Continuous-Time Self-Tuning Control
Volume II – Implementation

Peter Gawthrop

Originally published in 1990

doi:10.5281/zenodo.7004513

Preface

Volume 1 of this monograph discussed and described the *design* of continuous-time self-tuning controllers; this volume focuses on *implementation* issues. This emphasis on implementation is particularly important in the context of the continuous-time approach: digital implementation of continuous-time algorithms is less obvious than that of discrete-time algorithms. Thus a purpose of this volume is to convince a reader of volume 1 that continuous-time algorithms can be implemented in a digital form.

This volume is designed to be read in conjunction with volume 1. Corresponding section numbers are indicated where appropriate; thus

1.1.1. [1.2] Transfer Functions

implies that section 1.1.1 of this volume must be read in conjunction with section 1.2 of volume 1 with the same name. In addition, references to equations and sections in volume 1 are prefaced by 'I'.

On the suggestion of the Series Editor, Professor C.R. Burrows, a computer program **CSTC** has been developed to accompany this text. The text itself contains the full program together with numerous illustrative examples of its use. In addition, the software is available separately (for use only with this book) on an IBM type disc for use on and IBM PC or compatible. The Pascal source code is provided for those who would rather recompile and run the software on a different computer. The software is designed not only to simulate the illustrative examples that I have created but also to help readers create their own examples. It is my hope that this will help those starting to do research in this area to rapidly pass through the learning phase and on to new research ideas and results.

Access to a computer and the software is not essential; but interaction with the software will enrich the appreciation of the contents of the book.

Development of usable software is not an easy task. I must give special thanks to T.C. Tsang of Oxford University and A. Plummer of the University of Bath for evaluating the software and making numerous helpful suggestions for its improvement.

The ideas embodied in the software arise from those in Volume 1, and once again I would like to acknowledge the help given by those listed in the Preface to volume 1.

I hope that this experiment in enhancing the written word with computer software will prove to be a fruitful way of disseminating continuous-time self-tuning control.

University of Glasgow
Gilmorehill
August 1989

P.J. Gawthrop

Contents

CHAPTER 0. INTRODUCTION

CHAPTER 3. EMULATOR-BASED CONTROL

CHAPTER 4. NON-ADAPTIVE ROBUSTNESS

CHAPTER 5. LEAST-SQUARES IDENTIFICATION

CHAPTER 6. SELF-TUNING CONTROL

CHAPTER 10. TWO-INPUT TWO-OUTPUT SYSTEMS

CHAPTER 11. CSTC: THE PROGRAMME

CHAPTER 0
Introduction

0.1. INTRODUCTION

The traditional means of conveying the results of a research project to others (whether in academe or industry) is via the printed page; volume I of this work is an example of this process. However, this by no means conveys all of the efforts and achievements of the project. If the reader of a book wishes to implement the the algorithms described there, he has a long and daunting task ahead of him: he will repeat mistakes and resolve problems. Even worse, he has no way of knowing if he has provided a correct implementation of the author's ideas. Moreover, such a reader may be frustrated by the illustrative examples provided: he may ask "But what if the parameters of the simulation were changed?", but will have no direct means of finding an answer.

Many of us are now used to complementing our bookshelves of printed material with personal computers. This provides the opportunity of conveying the intellectual capital tied up in software to others, and so overcoming the problems inherent in the printed page. In the context of this volume, I perceive two advantages arising from the use of this new medium of communication: the book becomes a computer-illustrated text, and the algorithms implementing the ideas of volume I are precisely described in executable code.

As a computer-illustrated text, the reader can use the text examples as *starting points* for investigation of the properties of continuous-time self-tuning control; both advantages and disadvantages are there to be examined and discovered. A wide range of such starting points has been provided; at the risk of some repetition, the examples have been made largely independent of each other: they do not have to be examined serially.

0.1.1. ORGANISATION OF THE BOOK

The book is organised to reflect the chapter arrangement of volume I. Thus chapter 1 of this volume contains material corresponding to chapter 1 of volume 1 and has the same title. Starting at chapter 1, each chapter has two major sections: a section on *implementation* followed by a section containing *examples.*

Implementation

The implementation section provides a guide to the way in which the algorithms have been implemented in Pascal. It should be read in conjunction with the programme (listed at the end of the book) together with volume I. The connection to the programme listing can be made via the *cross-reference listing* following the programme. The connection to volume I is indicated by the section number in square parentheses after the appropriate section number. Thus section 1.1.1 [1.4] refers to section 1.4 of volume I. Alternatively, when scanning the programme listing itself, reference can be made to the textual descriptions of individual functions, procedures and variables by means of the *procedure and variable* index at the end of the book. These procedure and variable names are emphasised by the use of boldface in the text.

Examples

The examples section provides worked examples which turn volume I into a computer-illustrated text. The aim is to allow the reader to simulate examples arising from volume I. The examples do not have to be approached in numerical order; at the expense of some repetition, the examples have been made largely independent of each other although appropriate cross references are made.

Each example is further subdivided into five subsections as follows.

Reference

This subsection refers to the appropriate section of volume I, and allows the examples of volume II to be executed in conjunction with the relevant sections of volume I.

Description

This subsection sets the background to the example and explains its significance.

Programme interaction

A copy of the screen output generated by **CSTC** is given for comparison with the output from the modified examples which the reader is encouraged to investigate.

Discussion

The result of the simulation as displayed in the corresponding figure (and, hopefully, on the reader's computer display) is discussed in connection with volume I.

Further investigations.

The user is not constrained to use the given examples as they stand, but rather is encouraged to use them as a starting point for further investigations. Some ideas are given in this subsection.

0.2. USER GUIDE

0.2.1. PC CONFIGURATION

1. To run the software you will need an IBM PC, or compatible, with either 3.5" or 5.25" discs and a maths coprocessor.
2. To display the results you will need a graphics package capable of displaying columns of data from an ASCII file. The recommended package is PC-MATLAB.

0.2.2. INSTALLING THE PROGRAMME ON A PC

1. Boot your PC.
2. Create a directory called CSTC:

   md cstc

3. Move to directory CSTC:

   cd cstc

4. Insert the distribution diskette into drive A: of your PC.
5. Install the software by typing:

   a:install

0.2.3. RUNNING THE PROGRAMME ON A PC

1. Decide which example that you wish to run (say example 3 of chapter 6) and execute it by typing 'runex' followed by the chapter followed by the example number followed by the return key.

   runex 6 3

2. The computer should reply with the example name followed by the **CSTC** heading, for example:

   Using a setpoint filter
   ======== C S T C Version 5.3 ========
   Enter all variables (y/n)?

   Continued pressing of return will step though the programme input phase, leaving the defaults as displayed. These may be changed - see the next section. The output should correspond to that given in the appropriate section.

3. Plot the results using your favourite graphics package with the facility for plotting ASCII data files (columns of numbers). If you are lucky enough to have MATLAB, then the distributed .m files will help. For example, plot6.m plots data pertaining to the examples of chapter 6. If you don't have MATLAB, then you will have to read section 5 which describes the file format. (In this case, you can run CSTC from MATLAB using !run or !runex commands).

4. If you wish to rerun the same example, retaining whatever changes that you made, type

   run

0.2.4. USER INTERACTION

The programme, CSTC, is equipped with a simple, but quite powerful, user interface. The user is presented with a **variable name**, its **current value** and a **prompt** asking for a new value. For example, the real variable 'Sample interval' may be presented as:

Sample Interval = 0.050000 :=

The user may then type in a new value (and then press the 'return' key) or retain the default (0.05) by pressing the 'return' key. There are four types of variables:

1. **Integers**
2. **Reals**
3. **Booleans**
4. **Polynomials**

Integers are entered in the usual format. For example

Approximation Order = 5 :=
2

changes the integer variable 'Approximation Order' from 5 to 2.

Reals are also entered in the usual format. Each of the following examples changes the real variable 'Sample Interval' to 0.1.

Sample Interval = 0.500000 :=
0.1

Sample Interval = 0.500000 :=
1e-1

Booleans can be either 'TRUE' or 'FALSE'. The boolean variable 'Constant between samples' can be set to false by any of the following:

Constant between samples = TRUE :=
FALSE

Constant between samples = TRUE :=
F

Constant between samples = TRUE :=
f

Polynomials are entered in terms of their coefficients, highest order first. Thus the polynomial variable $A(s) = s^2 + 1$ can be changed to the value $A(s) = s^2 + 3s + 2$ by either of the following:

A (system denominator) = 1.000000 0.000000 1.000000 :=
1 3 2

A (system denominator) = 1.000000 0.000000 1.000000 :=
1.0 3.0 2.0

Alternatively, polynomials can be entered in factored form. If a polynomial entry is terminated by a '*', then another factor can be entered on the next line. Thus the polynomial

$$A(s) = s^3 + 3s^2 + 3s + 1 \qquad (0.2.4.1)$$

$$= (s+1)(s+1)(s+1)$$

$$= (s^2 + 2s + 1)(s+1)$$

can be changed from $A(s) = s^2 + 1$ by any of the following:

A (system denominator) = 1.000000 0.000000 1.000000 :=
1 3 3 1

A (system denominator) = 1.000000 0.000000 1.000000 :=
1 1*
1 1*
1 1

A (system denominator) = 1.000000 0.000000 1.000000 :=
1 2 1*
1 1

WARNING. According to the usual Pascal conventions, entering **0.1** as **.1** will lead to a run time error. **ALWAYS** prefix a decimal point with a digit, even if it is **0**.

0.2.5. HIDDEN VARIABLES

There are a lot of variables associated with this programme, so it is convenient to hide them from the user. The programme starts by asking

Enter all variables (y/n)?

the default response is no, and most variables are then hidden from the user.

On the other hand, a yes answer reveals all variables to the user. In addition, the programme then marks all variables which are changed by the user (or unchanged by default by typing a space). All other variables are hidden when a subsequent 'run' is invoked.

0.2.6. INPUT FILES

There are two input files to this programme:

1. inlog.dat
2. indata.dat

Inlog.dat

Inlog.dat contains a set of default parameters and is automatically copied into the working directory by the 'runex' command. The programme checks the variable names against what it expects to find; discrepancies lead to a built in default being used, and the corresponding variable is not hidden.

For example, the inlog.dat file corresponding to example 3 of chapter 6 is:

```
 Chapter                     6
======== Data Source     ========
External data          FALSE
Last time               25.000000
Print interval                1
======== Filters          ========
Sample Interval           0.050000
Approximation Order            5
Continuous-time?       TRUE
======== Control action  ========
Automatic controller mode TRUE
Integral action        FALSE
======== Assumed system  ========
#A (system denominator)      1.000000    0.000000    0.000000
#B (system numerator)        1.000000    1.000000
 Number of interactions         0
 D (initial conditions)     0.000000    0.000000
 Time delay                 0.000000
======== Emulator design ========
 Z has factor B         FALSE
 Z-+ (Z- not including B)    1.000000
 Z+ (nice model numerator)    1.000000
 Linear-quadratic poles   FALSE
#P (model denominator)       0.500000    1.000000
#C (emulator denominator)    0.500000    1.000000
```

```
 Pade approximation order          0
 Small positive number        0.000100
 ======== STC type        ========
 Explicit self-tuning      TRUE
 Using lambda filter       FALSE
 Identifying system        TRUE
#Tuning initial conditions FALSE
 ======== Identification  ========
#Initial Variance         100000.000000
#Forget time              1000.000000
 Dead band                  0.000000
 Estimator on             TRUE
 Tune interval                   1
#Cs (emulator denominator)    1.000000    2.000000    1.000000
 Identifying rational part TRUE
 Identifying delay        FALSE
 ======== Controller      ========
 Q numerator                0.000000
 Q denominator              1.000000
#R numerator                0.500000    1.000000
#R denominator              1.000000    1.414000    1.000000
#Maximum control signal     100.000000
#Minimum control signal     -100.000000
 Switched control signal   FALSE
 ======== Simulation      ========
 ======== Setpoint        ========
 Step amplitude             50.000000
 Square amplitude           25.000000
 Cos amplitude              0.000000
 Period                    20.000000
 ======== In Disturbance  ========
 Step amplitude             0.000000
 Square amplitude           0.000000
 Cos amplitude              0.000000
 Period                    20.000000
 ======== Out Disturbance ========
 Step amplitude             0.000000
 Square amplitude           0.000000
 Cos amplitude              0.000000
 Period                    20.000000
 ======== Actual system   ========
#A (system denominator)       1.000000    1.000000    0.000000
#B (system numerator)         1.000000    0.100000
 D (initial conditions)     0.000000    0.000000
 Time delay                 0.000000
 Number of lags                 0
 More time                FALSE
```

The leftmost column contains a '#' for each non-hidden variable, other variables are **not** displayed by default.

Indata.dat

Indata.dat allows external data to be read into the programme; for example, if some real system is to be identified. The columns of the file must be arranged as in the following table:

Column	Variable	Symbol
1	Time	t
2	System input	$u(t)$
3	System output	$y(t)$

Additional columns may be added for multi-input systems. Surplus columns are ignored; in particular, **outdata.dat** files can be copied and used as **indata.dat** files.

Blank rows initiate data splicing*. (See example 5.2.8)

0.2.7. OUTPUT FILES

There are three output files:

1. outdata.dat
2. outsyspar.dat
3. outempar.dat

Outdata.dat

Outdata.dat contains signals arising from the simulation. For single-input single-output systems, the columns of this file are as follows:

* Gawthrop, P.J. (1984) "Parameter identification from non-contiguous data", Proceedings IEE, vol. 131 pt. D, No. 6, pp261-265.

Column	Variable	Symbol
1	Time	t
2	System input	$u(t)$
3	System output	$y(t)$
4	Setpoint	$w(t)$
5	Model output	$y_m(t)$
6	Emulator output	$\hat{\phi}(t)$
7	Emulated signal	$\phi(t)$

The last column is not always relevant.

If you have **MATLAB**, the following .m file will help:

```
function [t,u,y,w,ym,phih,phi] = convert
%function [t,u,y,w,ym,phih,phi] = convert;
%Converts data from 'outdata.dat' into relevant column vectors.

%File convert.m
%P.J. Gawthrop, May 1988.

load outdata.dat;
t =    outdata(:,1);
u =    outdata(:,2);
y =    outdata(:,3);
w =    outdata(:,4);
ym =   outdata(:,5);
phih = outdata(:,6);
phi =  outdata(:,7);
```

For cascade, and multiple-input multiple-output systems, the situation is more complicated.

1. A column for each interaction variable is interposed immediately after the system input.
2. A new set of columns is created for each loop.

A useful .m file for two-loop cascade control (Chapter 9) is

```
function [t,u1,u2,y1,y2,w1,w2,ym1,ym2] = convert9
%function [t,u1,u2,y1,y2,w1,w2,ym1,ym2] = convert9;
%Converts data from 'outdata.dat' into relevant column vectors (Chapter 9).

%File convert9.m
```

```
%P.J. Gawthrop, May 1988.

load outdata.dat;
t = outdata(:,1);
u1 = outdata(:,2); u2 = outdata(:,9);
y1 = outdata(:,3); y2 = outdata(:,10);
w1 = outdata(:,5); w2 = outdata(:,11);
ym1 = outdata(:,6); ym2 = outdata(:,12);
```

and for two-loop multivariable control (Chapter 10):

```
    function [t,u1,u2,y1,y2,w1,w2,ym1,ym2] = convert10
%function [t,u1,u2,y1,y2,w1,w2,ym1,ym2] = convert10;
%Converts data from 'outdata.dat' into relevant column vectors. (Chapter 10)


%File convert10.m
%P.J. Gawthrop, May 1988.

load outdata.dat;
t =   outdata(:,1);
u1 = outdata(:,2); u2  = outdata(:,10);
y1 = outdata(:,3); y2  = outdata(:,11);
w1 = outdata(:,5); w2  = outdata(:,13);
ym1 = outdata(:,6); ym2 = outdata(:,14);
```

Outsyspar.dat

Outsyspar.dat contains estimated system polynomials arising from the simulation. For single-input single-output systems, the columns of this file are as follows:

Column	Variable	Symbol
1	Time	t
2	Estimation error	$\hat{e}(t)$
3	Sigma	σ
4	Estimated delay	T
5..	Estimated system numerator	$B(s)$
6..	Estimated system denominator	$A(s)$
7..	Estimated system initial conditions	$D(s)$

Columns labelled 5.., 6.. and 7.. are blocks of columns containing the polynomial coefficients in decreasing order. If you have **MATLAB,** the following .m file will help:

```
function [t,error,sigma,delay,A,B,D] = sysparconvert(nA,nB,nD);
%function [t,error,sigma,delay,A,B,D] = sysparconvert(nA,nB,nD);
%Gets system parameters from CSTC simulation
% nA, nB, nD:   Degrees of A, B and D
%P.J. Gawthrop, May 1988.
%File sysparconvert.m

load outsyspar.dat
t = outsyspar(:,1);
error = outsyspar(:,2);
sigma = outsyspar(:,3);
delay = outsyspar(:,4);
A = outsyspar(:,5:5+nA);
B = outsyspar(:,6+nA:6+nA+nB);
if nD>=0
    D = outsyspar(:,7+nA+nB:7+nA+nB+nD);
end;
```

For cascade, and multiple-input multiple-output systems, the situation is more complicated. A block of columns for each interaction polynomial is interposed immediately after the $B(s)$ polynomial.

Outempar.dat

Outempar.dat contains estimated emulator polynomials arising from the simulation. For single-input single-output systems, the columns of this file are as follows:

Column	Variable	Symbol
1	Time	t
2	Estimation error	$\hat{e}(t)$
3	Sigma	σ
4	Estimated delay	T
5..	Estimated emulator numerator (output)	$F(s)$
6..	Estimated emulator numerator (input)	$G(s)$
7..	Estimated emulator numerator (initial conditions)	$I(s)$

Columns labelled 5.., 6.. and 7.. are blocks of columns containing the polynomial coefficients in decreasing order. If you have **MATLAB,** the following .m file will help:

```
[t,error,sigma,F,G,I] = emparconvert(nF,nG,nI);
%[t,error,sigma,F,G,I] = emparconvert(nF,nG,nI);
%Gets emulator parameters from CSTC simulation
% nF, nG, nI:  Degrees of F, G and I

%P.J. Gawthrop, May 1988.

%File emparconvert.m

load outempar.dat

t=outempar(:,1);
error=outempar(:,2);
sigma=outempar(:,3);
F=outempar(:,4:nF+3);
G=outempar(:,nF+4:nF+4+nG);
if nI>0
      I=outempar(:,nF+5+nG:nF+5+nG+nI);
end;
```

For cascade, and multiple-input multiple-output systems, the situation is more complicated. A block of columns for each interaction polynomial is interposed immediately after the $G(s)$ polynomial.

0.2.8. PLOTTING RESULTS

All result files are in the form of columns of ASCII numbers as discussed in the previous section. Thus many plotting packages will be able to display them graphically. The figures in this book were obtained using **MATLAB.** The corresponding .m files are included with the distribution diskette as indicated in Table 0.1:

Table 0.1: MATLAB plotting commands

Chapter	Example	Command
1	1	plot1_1
1	2	plot1_2
1	3	plot1_3
3	All	plot3
4	All	plot4
5	All (parameters)	plot5p
5	All (data)	plot5
6	All	plot6
7	All	plot7
8	All	plot8
9	All	plot9
10	All	plot10

Note that when using **plot5** and **plot5p** the variables: **nA, nB** and **nI** must be set within **MATLAB** to correspond to the degrees of A, B and D respectively.

0.2.9. CREATING NEW EXAMPLES

It may be that you will not find a suitable starting example for the problem that you wish to simulate. If so, a new example may be created. There are three levels at which this may be done:

1. Same structure, same hidden variables
2. Same structure, new hidden variables
3. New structure

These possibilities are considered in turn. In each case, you can save your examples for later use by copying **inlog.dat** to a safe place. It can then be reused by copying back again. You may care to create your own version of **runex** for this purpose.

Same structure, same hidden variables

After using the **runex** command, a file **inlog.dat** is created containing the the values of the variables that have been changed by the user, together with those that have not been changed. Hidden variables remain hidden. Using **run**, in place of **runex**, uses this modified file. Thus repeated use of **run** allows *incremental* changes to be made to the exposed variables.

Same structure, new hidden variables

Answering **yes** to the initial question "Enter all variables (y/n)?" exposes *all* variables. The resultant **inlog.dat** contains not only modified values, but also a new list of hidden variables. Variables are marked as exposed in the inlog.dat file if

a) A value is changed

b) A value is left at default, but a <space> is inserted before the <return>.

Subsequent invocation of the **run** command uses this new file with the corresponding hidden variables.

Similar effects can be obtained by editing **inlog.dat**. Hidden variables are exposed by replacing the space forming the first character in a line by #; exposed variables may be hidden by replacing the '#' forming the first character in a line by a space.

New structure

If none of the supplied examples is appropriate, a completely new file can be created. Start by creating an empty file called **inlog.dat**. Execute the **run** command and Answer **yes** to the initial question "Enter all variables (y/n)?". No variables are hidden, and each defaults to an internal value. The first variable corresponds to the appropriate chapter of the book. Take particular care to choose the appropriate polynomial orders; there is no checking here, and unpredictable effects can occur if choices are made incorrectly. So it is important to think carefully before creating a new example.

CHAPTER 1

Continuous-Time Systems

Aims. To consider the representation of polynomials and transfer functions. To illustrate the properties of the continuous-time state-variable filter and to investigate the approximations involved in its discrete-time implementation.

1.1. IMPLEMENTATION DETAILS

1.1.1. [1.2] TRANSFER FUNCTIONS

The rational transfer functions considered in Vol. 1 are ratios of polynomials. Therefore the polynomial is a key data structure in the implementation of the corresponding algorithms in CSTC. In particular, the type **Polynomial** is defined as:

```
Polynomial =
 RECORD
  Deg: Degree;
  Coeff: ARRAY [0..MaxDegree] OF REAL
 END;
```

and the type **Degree** as

```
Degree = - 1..MaxDegree;
```

The two components of the record are **Deg** which is the degree of the polynomial, and **Coeff** the corresponding coefficients. It is convenient to allow a degree of -1 to indicate the absence of a particular polynomial.

CSTC includes a library of polynomial manipulation routines as indicated in the Table 1.1. The simpler routines are self explanatory, the more complex ones are as described in this volume. The source code for each algorithm is provided as part of CSTC to provide an executable description of the key algorithms of Vol 1.

Table 1.1: POLYNOMIAL MANIPULATION ROUTINES

Name	Function
PROCEDURE PolWrite	Writes a polynomial
PROCEDURE PolLineWrite	Writes a polynomial and appends newline
FUNCTION PolNorm	Finds the absolute value of the largest coefficient
PROCEDURE PolRemove	Removes unwanted coefficients
PROCEDURE PolTruncate	Removes small coefficients
PROCEDURE PolZero	Generates the zero polynomial of degree zero
PROCEDURE PolUnity	Generates the unit polynomial of degree zero
PROCEDURE PolEquate	Equates two polynomials
PROCEDURE PolOfMinusS	Generates polynomial with -s replacing s
PROCEDURE PolAdd	Adds two polynomials
PROCEDURE PolMinus	Subtracts two polynomials
PROCEDURE PolWeightedAdd	Adds two polynomials with weighting scalars
PROCEDURE PolScalarMultiply	Multiplies a polynomial by a scalar
PROCEDURE PolsMultiply	Multiplies a polynomial by s
PROCEDURE PolsDivide	Divides a polynomial by s
PROCEDURE PolMultiply	Multiplies two polynomials
PROCEDURE PolSquare	Given P(s) computes P(s)P(-s)
PROCEDURE PolSqrt	Given P(s)P(-s) computes P(s)
PROCEDURE PolNormalise	Normalises a pair of polynomials
FUNCTION PolGain	Steady-state gain of system represented by a polynomial
FUNCTION PolHFGain	High frequency transfer function gain
PROCEDURE PolUnitGain	Forces polynomail to have unit steady-state gain
PROCEDURE PolMarkovRecursion	Markov recursion algorithm
PROCEDURE PolDerivativeEmulator	Derivative emulator design
PROCEDURE PolDivide	Polynomial long division - gives quotient and remainder
PROCEDURE PolEuclid	Euclids algorithm for GCD of two polynomials
PROCEDURE PolDioRecursion	The diophantine recursion algorithm
PROCEDURE PolDiophantine	Solves the diophantine equation
PROCEDURE PolZeroCancellingEmulator	Zero cancelling emulator design
PROCEDURE PolInitialConditions	Emulator initial conditions
PROCEDURE PolEmulator	General emulator design
PROCEDURE PolPade	Pade polynomials
PROCEDURE PolDelayEmulator	General emulator design with time-delay

1.1.2. [1.4] THE MARKOV RECURSION ALGORITHM

This algorithm is implemented using procedure **PolMarkovRecursion.** Firstly, the algorithm checks whether

$$deg(F) < deg(A) - 1 \quad (1.1.2.1)$$

If so, then the corresponding Markov parameter is zero, and $F(s)$ is multiplied by s. The transfer function $F(s)/A(s)$ has the relative degree reduced by one, but, because of inequality 1 is still strictly proper.

If inequality 1 is not satisfied, then the three equations I-2 are implemented. The first equation is labelled 2a in the listing, the second 2b and the third 2c. In step 2b, E is multiplied by s. The zero degree coefficient is then made equal to the Markov parameter. In step 2c, F is multiplied by s and then added to $-hkA(s)$

1.1.3. [1.6] THE STATE-VARIABLE FILTER

A key algorithm in CSTC is the numerical solution of the state-variable filter given by the differential equation

$$\frac{d}{dt}\underline{X}(t) = A\underline{X}(t) + Uu \quad (1.1.3.1)$$

where the superscript c has been dropped for convenience. The algorithm given here is based on one given by Gawthrop and Roberts*. In our discrete-time implementation, values of $\underline{X}(t)$ are only required at the discrete time points

$$t = i\Delta \quad (1.1.3.2)$$

With this in mind, the differential equation equation can be integrated between two consecutive time point to yield

$$\underline{X}(i\Delta + \Delta) = e^{A\Delta}\underline{X}(i\Delta) + \int_{i\Delta}^{(i+1)\Delta} e^{A\Delta(t-\tau)}Uu(\tau)d\tau \quad (1.1.3.3)$$

At this stage, two *approximations* are made:

* Gawthrop, P.J. (1984): ' Parameter identification from non-contiguous data', Proceedings IEE, Vol 131 pt. D, No 6, pp261-265; Gawthrop, P.J., Kountzeris, A. and Roberts, J.B.(1988):' Parametric identification of non-linear roll motion from forced roll data' Journal of ship research, Vol. 32, No 2, pp101-111.

1. $e^{\underline{A}\Delta}$ is expanded in a truncated Maclaurin series

$$e^{\underline{A}\Delta} \cong \sum_{j=0}^{N} \frac{(\underline{A}\,\Delta)^j}{j!} \tag{1.1.3.4}$$

2. $u(\tau)$ is expanded in a truncated Maclaurin series

$$u(\tau) \cong \sum_{j=0}^{M} u^{[j]} \frac{\tau^j}{j!} \tag{1.1.3.5}$$

where

$$u^{[j]} = \frac{d^j}{dt^j} u(\tau) \tag{1.1.3.6}$$

Substituting these two expressions into 1.1.3.3 gives the following recursive scheme

$$\underline{X}(i\,\Delta + \Delta) = \sum_{k=0}^{N} \tilde{\underline{X}}_k \tag{1.1.3.7}$$

where

$$\tilde{\underline{X}}_k = \frac{\Delta}{k}\left[\underline{A}\,\tilde{\underline{X}}_{k-1} + \frac{\Delta^{k-1}}{(k-1)!}\underline{U}u^{[k-1]}\right] \quad 1 \le i \le M+1 \tag{1.1.3.8}$$

$$= \frac{\Delta}{k}\underline{A}\,\tilde{\underline{X}}_{k-1} \quad i > M+1 \tag{1.1.3.9}$$

and

$$\tilde{\underline{X}}_0 = \underline{X}(i\,\Delta) \tag{1.1.3.10}$$

This algorithm is implemented within procedure **cStateVariableFilter.** The Maclaurin series for $u(t)$ is truncated at M=2; thus the control signal is approximated by a ramp function joining two adjacent samples

$$u((i-1)\Delta + \tau) \cong u((i-1)\Delta) + u(i\,\Delta) - u((i-1)\Delta)\ \frac{\tau}{\Delta} \tag{1.1.3.11}$$

The current filter input $u(i\,\Delta)$ is held in variable **u;** the previous filter input $u((i-1)\Delta)$ is held in variable **FilterState.Old.** The variable **ApproximationOrder** corresponds to N, and the loop starting

```
FOR k := 1 TO ApproximationOrder DO
```

implements the recursive expressions 1.1.4.8&9. Variable **Increment** contains $\tilde{\underline{X}}$ and **FilterState.State** contains $\underline{X}$ The special cases k=1 and k=2 are handled by IF statements. The matrix multiplication is performed within the loops indicated in the listing; the sparseness of the matrix in I-1.6.3 is used to simplify the calculation.

1.1.4. IMPLEMENTATION OF THE DISCRETE-TIME STATE-VARIABLE FILTER

CSTC is primarily designed to implement the *continuous-time* algorithms to be found in Volume 1. However, only minor modifications are required to give a purely discrete-time implementation. The switch between the two domains is accomplished via the Boolean variable **ContinuousTime.**

Procedure **StateVariableFilter** encapsulates two versions of the state-variable filter algorithm: **cStateVariableFilter** for the continuous-time version and **dStateVariableFilter** for the discrete-time version. The choice between them is made in the statement

IF ContinuousTime THEN

appearing in procedure **StateVariableFilter.**

Procedure **dStateVariableFilter** has the same argument list as **cStateVariableFilter,** but the interpretation of the polynomial **A** is different: it contains the coefficients of

$$A(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_n \tag{1.1.4.1}$$

in place of the coefficients of

$$A(s) = a_0 s^n + a_1 s^{n-1} + \cdots + a_n \tag{1.1.4.2}$$

The corresponding state vector is given in the z-domain by

$$\underline{X} = \frac{1}{A(z)} \begin{bmatrix} z^n \\ z^{n-1} \\ \cdots \\ 1 \end{bmatrix} \tag{1.1.4.3}$$

The algorithm has two parts:

1. The components of the state are shifted (z corresponds to the forward shift operator),
2. The zeroth component (corresponding to z^n) is computed in terms of the other states and the filter input **u**.

The algorithm in **dStateVariableFilter** is simpler than that in **cStateVariableFilter;** this is a consequence of the algorithm domain matching the implementation domain. However, it is argued in volume 1 that the advantages of the continuous-time approach outweigh this disadvantage.

1.2. EXAMPLES

1.2.1. TRANSIENT RESPONSE OF OSCILLATOR.

Reference: Section 1.6; page 1-10

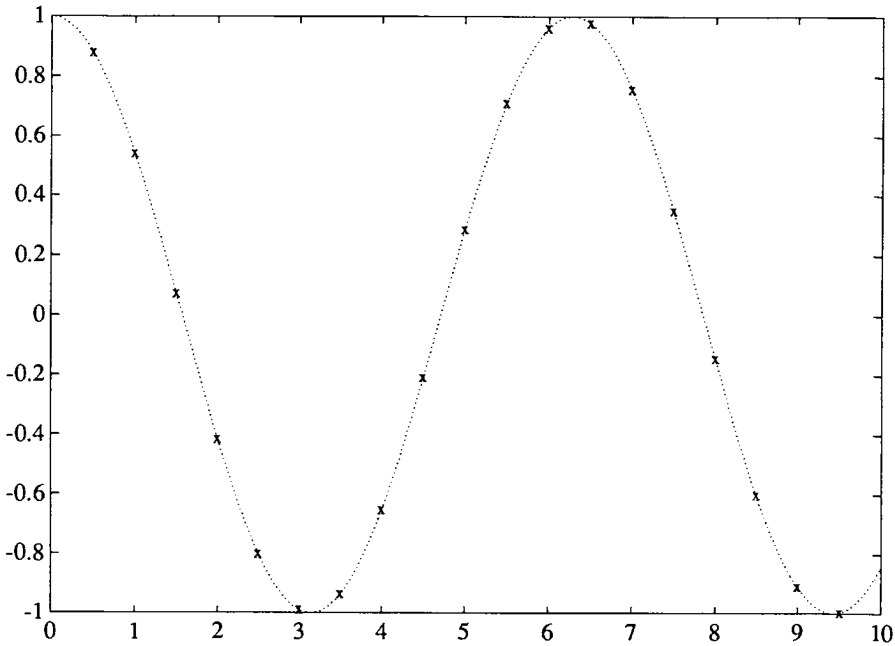

Figure 1.1. Transient response of oscillator.

Description

The state-variable filter provides a discrete-time approximation to a continuous-time transfer function. There are two ways of changing the accuracy of the approximation: the sample interval and the approximation order. There are two approximations involved in the the implementation: the series approximation to the state transition matrix and approximation of the input signal by a straight-line joining the samples. As the input is zero here, the latter approximation has no effect.

The simulated transfer function is:

$$\frac{1}{s^2 + 1} \tag{1.2.1.1}$$

and the initial 'position' is 1. Thus

$$D(s)=1 \tag{1.2.1.2}$$

The corresponding transient response is

$$\cos t \tag{1.2.1.3}$$

Programme interaction

```
runex 1 1
Example 1 of chapter 1: Transient response of oscillator.

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval         =    0.500000  :=
Approximation Order     =           5  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =    1.000000    0.000000    1.000000  :=
B (system numerator)     =    1.000000  :=
D (initial conditions)   =    1.000000    0.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
Step amplitude          =    0.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    10.000000
```

Discussion

The simulated step response, marked by 'x', is superimposed on the exact solution : cos t. The approximation is quite good. Note that the simulated output at t=0 (the initial condition) is not generated by the simulation programme.

Further investigations

1 Try the effect of varying the approximation order and the sample interval. As there is no input approximation, it should be possible to choose a large enough approximation order to work well for an arbitrarily large sample interval. What is the minimum acceptable value of the approximation order for sample intervals of: 0.1, 0.5, 1.0 and 2.0?

1.2.2. STEP RESPONSE OF OSCILLATOR.

Reference: Section 1.6; page 1-10

Description

This example is identical to example 1.2.1 except that the initial condition is zero and a step input is applied. Although the straight-line approximation is exact for t>0, it is incorrect during the initial timestep when the input changes from 0 to 1. For this reason, an alternative approximation is used where the input is deemed to be *constant* at the value measured at the current sample for the previous timestep. This gives an exact approximation for a step input, except that the output is delayed by one sample.

Programme interaction

```
runex 1 2
Example 1 of chapter 2: Step response of oscillator.

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source    ========
 ======== Filters        ========
Sample Interval      =    0.500000  :=
Approximation Order  =           5  :=
```

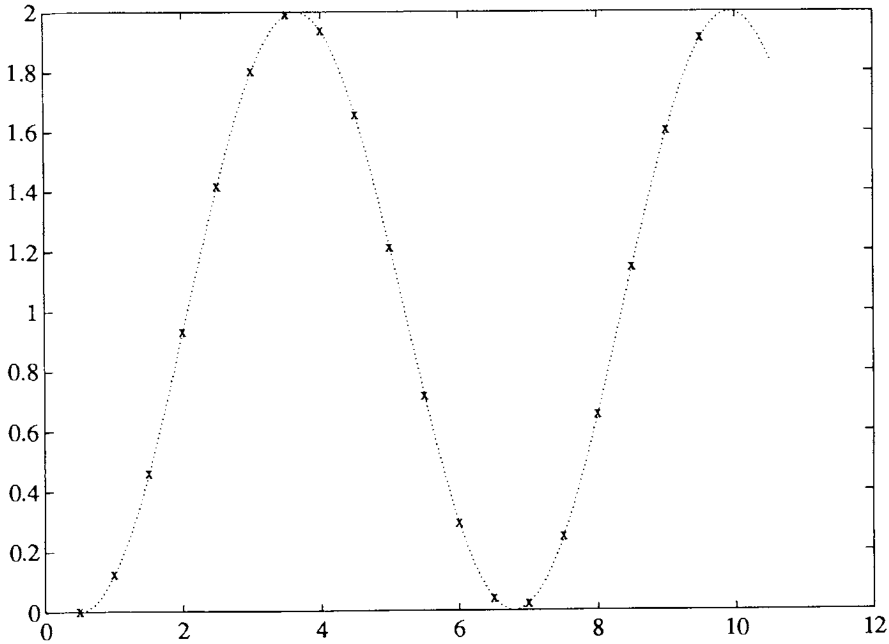

Figure 1.2. Step response of oscillator.

```
======== Control action  ========
======== Assumed system  ========
A (system denominator)    =    1.000000    0.000000    1.000000  :=
B (system numerator)      =    1.000000  :=
D (initial conditions)    =    0.000000    0.000000  :=
======== Simulation      ========
======== Setpoint        ========
Step amplitude            =    1.000000  :=
======== In Disturbance  ========
======== Out Disturbance ========
Constant between samples  = TRUE  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    10.000000
```

Discussion

The simulated step response, marked by 'x', is superimposed on the exact solution : 1 - cos t. The approximation is quite good.

Further investigations

1 Try the effect of varying the approximation order and the sample interval. As there is no input approximation, it should be possible to choose a large enough approximation order to work well for an arbitrarily large sample interval. What is the minimum acceptable value of the approximation order for sample intervals of: 0.1, 0.5, 1.0 and 2.0? Does this result correspond to that of example 1.2.1?

2 Set 'Constant between samples' to FALSE. This gives the straight-line approximation which is poor for the initial timestep. How does this affect the response? Is it possible to reduce the error by increasing the approximation order? Is it possible to reduce the error by decreasing the sample interval?

1.2.3. SINUSOIDAL RESPONSE OF OSCILLATOR.

Reference: Section 1.6; page 1-10

Description

This example is identical to example 1.2.1 except that the initial condition is zero and a sinusoidal sin t input is applied. The straight-line approximation is not exact in this case, but it is certainly better than the constant between samples approximation.

Unlike the previous two examples, then, the sample interval must be chosen so that the straight-line approximation is valid, and the approximation order then chosen appropriately.

The corresponding response is then:

$$0.5(\sin t - t \cos t) \quad (1.2.3.1)$$

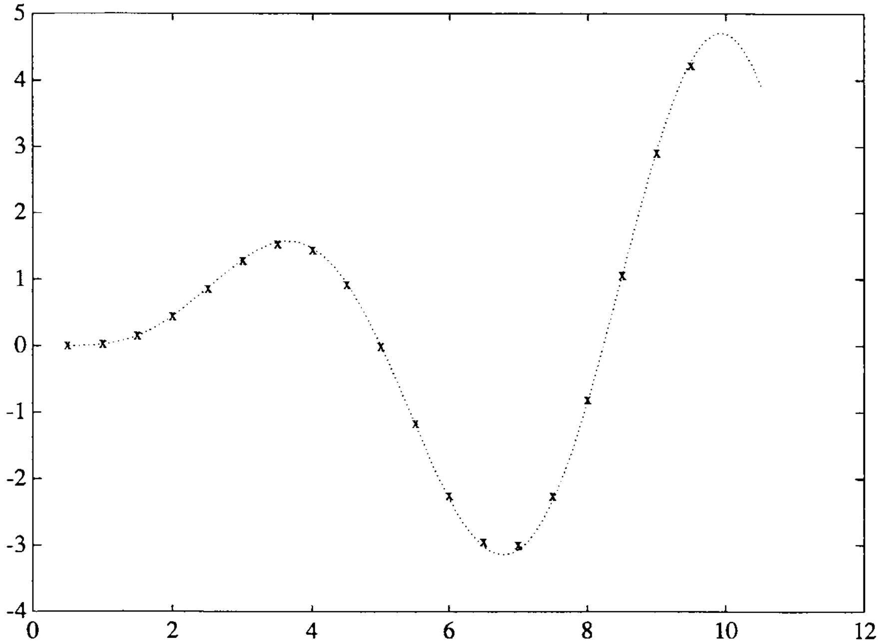

Figure 1.3. Sinusoidal response of oscillator.

Programme interaction

runex 1 3
Example 1 of chapter 3: Sinusoidal response of oscillator.

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
Sample Interval = 0.500000 :=
Approximation Order = 5 :=
======== Control action ========
======== Assumed system ========
A (system denominator) = 1.000000 0.000000 1.000000 :=

B (system numerator) = 1.000000 :=
D (initial conditions) = 0.000000 0.000000 :=
======== Simulation ========
======== Setpoint ========
Cos amplitude = 1.000000 :=
Period = 6.283185 :=
======== In Disturbance ========
======== Out Disturbance ========
Constant between samples = FALSE :=
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 10.000000

Discussion

The simulated step response, marked by 'x', is superimposed on the exact solution : 1 - cos t. The approximation is quite good.

Further investigations

1. Try the effect of varying the approximation order and the sample interval. As there is an input approximation, the approximation order cannot be increased to overcome inaccuracies due to a long sample interval. Choose a large approximation order (20), and find the longest acceptable sample interval. Using this interval, what is then the minimum acceptable approximation order?

2. Set 'Constant between samples' to TRUE. This gives the constant approximation which is poor for a sinusoidal input. How does this affect the response? Is it possible to reduce the error by increasing the approximation order? Is it possible to reduce the error by decreasing the sample interval?

CHAPTER 2

Emulators

Aims. To describe some implementation details of the emulator design methods of chapter I-2. To illustrate the emulator design methods of chapter I-2 by numerical examples.

2.1. IMPLEMENTATION DETAILS

2.1.1. [2.2] OUTPUT DERIVATIVES

The procedure **PolMarkovRecursion** for evaluating equation I-2.2.2 and I-2.2.5 is described in section 1.1.3. The polynomials $E_1(s)$ and $F_1(s)$ in equation I-2.2.19 are evaluated using equations I-2.2.23 and I-2.2.25 in conjunction with the Markov recursion algorithm in procedure **PolDerivativeEmulator.**

If A is zero degree (and assumed to be unity), then the solution is trivial

$$E(s) = P(s)C(s);\ F(s) = 0 \tag{2.1.1.1}$$

Otherwise the Markov recursion algorithm is initialised as in equation I-1.5.3. That is

$$E_0(s) = 0;\ F_0(s) = 1 \tag{2.1.1.2}$$

The Markov recursion algorithm is then used to recursively generate the $F_{1k}(s)$ and $E_{1k}(s)$ polynomials and take the weighted sum to form $E_1(s)$ and $F_1(s)$ as in equations I-2.2.23 and I-2.2.25.

It is important to note that

$$deg(C(s)) < deg(A(s)) \tag{2.1.1.3}$$

for this algorithm to work correctly.

Procedure **PolDerivativeEmulator** can be thought of as a way of performing **polynomial long division** as in equation I-2.2.26. This operation is needed later on, so the procedure **PolDivide** is defined exploiting this property. In general, the equation

$$\frac{B'(s)}{A'(s)} = E'(s) + \frac{F'(s)}{A'(s)} \tag{2.1.1.4}$$

can be solved by using procedure **PolDerivativeEmulator** with

$$P(s) = B'(s); \; C(s) = 1 \tag{2.1.1.5}$$

This is the method used in procedure **PolDivide.**

Procedure **PolDerivativeEmulator** is embedded in procedure **PolEmulator.** It is invoked if $Z(s)$ is a zero degree polynomial; it is assumed that, in this case

$$Z(s) = Z^-(s) = Z^+(s) = 1 \tag{2.1.1.6}$$

The emulator numerator polynomial

$$G_1(s) = E_1(s)B(s) = E_1(s)B^+(s) \tag{2.1.1.7}$$

is computed in procedure **PolEmulator,** together with the corresponding denominator polynomials GFilter and FFilter which are both equal to $C(s)$in this case.

Procedure **PolInitialConditions** is used to compute the initial condition terms. The first step is to compute the polynomial $E^D{}_1(s)$. The expression for $E^D{}_1(s)$ (equations I-2.2.14 and I-2.2.24) is the same as that for $E_1(s)$ but with $D(s)$ (the initial condition numerator) replacing $C(s)$ (the disturbance numerator). Hence $E^D{}_1(s)$ can be computed using procedure **PolDerivativeEmulator** but with the argument C replaced by D. The initial condition term is then calculated from equation I-2.2.21. In this case, $B^-(s) = 1$ so the statement

```
PolDivide(InitialCondition, Rem, InitialCondition,
        BMinus);
```

has no effect here. Finally, procedure **PolTruncate** is applied to clean up the initial condition polynomial.

2.1.2. [2.3] ZERO CANCELLING AND OTHER FILTERS

The algorithms required for the design of emulators corresponding to section I-2.4 are more complicated than those required for the previous section due to the additional design parameter $Z(s)$. There are many possible choices for $Z(s)$, but these must always obey the two design rules on page I-2-12. Here, $Z(s)$ is generated, in terms of user-chosen polynomials, by procedure **SetDesignKnobs.** The user supplies two polynomials **ZMinusPlus** ($Z^{-+}(s)$) and **ZPlus** ($Z^{+}(s)$), and two Boolean variables **ZHasFactorB** and **IntegralAction.** It is assumed that both polynomials ($Z^{-+}(s)$ and $Z^{+}(s)$) are stable.

Procedure **SetDesignKnobs** then generates Z, and decomposes B, as follows. There are three possibilities dependent on the two Boolean variables.

1. ZHasFactorB = FALSE

$B(s)$ is decomposed as

$$B^{+}(s) = B(s); \; B^{-}(s) = 1 \tag{2.1.2.1}$$

2. ZHasFactorB = TRUE, IntegralAction=FALSE

It is assumed that $B(0) \neq 0$. $B(s)$ is decomposed as

$$B^{+}(s) = 1; \; B^{-}(s) = B(s) \tag{2.1.2.2}$$

3. ZHasFactorB = TRUE, IntegralAction=TRUE

It is assumed that $B(s) = sB's$, $B'(0) \neq 0$. $B(s)$ is decomposed as

$$B^{+}(s) = s; \; B^{-}(s) = B'(s) \tag{2.1.2.3}$$

In each case, $B^{-}(s)$ is normalised so that $B^{-}(0) = 1$ and $B^{+}(s)$ adjusted accordingly. This is always possible as $B^{-}(0) \neq 0$.

$Z(s)$ and $Z^{-}(s)$ are generated as

$$Z^{-}(s) = Z^{-+}(s)B^{-}(s); \; Z(s) = Z^{+}(s)Z^{-}(s) \tag{2.1.2.4}$$

The emulator polynomials $E_2(s)$ and $F_2(s)$ are generated by procedure **PolZeroCancellingEmulator.**

Firstly, as discussed in section I-2.4 (page I-2-17), any common factors between $A(s)$ and $Z^-(s)$ are detected using Euclid's algorithm (see the next section) and removed. To retain the same $Z(s)$, this factor is then put into $Z^+(s)$. This latter step will only be useful if the factor is stable.

The Diophantine equation I-2.3.23 is solved for $E_2(s)$ and $F_2(s)$ using the procedure **PolDiophantine** described in the next section.

As with procedure **PolDerivativeEmulator,** procedure **PolZeroCancellingEmulator** is embedded in procedure **PolEmulator** to generate the remaining emulator polynomials. It is invoked if $Z(s)$ has degree greater than zero. It is assumed that this polynomial has been correctly generated (for example using **SetDesignKnobs**) to obey the two design rules on page I-2-12.

The emulator numerator polynomial

$$G_1(s) = E_1(s)B^+(s) \tag{2.1.2.5}$$

is computed in procedure **PolEmulator,** together with the corresponding denominator polynomials GFilter and FFilter which are both equal to $C(s)$in this case.

If a common factor g(s) of $A(s)Z^+(s)$ and $Z^-(s)$ is found, the resultant emulator parameters are not relevant - essentially the Diophantine equation has been solved with $g(s)P(s)$ replacing $P(s)$. The solution to the difficulty used here is to *remove* the factor g(s) from $Z^-(s)$ and append it to $Z^+(s)$. This removes the common factor whilst retaining the same $Z(s) = Z^+(s)Z^-(s)$ as before. However, this method will not be useful if g(s) is not stable.

Procedure **PolInitialConditions** is used to compute the initial condition terms. The first step is to compute the polynomial $E^D{}_2(s)$. The expression for $E^D{}_2(s)$ (equation I-2.3.7) is the same as that for $E_2(s)$ but with $D(s)$ (the initial condition numerator) replacing $C(s)$ (the disturbance numerator). Hence $E^D{}_2(s)$ can be computed using procedure **PolZeroCancellingEmulator** but with the argument C replaced by D. The initial condition term is then calculated from equation I-2.3.26.

2.1.3. SOLVING DIOPHANTINE EQUATIONS

As discussed on page I-2-18, there are three steps involved in solving Diophantine equations of the form

$$P(s)C(s) = E_2(s)A(s)Z^+(s) + F_2(s)Z^-(s) \tag{2.1.3.1}$$

for $E_2(s)$ and $F_2(s)$.

A Find the greatest common divisor of $Z^-(s)$ and $A(s)Z^+(s)$ using Euclid's algorithm.

B Solve

$$1 = e(s)A(s)Z^+(s) + f(s)Z^-(s) \qquad (2.1.3.2)$$

for e(s) and f(s).

C Use e(s) and f(s) to solve equation 1 recursively.

Steps A and B are implemented in procedure **PolEuclid;** step C is implemented in procedure **PolDioRecursion.**

Procedure **PolEuclid** has three main sections.

1. Procedure **FindGCD** which implements step A,
2. Procedure **DeduceEandF** which implements step B and
3. procedures to clean up E and F and to normalise the GCD.

Equations I-2.4.5&6 of the recursive algorithm are implemented in procedure **FindGCD.** The initialisation step of equation I-2.4.1 occurs implicitly in the parameter passing mechanism when

FindGCD(AlphaIminus1{A},AlphaI{B} : Polynomial);

is called within procedure **PolEuclid as**

FindGCD(a,b);

The corresponding quotients q_i are saved in an array for use in procedure **DeduceEandF.**

The equations for finding the polynomials e(s) and f(s) solving the Diophantine equation 2.1.4.2 are given on pages I-2-21&22. They are implemented in procedure **DeduceEandF.** The quotients q_i are always preceded by a minus sign in these equations, so as a first step in the algorithm, the quotients are all multiplied by -1. In the trivial case where the a(s) =a is scalar (N=0) then the particular solution

$$e = 1/a \,;\, f = 0 \textit{ is chosen,} \qquad (2.1.3.3)$$

otherwise, β and γ are initialised as in the equations following I-2.4.9.

Equations I.2.4.14 are recursively implemented in a FOR loop from N-1 down to 1.

The **Diophantine recursion algorithm** is summarised on page I-2-21. In the following discussion, we shall assume that $Z^-(s)$ has been adjusted to avoid factors in common with $A(s)Z^+(s)$ (see the previous section). The basic idea is that given $E_{2k}(s)$ and$F_{2k}(s)$ solving

$$\frac{s^k}{a(s)b(s)} = \frac{E_{2k}(s)}{b(s)} + \frac{F_{2k}(s)}{a(s)} \tag{2.1.3.4}$$

where

$$a(s) = A(s)Z^+(s);\ b(s) = Z^-(s) \tag{2.1.3.5}$$

It follows that

$$\frac{s^{k+1}}{a(s)b(s)} = \frac{sE_{2k}(s)}{b(s)} + \frac{sF_{2k}(s)}{a(s)} \tag{2.1.3.6}$$

$$= \frac{E_{2k+1}(s)}{b(s)} + \frac{F_{2k+1}(s)}{a(s)} \tag{2.1.3.7}$$

where $E_{2k+1}(s)$ and $F_{2k+1}(s)$ are defined by

$$E_{2k+1}(s) = sE_{2k}(s) + hkb(s);\ F_{2k+1}(s) = sF_{2k}(s) - hka(s) \tag{2.1.3.8}$$

If h_k is chosen as the first Markov parameter of $\frac{F_{2k}(s)}{a(s)}$, then $\frac{F_{2k+1}(s)}{a(s)}$ will be proper as required. As the solution for k=0 has been found using Euclid's algorithm as described above, this **Diophantine recursion algorithm** provides a means of solution for all k *without* needing to solve further Diophantine equations.

The recursive equations 2.1.4.6-8 are implemented in procedure **PolDioRecursion.** There are two cases. Firstly, if $deg(a(s)) < deg(sF(s)$, then the corresponding Markov parameter is zero and both $E_{2k}(s)$ and $F_{2k}(s)$ are multiplied by s. Otherwise, equations 2.1.4.6-8 are implemented.

The whole solution of the Diophantine equation is brought together in procedure **PolDiophantine.** Firstly, the Diophantine equation

$$1 = E0sA(s)Z^+(s) + F0sZ^-(s) \tag{2.1.3.9}$$

is solved using procedure **PolEuclid.** (If a common factor is found, then 1 is replaced by such a factor, but this situation is always avoided as described in section I-2.3.) Then the polynomials *Eks* and *Fks* solving

$$s^k = EksA(s)Z^+(s) + FksZ^-(s) \tag{2.1.3.10}$$

are computed recursively using procedure **PolDioRecursion** and these are summed, weighted by the corresponding coefficients of $P(s)C(s)$, to form $E_2(s)$ and $F_2(s)$.

Note

There is an inconsistency in notation between sections I-2.3 and I-2.4. The former defines the polynomials *Eks* and *Fks* by

$$s^k C(s) = EksA(s)Z^+(s) + FksZ^-(s) \tag{2.1.3.11}$$

and the latter by

$$s^k = EksA(s)Z^+(s) + FksZ^-(s) \tag{2.1.3.12}$$

It is the latter definition that is used here. If the former had been used then *Eks* and *Fks* would have been weighted by the coefficients of $P(s)$ (as in equations I-2.3.20&22), not by the coefficients of $P(s)C(s)$.

2.1.4. [2.6] APPROXIMATE TIME-DELAYS

The **Pade polynomial** of order N is computed by procedure **PolPade.** This recursively computes the coefficients using the formula I-2.6.7.

The emulator coefficients corresponding to $\overline{\phi}_4(s)$ are computed in procedure **PolDelayEmulator.** Firstly, procedure **PolPade** is used to find the denominator $T(s)$ of the time-delay approximation; the corresponding numerator $T(-s)$ is computed using procedure **PolOfMinusS.** The polynomials $P_T(s)$ and *Zminus* are computed according to equations I-2.6.8&9. Notice that this change is local to the procedure as **DesignKnobs** is passed by value. Procedure **PolEmulator** is then used to calculate the emulator coefficients based on the modified polynomials.

2.1.5. [2.7] LINEAR-IN-THE PARAMETERS FORM

In general, the emulator equation I-2.7.1 can be combined with I-2.7.6, I-2.7.7, or I-2.7.8 and rewritten as

$$\vec{\phi}^{**}(s) = \frac{G(s)}{G_f(s)}\overline{u}(s) + \frac{F(s)}{F_f(s)}\overline{y}(s) + \frac{I(s)Z^{-+}(s)}{G_f(s)} \tag{2.1.5.1}$$

where

$$G_f(s) = C(s)TsZ^{-+}(s);\ F_f(s) = C(s)Z^+(s) \tag{2.1.5.2}$$

This equation is implemented as function **Emulator**.

Firstly, the local variable **Em** is set to zero. The various components formed by filtering the system input, the system output and the initial condition term are in turn added to the local variable Em. Secondly, the terms corresponding to interaction variables (see chapter 10) are added to Em. Finally, Em is assigned to the function output.

In this implementation, the equation I-2.7.9 is not explicitly implemented. But if the operation of function **Filter**, and its associated procedures **StateVariableFilter** and **StateOutput**, are considered, it can be seen that the emulator output is found by generating the data vectors $\overline{\underline{X}}_u(s)$, $\overline{\underline{X}}_y(s)$ and $\overline{\underline{X}}_i(s)$, and then forming the inner product of these vectors with the vectors formed from the coefficients of the respective polynomials within function **StateOutput**.

2.1.6. DISCRETE-TIME IMPLEMENTATION

CSTC is primarily designed to implement the *continuous-time* algorithms to be found in Volume 1. However, it should be emphasised that *algebraicly* controller design in *discrete-time* is very similar.

The switch between the two domains is accomplished via the Boolean variable **ContinuousTime**. The minor modifications (with regard to the *design* algorithms) implied by setting this switch to FALSE can be seen from the listing of CSTC. The only place where **ContinuousTime** has any effect is in computing the steady-state gain of **BMinus**, using **PolGain** in procedure **SetDesignKnobs**. In the continuous-time case, the steady-state gain of $B^-(s)$ is $B^-(0)$; in the continuous-time case, the steady-state gain of $B^-(s)$ is $B^-(1)$. This is reflected in procedure **PolGain**.

2.2. EXAMPLES

2.2.1. OUTPUT DERIVATIVES

Reference: Section 2.2; pp 2-9 - 2-11.

Description

The example refers to the system:

$$\frac{B(s)}{A(s)} = \frac{0.1s+1}{s(s+1)} \tag{2.2.1.1}$$

with initial conditions defined by:

$$D(s) = 0.1s + 1 \tag{2.2.1.2}$$

The design is based on the output derivative approach with:

$$P(s) = C(s) = 0.5s+1 \tag{2.2.1.3}$$

Programme interaction

```
runex 2 1
Example 2 of chapter 1: Output derivatives

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000 :=
B (system numerator)      =    0.100000   1.000000 :=
D (initial conditions)    =    0.100000   1.000000 :=
 ======== Emulator design ========
P (model denominator)     =    0.500000   1.000000 :=
C (emulator denominator)  =    0.500000   1.000000 :=
-----------------------------------
      System polynomials
-----------------------------------
A         1.000000    1.000000    0.000000
B         0.100000    1.000000
D         0.100000    1.000000
-----------------------------------
```

Design polynomials

B+	*0.100000*	*1.000000*
B-	*1.000000*	
C	*0.500000*	*1.000000*
P	*0.500000*	*1.000000*
Z+	*1.000000*	
Z-	*1.000000*	
Z-+	*1.000000*	
F	*0.750000*	*1.000000*
F filter	*0.500000*	*1.000000*
G	*0.025000*	*0.250000*
G filter	*0.500000*	*1.000000*
I	*0.200000*	
E	*0.250000*	
ED	*0.050000*	

Discussion

The emulator parameters agree with those given in volume I. $G(s)$ has one root at s=10 corresponding to the system zero.

Further investigations

1. Try the effect of varying P, A and B. Take careful note of the degrees of the various polynomials. Check that $G(s)$ contains $B(s)$ as a factor.

2.2.2. ZERO CANCELLATION

Reference: Sections 2.4, pp I-22 - I-26

Description

This example illustrates the design of an emulator for multiple derivatives with zero cancellation: that is, multiple derivatives of the partial state. The first example on page I-2-25 is used, the second example appears under further investigation. Note that the Boolean variable 'Z has factor B' is now TRUE; this gives the zero cancellation effect.

Programme interaction

```
runex 2 2
Example 2 of chapter 2:  Zero cancellation

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000 :=
B (system numerator)      =    0.100000   1.000000 :=
D (initial conditions)    =    0.100000   1.000000 :=
 ======== Emulator design ========
Z has factor B            = TRUE  :=
P (model denominator)     =    0.250000   1.000000   1.000000 :=
C (emulator denominator)  =    0.500000   1.000000 :=
Small positive number     =    0.000100 :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000   1.000000   0.000000
B          0.100000   1.000000
D          0.100000   1.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000
B-         0.100000   1.000000
C          0.500000   1.000000
P          0.250000   1.000000   1.000000
Z+         1.000000
Z-         0.100000   1.000000
Z-+        1.000000
------------------------------------
F          0.861111   1.000000
F filter   0.500000   1.000000
G          0.125000   0.538889
G filter   0.500000   1.000000
I          0.288889
E          0.125000   0.538889
ED         0.025000   0.250000
------------------------------------
```

Discussion

Note that the emulator parameters agree with those of volume I. This emulator should be compared with that of the previous example; in particular, note that G does not contain $B(s)$ as a factor. What is the root of $G(s)$?

Further investigations

1. Try changing $B(s)$ to give a non-minimum phase system.

$$B(s) = 1-s$$

   The resultant emulator should correspond to equation 47 on page I-2-25.

2. Try the effect of varying P, A and B. Take careful note of the degrees of the various poynomials.
3. Try setting the Boolean variable 'Z contains factor B' to FALSE. The resultant emulator then corresponds to the multiple derivative emulator of the previous example.
4. Try using the following system which has a pole/zero cancellation:

$$\frac{B(s)}{A(s)} = \frac{s+1}{s^2+s} = \frac{1}{s} \tag{2.2.2.1}$$

   Notice that the algorithm finds the GCD of $Z^{-}(s)$ and $A(s)$.

5. Try using the following system which has an approximate pole/zero cancellation:

$$\frac{B(s)}{A(s)} = \frac{0.99s+1}{s^2+s} \doteq \frac{1}{s} \tag{2.2.2.2}$$

   Notice that the emulator now has rather strange coefficients due to this approximate cancellation. Try the effect of changing the 'Small positive number' to 0.1. The algorithm now finds the approximate cancellation.

2.2.3. PREDICTORS

Reference: Section 2.6; pp 2-33 - 2-36.

Description

This example illustrates emulator design for a system with a time delay using the Pade approximation. Essentially, as discussed in section I-2.5, the time delay translates into a rational non-minimum phase transfer function, and the zero-cancelling algorithm is applied.

