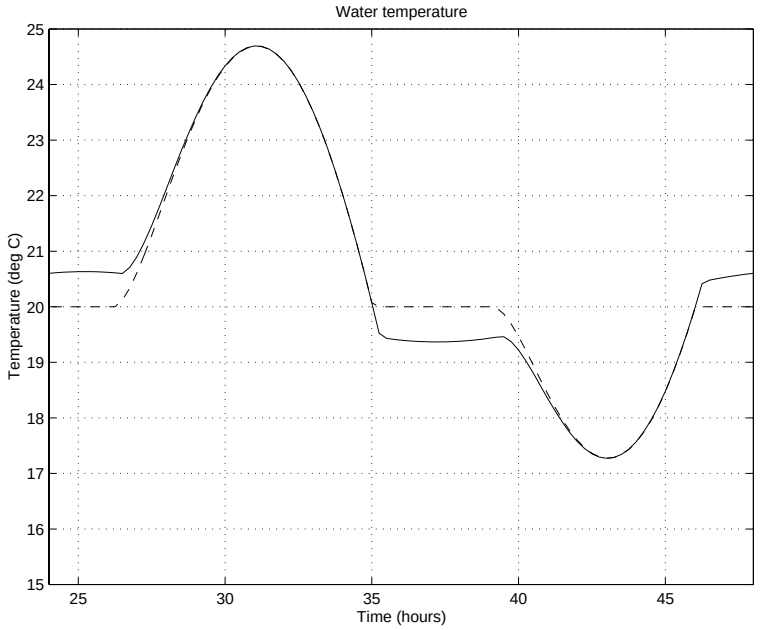

Figure 1: Exercise 5.1(c): water temperature when air temperature is unmeasured (solid line) and measured (dashed line).

But

$$\|a\|_1 + \|b\|_1 = \left\| \begin{bmatrix} a \\ b \end{bmatrix} \right\|_1$$

so

$$V(k) = \left\| \begin{bmatrix} \mathcal{Q}\times \\ \mathcal{R} \end{bmatrix} \Delta\mathcal{U}(k) - \begin{bmatrix} \mathcal{E}(k) \\ 0 \end{bmatrix} \right\|_1 \tag{2}$$

So again the resulting problem is a 1-norm minimization problem, subject to linear inequality constraints, and hence it is an LP problem.

5.3 We need to find a transfer function pair such that the numerator of $\tilde{N}(z)$ has the zeros of $P(z)$, and the numerator of $\tilde{M}(z)$ has the unstable poles of $P(z)$, with any unwanted factors cancelling out between the denominators of $\tilde{N}(z)$ and $\tilde{M}(z)$, ensuring that all the roots of the denominators lie within the unit circle. For example, any pair of the form

$$\begin{aligned} \tilde{N}(z) &= \frac{6.472z^2 - 2.476z + 7.794}{(z-p_1)(z-p_2)(z-0.675)} \\ \tilde{M}(z) &= \frac{(z-1.047+0.2352i)(z-1.047-0.2352i)}{(z-p_1)(z-p_2)} \end{aligned}$$

with $|p_1| < 1$ and $|p_2| < 1$, will be a solution. For example, $p_1 = p_2 = 0.5$ or $p_1 = 0$, $p_2 = 0.93$, would be solutions. Note that we need to have at least

two poles which cancel between $\tilde{N}(z)$ and $\tilde{M}(z)$, in order to ensure that both factors are proper, since $P(z)$ has two unstable poles.

(There are standard algorithms for obtaining coprime factors in state-space form, usually each pair having exactly the same set of poles, so that *all* the poles of $P(z)$ appear in the numerator of $\tilde{M}(z)$.)

5.4 If the predicted input signal is $\hat{u}(k+i|k) = u_0(k) + u_1(k)i$, this can be visualized as a superposition of the constant signal $u_0(k)$, which is present throughout the prediction horizon, with a step of amplitude $u_1(k)$ which starts at time $k+1$, another step of amplitude $u_1(k)$ starting at time $k+2$, and so on. The effect of each of these steps on the output at time $k+H_p$ depends on how much time has elapsed between the occurence of the step signal and $k+H_p$ (that is, the usual convolution argument). So the effect of the step which occurs at time k is given by the step response after H_p steps, namely $S(H_p)$, etc. Thus, if we were starting with zero initial conditions, we would have

$$\hat{y}(k+H_p|k) = \\ S(H_p)u_0(k) + S(H_p-1)u_1(k) + S(H_p-2)u_1(k) + \cdots + S(1)u_1(k) \quad (3)$$

However, we start with some general initial conditions, so we must also take into account the free response of the system. Hence we get

$$\hat{y}(k+H_p|k) = \\ S(H_p)u_0(k) + [S(H_p-1) + S(H_p-2) + \cdots + S(1)]u_1(k) + \hat{y}_f(k+H_p|k) \quad (4)$$

If $\hat{u}(k+i|k)$ denotes the actual control signal level to be applied to the plant then $\hat{y}_f(k+H_p|k)$ should be the free response which results from applying a zero input signal. To be consistent with the way that 'free response' is used in the book, it should be the response that would result if the input were kept constant at $u(k-1)$; in this case $\hat{u}(k+i|k)$ should denote the change made at time $k+i$ relative to $u(k-1)$.

5.5 (a). A ramp is a polynomial of degree $\nu = 1$. There must be at least $\nu + 1 = 2$ coincidence points. ('Common-sense' reasoning: at least 2 points are needed to define a straight line.)

(b). The future input trajectory should be structured as a polynomial of degree $\nu = 1$, that is, as a straight line:

$$\hat{u}(k+i|k) = u_0(k) + u_1(k)i$$

This has 2 parameters, $u_0(k)$ and $u_1(k)$, which can be chosen to give zero predicted error at the 2 coincidence points.

(c). If there are two inputs, then other possibilities may exist for achieving zero predicted error at the coincidence points. If the transfer functions from the two inputs to the output are different, then it may be sufficient to assume that each input is a constant, and choose the two levels appropriately. (This is only possible if the dynamics are different, not just the gains.) In some applications it may be preferable to structure the two inputs identically as ramps (one a multiple of the other), or to use only one input, keeping the other one in reserve.

(d). The other feature needed to get error-free tracking of the ramp set-point is a model of the plant-model error. Assuming an asymptotically stable model is used, the error can be modelled as a ramp, whose 2 parameters should be fitted from previous errors between the plant output and the (*independent*) model output.

5.6 (a). The *Model Predictive Control Toolbox* function `smpcsim` applies the unconstrained controller computed by `smpccon`, and merely 'clips' the control input signal at its saturation limits. Proceed as in the solution to Exercise 3.3(e); in the last step, instead of using `scmpc`, use the following *MATLAB* command:

```
[wtemp,power] = smpcsim(pmod,imod,Ks,tend,setpoint,...
                        ulim,[],[],[],airtemp,[]);
```

In this case this gives virtually the same results as the QP solution computed by `scmpc`.

(b). Since the input power slew rate is limited to 20 kW/hour and the control update interval is $T_s = 0.25$ hours, the control increment is limited to $|\Delta q| \leq 20T_s = 5$ kW. To represent this in the *Model Predictive Control Toolbox*, re-define `ulim` as: `ulim = [0, 40, 5]`. Now proceed as above, using `smpcsim` to get the 'clipped' solution and `scmpc` to get the optimal QP solution.

5.7 From (5.50) the first element of $\mathcal{Y}(t)$ is given by $Cx(t)$, which is $y(t)$, from (5.49). The second element is given by $CAx(t) + CBu(t) = C[Ax(t) + Bu(t)] = C\dot{x}(t) = \dot{y}(t)$, so this agrees. The third element is given by $CA^2x(t) + CABu(t) + CB\dot{u}(t) = CA[Ax(t) + Bu(t)] + CB\dot{u}(t) = C[A\dot{x}(t) + B\dot{u}(t)] = C\ddot{x}(t) = \ddot{y}(t)$, which agrees again. And so on ....

5.8 Let

$$\mathcal{U}(t) = \begin{bmatrix} u(t) \\ \dot{u}(t) \\ \vdots \\ u^{(m)}(t) \end{bmatrix} \tag{5}$$

Then, from (5.48),

$$\hat{u}(t+\tau|t) = \left[1, \tau, \frac{\tau^2}{2!}, \cdots, \frac{t^m}{m!}\right] \mathcal{U}(t) \tag{6}$$

and, from (5.50), $\mathcal{Y}(t) = \Psi x(t) + \Theta \mathcal{U}(t)$ for suitable matrices Ψ and Θ. Substituting this into (5.52) gives

$$\begin{aligned}\int_{\tau_1}^{\tau_2} ||r(t+\tau) - \hat{y}(t+\tau|t)||^2 d\tau &= [\Psi x(t) + \Theta \mathcal{U}(t)]^T \Phi(\tau_1, \tau_2)[\Psi x(t) + \Theta \mathcal{U}(t)] \\ &\quad - 2\phi(\tau_1, \tau_2)[\Psi x(t) + \Theta \mathcal{U}(t)] + \int_{\tau_1}^{\tau_2} r(t+\tau)^2 d\tau \\ &= \mathcal{U}(t)^T \Theta^T \Phi(\tau_1, \tau_2) \Theta \mathcal{U}(t) + 2\mathcal{U}(t)^T \Theta^T [\Phi(\tau_1, \tau_2)\Psi x(t) - \phi(\tau_1, \tau_2)^T] \\ &\quad + [x(t)^T \Psi^T \Phi(\tau_1, \tau_2)\Psi x(t) + \int_{\tau_1}^{\tau_2} r(t+\tau)^2 d\tau] \end{aligned} \tag{7}$$

Also,

$$\begin{aligned}\int_{\tau_1}^{\tau_2} ||\hat{u}(t+\tau|t)||^2 d\tau &= \int_{\tau_1}^{\tau_2} || \left[1, \tau, \frac{\tau^2}{2!}, \cdots, \frac{t^m}{m!}\right] \mathcal{U}(t)||^2 d\tau & (8) \\ &= \mathcal{U}(t)^T \Phi_m(\tau_1, \tau_2) \mathcal{U}(t) & (9)\end{aligned}$$

where $\Phi_m(\tau_1, \tau_2)$ is defined as in (5.53) but with n replaced by m. Hence, from (5.46),

$$\begin{aligned} V(t) &= \mathcal{U}(t)^T [\Theta^T \Phi(\tau_1, \tau_2)\Theta + \lambda \Phi_m(\tau_1, \tau_2)] \mathcal{U}(t) \\ &\quad + 2\mathcal{U}(t)^T \Theta^T [\Phi(\tau_1, \tau_2)\Psi x(t) - \phi(\tau_1, \tau_2)^T] \\ &\quad + \left[x(t)^T \Psi^T \Phi(\tau_1, \tau_2)\Psi x(t) + \int_{\tau_1}^{\tau_2} r(t+\tau)^2 d\tau\right] & (10) \\ &= \mathcal{U}(t)^T \mathcal{H} \mathcal{U}(t) - \mathcal{U}(t)^T \mathcal{G} + const & (11) \end{aligned}$$

for suitably defined matrices $\mathcal{G}$ and $\mathcal{H}$.

Chapter 6

Stability

6.1 There are many ways of doing this. Here is a simple way, suitable for *MATLAB* 5.

```
a = [-2, 0 ; 1, 0] % closed-loop A matrix
b = [ 0 ; 0 ] % B matrix (irrelevant)
c = [ 0 ; 1 ] % Output is second state
d = 0

clsys = ss(a,b,c,d,1);


x0 = [ 1 ; 0 ] % Initial state
initial(clsys, x0); % Plots response to initial conditions
```

6.2 (a). Using the model,

$$\begin{aligned}
\hat{x}(k+2|k) &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \hat{x}(k+1|k) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \hat{u}(k+1|k) && (1)\\
&= \begin{bmatrix} \hat{u}(k+1|k) \\ \hat{x}_1(k+1|k) \end{bmatrix} && (2)\\
&= \begin{bmatrix} \hat{u}(k|k) \\ \hat{u}(k|k) \end{bmatrix} && (3)\\
&= \begin{bmatrix} u(k) \\ u(k) \end{bmatrix} && (4)
\end{aligned}$$

since $H_u = 1$. As in Example 6.1, we have $\hat{x}_2(k+1|k) = x_1(k)$. Hence

we have

$$\begin{aligned} V(k) &= \underbrace{u(k)^2 + 6x_1(k) + 4u(k)x_1(k)}_{\text{first step}} \\ &\quad + \underbrace{u(k)^2 + 6u(k)^2 + 4u(k)^2}_{\text{second step}} \qquad (5) \\ &= 12u(k)^2 + 6x_1(k) + 4u(k)x_1(k) \qquad (6) \end{aligned}$$

Hence

$$\begin{aligned} \frac{\partial V(k)}{\partial u(k)} &= 24u(k) + 4x_1(k) \qquad (7) \\ &= 0 \text{ if } u(k) = -\frac{1}{6}x_1(k) \qquad (8) \end{aligned}$$

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

$$x(k+1) = \begin{bmatrix} -\frac{1}{6} & 0 \\ 1 & 0 \end{bmatrix} x(k) \qquad (9)$$

(b). With $H_u = 2$ we can choose both $\hat{u}(k|k)$ and $\hat{u}(k+1|k)$. With no disturbances and a perfect model, we can replace these by $u(k)$ and $u(k+1)$, respectively. Hence we have

$$\hat{x}(k+2|k) = \begin{bmatrix} u(k+1) \\ u(k) \end{bmatrix} \qquad (10)$$

so that

$$\begin{aligned} V(k) &= \underbrace{u(k)^2 + 6x_1(k) + 4u(k)x_1(k)}_{\text{first step}} \\ &\quad + \underbrace{u(k+1)^2 + 6u(k)^2 + 4u(k+1)u(k)}_{\text{second step}} \qquad (11) \\ &= 7u(k)^2 + u(k+1)^2 \\ &\quad + 4u(k)[x_1(k) + u(k+1)] + 6x_1(k) \qquad (12) \end{aligned}$$

and hence

$$\begin{aligned} \frac{\partial V(k)}{\partial u(k)} &= 14u(k) + 4[x_1(k) + u(k+1)] \qquad (13) \\ \frac{\partial V(k)}{\partial u(k+1)} &= 2u(k+1) + 4u(k) \qquad (14) \end{aligned}$$

Setting $\partial V(k)/\partial u(k+1) = 0$ gives $u(k+1) = -2u(k)$ and hence

$$\begin{aligned} \partial V(k)/\partial u(k) &= 14u(k) + 4[x_1(k) - 2u(k)] \\ &= 6u(k) + 4x_1(k) = 0 \qquad (15) \end{aligned}$$

if $u(k) = -(2/3)x_1(k)$. Substituting this back into the model gives the closed-loop equation:

$$x(k+1) = \begin{bmatrix} -\frac{2}{3} & 0 \\ 1 & 0 \end{bmatrix} x(k) \tag{16}$$

(c). Follow the same procedure as in Exercise 6.1, changing the matrix `a` appropriately. (Compare the results with those you get from Exercise 6.3.)

6.3 The solution to this has already been given as the solution to the second part of Exercise 3.7.

6.4 Since $\hat{u}(k+i-1) = 0$ for $i \geq H_u$, we have $\tilde{W}_s\hat{x}(k+i|k) = J_s\tilde{W}_s\hat{x}(k+i-1|k)$ for $i > H_u$, and hence $\tilde{W}_s\hat{x}(k+H_u+i|k) = J_s^i\tilde{W}_s\hat{x}(k+H_u|k)$. Consequently

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

$$\hat{x}(k+H_u|k)^T\tilde{W}_s^T\left[\sum_{i=0}^{\infty}(J_s^T)^i\tilde{W}_s^TC_z^TQC_z\tilde{W}_sJ_s^i\right]\tilde{W}_s\hat{x}(k+H_u|k)$$

$$= \hat{x}(k+H_u|k)^T\bar{Q}\hat{x}(k+H_u|k) \tag{17}$$

if

$$\bar{Q} = \tilde{W}_s^T\Pi\tilde{W}_s \qquad \text{((6.30) in the book)}$$

where

$$\Pi = \sum_{i=0}^{\infty}(J_s^T)^i\tilde{W}_s^TC_z^TQC_z\tilde{W}_sJ_s \tag{18}$$

Pre-multiplying each side of this by J_s^T and post-multiplying by J_s gives

$$J_s^T\Pi J_s = \sum_{i=1}^{\infty}(J_s^T)^i\tilde{W}_s^TC_z^TQC_z\tilde{W}_sJ_s \tag{19}$$

$$= \Pi - \tilde{W}_s^TC_z^TQC_z\tilde{W}_s \tag{20}$$

which is equation (6.31) in the book.

6.5 (a). The plant is stable, so from Section 6.2 we have the equivalent finite-horizon problem:

$$\min_{\Delta\mathcal{U}(k)} \sum_{j=0}^{H_u-1} ||\hat{z}(k+j+1|k) - r(k+j+1)||_{Q(j+1)}^2$$

where $Q(j) = Q$ for $j = 1, \ldots, H_u - 1$, $Q(H_u) = \bar{Q}$, and

$$A^T \bar{Q} A = \bar{Q} - C_z^T Q C_z$$

In this case, since we have $A = 0.9$, $C_z = 1$ and $Q = 1$, we find $\bar{Q} = 1/(1 - 0.9^2) = 5.263$. Since $H_u = 2$, we have

$$\mathcal{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 5.263 \end{bmatrix}$$

(b). The computation and simulation can be done with the *Model Predictive Control Toolbox*, using `ywt=[1 ; sqrt(5.263)]`.

(c). In general for multi-output systems, $\bar{Q}$ is not diagonal, even if each $Q(j)$ is diagonal. (See Example 6.3.) But the *Model Predictive Control Toolbox* assumes that the weighting matrices are always diagonal.