The system has a first order rational part with unit time constant together with a unit delay

$$e^{-sT}\,\frac{B(s)}{A(s)} = e^{-s}\,\frac{1}{1+s} \tag{2.2.3.1}$$

Programme interaction

```
runex 2 3
Example 2 of chapter 3:  Predictors

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Assumed system  ========
A (system denominator)   =    1.000000   1.000000  :=
B (system numerator)     =    1.000000  :=
Time delay               =    1.000000  :=
 ======== Emulator design ========
Z has factor B           = TRUE  :=
P (model denominator)    =    1.000000  :=
C (emulator denominator) =    1.000000  :=
Pade approximation order =           4  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    1.000000
B          1.000000
D          0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000
B-         1.000000
C          1.000000
P          1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
Pade       0.000595    0.011905    0.107143    0.500000    1.000000
-----------------------------------
F          0.367879
F filter   1.000000
G          0.000376    0.015908    0.051819    0.632121
G filter   0.000595    0.011905    0.107143    0.500000    1.000000
I
```

E	*0.000376*	*0.015908*	*0.051819*	*0.632121*
ED	*0.000000*	*0.000000*	*0.000000*	*0.000000*

Discussion

Note that the degree of $F(s)$ is the same as that of $C(s)$ but the degree of G is increased by the degree of the Pade approximation.

Further investigations

1. Try varying the order of the Pade approximation.
2. Try varying the time delay. Note that the gain of the emulator transfer function multiplying y decreases with delay; this is because the future becomes less dependent on the present for large delays.
3. Try including multiple derivatives by using P.

CHAPTER 3

Emulator-Based Control

Aims. To describe the implementation of the controller. To illustrate the behaviour of the various control algorithms when the system is precisely known.

3.1. IMPLEMENTATION DETAILS

3.1.1. [3.2] THE CONTROL LAW

On page I-3-2, the control law is written in two forms. As equation I-3.2.1 appears to give an explicit expression for the control signal, whereas equation I-3.2.2 gives an implicit expression for the control signal, we will refer to equation I-3.2.1 as the **explicit form** and to equation I-3.2.2 as the **implicit form.**

Both forms are implemented in CSTC within procedure **Control:** the implicit form is implemented in procedure **ImplicitSolution;** the explicit form appears directly in procedure **Control.** The choice between the two method is made on the basis of the relative degree of $Q(s)$. Now, as stated in section I3.2, $\frac{1}{Q(s)}$ must be proper, so there are two possibilities: either

1. $\frac{1}{Q(s)}$ is *strictly* proper or
2. $\frac{1}{Q(s)}$ has numerator and denominator of equal degrees.

In the former case, the **explicit form** of the control equation is appropriate as the right-hand side of I-3.2.1 does not depend instantaneously on the the control signal u. In the latter case, $Q(s)$ is also proper, and so I-3.2.2 contains proper transfer functions. This decision is made at the IF statement in **Control.**

Explicit solution

The explicit solution is straightforward, the error signal **w - PhiHat** is fed into a filter implementing the transfer function $\frac{1}{Q(s)}$. **PhiHat** itself is generated using procedure **Emulator** within the body of procedure **SelfTuningControl.**

Implicit solution

The implicit solution is more complex. Essentially, the equation has to decomposed into two terms: one independent on the current control signal, and the remainder. In principle, this can be done by considering the instantaneous gains other various transfer-functions involved, but a more direct approach is used here. Within function **ImplicitSolution,** the term

$$\overline{\phi}^{*}(s) + Q(s)\hat{u}(s) \tag{3.1.1.1}$$

is evaluated twice using the the state from the previous time step: firstly with the current value of control equal to zero; and secondly with the current value of control equal to unity. These values are stored in **PhiQ0Hat** and in **PhiQ1Hat** respectively. It is important to realise that two copies of the past states of the emulator and the $Q(s)$ filter are made for this purpose by passing **Em0State, Em0State, Q0State,** and **Q1State** by value. The term independent of the *current* control signal is then **PhiQ0Hat - w**; the remaining term is then the product of the current control signal **u** with **PhiQ1Hat - PhiQ0Hat.** The control signal can then be directly computed.

This approach has the disadvantage of requiring a lot of computation; procedure **Emulator** is executed an additional two times. But this approach has the merit of solving the equation *exactly* for the discrete-time approximation of the actual continuous-time control law. This has found to be more effective numerically than computing the two components of the control equation directly.

Limiting the control signal

As emphasised in section I-3.2, it is essential that all filters comprising the emulator-based control act on the actual control signal sent to the process - including the effect of any limiting - rather than the computed control signal. This is achieved in CSTC by implementing the procedure **Emulator** *after* the control signal limiting in **PutData** within the body of **SelfTuningControl.** In addition, when the implicit control law calculation is used, the filter corresponding to $Q(s)$ is updated immediately following the emulator.

PutData implements two forms of control signal modification dependent on the value of the Boolean variable **Switched.** If **Switched** is false then the control signal is truncated if greater than **Max** or less than **Min.** If, on the other hand, **Switched** is true then the the control signal is set to **Max** or to **Min** depending on which value is closest. This thus implements an elementary form of switched control; more advanced versions appear in a recent paper*.

3.1.2. INTEGRAL ACTION [3.10]

The PID design rule 1 (page I-3-23) requires that $A(s)$ and $B(s)$ have a common root s=0. This is achieved in CSTC via the Boolean variable **IntegralAction.** If this variable is 'TRUE' then the factor s is appended to $A(s)$ and $B(s)$ in procedure **SystemInitialise** using procedure **SystemInitialise.** In the multi-loop case, the interaction polynomials **BInteraction[i]** are also multiplied by s. The initial condition term is similarly treated.

PID design rule 2 (page I-3-23) requires that the factor s is put into $B^+(s)$. This is taken care of in procedure **SetDesignKnobs.**

3.2. EXAMPLES

3.2.1. MODEL REFERENCE CONTROL

Reference: Section 3.4; page 3-12.

---

* Demircioglu, H. and Gawthrop, P.J. (1988):" Continuous-time relay self-tuning control", Int. J. Control. **47**, pp. 1061-1080.

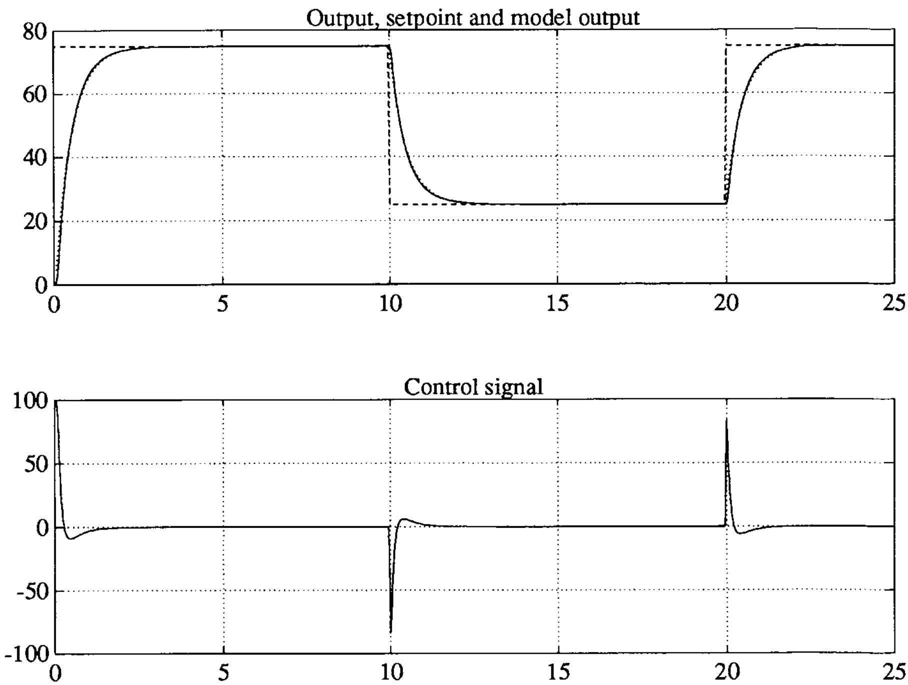

Figure 3.1. Model reference control

Description

As discussed in volume I, the emulator designed in example 2.2.1 may be embedded in a feedback loop to give model reference control. The system numerator has been multiplied by 10 for the purposes of this example.

The aim of the controller is to make the system output follow the model:

$$\bar{y}(s) = \frac{Z(s)}{P(s)}\bar{w}(s) \tag{3.2.1.1}$$

where, in this case, $Z(s)=1$ and $P(s) = 1+Ts$, where the model time-constant T = 0.5.

Programme interaction

```
runex 3 1
Example 3 of chapter 1:  Model reference control

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters        ========
Sample Interval         =    0.050000  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =     1.000000    1.000000    0.000000  :=
B (system numerator)     =    1.000000   10.000000  :=
D (initial conditions)   =    0.000000    0.000000  :=
 ======== Emulator design ========
P (model denominator)     =    0.500000    1.000000  :=
C (emulator denominator)  =    0.500000    1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000     1.000000     0.000000
B          1.000000    10.000000
D          0.000000     0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          1.000000    10.000000
B-         1.000000
C          0.500000     1.000000
P          0.500000     1.000000
Z+          1.000000
Z-         1.000000
Z-+         1.000000
------------------------------------
F          0.750000     1.000000
F filter   0.500000     1.000000
G          0.250000     2.500000
G filter   0.500000     1.000000
I
E          0.250000
ED          0.000000
------------------------------------
 ======== Controller     ========
Maximum control signal    =   100.000000  :=
```

Minimum control signal = -100.000000 :=
Switched control signal = FALSE :=
======== Simulation ========
======== Setpoint ========
======== In Disturbance ========
======== Out Disturbance ========
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

As expected, $y(t) \doteq y_m(t)$. Any discrepancy is due to numerical inaccuracy.

Further investigations

1. Try the effect of varying the time constant T of the inverse model P. How does this affect the system output and the control signal?

2. The emulator denominator $C(s)$ is also of the form 1+Ts. Try the effect of varying the time constant T of the emulator denominator C. How does this affect the system output and the control signal?

3. Try changing the limits of the control signal so that it is clipped; for example choose 'Maximum control signal' as 10 and 'Minimum control signal' as -10. How does this affect the system output and the control signal?

4. The controller and simulation are implemented as discrete-time systems. Try the effect of varying the sample interval on closed-loop performance.

5. Try using a switched controller by setting 'Switched control signal' to TRUE. How does the performance depend on:

   a) Sample interval

   b) The maximum and minimum control limits.

3.2.2. POLE-PLACEMENT CONTROL

Reference: Section 3.4; page 3-13.

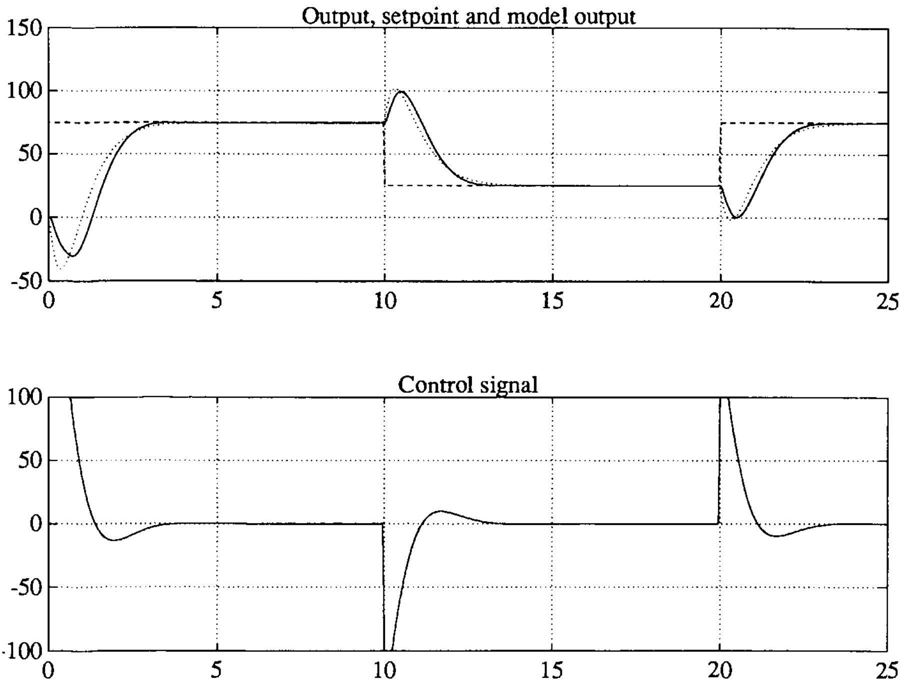

Figure 3.2. Pole-placement control

Description

As discussed in volume I, the emulator designed in the second example of section I-2.4 may be embedded in a feedback loop to give pole-placement control.

The aim of the controller is to make the system output follow the model:

$$\bar{y}(s) = \frac{Z(s)}{P(s)}\bar{w}(s) \tag{3.2.2.1}$$

where, in this case, $Z(s) = B(s)$ and $P(s) = (1+Ts)^2$ where the model time-constant T = 0.5.

Programme interaction

```
runex 3 2
Example 3 of chapter 2:  Pole-placement control

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters        ========
Continuous-time?          = TRUE  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =   -1.000000   1.000000  :=
 ======== Emulator design ========
Z has factor B            = TRUE  :=
P (model denominator)     =    0.250000   1.000000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A           1.000000    1.000000    0.000000
B          -1.000000    1.000000
D           0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          1.000000
B-         -1.000000    1.000000
C           0.500000    1.000000
P           0.250000    1.000000    1.000000
Z+          1.000000
Z-         -1.000000    1.000000
Z-+         1.000000
------------------------------------
F           0.937500    1.000000
F filter    0.500000    1.000000
G           0.125000    1.562500
G filter    0.500000    1.000000
I
E           0.125000    1.562500
ED          0.000000    0.000000
------------------------------------
 ======== Controller     ========
Maximum control signal   =   100.000000  :=
```

Minimum control signal = -100.000000 :=
Switched control signal = FALSE :=
======== Simulation ========
======== Setpoint ========
======== In Disturbance ========
======== Out Disturbance ========
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

In this case, note the typical behaviour of a system with right-hand plane zeros: the output initially goes the wrong way in response to a step change.

Further investigations

1. Try the effect of varying the time constant T of the inverse model P. How does this affect the system output and the control signal?
2. Try repeating this example using the same system as the previous section ($B(s) = 10+s$). How does the closed-loop response when using pole-placement differ from that when using model-reference control?
3. Try using a switched controller by setting 'Switched control signal' to TRUE. How does the performance depend on:
   a) Sample interval
   b) The maximum and minimum control limits.

3.2.3. USING A SETPOINT FILTER

Reference: Section 3.5; page 3-15.

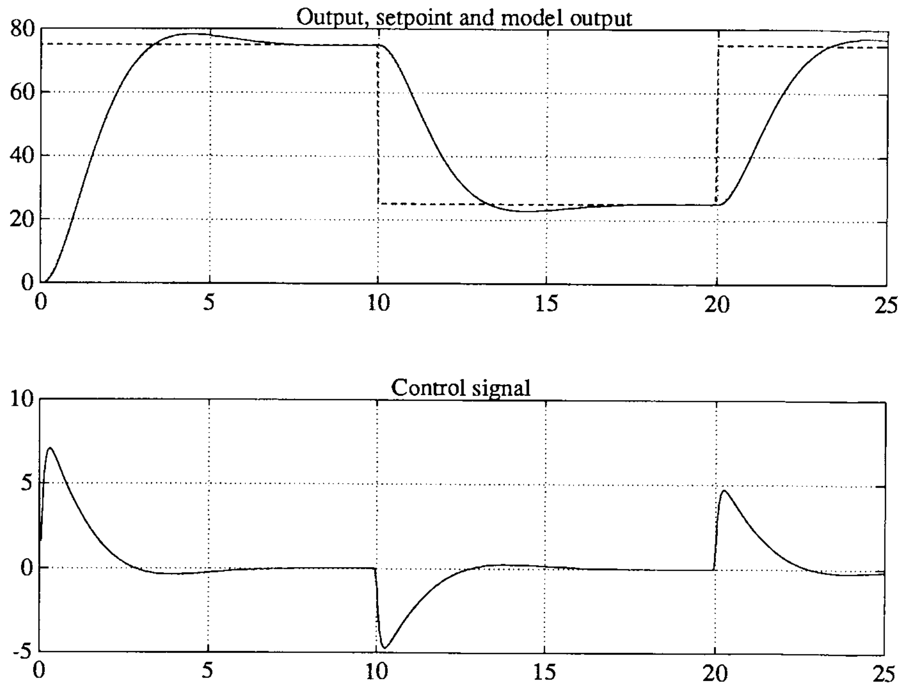

Figure 3.3. Using a setpoint filter

Description

This example is identical to example 3.2.1 except that a setpoint filter is added:

$$\overline{w}_R(s) = R(s)\overline{w}(s);\ R(s) = \frac{0.5s+1}{s^2 + \sqrt{2}s + 1} \tag{3.2.3.1}$$

The closed loop response is thus:

$$\overline{y}(s) = \frac{Z(s)}{P(s)}R(s)\overline{w}(s) = \frac{1}{0.5s+1}\ \frac{0.5s+1}{s^2 + \sqrt{2}s + 1}\overline{w}(s) \tag{3.2.3.2}$$

$$= \frac{1}{s^2 + \sqrt{2}s + 1}\overline{w}(s)$$

Programme interaction

```
runex 3 3
Example 3 of chapter 3:  Using a setpoint filter

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters        ========
Sample Interval          =    0.050000  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =    1.000000   10.000000  :=
D (initial conditions)    =    0.000000   0.000000  :=
 ======== Emulator design ========
P (model denominator)     =    0.500000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000    1.000000     0.000000
B          1.000000   10.000000
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000   10.000000
B-         1.000000
C          0.500000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          0.750000    1.000000
F filter   0.500000    1.000000
G          0.250000    2.500000
G filter   0.500000    1.000000
I
E          0.250000
ED         0.000000
------------------------------------
 ======== Controller      ========
R numerator              =    0.500000   1.000000  :=
```

R denominator = *1.000000 1.414000 1.000000 :=*
Maximum control signal = *100.000000 :=*
Minimum control signal = *-100.000000 :=*
======== Simulation ========
======== Setpoint ========
======== In Disturbance ========
======== Out Disturbance ========
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000
More time = FALSE :=

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

Note that the control signal is considerably reduced, and that the output is smoother.

Further investigations

1. Try the effect of different choices of $R(s)$ and $P(s)$ on the system input and output if $\frac{R(s)}{P(s)}$ does not change.

2. Try the effect of different choices of $R(s)$ and $P(s)$ on the system input and output if $\frac{R(s)}{P(s)}$ does change.

3. Choose $R(s)$ to give a critically damped response, for example:

$$R(s) = \frac{0.5s+1}{s^2 + 2s + 1} \tag{3.2.3.3}$$

3.2.4. CONTROL-WEIGHTED MODEL REFERENCE

Reference: Section 3.6; page 3-16.

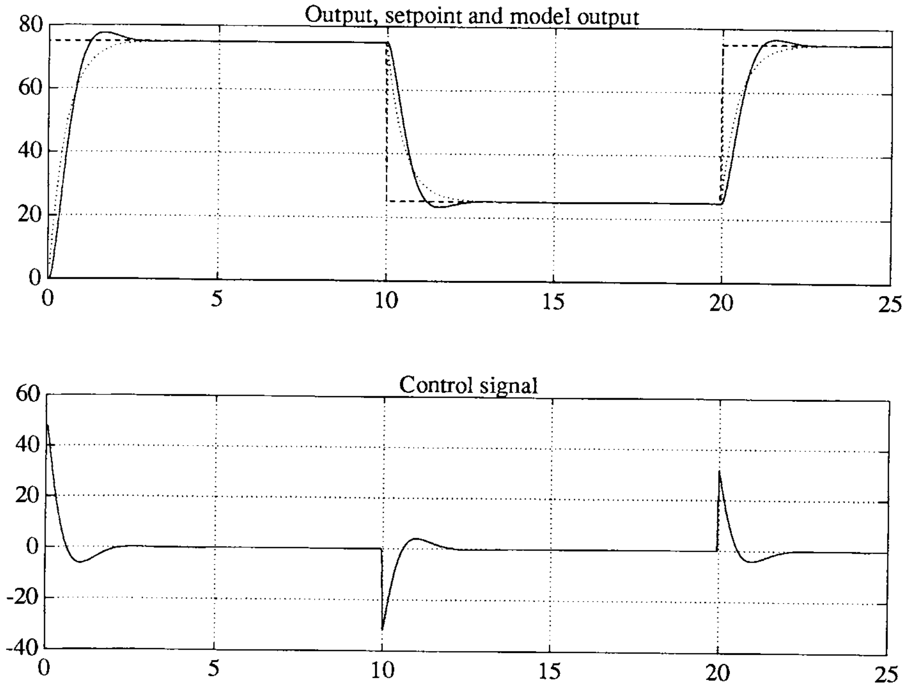

Figure 3.4. Control-weighted model reference

Description

In example 3.2.1, exact model-reference control was achieved by setting $Q(s)=0$. For this example, $Q(s)$ is chosen as

$$Q(s) = \frac{0.1s}{s+1} \qquad (3.2.4.1)$$

this satifies the $Q(s)$ design rule on page I-3-17: $Q(0) = 0$.

Programme interaction

runex 3 4
Example 3 of chapter 4: Control-weighted model reference

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval           =    0.050000  :=
Continuous-time?          = TRUE  :=
 ======== Control action  ========
Automatic controller mode = TRUE  :=
 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =    1.000000   10.000000  :=
D (initial conditions)    =    0.000000   0.000000  :=
 ======== Emulator design ========
P (model denominator)     =    0.500000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000  :=
--------------------------------------
      System polynomials
--------------------------------------
A          1.000000    1.000000    0.000000
B          1.000000    10.000000
D          0.000000    0.000000
--------------------------------------
      Design polynomials
--------------------------------------
B+          1.000000    10.000000
B-         1.000000
C          0.500000    1.000000
P          0.500000    1.000000
Z+          1.000000
Z-         1.000000
Z-+         1.000000
--------------------------------------
F          0.750000    1.000000
F filter   0.500000    1.000000
G          0.250000    2.500000
G filter   0.500000    1.000000
I
E          0.250000
ED          0.000000
--------------------------------------
 ======== Controller      ========
Q numerator               =    1.000000   0.000000  :=
Q denominator             =    1.000000   1.000000  :=
Maximum control signal    =    100.000000  :=
Minimum control signal    =   -100.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
```

```
======== In Disturbance ========
======== Out Disturbance ========
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is    25.000000
More time              = FALSE  :=
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

Notice that the control signal is reduced with respect to that of example 3.2.1. The model following is no longer exact, but the use of the $Q(s)$ design rule ensures that there is no steady-state offset.

Further investigations

1. Try the effect of varying q in:

$$Q(s) = \frac{qs}{1+s} \tag{3.2.4.2}$$

2. Try the effect of varying T in:

$$Q(s) = \frac{s}{1+Ts} \tag{3.2.4.3}$$

3. Replace $Q(s)$ by:

$$Q(s) = q \tag{3.2.4.4}$$

   There is still no offset as, in this case, the system contains an integrator and so the control signal is zero in the steady-state.

4. Replace $Q(s)$ by:

$$Q(s) = q \tag{3.2.4.5}$$

   and $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{3.2.4.6}$$

Note that there is now an offset dependent on q.

5. Use the default value of $Q(s)$ but replace $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{3.2.4.7}$$

Note that the offset disappears.

3.2.5. CONTROL-WEIGHTED POLE-PLACEMENT

Reference: Section 3.6; page 3-16.

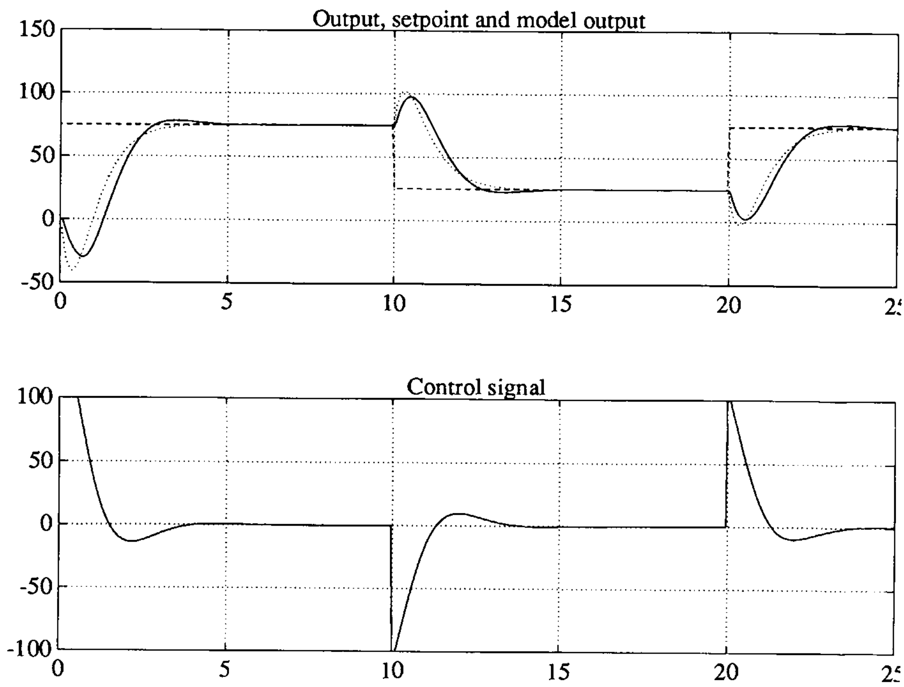

Figure 3.5. Control-weighted pole-placement

Description

In example 2, exact pole-placement control was achieved by setting $Q(s)=0$. For this example, $Q(s)$ is chosen as

$$Q(s) = \frac{s}{s+1} \tag{3.2.5.1}$$

As $Q(0)=0$, this satifies the $Q(s)$ design rule on page 3-17 of vol. 1.

Programme interaction

```
runex 3 5
Example 3 of chapter 5:  Control-weighted pole-placement

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters        ========
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =   -1.000000   1.000000  :=
 ======== Emulator design ========
Z has factor B            = TRUE  :=
P (model denominator)     =    0.250000   1.000000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A           1.000000    1.000000    0.000000
B          -1.000000    1.000000
D           0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          1.000000
B-         -1.000000    1.000000
C           0.500000    1.000000
P           0.250000    1.000000    1.000000
Z+          1.000000
Z-         -1.000000    1.000000
Z-+         1.000000
------------------------------------
```

```
F          0.937500    1.000000
F filter   0.500000    1.000000
G          0.125000    1.562500
G filter   0.500000    1.000000
I
E          0.125000    1.562500
ED         0.000000    0.000000
------------------------------------
 ======== Controller      ========
Q numerator             =    0.100000    0.000000  :=
Q denominator           =    1.000000    1.000000  :=
Maximum control signal   =  100.000000  :=
Minimum control signal   = -100.000000  :=
 ======== Simulation      ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

Notice that the control signal is reduced with respect to that of example 3.2.2. The model following is no longer exact, but the use of the $Q(s)$ design rule ensures that there is no steady-state offset.

Further investigations

1. Try the effect of varying q in:

$$Q(s) = \frac{qs}{1+s} \tag{3.2.5.2}$$

2. Try the effect of varying T in:

$$Q(s) = \frac{s}{1+Ts} \tag{3.2.5.3}$$

3. Replace $Q(s)$ by:

$$Q(s)=q \tag{3.2.5.4}$$

There is still no offset as, in this case, the system contains an integrator and so the control signal is zero in the steady-state.

4. Replace $Q(s)$ by:

$$Q(s)=q \tag{3.2.5.5}$$

and $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{3.2.5.6}$$

Note that there is now an offset dependent on q.

5. Use the default value of $Q(s)$ but replace $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{3.2.5.7}$$

Note that the offset disappears.

3.2.6. TIME-DELAY SYSTEM

Reference: Section 3.7; page 3-18.

Description

This example corresponds to example 3.2.1, except that the system now is first order and has a time delay of one unit.

$$\frac{B(s)}{A(s)} = e^{-s}\frac{1}{s} \tag{3.2.6.1}$$

The corresponding emulator is based on the Pade approximation approach discussed in section I-2.6. But note that the simulation of the system uses an exact time-delay algorithm.

Programme interaction

runex 3 6
Example 3 of chapter 6: Time-delay system

======== *C S T C Version 6.0* ========

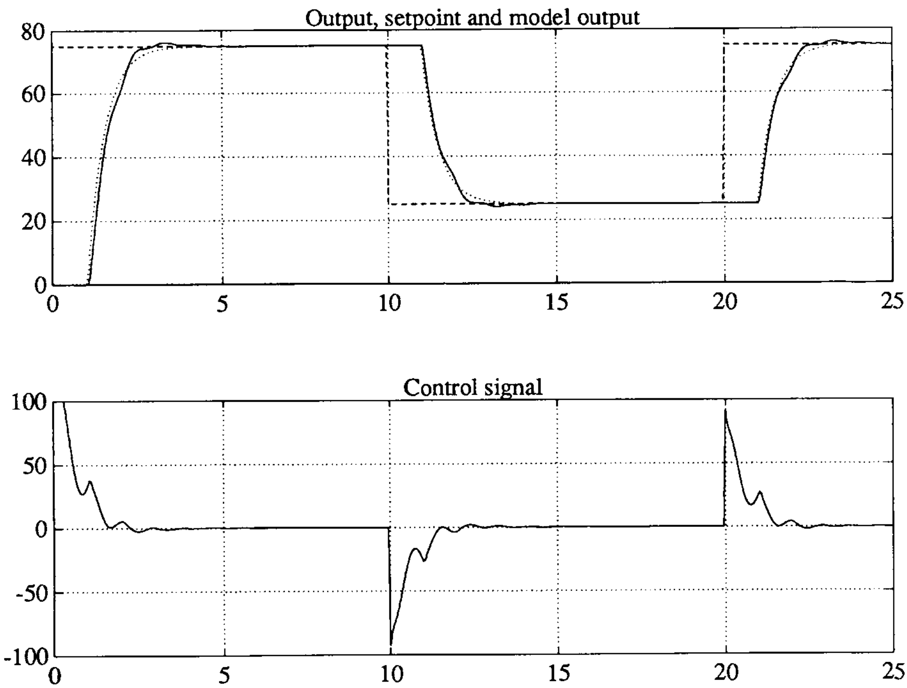

Figure 3.6. Time-delay system

Enter all variables (y/n, default n)?

```
======== Data Source     ========
======== Filters         ========
Sample Interval          =    0.050000  :=
======== Control action  ========
======== Assumed system  ========
A (system denominator)   =    1.000000   0.000000  :=
B (system numerator)     =    1.000000  :=
D (initial conditions)   =    0.000000  :=
======== Emulator design ========
P (model denominator)    =    0.500000   1.000000  :=
C (emulator denominator) =    1.000000  :=
Pade approximation order =         3  :=
----------------------------------
         System polynomials
----------------------------------
```

```
A          1.000000    0.000000
B          1.000000
D          0.000000
------------------------------------
     Design polynomials
------------------------------------
B+          1.000000
B-         1.000000
C          1.000000
P          0.500000    1.000000
Z+          1.000000
Z-         1.000000
Z-+         1.000000
Pade        0.008333    0.100000    0.500000    1.000000
------------------------------------
F          1.000000
F filter   1.000000
G          0.004167    0.066667    0.250000    1.500000
G filter   0.008333    0.100000    0.500000    1.000000
I
E          0.004167    0.066667    0.250000    1.500000
ED          0.000000    0.000000    0.000000    0.000000
------------------------------------
 ======== Controller      ========
Maximum control signal    =  100.000000 :=
Minimum control signal    = -100.000000 :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time              = FALSE  :=
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

Despite the approximation involved, the model following is close. Note that the system output is delayed by one time unit.

Further investigations

1. Try the effect of using a lower order (for example 1) approximation to a time delay in the emulator calculation.
2. Try the effect of using a higher order (for example 5) approximation to a time delay in the emulator calculation.

Note that 5 is the largest permissible value for the Pade approximation order in this implementation.

3.2.7. MODEL REFERENCE - DISTURBANCES

Reference: Section 3.9; page 3-20.

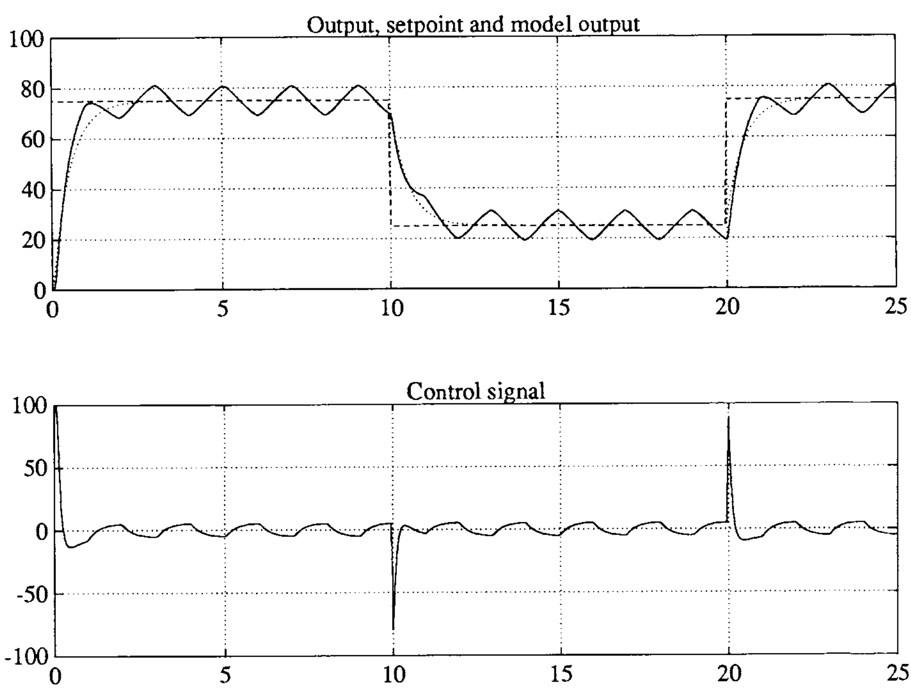

Figure 3.7. Model reference - disturbances

Description

This example is identical to example 3.2.1, except that a square wave disturbance of amplitude 5 units and period two units is added to the system input. The purpose of the example is to illustrate the role of the polynomial $C(s)$ in determining closed-loop disturbance rejection. Initially, $C(s)=0.5s+1$.

Programme interaction

```
runex 3 7
Example 3 of chapter 7: Model reference - disturbances

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval          =    0.050000 :=
 ======== Control action  ========
Automatic controller mode = TRUE :=
 ======== Assumed system  ========
A (system denominator)   =   1.000000   1.000000   0.000000 :=
B (system numerator)     =   1.000000  10.000000 :=
D (initial conditions)   =   0.000000   0.000000 :=
 ======== Emulator design ========
P (model denominator)    =   0.500000   1.000000 :=
C (emulator denominator) =   0.500000   1.000000 :=
------------------------------------
      System polynomials
------------------------------------
A         1.000000   1.000000   0.000000
B         1.000000  10.000000
D         0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+        1.000000  10.000000
B-        1.000000
C         0.500000   1.000000
P         0.500000   1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
------------------------------------
```

```
F            0.750000    1.000000
F filter     0.500000    1.000000
G            0.250000    2.500000
G filter     0.500000    1.000000
I
E            0.250000
ED           0.000000
-----------------------------------
 ======== Controller     ========
Maximum control signal   =  100.000000 :=
Minimum control signal   = -100.000000 :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
Square amplitude         =    5.000000 :=
Period                   =    2.000000 :=
 ======== Out Disturbance ========
Square amplitude         =    0.000000 :=
Period                   =   20.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time                = FALSE :=
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The effect of the disturbance is to perturb the system output; the control signal reacts to some extent to counteract this effect.

Further investigations

1. The emulator denominator $C(s)$ is of the form 1+Ts. Try the effect of varying the time constant T (try, for example, T=0.1) of the emulator denominator C. How does this affect the system output and the control signal?

2. Investigate the effect of an output disturbance on the control system.

3.2.8. MODEL REFERENCE PID

Reference: Section 3.10; page 3-24.

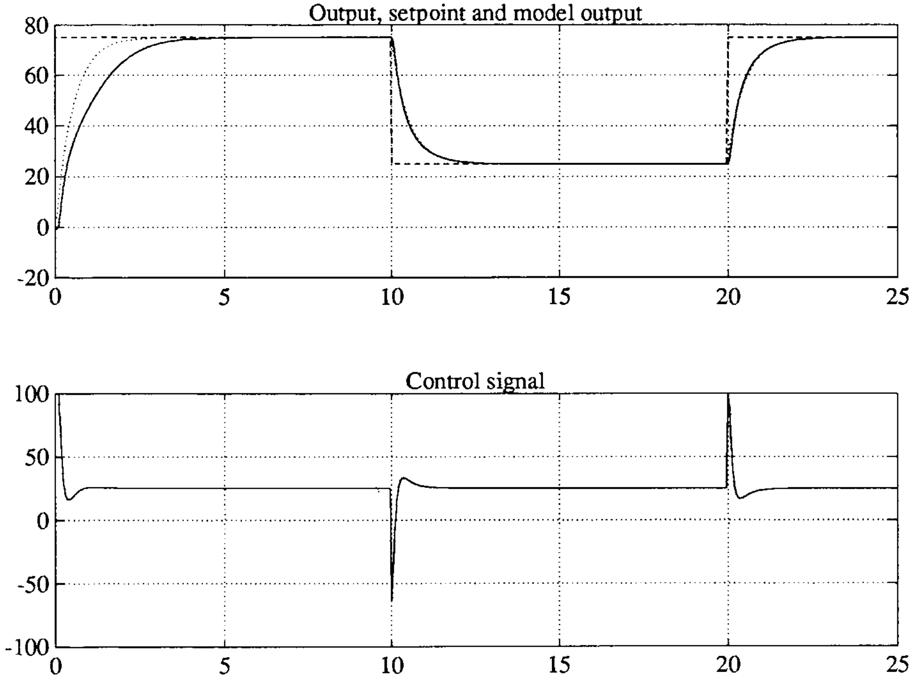

Figure 3.8. Model reference PID

Description

This example is identical to example 3.2.1 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral action" to "TRUE".

c) The degree of $C(s)$ is increased by one: $C(s) = (1+0.5s)^2$.

Programme interaction

```
runex 3 8
Example 3 of chapter 8:  Model reference PID

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters        ========
Sample Interval          =    0.050000  :=
 ======== Control action  ========
Integral action          = TRUE  :=
 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =    1.000000   10.000000  :=
D (initial conditions)    =    0.000000   0.000000  :=
 ======== Emulator design ========
P (model denominator)     =    0.500000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000 *  :=
Next factor ...
C (emulator denominator)  =    0.500000   1.000000  :=
-------------------------------------
      System polynomials
-------------------------------------
A          1.000000   1.000000    0.000000   0.000000
B          1.000000   10.000000   0.000000
D          0.000000   0.000000    0.000000
-------------------------------------
      Design polynomials
-------------------------------------
B+         1.000000   10.000000   0.000000
B-         1.000000
C          0.250000   1.000000    1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-------------------------------------
F          0.625000   1.500000    1.000000
F filter   0.250000   1.000000    1.000000
G          0.125000   1.250000    0.000000
G filter   0.250000   1.000000    1.000000
I
E          0.125000
ED         0.000000
```

```
-----------------------------------
 ======== Controller        ========
Maximum control signal   =  100.000000 :=
Minimum control signal   = -100.000000 :=
 ======== Simulation        ========
 ======== Setpoint          ========
 ======== In Disturbance  ========
Step amplitude              =  -25.000000 :=
 ======== Out Disturbance ========
Step amplitude              =    0.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time                 = FALSE :=
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The effect of the disturbance is, in the short term, to spoil the closed-loop response; but, in the long term, the response is not affected. Note that the steady-state control signal has a value of +25 to compensate for the disturbance. The controller has integral action.

Further investigations

1. Try the controller of example 1, but with the disturbance. (Set integral action to FALSE and set $C(s) = 0.5s+1$ by setting the second factor =1). What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input distubance. Explain what you observe.

3.2.9. POLE-PLACEMENT PID

Reference: Section 3.10; page 3-25.

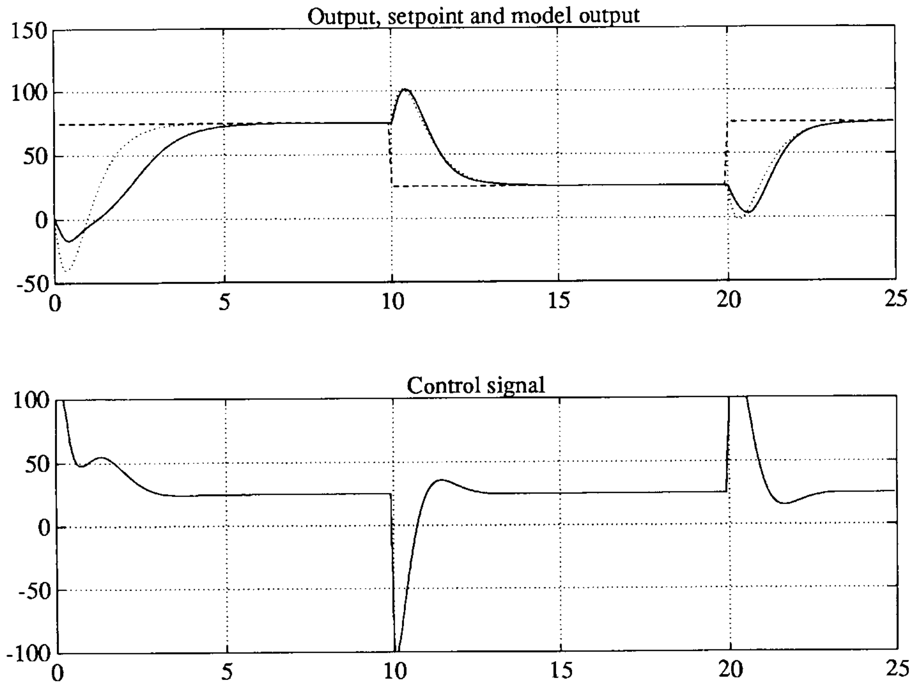

Figure 3.9. Pole-placement PID

Description

This example is identical to example 3.2.2 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral" action to "TRUE".

c) The degree of $C(s)$ is increased by one: $C(s) = (1{+}0.5s)^2$.

d) The sample interval is decreased to 0.01 to give a satisfactory approximation.

Programme interaction

runex 3 9
Example 3 of chapter 9: Pole-placement PID

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source     ========
======== Filters        ========
Sample Interval          =    0.010000  :=
======== Control action  ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =   -1.000000   1.000000  :=
======== Emulator design ========
Z has factor B            = TRUE  :=
P (model denominator)     =    0.250000   1.000000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000 *  :=
Next factor ...
C (emulator denominator)  =    0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   1.000000   0.000000   0.000000
B         -1.000000   1.000000   0.000000
D          0.000000   0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000   0.000000
B-        -1.000000   1.000000
C          0.250000   1.000000   1.000000
P          0.250000   1.000000   1.000000
Z+         1.000000
Z-        -1.000000   1.000000
Z-+        1.000000
-----------------------------------
F          2.031250   3.000000   1.000000
F filter   0.250000   1.000000   1.000000
G          0.062500   2.468750   0.000000
G filter   0.250000   1.000000   1.000000
I
E          0.062500   2.468750
ED         0.000000   0.000000
-----------------------------------
======== Controller      ========
Maximum control signal    =   100.000000  :=
Minimum control signal    =  -100.000000  :=
======== Simulation      ========
======== Setpoint        ========
======== In Disturbance  ========
```

Step amplitude = -25.000000 :=
======== Out Disturbance ========
Step amplitude = 0.000000 :=
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000
More time = FALSE :=

Discussion

The upper graph displays three signals: the system output $y(t)$,the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

The effect of the disturbance is, in the short term, to spoil the closed-loop response; but, in the long term, the response is not affected. Note that the steady-state control signal has a value of +25 to compensate for the disturbance: the controller has integral action.

Further investigations

1. Try the controller of example 3.2.1, but with the disturbance. (Set integral action to FALSE and set $C(s) = 0.5s+1$ by setting the second factor =1). What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input distubance. Explain what you observe.

3.2.10. DETUNED MODEL-REFERENCE

Reference: Section 3.11; page 3-28.

Description

The example on page I-3-28 illustrates the use of a reference model with one pole and one zero:

$$\frac{Z(s)}{P(s)} = \frac{0.03s+1}{0.3s+1} \tag{3.2.10.1}$$

together with control weighting:

$$Q(s) = \frac{qs}{0.03s+1} \tag{3.2.10.2}$$

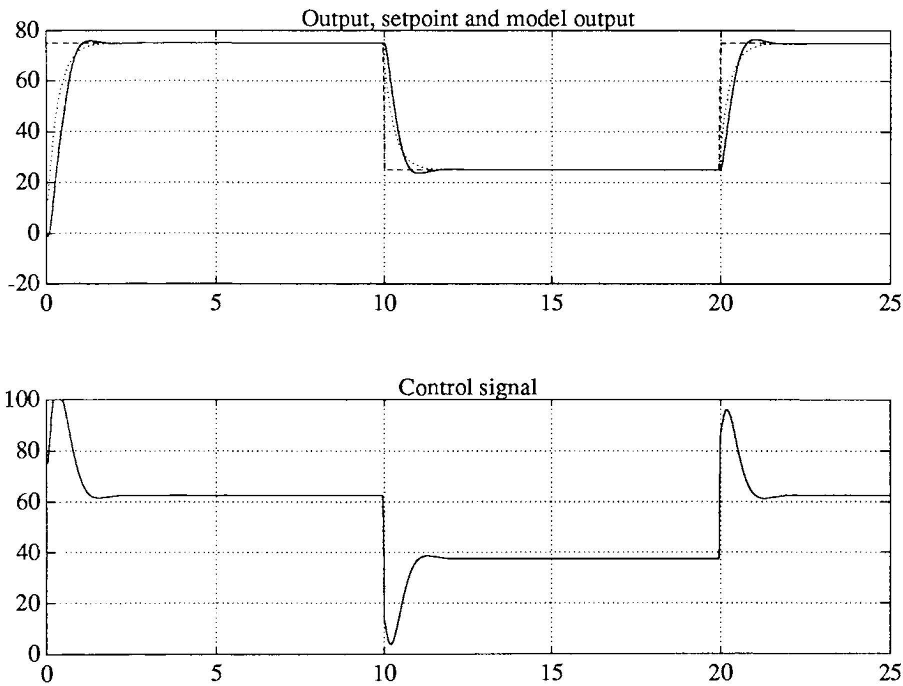

Figure 3.10. Detuned model-reference

In this example q=0.05 is used initially.

Programme interaction

runex 3 10
Example 3 of chapter 10: Detuned model-reference

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
Sample Interval = 0.050000 :=
======== Control action ========

```
 ======== Assumed system  ========
A (system denominator)    =    1.000000   1.000000  :=
B (system numerator)      =   2.000000  :=
 ======== Emulator design ========
Z-+ (Z- not including B)  =   0.030000   1.000000  :=
P (model denominator)     =    0.300000   1.000000  :=
C (emulator denominator)  =    0.300000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A           1.000000    1.000000    0.000000
B           2.000000    0.000000
D           0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+           2.000000    0.000000
B-          1.000000
C           0.300000    1.000000
P           0.300000    1.000000
Z+           1.000000
Z-          0.030000    1.000000
Z-+          0.030000    1.000000
-----------------------------------
F           0.494845    1.000000
F filter    0.300000    1.000000
G           0.150309    0.000000
G filter    0.009000    0.330000    1.000000
I
E           0.075155
ED
-----------------------------------
 ======== Controller     ========
Q numerator            =    0.100000   0.000000  :=
Q denominator          =    0.100000   1.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
Step amplitude         =  -25.000000  :=
 ======== Out Disturbance ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
```

Discussion

The model following is not exact due to the presence of the control weighting; but note that exact model following is not possible anyway as system with relative order $\rho=1$ cannot follow a model with relative order $\rho=0$.

Further investigations

1. Examine the effect of varying the parameter q.

3.2.11. PREDICTIVE CONTROL

Reference: Sections 3.7&8; page 3-18.

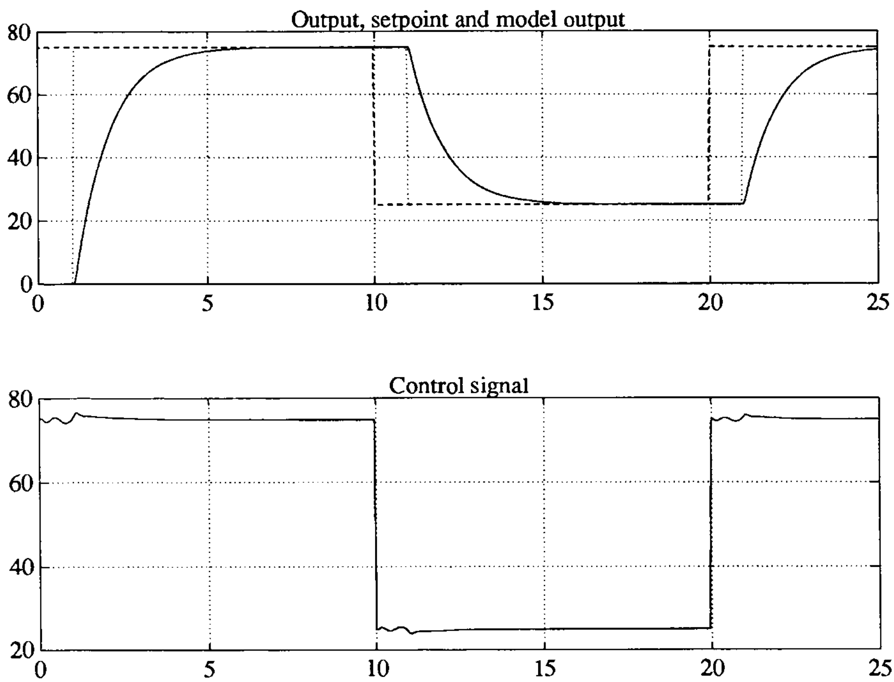

Figure 3.11. Predictive control

Description

A predictive emulator was designed in example 2.2.3. In this example, the emulator is embedded in a feedback loop to give predictive control. As discussed in section I-3.7, this is related to Smith's method.

The open loop system has a first order rational part with unit time constant ,together with a unit delay

$$e^{-sT} \frac{B(s)}{A(s)} = e^{-s} \frac{1}{1+s} \qquad (3.2.11.1)$$

$Q(s)$ is chosen to be an inverse PI controller:

$$\frac{1}{Q(s)} = 1 + \frac{1}{s} \qquad (3.2.11.2)$$

When the predictor is used, the nominal loop gain is:

$$L(s) = e^{s} \frac{1+s}{s} e^{-s} \frac{1}{1+s} = \frac{1}{s} \qquad (3.2.11.3)$$

giving a closed loop system setpoint response with the delay removed from the denominator:

$$\bar{y}(s) = e^{-s} \frac{1}{1+s} \bar{w}(s) \qquad (3.2.11.4)$$

Without the predictor, however, the nominal loop gain is

$$L(s) = e^{-s} \frac{1}{s} \qquad (3.2.11.5)$$

giving a closed loop system setpoint response:

$$\bar{y}(s) = e^{-s} \frac{1}{e^{-s}+s} \bar{w}(s) \qquad (3.2.11.6)$$

Programme interaction

runex 3 11
Example 3 of chapter 11: Predictive control

======== C S T C Version 6.0 ========

```
Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
Integral action          = FALSE  :=
 ======== Assumed system  ========
A (system denominator)   =    1.000000   1.000000  :=
B (system numerator)     =    1.000000  :=
Time delay               =    1.000000  :=
 ======== Emulator design ========
Z has factor B           = TRUE  :=
P (model denominator)    =    1.000000  :=
C (emulator denominator) =    1.000000  :=
Pade approximation order =           4  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    1.000000
B          1.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000
B-         1.000000
C          1.000000
P          1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
Pade       0.000595    0.011905    0.107143    0.500000    1.000000
-----------------------------------
F          0.367879
F filter   1.000000
G          0.000376    0.015908    0.051819    0.632121
G filter   0.000595    0.011905    0.107143    0.500000    1.000000
1
E          0.000376    0.015908    0.051819    0.632121
ED         0.000000    0.000000    0.000000    0.000000
-----------------------------------
 ======== Controller      ========
Q numerator              =    1.000000   0.000000  :=
Q denominator            =    1.000000   1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Step amplitude           =    0.000000  :=
```

Cos amplitude = *0.000000 :=*
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000

Discussion

The upper graph displays three signals: the system output $y(t)$,the setpoint $w(t)$ and the ideal model output $y_m(t) = y(t-1)$.

Note that the response is as predicted: a delayed first-order response delayed by one unit.

Further investigations

1. Try the effect of varying the order of the Pade approximation. Note that zero corresponds to having no predictor, and the response is not good. What is the smallest satisfactory order?
2. Try varying the system time delay. For each value of delay, find the minimum satisfactory Pade order. Note that for larger Pade orders, you may need to reduce the sample interval for numerical reasons.
3. Try putting integral action into the predictor (Integral action = TRUE, C = s+1) and use a Pade order of 3. Observe the performance with an output step disturbance, and compare to the integral-free case.
4. Add a sinusoidal disturbance to the system output; how does the performance depend on the amplitude of this signal and the system time-delay?

3.2.12. LINEAR-QUADRATIC POLE-PLACEMENT

Reference: Section 3.4; page 3-14.

Description

This example is identical to example 3.2.2, except that the closed-loop poles are chosen to solve equation I-3.4.23:

$$P(s)P(-s) = B(s)B(-s) + \lambda A(s)A(-s)$$

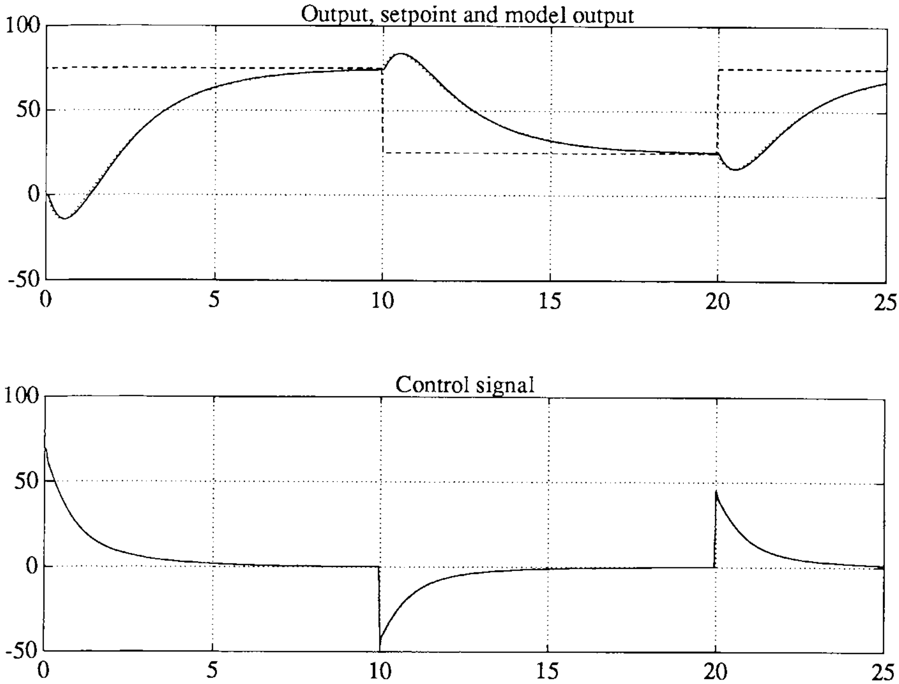

Figure 3.12. Linear-quadratic pole-placement

That is, the poles are chosen to correspond to those given by linear-quadratic optimisation theory where λ is the linear-quadratic weighting.

Programme interaction

runex 3 12
Example 3 of chapter 12: Linear-quadratic pole-placement

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========

```
======== Filters        ========
======== Control action ========
======== Assumed system ========
A (system denominator)  =   1.000000   1.000000   0.000000  :=
B (system numerator)    =  -1.000000   1.000000  :=
======== Emulator design ========
Z has factor B            = TRUE  :=
Linear-quadratic poles    = TRUE  :=
Linear-quadratic weight   =   1.000000  :=
C (emulator denominator)  =   0.500000   1.000000  :=
------------------------------------
     System polynomials
------------------------------------
A          1.000000    1.000000    0.000000
B         -1.000000    1.000000
D          0.000000    0.000000
------------------------------------
     Design polynomials
------------------------------------
B+         1.000000
B-        -1.000000    1.000000
C          0.500000    1.000000
P          1.000000    2.449490    1.000000
Z+         1.000000
Z-        -1.000000    1.000000
Z-+        1.000000
------------------------------------
F          1.112372    1.000000
F filter   0.500000    1.000000
G          0.500000    2.837117
G filter   0.500000    1.000000
I
E          0.500000    2.837117
ED         0.000000    0.000000
------------------------------------
======== Controller     ========
Maximum control signal   =  100.000000  :=
Minimum control signal   = -100.000000  :=
======== Simulation     ========
======== Setpoint       ========
======== In Disturbance ========
======== Out Disturbance ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

As in example 3.2.2 ,note the typical behaviour of a system with right-hand plane zeros: the output initially goes the wrong way in response to a step change.

The desired closed loop zeros are the same as in example 3.2.2, that is, the system zeros are unchanged, but the poles, and the step rise-time, now depend on $A(s)$, $B(s)$ and λ.

Further investigations

1. Try the effect of varying the linear-quadratic weighting λ. How does this affect the system output and the control signal?
2. Try repeating this example using the same system as example 3.2.1 ($B(s) = 10+s$). How does the closed-loop response when using linear-quadratic control differ from that when using model-reference control?

3.2.13. LINEAR-QUADRATIC PID

Reference: Section 3.4; page 3-14 and section 3.10; page 3-25.

Description

This example is identical to example 12 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral action" to "TRUE".

c) The degree of $C(s)$ is increased by one: $C(s) = (1+0.5s)^2$.

d) The sample interval is decreased to 0.01 to give a satisfactory approximation.