6.6 The pair $(\tilde{A}, \tilde{B})$ being stabilizable is the same as the pair

$$\left(\begin{bmatrix} A & B \\ 0 & I \end{bmatrix}, \begin{bmatrix} B \\ I \end{bmatrix} \right)$$

being stabilizable.

The pair (A, B) is stabilizable if and only if any vector x which is *not* in the range space (column span) of the controllability matrix $P = [B, AB, A^2B, \ldots]$ lies in a subspace spanned by eigenvectors associated with stable eigenvalues of A. (This is standard linear systems theory.)

Similarly the pair $(\tilde{A}, \tilde{B})$ is stabilizable if and only if any vector $\tilde{x}$ which is not in the range space of the controllability matrix $\tilde{P} = [\tilde{B}, \tilde{A}\tilde{B}, \tilde{A}^2\tilde{B}, \ldots]$ lies in a subspace spanned by eigenvectors associated with stable eigenvalues of $\tilde{A}$. But

$$\tilde{P} = \begin{bmatrix} B & AB + B & A^2B + AB + B & \cdots \\ I & I & I & \cdots \end{bmatrix} \tag{21}$$

$$= \begin{bmatrix} B & AB & A^2B & \cdots \\ I & 0 & 0 & \cdots \end{bmatrix} \begin{bmatrix} I & I & I & \cdots \\ 0 & I & I & \cdots \\ 0 & 0 & I & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{22}$$

Note that the last matrix has full rank, so every vector (of suitable dimension) is in its range space. Now suppose that we have a vector $\tilde{x}$ which is not in the range space of $\tilde{P}$, namely it cannot be expressed as $\tilde{P}v$ for any vector v. Partition it as $\tilde{x} = [\tilde{x}_1^T, \tilde{x}_2^T]^T$, where $\tilde{x}_1$ has as many elements as B has rows. $\tilde{x}_2$ is always in the range space of $[I, 0, 0, \ldots]$. So $\tilde{x}$ is not in the range space of $\tilde{P}$ if and only if $\tilde{x}_1$ is not in the range space of P. But if (A, B) is stabilizable then any such $\tilde{x}_1$ must be in the stable invariant subspace of A.

Note that equivalence of controllability for the two representations is easier to establish: (A, B) is controllable if and only if P has full row rank, and it is easy to see from equation (22) above that this happens if and only if $\tilde{P}$ has full row rank.

6.7 Applying (6.36) with $j = N$ gives $K_{N-1} = [0, 1.8824]$. This gives the closed-loop state-transition matrix

$$A - BK_{N-1} = \begin{bmatrix} 0 & 0.1176 \\ 2 & 0 \end{bmatrix}$$

which has eigenvalues ± 0.4851, which both have modulus smaller than 1. Solving (6.49) for $\mathcal{Q}_{N-1}$ gives

$$\mathcal{Q}_{N-1} = \begin{bmatrix} -48 & 0 \\ 0 & 12.2353 \end{bmatrix}$$

which is clearly not positive semi-definite.

Chapter 7

Tuning

7.1 From the definitions (7.6), (7.7),

$$\begin{aligned} S(z)+T(z) &= [I+P(z)K(z)H(z)]^{-1} \\ &\quad + [I+P(z)K(z)H(z)]^{-1}P(z)K(z)H(z) \qquad (1) \\ &= [I+P(z)K(z)H(z)]^{-1}[I+P(z)K(z)H(z)] \qquad (2) \\ &= I \qquad (3) \end{aligned}$$

7.2 From Figure 7.1 we have

$$u = K(z)\{r - H(z)[n+d+P(z)u]\} \qquad (4)$$

Hence

$$[I+K(z)H(z)P(z)]u = K(z)\{r - H(z)[n+d]\} \qquad (5)$$

So the transfer function from the disturbance d to the plant input u is

$$\begin{aligned} -[I+K(z)H(z)P(z)]^{-1}K(z)H(z) &= -\frac{X(z)/B(z)}{1+X(z)/A(z)} \qquad (6) \\ &= -\frac{X(z)A(z)}{B(z)[A(z)+X(z)]} \qquad (7) \end{aligned}$$

So zeros of $B(z)$ are poles of this transfer function. Zeros of $B(z)$ outside the unit circle would lead to internal instability of the feedback system. That is why you cannot use the controller to cancel such zeros.

7.3 (a). As suggested in the question, the easiest way of generating the data for this exercise is to run the *Model Predictive Control Toolbox* demo file `mpctutss`. Variable `pmod` holds the model of the plant. The demo concerns a multivariable control problem with two control inputs and two

controlled outputs. There is an unmeasured disturbance which acts on the two outputs through different transfer functions. (See the *Model Predictive Control Toolbox* User's Guide for details.) Although the original continuous-time transfer functions are simple, they include time delays, which result in a large number of states (14) being required for the discrete-time model.

For mean-level control the prediction horizon H_p should be large (infinite in principle), the control horizon should be $H_u = 1$, and the weight on control moves should be $R = 0$. The following code simulates the set-point response when $H_p = 50$, with equal penalty weights for tracking errors of the two outputs, over a simulation horizon of 40 steps:

```
imod = pmod; % Make model identical to plant
Hp = 50;
Hu = 1;
ywt = [1,1];
uwt = [0,0];
tend = 40;
r = [1,0];  % Step set-point change on output 1
Ks = smpccon(imod,ywt,uwt,Hu,Hp);
[y,u] = smpcsim(pmod,imod,Ks,tend,r);
```

Figure 1 shows the resulting response. Repeating the simulation, but with `r = [0,1]`, namely the set-point change on the second output, gives the response shown in Figure 2.

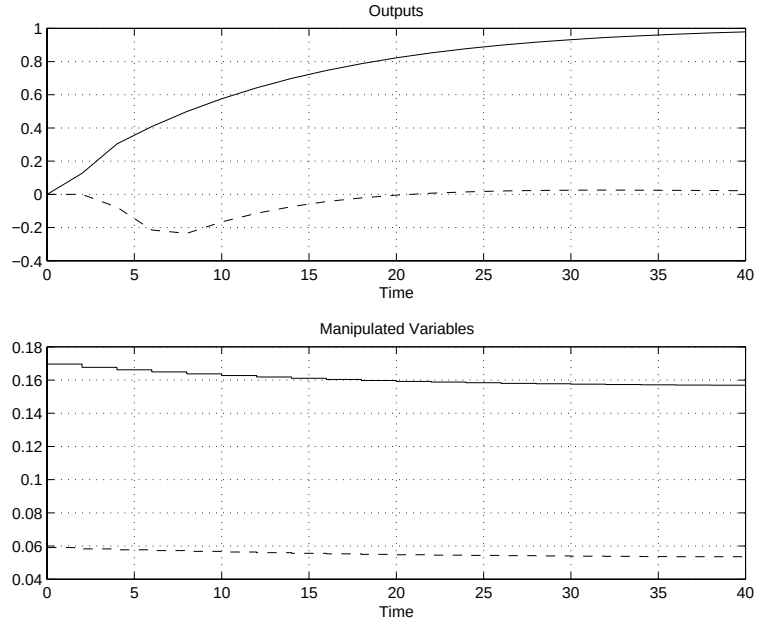

Figure 1: Exercise 7.3(a): mean-level control. Set-point change on output 1. Solid lines: input and output 1. Broken lines: input and output 2.

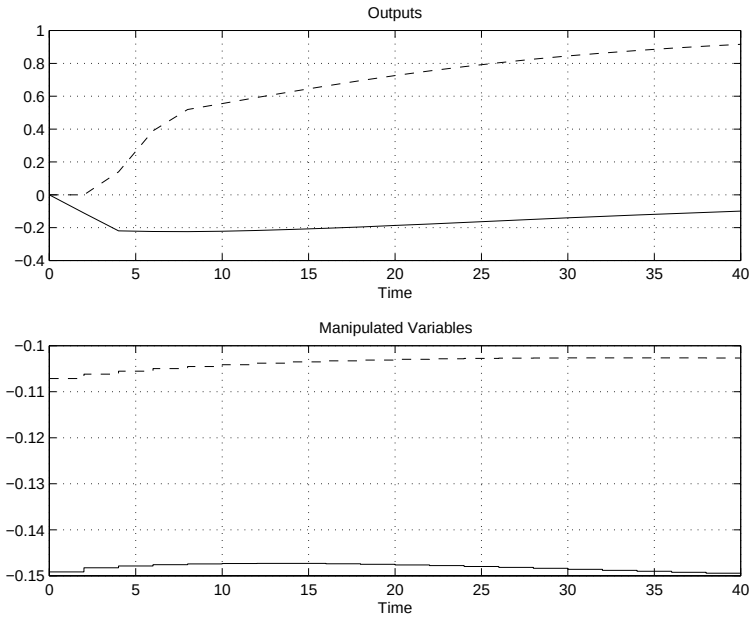

Figure 2: Exercise 7.3(a): mean-level control. Set-point change on output 2. Solid lines: input and output 1. Broken lines: input and output 2.