Programme interaction

runex 3 13
Example 3 of chapter 13: Linear-quadratic PID

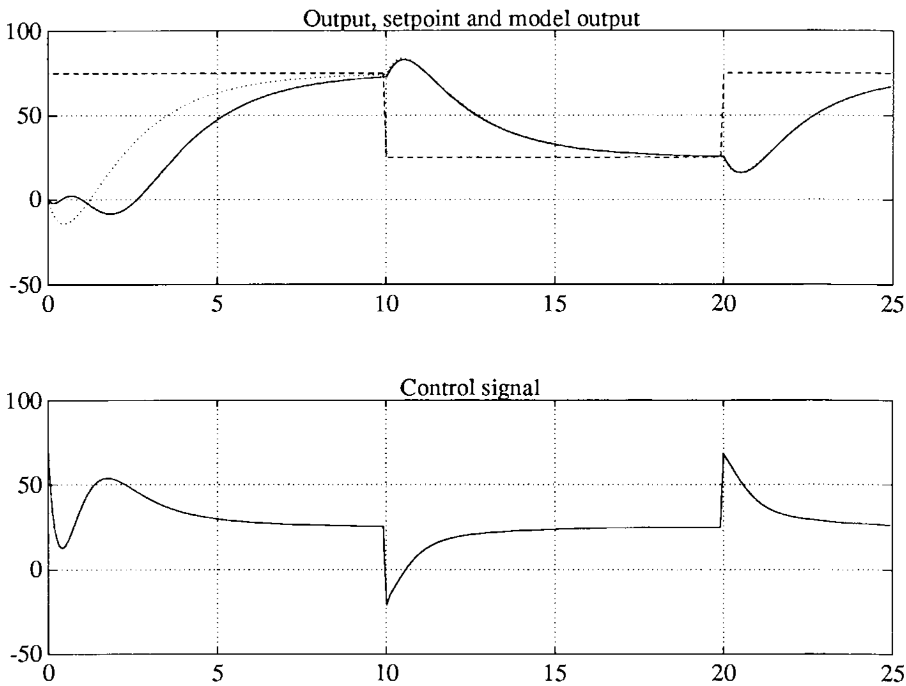

Figure 3.13. Linear-quadratic PID

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
Sample Interval = 0.010000 :=
======== Control action ========
Integral action = TRUE :=
======== Assumed system ========
A (system denominator) = 1.000000 1.000000 0.000000 :=
B (system numerator) = -1.000000 1.000000 :=
======== Emulator design ========
Z has factor B = TRUE :=
Linear-quadratic poles = TRUE :=
Linear-quadratic weight = 1.000000 :=
*C (emulator denominator) = 0.500000 1.000000 * :=*

```
Next factor ...
C (emulator denominator)  =     0.500000    1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000    1.000000    0.000000    0.000000
B         -1.000000    1.000000    0.000000
D          0.000000    0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000    0.000000
B-        -1.000000    1.000000
C          0.250000    1.000000    1.000000
P          1.000000    2.449490    1.000000
Z+         1.000000
Z-        -1.000000    1.000000
Z-+        1.000000
------------------------------------
F          3.393304    4.449490    1.000000
F filter   0.250000    1.000000    1.000000
G          0.250000    4.755676    0.000000
G filter   0.250000    1.000000    1.000000
I
E          0.250000    4.755676
ED         0.000000    0.000000
------------------------------------
 ======== Controller     ========
Maximum control signal    =  100.000000  :=
Minimum control signal    = -100.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
Step amplitude            =  -25.000000  :=
 ======== Out Disturbance ========
Step amplitude            =    0.000000  :=
Simulation running:
      25% complete
      50% complete
      75% complete
     100% complete
Time now is    25.000000
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

The effect of the disturbance is, in the short term, to spoil the closed-loop response; but, in the long term, the response is not affected. Note that the steady-state control signal has a value of +25 to compensate for the disturbance: the controller has integral action.

Further investigations

1. Try the controller of example 3.2.12, but with the disturbance. (Set integral action to FALSE and set $C(s) = 0.5s+1$ by setting the second factor =1). What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input distubance. Explain what you observe.

3.2.14. DISCRETE-TIME MODEL-REFERENCE CONTROL

Reference:

Description

Cstc can be used for simulation of discrete-time as well as continuous-time systems. This example considers the discrete-time time-delay system:

$$\frac{1}{z^4 - 2z^3 + z^2} = \frac{z^{-4}}{1 - 2z^{-1} + z^{-2}} \tag{3.2.14.1}$$

A minimum-variance type control stategy is used where the desired closed-loop model is:

$$\frac{Z}{P} = z^{-4} = \frac{1}{z^4} \tag{3.2.14.2}$$

Choosing the degree of C to one less than A (i.e.3) with all roots at the origin:

$$C = z^3 \tag{3.2.14.3}$$

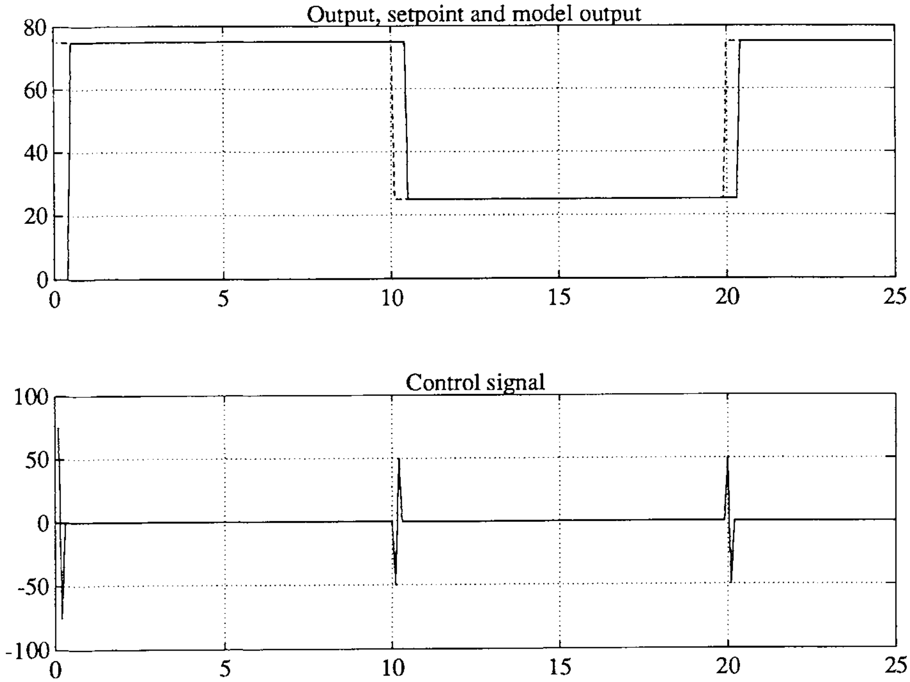

Figure 3.14. Discrete-time model-reference control

Programme interaction

runex 3 14
Example 3 of chapter 14: Discrete-time model-reference control

======== *C S T C Version 6.0* ========

Enter all variables (y/n, default n)?

======== *Data Source* ========
======== *Filters* ========
======== *Control action* ========
======== *Assumed system* ========
A (system denominator) = 1.000000 -2.000000 1.000000 0.000000 0.000000 :=
B (system numerator) = 1.000000 :=
======== *Emulator design* ========

```
Z+ (nice model numerator) =     1.000000 :=
P (model denominator)     =     1.000000  0.000000  0.000000  0.000000  0.000000 :=
C (emulator denominator)  =     1.000000  0.000000  0.000000  0.000000 :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000  -2.000000   1.000000   0.000000   0.000000
B          1.000000
D          0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000
B-         1.000000
C          1.000000   0.000000   0.000000   0.000000
P          1.000000   0.000000   0.000000   0.000000   0.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          5.000000  -4.000000   0.000000   0.000000
F filter   1.000000   0.000000   0.000000   0.000000
G          1.000000   2.000000   3.000000   4.000000
G filter   1.000000   0.000000   0.000000   0.000000
I
E          1.000000   2.000000   3.000000   4.000000
ED         0.000000   0.000000
-----------------------------------
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
 ======== Out Disturbance ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
```

Discussion

The upper graph displays three signals: the system output y_i and the setpoint w_i.

As requested, the system output follows the setpoint after a delay of four samples.

Further investigations

1. Try modifying the system to have a zero at z=-0.9:

$$B = z+0.9 \tag{3.2.14.4}$$

   Because the realtive order has now been reduced to 3, change P to z^{-3}. What is the effect on the output and the control signal?

2. Investigate the effect of moving two of the closed-loop poles to z=-0.9:

$$P = z^4 - 1.8z^3 + 0.81z^2 \tag{3.2.14.5}$$

   and set Z + = 0.01 to give unit steady-state gain. What is the effect on the output and the control signal?

3.2.15. DISCRETE-TIME POLE-PLACEMENT CONTROL

Reference:

Description

Cstc can be used for simulation of discrete-time as well as continuous-time systems. This example considers the discrete-time time-delay system:

$$\frac{z+0.9}{z^4 - 2z^3 + z^2} = \frac{z^{-4}}{1 - 2z^{-1} + z^{-2}} \tag{3.2.15.1}$$

A pole-placement type control stategy is used where the desired closed-loop model is:

$$\frac{Z}{P} = z^{-4}B = \frac{z+0.9}{z^4} \tag{3.2.15.2}$$

Choosing the degree of C to one less than A (i.e.3) with all roots at the origin:

$$C = z^3 \tag{3.2.15.3}$$

Programme interaction

runex 3 15
Example 3 of chapter 15: Discrete-time pole-placement control

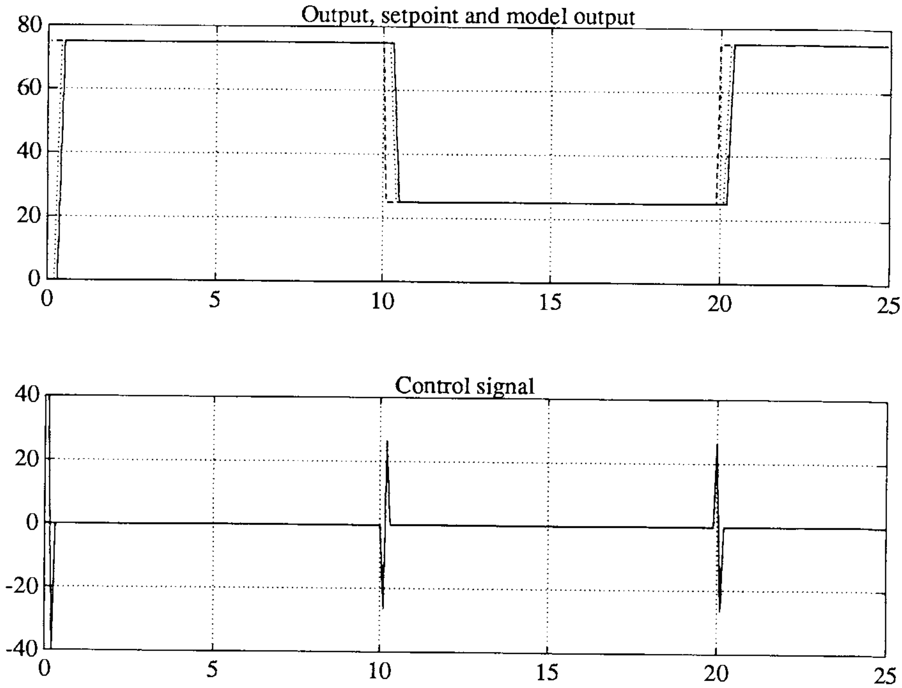

Figure 3.15. Discrete-time pole-placement control

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
======== Control action ========
======== Assumed system ========
A (system denominator) = 1.000000 -2.000000 1.000000 0.000000 0.000000 :=
B (system numerator) = 1.000000 0.900000 :=
======== Emulator design ========
Z has factor B = TRUE :=
Z+ (nice model numerator) = 1.000000 :=
P (model denominator) = 1.000000 0.000000 0.000000 0.000000 0.000000 :=
C (emulator denominator) = 1.000000 0.000000 0.000000 0.000000 :=

System polynomials

```
------------------------------------
A        1.000000  -2.000000  1.000000  0.000000  0.000000
B        1.000000   0.900000
D        0.000000   0.000000
------------------------------------
     Design polynomials
------------------------------------
B+       1.900000
B-       0.526316   0.473684
C        1.000000   0.000000  0.000000  0.000000
P        1.000000   0.000000  0.000000  0.000000  0.000000
Z+       1.000000
Z-       0.526316   0.473684
Z-+      1.000000
------------------------------------
F        4.473684  -3.473684  0.000000  0.000000
F filter 1.000000   0.000000  0.000000  0.000000
G        1.900000   3.800000  5.700000  3.126316
G filter 1.000000   0.000000  0.000000  0.000000
I
E        1.000000   2.000000  3.000000  1.645429
ED       0.000000   0.000000
------------------------------------
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
 ======== Out Disturbance ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
```

Discussion

The upper graph displays three signals: the system output y_i and the setpoint w_i.

Compared with further example 1 of the previous example, the control signal is not oscillatory: the zero at z=0.9 has not been cancelled.

Further investigations

1. Try modifying the system to have a zero at z=+0.9:

$$B = z - 0.9 \tag{3.2.15.4}$$

Why is the output now rather horrible? Recall that $z^{-4}10(z-0.9)$ was asked for.

CHAPTER 4

Non-Adaptive Robustness

Aims. To examine by simulation the effect of neglected dynamics on the performance of emulator based control.

4.1. IMPLEMENTATION DETAILS

The implementation is identical to that described in chapter 3. The neglected dynamics are introduced into the simulated system by including additional factors in the system polynomials. This approach is not very satisfactory from the numerical point of view and high precision floating point arithmetic is required.

4.2. EXAMPLES

4.2.1. DETUNED MODEL-REFERENCE CONTROL - NEGLECTED DYNAMICS

Reference: Section 4.7; page 4-11.

Description

This example is identical to example 3.2.10 except that neglected dynamics are included. As discussed in volume I, the system then corresponds to that of Rohrs. That is, the system is *assumed* to be

$$\frac{B(s)}{A(s)} = \frac{2b}{1+s} \tag{4.2.1.1}$$

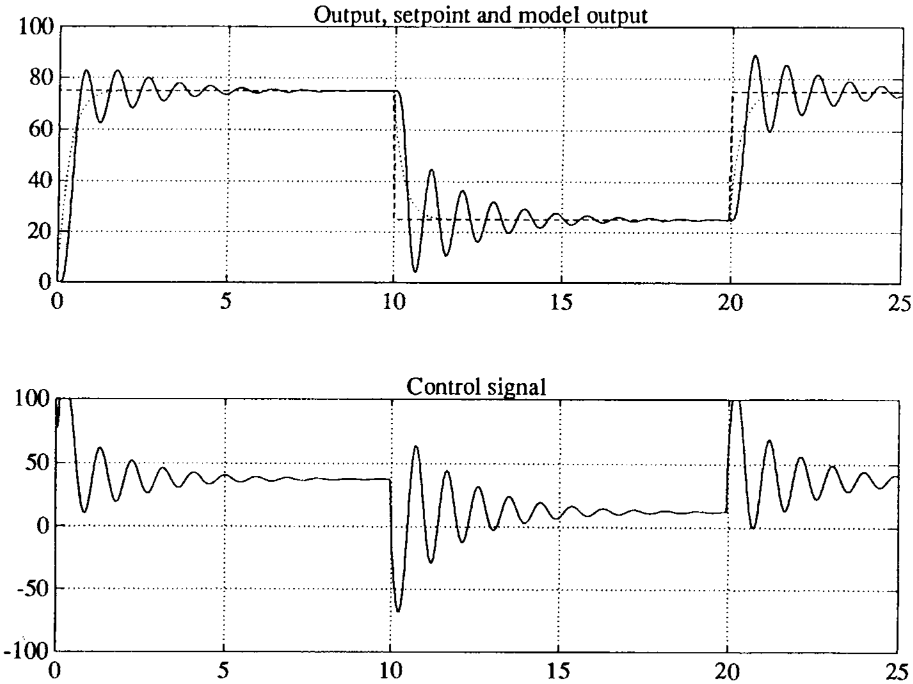

Figure 4.1. Detuned model-reference control - neglected dynamics

but is *actually* given by

$$N(s)\frac{B(s)}{A(s)} = \frac{100}{s^2 + 8s + 100}\ \frac{2}{1+s} = \frac{200}{s^3 + 9s^2 + 108s + 100} \tag{4.2.1.2}$$

In this example, b=1 and the control weighting is

$$Q(s) = \frac{qs}{0.03s+1} \tag{4.2.1.3}$$

with q = 0.05. This corresponds to the first row of the table on page I-4-16.

Programme interaction

```
runex 4 1
Example 4 of chapter 1:  Detuned model-reference control - neglected dynamics

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval          =    0.050000  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =    1.000000   1.000000  :=
B (system numerator)     =    2.000000  :=
 ======== Emulator design ========
P (model denominator)    =    0.300000   1.000000  :=
C (emulator denominator) =    0.300000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    1.000000    0.000000
B          2.000000    0.000000
D          0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         2.000000    0.000000
B-         1.000000
C          0.300000    1.000000
P          0.300000    1.000000
Z+         1.000000
Z-         0.030000    1.000000
Z-+        0.030000    1.000000
-----------------------------------
F          0.494845    1.000000
F filter   0.300000    1.000000
G          0.150309    0.000000
G filter   0.009000    0.330000    1.000000
I
E          0.075155
ED
-----------------------------------
 ======== Controller     ========
Q numerator          =    0.050000   0.000000  :=
Q denominator        =    0.030000   1.000000  :=
```

======== Simulation ========
======== Setpoint ========
======== In Disturbance ========
======== Out Disturbance ========
======== Actual system ========
Next factor ...
Next factor ...
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000

Discussion

The effect of the neglected dynamics is to give a rather nasty oscillatory closed-loop response; but, as predicted, the closed-loop system is stable. Note that, in this case, the notional feedback system is unstable, so the error term due to the emulator has a stabilising effect. See volume I for more discussions.

The adaptive case is more interesting, and is considered in chapter 7.

Further investigations

1. Try changing q to 0.2 as in the second row of the table on page I-4-16.
2. Try changing b to 0.5 ($B(s) = 1.0$) as in the third row of the table on page 4I--16. Does the closed-loop system verify the instability prediction? Note that the control signal is limited to the range -100 to +100.
3. Try changing b to 0.5 and q to 0.2 as in the fourth row of the table on page I-4-16.

CHAPTER 5

Least-Squares Identification

Aims. To describe the implementation of least-squares parameter estimation routines. To illustrate the behaviour of least-squares parameter estimation algorithms. To investigate, using a simple example, how least squares parameter estimation is affected by disturbance signals.

5.1. IMPLEMENTATION DETAILS

5.1.1. [5.2 & 6.3] LINEAR IN THE PARAMETERS SYSTEMS

As discussed in Vol. I, section 5.2, the standard linear in the parameters model to be used in this book is

$$\psi(t) = \underline{X}^T(t)\underline{\theta} + e(t) \tag{5.1.1.1}$$

where $\psi(t)$ is the scalar system output, $\underline{X}(t)$ is a column vector of measured variables and $\underline{\theta}$ a column vector of parameters and $e(t)$ the error.

As discussed Vol. I, section 6.3, for the purposes of system identification the special case:

$$y(t) = \underline{X}_s^T(t)\underline{\theta}_s + e_s(t) \tag{5.1.1.2}$$

where the **data vector** $\underline{X}_s(t)$ and the **parameter vector** $\underline{\theta}_s$ are given, in Laplace transform terms by

$$\overline{\underline{X}}_s(s) \triangleq \begin{bmatrix} \overline{\underline{X}}_i(s) \\ \overline{\underline{X}}_u(s) \\ \overline{\underline{X}}_y(s) \end{bmatrix}; \; \underline{\theta}_s = \begin{bmatrix} \underline{\theta}_i \\ \underline{\theta}_u \\ \underline{\theta}_y \end{bmatrix} \tag{5.1.1.3}$$

Where

$$\underline{\overline{X}}_u(s) = \frac{1}{C_s(s)}\begin{bmatrix} s^{n-1} \\ s^{n-2} \\ \cdot \\ 1 \end{bmatrix} e^{-sT}\,\overline{u}(s);\ \underline{\overline{X}}_y(s) = \frac{1}{C_s(s)}\begin{bmatrix} s^{n-1} \\ s^{n-2} \\ \cdot \\ 1 \end{bmatrix}\overline{y}(s) \tag{5.1.1.4}$$

$$\underline{\overline{X}}_i(s) = \frac{1}{C_s(s)}\begin{bmatrix} s^{n-1} \\ s^{n-2} \\ \cdot \\ 1 \end{bmatrix} \tag{5.1.1.5}$$

Note that the data vector has been reordered in CSTC as compared with that in Vol. I.

The data vector $\underline{\overline{X}}(s)$ is created in two stages with procedure **IdentifySystem:**

1. The data are filtered within procedure **Emulator** (which also gives $\hat{y}(t)$ based on the system parameters) and
2. the data are loaded into $\underline{\overline{X}}(s)$ using procedure **SetData.**

(There are also experimental procedures invoked when **IdentifyingDelay** is set to TRUE, but this is beyond the scope of this book).

Procedure **SetData** takes data from its input argument **State.** In this case, **SetData** is called with **State** replaced by **SysEmState,** of type **TypeEmState.** Amongst other things, this record contains: **ICState, uState** and **yState.** These three states contain the corresponding filtered data vectors: $\underline{\overline{X}}_i(s)$, $\underline{\overline{X}}_u(s)$ and $\underline{\overline{X}}_y(s)$. The purpose of **SetData** is to load the appropriate elements into the output argument **DataVector.** It does this by simply extracting the appropriate elements, incrementing the counter **j,** and loading into **DataVector.**

The integer variable **Integrating** is set to 1 if **IntegralAction** is set to TRUE, otherwise it is set to 0.

5.1.2. [5.7] DISCRETE-TIME PARAMETER ESTIMATION

CSTC does *not* use the continuous-time algorithm, but rather the discrete-time algorithm presented in section 5.7. But it is emphasised that *continuous-time* parameters are estimated; and that the discrete-time algorithm can be regarded as an approximation to the continuous-time algorithm.

Given the data vector discussed in the previous section, there are three stages to the algorithm within **IdentifySystem.**

1. The variable **EstimationError** is computed in the statement:

   EstimationError := yHat - y;

2. The least squares gain vector $\underline{S}_d{}^{-1})_m\underline{X}_m$ is updated in procedure **UpdateLeastSquaresGain.**

3. The parameters are updated in procedure **TuneEmulator.** Rather than update a parameter vector and then transfer the elements to the appropriate polynomials, the polynomial coefficients are updated directly. These polynomials are encapsulated in the record **Knobs** of type **TypeEmKnobs.**

5.2. EXAMPLES

5.2.1. ESTIMATION OF A 1ST ORDER SYSTEM

Reference: Section 5.2; pages 5-2 - 5-3.

Description

This example provides a simple introduction to parameter estimation with the noise-free system of the example on page I-5-2. with the following values:

$$a = 2;\ b = 3;\ c = 1;\ d = 4. \qquad (5.2.1.1)$$

The effects of initial variance, forgetting time and sample interval are investigated.

Programme interaction

runex 5 1
Example 5 of chapter 1: Estimation of a 1st order system

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter = 5 :=

======== Data Source ========
======== Filters ========
Sample Interval = 0.100000 :=
======== Control action ========
Automatic controller mode = FALSE :=

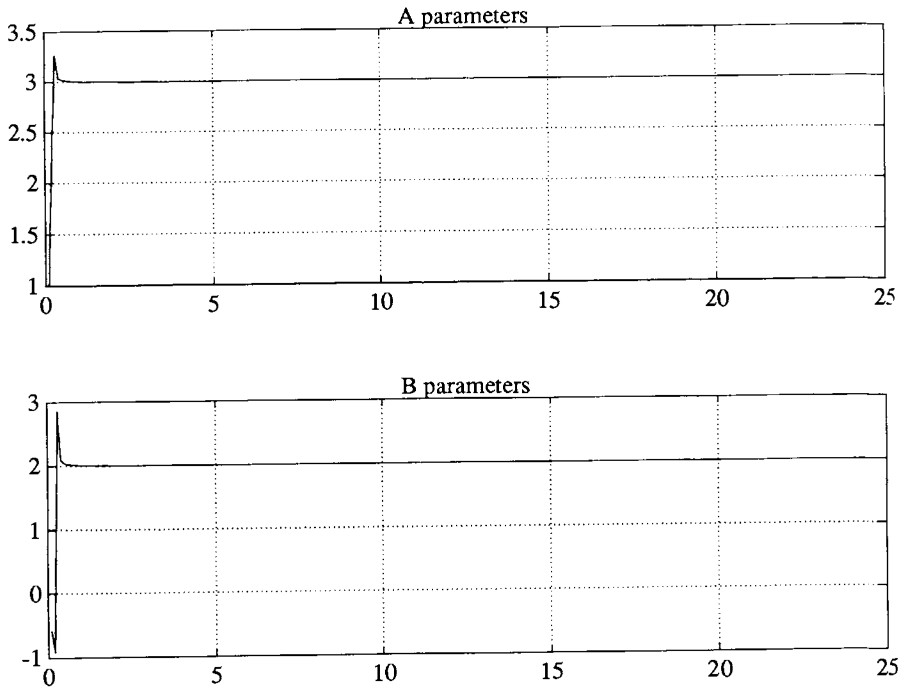

Figure 5.1. Estimation of a 1st order system

```
Integral action          = FALSE  :=
 ======== Assumed system  ========
A (system denominator)    =     1.000000    0.000000  :=
B (system numerator)      =     1.000000  :=
D (initial conditions)    =     0.000000  :=
Tuning initial conditions = TRUE  :=
 ======== Identification  ========
Initial Variance          = 100000.000000  :=
Forget time               = 1000.000000  :=
Dead band                 =     0.000000  :=
Cs (emulator denominator) =     1.000000    1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
Step amplitude            =    25.000000  :=
Square amplitude          =    25.000000  :=
Period                    =    10.000000  :=
 ======== In Disturbance  ========
```

```
======== Out Disturbance ========
======== Actual system   ========
A (system denominator)  =   1.000000   2.000000 :=
B (system numerator)    =   3.000000 :=
D (initial conditions)  =   4.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    2.000017
B          2.999816
D          3.846844
```

Discussion

The three system parameters rapidly converge to the correct values. This is because there is no noise and the initial variance is large. The initial condition parameter does not converge to as accurate a value; usually we are not interested in its value, it is just there to improve estimation of the system parameters. The statements can be examined by following the "further investigations".

The estimation error is non-zero; this is due to numerical innaccuracies in the implementation of the state-variable filter when step changes occur.

Further investigations

1. Try setting the Boolean variable 'Tuning initial conditions' to FALSE. How is the estimation of the other parameters affected?
2. Try using a smaller initial variance; for example use 1 instead of 100000. What happens to the rate of parameter convergence? Repeat for variance values of 0.1 and 10.0.
3. While using a small initial variance (for example 1.0) try the effect of using a small forget time (for example 10.0). What effect does this have on the rate of convergence? Repeat for forget times of 50 and 100. Note that the effect of noise is examined in a later example.
4. Try using a different sample interval (for example 0.5). How does this affect the estimation? Try some other values as well.

5. Repeat 2 but with different initial parameters. How does the choice of initial parameters affect the estimates for each value of initial variance?

6. Try setting the system emulator denominator $C(s)$ to be the same as the system denominator. Why is the estimation error now virtually zero?

5.2.2. THE EFFECT OF OUTPUT NOISE

Reference: Section 5.2; pages 5-2 - 5-3.

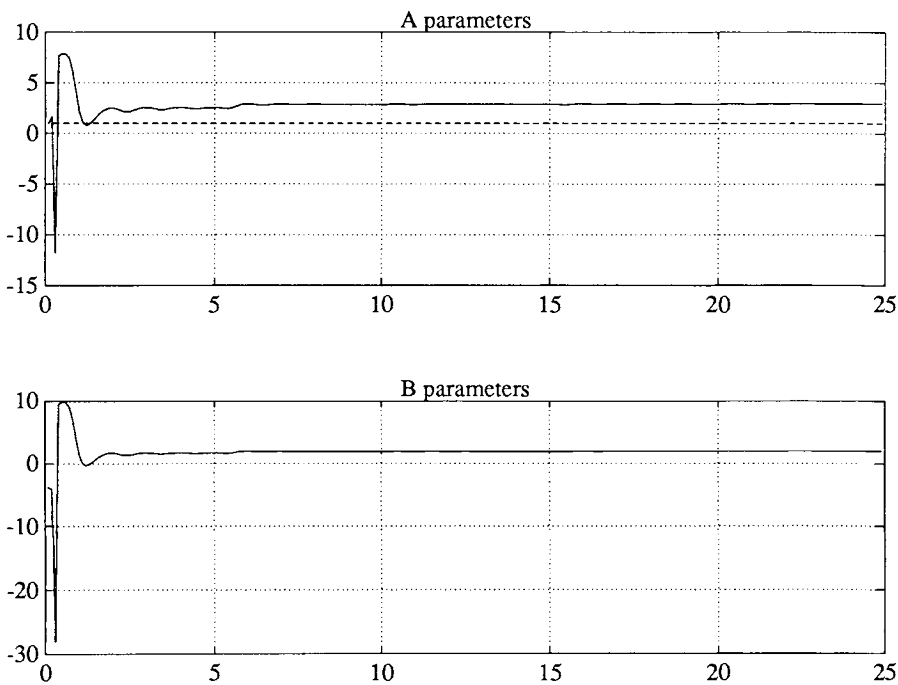

Figure 5.2. The effect of output noise

Description

This example is the same as example 5.2.1 except that a sinusoidal signal $0.25\sin 2\pi t$ is added to the output of the system; this may be thought of as measurement noise.

Programme interaction

```
runex 5 2
Example 5 of chapter 2: The effect of output noise

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval           =    0.100000  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000    0.000000  :=
B (system numerator)      =    1.000000  :=
D (initial conditions)    =    0.000000  :=
Tuning initial conditions = TRUE  :=
 ======== Identification  ========
Initial Variance          = 100000.000000  :=
Forget time               =  1000.000000  :=
Dead band                 =    0.000000  :=
Cs (emulator denominator) =    1.000000    1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Cos amplitude             =   10.000000  :=
Period                    =    1.000000  :=
 ======== Actual system   ========
A (system denominator)    =    1.000000    2.000000  :=
B (system numerator)      =    3.000000  :=
D (initial conditions)    =    4.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
```

```
------------------------------------
A          1.000000    1.959012
B          2.935745
D          9.797439
```

Discussion

The parameter estimates corresponding to $A(s)$ and $B(s)$ take longer to settle down as compared to the previous example; but they end up near to the correct values. The initial condition estimate is, however, completely spoiled by the noise.

Further investigations

1. Try increasing the amplitude of the sinusoidal measurement noise. What effect does this have on parameter convergence?
2. Try using a shorter forgetting time and/or a smaller initial variance. What effect does this have on parameter convergence?

5.2.3. THE EFFECT OF INPUT NOISE

Reference: Section 5.2; pages 5-2 - 5-3.

Description

This example is the same as example 5.2.1 except that a sinusoidal signal $0.25\sin 2\pi t$ is added to the input of the system; this may be thought of as a load disturbance.

Programme interaction

```
runex 5 3
Example 5 of chapter 3: The effect of input noise

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval         =    0.100000  :=
```

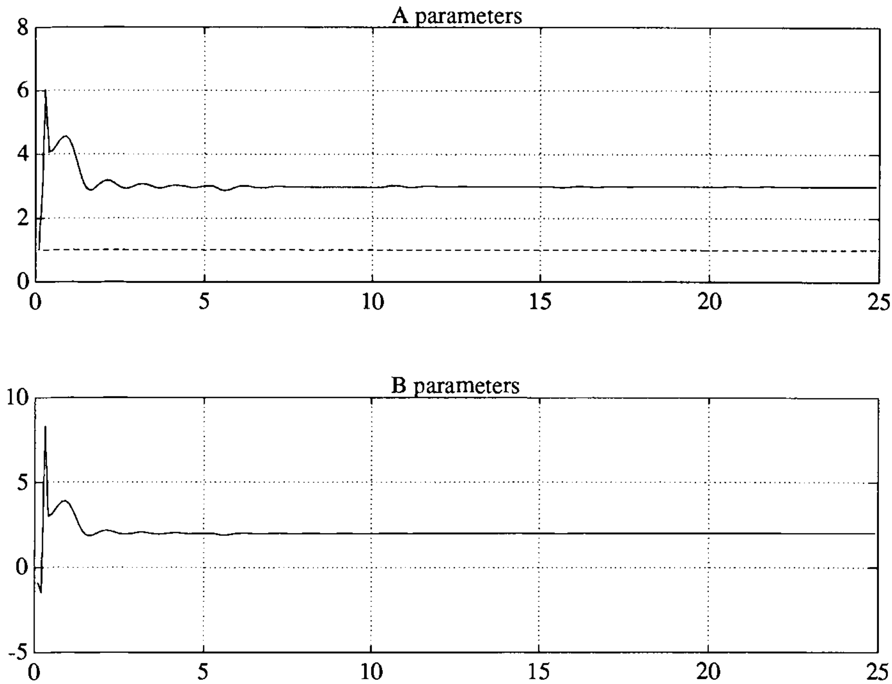

Figure 5.3. The effect of input noise

======== Control action ========
======== Assumed system ========
A (system denominator) = 1.000000 0.000000 :=
B (system numerator) = 1.000000 :=
D (initial conditions) = 0.000000 :=
Tuning initial conditions = TRUE :=
======== Identification ========
Initial Variance = 100000.000000 :=
Forget time = 1000.000000 :=
Dead band = 0.000000 :=
Cs (emulator denominator) = 1.000000 1.000000 :=
======== Simulation ========
======== Setpoint ========
======== In Disturbance ========
Cos amplitude = 10.000000 :=
Period = 1.000000 :=
======== Out Disturbance ========

```
======== Actual system   ========
A (system denominator)  =   1.000000  2.000000 :=
B (system numerator)    =  3.000000 :=
D (initial conditions)  =  4.000000 :=
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is   25.000000
------------------------------------
      System polynomials
------------------------------------
A          1.000000   1.995478
B          2.993337
D          8.910800
```

Discussion

The parameter estimates vary in a similar fashion to those of the previous example.

Further investigations

1. Try increasing the amplitude of the sinusoidal load disturbance. What effect does this have on parameter convergence?
2. Try using a shorter forgetting time and/or a smaller initial variance. What effect does this have on parameter convergence?

5.2.4. THE EFFECT OF AN OUTPUT OFFSET

Reference: Section 5.5; I-5-11&12

Description

This example is the same as example 5.2.1 except that a constant value of 0.5 is added to the output of the system; this may be thought of an offset due to scaling of variables or to linearisation about an operating point.

The idea of modelling a such an offset as the output of an integrator is discussed in section I-5.2 and section I-1.9. This is incorporated in this programme using the Boolean variable 'Integral action'; but note that, as on I-5-3, the order of $C(s)$ must be increased by 1.

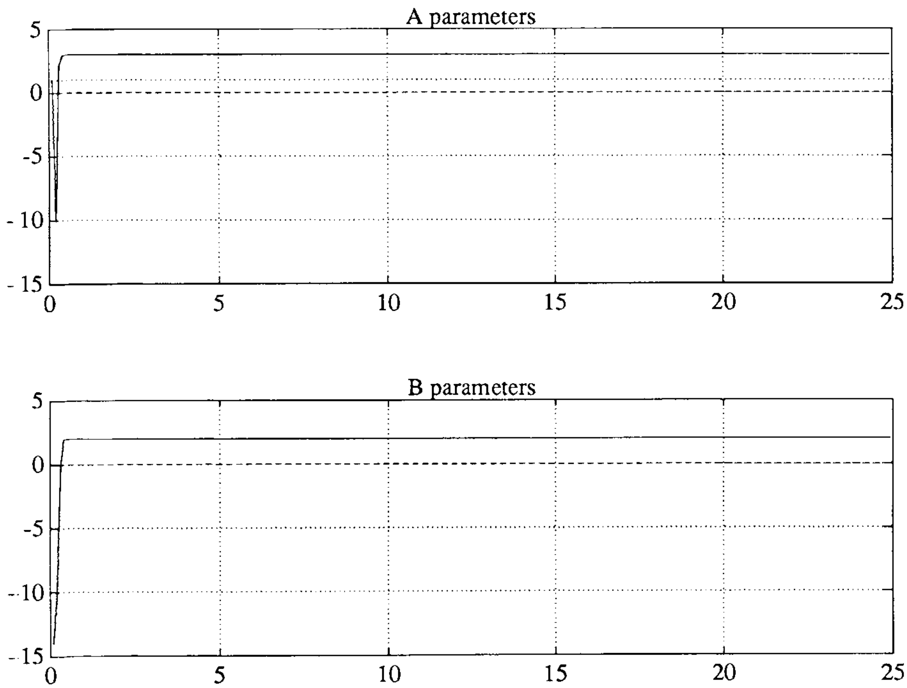

Figure 5.4. The effect of an output offset

Programme interaction

runex 5 4
Example 5 of chapter 4: The effect of an output offset

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter = 5 :=

======== Data Source ========
======== Filters ========
Sample Interval = 0.100000 :=
======== Control action ========
Automatic controller mode = FALSE :=
Integral action = TRUE :=

```
======== Assumed system ========
A (system denominator)   =   1.000000   0.000000 :=
B (system numerator)     =   1.000000 :=
D (initial conditions)   =   0.000000 :=
Tuning initial conditions = TRUE :=
======== Identification ========
Initial Variance         = 100000.000000 :=
Forget time              =  1000.000000 :=
Dead band                =     0.000000 :=
Cs (emulator denominator) =   1.000000   1.000000 * :=
Next factor ...
Cs (emulator denominator) =   1.000000   1.000000 :=
======== Simulation     ========
======== Setpoint       ========
Step amplitude           =   25.000000 :=
Square amplitude         =   25.000000 :=
Period                   =   10.000000 :=
======== In Disturbance ========
======== Out Disturbance ========
Step amplitude           =   25.000000 :=
======== Actual system  ========
A (system denominator)   =   1.000000   2.000000 :=
B (system numerator)     =   3.000000 :=
D (initial conditions)   =   4.000000 :=
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is   25.000000
-----------------------------------
     System polynomials
-----------------------------------
A        1.000000    1.998333    0.000000
B        2.997501    0.000000
D       28.951583   51.400217
```

Discussion

The parameter estimates converge rapidly. Note that there are now two initial condition terms as:

$$\frac{D(s)}{(s+2)} = \frac{4}{s+2} + \frac{25}{s} \tag{5.2.4.1}$$

giving

$$D(s) = 4s + 25(s+2) = 29s + 50 \tag{5.2.4.2}$$

Further investigations

1. Try the effect of not accounting for the offset. Do this by setting 'Integral action' to FALSE and changing $C(s)$ to 1+s. What is the effect on the parameter estimates?
2. Repeat 1 but with a shorter forgetting time. Why does this improve matters?

5.2.5. ESTIMATION OF A 4TH ORDER SYSTEM

Reference: Section 5.2; pages 5-2 - 5-3.

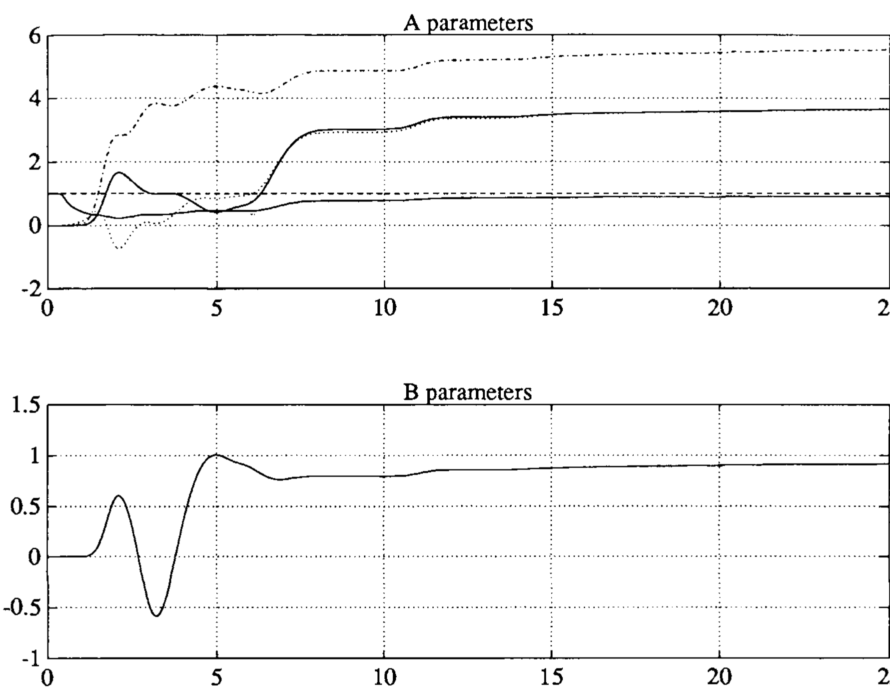

Figure 5.5. Estimation of a 4th order system

Description

This example corresponds to the 4th order system:

$$\frac{1}{(s+1)^4} = \frac{1}{s^4 + 4s^3 + 6s^2 + 4s + 1} \tag{5.2.5.1}$$

Programme interaction

```
runex 5 5
Example 5 of chapter 5:  Estimation of a 4th order system

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter                  =          5  :=

 ======== Data Source    ========
 ======== Filters        ========
Sample Interval          =   0.020000  :=
 ======== Control action ========
Automatic controller mode = FALSE  :=
Integral action          = FALSE  :=
 ======== Assumed system ========
A (system denominator)   =   1.000000   0.000000   0.000000   0.000000   0.000000  :=
B (system numerator)     =   1.000000  :=
D (initial conditions)   =   0.000000  :=
Tuning initial conditions = FALSE  :=
 ======== Identification ========
Initial Variance         =  100.000000  :=
Forget time              =  1000.000000  :=
Dead band                =   0.000000  :=
Cs (emulator denominator) =   0.500000   1.000000 *  :=
Next factor ...
Cs (emulator denominator) =   0.500000   1.000000 *  :=
Next factor ...
Cs (emulator denominator) =   0.500000   1.000000 *  :=
Next factor ...
Cs (emulator denominator) =   0.500000   1.000000  :=
Normalising Cs so that c0 = 1
Cs    1.000000   8.000000   24.000000   32.000000   16.000000
 ======== Simulation     ========
 ======== Setpoint       ========
Step amplitude           =   25.000000  :=
```

```
Square amplitude       =   25.000000 :=
Period                 =   10.000000 :=
======== In Disturbance ========
======== Out Disturbance ========
======== Actual system  ========
A (system denominator)   =   1.000000   1.000000 *  :=
Next factor ...
A (system denominator)   =   1.000000   1.000000 *  :=
Next factor ...
A (system denominator)   =   1.000000   1.000000 *  :=
Next factor ...
A (system denominator)   =   1.000000   1.000000  :=
B (system numerator)     =   1.000000  :=
D (initial conditions)   =   0.000000  :=
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is   25.020000
-----------------------------------
    System polynomials
-----------------------------------
A        1.000000    3.656695    5.542385    3.666092    0.918106
B        0.917702
D        0.000000
```

Discussion

The parameters all converge to their correct values, but this takes rather longer than for a first order system.

Further investigations

1. The assumed system denominator $B(s)$ is zero order. Repeat the estimation with $B(s) = s^3$ so that a third order numerator is estimated. What is the estimated $B(s)$ polynomial? How acccurate are the estimates now that the knowledge of the order of $B(s)$ has been removed? How small is the estimation error compared with the the previous case?
2. Try using a smaller initial variance; for example use 1 instead of 100000. What happens to the rate of parameter convergence? Repeat for variance values of 0.1 and 10.0.
3. While using a small initial variance (for example 1.0) try the effect of using a small forget time (for example 10.0). What effect does this have on the rate of convergence? Repeat for forget

times of 50 and 100.

4. Try using a different sample interval (for example 0.5). How does this affect the estimation? Try some other values as well.

5. Repeat 2 but with different initial parameters. How does the choice of initial parameters affect the estimates for each value of initial variance?

6. Try setting the system emulator denominator $C(s)$ to be the same as the system denominator. Why is the estimation error now virtually zero?

5.2.6. ESTIMATION OF A 5TH ORDER SYSTEM

Reference: Chapter 5.

Description

This example corresponds to the system:

$$\frac{0.0233s^4-0.1860s^3+0.6696s^2-1.2500s+1}{(0.0233s^4+0.1860s^3+0.6696s^2+1.2500s+1)(s+1)} \tag{5.2.6.1}$$

$$= \frac{0.0233s^4-0.1860s^3+0.6696s^2-1.2500s+1}{0.0233s^5+0.2093s^4+0.8556s^3+1.9196s^2+2.2500s+1}$$

$$= \frac{1.0000s^4-7.9828s^3+28.7382s^2-53.6481s+42.9185}{s^5+8.9828s^4+36.7210s^3+82.3863s^2+96.5665s+42.9185}$$

This is, in fact, a Pade approximation to the system:

$$\frac{e^{-2.5s}}{s+1} \tag{5.2.6.2}$$

Programme interaction

runex 5 6
Example 5 of chapter 6: Estimation of a 5th order system

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

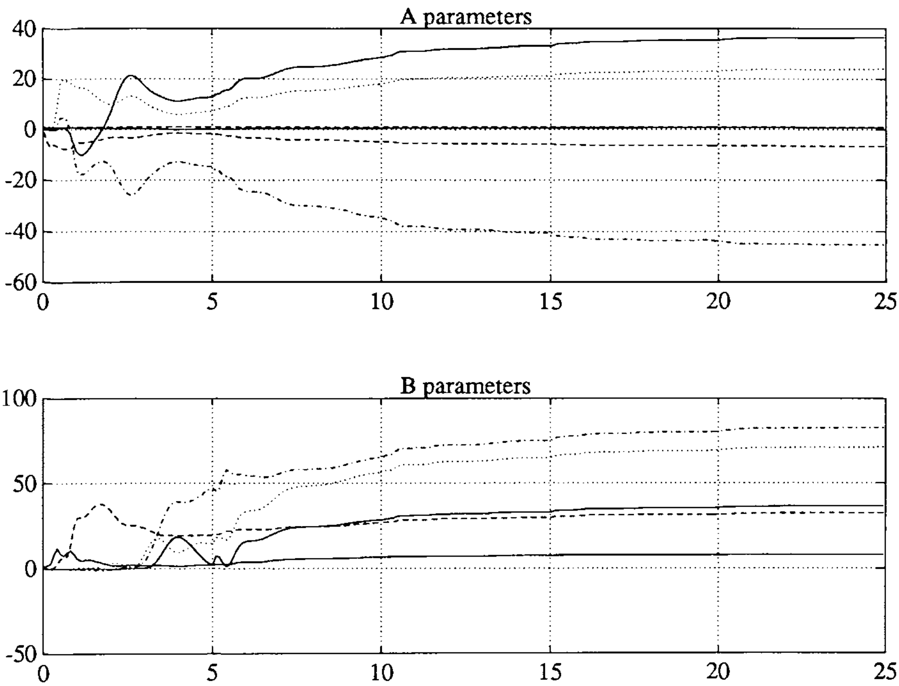

Figure 5.6. Estimation of a 5th order system

```
Chapter                    =          5  :=

 ======== Data Source     ========
 ======== Filters       ========
Sample Interval          =    0.010000  :=
 ======== Control action  ========
Automatic controller mode = FALSE  :=
Integral action          = FALSE  :=
 ======== Assumed system  ========
A (system denominator)    =     1.000000   0.000000   0.000000   0.000000   0.000000   0.000000  :=
B (system numerator)      =    1.000000   0.000000   0.000000   0.000000   0.000000  :=
D (initial conditions)    =    0.000000  :=
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance         =  100.000000  :=
Forget time              =  1000.000000  :=
Dead band                 =    0.000000  :=
```

```
Cs (emulator denominator) =    1.000000    1.000000 *  :=
Next factor ...
Cs (emulator denominator) =    1.000000    1.000000 *  :=
Next factor ...
Cs (emulator denominator) =    1.000000    1.000000 *  :=
Next factor ...
Cs (emulator denominator) =    1.000000    1.000000 *  :=
Next factor ...
Cs (emulator denominator) =    1.000000    1.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
Step amplitude          =   25.000000  :=
Square amplitude        =    25.000000  :=
Period                  =    10.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)    =    0.023300    0.186000    0.669600    1.250000    1.000000 *  :=
Next factor ...
A (system denominator)    =    1.000000    1.000000  :=
B (system numerator)      =    0.023300   -0.186000    0.669600   -1.250000    1.000000  :=
D (initial conditions)    =    0.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
------------------------------------
      System polynomials
------------------------------------
A          1.000000    8.007624    32.328589    71.309420    82.859892    36.619263
B          0.771162   -6.564503    23.979724   -45.324281    36.620029
D          0.000000
```

Discussion

To interpret the parameter estimates from this programme, it is important to realise that the estimates are *normalised* so that $A(s)$ is monic; that is, all coefficients are divided by the highest degree A coefficient:$a_0 = 0.0213$.

The parameters are not accurate, and the estimation error is not small. This is due to the poor approximation of a high order system by state-variable filter algorithm.

Further investigations

1. Try using a smaller initial covariance; for example use 1 instead of 100000. What happens to the rate of parameter convergence? What happens to the rate of estimation error convergence? Repeat for covariance values of 0.1 and 10.0.
2. While using a small initial variance (for example 1.0) try the effect of using a small forget time (for example 10.0). What effect does this have on the rate of convergence? Repeat for forget times of 50 and 100. Note that the effect of noise is examined in a later example.
3. Try using a different sample interval (for example 0.05). How does this affect the estimation? Try some other values as well.
4. Try setting the system emulator denominator $C(s)$ to be the same as the system denominator. Why is the estimation error now virtually zero?

5.2.7. ESTIMATION OF A TIME-VARYING SYSTEM

Reference: Section 5.2; pages 5-2 - 5-3.

Description

For the first 7.5 time units, the identified system is identical to that of example 1:

$$a = 2;\ b = 3;\ c = 1;\ d = 4. \qquad (5.2.7.1)$$

For the rest of the time, the system is given by:

$$a = 3;\ b = 4;\ c = 1 \qquad (5.2.7.2)$$

Thus an abrupt change in system parameters occurs at time 7.5.

The purpose of this example is to observe the behaviour of the least-squares algorithm when faced with time varying systems. In particular, the effect of the forgetting factor is investigated.

Programme interaction

runex 5 7
Example 5 of chapter 7: Estimation of a time-varying system

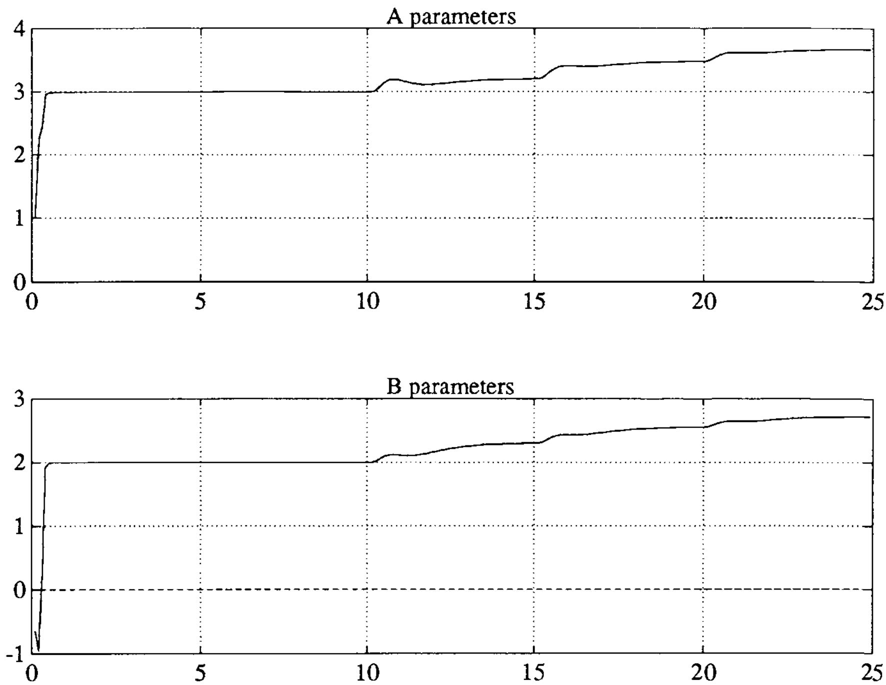

Figure 5.7. Estimation of a time-varying system

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter                =         5  :=

======== Data Source     ========
======== Filters        ========
Sample Interval          =     0.100000  :=
======== Control action  ========
Automatic controller mode = FALSE  :=
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)    =     1.000000    0.000000  :=
B (system numerator)      =     1.000000  :=
D (initial conditions)    =     0.000000  :=
Tuning initial conditions = TRUE  :=
======== Identification  ========
```

```
Initial Variance        =  100.000000 :=
Forget time             =  10.000000 :=
Dead band               =   0.000000 :=
Cs (emulator denominator) =   1.000000   2.000000   1.000000 :=
======== Simulation     ========
======== Setpoint       ========
Step amplitude          =  25.000000 :=
Square amplitude        =  25.000000 :=
Period                  =  10.000000 :=
======== In Disturbance ========
======== Out Disturbance ========
Cos amplitude           =   0.000000 :=
Period                  =   0.123000 :=
======== Actual system  ========
A (system denominator)  =   1.000000   2.000000 :=
B (system numerator)    =   3.000000 :=
D (initial conditions)  =   4.000000 :=
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is    7.600000
Extra time              =  17.500000 :=
======== Actual system  ========
A (system denominator)  =   1.000000   3.000000 :=
B (system numerator)    =   4.000000 :=
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is    25.000000
------------------------------------
     System polynomials
------------------------------------
A        1.000000    2.717764    0.000000
B        3.670209    0.000000
D       -9.043300   13.593639
```

Discussion

Initially, the three system parameters rapidly converge to the correct values. This is because there is no noise and the initial variance is large. After time 7.5, the estimates begin to move towards their new values. This is a slow process as the estimator is assuming a time invariant system.

Further investigations

1. Try using a smaller initial variance; for example,

   use 1. What happens to the rate of parameter convergence after the step parameter change? Repeat for variance values of 0.1 and 10.0.

2. Try using different forget times. How does the rate of parameter convergence after the step parameter change depend on the forget time?

3. As discussed in examples 2 and 3, noise adversely affects parameter convergence, particularly with small forget times. Investigate the effect of noise in this case.

5.2.8. DATA SPLICING

Reference: Chapter 5*

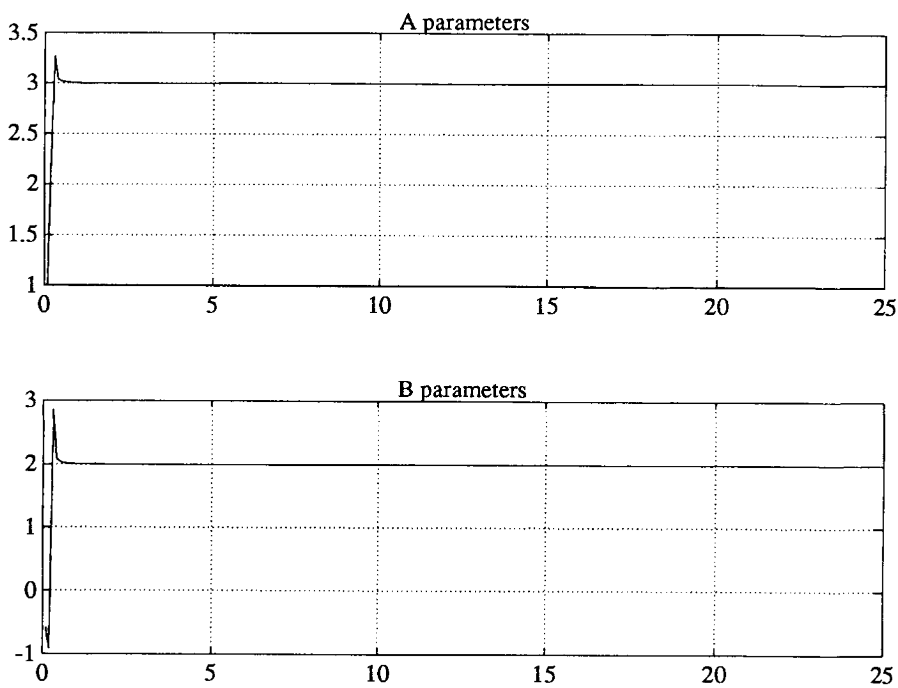

Figure 5.8. Data splicing

Description

There are a number of situations in which a number of data sets are available which, though arising from the same physical system, are not contiguous in time. Such data sets can be treated using the method of *data splicing**.

This example uses the 'outlog.dat' file from example 5.2.1, copied into 'indata.dat', and edited to remove all data from time 7.0 to time 12.0. This file is on the distribution disc as **indata.dat.** A blank line in this file marks the missing data, and the splicing procedure is invoked at this point.

Programme interaction

```
runex 5 8
Example 5 of chapter 8:  Data splicing

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source      ========
External data             = TRUE  :=
 ======== Real data        ========
 ======== Filters          ========
 ======== Control action   ========
Automatic controller mode = FALSE  :=
Integral action           = FALSE  :=
 ======== Assumed system   ========
A (system denominator)    =    1.000000   0.000000  :=
B (system numerator)      =    1.000000  :=
Tuning initial conditions = TRUE  :=
 ======== Identification   ========
Initial Variance          = 100000.000000  :=
Forget time               =  1000.000000  :=
Cs (emulator denominator) =    1.000000   1.000000  :=
Processing data in file ...
Splicing data
------------------------------------
      System polynomials
------------------------------------
```

*Gawthrop, P.J. (1984) "Parameter identification from non-contiguous data", Proceedings IEE, vol. 131 pt. D, No. 6, pp261-265.

A	*1.000000*	*1.999926*
B	*2.999672*	
D	*74.600965*	

Discussion

The graph is of the same format as for example 5.2.1. Data are missing beween times 7 and 12, and the plotting method used just joins up the graphs between these points.

Notice that the parameter estimates corresponding to $A(s)$ and $B(s)$ are entirely unaffected by the missing data; the values correspond to those in example 5.2.1. However, the initial condition estimate is quite different, as it is used in the data splicing procedure.

Further investigations

1 Investigate the effect of using the same data, but without the data splicing step. To do this, first put a copy of the **indata.dat** file in a safe place for later use. Then edit **indata.dat** and delete the blank line (just before the 12.0000 in the first column). Rerun the programme and observe the resultant parameter estimates - they are spoiled by the missing data.

2 Try out data splicing on some of the other examples. Run the relevant example, copy **outdata.dat** to **indata.dat** and delete one or more chunks of data. Leave a blank line to mark each block of missing data. Don't forget to make a safe copy of the original **indata.dat**.

5.2.9. ESTIMATION OF A DISCRETE-TIME SYSTEM

Reference: Section I-5.7; pages I-5-19&20

Description

This example is similar to example 1 except that the system is described in discrete-time. Note that all polynomials are now described in terms of z rather than s so that, in this example:

$$A(z) = z - 0.9;\ B(z) = 0.3;\ C_s(z) = z - 0.8$$

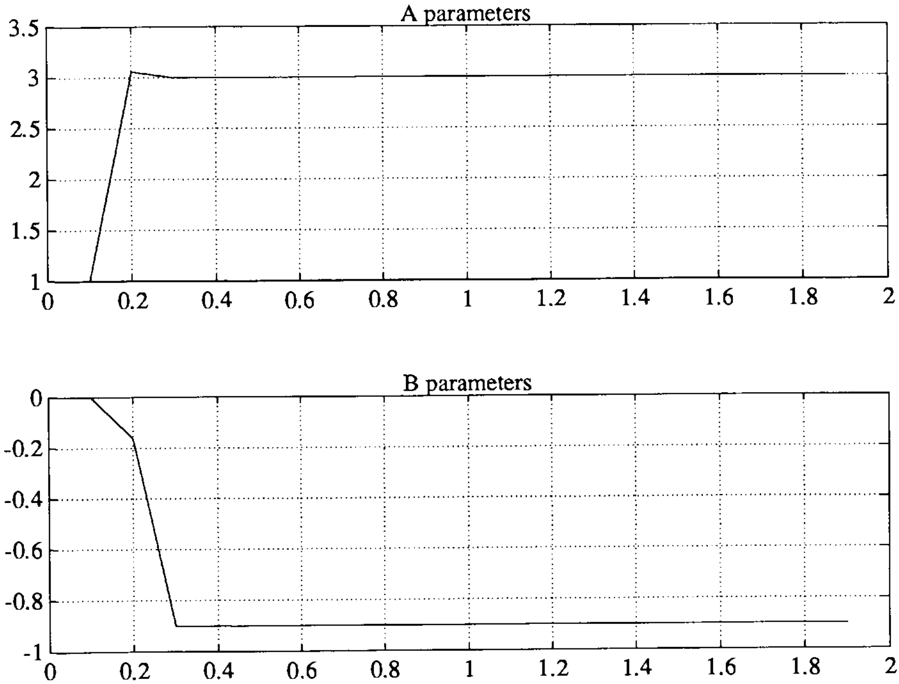

Figure 5.9. Estimation of a discrete-time system

Programme interaction

runex 5 9
Example 5 of chapter 9: Estimation of a discrete-time system

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter = 5 :=

======== Data Source ========
======== Filters ========
Sample Interval = 0.100000 :=
Continuous-time? = FALSE :=
======== Control action ========
Automatic controller mode = FALSE :=

```
Integral action          = FALSE  :=
 ======== Assumed system  ========
A (system denominator)   =   1.000000   0.000000  :=
B (system numerator)     =   1.000000  :=
D (initial conditions)   =   0.000000  :=
Tuning initial conditions = TRUE  :=
 ======== Identification  ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
Dead band                =   0.000000  :=
Cs (emulator denominator) =   1.000000  -0.800000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
Step amplitude           =  25.000000  :=
Square amplitude         =  25.000000  :=
Period                   =  10.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000  -0.900000  :=
B (system numerator)     =   3.000000  :=
D (initial conditions)   =   4.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is     2.000000
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   -0.900000
B          3.000000
D          3.999976
```

Discussion

The three estimated parameters converge to the correct value in three time steps. This is not surprising as the three simultaneous equations arising from the three sets of measurements are sufficient to derive the three unknown parameters.

Further investigations

1. Try changing the value of $C_s(z)$ to z. Does this make any difference? What is the effect of $C_s(z)$ on the filtered system input and output in this case?
2. Try discrete-time analogues of all the other continuous-time examples. The simplest way to do this is to run the appropriate continuous-time example and then edit the inlof.dat file to expose the 'Continuous-time' variable.

CHAPTER 6

Self-Tuning Control

Aims. To describe the implementation of the various self-tuning control algorithms in a unified manner. To provide a number of detailed examples comparing and contrasting the performance of the various algorithms.

6.1. IMPLEMENTATION DETAILS

6.1.1. [6.2] FEEDBACK CONTROL

As in chapter 3, the feedback controller is implemented by function **Control**. As discussed in that section, the control signal is computed by function **ImplicitSolution** if the transfer function $Q(s)$ has relative order zero, and by direct filtering if $\frac{1}{Q(s)}$ is strictly proper. In contrast to chapter 3, function **emulator** uses the *estimated* emulator parameters in place of the correct emulator parameters; thus the self-tuning controller is implemented as a self-tuning emulator in a feedback loop.

6.1.2. [6.4] EXPLICIT SELF-TUNING CONTROL

6.1.2.1. Off-line design

The off-line (a-priori) design phase

The emulator design parameters are chosen by the user during the preliminary interaction with CSTC. It is the user that makes the decision about the values of these polynomials. For example, $P(s)$ may be chosen to give a desired closed-loop response at low frequencies, and $Q(s)$ chosen to roll off the high-frequency gain. The choice of these parameters is discussed in detail in Volume I chapter 3.

The initial values of the polynomials $A(s)$ and $B(s)$ are set by the user in procedure **SystemInitialise** making use of procedure **EnterPolynomial.** This sets two things: the values of the coefficients of the polynomials used as initial values in the parameter estimator, and the degrees of the two polynomials.

The on-line tuning phase

This is accomplished within procedure **SelfTuningControl,** with **IdentifyingSystem** and **Explicit** both set to TRUE. Steps 1 and 2 (page I-6-12) are implemented within procedure **IdentifySystem** as discussed in chapter 5. Procedure **SetDesignKnobs** has no effect in this case.

Step 3 is implemented within procedure **DesignEmulator** using the procedures discussed in chapter 2.

Steps 4, 5 and 6 are implemented within procedure **Control.** Procedure **PutData** provides additional control signal modification. The corresponding parameters are contained within the record **PutDataKnobs** of type **TypePutDataKnobs.** The control signal is limited to be between **Max** and **Min.** If **Switched** is TRUE then a relay-type control is implemented.

The emulated signal **PhiHat** ($\hat{\phi}(t)$) is then generated using this modified control signal. This not only provides $\hat{\phi}(t)$ for control and identification at the next time instant, but also updates the corresponding emulator states.

Finally, if $Q(s)$ is of zero relative order, then **qState** is updated using **StateVariableFilter. This is because, in** a copy of **qState** , not the state itself, is updated within control procedure **ImplicitCon-**

trol.

6.1.2.2. On-line design

The off-line (a-priori) design phase

With reference to step 1, page I-6-13, two algorithms involving on-line design are implemented: pole-placement control and linear-quadratic control. In each case, **ZHasFactorB** is set to TRUE, and so $Z(s)$ depends on the estimated system numerator $B(s)$. If, in addition **LQ** is TRUE, then $P(s)$ is chosen on-line to satisfy equation 23 of section 3.4 of Volume I. The corresponding weight λ is set at this stage.