To see the response to a step disturbance, when both set-points are held at 0:

```
r = [0,0];
d = 1;
[y,u] = smpcsim(pmod,imod,Ks,tend,r,[],[],[],[],d,[]);
```

Figure 3 shows the resulting response.

(b). For 'perfect' control the horizons are $H_p = H_u = 1$. The 'perfect' controller is obtained as follows:

```
Hp = 1; Hu = 1;
ywt = [1 1];
uwt = [0 0];
Ks = smpccon(imod,ywt,uwt,Hu,Hp);
```

The response to a set-point change on output 1 is shown in Figure 4. It is much faster than the response of the mean-level controller. However, the controller does not respond at all to a set-point change on output 2. This is because there is a delay of more than one step between either input and the second output. So it is impossible to influence the second output within the prediction horizon $H_p = 1$, and hence the controller gain for set-point errors on the second output is zero. Increasing the prediction horizon to $H_p = 2$, and making $H_u = H_p$ (to give the controller a chance of hitting the set-point perfectly at both coincidence points), gives the result shown in Figure 5. Note the very oscillatory behaviour of the

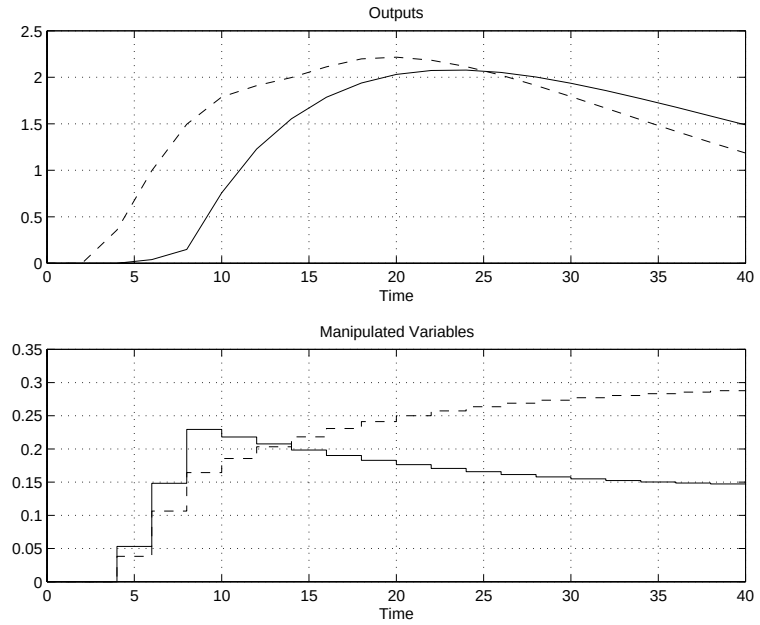

Figure 3: Exercise 7.3(a): mean-level control. Response to step unmeasured disurbance. Solid lines: input and output 1. Broken lines: input and output 2.

inputs, typical of 'perfect' control. The response to a step disturbance is shown in Figure 6. It can be seen that the disturbance is counteracted much more quickly than with mean-level control.

In order to make the controller behave more like the mean-level controller, but without changing the horizons, the main requirement is to reduce the agressiveness of the controller. We expect that this can be done by increasing the penalty on control moves (which is 0 for Figures 4–6). Using `uwt=[10,10]` (which corresponds to $R = 100I$) gives the results shown in Figures 7–9, which look rather similar to the behaviour obtained with the mean-level controller in Figures 1–3, at least as regards the behaviour of the controlled outputs. (The inputs are very different.) Note that the large penalty on input moves has slowed down the transients generally, but also has delayed the elimination of steady-state error, so that the outputs give the appearance of settling to incorrect values — but those would eventually be corrected. The reader may be able to get closer to the mean-level behaviour by further experimentation with R (parameter `uwt` corresponds to $R^{1/2}$).

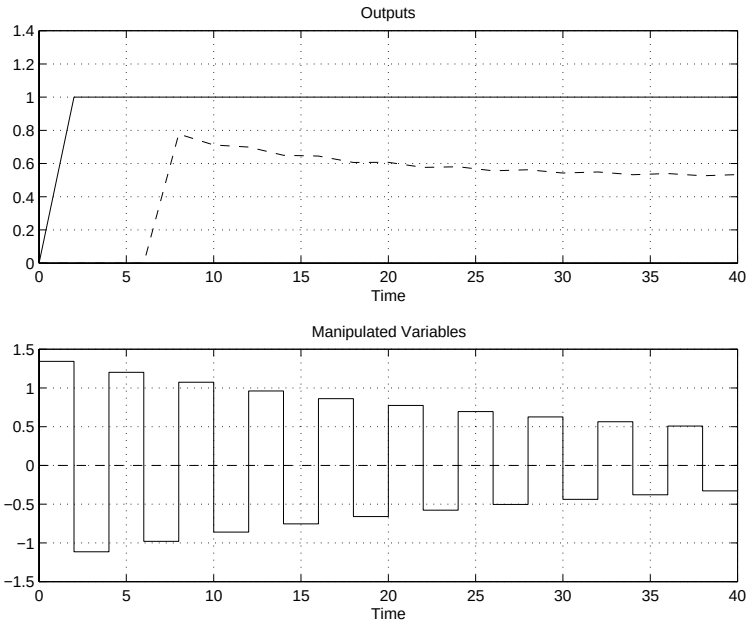

Figure 4: Exercise 7.3(b): 'perfect' control. Response to set-point change on output 1. Solid lines: input and output 1. Broken lines: input and output 2.

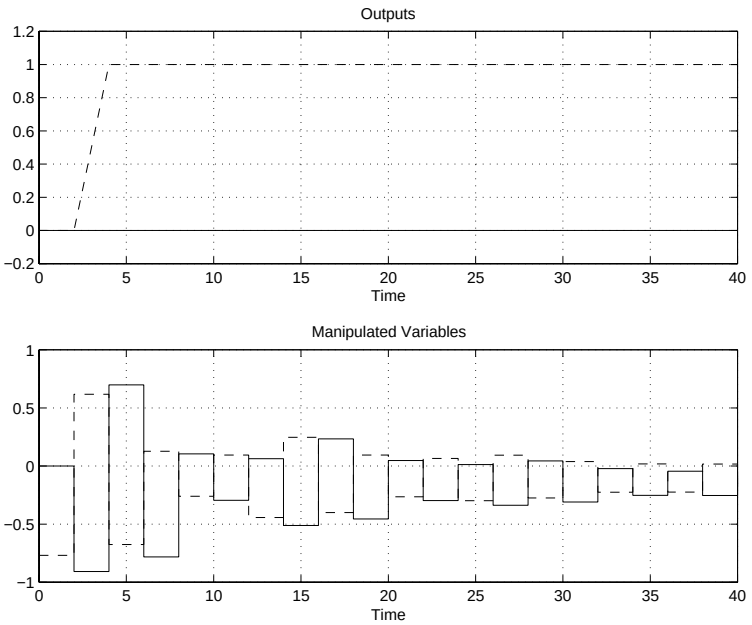

Figure 5: Exercise 7.3(b): 'Perfect' control with $H_p = H_u = 2$. Response to set-point change on output 2. Solid lines: input and output 1. Broken lines: input and output 2.

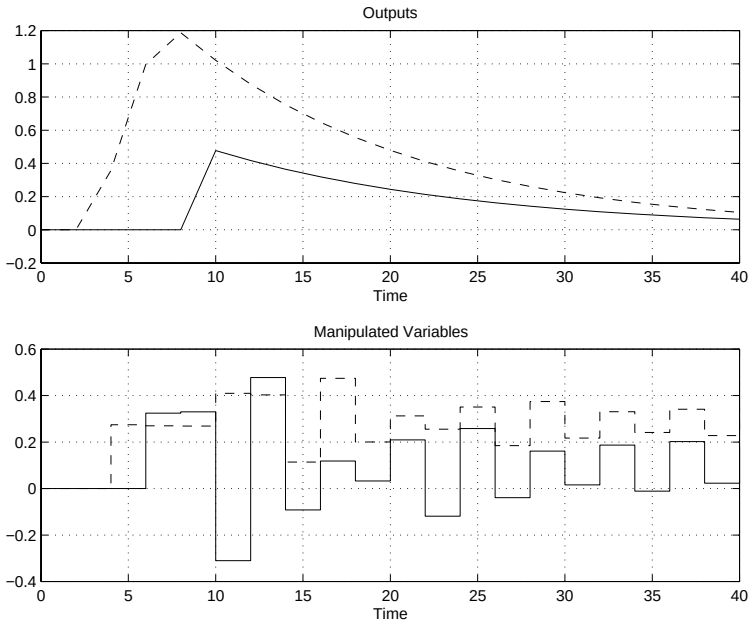

Figure 6: Exercise 7.3(b): 'Perfect' control with $H_p = H_u = 2$. Response to step disturbance. Solid lines: input and output 1. Broken lines: input and output 2.

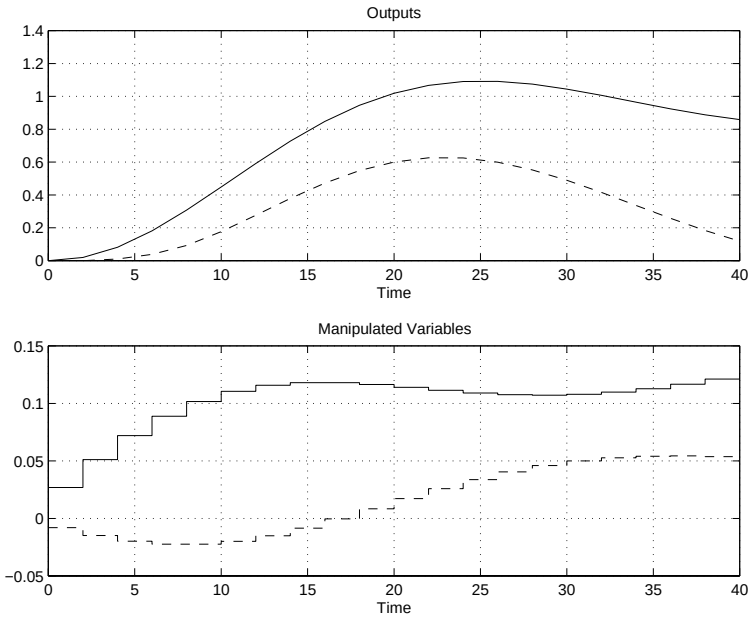

Figure 7: Exercise 7.3(b): $H_p = H_u = 2$, $R = 100I$. Response to set-point change on output 1. Solid lines: input and output 1. Broken lines: input and output 2.

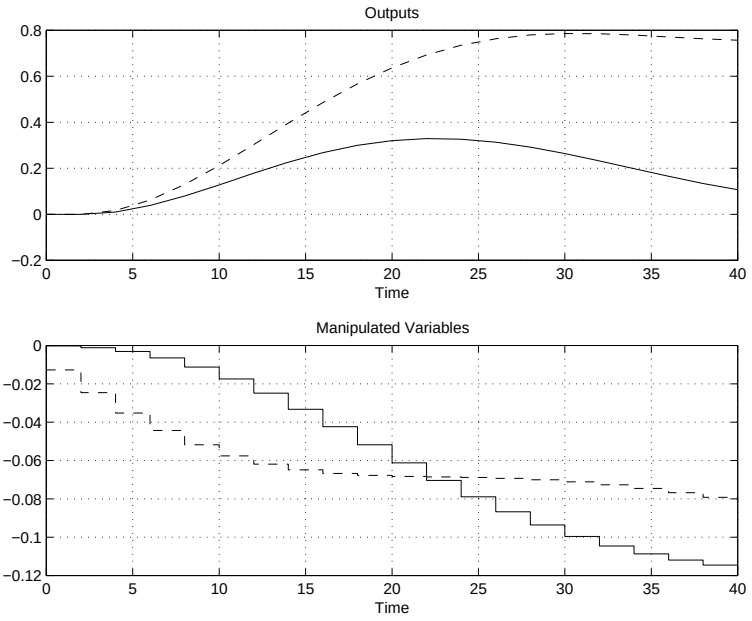

Figure 8: Exercise 7.3(b): $H_p = H_u = 2$, $R = 100I$. Response to set-point change on output 2. Solid lines: input and output 1. Broken lines: input and output 2.

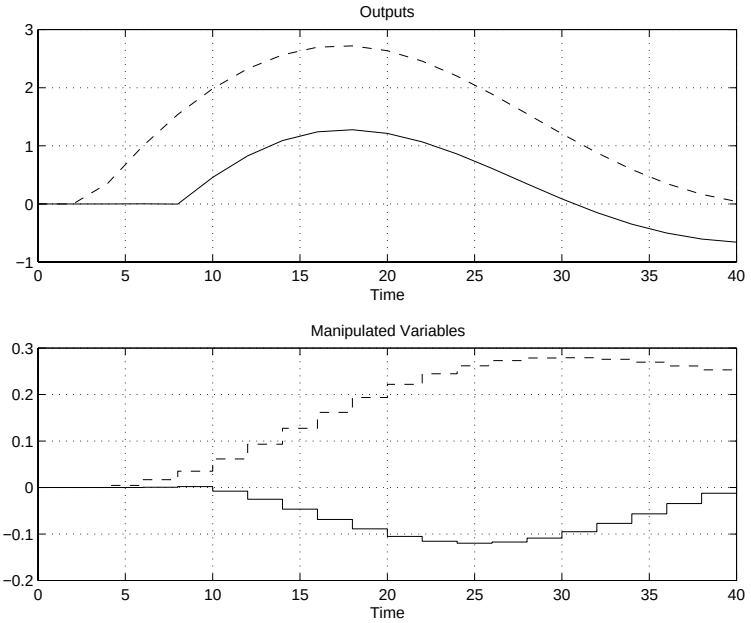

Figure 9: Exercise 7.3(b): $H_p = H_u = 2$, $R = 100I$. Response to step disturbance. Solid lines: input and output 1. Broken lines: input and output 2.

(c). The frequency responses of the sensitivity and complementary sensitivity can be evaluated for each controller by the following *Model Predictive Control Toolbox* commands. First the function `smpccl` is used to compute a state-space model of the complementary sensitivity T. Then the function `mod2frsp` is used to compute the frequency responses of both T and the sensitivity $S = I - T$. The frequencies at which T and S should be computed are determined as follows: in (b) the responses obtained exhibit a time constant of the order of 10 time steps, and $T_s = 2$; if this were a simple first-order lag response, the corner frequency would be of the order of $1/10T_s = 0.05$. So we expect the closed-loop bandwidth to be of this order. Therefore we are interested in seeing the frequency response from at least a factor of 10 lower than this, say $\omega_{min} = 0.005$ rad/sec. The highest frequency which can be seen in a discrete-time system is $\omega_{max} = \pi/T_s = 1.57$ rad/sec. We specify (somewhat arbitrarily) that the frequency response should be computed at 100 frequencies in this range. For function `mod2frsp` the minimum and maximum frequencies must be specified as powers of 10, so the range we want is covered by defining the vector `freq`= [-3,log10(pi/Ts),100];. Finally the singular values of S and T are computed using `svdfrsp`, and the largest one (at each frequency) is plotted.

```
clmod = smpccl(pmod,imod,Ks); % Ks computed previously
                              % by 'smpccon'
[T,S] = mod2frsp(clmod,freq,[1,2],[1,2]); % Select inputs
                                          % and outputs 1 and 2.
[svS,freqlist] = svdfrsp(S);
[svT,freqlist] = svdfrsp(T);
maxsvS = svS(:,1);  % Largest ones are in first column
maxsvT = svT(:,1);
loglog(freqlist,maxsvS,'-')
hold on
loglog(freqlist,maxsvT,'--')
grid, xlabel('Frequency'), ylabel('Singular values')
title('Max SV of S (solid) and T (broken line)')
```

Figure 10 shows the results obtained for the mean-level controller designed in (a), while Figure 11 shows those for the 'perfect' controller designed in (b). The sensitivity function S for the perfect controller has largest singular value of about 2 at frequencies above 0.6 rad/sec, which indicates (usually) unacceptably low robustness to modelling errors. (Typically peak values larger than about $\sqrt{2}$ are considered unacceptable.) The largest singular value of the complementary sensitivity T is constant at 1 at all frequencies. While this is good at low frequencies, it indicates that measurement noise is not attenuated at any frequency, and that the stability of the feedback loop is vulnerable to modelling errors. Figure 10 shows much more acceptable behaviour of the mean-level controller. The sensitivity function peaks at about $\sqrt{2}$, and the complementary function falls

at higher frequencies. Both show good behaviour at low frequencies; the 'integral action' which arises from the default 'DMC' disturbance model shows up as the gain of S falling to zero and the gain of T approaching 1 at low frequencies. The fact that 'perfect' control is more agressive and has faster reaction time appears shows up as a higher bandwidth of the 'perfect' controller, as measured by the frequency at which the gain of the sensitivity S becomes 1. It is also seen that for a given frequency below that bandwidth the gain of S for the 'perfect' controller is lower than that of the mean-level controller at the same frequency.