With reference to steps 2 and 3, page I-6-13, on line design of $Q(s)$ and $R(s)$ is not implemented in CSTC; these transfer functions are set to fixed values during the initialisation phase.

Step 4 is implemented during the initialisation phase when the $A(s)$ and $B(s)$ are set.

The on-line tuning phase

This is identical to that corresponding to the off-line design algorithm, except that step 3a is implemented within procedure **SetDesignKnobs**. **BMinus** is set equal to the estimated system numerator $B(s)$ except that if **ZeroAtOrigin** is TRUE, the corresponding term is transferred in to $B^+(s)$ as discussed in section 2.3 of Volume I.

If **LQ** is TRUE, $P(s)$ is designed using procedure **DesignP**. This calls the rather simple minded procedures **PolSquare** and **PolSqrt**. These compute $A(s)A(-s)$ from $A(s)$ and vice versa. They are only defined for first and second order polynomials.

6.1.3. [6.5] IMPLICIT SELF-TUNING CONTROL

As discussed in volume I, *implicit* self-tuning control involves direct tuning of the emulator parameters, thus avoiding the design phase taking estimated system parameters and deriving corresponding emulator parameters. As discussed in section I-6.5, the distinction between **off-line** and **on-line** design algorithms is made. The former class gives the simplest algorithms with straightforward implementation, the latter class gives rise to more complex algorithms.

6.1.3.1. Off-line design

The off-line (a-priori) design phase

Steps 1-4 (page I-6-17) are identical to those described in section 6.1.2.1 insofar as emulator design parameters are chosen by the user during the preliminary interaction with CSTC. There are two possibilities implemented for step 5: if the Boolean variable **UsingLambda** is TRUE then $\Lambda(s)$ is chosen according to equation I-6.5.1

$$\Lambda(s) = e^{-sT} \frac{Z(s)}{P(s)} \tag{6.1.3.1.1}$$

Otherwise $\Lambda(s) = 1$.

The on-line tuning phase

This is accomplished within procedure **SelfTuningControl,** with **IdentifyingSystem** and **Explicit** both set to FALSE.

If **UsingLambda** is TRUE then steps 1-3 are implemented within procedure **TuneLambdaEmulator.** Step 1 is implemented using the statement beginning:

```
PhiLambda := Filter(y, PLambda, ZLambda,
```

As **TuneLambdaEmulator** is with both **PLambda** and **ZLambda** equal to unit polynomials, this is equivalent to setting **PhiLambda** to **y.** The more general form is implemented to allow further research. Step 2 is implemented by the two statements beginning:

```
uLambda := DelayFilter(u, LambdaNumerator,
yLambda := DelayFilter(y, LambdaNumerator,
```

where **uLambda** and **yLambda** are the filtered versions of $u(t)$ and $y(t)$. Step 3 is implemented using the statement beginning:

```
PhiLamHat := Emulator(yLambda, uLambda,
```

where **PhiLamHat** is the corresponding emulator output and the emulator state vector is updated and saved in **LambdaEmState.** This information is extracted and put into **DataVector** ($\underline{X}_\Lambda(t)$) using the statement:

Step 4, the least-squares estimation, is accomplished using the statements

```
EstimationError := PhiLamHat - PhiLambda;
UpdateLeastSquaresGain(TunerState, TunerKnobs,
                       DataVector);
TuneEmulator(EmKnobs, TunerState);
```

See chapter II-5 for more details. Steps 5 and 6 are implemented as described in section 6.1.2.

If, on the other hand, **UsingLambda** is FALSE then step 1 is implemented within procedure **TunePhiEmulator.** As $\Lambda(s) = I$, $\phi_\Lambda(t) = \phi(t)$ and is generated by the statement

```
Phi := Filter(y, P, Z, FilterKnobs, PhicState);
```

Step 2 is not relevant here. Step 3 is implemented by using the statement

```
SetData(DataVector, EmState, EmKnobs);
```

to copy the information in the emulator state, **EmState** to **DataVector** ($\underline{X}_\Lambda(t)$). Step 4, the least-squares estimation, is accomplished in a similar fashion to **TuneLambdaEmulator,** and steps 5 and 6 are implemented as described in section 6.1.2.

6.1.3.2. On-line design

The off-line (a-priori) design phase

An in the explicit case, two algorithms involving on-line design are implemented: pole-placement control and linear-quadratic control. Steps 1-4 (page I-6-18&19) are implemented as for off-line design.

The design rule referred to in the additional step 5 (page I-6-19) is chosen according to equation 2 on page I-6-19:

$$\Lambda(s) = e^{-sT} \frac{Z(s)}{P(s)} \tag{6.1.3.2.1}$$

The on-line tuning phase

This is accomplished within procedure **SelfTuningControl.** As the method is implicit, the Boolean variable **Explicit** set to FALSE. However, an estimate of the polynomial $B(s)$ is need for the on-line design so the Boolean variable **IdentifyingSystem** set to TRUE. Steps 1 and 2 (page I-6-19) are implemented within procedure **IdentifySystem** as discussed in chapter 5. Step 3 is implemented by procedure **SetDesignKnobs** as discussed in chapter 2. Steps 4 and 5 are not implemented in CSTC. Step 6 is implemented automatically as $\Lambda(s)$ is expressed directly in terms of $P(s)$ ands $Z(s)$. Steps 7-12 are then identical to steps 1-6 of the off-line design method described in section 6.1.3.1.

6.2. EXAMPLES

6.2.1. EXPLICIT MODEL REFERENCE

Reference: Section 6.4; page 6-11. Section 3.4; page 3-12.

Description

This is the self-tuning equivalent of example 3.2.1 using the explicit approach with off-line choice of emulator design parameters.

The aim of the controller is to make the system output follow the model:

$$\bar{y}(s) = \frac{Z(s)}{P(s)}\bar{w}(s) \qquad (6.2.1.1)$$

where, in this case, $Z(s)=1$ and $P(s) = 1+Ts$ where the model time-constant T = 0.5.

The system parameters are estimated and the corresponding emulator parameters evaluated at each time step.

Programme interaction

runex 6 1
Example 6 of chapter 1: Explicit model reference

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

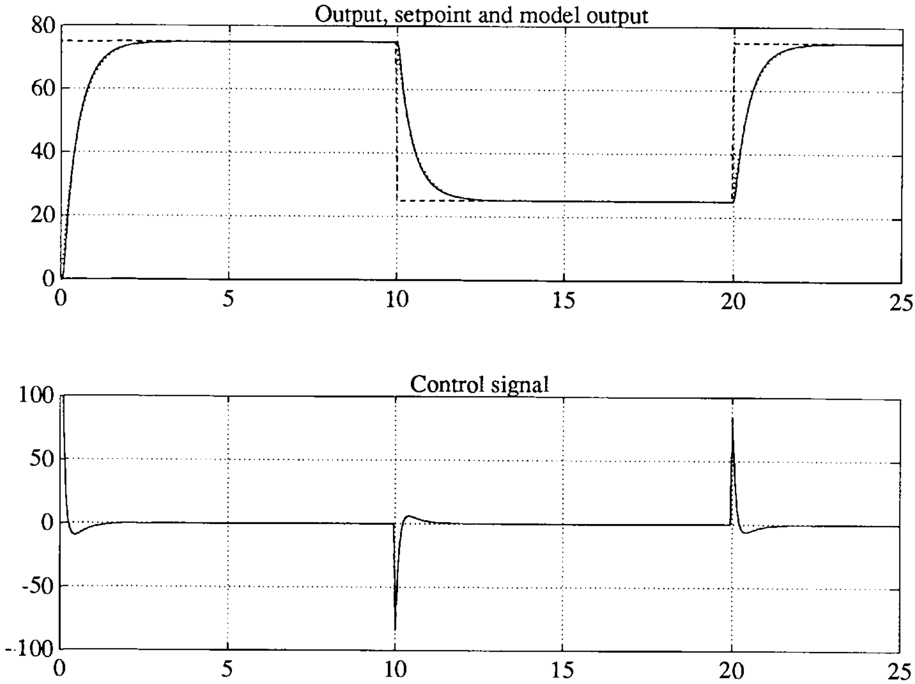

Figure 6.1. Explicit model reference

======== Data Source ========
======== Filters ========
======== Control action ========
Integral action = FALSE :=
======== Assumed system ========
A (system denominator) = 1.000000 0.000000 0.000000 :=
B (system numerator) = 1.000000 1.000000 :=
======== Emulator design ========
P (model denominator) = 0.500000 1.000000 :=
C (emulator denominator) = 0.500000 1.000000 :=

System polynomials

A	*1.000000*	*0.000000*	*0.000000*
B	*1.000000*	*1.000000*	
D	*0.000000*	*0.000000*	

```
      Design polynomials
------------------------------------
B+        1.000000   1.000000
B-        1.000000
C         0.500000   1.000000
P         0.500000   1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
------------------------------------
F         1.000000   1.000000
F filter  0.500000   1.000000
G         0.250000   0.250000
G filter  0.500000   1.000000
I
E         0.250000
ED        0.000000
------------------------------------
 ======== STC type        ========
 ======== Identification  ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
 ======== Controller      ========
Switched control signal  = FALSE  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000   1.000000   0.000000  :=
B (system numerator)     =   1.000000  10.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
------------------------------------
      System polynomials
------------------------------------
A         1.000000   1.000132   -0.000003
B         0.997695  10.001276
D         0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+        0.997695  10.001276
B-        1.000000
C         0.500000   1.000000
```

P	*0.500000*	*1.000000*
Z+	*1.000000*	
Z-	*1.000000*	
Z-+	*1.000000*	

F	*0.749967*	*1.000001*
F filter	*0.500000*	*1.000000*
G	*0.249424*	*2.500319*
G filter	*0.500000*	*1.000000*
I		
E	*0.250000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

After a short time, the output follows the model output closely despite the initially incorrect parameters.

Further investigations

1. Try the effect of varying the time constant T of the inverse model P. How does this affect the system output and the control signal?
2. The emulator denominator $C(s)$ is also of the form 1+Ts. Try the effect of varying the time constant T of the emulator denominator C. How does this affect the system output and the control signal?
3. Try changing the limits of the control signal so that it is clipped; for example choose 'Maximum control signal' as 10 and 'Minimum control signal' as -10. How does this affect the system output and the control signal?
4. The controller and simulation are implemented as discrete-time systems. Try the effect of varying the sample interval on closed-loop performance.
5. Try using a switched controller by setting 'Switched control signal' to TRUE. How does the performance depend on:

a) Sample interval

b) The maximum and minimum control limits.

6.2.2. EXPLICIT POLE-PLACEMENT CONTROL

Reference: Section 6.4; page 6-11. Section 3.4; page 3-13.

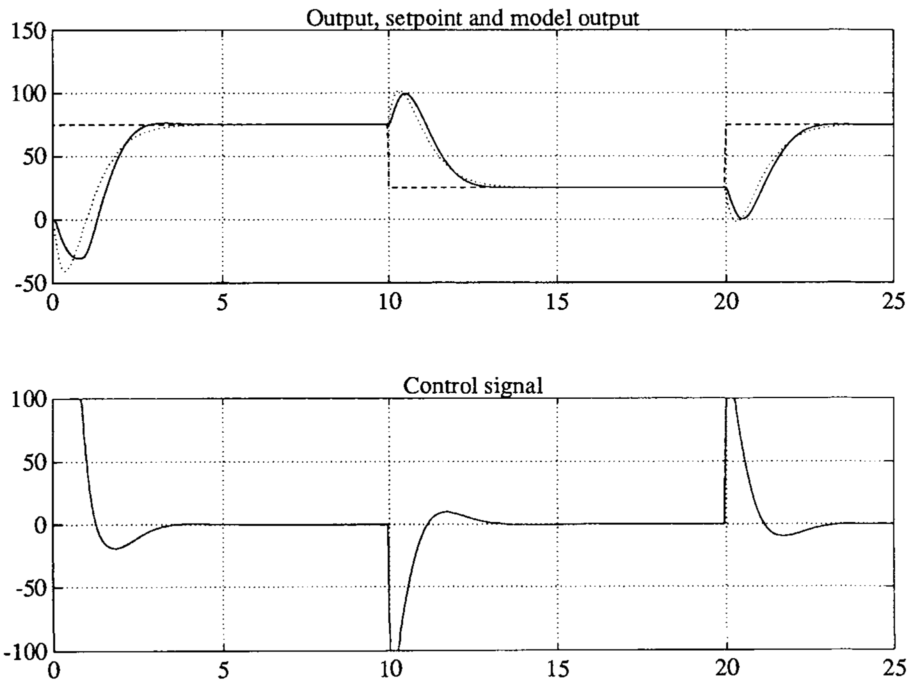

Figure 6.2. Explicit pole-placement control

Description

As discussed in volume I, the emulator designed in the second example of section I-2.4 may be embedded in a feedback loop to give pole-placement control.

The aim of the controller is to make the system output follow the model:

$$\bar{y}(s) = \frac{Z(s)}{P(s)}\bar{w}(s) \tag{6.2.2.1}$$

where, in this case, $Z(s) = B(s)$ and $P(s) = (1+Ts)^2$ where the model time-constant T = 0.5.

Programme interaction

```
runex 6 2
Example 6 of chapter 2:  Explicit pole-placement control

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source      ========
 ======== Filters          ========
 ======== Control action   ========
 ======== Assumed system   ========
A (system denominator)   =   1.000000   0.000000   0.000000  :=
B (system numerator)     =   1.000000   1.000000  :=
 ======== Emulator design ========
Z has factor B           = TRUE  :=
P (model denominator)    =   0.500000   1.000000 *  :=
Next factor ...
P (model denominator)    =   0.500000   1.000000  :=
C (emulator denominator) =   0.500000   1.000000  :=
-----------------------------------
     System polynomials
-----------------------------------
A        1.000000   0.000000   0.000000
B        1.000000   1.000000
D        0.000000   0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+       1.000000
B-       1.000000   1.000000
C        0.500000   1.000000
P        0.250000   1.000000   1.000000
Z+       1.000000
Z-       1.000000   1.000000
Z-+      1.000000
-----------------------------------
F        0.500000   1.000000
```

```
F filter    0.500000    1.000000
G           0.125000    0.250000
G filter    0.500000    1.000000
I
E           0.125000    0.250000
ED          0.000000    0.000000
-----------------------------------
 ======== STC type        ========
Tuning initial conditions = FALSE :=
 ======== Identification  ========
Initial Variance          = 100000.000000 :=
Forget time               =  1000.000000 :=
Cs (emulator denominator) =    1.000000   2.000000   1.000000 :=
 ======== Controller      ========
Switched control signal  = FALSE :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000   1.000000   0.000000 :=
B (system numerator)     =   -1.000000   1.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
-----------------------------------
      System polynomials
-----------------------------------
A            1.000000    0.999689    0.000009
B           -0.999991    0.999695
D            0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+           0.999695
B-          -1.000297    1.000000
C            0.500000    1.000000
P            0.250000    1.000000    1.000000
Z+           1.000000
Z-          -1.000297    1.000000
Z-+          1.000000
-----------------------------------
F            0.937725    0.999985
F filter     0.500000    1.000000
G            0.124962    1.562565
G filter     0.500000    1.000000
```

I		
E	*0.125000*	*1.563042*
ED	*0.000000*	*0.000000*

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

In this case, note the typical behaviour of a system with right-hand plane zeros: the output initially goes the wrong way in response to a step change. The model output $\overline{y}_m(s)$ is not exactly followed; this is because the control signal is limited.

Further investigations

1. Try the effect of varying the time constant T of the inverse model P. How does this affect the system output and the control signal?
2. Try repeating this example using the same system as the previous section ($B(s) = 10+s$). How does the closed-loop response when using pole-placement differ from that when using model-reference control?
3. Try changing the control limits. How is the response changed?

6.2.3. USING A SETPOINT FILTER

Reference: Section 6.4; page 6-11. Section 3.5; page 3-15.

Description

This example is identical to example 1 except that a setpoint filter is added:

$$wRs = R(s)\overline{w}(s);\ R(s) = \frac{0.5s+1}{s^2 + \sqrt{2}s + 1} \tag{6.2.3.1}$$

The closed loop response is thus:

$$\overline{y}(s) = \frac{Z(s)}{P(s)}R(s)\overline{w}(s) = \frac{1}{0.5s+1}\,\frac{0.5s+1}{s^2+\sqrt{2}s+1}\overline{w}(s) = \frac{1}{s^2+\sqrt{2}s+1}\overline{w}(s) \tag{6.2.3.2}$$

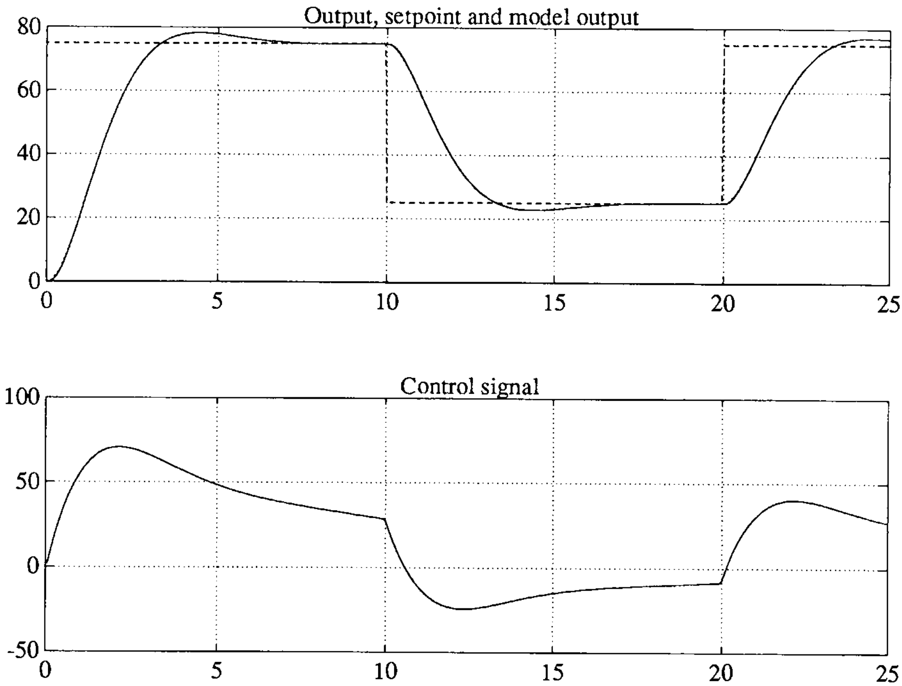

Figure 6.3. Using a setpoint filter

Programme interaction

runex 6 3
Example 6 of chapter 3: Using a setpoint filter

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
======== Control action ========
======== Assumed system ========
A (system denominator) = 1.000000 0.000000 0.000000 :=
B (system numerator) = 1.000000 1.000000 :=
======== Emulator design ========

```
P (model denominator)    =  0.500000  1.000000 :=
C (emulator denominator) =  0.500000  1.000000 :=
------------------------------------
     System polynomials
------------------------------------
A        1.000000  0.000000  0.000000
B        1.000000  1.000000
D        0.000000  0.000000
------------------------------------
     Design polynomials
------------------------------------
B+       1.000000  1.000000
B-       1.000000
C        0.500000  1.000000
P        0.500000  1.000000
Z+       1.000000
Z-       1.000000
Z-+      1.000000
------------------------------------
F        1.000000  1.000000
F filter 0.500000  1.000000
G        0.250000  0.250000
G filter 0.500000  1.000000
I
E        0.250000
ED       0.000000
------------------------------------
 ======== STC type       ========
Tuning initial conditions = FALSE :=
 ======== Identification ========
Initial Variance       = 100000.000000 :=
Forget time            = 1000.000000 :=
Cs (emulator denominator) =  1.000000  2.000000  1.000000 :=
 ======== Controller     ========
R numerator            =  0.500000  1.000000 :=
R denominator          =  1.000000  1.414000  1.000000 :=
Maximum control signal  =  100.000000 :=
Minimum control signal  =  -100.000000 :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
 ======== Out Disturbance ========
 ======== Actual system  ========
A (system denominator)  =  1.000000  1.000000  0.000000 :=
B (system numerator)    =  1.000000  0.100000 :=
Simulation running:
    25% complete
    50% complete
    75% complete
```

100% complete
Time now is 25.000000

	System polynomials		
A	*1.000000*	*0.999677*	*-0.000003*
B	*0.999759*	*0.099963*	
D	*0.000000*	*0.000000*	
	Design polynomials		
B+	*0.999759*	*0.099963*	
B-	*1.000000*		
C	*0.500000*	*1.000000*	
P	*0.500000*	*1.000000*	
Z+	*1.000000*		
Z-	*1.000000*		
Z-+	*1.000000*		
F	*0.750081*	*1.000001*	
F filter	*0.500000*	*1.000000*	
G	*0.249940*	*0.024991*	
G filter	*0.500000*	*1.000000*	
I			
E	*0.250000*		
ED	*0.000000*		

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \dfrac{Z(s)}{P(s)}\bar{w}(s)$.

Note that the control signal is considerably reduced.

Further investigations

1. Try the effect of different choices of $R(s)$ and $P(s)$, paying attention to the relative order ρ of $\dfrac{R(s)}{P(s)}$ and the steady-state gain $\dfrac{R(0)}{P(0)}$.

6.2.4. EXPLICIT CONTROL-WEIGHTED MODEL REFERENCE

Reference: Section 6.4; page 6-11. Section 3.6; page 3-16.

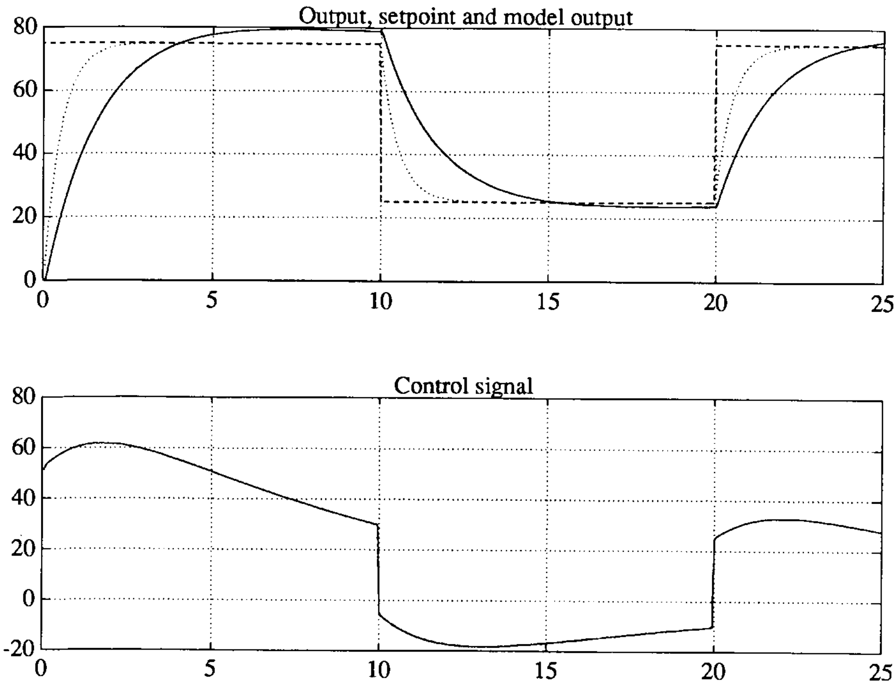

Figure 6.4. Explicit control-weighted model reference

Description

In example 6.2.1, exact model-reference control was achieved by setting $Q(s)=0$. For this example, $Q(s)$ is chosen as

$$Q(s) = \frac{s}{s+1} \tag{6.2.4.1}$$

This satifies the $Q(s)$ design rule on page I-3-17.

Programme interaction

```
runex 6 4
Example 6 of chapter 4:  Explicit control-weighted model reference

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =   1.000000   0.000000   0.000000  :=
B (system numerator)     =   1.000000   1.000000  :=
 ======== Emulator design ========
P (model denominator)    =   0.500000   1.000000  :=
C (emulator denominator) =   0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   1.000000
D          0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000   1.000000
B-         1.000000
C          0.500000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          1.000000   1.000000
F filter   0.500000   1.000000
G          0.250000   0.250000
G filter   0.500000   1.000000
I
E          0.250000
ED         0.000000
-----------------------------------
 ======== STC type        ========
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance          = 100000.000000  :=
```

```
Forget time                 = 1000.000000 :=
Cs (emulator denominator) =    1.000000   2.000000   1.000000 :=
 ======== Controller      ========
Q numerator               =    1.000000   0.000000 :=
Q denominator             =    1.000000   1.000000 :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)    =    1.000000   1.000000   0.000000 :=
B (system numerator)      =    1.000000   0.100000 :=
Number of lags            =           0 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time                 = FALSE :=
-----------------------------------
      System polynomials
-----------------------------------
A           1.000000    0.999684    -0.000003
B           0.999761    0.099964
D           0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          0.999761    0.099964
B-          1.000000
C           0.500000    1.000000
P           0.500000    1.000000
Z+          1.000000
Z-          1.000000
Z-+         1.000000
-----------------------------------
F           0.750079    1.000001
F filter    0.500000    1.000000
G           0.249940    0.024991
G filter    0.500000    1.000000
I
E           0.250000
ED          0.000000
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

Notice that the control signal is reduced with respect to that of example 6.2.1. The model following is no longer exact, but the use of the $Q(s)$ design rule ensures that there is no steady-state offset.

Further investigations

1. Try the effect of varying q in:

$$Q(s) = \frac{qs}{1+s} \tag{6.2.4.2}$$

2. Try the effect of varying T in:

$$Q(s) = \frac{s}{1+Ts} \tag{6.2.4.3}$$

3. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.4.4}$$

   There is still no offset as, in this case, the system contains an integrator and so the control signal is zero in the steady-state.

4. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.4.5}$$

   and $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.4.6}$$

   Note that there is now an offset dependent on q.

5. Use the default value of $Q(s)$ but replace $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.4.7}$$

Note that the offset disappears.

6.2.5. EXPLICIT CONTROL-WEIGHTED POLE-PLACEMENT

Reference: Section 6.4; page 6-11. Section 3.6; page 3-16.

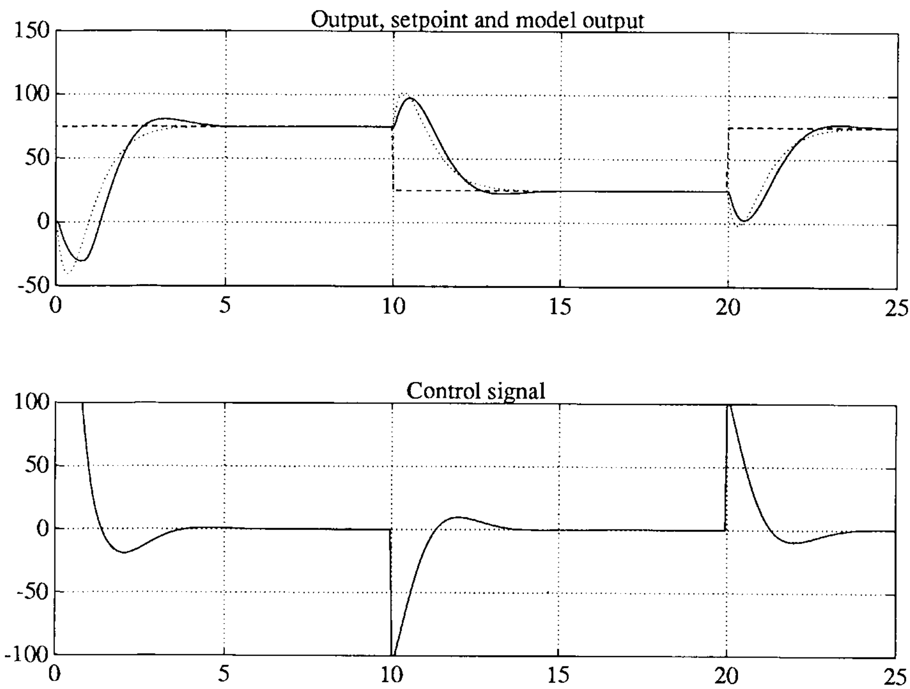

Figure 6.5. Explicit control-weighted pole-placement

Description

In example 6.2.2, exact pole-placement control was achieved by setting $Q(s)=0$. For this example, $Q(s)$ is chosen as

$$Q(s) = \frac{s}{s+1} \tag{6.2.5.1}$$

this satifies the $Q(s)$ design rule on page 3-17 of vol. 1.

Programme interaction

```
runex 6 5
Example 6 of chapter 5:  Explicit control-weighted pole-placement

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =    1.000000   0.000000   0.000000  :=
B (system numerator)     =    1.000000   1.000000  :=
 ======== Emulator design ========
Z has factor B           = TRUE  :=
P (model denominator)    =    0.500000   1.000000 *  :=
Next factor ...
P (model denominator)    =    0.500000   1.000000  :=
C (emulator denominator) =    0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   0.000000    0.000000
B          1.000000   1.000000
D          0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          1.000000
B-         1.000000   1.000000
C          0.500000   1.000000
P          0.250000   1.000000    1.000000
Z+          1.000000
Z-         1.000000   1.000000
Z-+         1.000000
-----------------------------------
F          0.500000   1.000000
F filter   0.500000   1.000000
G          0.125000   0.250000
G filter   0.500000   1.000000
I
E          0.125000   0.250000
ED          0.000000   0.000000
-----------------------------------
 ======== STC type        ========
```

```
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance        = 100000.000000  :=
Forget time             =  1000.000000  :=
Cs (emulator denominator) =    1.000000   2.000000   1.000000  :=
 ======== Controller      ========
Q numerator             =    0.100000   0.000000  :=
Q denominator           =    1.000000   1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)  =    1.000000   1.000000   0.000000  :=
B (system numerator)    =   -1.000000   1.000000  :=
Number of lags          =          0  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time               = FALSE  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.999703    0.000009
B         -0.999991    0.999709
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         0.999709
B-        -1.000282    1.000000
C          0.500000    1.000000
P          0.250000    1.000000    1.000000
Z+         1.000000
Z-        -1.000282    1.000000
Z-+        1.000000
------------------------------------
F          0.937715    0.999986
F filter   0.500000    1.000000
G          0.124964    1.562561
G filter   0.500000    1.000000
I
E          0.125000    1.563017
ED         0.000000    0.000000
------------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

Notice that the control signal is reduced with respect to that of example 6.2.2. The model following is no longer exact, but the use of the $Q(s)$ design rule ensures that there is no steady-state offset.

Further investigations

1. Try the effect of varying q (e.g. q = 0.1) in:

$$Q(s) = \frac{qs}{1+s} \tag{6.2.5.2}$$

2. Try the effect of varying T in:

$$Q(s) = \frac{s}{1+Ts} \tag{6.2.5.3}$$

3. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.5.4}$$

   There is still no offset as, in this case, the system contains an integrator and so the control signal is zero in the steady-state.

4. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.5.5}$$

   and $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.5.6}$$

   Note that there is now an offset dependent on q.

5. Use the default value of $Q(s)$ but replace $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.5.7}$$

Note that the offset disappears.

6.2.6. TIME-DELAY SYSTEM (EXPLICIT)

Reference: Section 6.4; page 6-11. Section 3.7; page 3-18.

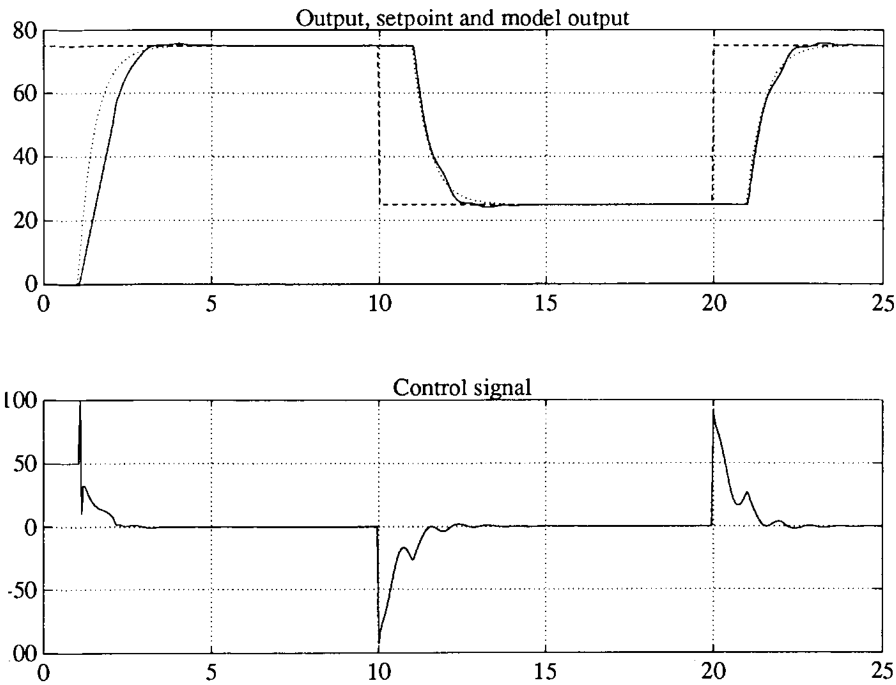

Figure 6.6. Time-delay system (explicit)

Description

This example corresponds to example 6.2.1, except that the system now is first order and has a time-delay of one unit.

$$\frac{B(s)}{A(s)} = e^{-s}\,\frac{1}{s} \qquad (6.2.6.1)$$

The corresponding emulator is based on the Pade approximation approach discussed in section I-2.6.

But note that the simulation of the system uses an exact time-delay algorithm.

Programme interaction

```
runex 6 6
Example 6 of chapter 6: Time-delay system (explicit)

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =    1.000000   2.000000  :=
B (system numerator)     =    3.000000  :=
Time delay               =    1.000000  :=
 ======== Emulator design ========
P (model denominator)    =    0.500000   1.000000  :=
C (emulator denominator) =    1.000000  :=
Pade approximation order =           3  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    2.000000
B          3.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          3.000000
B-         1.000000
C          1.000000
P          0.500000    1.000000
Z+          1.000000
Z-         1.000000
Z-+         1.000000
Pade        0.008333    0.100000    0.500000    1.000000
-----------------------------------
F          0.000000
F filter   1.000000
G          0.012500    0.150000    0.750000    1.500000
G filter   0.008333    0.100000    0.500000    1.000000
I
E          0.004167    0.050000    0.250000    0.500000
```

```
ED          0.000000    0.000000    0.000000    0.000000
------------------------------------
 ======== STC type       ========
Tuning initial conditions = FALSE  :=
 ======== Identification ========
Initial Variance        = 100000.000000  :=
Forget time             =  1000.000000  :=
Cs (emulator denominator) =    1.000000    1.000000  :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
 ======== Out Disturbance ========
 ======== Actual system  ========
A (system denominator)   =    1.000000   0.000000  :=
B (system numerator)     =    1.000000  :=
Time delay               =    1.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
------------------------------------
      System polynomials
------------------------------------
A           1.000000   -0.000010
B           0.999992
D           0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          0.999992
B-          1.000000
C           1.000000
P           0.500000    1.000000
Z+          1.000000
Z-          1.000000
Z-+         1.000000
Pade        0.008333    0.100000    0.500000    1.000000
------------------------------------
F           1.000005
F filter    1.000000
G           0.004167    0.066667    0.249999    1.499994
G filter    0.008333    0.100000    0.500000    1.000000
I
E           0.004167    0.066667    0.250001    1.500007
ED          0.000000    0.000000    0.000000    0.000000
------------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

Despite the approximation involved, the model following is close. Note that the system output is delayed by one time unit.

Further investigations

1. Try the effect of using a lower order (for example 1) approximation to a time delay in the emulator calculation.

6.2.7. EXPLICIT MODEL REFERENCE - DISTURBANCES

Reference: Section 6.4; page 6-11. Section 3.9; page 3-20.

Description

This example is identical to example 6.2.1 except that a square wave disturbance of amplitude 5 units and period two units is added to the system input. The purpose of the example is to illustrate the role of the polynomial $C(s)$ in determining closed-loop disturbance rejection. Initially, $C(s)=0.5s+1$.

Programme interaction

```
runex 6 7
Example 6 of chapter 7: Explicit model reference - disturbances

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source    ========
======== Filters        ========
======== Control action ========
======== Assumed system ========
A (system denominator)  =   1.000000   0.000000   0.000000  :=
B (system numerator)    =   1.000000   1.000000  :=
======== Emulator design ========
```

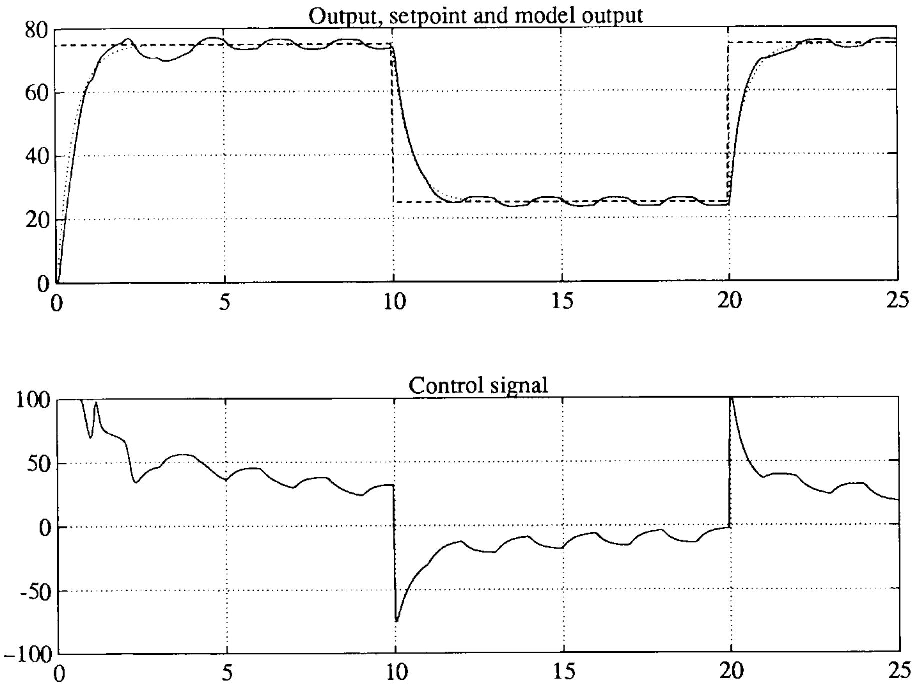

Figure 6.7. Explicit model reference - disturbances

P (model denominator) = 0.500000 1.000000 :=
C (emulator denominator) = 0.500000 1.000000 :=

System polynomials

A	*1.000000*	*0.000000*	*0.000000*
B	*1.000000*	*1.000000*	
D	*0.000000*	*0.000000*	

Design polynomials

B+	*1.000000*	*1.000000*
B-	*1.000000*	
C	*0.500000*	*1.000000*
P	*0.500000*	*1.000000*
Z+	*1.000000*	
Z-	*1.000000*	

```
Z-+        1.000000
------------------------------------
F          1.000000    1.000000
F filter   0.500000    1.000000
G          0.250000    0.250000
G filter   0.500000    1.000000
I
E          0.250000
ED         0.000000
------------------------------------
 ======== STC type        ========
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance          = 100000.000000  :=
Forget time               =  1000.000000  :=
Cs (emulator denominator) =   1.000000   2.000000   1.000000  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
Square amplitude          =    5.000000  :=
Period                    =    2.000000  :=
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =    1.000000   0.100000  :=
Number of lags            =          0  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
More time                 = FALSE  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.980893    -0.001185
B          0.989023    0.094623
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         0.989023    0.094623
B-         1.000000
C          0.500000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
```

Z-+	*1.000000*	
F	*0.754777*	*1.000296*
F filter	*0.500000*	*1.000000*
G	*0.247256*	*0.023656*
G filter	*0.500000*	*1.000000*
I		
E	*0.250000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

The effect of the disturbance is to perturb the system output; the control signal reacts to some extent to counteract this effect. The parameters do not converge to the 'correct' values, yet the control is reasonable.

Further investigations

1. The emulator denominator $C(s)$ is of the form 1+Ts. Try the effect of varying the time constant T (try, for example, T=0.1) of the emulator denominator C. How does this affect the system output and the control signal?

2. Try the effect of disturbances on example 6.2.2 - 6.2.5. (You will need to use the "Enter all variables" option to expose the disturbance variables).

6.2.8. EXPLICIT MODEL REFERENCE PID

Reference: Section 6.4; page 6-11. Section 3.10; page 3-24.

Description

This example is identical to example 6.2.1 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral action" to "TRUE".

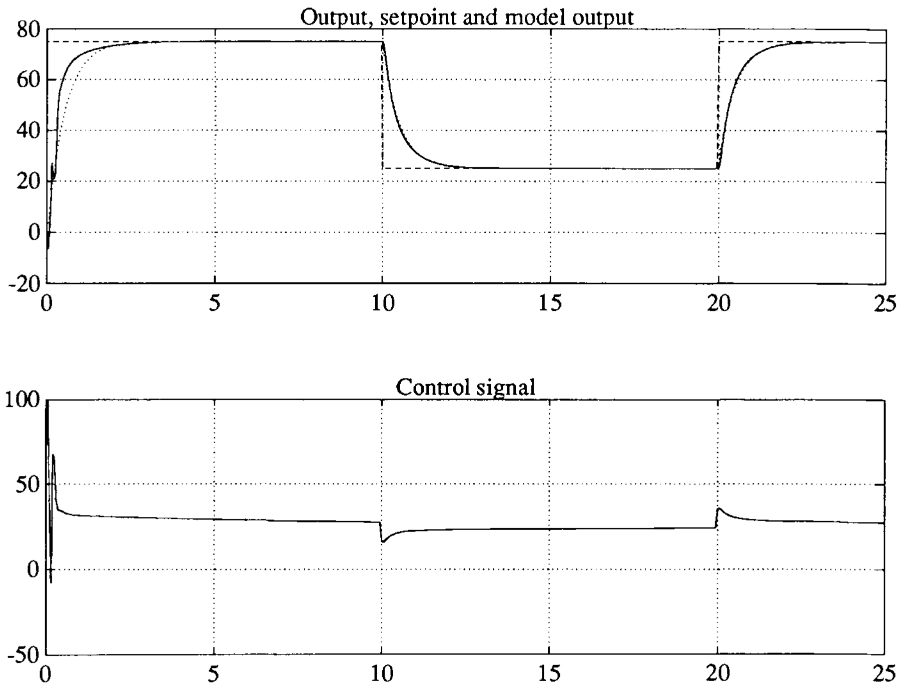

Figure 6.8. Explicit model reference PID

c) The degree of $C(s)$ is increased by one: $C(s) = (1+0.5s)^2$.

Programme interaction

runex 6 8
Example 6 of chapter 8: Explicit model reference PID

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
======== Control action ========
Integral action = TRUE :=

```
 ======== Assumed system  ========
A (system denominator)   =   1.000000  0.000000  0.000000 :=
B (system numerator)     =   1.000000  1.000000 :=
 ======== Emulator design ========
P (model denominator)    =   0.500000  1.000000 :=
C (emulator denominator) =   0.500000  1.000000 * :=
Next factor ...
C (emulator denominator) =   0.500000  1.000000 :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000  0.000000  0.000000  0.000000
B          1.000000  1.000000  0.000000
D          0.000000  0.000000  0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000  1.000000  0.000000
B-         1.000000
C          0.250000  1.000000  1.000000
P          0.500000  1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          0.750000  1.500000  1.000000
F filter   0.250000  1.000000  1.000000
G          0.125000  0.125000  0.000000
G filter   0.250000  1.000000  1.000000
I
E          0.125000
ED         0.000000
-----------------------------------
 ======== STC type       ========
Tuning initial conditions = TRUE :=
 ======== Identification ========
Initial Variance         = 100000.000000 :=
Forget time              =  1000.000000 :=
Cs (emulator denominator) =   1.000000  3.000000  3.000000  1.000000 :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
Step amplitude           =  -25.000000 :=
 ======== Out Disturbance ========
 ======== Actual system  ========
A (system denominator)   =   1.000000  1.000000  0.000000 :=
B (system numerator)     =  10.000000  1.000000 :=
Number of lags           =          0 :=
```

Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000
More time . = FALSE :=

System polynomials

A	*1.000000*	*0.998018*	*-0.000929*	*0.000000*
B	*9.996129*	*0.989964*	*0.000000*	
D	*6.498928*	*-248.968093*	*-24.673401*	

Design polynomials

B+	*9.996129*	*0.989964*	*0.000000*
B-	*1.000000*		
C	*0.250000*	*1.000000*	*1.000000*
P	*0.500000*	*1.000000*	
Z+	*1.000000*		
Z-	*1.000000*		
Z-+	*1.000000*		

F	*0.625248*	*1.500116*	*1.000000*
F filter	*0.250000*	*1.000000*	*1.000000*
G	*1.249516*	*0.123746*	*0.000000*
G filter	*0.250000*	*1.000000*	*1.000000*
I	*-34.370476*	*-6.333639*	
E	*0.125000*		
ED	*3.249464*		

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

Note that the response is better than that of the non-adaptive controller (example 3.2.8) because the initial conditions (corresponding to the offset) are estimated.

Further investigations

1. Try the controller of example 6.2.1, but with the disturbance. (Set integral action to FALSE and set $C(s) = 0.5s+1$. What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input disturbance. Explain what you observe.
3. Try the effect of *not* estimating an initial condition.

6.2.9. EXPLICIT POLE-PLACEMENT PID

Reference: Section 6.4; page 6-11. Section 3.10; page 3-25.

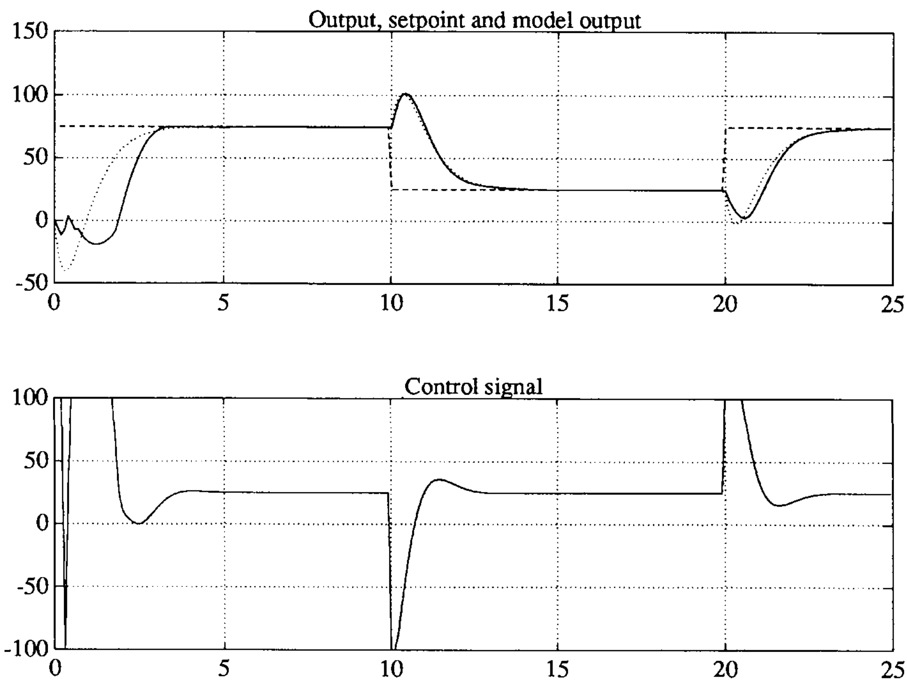

Figure 6.9. Explicit pole-placement PID

Description

This example is identical to example 6.2.2 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral" action to "TRUE".

c) The degree of $C(s)$ is increased by one: $C(s) = (1+0.5s)^2$.

d) The sample interval is decreased to 0.01 to give a satisfactory approximation.

Programme interaction

```
runex 6 9
Example 6 of chapter 9: Explicit pole-placement PID

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source    ========
 ======== Filters        ========
 ======== Control action ========
Integral action          = TRUE  :=
 ======== Assumed system ========
A (system denominator)     =   1.000000   0.000000   0.000000  :=
B (system numerator)       =   1.000000   1.000000  :=
 ======== Emulator design ========
Z has factor B           = TRUE  :=
P (model denominator)      =   0.500000   1.000000 *  :=
Next factor ...
P (model denominator)      =   0.500000   1.000000  :=
C (emulator denominator)   =   0.500000   1.000000 *  :=
Next factor ...
C (emulator denominator)   =   0.500000   1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A        1.000000   0.000000   0.000000   0.000000
B        1.000000   1.000000   0.000000
D        0.000000   0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+       1.000000   0.000000
B-       1.000000   1.000000
```

```
C          0.250000   1.000000   1.000000
P          0.250000   1.000000   1.000000
Z+         1.000000
Z-         1.000000   1.000000
Z-+        1.000000
------------------------------------
F          0.500000   1.000000   1.000000
F filter   0.250000   1.000000   1.000000
G          0.062500   0.000000   0.000000
G filter   0.250000   1.000000   1.000000
I
E          0.062500   0.000000
ED         0.000000   0.000000
------------------------------------
 ======== STC type        ========
Tuning initial conditions = TRUE  :=
 ======== Identification  ========
Initial Variance          = 1000.000000  :=
Forget time               = 1000.000000  :=
Cs (emulator denominator) =    1.000000   3.000000   3.000000   1.000000  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
Step amplitude            =  -25.000000  :=
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =   -1.000000   1.000000  :=
Number of lags            =          0  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time                 = FALSE  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000   0.999981   0.000050   0.000000
B         -0.999974   0.999907   0.000000
D         -0.125478  25.117734 -24.993554
------------------------------------
      Design polynomials
------------------------------------
B+         0.999907   0.000000
B-        -1.000067   1.000000
C          0.250000   1.000000   1.000000
```

P	*0.250000*	*1.000000*	*1.000000*
Z+	*1.000000*		
Z-	*-1.000067*	*1.000000*	
Z-+	*1.000000*		

F	*2.031275*	*2.999943*	*1.000000*
F filter	*0.250000*	*1.000000*	*1.000000*
G	*0.062494*	*2.468682*	*0.000000*
G filter	*0.250000*	*1.000000*	*1.000000*
I	*-1.284339*	*-61.734934*	
E	*0.062500*	*2.468913*	
ED	*-0.031370*	*0.028033*	

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

The effect of the disturbance is, in the short term, to spoil the closed-loop response; but, in the long term, the response is not affected. Note that the steady-state control signal has a value of +25 to compensate for the disturbance: the controller has integral action.

Further investigations

1. Try the controller of example 6.2.2, but with the disturbance. (Set integral action to FALSE and set $C(s) = 0.5s+1$. What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input disturbance. Explain what you observe.

6.2.10. DETUNED MODEL-REFERENCE

Reference: Section 3.11; page 3-28, section 6.4 p 6-11.

Description

Example 3.2.10 illustrates the use of a reference model with one pole and one zero:

$$\frac{Z(s)}{P(s)} = \frac{0.03s+1}{0.3s+1} \tag{6.2.10.1}$$

together with control weighting:

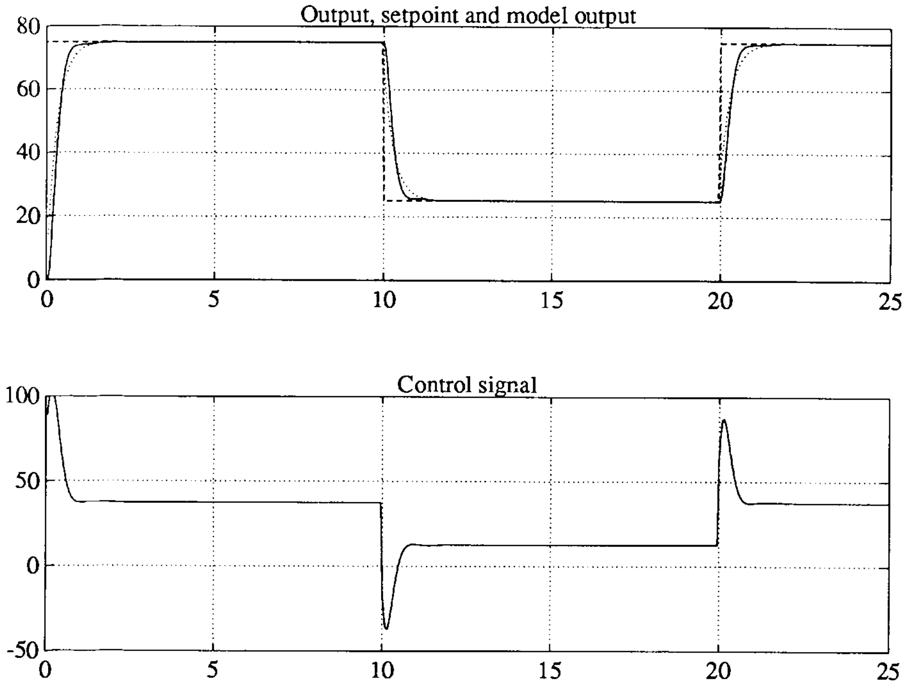

Figure 6.10. Detuned model-reference

$$Q(s) = \frac{qs}{0.03s+1} \qquad (6.2.10.2)$$

In this example q=0.05 is used initially. An explicit self-tuning version is used in this example.

Programme interaction

runex 6 10
Example 6 of chapter 10: Detuned model-reference

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========

```
======== Filters        ========
======== Control action ========
Integral action          = TRUE  :=
======== Assumed system ========
A (system denominator)   =   1.000000   0.000000  :=
B (system numerator)     =   1.000000  :=
======== Emulator design ========
Z-+ (Z- not including B) =   0.030000   1.000000  :=
P (model denominator)    =   0.300000   1.000000  :=
C (emulator denominator) =   0.300000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   0.000000
D          0.000000   0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000   0.000000
B-         1.000000
C          0.300000   1.000000
P          0.300000   1.000000
Z+         1.000000
Z-         0.030000   1.000000
Z-+        0.030000   1.000000
-----------------------------------
F          0.570000   1.000000
F filter   0.300000   1.000000
G          0.072900   0.000000
G filter   0.009000   0.330000   1.000000
I
E          0.072900
ED         0.000000
-----------------------------------
======== STC type       ========
======== Identification ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
Cs (emulator denominator) =   1.000000   2.000000   1.000000  :=
======== Controller     ========
Q numerator              =   0.050000   0.000000  :=
Q denominator            =   0.030000   1.000000  :=
======== Simulation     ========
======== Setpoint       ========
======== In Disturbance ========
======== Out Disturbance ========
======== Actual system  ========
A (system denominator)   =   1.000000   1.000000  :=
```

B (system numerator) = 2.000000 :=
Number of lags = 0 :=
Simulation running:
25% complete
50% complete
75% complete
100% complete
Time now is 25.000000
More time = FALSE :=

System polynomials

A	*1.000000*	*0.999643*	*0.000000*
B	*1.999582*	*0.000000*	
D	*0.000000*	*0.000000*	*0.000000*

Design polynomials

B+	*1.999582*	*0.000000*	
B-	*1.000000*		
C	*0.300000*	*1.000000*	
P	*0.300000*	*1.000000*	
Z+	*1.000000*		
Z-	*0.030000*	*1.000000*	
Z-+	*0.030000*	*1.000000*	

F	*0.494873*	*1.000000*	
F filter	*0.300000*	*1.000000*	
G	*0.150276*	*0.000000*	
G filter	*0.009000*	*0.330000*	*1.000000*
I			
E	*0.075154*		
ED	*0.000000*		

Discussion

The performance is similar to the non-adaptive case as the parameters rapidly converge to their correct values.

Further investigations

1. Examine the effect of varying the parameter q.

2. Examine the effect of varying the initial variance.

6.2.11. EXPLICIT PREDICTIVE CONTROL

Reference: Sections 3.7&8; page 3-18 and section 6.4 page 6-11.

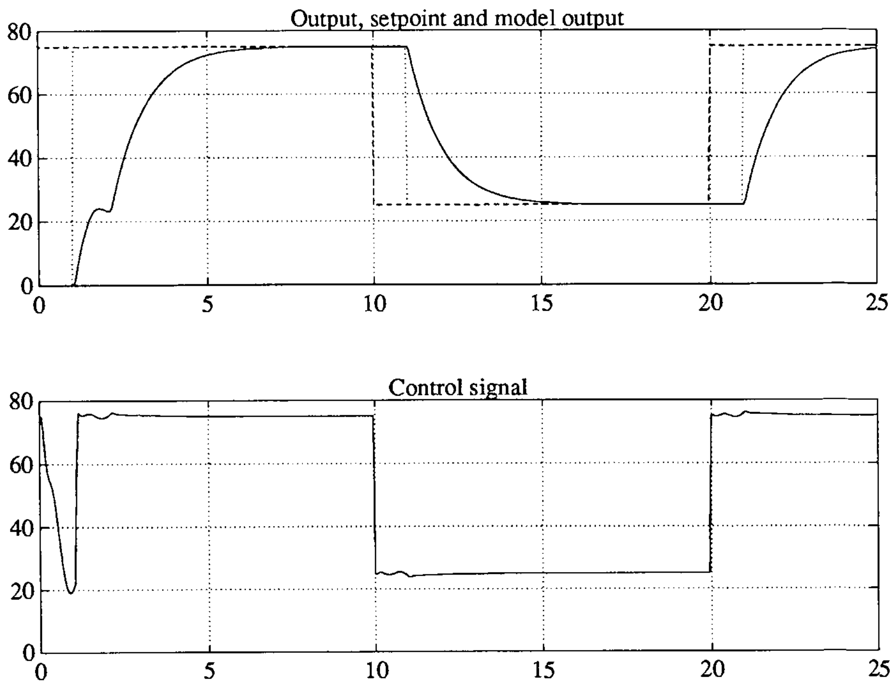

Figure 6.11. Explicit predictive control

Description

A predictive emulator in a feedback loop was discussed in example 3.2.11. In this example, the emulator is tuned using an explicit algorithm.

The open loop system has a first order rational part with unit time constant together with a unit delay

$$e^{-sT}\frac{B(s)}{\ldots} = e^{-s}\frac{1}{\ldots} \quad (6.2.11.1)$$

$Q(s)$ is chosen to be an inverse PI controller:

$$\frac{1}{Q(s)} = 1 + \frac{1}{s} \tag{6.2.11.2}$$

Programme interaction

```
runex 6 11
Example 6 of chapter 11: Explicit predictive control

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval          =    0.050000  :=
 ======== Control action ========
Integral action          = FALSE  :=
 ======== Assumed system ========
A (system denominator)    =    1.000000    0.000000  :=
B (system numerator)      =    2.000000  :=
Time delay                =    1.000000  :=
 ======== Emulator design ========
P (model denominator)     =    1.000000  :=
C (emulator denominator)  =    1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.000000
B          2.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         2.000000
B-         1.000000
C          1.000000
P          1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
Pade       0.000595    0.011905    0.107143    0.500000    1.000000
-----------------------------------
F          1.000000
F filter   1.000000
```

```
G          0.000000   0.047619   0.000000   2.000000
G filter   0.000595   0.011905   0.107143   0.500000   1.000000
I
E          0.000000   0.023810   0.000000   1.000000
ED         0.000000   0.000000   0.000000   0.000000
-----------------------------------
 ======== STC type       ========
Tuning initial conditions = FALSE  :=
 ======== Identification ========
Initial Variance          = 100000.000000  :=
Forget time               = 1000.000000  :=
Cs (emulator denominator) =    1.000000   1.000000  :=
 ======== Controller     ========
Q numerator               =    1.000000   0.000000  :=
Q denominator             =    1.000000   1.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
 ======== Out Disturbance ========
Step amplitude            =    0.000000  :=
Cos amplitude             =    0.000000  :=
 ======== Actual system  ========
A (system denominator)    =    1.000000   1.000000  :=
B (system numerator)      =    1.000000  :=
Time delay                =    1.000000  :=
Number of lags            =           0  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time                 = FALSE  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   1.000000
B          1.000000
D          0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000
B-         1.000000
C          1.000000
P          1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
```

Pade	*0.000595*	*0.011905*	*0.107143*	*0.500000*	*1.000000*
F	*0.367879*				
F filter	*1.000000*				
G	*0.000376*	*0.015908*	*0.051819*	*0.632121*	
G filter	*0.000595*	*0.011905*	*0.107143*	*0.500000*	*1.000000*
I					
E	*0.000376*	*0.015908*	*0.051819*	*0.632121*	
ED	*0.000000*	*0.000000*	*0.000000*	*0.000000*	

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t) = y(t-1)$.

Note that the response is as predicted: a delayed first-order response delayed by one unit.

Further investigations

1. Try the effect of varying the order of the Pade approximation. Note that zero corresponds to having no predictor, and the response is not good. What is the smallest satisfactory order?

2. Try varying the system time delay, keeping the assumed and actual delay the same. For each value of delay, find the minimum satisfactory Pade order. Note that for larger Pade orders, you may need to reduce the sample interval for numerical reasons.

3. Try the effect of choosing an incorrect time-delay, say 0.9 in place of 1.0. Find the maximum and minimum values of the assumed delay (actual delay=1) giving satisfactory performance.

4. Try putting integral action into the predictor (Integral action = TRUE, C = s+1) and use a Pade order of 3. Observe the performance with an output step disturbance, and compare to the integral-free case.

5. Add a sinusoidal disturbance to the system output; how does the performance depend on the amplitude of this signal and the system time-delay?

6.2.12. EXPLICIT LINEAR-QUADRATIC POLE-PLACEMENT

Reference: Section 3.4; page 3-14.

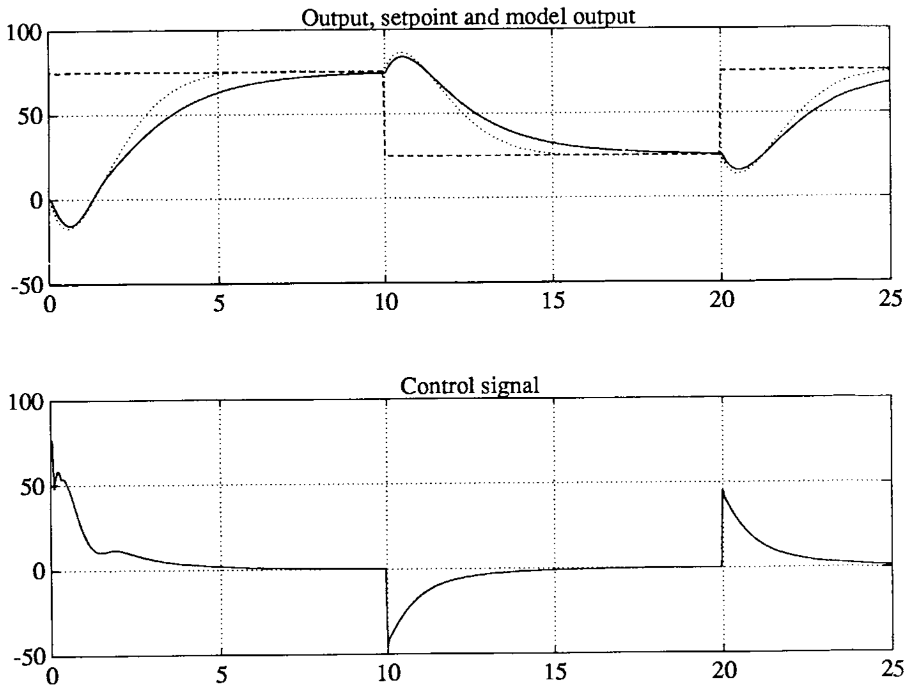

Figure 6.12. Explicit linear-quadratic pole-placement

Description

This example is identical to example 6.2.2, except that the closed-loop poles are chosen to solve equation I-3.4.23:

$$P(s)P(-s) = B(s)B(-s) + \lambda A(s)A(-s)$$

That is, the poles are chosen to correspond to those given by linear-quadratic optimisation theory where λ is the linear-quadratic weighting.

Programme interaction

```
runex 6 12
Example 6 of chapter 12:  Explicit linear-quadratic pole-placement

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000   0.000000   0.000000  :=
B (system numerator)      =    1.000000   1.000000  :=
 ======== Emulator design ========
Linear-quadratic weight   =    1.000000  :=
C (emulator denominator) =    0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    1.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000
B-         1.000000    1.000000
C          0.500000    1.000000
P          1.000000    1.732051    1.000000
Z+         1.000000
Z-         1.000000    1.000000
Z-+        1.000000
-----------------------------------
F          1.232051    1.000000
F filter   0.500000    1.000000
G          0.500000    0.633975
G filter   0.500000    1.000000
I
E          0.500000    0.633975
ED         0.000000    0.000000
-----------------------------------
 ======== STC type        ========
Tuning initial conditions = TRUE  :=
 ======== Identification  ========
Initial Variance         = 100000.000000  :=
```

```
Forget time                = 1000.000000 :=
Cs (emulator denominator) =    1.000000   2.000000   1.000000 :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000   1.000000   0.000000 :=
B (system numerator)     =   -1.000000   1.000000 :=
Number of lags           =          0 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
More time                = FALSE :=
-----------------------------------
      System polynomials
-----------------------------------
A           1.000000   0.999840   0.000002
B          -1.000117   0.999843
D           0.008655   0.000834
-----------------------------------
      Design polynomials
-----------------------------------
B+          0.999843
B-         -1.000274   1.000000
C           0.500000   1.000000
P           1.000157   2.449737   1.000000
Z+          1.000000
Z-         -1.000274   1.000000
Z-+         1.000000
-----------------------------------
F           1.112563   0.999994
F filter    0.500000   1.000000
G           0.500000   2.837449
G filter    0.500000   1.000000
I          -0.010091
E           0.500078   2.837895
ED          0.008656   0.012457
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

As in example 3.2.2 note the typical behaviour of a system with right-hand plane zeros: the output initially goes the wrong way in response to a step change.

The desired closed-loop zeros are the same as in example 3.2.2; that is, the system zeros are unchanged; but the poles, and the step rise-time, now depend on $A(s)$, $B(s)$ and λ.

Further investigations

1. Try the effect of varying the linear-quadratic weighting λ. How does this affect the system output and the control signal?
2. Try repeating this example using the same system as example 3.2.1 ($B(s) = 10+s$). How does the closed-loop response when using linear-quadratic control differ from that when using model-reference control?

6.2.13. EXPLICIT LINEAR-QUADRATIC PID

Reference: Section 3.4; page 3-14 and section 3.10; page 3-25.

Description

This example is identical to example 12 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral action" to "TRUE".

c) The degree of $C(s)$ is increased by one: $C(s) = (1+0.5s)^2$.

d) The sample interval is decreased to 0.01 to give a satisfactory approximation.

Programme interaction

runex 6 13
Example 6 of chapter 13: Explicit linear-quadratic PID

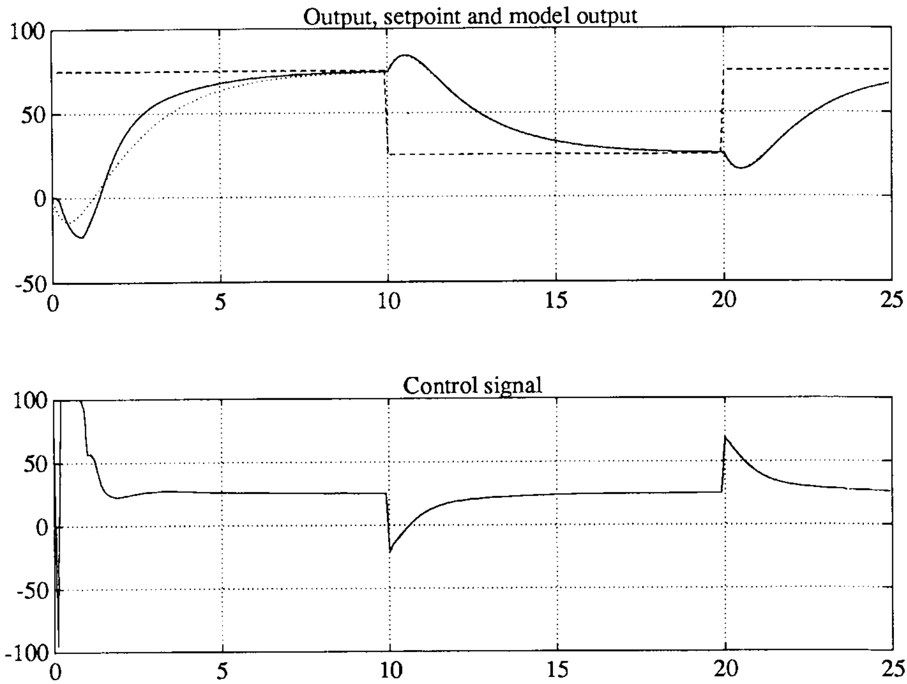

Figure 6.13. Explicit linear-quadratic PID

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source     ========
======== Filters         ========
Sample Interval          =    0.010000  :=
======== Control action ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)   =    1.000000    1.000000    0.000000  :=
B (system numerator)     =   -1.000000    1.000000  :=
======== Emulator design ========
Z has factor B           = TRUE  :=
Linear-quadratic poles   = TRUE  :=
Linear-quadratic weight  =    1.000000  :=
C (emulator denominator) =    0.500000    1.000000 *  :=
```

```
Next factor ...
C (emulator denominator) =    0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   1.000000   0.000000   0.000000
B         -1.000000   1.000000   0.000000
D          0.000000   0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000   0.000000
B-        -1.000000   1.000000
C          0.250000   1.000000   1.000000
P          1.000000   2.449490   1.000000
Z+         1.000000
Z-        -1.000000   1.000000
Z-+        1.000000
-----------------------------------
F          3.393304   4.449490   1.000000
F filter   0.250000   1.000000   1.000000
G          0.250000   4.755676   0.000000
G filter   0.250000   1.000000   1.000000
I
E          0.250000   4.755676
ED         0.000000   0.000000
-----------------------------------
 ======== STC type        ========
Tuning initial conditions = TRUE  :=
 ======== Identification  ========
Initial Variance          = 100000.000000  :=
Forget time               =  1000.000000  :=
Cs (emulator denominator) =    0.500000   1.000000 *  :=
Next factor ...
Cs (emulator denominator) =    0.500000   1.000000 *  :=
Next factor ...
Cs (emulator denominator) =    0.500000   1.000000  :=
Normalising Cs so that c0 = 1
Cs      1.000000    6.000000   12.000000    8.000000
 ======== Controller      ========
Maximum control signal   =  100.000000  :=
Minimum control signal   =  -100.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
Step amplitude           =   -25.000000  :=
 ======== Out Disturbance ========
Step amplitude           =     0.000000  :=
 ======== Actual system   ========
```

```
A (system denominator)   =    1.000000   1.000000   0.000000  :=
B (system numerator)     =   -1.000000   1.000000  :=
D (initial conditions)   =    0.000000   0.000000   0.000000  :=
Time delay               =    0.000000  :=
Number of lags           =           0  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
More time                = FALSE  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.999936    0.000054    0.000000
B         -0.999996    0.999727    0.000000
D         -0.128114   25.128045  -24.994685
------------------------------------
      Design polynomials
------------------------------------
B+         0.999727    0.000000
B-        -1.000269    1.000000
C          0.250000    1.000000    1.000000
P          1.000273    2.449849    1.000000
Z+         1.000000
Z-        -1.000269    1.000000
Z-+        1.000000
------------------------------------
F          3.393938    4.449862    1.000000
F filter   0.250000    1.000000    1.000000
G          0.250000    4.756236    0.000000
G filter   0.250000    1.000000    1.000000
I         -5.769595 -119.037979
E          0.250068    4.757535
ED        -0.128149    0.124881
------------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The effect of the disturbance is, in the short term, to spoil the closed-loop response; but, in the long term, the response is not affected. Note that the steady-state control signal has a value of +25 to compensate for the disturbance: the controller has integral action.

Further investigations

1. Try the controller of example 12, but with the disturbance. (Set "Integral action" to FALSE and reduce the orders of $C(s)$ and $C_s(s)$ by one by setting a factor equal to 1). What is the effect of the input disturbance?

2. Repeat step 1, but with an output disturbance in place of an input distubance. Explain what you observe.

6.2.14. IMPLICIT MODEL REFERENCE

Reference: Section 6.4; page 6-11. Section 3.4; page 3-12.

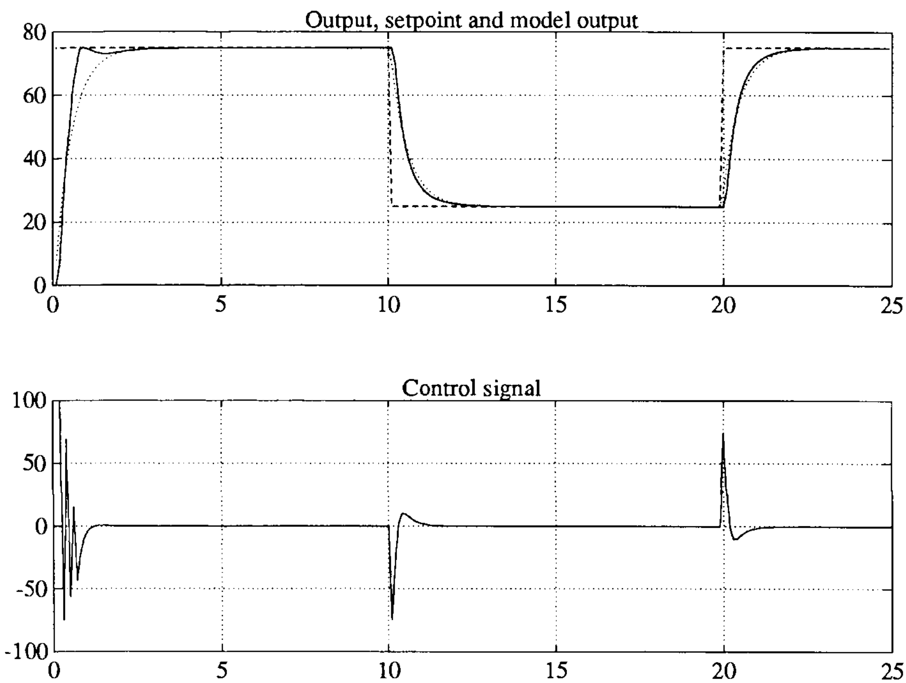

Figure 6.14. Implicit model reference

Description

This is the self-tuning equivalent of example 3.2.1 using the implicit approach with off-line choice of emulator design parameters.

The aim of the controller is to make the system output follow the model:

$$\bar{y}(s) = \frac{Z(s)}{P(s)}\bar{w}(s) \tag{6.2.14.1}$$

where, in this case, $Z(s)=1$ and $P(s) = 1+Ts$ where the model time-constant T = 0.5.

The system parameters are estimated and the corresponding emulator parameters evaluated at each time-step.

As $P(s)/Z(s)$ is improper, a Λ filter is used with

$$\Lambda = \frac{Z(s)}{P(s)} = \frac{1}{0.5s+1} \tag{6.2.14.2}$$

Programme interaction

```
runex 6 14
Example 6 of chapter 14:  Implicit model reference

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
Integral action           = FALSE  :=
 ======== Assumed system  ========
A (system denominator)    =   1.000000   0.000000   0.000000  :=
B (system numerator)      =   1.000000   1.000000  :=
 ======== Emulator design ========
P (model denominator)     =   0.500000   1.000000  :=
C (emulator denominator)  =   0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    1.000000
D          0.000000    0.000000
-----------------------------------
```

```
      Design polynomials
------------------------------------
B+          1.000000    1.000000
B-          1.000000
C           0.500000    1.000000
P           0.500000    1.000000
Z+          1.000000
Z-          1.000000
Z-+         1.000000
------------------------------------
F           1.000000    1.000000
F filter    0.500000    1.000000
G           0.250000    0.250000
G filter    0.500000    1.000000
I
E           0.250000
ED          0.000000
------------------------------------
 ======== STC type       ========
 ======== Tuner          ========
Initial Variance        = 100000.000000 :=
Forget time             = 1000.000000 :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
 ======== Out Disturbance ========
 ======== Actual system  ========
A (system denominator)  =   1.000000   1.000000   0.000000 :=
B (system numerator)    =   1.000000   10.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
------------------------------------
      System polynomials
------------------------------------
A           1.000000    0.000000    0.000000
B           1.000000    1.000000
D           0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          1.000000    1.000000
B-          1.000000
C           0.500000    1.000000
P           0.500000    1.000000
```

Z+	*1.000000*	
Z-	*1.000000*	
Z-+	*1.000000*	

F	*0.746634*	*1.000119*
F filter	*0.500000*	*1.000000*
G	*0.241125*	*2.532036*
G filter	*0.500000*	*1.000000*
I		
E	*0.250000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

After a short time, the output follows the model output closely despite the initially incorrect parameters.

Further investigations

1. Try the effect of varying the time constant T of the inverse model P. How does this affect the system output and the control signal?
2. The emulator denominator $C(s)$ is also of the form 1+Ts. Try the effect of varying the time constant T of the emulator denominator C. How does this affect the system output and the control signal?
3. Try changing the limits of the control signal so that it is clipped; for example choose 'Maximum control signal' as 10 and 'Minimum control signal' as -10. How does this affect the system output and the control signal?
4. The controller and simulation are implemented as discrete-time systems. Try the effect of vary-

ing the sample interval on closed-loop performance.

6.2.15. IMPLICIT POLE-PLACEMENT CONTROL

Reference: Section 6.4; page 6-11. Section 3.4; page 3-13.

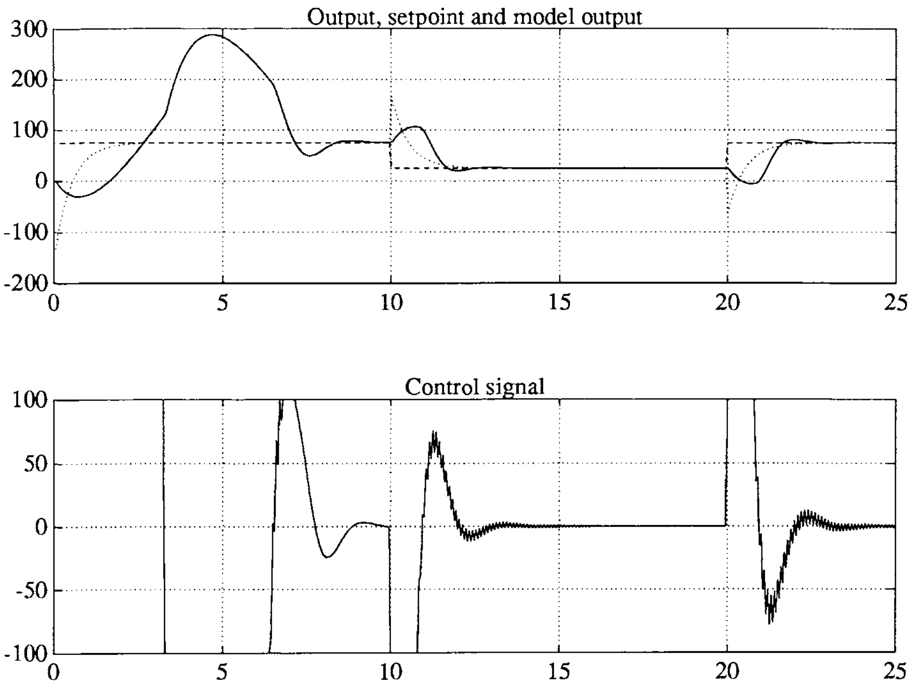

Figure 6.15. Implicit pole-placement control

Description

As discussed in vol.I, the emulator designed in the second example of section I-2.4 may be embedded in a feedback loop to give pole-placement control.

The aim of the controller is to make the system output follow the model:

$$\bar{y}(s) = \frac{Z(s)}{P(s)}\bar{w}(s) \tag{6.2.15.1}$$

where, in this case, $Z(s) = B(s)$ and $P(s) = (1+Ts)^2$ where the model time-constant T = 0.5. Unlike example 6.2.2, the algorithm is implicit. The design, however, is on-line: the system parameters are estimated and $B(s)$ used in the $\Lambda(s)$ filter.

Programme interaction

```
runex 6 15
Example 6 of chapter 15: Implicit pole-placement control

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =   1.000000   0.000000   0.000000 :=
B (system numerator)     =   1.000000   1.000000 :=
 ======== Emulator design ========
Z has factor B           = TRUE :=
P (model denominator)    =   0.500000   1.000000 :=
C (emulator denominator) =   0.500000   1.000000 :=
------------------------------------
       System polynomials
------------------------------------
A         1.000000    0.000000    0.000000
B         1.000000    1.000000
D         0.000000    0.000000
------------------------------------
       Design polynomials
------------------------------------
B+        1.000000
B-        1.000000    1.000000
C         0.500000    1.000000
P         0.500000    1.000000
Z+        1.000000
Z-        1.000000    1.000000
Z-+       1.000000
------------------------------------
F         0.000000    1.000000
F filter  0.500000    1.000000
G         0.250000
G filter  0.500000    1.000000
I
```

```
E          0.250000
ED         0.000000
------------------------------------
======== STC type        ========
Tuning initial conditions = FALSE  :=
======== Identification  ========
Initial Variance          = 100000.000000  :=
Forget time               = 1000.000000  :=
Cs (emulator denominator) =    1.000000    2.000000    1.000000  :=
======== Tuner           ========
Initial Variance          = 100000.000000  :=
Forget time               = 1000.000000  :=
======== Controller      ========
======== Simulation      ========
======== Setpoint        ========
======== In Disturbance  ========
======== Out Disturbance ========
======== Actual system   ========
A (system denominator)   =    1.000000    1.000000    0.000000  :=
B (system numerator)     =   -1.000000    1.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.999768    0.000013
B         -0.999997    0.999759
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         0.999759
B-        -1.000238    1.000000
C          0.500000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-        -1.000238    1.000000
Z-+        1.000000
------------------------------------
F          0.842645    0.990154
F filter   0.500000    1.000000
G          1.080971
G filter   0.500000    1.000000
I
E          0.250000
```

ED *0.000000*

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

In this case, note the typical behaviour of a system with right-hand plane zeros: the output initially goes the wrong way in response to a step change. How does the performance compare with that of example 6.2.2?

Further investigations

1. Try the effect of varying the time constant T of the inverse model P. How does this affect the system output and the control signal?
2. Try repeating this example using the same system as the previous example ($B(s) = 10+s$). How does the closed-loop response when using pole-placement differ from that when using model-reference control?

6.2.16. USING A SETPOINT FILTER

Reference: Section 6.4; page 6-11. Section 3.5; page 3-15.

Description

This example is identical to example 6.2.14 except that a setpoint filter is added:

$$w_R(s) = R(s)\bar{w}(s);\ R(s) = \frac{0.5s+1}{s^2 + \sqrt{2}s + 1} \tag{6.2.16.1}$$

The closed loop response is thus:

$$\bar{y}(s) = \frac{Z(s)}{P(s)}R(s)\bar{w}(s) = \frac{1}{0.5s+1}\,\frac{0.5s+1}{s^2 + \sqrt{2}s + 1}\bar{w}(s) = \frac{1}{s^2 + \sqrt{2}s + 1}\bar{w}(s) \tag{6.2.16.2}$$

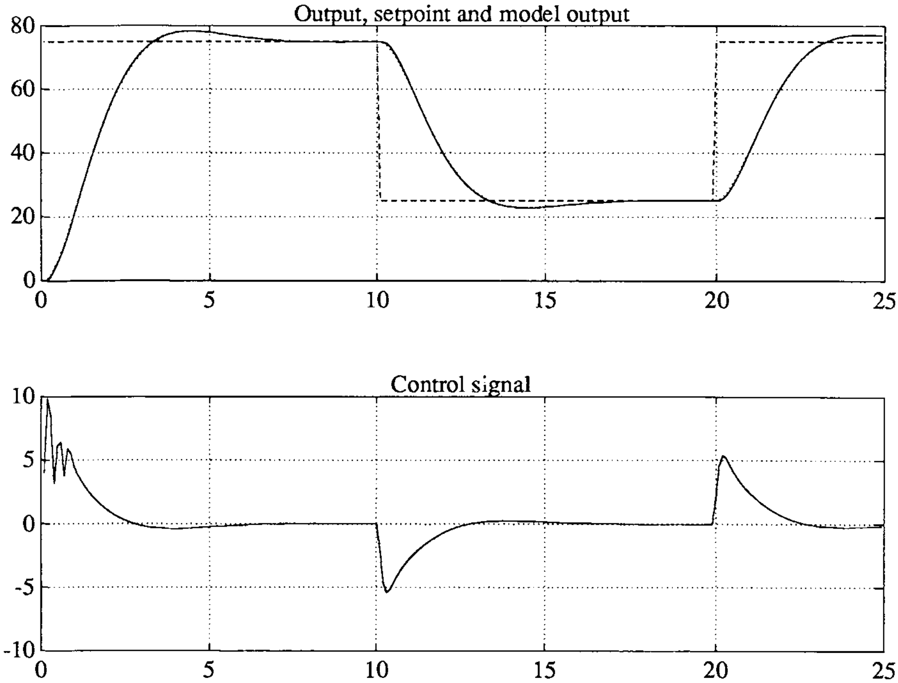

Figure 6.16. Using a setpoint filter

Programme interaction

runex 6 16
Example 6 of chapter 16: Using a setpoint filter

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
======== Control action ========
Integral action = FALSE :=
======== Assumed system ========
A (system denominator) = 1.000000 0.000000 0.000000 :=
B (system numerator) = 1.000000 1.000000 :=

```
======== Emulator design ========
P (model denominator)     =    0.500000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    1.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000    1.000000
B-         1.000000
C          0.500000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          1.000000    1.000000
F filter   0.500000    1.000000
G          0.250000    0.250000
G filter   0.500000    1.000000
I
E          0.250000
ED         0.000000
-----------------------------------
======== STC type       ========
======== Tuner          ========
Initial Variance         = 100000.000000  :=
Forget time              =  1000.000000  :=
======== Controller     ========
R numerator              =    0.500000   1.000000  :=
R denominator            =    1.000000   1.414000   1.000000  :=
Maximum control signal   =   100.000000  :=
Minimum control signal   =  -100.000000  :=
======== Simulation     ========
======== Setpoint       ========
======== In Disturbance ========
======== Out Disturbance ========
======== Actual system  ========
A (system denominator)   =    1.000000   1.000000   0.000000  :=
B (system numerator)     =    1.000000  10.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
```

Time now is 25.000000

System polynomials			
A	*1.000000*	*0.000000*	*0.000000*
B	*1.000000*	*1.000000*	
D	*0.000000*	*0.000000*	
Design polynomials			
B+	*1.000000*	*1.000000*	
B-	*1.000000*		
C	*0.500000*	*1.000000*	
P	*0.500000*	*1.000000*	
Z+	*1.000000*		
Z-	*1.000000*		
Z-+	*1.000000*		
F	*0.748986*	*1.000014*	
F filter	*0.500000*	*1.000000*	
G	*0.241512*	*2.509930*	
G filter	*0.500000*	*1.000000*	
I			
E	*0.250000*		
ED	*0.000000*		

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

Note that the control signal is considerably reduced.

Further investigations

1. Try the effect of different choices of $R(s)$ and $P(s)$.

6.2.17. IMPLICIT CONTROL-WEIGHTED MODEL REFERENCE

Reference: Section 6.4; page 6-11. Section 3.6; page 3-16.

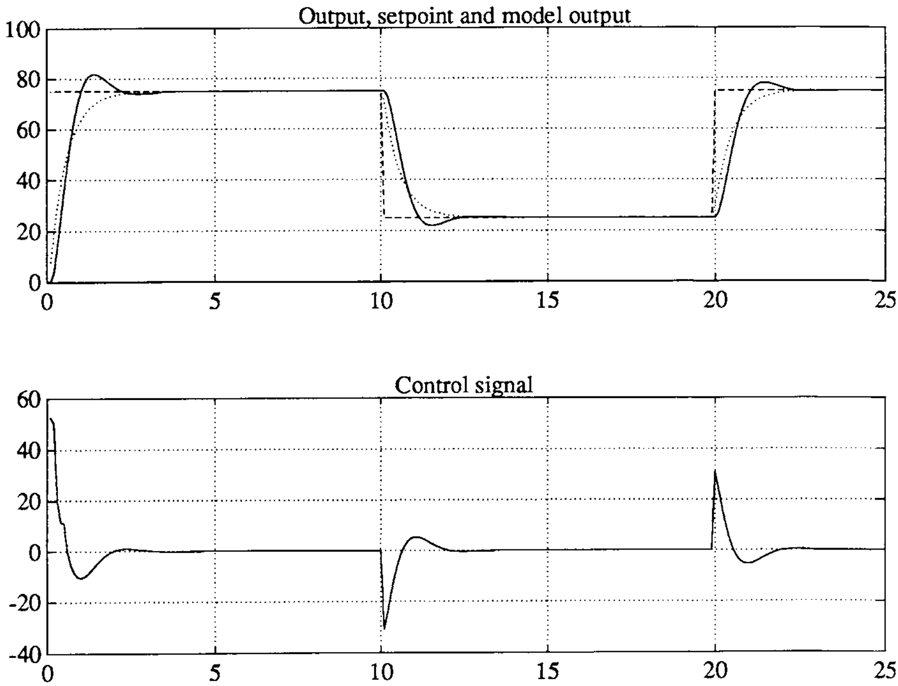

Figure 6.17. Implicit control-weighted model reference

Description

In example 6.2.14, exact model-reference control was achieved by setting $Q(s)=0$. For this example, $Q(s)$ is chosen as

$$Q(s) = \frac{s}{s+1} \tag{6.2.17.1}$$

this satifies the $Q(s)$ design rule on page 3-17 of vol. 1.

Programme interaction

runex 6 17
Example 6 of chapter 17: Implicit control-weighted model reference

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source     ========
======== Filters         ========
======== Control action  ========
Integral action           = FALSE  :=
======== Assumed system  ========
A (system denominator)  =   1.000000   0.000000   0.000000  :=
B (system numerator)    =   1.000000   1.000000  :=
======== Emulator design ========
P (model denominator)   =   0.500000   1.000000  :=
C (emulator denominator) =  0.500000   1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   1.000000
D          0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000   1.000000
B-         1.000000
C          0.500000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          1.000000   1.000000
F filter   0.500000   1.000000
G          0.250000   0.250000
G filter   0.500000   1.000000
I
E          0.250000
ED         0.000000
------------------------------------
======== STC type        ========
======== Tuner           ========
Initial Variance       = 100000.000000  :=
Forget time            = 1000.000000  :=
======== Controller      ========
Q numerator            =   1.000000   0.000000  :=
Q denominator          =   1.000000   1.000000  :=
======== Simulation      ========
======== Setpoint        ========
======== In Disturbance  ========
```

```
======== Out Disturbance ========
======== Actual system   ========
A (system denominator)   =    1.000000   1.000000   0.000000  :=
B (system numerator)     =    1.000000  10.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
-----------------------------------
A           1.000000    0.000000    0.000000
B           1.000000    1.000000
D           0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          1.000000    1.000000
B-          1.000000
C           0.500000    1.000000
P           0.500000    1.000000
Z+          1.000000
Z-          1.000000
Z-+         1.000000
-----------------------------------
F           0.747812    1.000071
F filter    0.500000    1.000000
G           0.241442    2.521147
G filter    0.500000    1.000000
I
E           0.250000
ED          0.000000
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \dfrac{Z(s)}{P(s)}\bar{w}(s)$.

Notice that the control signal is reduced with respect to that of example 6.2.14. The model following is no longer exact, but the use of the $Q(s)$ design rule ensures that there is no steady-state offset.

Further investigations

1. Try the effect of varying q in:

$$Q(s) = \frac{qs}{1+s} \tag{6.2.17.2}$$

2. Try the effect of varying T in:

$$Q(s) = \frac{s}{1+Ts} \tag{6.2.17.3}$$

3. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.17.4}$$

There is still no offset as, in this case, the system contains an integrator and so the control signal is zero in the steady-state.

4. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.17.5}$$

and $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.17.6}$$

Note that there is now an offset dependent on q.

5. Use the default value of $Q(s)$ but replace $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.17.7}$$

Note that the offset disappears.

6.2.18. IMPLICIT CONTROL-WEIGHTED POLE-PLACEMENT

Reference: Section 6.4; page 6-11. Section 3.6; page 3-16.

Description

In example 6.2.15, exact pole-placement control was achieved by setting $Q(s)=0$. For this example, $Q(s)$ is chosen as

$$Q(s) = \frac{0.1s}{s+1} \tag{6.2.18.1}$$

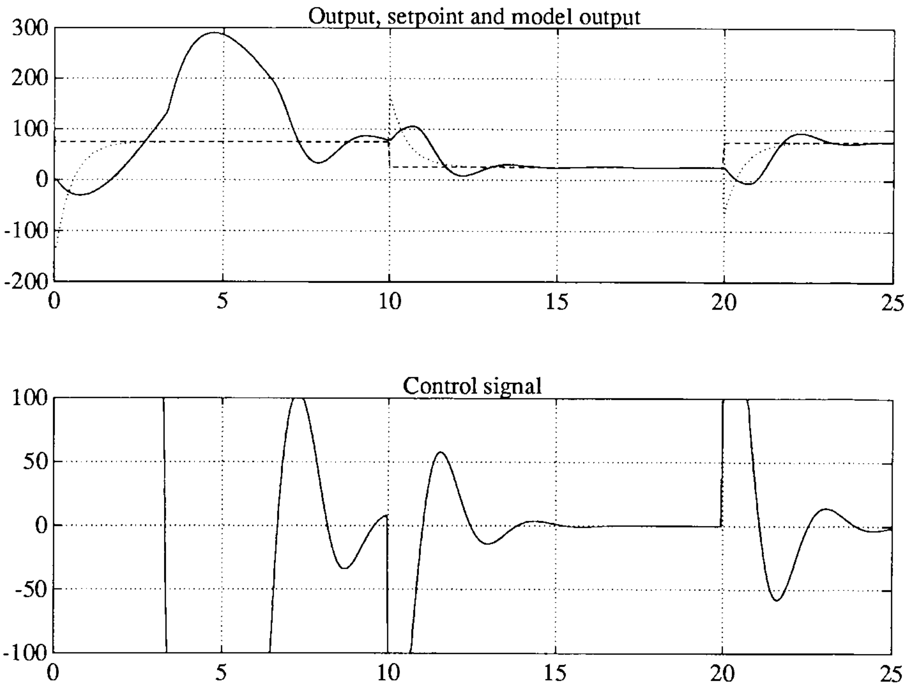

Figure 6.18. Implicit control-weighted pole-placement

this satifies the $Q(s)$ design rule on page 3-17 of vol. 1.

Programme interaction

runex 6 18
Example 6 of chapter 18: Implicit control-weighted pole-placement

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
======== Control action ========
======== Assumed system ========

```
A (system denominator)   =   1.000000   0.000000   0.000000  :=
B (system numerator)     =   1.000000   1.000000  :=
 ======== Emulator design ========
Z has factor B           = TRUE  :=
P (model denominator)    =   0.500000   1.000000  :=
C (emulator denominator) =   0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   1.000000
D          0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000
B-         1.000000   1.000000
C          0.500000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000   1.000000
Z-+        1.000000
-----------------------------------
F          0.000000   1.000000
F filter   0.500000   1.000000
G          0.250000
G filter   0.500000   1.000000
I
E          0.250000
ED         0.000000
-----------------------------------
 ======== STC type       ========
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
Cs (emulator denominator) =   1.000000   2.000000   1.000000  :=
 ======== Tuner          ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
 ======== Controller     ========
Q numerator              =   0.100000   0.000000  :=
Q denominator            =   1.000000   1.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000   1.000000   0.000000  :=
```

```
B (system numerator)    =  -1.000000   1.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
------------------------------------
      System polynomials
------------------------------------
A           1.000000   0.999768    0.000013
B          -0.999997   0.999758
D           0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          0.999758
B-         -1.000239   1.000000
C           0.500000   1.000000
P           0.500000   1.000000
Z+          1.000000
Z-         -1.000239   1.000000
Z-+         1.000000
------------------------------------
F           0.844699   0.990051
F filter    0.500000   1.000000
G           1.082946
G filter    0.500000   1.000000
I
E           0.250000
ED          0.000000
------------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

Notice that the control signal is reduced with respect to that of example 6.2.15. The model following is no longer exact, but the use of the $Q(s)$ design rule ensures that there is no steady-state offset.

Further investigations

1. Try the effect of varying q in:

$$Q(s) = \frac{qs}{1+s} \tag{6.2.18.2}$$

2. Try the effect of varying T in:

$$Q(s) = \frac{s}{1+Ts} \tag{6.2.18.3}$$

3. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.18.4}$$

   There is still no offset as, in this case, the system contains an integrator and so the control signal is zero in the steady-state.

4. Replace $Q(s)$ by:

$$Q(s)=q \tag{6.2.18.5}$$

   and $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.18.6}$$

   Note that there is now an offset dependent on q.

5. Use the default value of $Q(s)$ but replace $A(s)$ by:

$$A(s) = s^2 + 2s + 1 \tag{6.2.18.7}$$

   Note that the offset disappears.

6.2.19. TIME-DELAY SYSTEM (IMPLICIT)

Reference: Section 6.4; page 6-11. Section 3.7; page 3-18.

Description

This example corresponds to example 6.2.14, except that the system now is first order and has a time-delay of one unit.

$$\frac{B(s)}{A(s)} = e^{-s}\frac{1}{s} \tag{6.2.19.1}$$

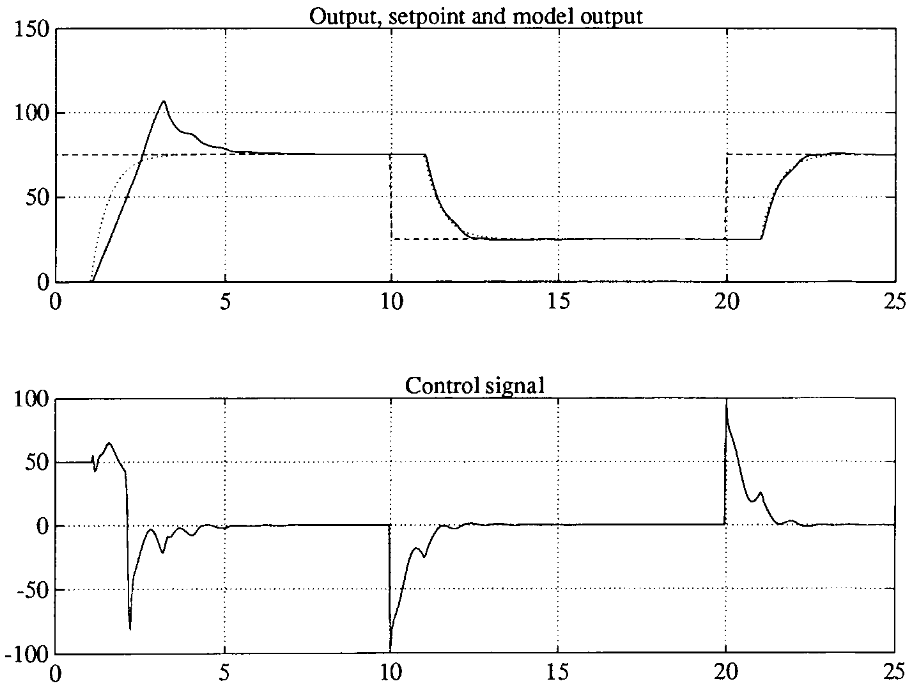

Figure 6.19. Time-delay system (implicit)

The corresponding emulator is based on the Pade approximation approach discussed in section I-2.6. But note that the simulation of the system uses an exact time-delay algorithm.

Programme interaction

runex 6 19
Example 6 of chapter 19: Time-delay system (implicit)

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========

```
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)   =   1.000000   2.000000  :=
B (system numerator)     =   3.000000  :=
 ======== Emulator design ========
P (model denominator)    =   0.500000   1.000000  :=
C (emulator denominator) =   1.000000  :=
Pade approximation order =          3  :=
-----------------------------------
     System polynomials
-----------------------------------
A         1.000000    2.000000
B         3.000000
D         0.000000    0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+         3.000000
B-        1.000000
C         1.000000
P         0.500000    1.000000
Z+         1.000000
Z-        1.000000
Z-+        1.000000
Pade       0.008333    0.100000    0.500000    1.000000
-----------------------------------
F         0.000000
F filter  1.000000
G         0.012500    0.150000    0.750000    1.500000
G filter  0.008333    0.100000    0.500000    1.000000
I
E         0.004167    0.050000    0.250000    0.500000
ED         0.000000    0.000000    0.000000    0.000000
-----------------------------------
 ======== STC type       ========
 ======== Tuner         ========
Initial Variance       =   10.000000  :=
Forget time            =  1000.000000  :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000   0.000000  :=
B (system numerator)     =   1.000000  :=
Time delay             =   1.000000  :=
Simulation running:
     25% complete
```

```
    50% complete
    75% complete
   100% complete
Time now is    25.000000
------------------------------------
      System polynomials
------------------------------------
A          1.000000    2.000000
B          3.000000
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         3.000000
B-         1.000000
C          1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
Pade       0.008333    0.100000    0.500000    1.000000
------------------------------------
F          0.999643
F filter   1.000000
G          0.003848    0.069639    0.246801    1.503904
G filter   0.008333    0.100000    0.500000    1.000000
I
E          0.004167    0.050000    0.250000    0.500000
ED         0.000000    0.000000    0.000000    0.000000
------------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

Despite the approximation involved, the model following is close. Note that the system output is delayed by one time unit.

Further investigations

1. Try the effect of using a lower order (for example 1) approximation to a time delay in the emula-

tor calculation.

6.2.20. IMPLICIT MODEL REFERENCE - DISTURBANCES

Reference: Section 6.4; page 6-11. Section 3.9; page 3-20.

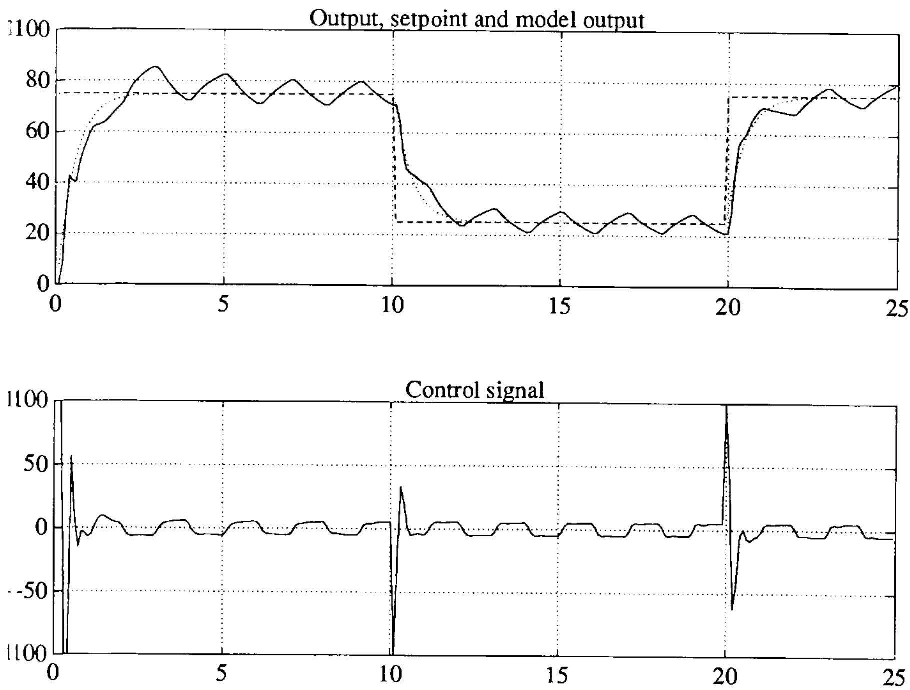

Figure 6.20. Implicit model reference - disturbances

Description

This example is identical to example 6.2.14 except that a square wave disturbance of amplitude 5 units and period two units is added to the system input. The purpose of the example is to illustrate the role of the polynomial $C(s)$ in determining closed-loop disturbance rejection. Initially, $C(s)=0.5s+1$.

Programme interaction

```
runex 6 20
Example 6 of chapter 20:  Implicit model reference - disturbances

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
Integral action           = FALSE  :=
 ======== Assumed system  ========
A (system denominator)    =    1.000000   0.000000   0.000000  :=
B (system numerator)      =    1.000000   1.000000  :=
 ======== Emulator design ========
P (model denominator)     =    0.500000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   1.000000
D          0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000   1.000000
B-         1.000000
C          0.500000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          1.000000   1.000000
F filter   0.500000   1.000000
G          0.250000   0.250000
G filter   0.500000   1.000000
I
E          0.250000
ED         0.000000
------------------------------------
 ======== STC type        ========
 ======== Tuner           ========
Initial Variance          = 100000.000000  :=
```

```
Forget time              = 1000.000000 :=
======== Controller      ========
======== Simulation      ========
======== Setpoint        ========
======== In Disturbance  ========
Square amplitude         =    5.000000 :=
Period                   =    2.000000 :=
======== Out Disturbance ========
======== Actual system   ========
A (system denominator)   =   1.000000   1.000000   0.000000 :=
B (system numerator)     =   1.000000  10.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
-----------------------------------
A           1.000000    0.000000    0.000000
B           1.000000    1.000000
D           0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          1.000000    1.000000
B-          1.000000
C           0.500000    1.000000
P           0.500000    1.000000
Z+          1.000000
Z-          1.000000
Z-+         1.000000
-----------------------------------
F           0.883282    0.995968
F filter    0.500000    1.000000
G           0.067507    1.783383
G filter    0.500000    1.000000
I
E           0.250000
ED          0.000000
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The effect of the disturbance is to perturb the system output; the control signal reacts to some extent to counteract this effect. The parameters do not converge to the 'correct' values, yet the control is reasonable.

Further investigations

1. The emulator denominator $C(s)$ is of the form 1+Ts. Try the effect of varying the time constant T (try, for example, T=0.1) of the emulator denominator C. How does this affect the system output and the control signal?

6.2.21. IMPLICIT MODEL REFERENCE PID

Reference: Section 6.4; page 6-11. Section 3.10; page 3-24.

Description

This example is identical to example 6.2.14 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral action" to "TRUE".

c) The degree of C is increased by one: $C = (1+0.5s)^2$.

d) The degree of $C(s)$ is increased by one: $C(s) = (1+0.5s)^3$.

Programme interaction

runex 6 21
Example 6 of chapter 21: Implicit model reference PID

======== *C S T C Version 6.0* ========

Enter all variables (y/n, default n)?

======== *Data Source* ========

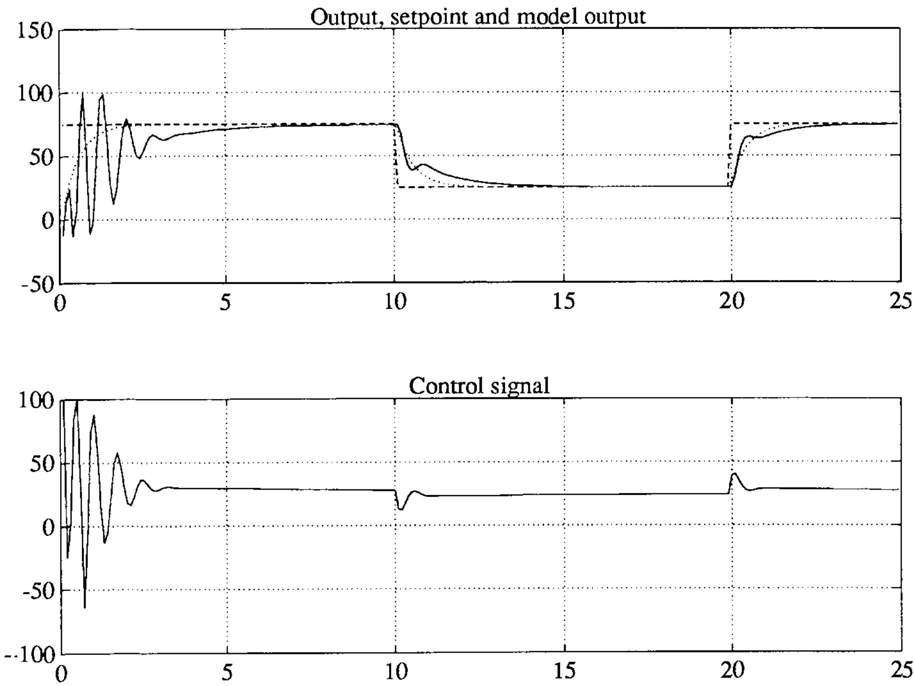

Figure 6.21. Implicit model reference PID

======== Filters ========
======== Control action ========
Integral action = TRUE :=
======== Assumed system ========
A (system denominator) = 1.000000 0.000000 0.000000 :=
B (system numerator) = 1.000000 1.000000 :=
======== Emulator design ========
P (model denominator) = 0.500000 1.000000 :=
*C (emulator denominator) = 0.500000 1.000000 * :=*
Next factor ...
C (emulator denominator) = 0.500000 1.000000 :=

System polynomials

A	*1.000000*	*0.000000*	*0.000000*	*0.000000*
B	*1.000000*	*1.000000*	*0.000000*	
D	*0.000000*	*0.000000*	*0.000000*	

```
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000    1.000000    0.000000
B-         1.000000
C          0.250000    1.000000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          0.750000    1.500000    1.000000
F filter   0.250000    1.000000    1.000000
G          0.125000    0.125000    0.000000
G filter   0.250000    1.000000    1.000000
I
E          0.125000
ED         0.000000
------------------------------------
 ======== STC type       ========
Tuning initial conditions = TRUE  :=
 ======== Tuner          ========
Initial Variance       =  100.000000  :=
Forget time            =  1000.000000  :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance ========
Step amplitude         =  -25.000000  :=
 ======== Out Disturbance ========
 ======== Actual system  ========
A (system denominator)  =   1.000000   1.000000   0.000000  :=
B (system numerator)    =  10.000000   1.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.000000    0.000000    0.000000
B          1.000000    1.000000    0.000000
D          0.000000    0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000    1.000000    0.000000
```

B-	*1.000000*		
C	*0.250000*	*1.000000*	*1.000000*
P	*0.500000*	*1.000000*	
Z+	*1.000000*		
Z-	*1.000000*		
Z-+	*1.000000*		

F	*0.593330*	*1.599766*	*1.000000*
F filter	*0.250000*	*1.000000*	*1.000000*
G	*1.018953*	*-0.108729*	*0.000000*
G filter	*0.250000*	*1.000000*	*1.000000*
I			
E	*0.125000*		
ED	*0.000000*		

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

Note that the response is better than that of the non-adaptive controller (example 3.2.8) because the initial conditions (corresponding to the offset) are estimated.

Further investigations

1. Try the controller of example 6.2.14, but with the disturbance. (Set integral action to FALSE and set $C(s) = 0.5s+1$. What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input disturbance. Explain what you observe.
3. Try the effect of not estimating an initial condition.

6.2.22. IMPLICIT POLE-PLACEMENT PID

Reference: Section 6.4; page 6-11. Section 3.10; page 3-25.

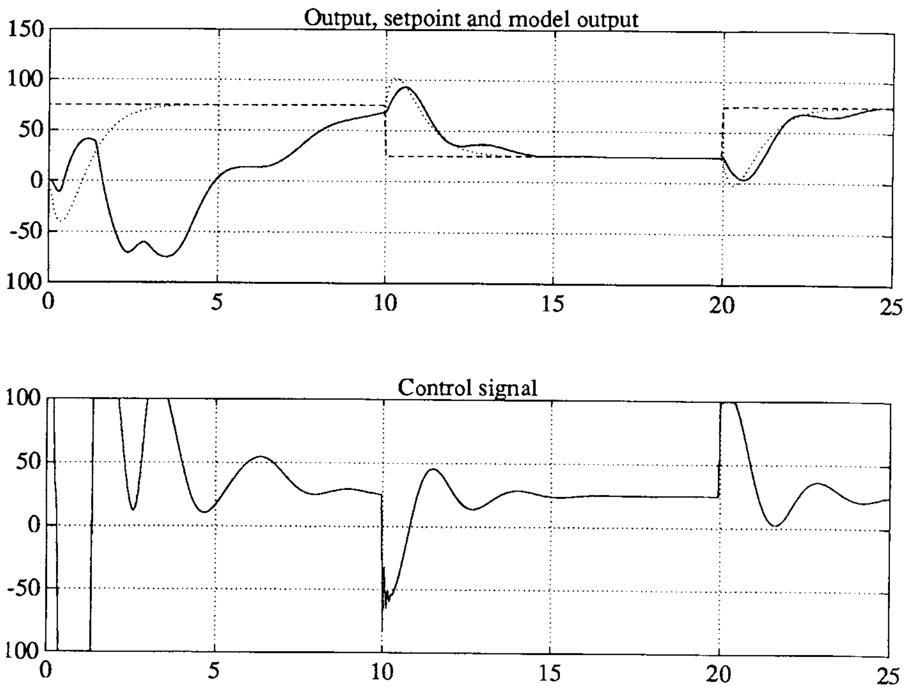

Figure 6.22. Implicit pole-placement PID

Description

This example is identical to example 6.2.15 except that:

a) A constant of value -25 is added to the system input

b) The assumption that there is a constant offset is built in by setting "Integral action" to "TRUE".

c) The degree of C is increased by one: $C = (1+0.5s)^2$.

d) The sample interval is decreased to 0.01 to give a satisfactory approximation.

Programme interaction

runex 6 22
Example 6 of chapter 22: Implicit pole-placement PID

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source     ========
======== Filters         ========
======== Control action  ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)   =   1.000000   0.000000   0.000000  :=
B (system numerator)     =   1.000000   1.000000  :=
======== Emulator design ========
Z has factor B           = TRUE  :=
P (model denominator)    =   0.500000   1.000000 *  :=
Next factor ...
P (model denominator)    =   0.500000   1.000000  :=
C (emulator denominator) =   0.500000   1.000000 *  :=
Next factor ...
C (emulator denominator) =   0.500000   1.000000  :=
------------------------------------
     System polynomials
------------------------------------
A          1.000000   0.000000   0.000000   0.000000
B          1.000000   1.000000   0.000000
D          0.000000   0.000000   0.000000
------------------------------------
     Design polynomials
------------------------------------
B+         1.000000   0.000000
B-         1.000000   1.000000
C          0.250000   1.000000   1.000000
P          0.250000   1.000000   1.000000
Z+         1.000000
Z-         1.000000   1.000000
Z-+        1.000000
------------------------------------
F          0.500000   1.000000   1.000000
F filter   0.250000   1.000000   1.000000
G          0.062500   0.000000   0.000000
G filter   0.250000   1.000000   1.000000
I
E          0.062500   0.000000
ED         0.000000   0.000000
------------------------------------
======== STC type        ========
Tuning initial conditions = TRUE  :=
======== Identification  ========
Initial Variance         =  100.000000  :=
Forget time              = 1000.000000  :=
```

```
Identifying rational part = TRUE  :=
Identifying delay         = FALSE  :=
 ======== Tuner          ========
Dead band                 =    0.000000  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
Step amplitude            =   -25.000000  :=
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)    =    1.000000   1.000000   0.000000  :=
B (system numerator)      =   -1.000000   1.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
-----------------------------------
A           1.000000    1.000123    0.001197    0.000000
B          -1.001057    0.995579    0.000000
D          -0.534429   25.114912  -24.766324
-----------------------------------
      Design polynomials
-----------------------------------
B+          0.995579    0.000000
B-         -1.005502    1.000000
C           0.250000    1.000000    1.000000
P           0.250000    1.000000    1.000000
Z+          1.000000
Z-         -1.005502    1.000000
Z-+         1.000000
-----------------------------------
F           2.293963    3.224380    1.000000
F filter    0.250000    1.000000    1.000000
G           0.058552    2.731816    0.000000
G filter    0.250000    1.000000    1.000000
I
E           0.062500    0.000000
ED          0.000000    0.000000
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The effect of the disturbance is, in the short term, to spoil the closed-loop response; but, in the long term, the response is not affected. Note that the steady-state control signal has a value of +25 to compensate for the disturbance: the controller has integral action.

Further investigations

1. Try the controller of example 6.2.15, but with the disturbance. (Set "integral action" to "FALSE" and set $C(s) = 0.5s+1$. What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input disturbance. Explain what you observe.

6.2.23. DETUNED MODEL-REFERENCE

Reference: Section 3.11; page 3-28, section 6.4 p 6-11.

Description

Example 3.2.10 illustrates the use of a reference model with one pole and one zero:

$$\frac{Z(s)}{P(s)} = \frac{0.03s+1}{0.3s+1} \tag{6.2.23.1}$$

together with control weighting:

$$Q(s) = \frac{qs}{0.03s+1} \tag{6.2.23.2}$$

In this example q=0.05 is used initially. An implicit self-tuning version is used in this example.

Programme interaction

runex 6 23
Example 6 of chapter 23: Detuned model-reference

======== C S T C Version 6.0 ========

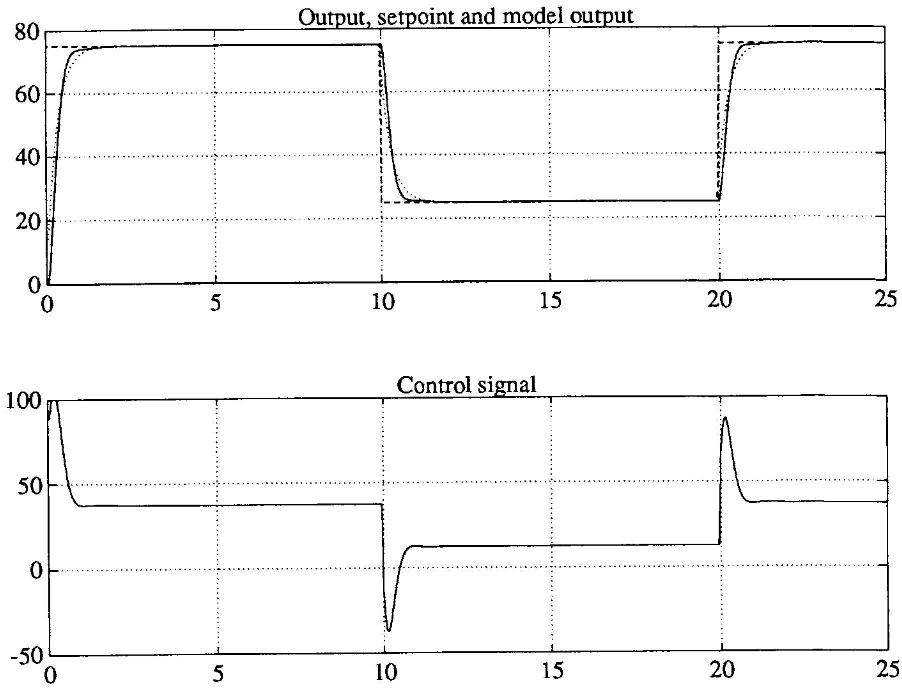

Figure 6.23. Detuned model-reference

Enter all variables (y/n, default n)?

```
 ======== Data Source      ========
 ======== Filters          ========
 ======== Control action   ========
Integral action           = TRUE  :=
 ======== Assumed system   ========
A (system denominator)    =   1.000000   0.000000  :=
B (system numerator)      =   1.000000  :=
 ======== Emulator design ========
Z-+ (Z- not including B)  =   0.030000   1.000000  :=
P (model denominator)     =   0.300000   1.000000  :=
C (emulator denominator)  =   0.300000   1.000000  :=
------------------------------------
       System polynomials
------------------------------------
A           1.000000    0.000000    0.000000
```

```
B          1.000000    0.000000
D          0.000000    0.000000    0.000000
------------------------------------
     Design polynomials
------------------------------------
B+          1.000000    0.000000
B-         1.000000
C          0.300000    1.000000
P          0.300000    1.000000
Z+          1.000000
Z-         0.030000    1.000000
Z-+         0.030000    1.000000
------------------------------------
F          0.570000    1.000000
F filter   0.300000    1.000000
G          0.072900    0.000000
G filter   0.009000    0.330000    1.000000
I
E          0.072900
ED          0.000000
------------------------------------
 ======== STC type       ========
Tuning initial conditions = FALSE  :=
 ======== Tuner          ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
 ======== Controller     ========
Q numerator            =    0.050000    0.000000  :=
Q denominator          =    0.030000    1.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000    1.000000  :=
B (system numerator)    =    2.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    0.000000
D          0.000000    0.000000    0.000000
------------------------------------
```

Design polynomials

B+	*1.000000*	*0.000000*	
B-	*1.000000*		
C	*0.300000*	*1.000000*	
P	*0.300000*	*1.000000*	
Z+	*1.000000*		
Z-	*0.030000*	*1.000000*	
Z-+	*0.030000*	*1.000000*	
F	*0.495087*	*1.000000*	
F filter	*0.300000*	*1.000000*	
G	*0.150603*	*0.000000*	
G filter	*0.009000*	*0.330000*	*1.000000*
I			
E	*0.072900*		
ED	*0.000000*		

Discussion

The performance is similar to the non-adaptive case as the parameters rapidly converge to their correct values. This example is taken further in chapter 7, where robustness to neglected dynamics is considered.

Further investigations

1. Examine the effect of varying the parameter q.
2. Examine the effect of varying the initial variance.

6.2.24. IMPLICIT PREDICTIVE CONTROL

Reference: Sections 3.7&8; page 3-18 and section 6.4 page 6-11.

Description

A predictive emulator in a feedback loop was discussed in example 3.2.11. In this example, the emulator is tuned using an implicit algorithm.

The open loop system has a first order rational part with unit time constant together with a unit delay

$$e^{-sT}\,\frac{B(s)}{A(s)} = e^{-s}\,\frac{1}{1+s} \qquad (6.2.24.1)$$

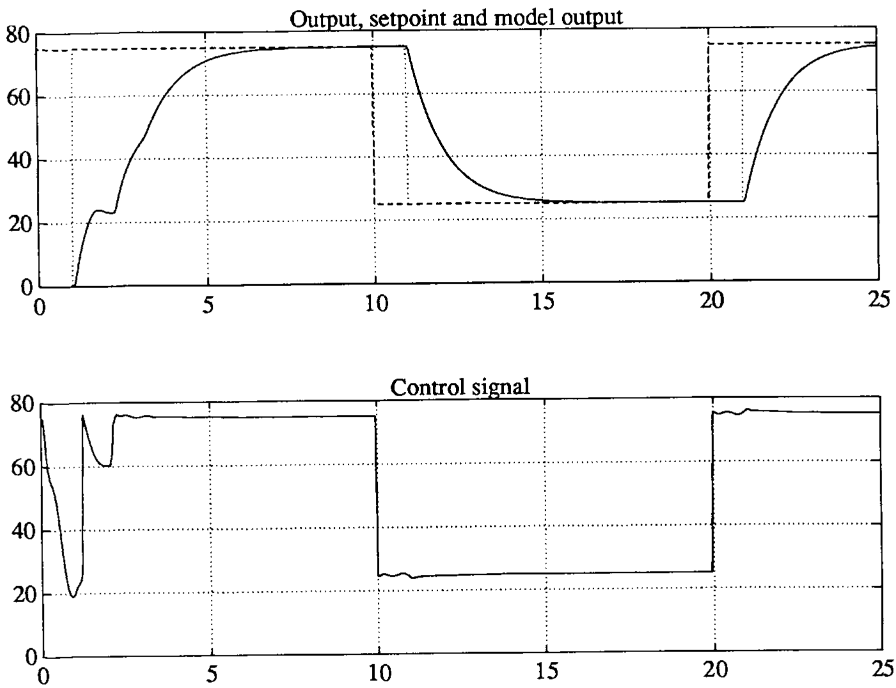

Figure 6.24. Implicit predictive control

$Q(s)$ is chosen to be an inverse PI controller:

$$\frac{1}{Q(s)} = 1 + \frac{1}{s} \tag{6.2.24.2}$$

Programme interaction

runex 6 24
Example 6 of chapter 24: Implicit predictive control

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

```
======== Data Source     ========
======== Filters         ========
Sample Interval          =   0.050000  :=
======== Control action  ========
Integral action          = FALSE  :=
======== Assumed system  ========
A (system denominator)   =   1.000000   0.000000  :=
B (system numerator)     =   2.000000  :=
Time delay               =   1.000000  :=
======== Emulator design ========
P (model denominator)    =   1.000000  :=
C (emulator denominator) =   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A         1.000000   0.000000
B         2.000000
D         0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+        2.000000
B-        1.000000
C         1.000000
P         1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
Pade      0.000595   0.011905   0.107143   0.500000   1.000000
-----------------------------------
F         1.000000
F filter  1.000000
G         0.000000   0.047619   0.000000   2.000000
G filter  0.000595   0.011905   0.107143   0.500000   1.000000
I
E         0.000000   0.023810   0.000000   1.000000
ED        0.000000   0.000000   0.000000   0.000000
-----------------------------------
======== STC type        ========
Tuning initial conditions = FALSE  :=
======== Tuner           ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
======== Controller      ========
Q numerator              =   1.000000   0.000000  :=
Q denominator            =   1.000000   1.000000  :=
======== Simulation      ========
======== Setpoint        ========
======== In Disturbance  ========
```

```
======== Out Disturbance ========
Step amplitude           =    0.000000 :=
Cos amplitude            =    0.000000 :=
======== Actual system   ========
A (system denominator)   =    1.000000  1.000000 :=
B (system numerator)     =    1.000000 :=
Time delay               =    1.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
-----------------------------------
      System polynomials
-----------------------------------
A         1.000000   0.000000
B         2.000000
D         0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+        2.000000
B-        1.000000
C         1.000000
P         1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
Pade      0.000595   0.011905   0.107143   0.500000   1.000000
-----------------------------------
F         0.366749
F filter  1.000000
G         0.000290   0.016045   0.050366   0.633169
G filter  0.000595   0.011905   0.107143   0.500000   1.000000
I
E         0.000000   0.023810   0.000000   1.000000
ED        0.000000   0.000000   0.000000   0.000000
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t) = y(t-1)$.