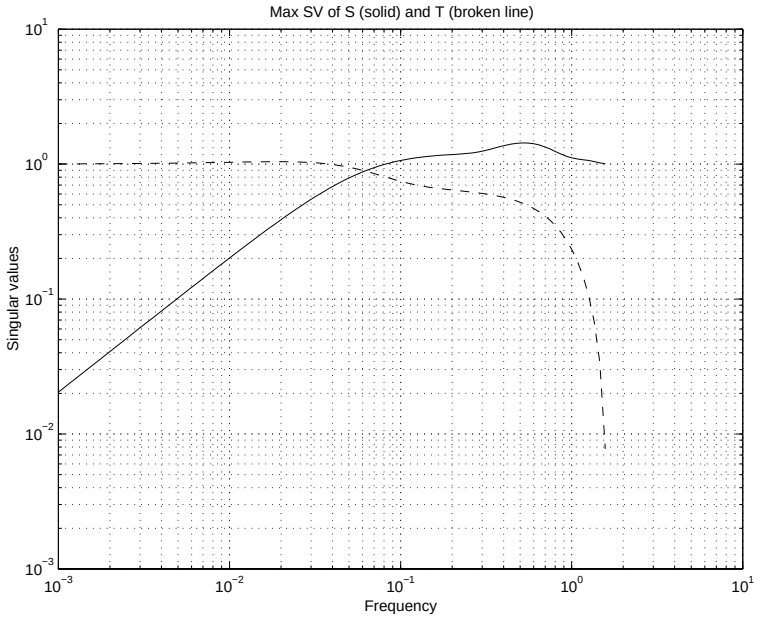

Figure 10: Exercise 7.3(c): mean-level controller.

The same procedure can be followed to investigate the frequency variation of the gains of S and T for any other design.

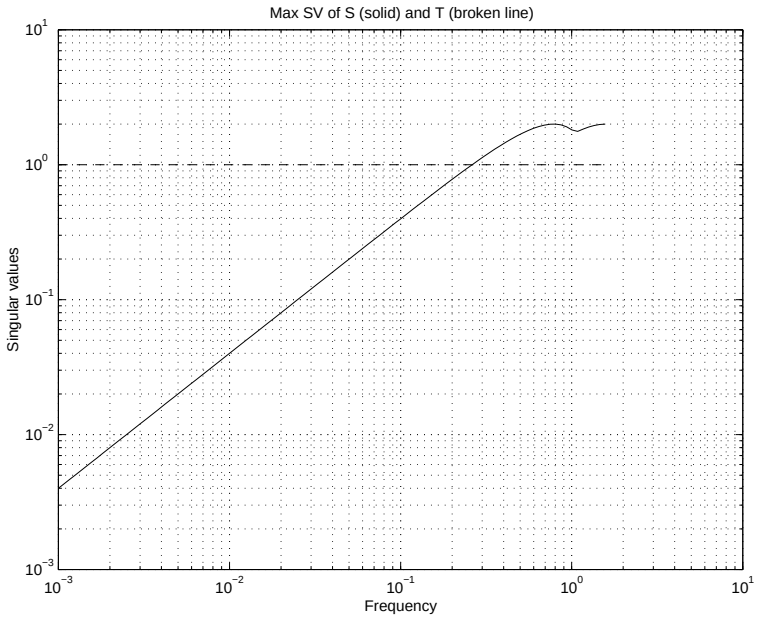

Figure 11: Exercise 7.3(c): 'perfect' controller.

7.4 This question is rather ill-posed, since the precise meaning of 'best' has not been specified. However, the reader can interpret this as referring to the quality of step responses and disturbance rejection in the time domain, or to the sensitivity function in the frequency domain, etc. The simplest thing to do is to judge the quality by the step response and disturbance rejection plots already produced by the *Model Predictive Control Toolbox* demo file `pmlin.m`. Frequency domain properties of the controller can be investigated as in the solution to Exercise 7.3(c).

The `pmlin` demo is discussed at length in the *Model Predictive Control Toolbox* User's Guide, in the section entitled *Application: Paper machine headbox control*.

(a). The first part of the demo file `pmlin.m` takes the user through two designs in which the output weights are changed by changing the parameter `ywt`. The user can edit this file to change the input weights by changing the parameter `uwt`, and the prediction and control horizon parameters `P` and `M`.

(b). The demo file `pmlin.m` also shows the effect of changing from the default observer (estimator) to two alternatives. See the *Model Predictive Control Toolbox* User's Guide for details of these, and Sections 8.2 and 8.3 in the next chapter. But it is possible to experiment with the observer design by editing the file to change the parameters `Q`, `R`, `taus` and `signoise`, without complete understanding of what is going on. Note that making `R` or `signoise` very small should make the control more agressive (the observer becomes approximately 'deadbeat').

7.5 (a). A 'perfect' controller for this example is designed and simulated by the following *MATLAB* commands, using the *Model Predictive Control Toolbox*:

```
plant = tf([-10,1],conv([1,1],[1,30])); % continuous-time
plant = c2d(plant,0.2); % discrete-time
plant = ss(plant); % convert to state-space form
a = get(plant,'a'); b = get(plant,'b'); % get matrices
c = get(plant,'c'); d = get(plant,'d');
pmod = ss2mod(a,b,c,d,0.2); % put plant into MOD format
% Compute 'perfect' controller:
Ksperf = smpccon(pmod,1,0,1,1);
% Simulate step response:
[y,u] = smpcsim(pmod,pmod,Ksperf,150,1);
plotall(y,u,0.2) % Plot results.
```

The simulation shows that the closed loop is unstable. As shown in the text, the 'perfect' controller attempts to invert the plant transfer

function. The (continuous-time) plant in this case has a zero with positive value, at +10, which becomes a zero located outside the unit circle in the discrete-time equivalent model. The 'perfect' controller has a pole at the same location as this zero. This results in an unstable 'hidden mode' in the controller-plant combination (in this case an unobservable unstable mode), which is the source of the instability of the closed loop. The simulated step response shows the plant output tracking the set-point exactly, but the plant input going off to $-\infty$.

(b). As discussed in Chapter 6, closed-loop stability is ensured if an infinite prediction horizon is used (assuming constraints are not active). The infinite-horizon controller is approached more and more closely as the prediction horizon is increased. Thus increasing H_p will eventually result in closed-loop stability.

By trial-and-error I found that closed-loop stability is obtained for $H_p \geq 52$. Using the *Model Predictive Control Toolbox* functions `smpccl` and `smpcpole` the closed-loop poles with $H_p = 52$ were 0.9995, 0.8187, 0.0025, 0.0017 and 0. From the set-point response (or from the 'dominant' pole at 0.9995) the time constant is approximately (treating it as if it were a first-order exponential) 1/0.0005=2000 steps, which corresponds to $2000 \times 0.2 = 400$ sec. Notice that this is very different from 1 sec, which is the time constant expected from the 'mean-level' controller, as $H_p \to \infty$. Values as high as $H_p \approx 1000$ are needed to get close to this speed of response for this plant (with $H_u = 1$ and $R = 0$).

(c). *Error in question: To get stability H_p must be greater than 30. Try $H_p = 60$ for instance.*

The set-point response without constraints is obtained as in part (a) above, making the necessary changes to the parameters. (Note that the correct setting is `uwt = sqrt(0.1)`, since the parameter `uwt` corresponds to $Q^{1/2}$.) To constrain the input moves to $|\Delta u(k)| < 0.1$, use the following *Model Predictive Control Toolbox* commands:

```
ulim = [-inf,inf,0.1];
[y,u] = scmpc(pmod,pmod,1,sqrt(0.1),2,60,200,1,ulim);
plotall(y,u)
```

Note that it is now necessary to use the function `scmpc` in place of the pair of functions `smpccon` and `smpcsim`.

7.6 (a). See the function `swimpool` available on the web site. The only modification needed to run the simulation with the default observer is in the call to function `scmpc`: replace the argument `Kest` by the empty matrix `[]`.

(b). The only modification required here is in the model of the disturbance. A ramp air temperature disturbance has the continuous-time model $\ddot{\theta}_a = 0$, which in state-space form becomes

$$\frac{d}{dt}\begin{bmatrix} \theta_a \\ \dot{\theta}_a \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \theta_a \\ \dot{\theta}_a \end{bmatrix}$$

Thus the only change required in function `swimpool` is to replace the line

```
aairc = [0, 1; -(2*pi/24)^2, 0];
```

by

```
aairc = [0, 1; 0, 0];
```

(c). The sensitivity function can be computed using the *Model Predictive Control Toolbox* in the same way as in Exercise 7.3, once the unconstrained controller has been obtained by using the *Model Predictive Control Toolbox* function `smpccon`. Figure 12 shows the gains of the sensitivity functions for the two disturbance models. (The easiest way to generate this figure is to edit the function `swimpool.m`: add commands similar to those used in Exercise 7.3 at the end of `swimpool`.)

Note: I could not get the Model Predictive Control Toolbox *function* `smpccl` *to work correctly on this example for some reason, so I used* Control System Toolbox *functions instead. Code extracts are shown at the end of this solution.*

The most prominent feature is that when the sinusoidal disturbance model is used, the sensitivity falls sharply to zero at the frequency at which the disturbance occurs ($2\pi/24 = 0.2618$ rad/sec). Note that when the ramp disturbance model is assumed, the sensitivity at this frequency is about -75 dB, which confirms that the ramp disturbance model is also very effective at attenuating the effect of the sinusoidal disturbance.