Note that the response is as predicted: a delayed first-order response delayed by one unit.

Further investigations

1. Try the effect of varying the order of the Pade approximation. Note that zero corresponds to having no predictor, and the response is not good. What is the smallest satisfactory order?
2. Try varying the system time delay, keeping the assumed and actual delay the same. For each value of delay, find the minimum satisfactory Pade order. Note that for larger Pade orders, you may need to reduce the sample interval for numerical reasons.
3. Try the effect of choosing an incorrect time-delay, say 0.9 in place of 1.0. Find the maximum and minimum values of the assummed delay (actual delay=1) giving satisfactory performance.
4. Try putting integral action into the predictor (Integral action = TRUE, C = s+1) and use a Pade order of 3. Observe the performance with an output step disturbance, and compare to the integral-free case.
5. Add a sinusoidal disturbance to the system output, how does the performance depend on the amplitude of this signal and the system time-delay?

6.2.25. IMPLICIT LINEAR-QUADRATIC POLE-PLACEMENT

Reference: Section 3.4; page 3-14.

Description

This example is identical to example 6.2.15, except that the closed-loop poles are chosen to solve equation I-3.4.23:

$$P(s)P(-s) = B(s)B(-s) + \lambda A(s)A(-s)$$

That is, the poles are chosen to correspond to those given by linear-quadratic optimisation theory where λ is the linear-quadratic weighting.

Programme interaction

runex 6 25
Example 6 of chapter 25: Implicit linear-quadratic pole-placement

======== *C S T C Version 6.0* ========

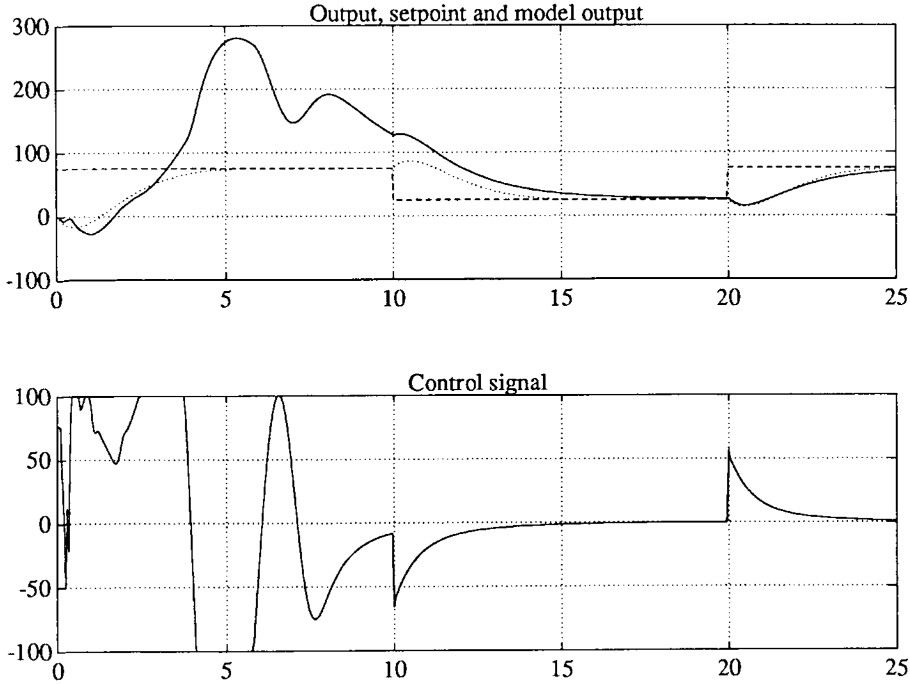

Figure 6.25. Implicit linear-quadratic pole-placement

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
======== Control action ========
======== Assumed system ========
A (system denominator) = 1.000000 0.000000 0.000000 :=
B (system numerator) = 1.000000 1.000000 :=
======== Emulator design ========
Linear-quadratic weight = 1.000000 :=
C (emulator denominator) = 0.500000 1.000000 :=

System polynomials

A	*1.000000*	*0.000000*	*0.000000*
B	*1.000000*	*1.000000*	
D	*0.000000*	*0.000000*	

```
------------------------------------
      Design polynomials
------------------------------------
B+          1.000000
B-          1.000000    1.000000
C           0.500000    1.000000
P           1.000000    1.732051    1.000000
Z+          1.000000
Z-          1.000000    1.000000
Z-+         1.000000
------------------------------------
F           1.232051    1.000000
F filter    0.500000    1.000000
G           0.500000    0.633975
G filter    0.500000    1.000000
I
E           0.500000    0.633975
ED          0.000000    0.000000
------------------------------------
 ======== STC type        ========
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance          = 100000.000000  :=
Forget time               = 1000.000000  :=
Cs (emulator denominator) =    1.000000    2.000000    1.000000  :=
 ======== Tuner           ========
Initial Variance          = 100000.000000  :=
Forget time               = 1000.000000  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)  =    1.000000    1.000000    0.000000  :=
B (system numerator)    =   -1.000000    1.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
------------------------------------
      System polynomials
------------------------------------
A           1.000000    0.999750    0.000011
B          -0.999998    0.999742
D           0.000000    0.000000
------------------------------------
```

Design polynomials

B+	*0.999742*		
B-	*-1.000256*	*1.000000*	
C	*0.500000*	*1.000000*	
P	*1.000258*	*2.449587*	*1.000000*
Z+	*1.000000*		
Z-	*-1.000256*	*1.000000*	
Z-+	*1.000000*		
F	*1.090896*	*0.976832*	
F filter	*0.500000*	*1.000000*	
G	*0.402335*	*2.656042*	
G filter	*0.500000*	*1.000000*	
I			
E	*0.500000*	*0.633975*	
ED	*0.000000*	*0.000000*	

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

As in example 3.2.2 note the typical behavious of a system with right-hand plane zeros: the output initially goes the wrong way in response to a step change.

Further investigations

1. Try the effect of varying the linear-quadratic weighting λ. How does this affect the system output and the control signal?

2. Try repeating this example using the same system as example 3.2.1 ($B(s) = 10+s$). How does the closed-loop response when using linear-quadratic control differ from that when using model-reference control?

6.2.26. IMPLICIT LINEAR-QUADRATIC PID

Reference: Section 3.4; page 3-14 and section 3.10; page 3-25.

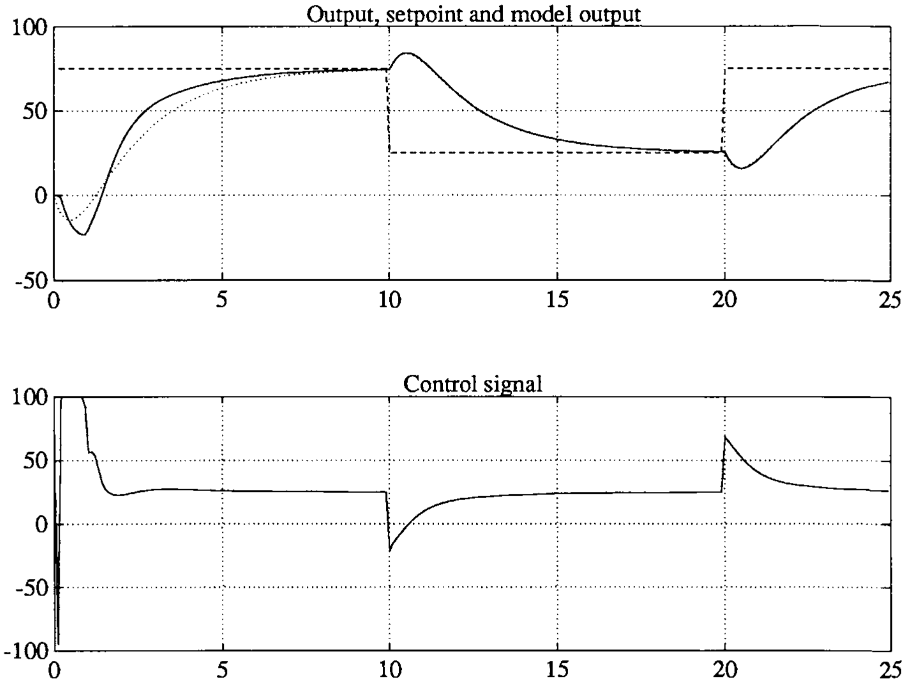

Figure 6.26. Implicit linear-quadratic PID

Description

This example is identical to example 25 except that:

a) A constant of value -25 is added to the system input.

b) The assumption that there is a constant offset is built in by setting "Integral action" to "TRUE".

c) The degree of $C(s)$ is increased by one: $C(s) = (1{+}0.5s)^2$.

d) The sample interval is decreased to 0.01 to give a satisfactory approximation.

Programme interaction

runex 6 26
Example 6 of chapter 26: Implicit linear-quadratic PID

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source      ========
======== Filters          ========
Sample Interval          =   0.010000  :=
======== Control action  ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)   =   1.000000   1.000000   0.000000  :=
B (system numerator)     =  -1.000000   1.000000  :=
======== Emulator design ========
Z has factor B           = TRUE  :=
Linear-quadratic poles   = TRUE  :=
Linear-quadratic weight  =   1.000000  :=
C (emulator denominator) =   0.500000   1.000000 *  :=
Next factor ...
C (emulator denominator) =   0.500000   1.000000  :=
-----------------------------------
     System polynomials
-----------------------------------
A          1.000000   1.000000   0.000000   0.000000
B         -1.000000   1.000000   0.000000
D          0.000000   0.000000   0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+         1.000000   0.000000
B-        -1.000000   1.000000
C          0.250000   1.000000   1.000000
P          1.000000   2.449490   1.000000
Z+         1.000000
Z-        -1.000000   1.000000
Z-+        1.000000
-----------------------------------
F          3.393304   4.449490   1.000000
F filter   0.250000   1.000000   1.000000
G          0.250000   4.755676   0.000000
G filter   0.250000   1.000000   1.000000
I
E          0.250000   4.755676
ED         0.000000   0.000000
-----------------------------------
======== STC type        ========
Tuning initial conditions = TRUE  :=
======== Identification  ========
Initial Variance         = 100000.000000  :=
Forget time              =  1000.000000  :=
```

```
Cs (emulator denominator) =     0.500000    1.000000 *  :=
Next factor ...
Cs (emulator denominator) =     0.500000    1.000000 *  :=
Next factor ...
Cs (emulator denominator) =     0.500000    1.000000  :=
Normalising Cs so that c0 = 1
Cs     1.000000     6.000000    12.000000     8.000000
======== Controller     ========
Maximum control signal    =  100.000000  :=
Minimum control signal    =  -100.000000  :=
======== Simulation     ========
======== Setpoint       ========
======== In Disturbance  ========
Step amplitude            =   -25.000000  :=
======== Out Disturbance ========
Step amplitude            =     0.000000  :=
======== Actual system   ========
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
------------------------------------
       System polynomials
------------------------------------
A          1.000000    0.999936    0.000054    0.000000
B         -0.999996    0.999727    0.000000
D         -0.128114   25.128045  -24.994685
------------------------------------
       Design polynomials
------------------------------------
B+         0.999727    0.000000
B-        -1.000269    1.000000
C          0.250000    1.000000    1.000000
P          1.000273    2.449849    1.000000
Z+         1.000000
Z-        -1.000269    1.000000
Z-+        1.000000
------------------------------------
F          3.393938    4.449862    1.000000
F filter   0.250000    1.000000    1.000000
G          0.250000    4.756236    0.000000
G filter   0.250000    1.000000    1.000000
I         -5.769595 -119.037979
E          0.250068    4.757535
ED        -0.128149    0.124881
------------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The effect of the disturbance is, in the short term, to spoil the closed-loop response; but, in the long term, the response is not affected. Note that the steady-state control signal has a value of +25 to compensate for the disturbance: the controller has integral action.

Further investigations

1. Try the controller of example 6.2.25, but with the disturbance. (Set "integral action" to "FALSE" and set $C(s) = 0.5s+1$ by setting the second factor =1). What is the effect of the input disturbance?
2. Repeat step 1, but with an output disturbance in place of an input disturbance. Explain what you observe.

6.2.27. DISCRETE-TIME IMPLICIT CONTROL

Reference: Section 6.4; page 6-11. Section 3.4; page 3-13.

Description

CSTC may be used for discrete-time simulation as well as continuous-time simulation. The system considered in this example is

$$\frac{z+0.9}{z^2 - 1.8z + 0.81} = \frac{z+0.9}{(z-0.9)^2} \qquad (6.2.27.1)$$

This system is controlled by a self-tuning model-reference controller with desired closed-loop system

$$\frac{1}{z-0.5} \qquad (6.2.27.2)$$

Programme interaction

runex 6 27
Example 6 of chapter 27: Discrete-time implicit control

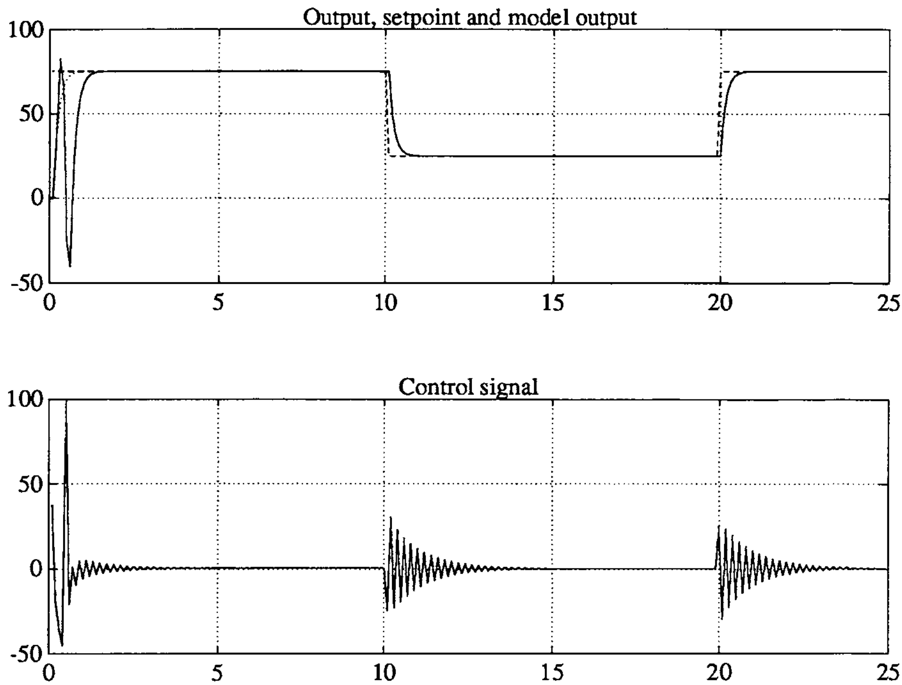

Figure 6.27. Discrete-time implicit control

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
Continuous-time? = FALSE :=
======== Control action ========
======== Assumed system ========
A (system denominator) = 1.000000 -2.000000 1.000000 :=
B (system numerator) = 1.000000 0.000000 :=
======== Emulator design ========
Z has factor B = FALSE :=
Z+ (nice model numerator) = 0.500000 :=
P (model denominator) = 1.000000 -0.500000 :=
C (emulator denominator) = 1.000000 -0.500000 :=

```
      System polynomials
-----------------------------------
A          1.000000   -2.000000    1.000000
B          1.000000    0.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000    0.000000
B-         1.000000
C          1.000000   -0.500000
P          1.000000   -0.500000
Z+         0.500000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          1.000000   -0.750000
F filter   0.500000   -0.250000
G          2.000000    0.000000
G filter   1.000000   -0.500000
I
E          2.000000
ED         0.000000
-----------------------------------
 ======== STC type        ========
Using lambda filter       = TRUE  :=
Tuning initial conditions = FALSE  :=
 ======== Tuner           ========
Initial Variance          = 100000.000000  :=
Forget time               = 1000.000000  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000  -1.800000   0.810000  :=
B (system numerator)     =   1.000000   0.900000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   -2.000000    1.000000
B          1.000000    0.000000
```

D	*0.000000*	*0.000000*

Design polynomials

B+	*1.000000*	*0.000000*
B-	*1.000000*	
C	*1.000000*	*-0.500000*
P	*1.000000*	*-0.500000*
Z+	*0.500000*	
Z-	*1.000000*	
Z-+	*1.000000*	
F	*0.800000*	*-0.560000*
F filter	*0.500000*	*-0.250000*
G	*2.000000*	*1.800000*
G filter	*1.000000*	*-0.500000*
I		
E	*2.000000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals: the system output y_i, the setpoint w_i and the ideal model output y_m

After an initial tuning period, the self-tuning controller adjusts its parameters to give exact model-following control. Note, however, the rather oscillatory control signal due to the cancellation of the system zero at z=-0.9. Such zeros are typical of discrete-time models of continuous-time systems.

Further investigations

1. In discrete-time, it is possible to set the roots of both $P(s)$ and $C(s)$ to be at the z-plane origin. Try this by setting $P(s) = C(s) = z + 0$ and $Z+ = 1$ to give unit steady-state gain.

6.2.28. DISCRETE-TIME EXPLICIT CONTROL

Reference: Section 6.4; page 6-11. Section 3.4; page 3-13.

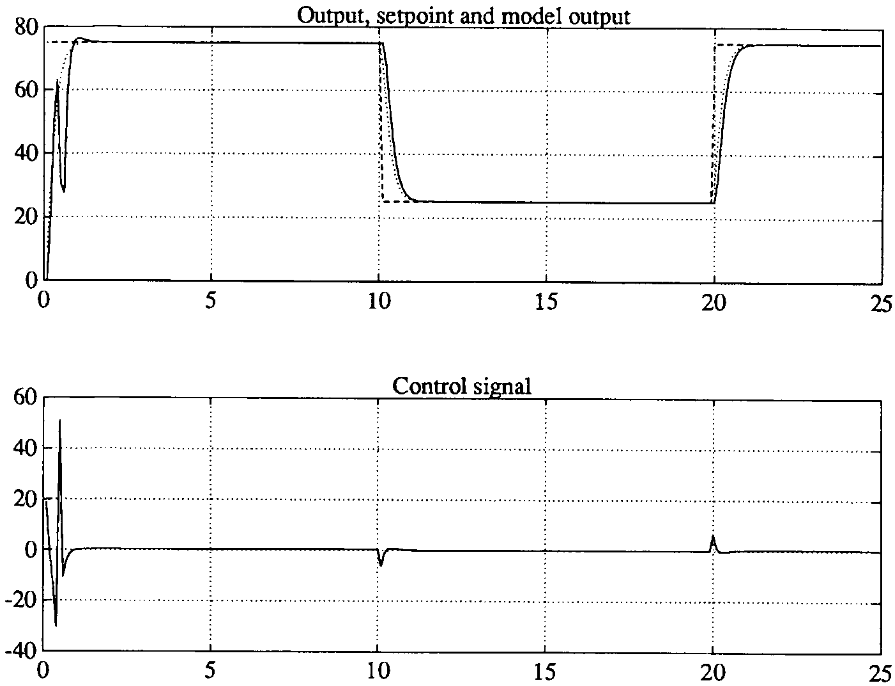

Figure 6.28. Discrete-time explicit control

Description

This is another discrete-time example based on the previous example. The difference here is that an explicit pole-placement algorithm is used in place of the implicit model-reference algorithm. The two closed-loop poles are placed at z=0.5, using

$$\frac{Z(z)}{P(z)} = \frac{0.25B(z)}{(z-0.5)^2} \qquad (6.2.28.1)$$

Programme interaction

runex 6 28
Example 6 of chapter 28: Discrete-time explicit control

```
 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters        ========
Continuous-time?          = FALSE  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000  -2.000000   1.000000  :=
B (system numerator)      =    1.000000   0.000000  :=
 ======== Emulator design ========
Z has factor B            = TRUE  :=
Z+ (nice model numerator) =    0.250000  :=
P (model denominator)     =    1.000000  -0.500000 *  :=
Next factor ...
P (model denominator)     =    1.000000  -0.500000  :=
C (emulator denominator)  =    1.000000  -0.500000  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000   -2.000000    1.000000
B          1.000000    0.000000
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000
B-         1.000000    0.000000
C          1.000000   -0.500000
P          1.000000   -1.000000    0.250000
Z+         0.250000
Z-         1.000000    0.000000
Z-+        1.000000
------------------------------------
F          0.625000   -0.500000
F filter   0.250000   -0.125000
G          4.000000   -0.500000
G filter   1.000000   -0.500000
I
E          4.000000   -0.500000
ED         0.000000    0.000000
------------------------------------
 ======== STC type       ========
Explicit self-tuning      = TRUE  :=
Identifying system        = TRUE  :=
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance          = 100000.000000  :=
```

```
Forget time              = 1000.000000 :=
Cs (emulator denominator) =   1.000000   2.000000   1.000000  :=
======== Controller     ========
======== Simulation     ========
======== Setpoint       ========
======== In Disturbance ========
======== Out Disturbance ========
======== Actual system  ========
A (system denominator)   =    1.000000  -1.800000   0.810000  :=
B (system numerator)     =    1.000000   0.900000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   -1.800000    0.810000
B          1.000000    0.900000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          1.900000
B-         0.526316    0.473684
C          1.000000   -0.500000
P          1.000000   -1.000000    0.250000
Z+          0.250000
Z-         0.526316    0.473684
Z-+         1.000000
-----------------------------------
F          0.469136   -0.354667
F filter   0.250000   -0.125000
G          7.600000    0.403457
G filter   1.000000   -0.500000
I
E          4.000000    0.212346
ED          0.000000    0.000000
-----------------------------------
```

Discussion

The displyed results are similar to those of the previous example exept taht the control is now much smoother as the system zero at z=-0.9 is no longer cancelled.

Further investigations

1. In discrete-time, it is possible to set the roots of both P and C to be at the Z plane origin. Try this by setting $P = (z + 0)*(z + 0)$ and $C = z + 0$ and Z + = 1 to give unit steady-state gain.

CHAPTER 7

Robustness of Self-Tuning Controllers

Aims. To investigate the effect of unmodelled system dynamics on the performance of self-tuning controllers. To investigate the role of control weighting on the robustness of self-tuning controllers with respect to unmodelled system dynamics.

7.1. IMPLEMENTATION DETAILS

The implementation is identical to that described in chapter 6, except that two additional methods of describinng the actual system are used. As in chapter 4, the neglected dynamics can be introduced into the simulated system by including additional factors in the system polynomials. This approach is not very satisfactory from the numerical point of view and high precision floating point arithmetic is required. Additionally, dynamics of two additional types can be included in the actual system:

$$G(s) = \frac{1}{(1+s\frac{T}{N})^N} \tag{7.1.1}$$

and

$$G(s) = be^{-\sqrt{sT}} \tag{7.1.2}$$

The former can be regarded as N (non-interacting) lags with time-constant $\frac{T}{N}$ in series; the latter can be regarded as the limiting case of N *interacting* lags with time constant $\frac{T}{\sqrt{N}}$ in series.

Both systems are implemented in procedure **MultiLag.** The former is implemented if the Boolean variable **Interactive** is FALSE; the latter is implemented if the Boolean variable **Interactive** is TRUE. N is contained in the variable **Lags**; if N=0, these additional dynamics are not implemented.

7.2. EXAMPLES

7.2.1. RHORS EXAMPLE: MODEL REFERENCE

Reference: Section 7.6&7, pages 7-20 - 7-30.

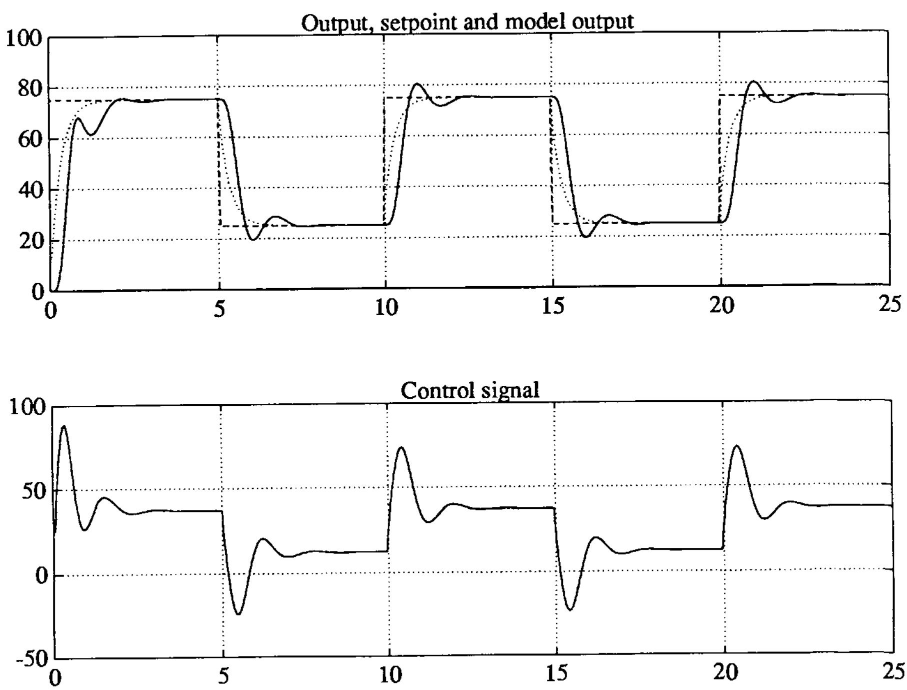

Figure 7.1. Rhors example: model reference

Description

In section I-7.7, a number of simulations are presented relating to an example of Rohrs. This example corresponds to the first simulation in that section; simulations 2 to 4 can be performed by changing parameters as described under "further investigations".

Programme interaction

```
runex 7 1
Example 7 of chapter 1:  Rhors example: model reference

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
Integral action          = TRUE  :=
 ======== Assumed system  ========
A (system denominator)    =   1.000000   3.000000  :=
B (system numerator)      =   0.000000  :=
 ======== Emulator design ========
Z has factor B           = FALSE  :=
Z-+ (Z- not including B)  =   0.030000   1.000000  :=
P (model denominator)     =   0.300000   1.000000  :=
C (emulator denominator)  =   0.300000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000   3.000000   0.000000
B          0.000000   0.000000
D          0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         0.000000   0.000000
B-         1.000000
C          0.300000   1.000000
P          0.300000   1.000000
Z+         1.000000
Z-         0.030000   1.000000
Z-+        0.030000   1.000000
-----------------------------------
F          0.329670   1.000000
```

```
F filter     0.300000    1.000000
G            0.000000    0.000000
G filter     0.009000    0.330000    1.000000
I
E            0.080110
ED
-----------------------------------
 ======== STC type        ========
Using lambda filter      = FALSE  :=
 ======== Tuner           ========
Initial Variance         =  100.000000  :=
Forget time              =  100.000000  :=
 ======== Controller      ========
Q numerator              =    0.200000    0.000000  :=
Q denominator            =    0.030000    1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
Square amplitude         =    0.000000  :=
Period                   =   25.000000  :=
 ======== Out Disturbance ========
Cos amplitude            =    0.000000  :=
Period                   =    0.123400  :=
 ======== Actual system   ========
A (system denominator)   =    1.000000    1.000000 *  :=
Next factor ...
A (system denominator)   =    1.000000    8.000000  100.000000  :=
B (system numerator)     =    2.000000 *  :=
Next factor ...
B (system numerator)     =  100.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000
-----------------------------------
      System polynomials
-----------------------------------
A            1.000000    3.000000    0.000000
B            0.000000    0.000000
D            0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+           0.000000    0.000000
B-           1.000000
C            0.300000    1.000000
P            0.300000    1.000000
```

```
Z+        1.000000
Z-        0.030000   1.000000
Z-+       0.030000   1.000000
-----------------------------------
F         0.598435   1.000000
F filter  0.300000   1.000000
G         0.186110   0.000000
G filter  0.009000   0.330000   1.000000
I
E         0.080110
ED
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The system output does not follow the model due to the neglected dynamics, but the self-tuning control system is stable.

Further investigations

1. Perform simulation 2 of section 1-7.7 by setting

$$Q_n(s) = 0.05s + 0.0 \qquad (7.2.1.1)$$

(Q numerator := 0.05 0).

2. Perform simulation 3 of section 1-7.7 by setting

$$Z^{-+}(s) = 1;\ Q_d(s) = 1 \qquad (7.2.1.2)$$

and setting "Using lambda filter" to TRUE.

3. Perform simulation 4 of section I-7.7 by setting

$$Z^{-+}(s) = 1;\ Q_d(s) = 1 \qquad (7.2.1.3)$$

setting "Using lambda filter" to TRUE and

$$Q_n(s) = 0.05s + 0.0 \qquad (7.2.1.4)$$

4. Try the effect of the other example of Rohrs where:

$$N(s) = \frac{229}{s^2 + 30s + 229} \tag{7.2.1.5}$$

5. Try the effect of a sinusoidal output disturbance on simulations 1 and 3 .

6. Try the effect of an input square-wave disturbance on simulations 1 and 3.

7.2.2. RHORS EXAMPLE: POLE PLACEMENT

Reference: Section 7.6&7, pages 7-20 - 7-30.

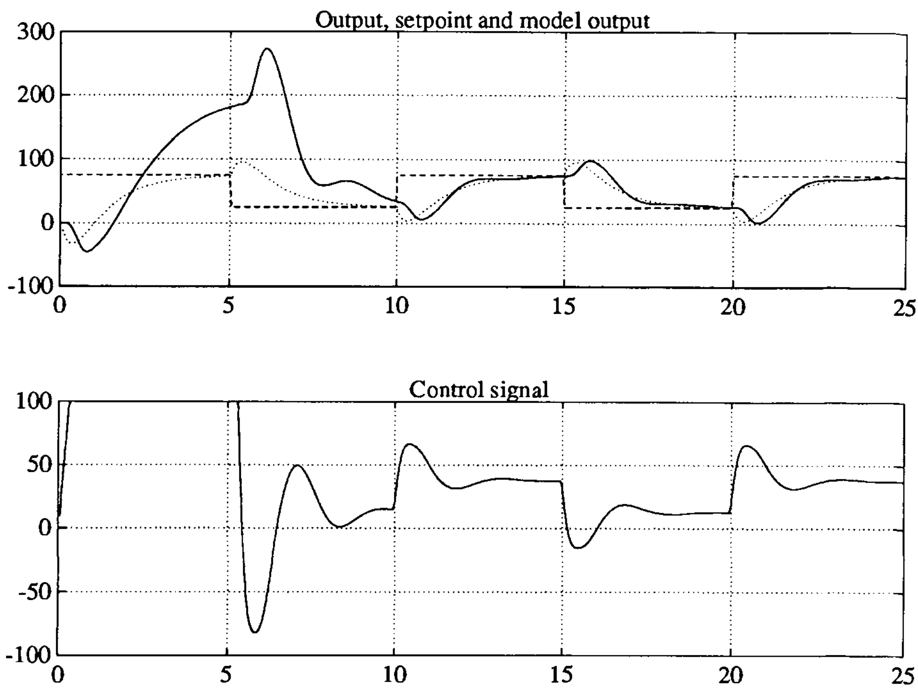

Figure 7.2. Rhors example: pole placement

Description

In section I-7.7, a number of simulations are presented relating to an example of Rohrs. This example corresponds to the 5th simulation; simulation 6 can be performed by changing parameters as described under "further investigations".

The control signal has been limited to a maximum value of 100.

Programme interaction

```
runex 7 2
Example 7 of chapter 2: Rhors example: pole placement

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000   0.000000   0.000000  :=
B (system numerator)      =    1.000000   1.000000  :=
 ======== Emulator design ========
Z has factor B            = TRUE  :=
P (model denominator)     =    0.300000   1.000000 *  :=
Next factor ...
P (model denominator)     =    1.000000   1.000000  :=
C (emulator denominator)  =    0.300000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A         1.000000   0.000000   0.000000
B         1.000000   1.000000
D         0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+        1.000000
B-        1.000000   1.000000
C         0.300000   1.000000
P         0.300000   1.300000   1.000000
Z+        1.000000
Z-        1.000000   1.000000
Z-+       1.000000
-----------------------------------
```

```
F           0.600000    1.000000
F filter    0.300000    1.000000
G           0.090000    0.090000
G filter    0.300000    1.000000
I
E           0.090000    0.090000
ED          0.000000    0.000000
-----------------------------------
 ======== STC type       ========
Tuning initial conditions = FALSE  :=
 ======== Identification  ========
Initial Variance          =  100.000000  :=
Forget time               =  100.000000  :=
Cs (emulator denominator) =    1.000000    2.000000    1.000000  :=
 ======== Controller      ========
Q numerator               =    0.200000    0.000000  :=
Q denominator             =    1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000    2.000000    1.000000 *  :=
Next factor ...
A (system denominator)   =    1.000000    8.000000  100.000000  :=
B (system numerator)     =   -2.000000    2.000000 *  :=
Next factor ...
B (system numerator)     =  100.000000  :=
Simulation running:
      25% complete
      50% complete
      75% complete
     100% complete
Time now is    25.000000
-----------------------------------
       System polynomials
-----------------------------------
A            1.000000    1.787453    0.891984
B           -1.893522    1.770595
D            0.000000    0.000000
-----------------------------------
       Design polynomials
-----------------------------------
B+           1.770595
B-          -1.069427    1.000000
C            0.300000    1.000000
P            0.300000    1.300000    1.000000
Z+           1.000000
Z-          -1.069427    1.000000
```

Z-+	*1.000000*	
F	*0.289599*	*0.251774*
F filter	*0.300000*	*1.000000*
G	*0.159354*	*1.485235*
G filter	*0.300000*	*1.000000*
I		
E	*0.090000*	*0.838834*
ED	*0.000000*	*0.000000*

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The system output does not follow the model due to the neglected dynamics, but the self-tuning control system is stable.

Further investigations

1. Perform simulation 6 of section 7.7 by setting

$$Q_n(s) = 0.05s + 0.0 \tag{7.2.2.1}$$

(Q numerator := 0.05 0).

2. Try the effect of the other example of Rohrs where:

$$N(s) = \frac{229}{s^2 + 30s + 229} \tag{7.2.2.2}$$

3. Try the effect of a sinusoidal output disturbance on simulations 1 and 3.
4. Try the effect of an input square-wave disturbance on simulation 5.

7.2.3. CONTROL OF TRANSMISSION LINE

Reference: Section 7.6&7, pages 7-20 - 7-30.

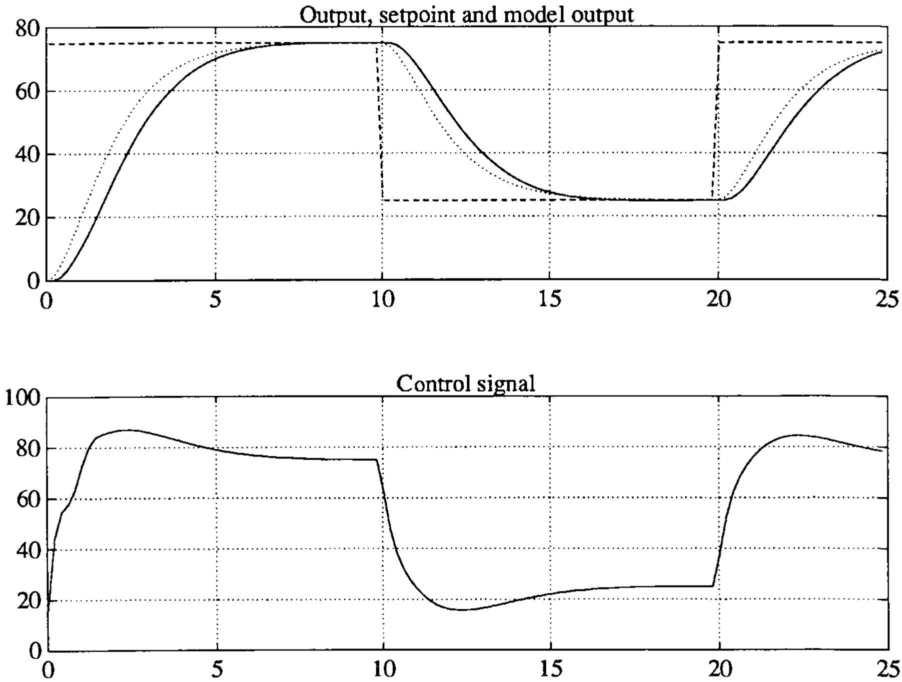

Figure 7.3. Control of transmission line

Description

A number of industrial processes, for example extruder barrel temperature, can be modelled as a large number of interacting first-order lags in series. An electrical analogy is an R-C ladder network. Such systems can be approximated by the transfer function:

$$G(s) = be^{-\sqrt{sT}} \tag{7.2.3.1}$$

where

$$T = \frac{RC}{N^2};\ b=1 \tag{7.2.3.2}$$

This simulation examines such a system with 5 interacting lags with time constant T=5 and b=1. The STC assumes a system with two poles and no zeros and an integrator in the disturbance. As discussed in chapter 6, the corresponding controller structure is PID.

These assumptions lead to neglected dynamics, and the purpose of this example is to investigate this.

Programme interaction

```
runex 7 3
Example 7 of chapter 3:  Control of transmission line

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter                  =          7  :=

 ======== Data Source     ========
 ======== Filters         ========
 ======== Control action  ========
Integral action          = TRUE  :=
 ======== Assumed system  ========
A (system denominator)    =    1.000000   2.000000   1.000000  :=
B (system numerator)      =    1.000000  :=
 ======== Emulator design ========
Z-+ (Z- not including B)  =    1.000000  :=
P (model denominator)     =    1.000000   1.000000 *  :=
Next factor ...
P (model denominator)     =    1.000000   1.000000  :=
C (emulator denominator)  =    0.500000   1.000000 *  :=
Next factor ...
C (emulator denominator)  =    0.500000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    2.000000    1.000000    0.000000
B          1.000000    0.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000    0.000000
B-         1.000000
C          0.250000    1.000000    1.000000
P          1.000000    2.000000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
```

```
------------------------------------
F          1.000000   2.000000   1.000000
F filter   0.250000   1.000000   1.000000
G          0.250000   1.000000   0.000000
G filter   0.250000   1.000000   1.000000
I
E          0.250000   1.000000
ED         0.000000
------------------------------------
 ======== STC type        ========
Using lambda filter      = TRUE :=
 ======== Tuner           ========
 ======== Controller      ========
Q numerator              =    0.400000   0.000000 :=
Q denominator            =    0.100000   1.000000 :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
Cos amplitude            =    0.000000 :=
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000 :=
B (system numerator)     =    1.000000 :=
D (initial conditions)   =    0.000000 :=
Number of lags           =           5 :=
Lag time constant        =    5.000000 :=
Interactive lags         = TRUE :=
Simulation running:
      25% complete
      50% complete
      75% complete
     100% complete
Time now is   25.020000
------------------------------------
      System polynomials
------------------------------------
A          1.000000   2.000000   1.000000   0.000000
B          1.000000   0.000000
D          0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000   0.000000
B-         1.000000
C          0.250000   1.000000   1.000000
P          1.000000   2.000000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
```

F	*-0.128252*	*1.790089*	*1.000000*
F filter	*0.250000*	*1.000000*	*1.000000*
G	*0.100435*	*1.212803*	*0.000000*
G filter	*0.250000*	*1.000000*	*1.000000*
I			
E	*0.250000*	*1.000000*	
ED	*0.000000*		

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The system output does not exactly follow the model due to the neglected dynamics and to control weighting, but the self-tuning control system is stable.

Further investigations

1. Try varying P and Q to investigate the region of stability.

7.2.4. LQ CONTROL OF TRANSMISSION LINE

Reference: Section 7.6&7, pages 7-20 - 7-30.

Description

This example uses the identical system to the previous example, but the controller is of the explicit LQ variety.

Programme interaction

runex 7 4
Example 7 of chapter 4: LQ control of transmission line

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========

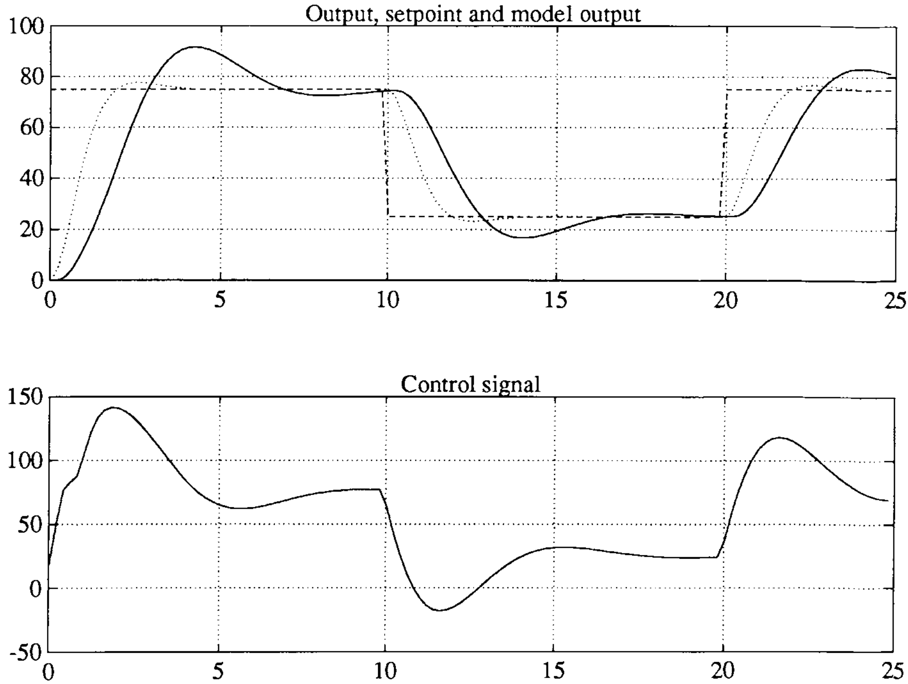

Figure 7.4. LQ control of transmission line

```
======== Filters         ========
======== Control action  ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)   =    1.000000    2.000000    1.000000  :=
B (system numerator)     =    1.000000  :=
======== Emulator design ========
Z-+ (Z- not including B)  =    1.000000  :=
C (emulator denominator)  =     0.500000     1.000000 *  :=
Next factor ...
C (emulator denominator)  =     0.500000     1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    2.000000    1.000000    0.000000
B          1.000000    0.000000
D          0.000000    0.000000
```

```
------------------------------------
     Design polynomials
------------------------------------
B+        1.000000   0.000000
B-        1.000000
C         0.250000   1.000000   1.000000
P         0.301511   0.799616   1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
------------------------------------
F         0.574430   1.448957   1.000000
F filter  0.250000   1.000000   1.000000
G         0.075378   0.350660   0.000000
G filter  0.250000   1.000000   1.000000
I
E         0.075378   0.350660
ED        0.000000
------------------------------------
 ======== STC type       ========
Explicit self-tuning     = TRUE  :=
Identifying system       = TRUE  :=
Tuning initial conditions = FALSE  :=
 ======== Identification ========
 ======== Controller     ========
Q numerator            =    0.400000   0.000000  :=
Q denominator          =    0.100000   1.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Cos amplitude          =    0.000000  :=
 ======== Actual system  ========
A (system denominator)   =    1.000000  :=
B (system numerator)     =    1.000000  :=
D (initial conditions)   =    0.000000  :=
Number of lags           =          5  :=
Lag time constant        =    5.000000  :=
Interactive lags         = TRUE  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.020000
------------------------------------
     System polynomials
------------------------------------
A         1.000000   2.501832   0.959328   0.000000
```

B	*0.955590*	*0.000000*	
D	*0.000000*	*0.000000*	

Design polynomials

B+	*0.955590*	*0.000000*	
B-	*1.000000*		
C	*0.250000*	*1.000000*	*1.000000*
P	*0.315411*	*0.904572*	*1.000000*
Z+	*1.000000*		
Z-	*1.000000*		
Z-+	*1.000000*		

F	*0.533013*	*1.574297*	*1.000000*
F filter	*0.250000*	*1.000000*	*1.000000*
G	*0.075351*	*0.328989*	*0.000000*
G filter	*0.250000*	*1.000000*	*1.000000*
I			
E	*0.078853*	*0.344278*	
ED	*0.000000*		

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$. In this case, the value of $P(s)$ used to generate $\bar{y}_m(s)$ is the initial value based on initial parameter estimates. The desired closed-loop denominator $P(s)$ is now chosen automatically by the LQ algorithm.

Further investigations

1. Try varying the control weighting λ to investigate the region of stability.
2. Try varying T, and note how both $A(s)$ and $P(s)$ change in sympathy.

CHAPTER 8

Non-Adaptive and Adaptive Robustness

Aims. To compare and contrast the non-adaptive and adaptive control of uncertain systems.

8.1. IMPLEMENTATION DETAILS

In addition to the self-tuning control algorithms described in chapter 6, the high-gain control given in equation I-8.3.3 (page I-8-9) is implemented. If the Boolean variable **UsingHighGainControl** is TRUE, then procedure **HighGainControl** is invoved in place of the self-tuning controller implemented in procedure

SelfTuningControl. To enable ready comparison between the two approaches, CSTC is organised so that the same parameters are read in for each case: the only difference between the **inlog.dat** files is the value of the Boolean variable **UsingHighGainControl.**

8.2. EXAMPLES

8.2.1. AN EXAMPLE OF HOROWITZ (IMPLICIT)

Reference: Section 8.2; page 8-4.

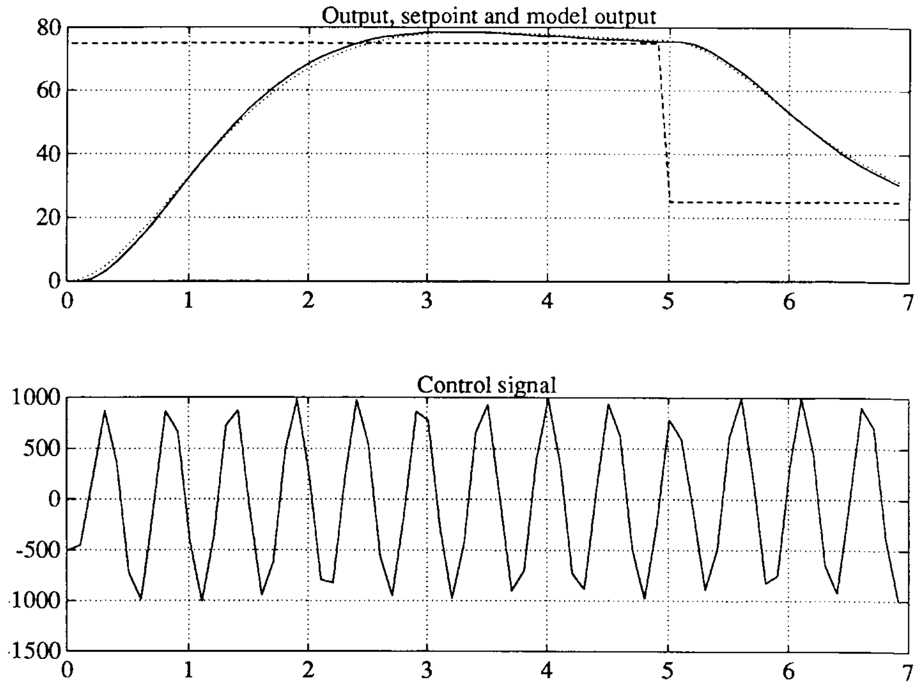

Figure 8.1. An example of Horowitz (Implicit)

Description

In section I-8.2, an example due to Horowitz is used to illustrate the theme of chapter 8, namely that self-tuning can be used in conjunction with Quantitative Feedback Theory to give a design which achieves robust control of uncertain but constant plants without undesirable sensor noise amplification.

This simulation uses this example with the time scales normalised by a factor of 10. The assumed plant has a gain K=1 and poles at : 0 , -0.1000 + 0.1732j and -0.1000 - 0.1732j.
The actual plant has a gain K=4 and poles at: 0, 0 + 0.2000j 0 - 0.2000j. The non-adaptive two-degree of freedom QFT controller is used in the simulation.

An implicit algorithm without the $\Lambda(s)$ filter is used. $Z(s)$ is chosen as:

$$Z(s) = Z^{-+}(s) = P(0.1s) = 0.005970s^2 + 0.107460s + 1 \quad (8.2.1.1)$$

Consequently, assuming no setpoint prefilter in the QFT design, $R(s)$ is chosen as:

$$R(s) = \frac{1}{Z(s)} = \frac{1}{0.005970s^2 + 0.107460s + 1} \tag{8.2.1.2}$$

The sensor noise properties are illustrated by adding a sinusoidal signal of amplitude 0.1 (0.2% of the average setpoint) to the plant output.

The emulator-based and the self-tuning versions are illustrated as further examples.

Programme interaction

```
runex 8 1
Example 8 of chapter 1: An example of Horowitz (Implicit)

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter                  =          8 :=

 ======== Data Source     ========
 ======== Filters        ========
Sample Interval          =    0.010000 :=
 ======== Control action  ========
Integral action          = FALSE  :=
 ======== Assumed system  ========
A (system denominator)    = 1000.000000 200.000000  40.000000   0.000000 :=
B (system numerator)      = 1250.000000 *  :=
Next factor ...
B (system numerator)      =    1.000000  :=
Normalising A and B so that a0 = 1
A      1.000000    0.200000    0.040000    0.000000
B      1.250000
 ======== Emulator design ========
Z-+ (Z- not including B)  =    0.005970    0.107460    1.000000  :=
P (model denominator)     =    0.597000    1.074600    1.000000  :=
C (emulator denominator)  =    0.250000    1.000000    1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.200000    0.040000    0.000000
B          1.250000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
```

```
B+        1.250000
B-        1.000000
C         0.250000   1.000000   1.000000
P         0.597000   1.074600   1.000000
Z+        1.000000
Z-        0.005970   0.107460   1.000000
Z-+       0.005970   0.107460   1.000000
------------------------------------
F         1.569997   1.940845   1.000000
F filter  0.250000   1.000000   1.000000
G         0.174846   0.821720
G filter  0.001493   0.032835   0.363430   1.107460   1.000000
I
E         0.139877   0.657376
ED
------------------------------------
 ======== STC type       ========
Using lambda filter     = FALSE  :=
 ======== Tuner          ========
Estimator on            = FALSE  :=
 ======== Controller     ========
Q numerator             =   0.116700 *  :=
Next factor ...
Q numerator             =   0.000287   0.029880   1.000000  :=
Q denominator           =   0.005970   0.107460   1.000000  :=
R numerator             =   1.000000  :=
R denominator           =   0.005970   0.107460   1.000000  :=
 ======== Simulation     ========
 ======== Setpoint       ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Cos amplitude           =   0.100000  :=
 ======== Actual system  ========
A (system denominator)  =  1000.000000   0.000000   40.000000   0.000000  :=
B (system numerator)    =  1250.000000 *  :=
Next factor ...
B (system numerator)    =   2.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is     7.010000
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

The system output is held close to the model output by the high gain control. The small ripple is due to the additive sinusoidal sensor noise. The control signal is, however, unsatisfactory due to the large sinusoidal component caused by the sensor noise.

Further investigations

1. Repeat the example but with the high-gain control replaced by the emulator version. This is achieved by setting "Chapter" to 7, and "Estimator on" to FALSE. Notice that the control signal fluctuations (due to the sinusoidal disturbances) are reduced. However, the reponse is now unsatifactory due to the sensitivity of this design method coupled with the error in the assumed system.
2. Repeat the example but with the high-gain control replaced by the self-tuning version. This is achieved by setting "Chapter" to 7, and "Estimator on" to TRUE. As in the previous case, the control signal fluctuations (due to the sinousoidal disturbances) are reduced. However, in contrast, the output response is now satisfactory.
3. If the $\Lambda(s)$ filter is used, then the use of $Z^{-+}(s)$ is not necessary. Try this by setting "Using lambda filter" to TRUE and resetting the terms corresponding to $Z^{-+}(s)$ in $Z^{-+}(s)$, $Q(s)$ and $R(s)$. Why is the result better?

8.2.2. AN EXAMPLE OF HOROWITZ (EXPLICIT)

Reference: Section 8.2; page 8-4.

Description

This is identical to example 8.2.1 except that an explicit self-tuning emulator is used in place of the implicit version. The theory does not cover this case - but it seems to work rather better than the implicit version.

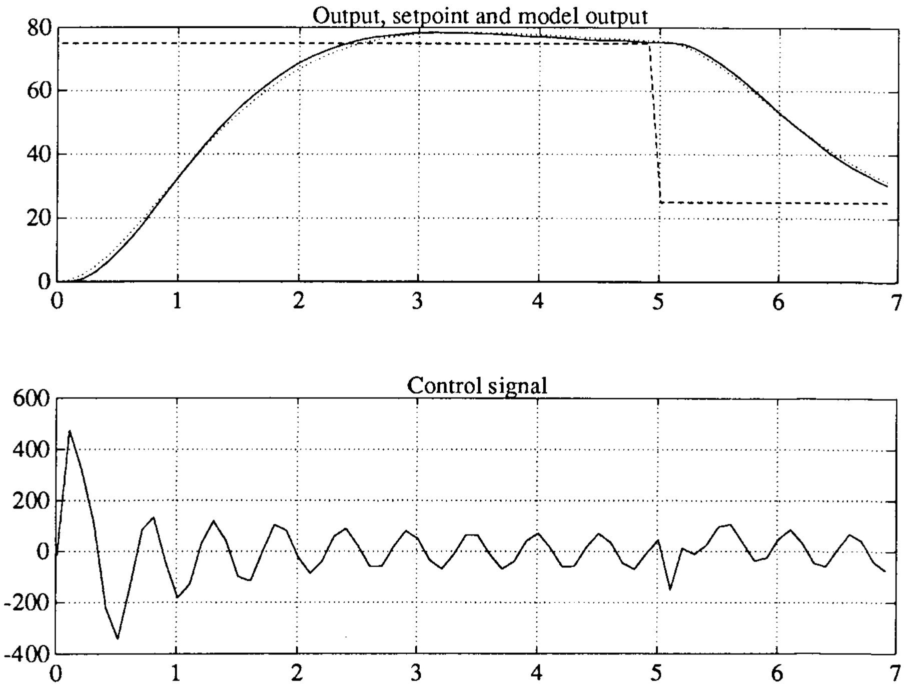

Figure 8.2. An example of Horowitz (Explicit)

The self-tuning version is implemented in the example; other versions are examined under 'further investigations'.

Programme interaction

runex 8 2
Example 8 of chapter 2: An example of Horowitz (Explicit)

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter = 7 :=

======== Data Source ========

```
 ======== Filters          ========
Sample Interval          =    0.010000  :=
 ======== Control action  ========
Integral action          = FALSE  :=
 ======== Assumed system  ========
A (system denominator)    =  1000.000000 200.000000  40.000000   0.000000  :=
B (system numerator)      =  1250.000000 *  :=
Next factor ...
B (system numerator)      =     1.000000  :=
Normalising A and B so that a0 = 1
A     1.000000    0.200000    0.040000    0.000000
B     1.250000
 ======== Emulator design ========
Z-+ (Z- not including B)  =    0.005970    0.107460    1.000000  :=
P (model denominator)     =    0.597000    1.074600    1.000000  :=
C (emulator denominator)  =    0.250000    1.000000    1.000000  :=
Small positive number     =    0.000001  :=
-------------------------------------
     System polynomials
-------------------------------------
A          1.000000    0.200000    0.040000    0.000000
B          1.250000
D          0.000000    0.000000
-------------------------------------
     Design polynomials
-------------------------------------
B+         1.250000
B-         1.000000
C          0.250000    1.000000    1.000000
P          0.597000    1.074600    1.000000
Z+         1.000000
Z-         0.005970    0.107460    1.000000
Z-+        0.005970    0.107460    1.000000
-------------------------------------
F          1.569997    1.940845    1.000000
F filter   0.250000    1.000000    1.000000
G          0.174846    0.821720
G filter   0.001493    0.032835    0.363430    1.107460    1.000000
I
E          0.139877    0.657376
ED
-------------------------------------
 ======== STC type         ========
 ======== Identification  ========
Estimator on             = TRUE  :=
 ======== Controller       ========
Q numerator              =    0.116700 *  :=
Next factor ...
Q numerator              =    0.000287    0.029880    1.000000  :=
```

```
Q denominator          =  0.005970  0.107460  1.000000 :=
R numerator            =  1.000000 :=
R denominator          =  0.005970  0.107460  1.000000 :=
======== Simulation    ========
======== Setpoint      ========
======== In Disturbance ========
======== Out Disturbance ========
Cos amplitude          =  0.100000 :=
======== Actual system ========
A (system denominator) = 1000.000000  0.000000  40.000000  0.000000 :=
B (system numerator)   = 1250.000000 * :=
Next factor ...
B (system numerator)   =  2.000000 :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   7.010000
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000  -0.000049  0.040018  -0.000003
B          2.499750
D          0.000000  0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         2.499750
B-         1.000000
C          0.250000  1.000000  1.000000
P          0.597000  1.074600  1.000000
Z+         1.000000
Z-         0.005970  0.107460  1.000000
Z-+        0.005970  0.107460  1.000000
-----------------------------------
F          1.701595  1.940279  1.000002
F filter   0.250000  1.000000  1.000000
G          0.347694  1.677883
G filter   0.001493  0.032835  0.363430  1.107460  1.000000
I
E          0.139091  0.671220
ED
-----------------------------------
```

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

The system output is held close to the model output; not by the high gain control, but by the tuning of the emulator. The small ripple is due to the additive sinusoidal sensor noise. The control signal is does not have a large sensor noise component.

Further investigations

1. Repeat the example but with the tuning switched off. This is achieved by setting"Estimator on" to FALSE. T. As in the previous case, the control signal fluctuations (due to the sinusoidal disturbances) are small, but the response is now poor.
2. In this explicit version, it is not necessary to use a realisable ϕ. Try the effect of deleting the $0.005970s^2+0.107460s+1$ from $Z^{-+}(s)$, $R(s)$ and $Q(s)$.

8.2.3. AN EXAMPLE OF ASTROM (IMPLICIT)

Reference: Section 8.2; page 8-4.

Description

Another example of applying QFT design to self-tuning controllers appears in a recent paper [1]. This example corresponds to that presented in the paper; the QFT design is to found in a paper by Astrom et al [2].

The plant is described by

$$\frac{B(s)}{A(s)} = \frac{k}{(1+Ts)^2};\ 1<k<4,\ 0.5<T<2. \tag{8.2.3.1}$$

As discussed in the paper [1] a possible set of self-tuning controller design parameters is:

1 Gawthrop, P.J.: "Quantitative feedback theory and self-tuning control", Proceedings of IEE conference "Control 88", Oxford, 1988.

2 Astrom, K.J., Neumann, L. and Gutman, P.O: (1986) "A comparison between robust and adaptive control of uncertain systems", Proceedings of the 2nd IFAC workshop on "Adaptive systems in Control and Signal Processing" Lund, Sweden.

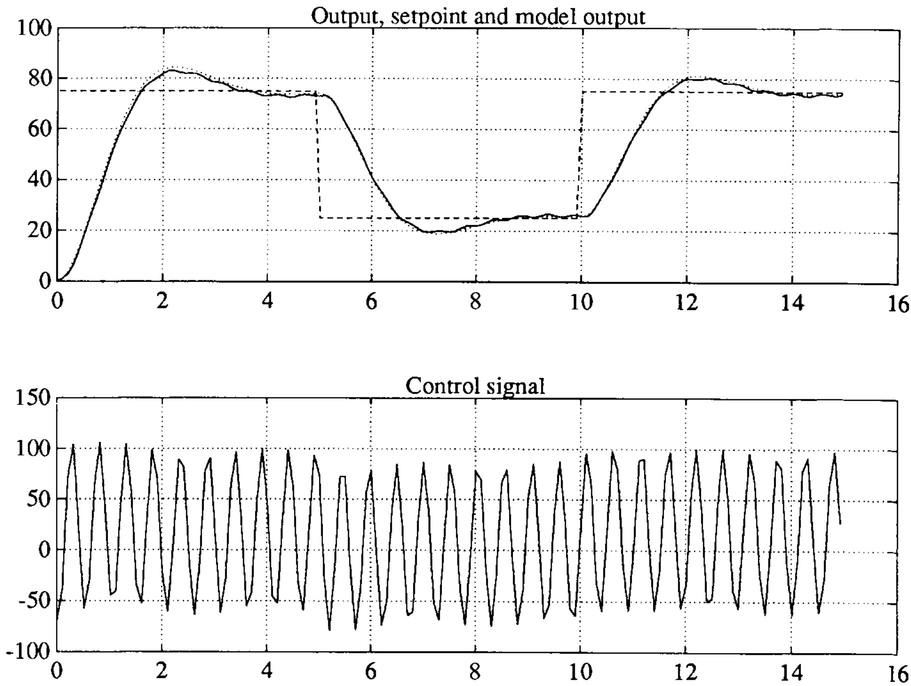

Figure 8.3. An example of Astrom (Implicit)

$$P(s) = (1+4s)(1+0.6666s); \tag{8.2.3.2}$$

$$Z(s) = P(0.1s) = (1+0.4s)(1+0.06666s) \tag{8.2.3.3}$$

$$Q(s) \tag{8.2.3.4}$$

$$= \frac{s(1+0.03333s)(1+0.0020s+0.0000040s^2)}{2Z(s)}$$

$$= \frac{s(1+0.03333s)(1+0.0020s+0.0000040s^2)}{2(1+0.4s)(1+0.06666s)}$$

and

$$R(s) = \frac{(1+4s)(1+0.6666s)}{(1+0.4s)(1+0.06666s)(1+0.6471s+0.3460s^2)} \tag{8.2.3.5}$$

In addition, the effect of the low-pass neglected dynamics:

$$N(s) = \frac{1}{1+0.05s} \tag{8.2.3.6}$$

will be illustrated.

This example is set up to use the QFT controller; the corresponding emulator and self-tuning versions are discussed as further examples.

Programme interaction

```
runex 8 3
Example 8 of chapter 3:  An example of Astrom (Implicit)

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter                  =          8  :=

 ======== Data Source     ========
 ======== Filters       ========
Sample Interval          =    0.010000  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000    1.000000 *  :=
Next factor ...
A (system denominator)    =    1.000000    1.000000  :=
B (system numerator)     =    2.000000  :=
 ======== Emulator design ========
Z-+ (Z- not including B)  =    0.400000    1.000000 *  :=
Next factor ...
Z-+ (Z- not including B)  =    0.066660    1.000000  :=
P (model denominator)     =    4.000000    1.000000 *  :=
Next factor ...
P (model denominator)     =    0.666600    1.000000  :=
C (emulator denominator)  =    1.000000    1.000000    1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A         1.000000    2.000000    1.000000    0.000000
B         2.000000    0.000000
D         0.000000    0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
```

```
B+        2.000000   0.000000
B-        1.000000
C         1.000000   1.000000   1.000000
P         2.666400   4.666600   1.000000
Z+        1.000000
Z-        0.026664   0.466660   1.000000
Z-+       0.026664   0.466660   1.000000
-----------------------------------
F         0.871367   3.657425   1.000000
F filter  1.000000   1.000000   1.000000
G         5.286332   3.085029   0.000000
G filter  0.026664   0.493324   1.493324   1.466660   1.000000
I
E         2.643166   1.542515
ED
-----------------------------------
 ======== STC type        ========
 ======== Tuner           ========
Initial Variance        =  100.000000  :=
Forget time             =  1000.000000  :=
Estimator on            = TRUE  :=
 ======== Controller      ========
Q numerator             =   1.000000   0.000000 *  :=
Next factor ...
Q numerator             =   0.033300   1.000000  :=
Q denominator           =   2.000000 *  :=
Next factor ...
Q denominator           =   0.400000   1.000000 *  :=
Next factor ...
Q denominator           =   0.066660   1.000000  :=
R numerator             =   4.000000   1.000000 *  :=
Next factor ...
R numerator             =   0.666600   1.000000  :=
R denominator           =   0.346000   0.647000   1.000000 *  :=
Next factor ...
R denominator           =   0.400000   1.000000 *  :=
Next factor ...
R denominator           =   0.066660   1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Cos amplitude           =   0.500000  :=
 ======== Actual system   ========
A (system denominator)  =   0.500000   1.000000 *  :=
Next factor ...
A (system denominator)  =   0.500000   1.000000  :=
B (system numerator)    =   4.000000  :=
Simulation running:
```

25% complete
50% complete
75% complete
100% complete
Time now is 15.010000

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The system output is held close to the model output; not by the high gain control, but by the tuning of the emulator. The small ripple is due to the additive sinusoidal sensor noise. The control signal is does not have a large sensor noise component.

Further investigations

1. Repeat the example but with the high-gain control replaced by the emulated version. This is achieved by setting "Chapter" to 7, and "Estimator on" to FALSE. To speed things up, it is possible to reduce the sample interval from 0.002 to 0.1 and still retain stability. Notice that the control signal fluctuations (due to the sinusoidal disturbances) are reduced. However, the reponse is now unsatifactory due to the sensitivity of this design method coupled with the error in the assumed system.

2. Repeat the example but with the high-gain control replaced by the emulated version. This is achieved by setting "Chapter" to 7, and "Estimator on" to TRUE. To speed things up, it is possible to reduce the sample interval from 0.002 to 0.1 and still retain stability. As in the previous case, the control signal fluctuations (due to the sinusoidal disturbances) are reduced. However, in contrast, the output response is now satisfactory.

8.2.4. AN EXAMPLE OF ASTROM (EXPLICIT)

Reference: Section 8.2; page 8-4.

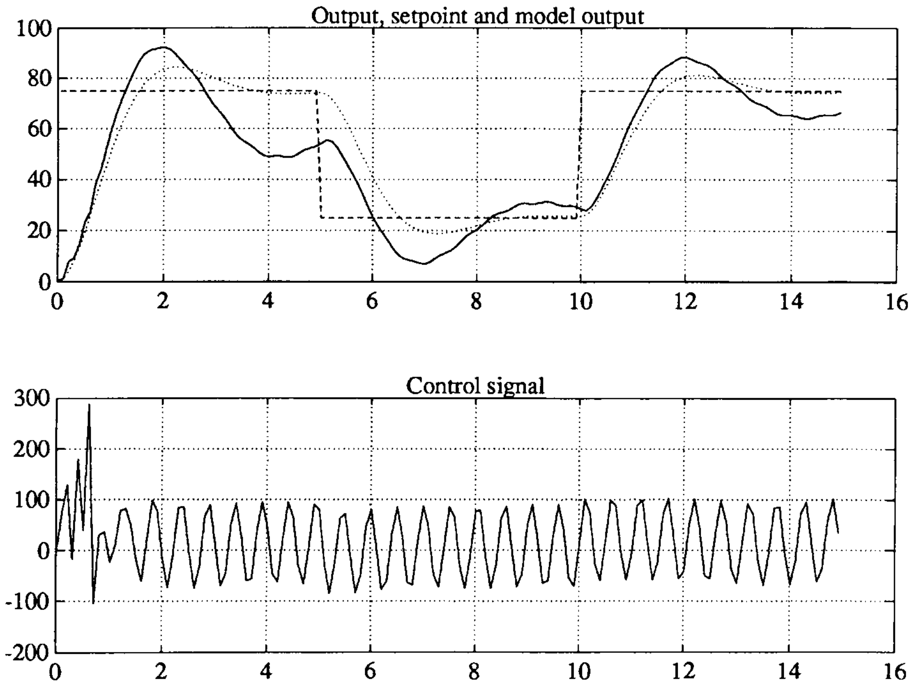

Figure 8.4. An example of Astrom (Explicit)

Description

This is identical to example 8.2.3 except that an explicit self-tuning emulator is used in place of the implicit version. The theory does not cover this case - but it seems to work.

The self-tuning version is used initially.

Programme interaction

runex 8 4
Example 8 of chapter 4: An example of Astrom (Explicit)

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

```
Chapter                    =          7  :=

 ======== Data Source     ========
 ======== Filters        ========
Sample Interval          =    0.010000  :=
 ======== Control action  ========
 ======== Assumed system  ========
A (system denominator)    =    1.000000    1.000000 *  :=
Next factor ...
A (system denominator)    =    1.000000    1.000000  :=
B (system numerator)      =    2.000000  :=
 ======== Emulator design ========
Z-+ (Z- not including B)  =    0.400000    1.000000 *  :=
Next factor ...
Z-+ (Z- not including B)  =    0.066660    1.000000  :=
P (model denominator)     =    4.000000    1.000000 *  :=
Next factor ...
P (model denominator)     =    0.666600    1.000000  :=
C (emulator denominator)  =    1.000000    1.000000    1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    2.000000    1.000000    0.000000
B          2.000000    0.000000
D          0.000000    0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          2.000000    0.000000
B-         1.000000
C          1.000000    1.000000    1.000000
P          2.666400    4.666600    1.000000
Z+          1.000000
Z-         0.026664    0.466660    1.000000
Z-+         0.026664    0.466660    1.000000
-----------------------------------
F          0.871367    3.657425    1.000000
F filter   1.000000    1.000000    1.000000
G          5.286332    3.085029    0.000000
G filter   0.026664    0.493324    1.493324    1.466660    1.000000
I
E          2.643166    1.542515
ED
-----------------------------------
 ======== STC type        ========
 ======== Identification  ========
Initial Variance        =  100.000000  :=
Forget time             =  1000.000000  :=
Estimator on            =  TRUE  :=
```

```
Cs (emulator denominator) =    1.000000   3.000000   3.000000   1.000000  :=
 ======== Controller      ========
Q numerator            =   1.000000   0.000000 *  :=
Next factor ...
Q numerator            =   0.033300   1.000000  :=
Q denominator          =   2.000000 *  :=
Next factor ...
Q denominator          =   0.400000   1.000000 *  :=
Next factor ...
Q denominator          =   0.066660   1.000000  :=
R numerator            =   4.000000   1.000000 *  :=
Next factor ...
R numerator            =   0.666600   1.000000  :=
R denominator          =   0.346000   0.647000   1.000000 *  :=
Next factor ...
R denominator          =   0.400000   1.000000 *  :=
Next factor ...
R denominator          =   0.066660   1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
Cos amplitude          =   0.500000  :=
 ======== Actual system   ========
A (system denominator)   =   0.500000   1.000000 *  :=
Next factor ...
A (system denominator)   =   0.500000   1.000000  :=
B (system numerator)     =   4.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    15.010000
-----------------------------------
     System polynomials
-----------------------------------
A        1.000000    4.000064    4.000088    0.000000
B       16.000220    0.000000
D        0.000000    0.000000    0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+      16.000220    0.000000
B-       1.000000
C        1.000000    1.000000    1.000000
P        2.666400    4.666600    1.000000
Z+       1.000000
Z-       0.026664    0.466660    1.000000
```

Z-+	*0.026664*	*0.466660*	*1.000000*		
F	*104.214448*	*188.734479*	*1.000000*		
F filter	*1.000000*	*1.000000*	*1.000000*		
G	*-1.798009*	*-734.132125*	*0.000000*		
G filter	*0.026664*	*0.493324*	*1.493324*	*1.466660*	*1.000000*
I					
E	*-0.112374*	*-45.882627*			
ED					

Discussion

The upper graph displays three signals: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\bar{y}_m(s) = \frac{Z(s)}{P(s)}\bar{w}(s)$.

The system output is held close to the model output; not by the high gain control, but by the tuning of the emulator. The small ripple is due to the additive sinusoidal sensor noise. The control signal is does not have a large sensor noise component. The performance is not as good as that of the implicit algorithm.

Further investigations

1. Repeat the example but with the tuning switched off. This is achieved by setting"Estimator on" to FALSE. T. As in the previous case, the control signal fluctuations (due to the sinusoidal disturbances) are small, but the response is now poor.

CHAPTER 9

Cascade Control

Aims. To investigate the effect of a neglected inner loop adaptive controller on outer-loop performance when two self-tuning controllers are operated in cascade.

9.1. IMPLEMENTATION DETAILS

The implementation of the self-tuning controllers, and the corresponding simulation, is identical to that described in chapter 6 except that CSTC is configured in a multi-loop form. This is accomplished by storing all variables pertaining to a given loop in the a record data structure called **LoopVAR** of type **TypeLoopVAR.** In cascade mode, the output of one system forms the input to the next; this is implemented (when the Boolean variable Cascade is set to TRUE) in procedure **Simulate** by the statement:

```
uD := LoopVAR[ThisLoop - 1].y;
```

Similarly, in cascade mode, the setpoint of one controller is the control signal from the next controller; this is implemented in procedure **Simulate** by the statement:

```
w := LoopVAR[ThisLoop + 1].u;
```

The Boolean variable Cascade is set to TRUE when **Chapter** is set to 9.

9.2. EXAMPLES

9.2.1. IMPLICIT CASCADE CONTROL

Reference: Section 9.2, page 9-2.

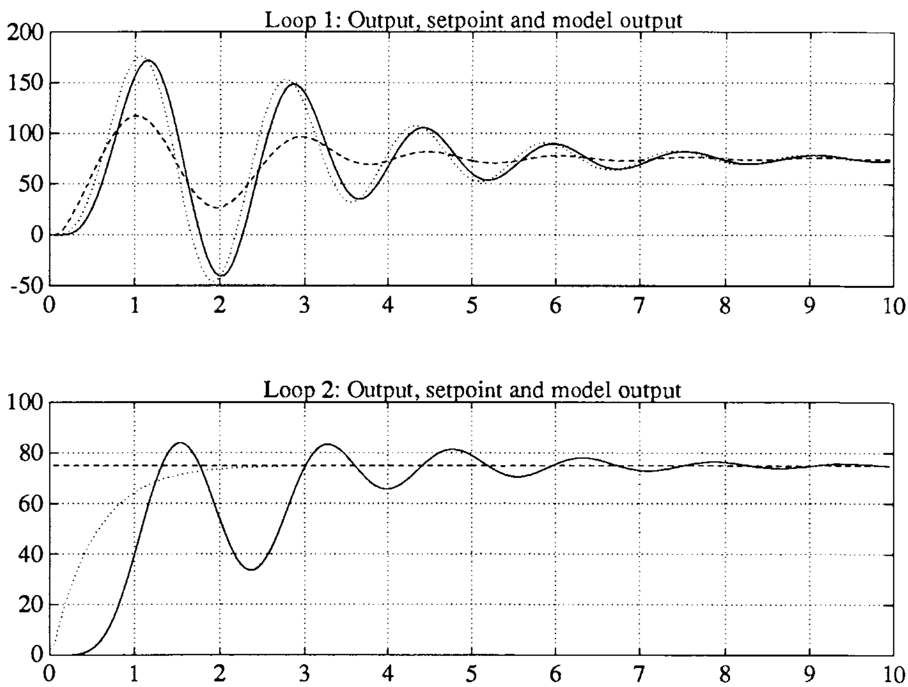

Figure 9.1. Implicit cascade control

Description

Chapter 9 of volume I considers various possible ways of implementing self-tuning cascade control. This example looks at the naive approach of ignoring the inner loop when designing the outer loop.

The example is based on a paper* which also discusses the theoretical properties of this method.

* Gawthrop, P.J. and Kharbouch, M. (1988): "Two-loop self-tuning cascade control", Proc. IEE pt D, Vol. 135, No. 4, pp. 232-238.

The inner loop system, and corresponding design parameters ,are given by:

$$\frac{B_1(s)}{A_1(s)} = \frac{1}{s+1} \tag{9.2.1.1}$$

$$P_1(s) = C_1(s) = 0.1s+1;\ Z_1(s) = 1;\ Q_1(s) = 0.05s \tag{9.2.1.2}$$

The corresponding outer-loop parameters are:

$$\frac{B_2(s)}{A_2(s)} = \frac{1}{s+1} \tag{9.2.1.3}$$

$$P_2(s) = C_2(s) = 0.5s+1;\ Z_2(s) = 1;\ Q_2(s) = 0.4s \tag{9.2.1.4}$$

Notice that the inner loop is chosen to be faster than the outer loop.

Programme interaction

```
runex 9 1
Example 9 of chapter 1: Implicit cascade control

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter                =          9 Example 1 of chapter 9                    :=

 ======== Data Source    ========
 ======== Filters      ========

 ======== LOOP 1        ========
 ======== Control action  ========
Integral action         = TRUE  :=
 ======== Assumed system  ========
A (system denominator)   =    1.000000    0.000000 Denominator: s+1 :=
B (system numerator)     =    1.000000 Numerator: 1 :=
 ======== Emulator design ========
Z-+ (Z- not including B)  =    1.000000  :=
P (model denominator)     =    0.100000    1.000000 (0.1s+1) :=
C (emulator denominator)  =    0.100000    1.000000 (0.1s+1)  :=
-----------------------------------
      System polynomials
```

```
-----------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    0.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000    0.000000
B-         1.000000
C          0.100000    1.000000
P          0.100000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          0.200000    1.000000
F filter   0.100000    1.000000
G          0.010000    0.000000
G filter   0.100000    1.000000
I
E          0.010000
ED         0.000000
-----------------------------------
 ======== STC type        ========
Using lambda filter      = TRUE  :=
 ======== Tuner           ========
Initial Variance         = 100000.000000  :=
Forget time              =  1000.000000  :=
Estimator on             = TRUE  :=
 ======== Controller      ========
Q numerator              =    0.050000    0.000000  :=
Q denominator            =    1.000000  :=
 ======== Simulation      ========
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000    1.000000  :=
B (system numerator)     =    1.000000  :=

 ======== LOOP 2          ========
 ======== Control action  ========
Integral action          = TRUE  :=
 ======== Assumed system  ========
A (system denominator)   =    1.000000    1.000000 Denominator: s+1 :=
B (system numerator)     =    1.000000 Numerator: 1 :=
 ======== Emulator design ========
Z-+ (Z- not including B) =    1.000000  :=
P (model denominator)    =    0.500000    1.000000 (0.5s+1) :=
C (emulator denominator) =    0.500000    1.000000 (0.5s+1)  :=
```

```
------------------------------------
      System polynomials
------------------------------------
A           1.000000    1.000000    0.000000
B           1.000000    0.000000
D           0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+           1.000000    0.000000
B-          1.000000
C           0.500000    1.000000
P           0.500000    1.000000
Z+           1.000000
Z-          1.000000
Z-+          1.000000
------------------------------------
F           0.750000    1.000000
F filter    0.500000    1.000000
G           0.250000    0.000000
G filter    0.500000    1.000000
I
E           0.250000
ED           0.000000
------------------------------------
 ======== STC type        ========
 ======== Tuner          ========
Estimator on            = TRUE  :=
 ======== Controller      ========
Q numerator             =     0.400000    0.000000  :=
Q denominator           =      1.000000  :=
 ======== Simulation      ========
 ======== Setpoint        ========
Step amplitude          =    50.000000  :=
Square amplitude        =    25.000000  :=
Period                  =   20.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =     1.000000    1.000000  :=
B (system numerator)     =    1.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    10.000000

 ======== LOOP 1          ========
```

```
------------------------------------
      System polynomials
------------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   0.000000
D          0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000   0.000000
B-         1.000000
C          0.100000   1.000000
P          0.100000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          0.190241   1.000000
F filter   0.100000   1.000000
G          0.010006   0.000000
G filter   0.100000   1.000000
I
E          0.010000
ED         0.000000
------------------------------------

 ======== LOOP 2        ========
------------------------------------
      System polynomials
------------------------------------
A          1.000000   1.000000   0.000000
B          1.000000   0.000000
D          0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000   0.000000
B-         1.000000
C          0.500000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          0.849471   1.000000
F filter   0.500000   1.000000
G          0.314538   0.000000
G filter   0.500000   1.000000
I
```

E *0.250000*
ED *0.000000*

Discussion

The upper graph shows the system output, setpoint, and model output for the inner loop; the lower graph shows the corresponding signals for the outer loop. The setpoint for the inner loop is the control signal generated by the outer loop.

As designed, the inner loop gives close tracking of the setpoint. The outer-loop response displays overshoot due to the neglected inner-loop dynamics; it is, however, stable. The algorithm treated in the cited paper does not use the $\Lambda(s)$ filter, so the theory is not strictly applicable, but nevertheless, does give the correct prediction in this case.

Further investigations

1. Following the cited paper, try $Q_2(s) = 0.15s$ (set Q numerator to 0.15 in loop 2). This is not predicted to give stability.
2. Try the algorithm given in the paper by setting:

$$Z^{-+}(s) = Q_d(s) = 0.01s+1 \tag{9.2.1.5}$$

in loop 1 and

$$Z^{-+}(s) = Q_d(s) = 0.05s+1 \tag{9.2.1.6}$$

in loop 2. For each loop set "Using lambda filter" to FALSE, and you will need to set the sample interval to 0.02. How does the performance compare?

9.2.2. EXPLICIT CASCADE CONTROL

Reference: Section 9.2, page 9-2.

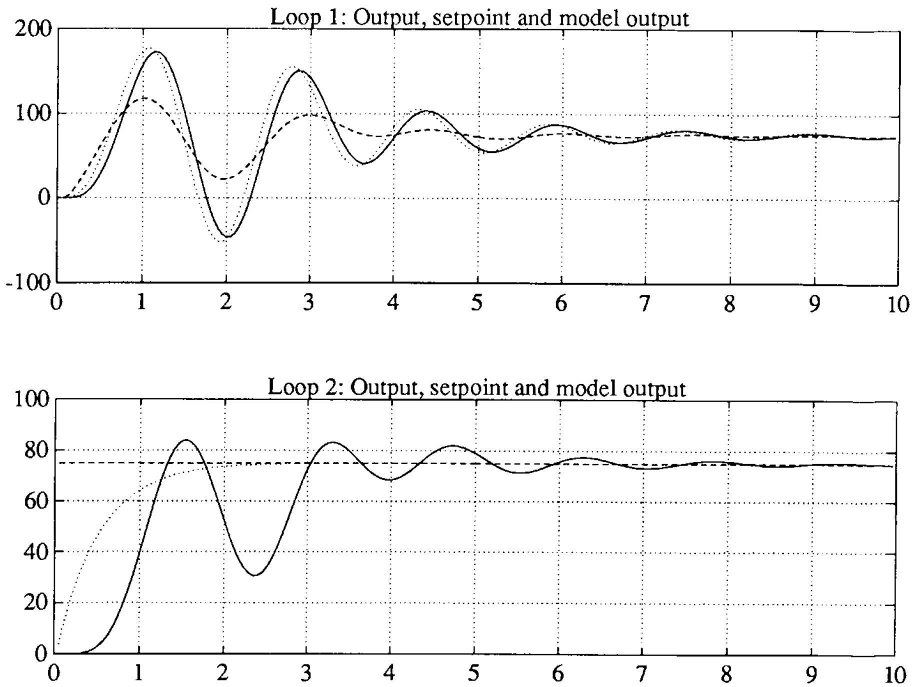

Figure 9.2. Explicit cascade control

Description

This example is identical to the previous one except that an explicit algorithm is used.

Programme interaction

runex 9 2
Example 9 of chapter 2: Explicit cascade control

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?
Chapter = 9 Example 1 of chapter 9 :=

```
======== Data Source    ========
======== Filters        ========

======== LOOP 1         ========
======== Control action ========
Integral action          = TRUE  :=
======== Assumed system ========
A (system denominator)    =    1.000000    0.000000 Denominator: s+1 :=
B (system numerator)      =    1.000000 Numerator: 1 :=
======== Emulator design ========
Z-+ (Z- not including B)  =    1.000000  :=
P (model denominator)     =    0.100000    1.000000 (0.1s+1) :=
C (emulator denominator)  =    0.100000    1.000000 (0.1s+1)  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000     0.000000     0.000000
B          1.000000     0.000000
D          0.000000     0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000     0.000000
B-         1.000000
C          0.100000     1.000000
P          0.100000     1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          0.200000     1.000000
F filter   0.100000     1.000000
G          0.010000     0.000000
G filter   0.100000     1.000000
I
E          0.010000
ED         0.000000
-----------------------------------
======== STC type       ========
Explicit self-tuning     = TRUE  :=
======== Identification ========
Initial Variance         = 100000.000000  :=
Forget time              = 1000.000000  :=
Estimator on             = TRUE  :=
======== Controller     ========
Q numerator              =    0.050000    0.000000  :=
Q denominator            =    1.000000  :=
======== Simulation     ========
======== In Disturbance ========
```

```
======== Out Disturbance ========
======== Actual system   ========
A (system denominator)    =    1.000000    1.000000  :=
B (system numerator)      =    1.000000  :=

======== LOOP 2          ========
======== Control action  ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)    =    1.000000    1.000000 Denominator: s+1 :=
B (system numerator)      =    1.000000 Numerator: 1  :=
======== Emulator design ========
Z-+ (Z- not including B)  =    1.000000  :=
P (model denominator)     =    0.500000    1.000000 (0.5s+1) :=
C (emulator denominator)  =    0.500000    1.000000 (0.5s+1)  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000    1.000000    0.000000
B          1.000000    0.000000
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000    0.000000
B-         1.000000
C          0.500000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          0.750000    1.000000
F filter   0.500000    1.000000
G          0.250000    0.000000
G filter   0.500000    1.000000
I
E          0.250000
ED         0.000000
------------------------------------
======== STC type        ========
Explicit self-tuning     = TRUE  :=
======== Identification ========
Initial Variance         = 100000.000000  :=
Forget time              =  1000.000000  :=
Estimator on             = TRUE  :=
======== Controller      ========
Q numerator              =    0.400000    0.000000  :=
Q denominator            =    1.000000  :=
```

```
======== Simulation      ========
======== Setpoint        ========
Step amplitude          =   50.000000  :=
Square amplitude         =   25.000000  :=
Period                 =   20.000000  :=
======== In Disturbance  ========
======== Out Disturbance ========
======== Actual system   ========
A (system denominator)   =    1.000000   1.000000  :=
B (system numerator)     =    1.000000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    10.000000

======== LOOP 1          ========
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.998652     0.000000
B          0.999791    0.000000
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          0.999791    0.000000
B-         1.000000
C          0.100000    1.000000
P          0.100000    1.000000
Z+          1.000000
Z-         1.000000
Z-+         1.000000
------------------------------------
F          0.190013    1.000000
F filter   0.100000    1.000000
G          0.009998    0.000000
G filter   0.100000    1.000000
I
E          0.010000
ED          0.000000
------------------------------------

======== LOOP 2          ========
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.922202     0.000000
```

B	*1.088899*	*0.000000*
D	*0.000000*	*0.000000*

Design polynomials

B+	*1.088899*	*0.000000*
B-	*1.000000*	
C	*0.500000*	*1.000000*
P	*0.500000*	*1.000000*
Z+	*1.000000*	
Z-	*1.000000*	
Z-+	*1.000000*	

F	*0.769449*	*1.000000*
F filter	*0.500000*	*1.000000*
G	*0.272225*	*0.000000*
G filter	*0.500000*	*1.000000*
I		
E	*0.250000*	
ED	*0.000000*	

Discussion

The upper graph shows the system output, setpoint, and model output for the inner loop; the lower graph shows the corresponding signals for the outer loop. The setpoint for the inner loop is the control signal generated by the outer loop.

As designed, the inner loop gives close tracking of the setpoint. The outer-loop response displays overshoot due to the neglected inner-loop dynamics; it is, however, stable. The algorithm treated in the cited paper does not use explicit estimation, so the theory is not strictly applicable, but nevertheless, does give the correct prediction in this case.

Note that the inner loop estimated parameters are correct, but that the outer-loop estimated parameters are not. Why is this?

Further investigations

1. Following the cited paper, try $Q_2(s) = 0.15s$ (set Q numerator to 0.15 in loop 2). This is not predicted to give stability. What do you observe?

CHAPTER 10

Two-Input Two-Output Systems

Aims. To investigate the behaviour of self-tuning controllers operating in a multi-loop environment. To compare the performance when coupling is ignored and included in the algorithm.

10.1. IMPLEMENTATION DETAILS

As with the implementation discussed in Chapter 9 (section 9.1), this chapter requires a multi-loop simulation. Once again, this is handled using the record data structure **LoopVAR** of type **TypeLoopVAR.**

The difference between the impementation of Chapter 9 and this chapter is that the Boolean variable **Cascade** is set to FALSE. In this case, the interaction signals are set equal the systems output if the Boolean variable **OutputCoupled** is TRUE; otherwise they are set equal to the system inputs. This occurs within procedure **Run** using the statements

```
FOR Loop := 1 TO Loops DO
  IF OutputCoupled THEN
    LoopInteraction[Loop] := LoopVAR[Loop].y
  ELSE
    LoopInteraction[Loop] := LoopVAR[Loop].u;
```

The self-tuning algorithms are extended to incorporate these additional interaction signals. In particular, procedures: **Emulator, SetData** and **TuneEmulator** all include loops involving the variable

NumberInteractions.

10.2. EXAMPLES

10.2.1. OUTPUT-COUPLED TANKS (IMPLICIT)

Reference: Section 10.2; page 10-7.

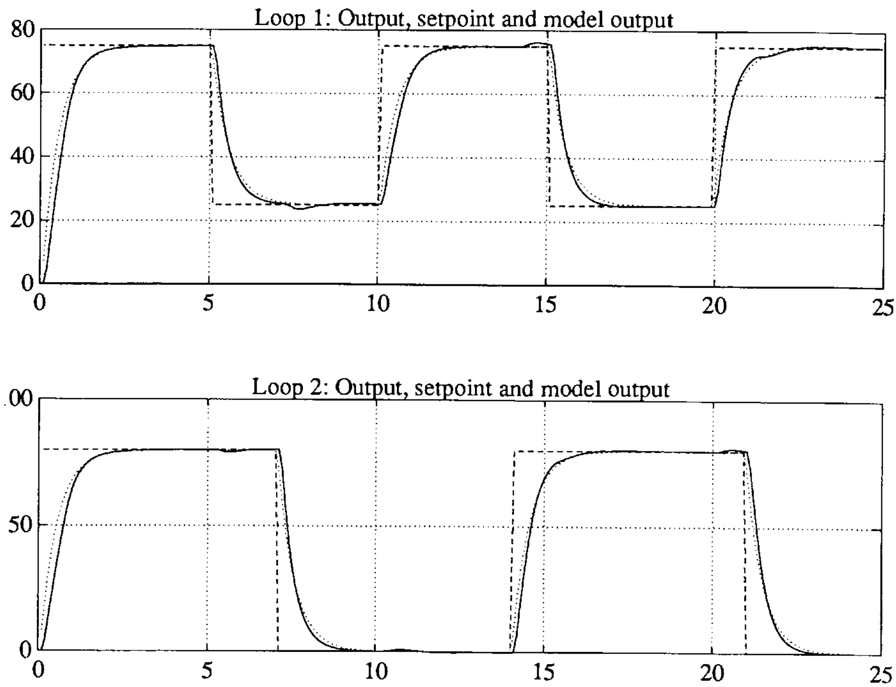

Figure 10.1. Output-coupled tanks (Implicit)

Description

Example 1 of chapter I-10 considers a simple model comprising two connected tanks of liquid ; the effective interaction is thus from output to input. This example is a special case where the outflow constants (k_1 and k_2) are both 0.5, giving two identical sets of transfer functions:

$$R_{11}(s) = R_{22}(s) = \frac{1}{s+1} \tag{10.2.1.1}$$

$$R_{12}(s) = R_{21}(s) = 0.5 \tag{10.2.1.2}$$

That is:

$$A_1(s) = A_2(s) = s+1 \tag{10.2.1.3}$$

$$B_1(s) = B_2(s) = 1 \tag{10.2.1.4}$$

$$B_{12}(s) = B_{21}(s) = 0.5 \tag{10.2.1.5}$$

In this example, the initial parameters for $A_1(s)$, $A_2(s)$, $B_1(s)$, $B_2(s)$ are correct, but $B_{12}(s)$ and $B_{21}(s)$ are set to zero. An implicit self-tuning algorithm with $\Lambda = 1/P(s)$ is used.

Programme interaction

```
runex 10 1
Example 10 of chapter 1:  Output-coupled tanks (Implicit)

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters        ========
Sample Interval          =    0.100000  :=

 ======== LOOP 1          ========
 ======== Control action  ========
Integral action          = TRUE  :=
 ======== Assumed system  ========
A (system denominator)   =    1.000000    0.000000 Denominator: s :=
B (system numerator)     =    1.000000 Numerator: 1 :=
Number of interactions   =           1 Account for the other loop          :=
Interaction polynomial   =    0.000000  :=
 ======== Emulator design ========
P (model denominator)    =    0.500000    1.000000 (0.5s+1) :=
C (emulator denominator) =    1.000000    1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    0.000000
B[1]        0.000000    0.000000
```

```
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000    0.000000
B-         1.000000
C          1.000000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          1.500000    1.000000
F filter   1.000000    1.000000
G          0.500000    0.000000
G filter   1.000000    1.000000
G[1]       0.000000    0.000000
I
E          0.500000
ED         0.000000
------------------------------------
 ======== STC type       ========
 ======== Tuner          ========
Estimator on          = TRUE  :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
Step amplitude          =   50.000000  :=
Square amplitude        =   25.000000  :=
Period                  =   10.000000  :=
 ======== In Disturbance ========
 ======== Out Disturbance ========
 ======== Actual system  ========
A (system denominator)   =    1.000000    1.000000  :=
B (system numerator)     =    1.000000  :=
Number of interactions   =          1 Account for the other loop            :=
Interaction polynomial   =    0.500000 k=0.5 :=

 ======== LOOP 2         ========
 ======== Control action ========
Integral action       = TRUE  :=
 ======== Assumed system ========
A (system denominator)   =    1.000000    0.000000 Denominator: s :=
B (system numerator)     =    1.000000 Numerator: 1 :=
Interaction polynomial   =    0.000000  :=
 ======== Emulator design ========
P (model denominator)    =    0.500000    1.000000 (0.5s+1) :=
C (emulator denominator) =    1.000000    1.000000  :=
------------------------------------
```

```
        System polynomials
------------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   0.000000
B[1]       0.000000   0.000000
D          0.000000   0.000000
------------------------------------
        Design polynomials
------------------------------------
B+         1.000000   0.000000
B-         1.000000
C          1.000000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          1.500000   1.000000
F filter   1.000000   1.000000
G          0.500000   0.000000
G filter   1.000000   1.000000
G[1]       0.000000   0.000000
I
E          0.500000
ED         0.000000
------------------------------------
 ======== STC type        ========
 ======== Tuner           ========
Estimator on          = TRUE  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
Step amplitude          =   40.000000  :=
Square amplitude        =   40.000000  :=
Period                  =   14.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000   1.000000  :=
B (system numerator)     =   1.000000  :=
Number of interactions   =        1 Account for the other loop          :=
Interaction polynomial   =   0.500000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is   25.000000
```

```
======== LOOP 1        ========
-----------------------------------
      System polynomials
-----------------------------------
A         1.000000   0.000000   0.000000
B         1.000000   0.000000
B[1]      0.000000   0.000000
D         0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+        1.000000   0.000000
B-        1.000000
C         1.000000   1.000000
P         0.500000   1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
-----------------------------------
F         1.000787   1.000000
F filter  1.000000   1.000000
G         0.499590   0.000000
G filter  1.000000   1.000000
G[1]      0.249854   0.000000
I
E         0.500000
ED        0.000000
-----------------------------------

======== LOOP 2        ========
-----------------------------------
      System polynomials
-----------------------------------
A         1.000000   0.000000   0.000000
B         1.000000   0.000000
B[1]      0.000000   0.000000
D         0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+        1.000000   0.000000
B-        1.000000
C         1.000000   1.000000
P         0.500000   1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
-----------------------------------
F         1.000788   1.000000
```

F filter	*1.000000*	*1.000000*
G	*0.499571*	*0.000000*
G filter	*1.000000*	*1.000000*
G[1]	*0.249878*	*0.000000*
I		
E	*0.500000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals associated with the first loop: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$; the lower graph displays the corresponding signals for the second loop.

The interaction is essentially eliminated by the self-tuning decoupling terms. Note that the G[1] parameter is correctly estimated to be 0.25 in each loop.

Further investigations

1. Observe the effect of the tuning by setting "Estimation on" to FALSE in each loop. Note that the $A(s)$ and $B(s)$ parameters are correct, but that the interaction terms are (incorrectly) set to zero.
2. Repeat for different values of a and k; observe that the interaction is always eliminated when tuning is enabled.

10.2.2. OUTPUT-COUPLED TANKS IGNORING INTERACTION

Reference: Section 10.2; page 10-7.

Description

This example is identical to example 10.2.1, except that the interaction terms are ignored in the emulators for each loop.

Programme interaction

runex 10 2
Example 10 of chapter 2: Output-coupled tanks ignoring interaction

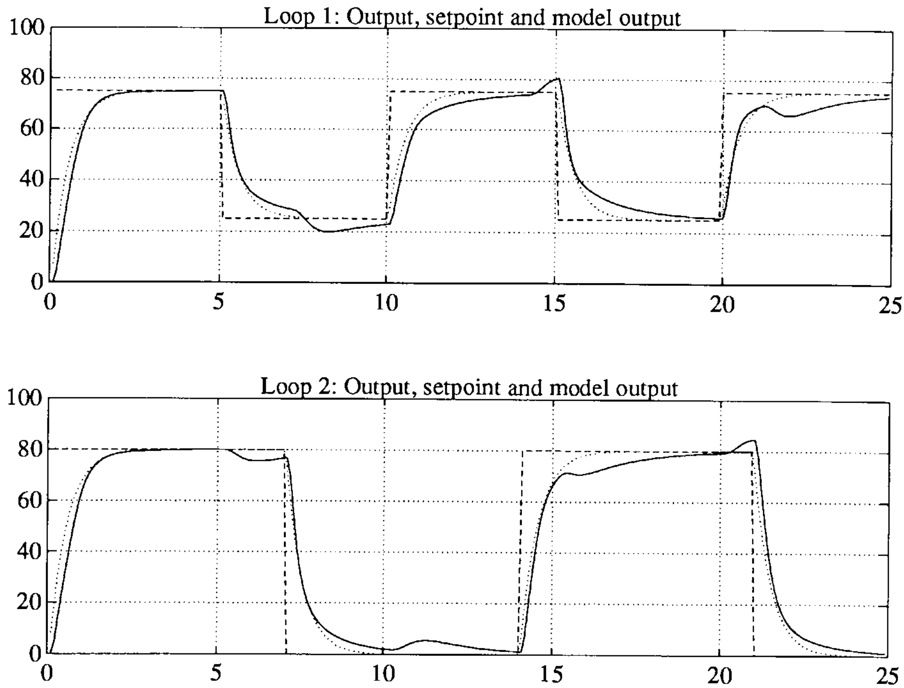

Figure 10.2. Output-coupled tanks ignoring interaction

```
======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source     ========
======== Filters         ========
Sample Interval          =   0.100000  :=

======== LOOP 1          ========
======== Control action  ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)   =   1.000000   0.000000 Denominator: s :=
B (system numerator)     =   1.000000 Numerator: 1 :=
Number of interactions   =          0 Ignore the other loop        :=
======== Emulator design ========
P (model denominator)    =   0.500000   1.000000 (0.5s+1) :=
```

```
C (emulator denominator)  =   1.000000   1.000000  :=
-----------------------------------
     System polynomials
-----------------------------------
A         1.000000   0.000000   0.000000
B         1.000000   0.000000
D         0.000000   0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+        1.000000   0.000000
B-        1.000000
C         1.000000   1.000000
P         0.500000   1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
-----------------------------------
F         1.500000   1.000000
F filter  1.000000   1.000000
G         0.500000   0.000000
G filter  1.000000   1.000000
I
E         0.500000
ED        0.000000
-----------------------------------
 ======== STC type        ========
 ======== Tuner           ========
Estimator on             = TRUE  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
Step amplitude           =   50.000000  :=
Square amplitude         =   25.000000  :=
Period                   =   10.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000   1.000000  :=
B (system numerator)     =   1.000000  :=
Number of interactions   =          1 Account for the other loop       :=
Interaction polynomial   =   0.500000 k=0.5 :=

 ======== LOOP 2          ========
 ======== Control action  ========
Integral action          = TRUE  :=
 ======== Assumed system  ========
A (system denominator)   =   1.000000   0.000000 Denominator: s :=
B (system numerator)     =   1.000000 Numerator: 1 :=
```

```
Number of interactions    =         0 Ignore the other loop            :=
 ======== Emulator design ========
P (model denominator)     =    0.500000   1.000000 (0.5s+1) :=
C (emulator denominator)  =    1.000000   1.000000  :=
-----------------------------------
     System polynomials
-----------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   0.000000
D          0.000000   0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+         1.000000   0.000000
B-         1.000000
C          1.000000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          1.500000   1.000000
F filter   1.000000   1.000000
G          0.500000   0.000000
G filter   1.000000   1.000000
I
E          0.500000
ED         0.000000
-----------------------------------
 ======== STC type      ========
 ======== Tuner         ========
Estimator on           = TRUE  :=
 ======== Controller    ========
 ======== Simulation    ========
 ======== Setpoint      ========
Step amplitude          =  40.000000  :=
Square amplitude        =  40.000000  :=
Period                  =  14.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =   1.000000   1.000000  :=
B (system numerator)     =   1.000000  :=
Number of interactions   =        1 Account for the other loop            :=
Interaction polynomial   =   0.500000  :=
Simulation running:
     25% complete
     50% complete
     75% complete
```

```
    100% complete
Time now is    25.000000

 ======== LOOP 1         ========
-----------------------------------
     System polynomials
-----------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   0.000000
D          0.000000   0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+         1.000000   0.000000
B-         1.000000
C          1.000000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          1.195862   1.000000
F filter   1.000000   1.000000
G          0.343553   0.000000
G filter   1.000000   1.000000
I
E          0.500000
ED         0.000000
-----------------------------------

 ======== LOOP 2         ========
-----------------------------------
     System polynomials
-----------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   0.000000
D          0.000000   0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+         1.000000   0.000000
B-         1.000000
C          1.000000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          1.096956   1.000000
```

```
F filter    1.000000    1.000000
G           0.437673    0.000000
G filter    1.000000    1.000000
I
E           0.500000
ED           0.000000
------------------------------------
```

Discussion

The upper graph displays three signals associated with the first loop: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$; the lower graph displays the corresponding signals for the second loop.

The interaction cannot any longer be eliminated by the self-tuning controllers as the corresponding terms do not appear in the emulators. Note that the estimated parameters are incorrect due to the neglected interaction.

Further investigations

1. Repeat for different values of a and k; determine the maximum value of k for which the response remains satisfactory.

10.2.3. OUTPUT-COUPLED TANKS (EXPLICIT&IMPLICIT)

Reference: Section 10.2; page 10-7.

Description

This example is identical to example 10.2.1 except that an *explicit* self-tuning algorithm is used in loop 1 and an *implicit* algorithm in loop 2.

Programme interaction

runex 10 3
Example 10 of chapter 3: Output-coupled tanks (Explicit&implicit)

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

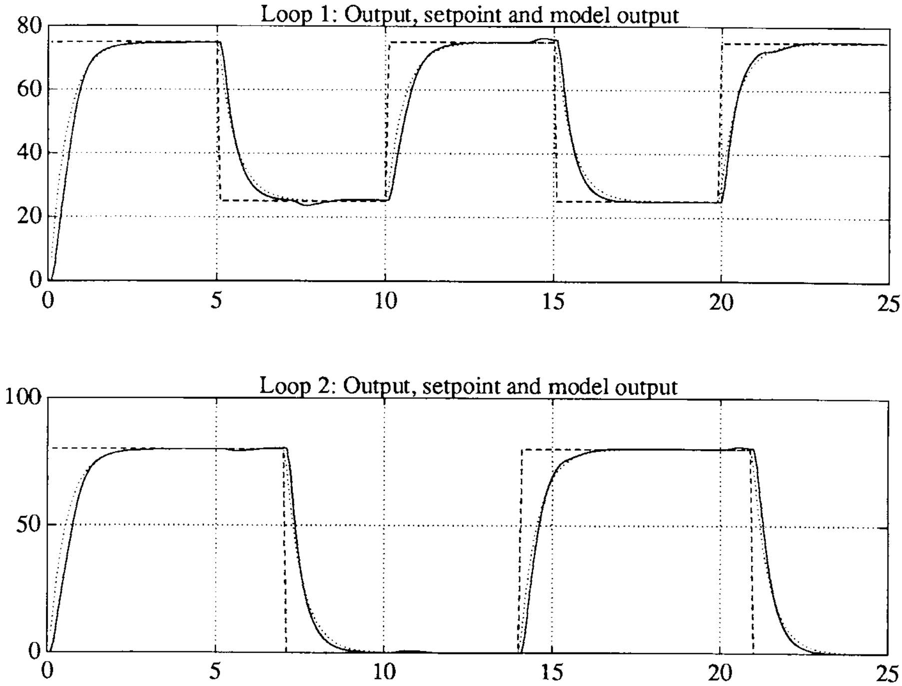

Figure 10.3. Output-coupled tanks (Explicit&implicit)

```
======== Data Source     ========
======== Filters         ========
Sample Interval          =    0.100000  :=

======== LOOP 1          ========
======== Control action  ========
Integral action          = TRUE  :=
======== Assumed system  ========
A (system denominator)   =    1.000000    0.000000 Denominator: 1 :=
B (system numerator)     =    1.000000 Numerator: 1 :=
Number of interactions   =           1 Account for the other loop          :=
Interaction polynomial   =    0.000000  :=
======== Emulator design ========
P (model denominator)    =    0.500000    1.000000 (0.5s+1) :=
C (emulator denominator) =    1.000000    1.000000  :=
----------------------------------
        System polynomials
```

```
------------------------------------
A          1.000000   0.000000   0.000000
B          1.000000   0.000000
B[1]       0.000000   0.000000
D          0.000000   0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000   0.000000
B-         1.000000
C          1.000000   1.000000
P          0.500000   1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          1.500000   1.000000
F filter   1.000000   1.000000
G          0.500000   0.000000
G filter   1.000000   1.000000
G[1]       0.000000   0.000000
I
E          0.500000
ED         0.000000
------------------------------------
 ======== STC type       ========
 ======== Identification ========
Estimator on              = TRUE  :=
Cs (emulator denominator) =  1.000000   0.000000   0.000000  :=
Identifying rational part = TRUE  :=
Identifying delay         = FALSE  :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
Step amplitude            =   50.000000  :=
Square amplitude          =   25.000000  :=
Period                    =   10.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)    =   1.000000   1.000000  :=
B (system numerator)      =   1.000000  :=
Number of interactions    =          1 Account for the other loop          :=
Interaction polynomial    =   0.500000 k=0.5 :=

 ======== LOOP 2          ========
 ======== Control action  ========
Integral action           = TRUE  :=
 ======== Assumed system  ========
```

```
A (system denominator)    =    1.000000    0.000000 Denominator: 1 :=
B (system numerator)      =    1.000000 Numerator: 1 :=
Interaction polynomial    =    0.000000  :=
 ======== Emulator design ========
P (model denominator)     =    0.500000    1.000000 (0.5s+1) :=
C (emulator denominator)  =    1.000000    1.000000  :=
-------------------------------------
      System polynomials
-------------------------------------
A         1.000000    0.000000    0.000000
B         1.000000    0.000000
B[1]      0.000000    0.000000
D         0.000000    0.000000
-------------------------------------
      Design polynomials
-------------------------------------
B+        1.000000    0.000000
B-        1.000000
C         1.000000    1.000000
P         0.500000    1.000000
Z+        1.000000
Z-        1.000000
Z-+       1.000000
-------------------------------------
F         1.500000    1.000000
F filter  1.000000    1.000000
G         0.500000    0.000000
G filter  1.000000    1.000000
G[1]      0.000000    0.000000
I
E         0.500000
ED        0.000000
-------------------------------------
 ======== STC type        ========
 ======== Tuner           ========
Estimator on          = TRUE  :=
 ======== Controller      ========
 ======== Simulation      ========
 ======== Setpoint        ========
Step amplitude        =    40.000000  :=
Square amplitude      =    40.000000  :=
Period                =    14.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)    =    1.000000    1.000000  :=
B (system numerator)      =    1.000000  :=
Number of interactions    =          1 Account for the other loop          :=
Interaction polynomial    =    0.500000  :=
```

```
Simulation running:
     25% complete
     50% complete
     75% complete
    100% complete
Time now is    25.000000

 ======== LOOP 1        ========
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    1.000105    0.000000
B          1.000048    0.000000
B[1]        0.500084    0.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          1.000048    0.000000
B-         1.000000
C          1.000000    1.000000
P          0.500000    1.000000
Z+          1.000000
Z-         1.000000
Z-+         1.000000
-----------------------------------
F          0.999948    1.000000
F filter   1.000000    1.000000
G          0.500024    0.000000
G filter   1.000000    1.000000
G[1]        0.250042    0.000000
I
E          0.500000
ED          0.000000
-----------------------------------

 ======== LOOP 2        ========
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    0.000000
B[1]        0.000000    0.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+          1.000000    0.000000
B-         1.000000
```

C	*1.000000*	*1.000000*
P	*0.500000*	*1.000000*
Z+	*1.000000*	
Z-	*1.000000*	
Z-+	*1.000000*	

F	*1.000788*	*1.000000*
F filter	*1.000000*	*1.000000*
G	*0.499571*	*0.000000*
G filter	*1.000000*	*1.000000*
G[1]	*0.249878*	*0.000000*
I		
E	*0.500000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals associated with the first loop: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$; the lower graph displays the corresponding signals for the second loop.

The interaction is essentially eliminated by the self-tuning decoupling terms.

Further investigations

1. Observe the effect of the tuning by setting "Estimation on" to FALSE in each loop. Note that the $A(s)$ and $B(s)$ parameters are correct, but that the interaction terms are (incorrectly) set to zero.
2. Repeat for different values of a and k; observe that the interaction is always eliminated when tuning is enabled.

10.2.4. INPUT-COUPLED TANKS (IMPLICIT)

Reference: Section 10.2; page 10-7.

Description

Example 2 of chapter I-10 (page I-10-8) considers a simple model comprising two tanks of liquid which share a common inflow; the effective interaction is thus from input to input. Unlike the example in the book, however, the input, not the output, is used as a feedforward signal so it is possible to decouple the system exactly. In practice, discrepancies arise due to the non-zero sample

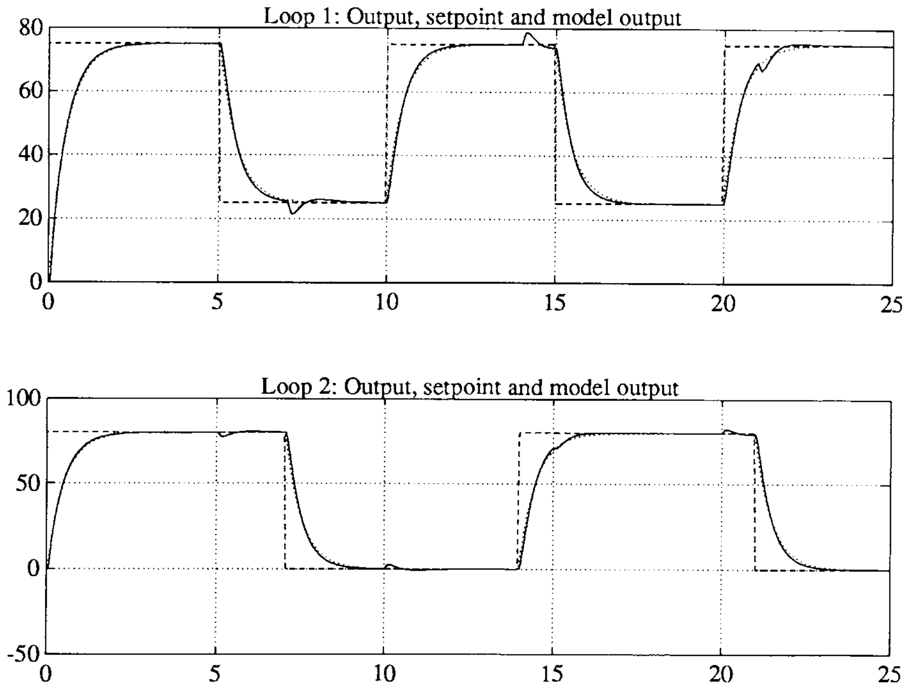

Figure 10.4. Input-coupled tanks (Implicit)

interval.

Programme interaction

runex 10 4
Example 10 of chapter 4: Input-coupled tanks (Implicit)

======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

======== Data Source ========
======== Filters ========
Sample Interval = 0.050000 :=

```
======== LOOP 1          ========
======== Control action  ========
Integral action           = TRUE  :=
======== Assumed system  ========
A (system denominator)    =   1.000000   0.000000 Denominator: s :=
B (system numerator)      =   1.000000 Numerator: 1 :=
Number of interactions    =          1 Account for the other loop          :=
Interaction polynomial    =   0.000000  :=
======== Emulator design ========
P (model denominator)     =   0.500000   1.000000 (0.5s+1) :=
C (emulator denominator)  =   1.000000   1.000000  :=
-----------------------------------
      System polynomials
-----------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    0.000000
B[1]       0.000000    0.000000
D          0.000000    0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+         1.000000    0.000000
B-         1.000000
C          1.000000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
-----------------------------------
F          1.500000    1.000000
F filter   1.000000    1.000000
G          0.500000    0.000000
G filter   1.000000    1.000000
G[1]       0.000000    0.000000
I
E          0.500000
ED         0.000000
-----------------------------------
======== STC type        ========
======== Tuner           ========
Estimator on              = TRUE  :=
======== Controller      ========
======== Simulation      ========
======== Setpoint        ========
Step amplitude            =   50.000000  :=
Square amplitude          =   25.000000  :=
Period                    =   10.000000  :=
======== In Disturbance  ========
======== Out Disturbance ========
```

```
======== Actual system ========
A (system denominator)  =  1.000000  1.000000 :=
B (system numerator)    =  1.000000 :=
Number of interactions  =  1 Account for the other loop   :=
Interaction polynomial  =  0.500000 k=0.5 :=

======== LOOP 2       ========
======== Control action ========
Integral action        = TRUE :=
======== Assumed system ========
A (system denominator)  =  1.000000  0.000000 Denominator: s :=
B (system numerator)    =  1.000000 Numerator: 1 :=
Interaction polynomial  =  0.000000 :=
======== Emulator design ========
P (model denominator)   =  0.500000  1.000000 (0.5s+1) :=
C (emulator denominator) = 1.000000  1.000000 :=
-----------------------------------
      System polynomials
-----------------------------------
A        1.000000   0.000000   0.000000
B        1.000000   0.000000
B[1]     0.000000   0.000000
D        0.000000   0.000000
-----------------------------------
      Design polynomials
-----------------------------------
B+       1.000000   0.000000
B-       1.000000
C        1.000000   1.000000
P        0.500000   1.000000
Z+       1.000000
Z-       1.000000
Z-+      1.000000
-----------------------------------
F        1.500000   1.000000
F filter 1.000000   1.000000
G        0.500000   0.000000
G filter 1.000000   1.000000
G[1]     0.000000   0.000000
I
E        0.500000
ED       0.000000
-----------------------------------
======== STC type    ========
======== Tuner       ========
Estimator on         = TRUE :=
======== Controller  ========
======== Simulation  ========
======== Setpoint    ========
```

```
Step amplitude          =   40.000000 :=
Square amplitude        =   40.000000 :=
Period                  =   14.000000 :=
======== In Disturbance  ========
======== Out Disturbance ========
======== Actual system   ========
A (system denominator)  =   1.000000   1.000000 :=
B (system numerator)    =   1.000000 :=
Number of interactions  =          1 Account for the other loop          :=
Interaction polynomial  =   0.500000 :=
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is   25.000000

======== LOOP 1         ========
-----------------------------------
     System polynomials
-----------------------------------
A        1.000000   0.000000   0.000000
B        1.000000   0.000000
B[1]     0.000000   0.000000
D        0.000000   0.000000
-----------------------------------
     Design polynomials
-----------------------------------
B+       1.000000   0.000000
B-       1.000000
C        1.000000   1.000000
P        0.500000   1.000000
Z+       1.000000
Z-       1.000000
Z-+      1.000000
-----------------------------------
F        1.000198   1.000000
F filter 1.000000   1.000000
G        0.499897   0.000000
G filter 1.000000   1.000000
G[1]     0.249952   0.000000
I
E        0.500000
ED       0.000000
-----------------------------------

======== LOOP 2         ========
-----------------------------------
     System polynomials
```

A	*1.000000*	*0.000000*	*0.000000*
B	*1.000000*	*0.000000*	
B[1]	*0.000000*	*0.000000*	
D	*0.000000*	*0.000000*	

Design polynomials

B+	*1.000000*	*0.000000*
B-	*1.000000*	
C	*1.000000*	*1.000000*
P	*0.500000*	*1.000000*
Z+	*1.000000*	
Z-	*1.000000*	
Z-+	*1.000000*	

F	*1.000200*	*1.000000*
F filter	*1.000000*	*1.000000*
G	*0.499899*	*0.000000*
G filter	*1.000000*	*1.000000*
G[1]	*0.249961*	*0.000000*
I		
E	*0.500000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals associated with the first loop: the system output $y(t)$, the setpoint $w(t)$ and the ideal model output $y_m(t)$; the lower graph displays the corresponding signals for the second loop.

The interaction is essentially eliminated by the self-tuning decoupling terms; and the estimated parameters are correct.

Further investigations

1. Observe the effect of the tuning by setting "Estimation on" to FALSE in each loop. Note that the $A(s)$ and $B(s)$ parameters are correct, but that the interaction terms are (incorrectly) set to zero.
2. Repeat for different values of a and k; observe that the interaction is always eliminated when

tuning is enabled.

10.2.5. INPUT-COUPLED TANKS IGNORING INTERACTION

Reference: Section 10.2; page 10-7.

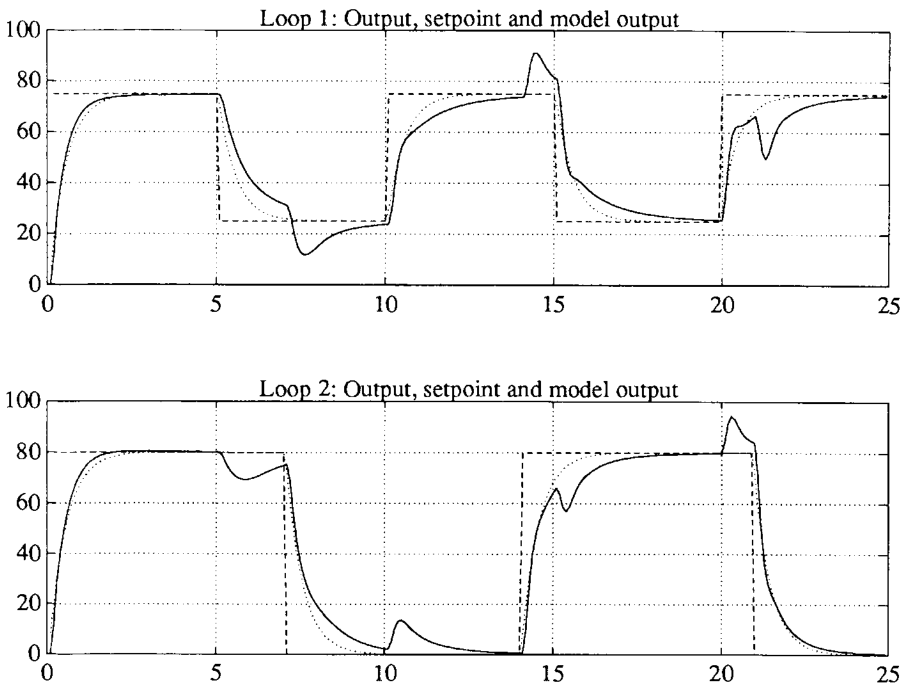

Figure 10.5. Input-coupled tanks ignoring interaction

Description

This example is identical to example 10.2.4, except that the interaction terms are ignored in the emulators for each loop.