When the disturbance is assumed to be a sinusoid, a step disturbance is still assumed to be present, by default. The disturbance is thus assumed to have a pole at $+1$, in addition to the two poles at $\exp(\pm 2\pi i/24)$. The gain of the disturbance transfer function thus increases with frequency as ω^{-1} when $\omega \to 0$, and the gain of the sensitivity falls linearly with frequency to counteract this (slope $+1$ on Bode magnitude plot). When the disturbance is assumed to be a ramp, this corresponds to two poles at $+1$; the gain of the disturbance transfer function increases as ω^{-2} at low frequencies. The sensitivity falls cubically with frequency (slope $+3$ on Bode magnitude plot) because the disturbance model now assumes that there are three poles at $+1$, two for the ramp and one for the implicit step disturbance.

Here are extracts of the code used to generate Figure 12:

```
% Define plant:

% Continuous-time:
aplantc = -1/Tplant;  bplantc = [kplant, 1]/Tplant;
cplant = 1;           dplant = [0, 0];
plantc = ss(aplantc,bplantc,cplant,dplant);  % LTI object
```

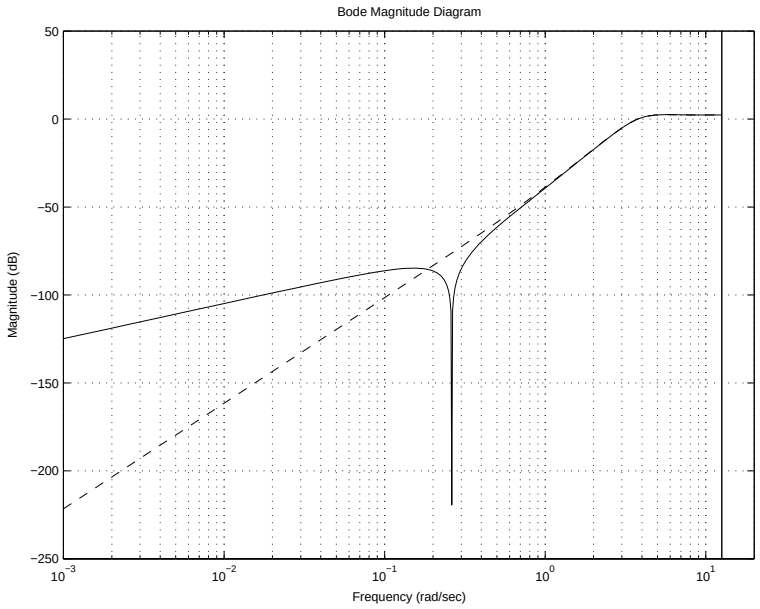

Figure 12: Exercise 7.6(c): sensitivity functions with the sinusoidal (solid line) and the ramp (broken line) disturbance models.

```
% Discrete-time equivalent:
plantd = c2d(plantc,Ts);
[aplantd,bplantd] = ssdata(plantd); % A and B matrices.
                                % (C and D stay unchanged)

plantinfo = [Ts,1,1,0,1,1,0]; % information for MOD format
                % (1 state, SISO, 1 unmeasured disturbance)
plant = ss2mod(aplantd,bplantd,cplant,dplant,plantinfo);
                      % plant in MOD format

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define controller's internal model:

% Disturbance dynamics (diurnal variation):
% Continuous-time:
if disturbance == 'sine',
  aairc = [0, 1; -(2*pi/24)^2, 0]; % Air temp A matrix
elseif disturbance == 'ramp',
  aairc = [0, 1; 0, 0];
end
% Complete continuous-time model, with state vector =
%   [water temp, air temp, air temp derivative]':
amodelc = [-1/Tmodel, 1/Tmodel, 0;
```

```
            zeros(2,1),  aairc    ];
bmodelc = [kmodel/Tmodel; 0; 0];
cmodel = [1, 0, 0];
dmodel = 0;
modelc = ss(amodelc,bmodelc,cmodel,dmodel);  % LTI object

% Discrete-time equivalent:
modeld = c2d(modelc,Ts);
[amodeld,bmodeld] = ssdata(modeld); % A and B matrices.
                              % (C and D stay unchanged)

modinfo = [Ts,3,1,0,0,1,0];
model = ss2mod(amodeld,bmodeld,cmodel,dmodel,modinfo);
                                     % model in MOD format

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute observer gain which gives
% asymptotically stable observer:

% First need slightly different model, with
% artificial process noise:
% Use same 'A' matrix, different 'B' matrix.
bobs = [bmodeld, [0;0;1]];
obsinfo = [Ts,3,1,0,1,1,0]; % minfo vector for MOD format
% Model for observer:
modobs = ss2mod(amodeld,bobs,cmodel,[0,0],obsinfo);

% Now compute observer gain:
Kest = smpcest(modobs,W,V);

% Compute and display observer poles:
% Augment model:
[auga,augb,augc]=mpcaugss(amodeld,bobs,cmodel);
obspoles = eig(auga*(eye(4)-Kest*augc)); % pole computation
disp('   Observer poles are:'), disp(obspoles)
if any(abs(obspoles)>=1),
  disp('Warning: Observer not asymptotically stable')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute MPC controller Ks:
Ks = smpccon(model,Q,R,Hu,Hp);
% Compute closed loop system and controller in MOD format:
[clmod,cmod] = smpccl(plant,model,Ks,Kest);
```

```
% Compute T and S frequency responses:
% Next 2 lines don't work correctly - don't know why
%[T,S] = mod2frsp(clmod,[-3,log10(pi/Ts),50],1,1);
%[svS,omega] = svdfrsp(S);
% So use Control Toolbox instead:
plantlti = ss(aplantd,bplantd,cplant,dplant,Ts);
% Make controller as LTI object:
[ka,kb,kc,kd] = mod2ss(cmod);
K = ss(ka,kb,kc,kd,Ts);
% Make T and S systems:
GK = plantlti(1,1)*K(1,1);
T = feedback(GK,1);  % Complementary sensitivity
S = 1-T;          % Sensitivity (display using 'bodemag')
```

7.7 (a). If $\Lambda = 0$ then (7.36) becomes

$$\mathcal{T}(k) = \begin{bmatrix} Iz \\ Iz^2 \\ \vdots \\ Iz^{H_p} \end{bmatrix} s(k) = \begin{bmatrix} s(k+1) \\ s(k+2) \\ \vdots \\ s(k+H_p) \end{bmatrix} \tag{8}$$

(b). Suppose that the reference trajectory is specified as

$$r(k+j|k) = s(k+j) - \frac{e^{\alpha j}}{2}[s(k+j) - z(k)] - \frac{e^{\beta j}}{2}[s(k+j) - z(k)]$$

Note that a simplification here, compared with the development in Section 7.5 of the book, is that we assume the same constants α and β apply to each component of the reference trajectory (if there is more than one output).

Assuming that future set-point changes are not known, the reference trajectory can be rewritten in the same form as (7.32):

$$r(k+j|k) = \left(1 - \frac{e^{\alpha j} + e^{\beta j}}{2}\right) s(k) + \frac{e^{\alpha j} + e^{\beta j}}{2} z(k) \tag{9}$$

and hence (7.33) becomes

$$\mathcal{T}(k) = \begin{bmatrix} \left(1 - \frac{e^{\alpha H_w} + e^{\beta H_w}}{2}\right) I \\ \left(1 - \frac{e^{\alpha (H_w+1)} + e^{\beta (H_w+1)}}{2}\right) I \\ \vdots \\ \left(1 - \frac{e^{\alpha H_p} + e^{\beta H_p}}{2}\right) I \end{bmatrix} s(k) + \begin{bmatrix} \frac{e^{\alpha H_w} + e^{\beta H_w}}{2} I \\ \frac{e^{\alpha (H_w+1)} + e^{\beta (H_w+1)}}{2} I \\ \vdots \\ \frac{e^{\alpha H_p} + e^{\beta H_p}}{2} I \end{bmatrix} z(k) \tag{10}$$

If the controller is aware of future set-point changes then, taking $H_w = 1$ for simplicity, (7.35) becomes

$$\mathcal{T}(k) = \begin{bmatrix} \left(1 - \frac{e^\alpha + e^\beta}{2}\right) s(k+1) \\ \left(1 - \frac{e^\alpha + e^\beta}{2}\right) s(k+2) \\ \vdots \\ \left(1 - \frac{e^\alpha + e^\beta}{2}\right) s(k+H_p) \end{bmatrix} + \begin{bmatrix} \frac{e^\alpha + e^\beta}{2} r(k|k) \\ \frac{e^\alpha + e^\beta}{2} r(k+1|k) \\ \vdots \\ \frac{e^\alpha + e^\beta}{2} r(k+H_p-1|k) \end{bmatrix} \quad (11)$$

and hence (7.36) becomes

$$\begin{aligned} \mathcal{T}(k) \;=\; & \begin{bmatrix} \left(1 - \frac{e^\alpha + e^\beta}{2}\right) Iz \\ \left(1 - \frac{e^\alpha + e^\beta}{2}\right) Iz^2 \\ \vdots \\ \left(1 - \frac{e^\alpha + e^\beta}{2}\right) Iz^{H_p} \end{bmatrix} s(k) + \begin{bmatrix} \frac{e^\alpha + e^\beta}{2} I \\ 0 \\ \vdots \\ 0 \end{bmatrix} z(k) \\ & + \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ \frac{e^\alpha + e^\beta}{2} I & 0 & \cdots & 0 & 0 \\ 0 & \frac{e^\alpha + e^\beta}{2} I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{e^\alpha + e^\beta}{2} I & 0 \end{bmatrix} \mathcal{T}(k) \end{aligned} \quad (12)$$

The equations corresponding to (7.37) and (7.38) now follow by making the obvious changes.

Chapter 8

Robust predictive control

8.1 The controller in feedback around the actual plant is shown in Figure 1. This can be redrawn as in Figure 2. The transfer function matrix of the block enclosed within the broken line is $P_0(z)K(z)[I + P_0(z)K(z)]^{-1}$, which is the complementary sensitivity $T(z)$. Thus the loop gain is $T(z)\Delta$, and hence a sufficient condition for the feedback loop to remain stable is $\bar{\sigma}[T(e^{j\omega T_s})]\|\Delta\|_\infty < 1$.

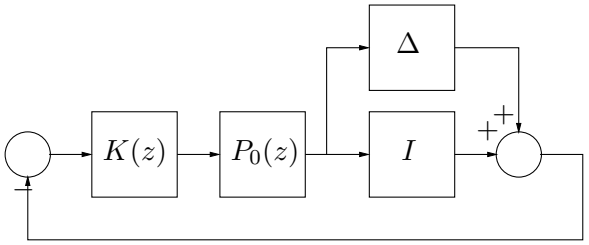

Figure 1: Exercise 8.1: plant with output multiplicative uncertainty, and feedback controller.

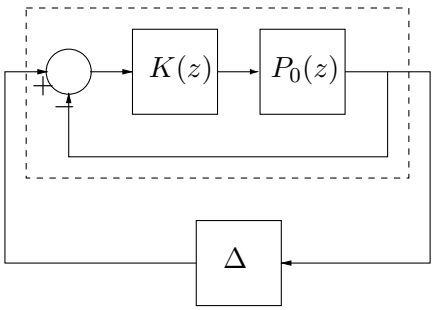

Figure 2: Exercise 8.1: previous figure redrawn.

8.2 The observer's state-transition matrix is

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A_w & 0 \\ CA & A_w & I \end{bmatrix} - \begin{bmatrix} 0 \\ L_{\Delta w} \\ L_\eta \end{bmatrix} [0 \quad 0 \quad I] = \begin{bmatrix} A & 0 & 0 \\ 0 & A_w & -L_{\Delta w} \\ CA & A_w & I - L_\eta \end{bmatrix} \tag{1}$$

This has a block-triangular structure, so its eigenvalues are the eigenvalues of A together with those of

$$\begin{bmatrix} A_w & -L_{\Delta w} \\ A_w & I - L_\eta \end{bmatrix} = \begin{bmatrix} \text{diag}\{\alpha_i\} & -\text{diag}\{\phi_i\} \\ \text{diag}\{\alpha_i\} & \text{diag}\{1 - \psi_i\} \end{bmatrix} \tag{2}$$

8.3 The observer gain is determined by (8.28). But in the model (8.41)–(8.42) C is replaced by $[0 \quad 0 \quad I]$, so with $V = 0$ we have

$$L_\infty = \begin{bmatrix} L_{\Delta x} \\ L_{\Delta w} \\ L_\eta \end{bmatrix} = P_\infty \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix} \left([0 \quad 0 \quad I] P_\infty \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix} \right)^{-1} = \begin{bmatrix} P_{13} \\ P_{23} \\ P_{33} \end{bmatrix} P_{33}^{-1} \tag{3}$$

and hence $L_\eta = I$.

8.4 If $F(v)$ is symmetric and affine in v then we can write

$$F(v) = F_0 + \sum_{i=1}^{n} F_i(v_i) \tag{4}$$

where each F_i is a symmetric matrix which depends only on v_i, the ith element of v (since no products such as $v_i v_j$, or more complicated functions of v, occur in $F(v)$). Furthermore, by putting all terms independent of v into F_0, we have $F_i(v_i) = v_i F_i$, where each F_i is a symmetric constant matrix.

8.5 (a). Applying (8.60) and (8.61) to the first matrix inequality in (8.65), we see that this inequality is equivalent to $\gamma - r - q^T x - x^T Q x \geq 0$ (since $I > 0$). The second matrix inequality in (8.65) is clearly equivalent to $Ax - b \geq 0$. So the problem (8.65) is equivalent to the problem (8.64), which is clearly equivalent to the original QP problem (8.63).

(b). The variables over which the optimization in (8.65) is to be performed are γ and x. The cost to be minimized is $\gamma = 1\gamma + 0x$, which is in the form of the cost function in (8.59). The two matrix inequalities which appear in (8.65) involve symmetric matrices which are affine in the variables γ

and x. So, by Exercise 8.4, they are *LMI*s. But we can write these two *LMI*s as the single *LMI*:

$$\begin{bmatrix} I & Q^{1/2}x & 0 & \cdots & 0 \\ x^TQ^{1/2} & \gamma - r - q^Tx & 0 & \cdots & 0 \\ 0 & 0 & (Ax-b)_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (Ax-b)_m \end{bmatrix} \geq 0 \quad (5)$$

Thus problem (8.65) is indeed in the form of problem (8.59).

8.6 Clearly the cost function is already in the required form. The vector inequality can be written as a matrix inequality, as in (8.65). But this is affine in x and symmetric, so it is an *LMI*, by Exercise 8.4.

8.7 From the *LMI* (8.85) we get the following inequality using (8.62):

$$z_i^2 - \rho^2 C_i[A(k+j)Q + B(k+j)Y]Q^{-1}[A(k+j)Q + B(k+j)Y]^TC_i^T \geq 0 \quad (6)$$

which is the same as

$$z_i^2 - \rho^2 C_i[A(k+j) + B(k+j)YQ^{-1}]Q[A(k+j) + B(k+j)YQ^{-1}]^TC_i^T \geq 0 \quad (7)$$

from which we get, since $K_k = YQ^{-1}$,

$$z_i^2 - \rho^2 ||C_i[A(k+j) + B(k+j)K_k]Q^{1/2}||_2^2 \geq 0 \quad (8)$$

which is the same as (8.84).

8.8 Recall that feasibility is obtained at time k if the LMIs (8.67), (8.72), (8.79) and (8.86) hold. As stated in the question, only the first of these depends explicitly on $x(k|k)$. Let Q_k denote the value of Q obtained as the solution obtained by minimising γ subject to these LMI constraints at time k, so that

$$\begin{bmatrix} Q_k & x(k|k) \\ x(k|k)^T & 1 \end{bmatrix} \geq 0 \quad (9)$$

and hence $x(k|k) \in \mathcal{E}_{V(x(k|k))}$. But, as argued in the text, this is an invariant set for the predicted state $\hat{x}(k+j|k)$, namely $\hat{x}(k+j|k) \in \mathcal{E}_{V(x(k|k))}$ for all j, if $\hat{u}(k+j|k) = K_k\hat{x}(k+j|k)$. But if no disturbances occur then the actual state trajectory will also remain in this set, so that $x(k+1|k+1) \in \mathcal{E}_{V(x(k|k))}$, or

$$\begin{bmatrix} Q_k & x(k+1|k+1) \\ x(k+1|k+1)^T & 1 \end{bmatrix} \geq 0 \quad (10)$$

This means that there will exist at least one Q for which (8.67) will hold at time $k+1$, so the optimization problem to be solved at time $k+1$ will be feasible.

8.9 Let $\mathcal{O}_\infty$ be the maximal output admissible set for $x(k+1) = Fx(k)$, with the constraint $y(k) = Hx(k) \in Y$. Suppose that $x(0) \in \mathcal{O}_\infty$, then $y(k) = Hx(k) = HF^k x(0) \in Y$ for all $k > 0$. Now suppose that $x(k) \notin \mathcal{O}_\infty$ for some k. Then, by the definition of $\mathcal{O}_\infty$ (and time-invariance), $y(k+j) \notin Y$ for some $j > 0$. But this contradicts the earlier statement that $y(k) \in Y$ for all $k > 0$. Hence $x(k) \in \mathcal{O}_\infty$ for all $k > 0$, and so $\mathcal{O}_\infty$ is positively invariant.

Chapter 9

Two case studies

Solutions are not provided for the exercises in this chapter. As stated in the book, all of these exercises are open-ended, without unique correct answers. They can all be solved by using and/or editing *MATLAB* and *Simulink* software provided on the book's web site.

Note on Exercise 9.2: This is worded rather carelessly. It is the singular values of the sensitivity and of the complementary sensitivity functions that should be investigated, not the singular values of the controller itself.