Programme interaction

```
runex 10 5
Example 10 of chapter 5: Input-coupled tanks ignoring interaction

 ======== C S T C Version 6.0 ========

Enter all variables (y/n, default n)?

 ======== Data Source     ========
 ======== Filters         ========
Sample Interval          =    0.100000  :=

 ======== LOOP 1          ========
 ======== Control action  ========
Integral action          = TRUE  :=
 ======== Assumed system  ========
A (system denominator)   =    1.000000   0.000000 Denominator: s :=
B (system numerator)     =    1.000000 Numerator: 1 :=
Number of interactions   =           0 Ignore the other loop         :=
 ======== Emulator design ========
P (model denominator)    =    0.500000   1.000000 (0.5s+1) :=
C (emulator denominator) =    1.000000   1.000000  :=
-------------------------------------
      System polynomials
-------------------------------------
A           1.000000    0.000000    0.000000
B           1.000000    0.000000
D           0.000000    0.000000
-------------------------------------
      Design polynomials
-------------------------------------
B+          1.000000    0.000000
B-          1.000000
C           1.000000    1.000000
P           0.500000    1.000000
Z+          1.000000
Z-          1.000000
Z-+         1.000000
-------------------------------------
F           1.500000    1.000000
F filter    1.000000    1.000000
G           0.500000    0.000000
G filter    1.000000    1.000000
I
E           0.500000
ED          0.000000
```

```
------------------------------------
 ======== STC type        ========
 ======== Tuner          ========
Estimator on            = TRUE  :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
Step amplitude          =   50.000000  :=
Square amplitude        =   25.000000  :=
Period                  =   10.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000    1.000000  :=
B (system numerator)     =    1.000000  :=
Number of interactions   =          1 Account for the other loop          :=
Interaction polynomial   =    0.500000 k=0.5 :=

 ======== LOOP 2          ========
 ======== Control action  ========
Integral action         = TRUE  :=
 ======== Assumed system  ========
A (system denominator)   =    1.000000    0.000000 Denominator: s :=
B (system numerator)     =    1.000000 Numerator: 1 :=
Number of interactions   =          0 Ignore the other loop          :=
 ======== Emulator design ========
P (model denominator)    =    0.500000    1.000000 (0.5s+1) :=
C (emulator denominator) =    1.000000    1.000000  :=
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    0.000000
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+          1.000000    0.000000
B-         1.000000
C          1.000000    1.000000
P          0.500000    1.000000
Z+          1.000000
Z-         1.000000
Z-+         1.000000
------------------------------------
F          1.500000    1.000000
F filter   1.000000    1.000000
G          0.500000    0.000000
G filter   1.000000    1.000000
```

```
I
E          0.500000
ED         0.000000
------------------------------------
 ======== STC type       ========
 ======== Tuner          ========
Estimator on            = TRUE  :=
 ======== Controller     ========
 ======== Simulation     ========
 ======== Setpoint       ========
Step amplitude          =   40.000000  :=
Square amplitude        =   40.000000  :=
Period                  =   14.000000  :=
 ======== In Disturbance  ========
 ======== Out Disturbance ========
 ======== Actual system   ========
A (system denominator)   =    1.000000   1.000000  :=
B (system numerator)     =    1.000000  :=
Number of interactions   =         1 Account for the other loop          :=
Interaction polynomial   =    0.500000  :=
Simulation running:
    25% complete
    50% complete
    75% complete
   100% complete
Time now is   25.000000

 ======== LOOP 1          ========
------------------------------------
      System polynomials
------------------------------------
A          1.000000    0.000000    0.000000
B          1.000000    0.000000
D          0.000000    0.000000
------------------------------------
      Design polynomials
------------------------------------
B+         1.000000    0.000000
B-         1.000000
C          1.000000    1.000000
P          0.500000    1.000000
Z+         1.000000
Z-         1.000000
Z-+        1.000000
------------------------------------
F          1.245325    1.000000
F filter   1.000000    1.000000
G          0.240999    0.000000
G filter   1.000000    1.000000
```

I			
E	*0.500000*		
ED	*0.000000*		

======== LOOP 2 ========

System polynomials

A	*1.000000*	*0.000000*	*0.000000*
B	*1.000000*	*0.000000*	
D	*0.000000*	*0.000000*	

Design polynomials

B+	*1.000000*	*0.000000*
B-	*1.000000*	
C	*1.000000*	*1.000000*
P	*0.500000*	*1.000000*
Z+	*1.000000*	
Z-	*1.000000*	
Z-+	*1.000000*	

F	*1.034773*	*1.000000*
F filter	*1.000000*	*1.000000*
G	*0.354875*	*0.000000*
G filter	*1.000000*	*1.000000*
I		
E	*0.500000*	
ED	*0.000000*	

Discussion

The upper graph displays three signals: the system output $y(t)$ the setpoint $w(t)$ and the ideal model output $y_m(t)$ where $\overline{y}_m(s) = \frac{Z(s)}{P(s)}\overline{w}(s)$.

Note that the interaction cannot any longer be eliminated by the self-tuning controllers as the corresponding terms do not appear in the emulators.

Further investigations

1. Repeat for different values of a and k; determine the maximum value of k for which the response remains satisfactory.

CHAPTER 11

CSTC: The Programme

Aims. To explicitly describe the continuous-time self tuning algorithm as Pascal code. There are three sections: a summary of the programme in terms of procedure headings, procedure cross-references and the line-numbered code.

11.1. THE PROGRAMME SUMMARY

```
  4 PROGRAM CSTC(Input, Output, InLog, OutLog, InData, OutData,
  5  OutSysPar, OutEmPar);

 {-----------------------------------------------}
 {--    Polynomial output procedures          --}
 {-----------------------------------------------}

304     PROCEDURE PolWrite(VAR ListFile: TEXT;
305                    Pol: Polynomial);
323     PROCEDURE PolLineWrite(VAR ListFile: TEXT;
324                        Pol: Polynomial);

 {-----------------------------------------------}
 {--    Polynomial manipulation procedures    --}
 {-----------------------------------------------}

339     FUNCTION PolNorm(Pol: Polynomial): REAL;
359     PROCEDURE PolRemove(VAR A: Polynomial;
360                    N: INTEGER);
382     PROCEDURE PolTruncate(VAR A: Polynomial);
403     PROCEDURE PolZero(VAR Result: Polynomial;
404                   Deg: Degree);
417     PROCEDURE PolUnity(VAR Result: Polynomial;
418                    Deg: Degree);
434     PROCEDURE PolEquate(VAR Result: Polynomial;
435                     Pol: Polynomial);
450     PROCEDURE PolOfMinusS(VAR Result: Polynomial;
```

```
                Pol: Polynomial);
   PROCEDURE PolAdd(VAR Result: Polynomial;
             A, B: Polynomial);
   PROCEDURE PolMinus(VAR Result: Polynomial;
              A, B: Polynomial);
   PROCEDURE PolWeightedAdd(VAR Result: Polynomial;
                  u: REAL;
                  A: Polynomial;
                  v: REAL;
                  B: Polynomial);
   PROCEDURE PolScalarMultiply(VAR Result: Polynomial;
                    A: REAL;
                    B: Polynomial);
   PROCEDURE PolsMultiply(VAR sA: Polynomial;
                A: Polynomial);
   PROCEDURE PolsDivide(VAR A: Polynomial;
               sA: Polynomial);
   PROCEDURE PolMultiply(VAR Result: Polynomial;
                A, B: Polynomial);
   PROCEDURE PolSquare(VAR S: Polynomial;
               A: Polynomial);
      FUNCTION Even(i: INTEGER): BOOLEAN;
   PROCEDURE PolSqrt(VAR A: Polynomial;
              S: Polynomial);
   PROCEDURE PolNormalise(VAR A, B: Polynomial;
                  i: INTEGER);
   FUNCTION PolGain(VAR A: Polynomial;
             ContinuousTime: BOOLEAN): REAL;
   PROCEDURE PolUnitGain(VAR A: Polynomial;
                ContinuousTime: BOOLEAN);
   PROCEDURE PolMarkovRecursion(VAR MarkovParameter: REAL;
                    VAR E, F: Polynomial;
                    A: Polynomial);
   PROCEDURE PolDerivativeEmulator
       (VAR E, F: Polynomial;
        P, C, A: Polynomial);
   PROCEDURE PolDivide(VAR E, F: Polynomial;
               B, A: Polynomial);
   PROCEDURE PolEuclid(VAR GCD, E, F: Polynomial;
               A, B: Polynomial);
      PROCEDURE FindGCD(Alphalminus1 {A} ,
                Alphal {B} : Polynomial);
      PROCEDURE DeduceEandF(VAR Beta {F} ,
                  Gamma {E} : Polynomial);
      PROCEDURE NormaliseGCD;
   PROCEDURE PolDioRecursion(VAR E, F: Polynomial;
                  A, B: Polynomial);
   PROCEDURE PolDiophantine(VAR E, F: Polynomial;
                  VAR GCD: Polynomial;
                  A, B, PC: Polynomial);
   PROCEDURE PolZeroCancellingEmulator
       (VAR E, F: Polynomial;
        P, C, A: Polynomial;
        ZMinus, ZPlus: Polynomial;
        VAR GCDAZ: Polynomial {GCD of A and Z-} );
   PROCEDURE PolPade(VAR Pade: Polynomial;
              Deg: INTEGER;
              Delay: REAL);
```

```
{----------------------------------------------}
{--    Emulator design for self-tuning        --}
{----------------------------------------------}

1118    PROCEDURE SetDesignKnobs(VAR DesignKnobs: TypeDesignKnobs;
1119                             A, B: Polynomial;
1120                             ZeroAtOrigin, ZHasFactorB,
1121                              ContinuousTime: BOOLEAN);
1126       PROCEDURE DesignP(A, B: Polynomial);
1185    PROCEDURE PolInitialConditions
1186        (VAR InitialCondition, ED: Polynomial;
1187         A, D, E: Polynomial;
1188         DesignKnobs: TypeDesignKnobs);
1226    PROCEDURE PolEmulator(VAR F, G, InitialCondition,
1227     {Emulator numerators}
1228                           FFilter, GFilter {Emulator
1229                                          denominators}
1230                          : Polynomial;
1231                          VAR E, ED: {error} Polynomial;
1232                          VAR GCDAZ: Polynomial {GCD of A and
1233                           Z-} ;
1234                          A: Polynomial {System denominator} ;
1235                          D: Polynomial {Initial condition} ;
1236                          DesignKnobs: TypeDesignKnobs);
1247       PROCEDURE FindEandF;
1291    PROCEDURE PolDelayEmulator(VAR F, G, InitialCondition,
1292                             FFilter, GFilter: Polynomial;
1293                             VAR E, ED: Polynomial;
1294                             VAR GCDAZ, PadeDenominator,
1295                              PadeNumerator: Polynomial;
1296                             A, D: Polynomial;
1297                             DesignKnobs: TypeDesignKnobs;
1298                             Delay: REAL;
1299                             PadeOrder: INTEGER);
1336    PROCEDURE DesignEmulator(VAR STCKnobs: TypeSTCKnobs;
1337                             VAR STCState: TypeSTCState);

{----------------------------------------------}
{-- Input output  procedures                  --}
{----------------------------------------------}

1368    PROCEDURE WriteTitle(Title: TypeTitle);
1376    PROCEDURE WriteLoopTitle(Loop, Loops: INTEGER);
1397    PROCEDURE Skip(VAR F: TEXT);
1406    PROCEDURE GetSymbol(VAR F: TEXT;
1407                   VAR ChangeChar: CHAR);
1416    FUNCTION NameMatched(i: INTEGER;
1417                    VAR Name: TypeName): BOOLEAN;
1441    FUNCTION NewValue(All: BOOLEAN;
1442                  VAR ChangeChar: CHAR): BOOLEAN;
1461    PROCEDURE GetComment(VAR F: TEXT;
1462                    VAR Comment: TypeComment);
1477    PROCEDURE PutComment(VAR F: TEXT;
1478                    Comment: TypeComment);
1492    PROCEDURE EnterReal(VAR x: REAL;
1493                   Default: REAL;
1494                   Name: TypeName;
1495                   All: BOOLEAN);
```

```
1540    PROCEDURE EnterInteger(VAR x: INTEGER;
1541                    Default: INTEGER;
1542                    Name: TypeName;
1543                    All: BOOLEAN);
1587    PROCEDURE EnterBoolean(VAR x: BOOLEAN;
1588                    Default: BOOLEAN;
1589                    Name: TypeName;
1590                    All: BOOLEAN);
1598      PROCEDURE ReadBoolean(VAR F: TEXT);
1646    PROCEDURE EnterPolynomial(VAR x: Polynomial;
1647                      Default: Polynomial;
1648                      Name: TypeName;
1649                      All: BOOLEAN);
1659      PROCEDURE GetPolynomial(VAR F: TEXT;
1660                       VAR x: Polynomial;
1661                       VAR AnotherFactor: BOOLEAN);
1741    PROCEDURE WriteParameters(VAR LoopVAR: TypeLoopVAR);
1768    PROCEDURE WriteDesign(VAR LoopVAR: TypeLoopVAR);

{---------------------------------------------}
{-- Data filtering procedures               --}
{---------------------------------------------}

1845    PROCEDURE StateVariableFilter
1846        (u {Signal to be filtered} : REAL;
1847         A: Polynomial;
1848         FilterKnobs: TypeFilterKnobs;
1849         VAR FilterState: TypeFilterState);
1851      PROCEDURE cStateVariableFilter
1852          (u {Signal to be filtered} : REAL;
1853           A: Polynomial;
1854           FilterKnobs: TypeFilterKnobs;
1855           VAR FilterState: TypeFilterState);
1931      PROCEDURE dStateVariableFilter
1932          (u {Signal to be filtered} : REAL;
1933           A: Polynomial; {Now a discrete-time polynomial}
1934           FilterKnobs: TypeFilterKnobs;
1935           VAR FilterState: TypeFilterState);
1974    PROCEDURE FilterInitialise(VAR FilterState:
1975                      TypeFilterState;
1976                      ContinuousTime: BOOLEAN;
1977                      InitialValue: REAL);
1992    FUNCTION StateOutput(FilterState: TypeFilterState;
1993                  Numerator,
1994     Denominator: Polynomial): REAL;
2014    FUNCTION Filter(u {Input to filter} : REAL;
2015                Numerator, Denominator: Polynomial;
2016                FilterKnobs: TypeFilterKnobs;
2017                VAR FilterState: TypeFilterState): REAL;
2030    FUNCTION Delayed(u: REAL;
2031                 Delay: INTEGER;
2032                 VAR State: TypeDelayState): REAL;
2047    FUNCTION DelayFilter(u {Input to filter} : REAL;
2048                  Numerator, Denominator: Polynomial;
2049                  Delay: REAL;
2050                  FilterKnobs: TypeFilterKnobs;
2051                  VAR FilterState: TypeFilterState;
2052                  VAR DelFilterState: TypeDelayState):
```

```
2053      REAL;
2080     PROCEDURE TimeDelayInitialise
2081         (VAR State: TypeDelayState;
2082          InitialValue: REAL);
2096     FUNCTION TimeFor(Interval: INTEGER;
2097                    VAR Counter: INTEGER): BOOLEAN;
  END {TimeFor} ;

 {--------------------------------------------}
 {--    CSTC initialisation procedures      --}
 {--------------------------------------------}

2108     PROCEDURE SigGenInitialise(VAR SigGenKnobs:
2109                            TypeSigGenKnobs);
2125     PROCEDURE tSystemInitialise(STCKnobs: TypeSTCKnobs;
2126                            STCState: TypeSTCState;
2127                            VAR tSystemKnobs:
2128                            TypeSystemKnobs;
2129                            VAR tSystemState:
2130                            TypeSystemState;
2131                            ContinuousTime: BOOLEAN;
2132                            RunKnobs: TypeRunKnobs);
2211     PROCEDURE ModelKnobsInitialise
2212         (STCKnobs: TypeSTCKnobs;
2213          tSystemKnobs: TypeSystemKnobs;
2214          VAR ModelKnobs: TypeSystemKnobs;
2215          ContinuousTime: BOOLEAN);
2232     PROCEDURE ModelInitialise(STCKnobs: TypeSTCKnobs;
2233                           STCState: TypeSTCState;
2234                           tSystemKnobs: TypeSystemKnobs;
2235                           VAR ModelKnobs: TypeSystemKnobs;
2236                           VAR ModelState: TypeSystemState;
2237                           ContinuousTime: BOOLEAN);
2255     PROCEDURE SystemInitialise(VAR STCKnobs: TypeSTCKnobs;
2256                            VAR STCState: TypeSTCState;
2257                            RunKnobs: TypeRunKnobs);
2318     PROCEDURE DesignInitialise(VAR STCKnobs: TypeSTCKnobs;
2319                            VAR STCState: TypeSTCState;
2320                            ContinuousTime: BOOLEAN);
2384     PROCEDURE InitFilterKnobs(VAR FilterKnobs: TypeFilterKnobs
2385                           );
2400     PROCEDURE STCInitialise(VAR LoopVAR: TypeLoopVAR;
2401                         FilterKnobs: TypeFilterKnobs;
2402                         RunKnobs: TypeRunKnobs);
2407        PROCEDURE KnobsInitialise(FilterKnobs: TypeFilterKnobs;
2408                              RunKnobs: TypeRunKnobs;
2409                              VAR PutDataKnobs:
2410                              TypePutDataKnobs;
2411                              VAR STCKnobs: TypeSTCKnobs;
2412                              VAR STCState: TypeSTCState);
2414           PROCEDURE TunerInitialise
2415               (SampleInterval: REAL;
2416                VAR Knobs: TypeTunerKnobs;
2417                VAR State: TypeTunerState);
2459           PROCEDURE TuneEmInitialise
2460               (SampleInterval: REAL;
2461                VAR TunerKnobs: TypeTunerKnobs;
2462                VAR TunerState: TypeTunerState);
```

```
2469            PROCEDURE IdentifyInitialise
2470                (SampleInterval: REAL;
2471                 VAR STCKnobs: TypeSTCKnobs;
2472                 VAR STCState: TypeSTCState);
2522            PROCEDURE ControlInitialise
2523                (VAR ControlKnobs: TypeControlKnobs);
2545            PROCEDURE PutDatInitialise
2546                (VAR PutDataKnobs: TypePutDataKnobs);
2622         PROCEDURE StateInitialise(STCKnobs: TypeSTCKnobs;
2623                             VAR STCState: TypeSTCState;
2624                             ContinuousTime: BOOLEAN);
2629            PROCEDURE InitEmulator(EmKnobs: TypeEmKnobs;
2630                             VAR EmState: TypeEmState;
2631                             ContinuousTime: BOOLEAN);

{----------------------------------------------}
{--    System simulation procedures          --}
{----------------------------------------------}

2719      FUNCTION SigGen(SigGenKnobs: TypeSigGenKnobs;
2720                 Time: REAL): REAL;
2744      FUNCTION System(u: REAL;
2745                 Knobs: TypeSystemKnobs;
2746                 FilterKnobs: TypeFilterKnobs;
2747                 VAR State: TypeSystemState): REAL;
2766      FUNCTION MultiLag(u: REAL;
2767                   Lags: INTEGER;
2768                   TimeConstant: REAL;
2769                   Interactive: BOOLEAN;
2770                   FilterKnobs: TypeFilterKnobs;
2771                   VAR State: TypeLagState): REAL;

{----------------------------------------------}
{--    Self-tuner input/output procedures    --}
{----------------------------------------------}

2799      PROCEDURE GetData(VAR ThisLoopVAR: TypeLoopVAR;
2800                   VAR LoopVAR: LoopVARs;
2801                   VAR InData: TEXT;
2802                   VAR Time: REAL;
2803                   RunKnobs: TypeRunKnobs;
2804                   FilterKnobs: TypeFilterKnobs);
2806         PROCEDURE GetDataFromFile(VAR InData: TEXT);
2827         PROCEDURE Simulate;
2876      PROCEDURE PutData(VAR u: REAL;
2877                   PutDataKnobs: TypePutDataKnobs);

{----------------------------------------------}
{--    High-gain (emulator-free) control     --}
{----------------------------------------------}

2895      PROCEDURE HighGainControl(VAR u: REAL;
2896                          w, y: REAL;
2897                          FilterKnobs: TypeFilterKnobs;
2898                          VAR STCKnobs: TypeSTCKnobs;
2899                          VAR STCState: TypeSTCState);

{----------------------------------------------}
```

```
{--    Self-tuning  control                    --}
{----------------------------------------------}

2917      PROCEDURE SelfTuningControl(VAR u: REAL
2918       { The control signal } ;
2919                          w, y: REAL
2920                    { The setpoint and system output }
2921                          ;
2922                          Interaction: TypeInteraction
2923                          {Interaction terms} ;
2924                          FilterKnobs: TypeFilterKnobs
2925                     { The digital filter parameters }
2926                          ;
2927                          ExternalData: BOOLEAN;
2928                          VAR PutDataKnobs:
2929                          TypePutDataKnobs
2930                    { The control signal limits etc.  }
2931                          ;
2932                          VAR STCKnobs: TypeSTCKnobs
2933                    { The user defined STC variables }
2934                          ;
2935                          VAR STCState: TypeSTCState
2936                     { The internal state of the STC }
2937                          );
    {----------------------------------------------}
    {--   Emulator-based control procedures       --}
    {----------------------------------------------}

2949        FUNCTION Emulator(y, u: REAL;
2950                      Interaction: TypeInteraction;
2951                      NumberInteractions: INTEGER;
2952                      F, FFilter, G, GFilter,
2953                       InitialCondition: Polynomial;
2954                      GInteraction: InterPolynomial;
2955                      InputDelay: REAL;
2956                      FilterKnobs: TypeFilterKnobs;
2957                      VAR EmState: TypeEmState): REAL;
2998        FUNCTION Control(y, w: REAL;
2999                      Interaction: TypeInteraction;
3000                      STCKnobs: TypeSTCKnobs;
3001                      VAR STCState: TypeSTCState;
3002                      FilterKnobs: TypeFilterKnobs): REAL;
3004          FUNCTION ImplicitSolution
3005             (y, w: REAL;
3006              Interaction: TypeInteraction;
3007              Em0State, Em1State: TypeEmState;
3008              Q0State, Q1State: TypeFilterState;
3009              STCKnobs: TypeSTCKnobs;
3010              VAR STCState: TypeSTCState;
3011              FilterKnobs: TypeFilterKnobs): REAL;

    {----------------------------------------------}
    {--   Emulator tuning procedures              --}
    {----------------------------------------------}

3068        PROCEDURE SetData(VAR DataVector: TypeDataVector;
3069                      State: TypeEmState;
3070                      Knobs: TypeEmKnobs;
```

```
3071                    TuningInitialConditions,
3072                     IntegralAction: BOOLEAN;
3073                    NumberInteractions: INTEGER);
3126      PROCEDURE TuneEmulator(VAR Knobs: TypeEmKnobs;
3127                          State: TypeTunerState;
3128                          TuningInitialConditions,
3129                           IntegralAction: BOOLEAN;
3130                          NumberInteractions: INTEGER);
3182      PROCEDURE UpdateLeastSquaresGain
3183          (VAR TunerState: TypeTunerState;
3184           TunerKnobs: TypeTunerKnobs;
3185           DataVector: TypeDataVector);
3193         FUNCTION UTX(j: INTEGER): REAL; { computes jth element
3275      PROCEDURE IdentifySystem(y, u: REAL;
3276                             Interaction: TypeInteraction;
3277                             FilterKnobs: TypeFilterKnobs;
3278                             VAR STCKnobs: TypeSTCKnobs;
3279                             VAR STCState: TypeSTCState);
3286         PROCEDURE TuneDelay(VAR Delay: REAL;
3287                            State: TypeTunerState;
3288                            NumberOfParameters: INTEGER);
3299         PROCEDURE SetDelayData(VAR DataVector: TypeDataVector;
3300                               State: TypeEmState;
3301                               Knobs: TypeEmKnobs);
3378      PROCEDURE TunePhiEmulator(y: REAL;
3379                              FilterKnobs: TypeFilterKnobs;
3380                              VAR STCKnobs: TypeSTCKnobs;
3381                              VAR STCState: TypeSTCState);
3415      PROCEDURE TuneLambdaEmulator
3416          (y, u: REAL;
3417           Interaction: TypeInteraction;
3418           FilterKnobs: TypeFilterKnobs;
3419           LambdaNumerator, LambdaDenominator: Polynomial;
3420           ZLambda, PLambda: Polynomial;
3421           VAR STCKnobs: TypeSTCKnobs;
3422           VAR STCState: TypeSTCState);

   {-----------------------------------------------}
   {--   Self-tuning  control: procedure body   --}
   {-----------------------------------------------}

3494    BEGIN {SelfTuningControl}

3572     PROCEDURE RunInitialise;
3611     PROCEDURE SimulationInitialise
3612         (VAR ThisLoopVAR: TypeLoopVAR;
3613          FilterKnobs: TypeFilterKnobs;
3614          RunKnobs: TypeRunKnobs);
3648     PROCEDURE Run;
3655        PROCEDURE WriteData(VAR ThisLoopVAR: TypeLoopVAR);
3706        PROCEDURE WriteLnData;
3714        PROCEDURE OneTimeStep(VAR ThisLoopVAR: TypeLoopVAR);
3749        PROCEDURE Splice(VAR ThisLoopVAR: TypeLoopVAR);
3777        FUNCTION NoMore: BOOLEAN;
3783           PROCEDURE PreventBump;
3937     FUNCTION Chapter(VAR All: BOOLEAN): INTEGER;

   {-----------------------------------------------}
```

```
{--    Body of CSTC                          --}
{--------------------------------------------}

3967  BEGIN {CSTC}
```

11.2. THE PROGRAMME PROCEDURAL INDEX

```
    Chapter          Head: 3937  Body: 3943
    Calls      EnterInteger
    Called by  CSTC
Control          Head: 2998  Body: 3049
    Calls      ImplicitSolution Filter
    Called by  SelfTuningContro
ControlInitialis Head: 2522  Body: 2525
    Calls      WriteTitle       EnterPolynomial
    Called by  KnobsInitialise
cStateVariableFi Head: 1851  Body: 1876
    Called by  StateVariableFil
CSTC             Head: 4  Body: 3967
    Calls      PolZero          PolUnity         Chapter
               RunInitialise    InitFilterKnobs  STCInitialise
               SimulationInitia EnterBoolean     Run
               SystemInitialise DesignInitialise DesignEmulator
               WriteDesign      WriteParameters  EnterInteger
               WriteLoopTitle
DeduceEandF      Head: 919  Body: 926
    Calls      PolScalarMultipl PolZero          PolUnity
               PolEquate        PolMultiply      PolAdd
    Called by  PolEuclid
Delayed          Head: 2030  Body: 2036
    Called by  DelayFilter      System
DelayFilter      Head: 2047  Body: 2064
    Calls      Delayed          StateVariableFil StateOutput
    Called by  Emulator         TuneLambdaEmulat
DesignEmulator   Head: 1336  Body: 1346
    Calls      PolDelayEmulator PolMultiply      PolZero
    Called by  KnobsInitialise  SelfTuningContro CSTC
DesignInitialise Head: 2318  Body: 2325
    Calls      PolZero          PolUnity         WriteTitle
               EnterBoolean     EnterPolynomial  EnterReal
               SetDesignKnobs   EnterInteger
    Called by  KnobsInitialise  CSTC
DesignP          Head: 1126  Body: 1131
    Calls      PolsDivide       PolSquare        PolWeightedAdd
               PolSqrt          PolScalarMultipl
    Called by  SetDesignKnobs
dStateVariableFi Head: 1931  Body: 1944
    Called by  StateVariableFil
Emulator         Head: 2949  Body: 2966
    Calls      DelayFilter      Filter
    Called by  ImplicitSolution IdentifySystem   TuneLambdaEmulat
               SelfTuningContro
EnterBoolean     Head: 1587  Body: 1608
    Calls      GetSymbol        NameMatched      ReadBoolean
               GetComment       PutComment       NewValue
    Called by  tSystemInitialis DesignInitialise InitFilterKnobs
               TunerInitialise  IdentifyInitiali PutDatInitialise
               KnobsInitialise  STCInitialise    RunInitialise
               NoMore           CSTC
EnterInteger     Head: 1540  Body: 1550
    Calls      GetSymbol        NameMatched      GetComment
               PutComment       NewValue
    Called by  tSystemInitialis SystemInitialise DesignInitialise
```

```
            PolTruncate
    Called by  SelfTuningContro
ImplicitSolution  Head: 3004  Body: 3016
    Calls      Emulator      Filter
    Called by  Control
InitEmulator    Head: 2629  Body: 2636
    Calls      FilterInitialise
    Called by  StateInitialise
InitFilterKnobs  Head: 2384  Body: 2387
    Calls      WriteTitle      EnterReal       EnterInteger
            EnterBoolean
    Called by  CSTC
KnobsInitialise  Head: 2407  Body: 2560
    Calls      SystemInitialise DesignInitialise DesignEmulator
            WriteDesign     WriteTitle      EnterBoolean
            IdentifyInitiali TuneEmInitialise ControlInitialis
            PutDatInitialise PolEquate
    Called by  STCInitialise
ModelInitialise  Head: 2232  Body: 2239
    Calls      ModelKnobsInitia PolEquate       FilterInitialise
            TimeDelayInitial
    Called by  SimulationInitia
ModelKnobsInitia  Head: 2211  Body: 2217
    Calls      PolUnitGain     PolMultiply     PolEquate
    Called by  ModelInitialise
MultiLag        Head: 2766  Body: 2776
    Called by  Simulate
NameMatched       Head: 1416  Body: 1422
    Calls      NameMatched
    Called by  NameMatched     EnterReal       EnterInteger
            EnterBoolean    EnterPolynomial
NewValue          Head: 1441  Body: 1447
    Called by  EnterReal       EnterInteger    EnterBoolean
            EnterPolynomial
NoMore            Head: 3777  Body: 3803
    Calls      EnterBoolean    EnterReal       WriteLoopTitle
            WriteTitle      EnterPolynomial  PreventBump
    Called by  Run
NormaliseGCD      Head: 956  Body: 961
    Calls      PolScalarMultipl
    Called by  PolEuclid
OneTimeStep       Head: 3714  Body: 3716
    Calls      GetData         Filter          System
            HighGainControl PutData         SelfTuningContro
            WriteData
    Called by  Run
PolAdd            Head: 469  Body: 477
    Called by  DeduceEandF
PolDelayEmulator  Head: 1291  Body: 1314
    Calls      PolPade         PolOfMinusS     PolMultiply
            PolEmulator
    Called by  DesignEmulator
PolDerivativeEmu  Head: 813  Body: 822
    Calls      PolZero         PolMultiply     PolEquate
            PolScalarMultipl PolMarkovRecursi PolWeightedAdd
    Called by  PolDivide       PolInitialCondit FindEandF
PolDiophantine    Head: 1022  Body: 1045
    Calls      PolEuclid       PolEquate       PolScalarMultipl
```

PolDioRecursion PolWeightedAdd
Called by PolZeroCancellin
PolDioRecursion Head: 981 Body: 997
Calls PolsMultiply PolWeightedAdd PolTruncate
Called by PolDiophantine
PolDivide Head: 848 Body: 860
Calls PolUnity PolDerivativeEmu
Called by FindGCD PolInitialCondit PolEmulator
PolEmulator Head: 1226 Body: 1263
Calls FindEandF PolMultiply PolDivide
PolInitialCondit
Called by PolDelayEmulator
PolEquate Head: 434 Body: 442
Called by PolDerivativeEmu FindGCD DeduceEandF
PolDiophantine SetDesignKnobs EnterPolynomial
tSystemInitialis ModelKnobsInitia ModelInitialise
IdentifyInitiali KnobsInitialise IdentifySystem
PolEuclid Head: 865 Body: 968
Calls FindGCD DeduceEandF PolTruncate
NormaliseGCD
Called by PolDiophantine
PolGain Head: 743 Body: 755
Called by PolUnitGain SetDesignKnobs
PolInitialCondit Head: 1185 Body: 1201
Calls PolMultiply PolDerivativeEmu PolZeroCancellin
PolWeightedAdd PolDivide PolTruncate
Called by PolEmulator
PolLineWrite Head: 323 Body: 330
Calls PolWrite
Called by WriteParameters WriteDesign SystemInitialise
IdentifyInitiali
PolMarkovRecursi Head: 782 Body: 792
Calls PolsMultiply PolWeightedAdd PolRemove
Called by PolDerivativeEmu
PolMinus Head: 504 Body: 512
Called by IdentifyInitiali IdentifySystem
PolMultiply Head: 628 Body: 636
Calls PolZero
Called by PolDerivativeEmu DeduceEandF PolZeroCancellin
SetDesignKnobs PolInitialCondit FindEandF
PolEmulator PolDelayEmulator DesignEmulator
EnterPolynomial ModelKnobsInitia
PolNorm Head: 339 Body: 349
Called by FindGCD
PolNormalise Head: 718 Body: 729
Calls PolScalarMultipl
Called by SystemInitialise
PolOfMinusS Head: 450 Body: 458
Called by PolDelayEmulator
PolPade Head: 1089 Body: 1100
Called by PolDelayEmulator
PolRemove Head: 359 Body: 369
Called by PolTruncate PolMarkovRecursi IdentifyInitiali
PolScalarMultipl Head: 577 Body: 586
Called by PolNormalise PolUnitGain PolDerivativeEmu
DeduceEandF NormaliseGCD PolDiophantine
DesignP SetDesignKnobs IdentifyInitiali
PolsDivide Head: 610 Body: 621

```
    Called by  DesignP        SetDesignKnobs  tSystemInitialis
PolsMultiply    Head: 594  Body: 603
    Called by  PolMarkovRecursi PolDioRecursion  SetDesignKnobs
               tSystemInitialis SystemInitialise SetDelayData
PolSqrt        Head: 685  Body: 691
    Called by  DesignP
PolSquare      Head: 647  Body: 664
    Calls      PolZero        Even
    Called by  DesignP
PolTruncate    Head: 382  Body: 391
    Calls      PolRemove
    Called by  PolEuclid      PolDioRecursion  PolInitialCondit
               IdentifySystem
PolUnitGain    Head: 766  Body: 776
    Calls      PolGain        PolScalarMultipl
    Called by  ModelKnobsInitia
PolUnity       Head: 417  Body: 425
    Called by  PolDivide      DeduceEandF     SetDesignKnobs
               SystemInitialise DesignInitialise STCInitialise
               SelfTuningContro CSTC
PolWeightedAdd  Head: 539  Body: 550
    Called by  PolMarkovRecursi PolDerivativeEmu PolDioRecursion
               PolDiophantine  DesignP        PolInitialCondit
PolWrite       Head: 304  Body: 317
    Called by  PolLineWrite   WriteData
PolZero        Head: 403  Body: 412
    Called by  PolMultiply    PolSquare      PolDerivativeEmu
               DeduceEandF    DesignEmulator  SystemInitialise
               DesignInitialise STCInitialise  CSTC
PolZeroCancellin Head: 1068  Body: 1083
    Calls      PolMultiply    PolDiophantine
    Called by  PolInitialCondit FindEandF
PreventBump    Head: 3783  Body: 3792
    Calls      StateOutput
    Called by  NoMore
PutComment     Head: 1477  Body: 1483
    Called by  EnterReal      EnterInteger    EnterBoolean
               EnterPolynomial
PutData        Head: 2876  Body: 2879
    Called by  SelfTuningContro OneTimeStep
PutDatInitialise Head: 2545  Body: 2548
    Calls      EnterReal      EnterBoolean
    Called by  KnobsInitialise
ReadBoolean    Head: 1598  Body: 1600
    Called by  EnterBoolean
Run            Head: 3648  Body: 3856
    Calls      TimeFor        Splice         OneTimeStep
               WriteLnData    NoMore
    Called by  CSTC
RunInitialise  Head: 3572  Body: 3577
    Calls      WriteTitle     EnterBoolean    EnterReal
               EnterInteger
    Called by  CSTC
SelfTuningContro Head: 2917  Body: 3494
    Calls      PolUnity       Control        PutData
               IdentifySystem  SetDesignKnobs  DesignEmulator
               TuneLambdaEmulat TunePhiEmulator  Emulator
               StateVariableFil
```

```
             EnterBoolean    PolsMultiply     FilterInitialise
             TimeDelayInitial
    Called by  SimulationInitia
TuneDelay        Head: 3286  Body: 3290
    Called by  IdentifySystem
TuneEmInitialise Head: 2459  Body: 2464
    Calls      TunerInitialise
    Called by  KnobsInitialise
TuneEmulator     Head: 3126  Body: 3137
    Called by  IdentifySystem  TunePhiEmulator TuneLambdaEmulat
TuneLambdaEmulat Head: 3415  Body: 3430
    Calls      Filter          DelayFilter     Emulator
               TimeFor         SetData         UpdateLeastSquar
               TuneEmulator
    Called by  SelfTuningContro
TunePhiEmulator  Head: 3378  Body: 3386
    Calls      Filter          TimeFor         SetData
               UpdateLeastSquar TuneEmulator
    Called by  SelfTuningContro
TunerInitialise  Head: 2414  Body: 2422
    Calls      EnterReal       EnterBoolean    EnterInteger
    Called by  TuneEmInitialise IdentifyInitiali
UpdateLeastSquar Head: 3182  Body: 3227
    Calls      UTX
    Called by  IdentifySystem  TunePhiEmulator TuneLambdaEmulat
UTX              Head: 3193  Body: 3200
    Called by  UpdateLeastSquar
WriteData        Head: 3655  Body: 3660
    Calls      PolWrite
    Called by  OneTimeStep
WriteDesign      Head: 1768  Body: 1775
    Calls      WriteParameters PolLineWrite
    Called by  KnobsInitialise  CSTC
WriteLnData      Head: 3706  Body: 3708
    Called by  Run
WriteLoopTitle   Head: 1376  Body: 1382
    Calls      WriteTitle
    Called by  NoMore          CSTC
WriteParameters  Head: 1741  Body: 1748
    Calls      PolLineWrite
    Called by  WriteDesign     CSTC
WriteTitle       Head: 1368  Body: 1370
    Called by  WriteLoopTitle  tSystemInitialis SystemInitialise
               DesignInitialise InitFilterKnobs  IdentifyInitiali
               ControlInitialis KnobsInitialise  STCInitialise
               RunInitialise    SimulationInitia NoMore
```

11.3. THE PROGRAMME CODE

```
{$double} {Oregon Pascal-2 double precision switch}
{[b+]}

PROGRAM CSTC(Input, Output, InLog, OutLog, InData, OutData,
 OutSysPar, OutEmPar);

{*******************************************************
 NOTICE OF COPYRIGHT AND OWNERSHIP OF SOFTWARE:

 Copyright (c) 1990 Research Studies Press Ltd.
 All Rights Reserved.

 This software is supplied in conjunction with the book:
 "Continuous-Time Self-Tuning Control: Volume II - Implementation"
 written by P.J. Gawthrop and published by:

      Research Studies Press Ltd.,
      24 Belvedere Road,
      Taunton,
      Somerset,
      England. TA1 1HD

 The software is provided for use only with this book and
 the software, or any part or extension of it,
 may not be used, marketed or distributed for profit.

 Copies of the software may be made only for the purposes
 of security or convenience of use and may not be made
 for use dissociated from the copy of the book with
 which is is supplied or for use other than simulation
 of the algorithms described in the book.

 DISCLAIMER. The software is supplied on the understanding
 that there is NO warranty as to errors, mathematically
 incorrect results or recompilation difficulties.

 The author and Reseach Studies Press do not accept any
 responsibility for errors, misstatements or omissions
 that may or may not lead to damage or loss of property.

 Whether this program is copied in whole or in part and
 whether this program is copied in original or in modified
 form, ALL COPIES OF THIS PROGRAM MUST DISPLAY THIS NOTICE
 OF COPYRIGHT AND OWNERSHIP IN FULL.
*******************************************************}

 CONST
  Version = 'Version 6.0';

  TwoPi = 6.2831853;

  MaxParameters = 10;
  MaxUFactor = 45;
  MaxDegree = 10;
  MaxState = 5;
  MaxDelay = 100;
```

```
  MaxLags = 21;
  MaxNumberInteractions = 2;
  MaxLoops = 2;

  LengthName = 25;
  LengthComment = 100;
  LengthTitle = 15;

  fw = 12; {Output format for real numbers}
  dp = 6;
  ChangeSymbol = '#';
  MultiplySymbol = '*';
  Pretty = ' ====== ';
  ProgressReports = 4;

TYPE
  TypeName = PACKED ARRAY [1..LengthName] OF CHAR;
  TypeTitle = PACKED ARRAY [1..LengthTitle] OF CHAR;

  TypeComment =
    RECORD
      Str: PACKED ARRAY [1..LengthComment] OF CHAR;
      Length: INTEGER;
    END;

  Degree = - 1..MaxDegree;

  Polynomial =
    RECORD
      Deg: Degree;
      Coeff: ARRAY [0..MaxDegree] OF REAL
    END;

  Vector = ARRAY [1..MaxParameters] OF REAL;

  StateVector = ARRAY [0..MaxState] OF REAL;

  TypeFilterKnobs =
    RECORD
      SampleInterval: REAL;
      ApproximationOrder: INTEGER;
      ContinuousTime: BOOLEAN;
      ConstantBetweenSamples: BOOLEAN;
    END;

  TypeFilterState =
    RECORD
      State: StateVector;
      Old: REAL;
    END;

  TypeDelayState =
    RECORD
      InPointer, OutPointer: 0..MaxDelay;
      Buffer: ARRAY [0..MaxDelay] OF REAL;
    END;

  TypeLagState = ARRAY [0..MaxLags] OF REAL;
```

```

TypeInteraction = ARRAY
  [1..MaxNumberInteractions] OF REAL;

InterPolynomial = ARRAY
  [1..MaxNumberInteractions] OF Polynomial;

TypeIState = ARRAY
  [1..MaxNumberInteractions] OF TypeFilterState;

TypeDesignKnobs =
  RECORD
    P, Z, ZPlus, ZMinus, ZMinusPlus, BMinus, BPlus,
    C: Polynomial;
    LQWeight: REAL;
    LQ: BOOLEAN;
  END;

TypeSystemKnobs =
  RECORD
    A, B, D: Polynomial;
    NumberInteractions: INTEGER;
    BInteraction: InterPolynomial;
    Delay: REAL;
    Lags: INTEGER;
    LagTimeConstant: REAL;
    Interactive: BOOLEAN;
  END;

TypeSystemState =
  RECORD
    FilterState, ICState: TypeFilterState;
    DelayState: TypeDelayState;
    LagState: TypeLagState;
  END;

TypeEmKnobs =
  RECORD
    F, G, FFilter, GFilter,
    InitialCondition: Polynomial;
    GInteraction: InterPolynomial;
  END;

TypeEmState =
  RECORD
    uState, yState, ICState: TypeFilterState;
    InterState, InterFState: TypeIState;
    DelFiltState: TypeDelayState;
  END;

TypeErrorPolynomials =
  RECORD
    E, ED: Polynomial;
    EI: InterPolynomial;
  END;

TypeControlKnobs =
  RECORD
```

```
    qNumerator, qDenominator: Polynomial;
    rNumerator, rDenominator: Polynomial;
  END;

TypeTunerKnobs =
  RECORD
    InitialVariance: REAL;
    ForgetTime, ForgetFactor: REAL;
    DeadBand: REAL;
    On: BOOLEAN;
    TuneInterval: INTEGER;
  END;

TypeTunerState =
  RECORD
    TuningGain, Variance: ARRAY
      [1..MaxParameters] OF REAL;
    UFactor: ARRAY [1..MaxUFactor] OF REAL;
    EstimationError, Sigma, Sigma1: REAL;
    TuneCounter: INTEGER;
  END;

TypeDataVector =
  RECORD
    NumberOfParameters: INTEGER;
    Data: Vector;
  END;

TypeSTCKnobs =
  RECORD
    IdentifyingSystem, IdentifyingRational,
     IdentifyingDelay, SelfTuning, Explicit,
     UsingLambda, TuningInitialConditions, ZHasFactorB,
     IntegralAction, Auto: BOOLEAN;
    DesignKnobs: TypeDesignKnobs;
    TunerKnobs, IdentifyKnobs: TypeTunerKnobs;
    ControlKnobs: TypeControlKnobs;
    PadeOrder: INTEGER;
    PadeDenominator, PadeNumerator: Polynomial;
    GCDAZ: Polynomial;
  END;

TypeSTCState =
  RECORD
    SystemKnobs: TypeSystemKnobs;
    EmKnobs, SysEmKnobs: TypeEmKnobs;
    ErrorPolynomials: TypeErrorPolynomials;
    Phi, PhiHat: REAL;
    EmState, LambdaEmState, SysEmState: TypeEmState;
    qState, wState, PhicState, yLambdaState,
     uLambdaState: TypeFilterState;
    iLambdaState: TypeIState;
    uDelayState, yDelayState: TypeDelayState;
    TunerState, IdentState: TypeTunerState;
  END;

TypeSigGenKnobs =
  RECORD
```

```
      StepAmplitude, SquareAmplitude, CosAmplitude,
       Period: REAL;
    END;

  TypePutDataKnobs =
    RECORD
      Max, Min: REAL;
      Switched: BOOLEAN;
    END;

  TypeLoopVAR =
    RECORD
      ThisLoop: INTEGER;
      UsingHighGainControl: BOOLEAN;

      SetPointKnobs, InDisturbKnobs,
      OutDisturbKnobs: TypeSigGenKnobs;

      tSystemKnobs, ModelKnobs: TypeSystemKnobs;
      PutDataKnobs: TypePutDataKnobs;

      STCKnobs: TypeSTCKnobs;
      STCState: TypeSTCState;

      tSystemState, ModelState: TypeSystemState;
      InteractionState: ARRAY
        [1..MaxLoops] OF TypeFilterState;

      y, y0, u, w, wf, InDist, OutDist: REAL;
      Interaction: TypeInteraction;

    END {LoopVAR} ;

  TypeRunKnobs =
    RECORD
      PrintInterval: INTEGER;
      LastTime, ExtraTime: REAL;
      Loops: INTEGER;
      ExternalData: BOOLEAN;
      Cascade, OutputCoupled: BOOLEAN;
      CorrectSystem: BOOLEAN;
    END;

  LoopVARs = ARRAY [1..MaxLoops] OF TypeLoopVAR;

VAR
  All: BOOLEAN;
  InLog,
  OutLog {Input and output entry parameters} : TEXT;
  InData {Input data} : TEXT;
  OutData {Output data} : TEXT;
  OutEmPar,
  OutSysPar {The parameter estimates etc.} : TEXT;

  Small: REAL;
  Zero, One: Polynomial;

  Time: REAL;
```

```
    PrintCounter: INTEGER;
    PrintNow: BOOLEAN;

    RunKnobs: TypeRunKnobs;

    FilterKnobs: TypeFilterKnobs;

    Loop: INTEGER;
    LoopVAR: LoopVARs;
    LoopInteraction: TypeInteraction;

    {----------------------------------------------}
    {--   Polynomial output procedures           --}
    {----------------------------------------------}

  PROCEDURE PolWrite(VAR ListFile: TEXT;
                     Pol: Polynomial);

{--    Writes polynomial Pol to ListFile
c --}

    CONST
      fw = 12;
      dp = 6;

    VAR
      i: Degree;

    BEGIN
      WITH Pol DO
        FOR i := 0 TO Deg DO
          Write(ListFile, ' ', Coeff[i]: fw: dp);
    END;

  PROCEDURE PolLineWrite(VAR ListFile: TEXT;
                         Pol: Polynomial);

{--    Writes polynomial Pol to ListFile -
c      and appends WriteLn
c --}

    BEGIN
      PolWrite(ListFile, Pol);
      WriteLn(ListFile);
    END;

    {----------------------------------------------}
    {--   Polynomial manipulation procedures     --}
    {----------------------------------------------}

  FUNCTION PolNorm(Pol: Polynomial): REAL;

{--    Finds the maximum  absolute value of the
c      coefficients ofthe polynomial Pol
c --}

    VAR
      i: INTEGER;
```

```
      Max: REAL;

    BEGIN {PolNorm}
      Max := 0.0;

      WITH Pol DO
        FOR i := 0 TO Deg DO
          IF Abs(Coeff[i]) > Max THEN Max := Abs(Coeff[i]);

      PolNorm := Max;
    END {PolNorm} ;

  PROCEDURE PolRemove(VAR A: Polynomial;
                    N: INTEGER);

 {-- Removes N leading coefficients from
c     polynomial A
c  --}

    VAR
      i: INTEGER;

    BEGIN {PolRemove}

      WITH A DO
        BEGIN

        A.Deg := A.Deg - N;

        IF N > 0 THEN
          FOR i := 0 TO Deg DO Coeff[i] := Coeff[i + N];

        END;
    END {PolRemove} ;

  PROCEDURE PolTruncate(VAR A: Polynomial);

 {-- Removes leading coefficients from
c    polynomial Awith value <= Small
c  --}

    VAR
      N: INTEGER;

    BEGIN {PolTruncate}

      WITH A DO
        BEGIN
        N := 0;
        WHILE (N <= Deg) AND (Abs(Coeff[N]) <= Small) DO
          N := N + 1;

        PolRemove(A, N);
        END;
    END {PolTruncate} ;

  PROCEDURE PolZero(VAR Result: Polynomial;
                  Deg: Degree);
```

```
  { Sets result to zero polynomial
c   of specifed degree }

   VAR
     i: INTEGER;

   BEGIN
     Result.Deg := Deg;
     WITH Result DO FOR i := 0 TO Deg DO Coeff[i] := 0.0;
   END;

PROCEDURE PolUnity(VAR Result: Polynomial;
                   Deg: Degree);

  { Sets Result to unit polynomial of specified degree }

   VAR
     i: INTEGER;

   BEGIN

     Result.Deg := Deg;
     WITH Result DO
       FOR i := 0 TO Deg - 1 DO Coeff[i] := 0.0;
     Result.Coeff[Deg] := 1.0;

   END;

PROCEDURE PolEquate(VAR Result: Polynomial;
                    Pol: Polynomial);

  { Result := Pol }

   VAR
     i: Degree;

   BEGIN

     Result.Deg := Pol.Deg;
     FOR i := 0 TO Result.Deg DO
       Result.Coeff[i] := Pol.Coeff[i];

   END;

PROCEDURE PolOfMinusS(VAR Result: Polynomial;
                      Pol: Polynomial);

  { Result(s) := Pol(-s) }

   VAR
     Minus, i: INTEGER;

   BEGIN
     Result.Deg := Pol.Deg;
     Minus := - 1;

     FOR i := Result.Deg DOWNTO 0 DO
```

```
      BEGIN
      Minus := - 1 * Minus;
      Result.Coeff[i] := Minus * Pol.Coeff[i];
      END;
  END;

PROCEDURE PolAdd(VAR Result: Polynomial;
                 A, B: Polynomial);

 {Result := A + B}

  VAR
    i, Shift: Degree;

  BEGIN {PolAdd}
    IF A.Deg > B.Deg THEN Result.Deg := A.Deg
    ELSE Result.Deg := B.Deg;

    IF A.Deg > B.Deg THEN
      BEGIN
      Shift := A.Deg - B.Deg;
      FOR i := 0 TO Result.Deg DO
        IF i - Shift < 0 THEN
          Result.Coeff[i] := A.Coeff[i]
        ELSE
          Result.Coeff[i] := A.Coeff[i] + B.Coeff[i -
                             Shift];
      END
    ELSE
      BEGIN
      Shift := B.Deg - A.Deg;
      FOR i := 0 TO Result.Deg DO
        IF i - Shift < 0 THEN
          Result.Coeff[i] := B.Coeff[i]
        ELSE
          Result.Coeff[i] := B.Coeff[i] + A.Coeff[i -
                             Shift];
      END;

  END {PolAdd} ;

PROCEDURE PolMinus(VAR Result: Polynomial;
                   A, B: Polynomial);

 {Result := A - B}

  VAR
    i, Shift: Degree;

  BEGIN {PolMinus}
    IF A.Deg > B.Deg THEN Result.Deg := A.Deg
    ELSE Result.Deg := B.Deg;

    IF A.Deg > B.Deg THEN
      BEGIN
      Shift := A.Deg - B.Deg;
      FOR i := 0 TO Result.Deg DO
        IF i - Shift < 0 THEN
```

```
        Result.Coeff[i] := A.Coeff[i]
      ELSE
        Result.Coeff[i] := A.Coeff[i] - B.Coeff[i -
                           Shift];
    END
  ELSE
    BEGIN
    Shift := B.Deg - A.Deg;
    FOR i := 0 TO Result.Deg DO
      IF i - Shift < 0 THEN
        Result.Coeff[i] := - B.Coeff[i]
      ELSE
        Result.Coeff[i] := - B.Coeff[i] + A.Coeff[i -
                           Shift];
    END;

  END {PolMinus} ;

PROCEDURE PolWeightedAdd(VAR Result: Polynomial;
                         u: REAL;
                         A: Polynomial;
                         v: REAL;
                         B: Polynomial);

 {Result := uA + vB}

  VAR
   i, Shift: Degree;

  BEGIN {PolWeightedAdd}
   IF A.Deg > B.Deg THEN Result.Deg := A.Deg
   ELSE Result.Deg := B.Deg;

   IF A.Deg > B.Deg THEN
     BEGIN
     Shift := A.Deg - B.Deg;
     FOR i := 0 TO Result.Deg DO
       IF i - Shift < 0 THEN
         Result.Coeff[i] := u * A.Coeff[i]
       ELSE
         Result.Coeff[i] := u * A.Coeff[i] + v *
                            B.Coeff[i - Shift];
     END
   ELSE
     BEGIN
     Shift := B.Deg - A.Deg;
     FOR i := 0 TO Result.Deg DO
       IF i - Shift < 0 THEN
         Result.Coeff[i] := v * B.Coeff[i]
       ELSE
         Result.Coeff[i] := v * B.Coeff[i] + u *
                            A.Coeff[i - Shift];
     END;

  END {PolWeightedAdd} ;

PROCEDURE PolScalarMultiply(VAR Result: Polynomial;
                            A: REAL;
```

```
                          B: Polynomial);

  { Computes Result = a*B , a real}

   VAR
     i: INTEGER;

   BEGIN

     FOR i := 0 TO B.Deg DO
       Result.Coeff[i] := A * B.Coeff[i];

     Result.Deg := B.Deg;
   END;

 PROCEDURE PolsMultiply(VAR sA: Polynomial;
                        A: Polynomial);

{-- Multiplies A by s to give sA
--}

   VAR
     i: INTEGER;

   BEGIN {PolsMultiply}
     FOR i := 0 TO A.Deg DO sA.Coeff[i] := A.Coeff[i];

     sA.Deg := A.Deg + 1;
     sA.Coeff[sA.Deg] := 0.0;
   END {PolsMultiply} ;

 PROCEDURE PolsDivide(VAR A: Polynomial;
                      sA: Polynomial);

{-- Divides sA by s to give A.
sA is assumed to have a zero coefficient
in the appropriate place.
--}

   VAR
     i: INTEGER;

   BEGIN {PolsDivide}
     A.Deg := sA.Deg - 1;

     FOR i := 0 TO A.Deg DO A.Coeff[i] := sA.Coeff[i];

   END {PolsDivide} ;

 PROCEDURE PolMultiply(VAR Result: Polynomial;
                       A, B: Polynomial);

  { Computes Result = A*B }

   VAR
     i, j: INTEGER;

   BEGIN
```

```
      PolZero(Result, A.Deg + B.Deg);

      WITH Result DO
        FOR i := 0 TO A.Deg DO
          FOR j := 0 TO B.Deg DO
            Coeff[i + j] := Coeff[i + j] + A.Coeff[i] *
                            B.Coeff[j];

    END;

  PROCEDURE PolSquare(VAR S: Polynomial;
                      A: Polynomial);

   { Computes S(s^2) = A(s)A(-s)

     I.e. the coefficients of S are the even index
     coefficients  of A(s)A(-s) }

    VAR
      i, j, ij, Minus: INTEGER;

    FUNCTION Even(i: INTEGER): BOOLEAN;

      BEGIN {Even}
        Even := (i MOD 2) = 0;
      END {Even} ;

    BEGIN {PolSquare}
      PolZero(S, A.Deg);

      WITH S DO
        FOR i := A.Deg DOWNTO 0 DO
          BEGIN
          Minus := - 1;
          FOR j := A.Deg DOWNTO 0 DO
            BEGIN
            Minus := - Minus;
            IF Even(i + j) THEN
              BEGIN
              ij := (i + j) DIV 2;
              Coeff[ij] := Coeff[ij] + Minus * A.Coeff[i] *
                           A.Coeff[j];
              END;
            END;
          END;

    END {PolSquare} ;

  PROCEDURE PolSqrt(VAR A: Polynomial;
                    S: Polynomial);

 { Computes A(s) where A is stable and A(s)A(-s) = S(s^2)
   Only programmed for degree of A two or less}

    BEGIN {PolSqrt}
      A.Deg := S.Deg;

      WITH A DO
```

```
      IF Deg > 2 THEN
        WriteLn(
           '*** PolSqrt defined only for degrees up to 2'
               )
      ELSE
        CASE Deg OF
         0: Coeff[0] := Sqrt(S.Coeff[0]);
         1:
           BEGIN
           Coeff[0] := Sqrt( - S.Coeff[0]);
           Coeff[1] := Sqrt(S.Coeff[1]);
           END;
         2:
           BEGIN
           Coeff[0] := Sqrt(S.Coeff[0]);
           Coeff[2] := Sqrt(S.Coeff[2]);
           Coeff[1] := Sqrt(Sqr(S.Coeff[1]) + 2 *
                            Coeff[0] * Coeff[2]);
           END;
         END {CASE} ;

  END {PolSqrt} ;

 PROCEDURE PolNormalise(VAR A, B: Polynomial;
                        i: INTEGER);
{-- Normalises both polynomials with respect
    to ith coefficient of A.
    If i> Deg(A) or ith coefficient of A is zero
    then nothing is done.
--}

  VAR
    ai: REAL;

  BEGIN {PolNormalise}

    IF (i <= A.Deg) AND (i >= 0) THEN
      BEGIN
      ai := A.Coeff[i];

      IF Abs(ai) > Small THEN
        BEGIN
        PolScalarMultiply(A, 1.0 / ai, A);
        PolScalarMultiply(B, 1.0 / ai, B);
        END;
      END;
  END {PolNormalise} ;

 FUNCTION PolGain(VAR A: Polynomial;
                  ContinuousTime: BOOLEAN): REAL;

{-- Finds steady state gain of polynomial:
        A(0) in continuous time
        A(1) in discrete-time
--}

  VAR
    Gain: REAL;
```

```
      i: INTEGER;

    BEGIN {PolGain}
      WITH A DO
        BEGIN
        Gain := Coeff[Deg];
        IF NOT ContinuousTime THEN
          FOR i := Deg - 1 DOWNTO 0 DO
            Gain := Gain + Coeff[i];
        END;
      PolGain := Gain;
    END {PolGain} ;

  PROCEDURE PolUnitGain(VAR A: Polynomial;
                        ContinuousTime: BOOLEAN);

{-- Normalises polynomial A(s) such that A(0) = 1
c    But nothing is done if A(0)<Small initially.
c --}

    VAR
      Gain: REAL;

    BEGIN {PolUnitGain}
      Gain := PolGain(A, ContinuousTime);
      IF Abs(Gain) > Small THEN
        PolScalarMultiply(A, 1.0 / Gain, A);
    END {PolUnitGain} ;

  PROCEDURE PolMarkovRecursion(VAR MarkovParameter: REAL;
                               VAR E, F: Polynomial;
                               A: Polynomial);
{ Given polynomials A, E and F satisfying:
c       s^i/A = E + F/A
c  computes the new polynomials E and F satisfying
c       s^(i+1)/A = E + F/A
c   together with the (scalar)  i+1th Markov
c    parameter of 1/A }

    BEGIN { PolMarkovRecursion }

      IF F.Deg < A.Deg - 1 THEN
        BEGIN
        MarkovParameter := 0.0;
        PolsMultiply(F, F);
        END
      ELSE
        BEGIN
        MarkovParameter := F.Coeff[0] / A.Coeff[0]; { (2a) }

        PolsMultiply(E, E); { (2b) }
        WITH E DO Coeff[Deg] := MarkovParameter;

        PolsMultiply(F, F); { (2c) }
        PolWeightedAdd(F, 1.0, F, - MarkovParameter, A);
        PolRemove(F, 1); {Highest coeff should be zero}
        END

```

```
  END { PolMarkovRecursion } ;

PROCEDURE PolDerivativeEmulator
    (VAR E, F: Polynomial;
     P, C, A: Polynomial);

  VAR
    i: INTEGER;
    hk: REAL;
    Ek, Fk: Polynomial;

  BEGIN {PolDerivativeEmulator}
    IF A.Deg = 0 THEN
      BEGIN
      PolZero(F, - 1);
      PolMultiply(E, P, C);
      END
    ELSE
      BEGIN
      PolZero(Ek, - 1);
      PolEquate(Fk, C);

      PolEquate(E, Ek);
      PolEquate(F, Fk);
      PolScalarMultiply(F, P.Coeff[P.Deg], F);

      WITH P DO
        FOR i := 1 TO Deg DO
          BEGIN
          PolMarkovRecursion(hk, Ek, Fk, A);
          PolWeightedAdd(E, 1.0, E, Coeff[Deg - i], Ek);
          PolWeightedAdd(F, 1.0, F, Coeff[Deg - i], Fk);
          END;
      END;

  END {PolDerivativeEmulator} ;

PROCEDURE PolDivide(VAR E, F: Polynomial;
                    B, A: Polynomial);

  VAR
    One: Polynomial;

{-- Computes quotient E and remainder F
            when B is divided by A:

    B = E*A + F
 --}

  BEGIN {PolDivide}
    PolUnity(One, 0);
    PolDerivativeEmulator(E, F, B, One, A);
  END {PolDivide} ;

PROCEDURE PolEuclid(VAR GCD, E, F: Polynomial;
                    A, B: Polynomial);

{-- Given a(s) and b(s), finds GCD of a(s) and b(s)
```

```
c   and solves for E and F in:
c
c      Ea + Fb = GCD
c
c   Small is a small positive number to
c   determine termination of Euclids algorithm
c
c --}

  VAR
    Quotient: ARRAY [1..MaxDegree] OF Polynomial;
    N: INTEGER;

  PROCEDURE FindGCD(AlphaIminus1 {A} ,
                    AlphaI {B} : Polynomial);

  {Finds the Greates Common Divisor of A and B, together
c    with the corresponding quotients E and F,
c    using Euclid's algorithm.
c    Note that the algorithm terminates when the largest
c    absolute value of the coefficients of the remainder
c    is < Small;ideally, the remainder should be exactly
c    zero}

    VAR
      i: INTEGER;
      Remainder: Polynomial;

    BEGIN {FindGCD}
      i := 0;

      REPEAT
        i := i + 1;

        PolDivide(Quotient[i], Remainder, AlphaIminus1,
                  AlphaI); {I-2.4.5}

        IF NOT (PolNorm(Remainder) <= Small) THEN
          BEGIN
          PolEquate(AlphaIminus1, AlphaI);
          PolEquate(AlphaI, Remainder); {I-2.4.6}
          END;

      UNTIL PolNorm(Remainder) <= Small;

      PolEquate(GCD, AlphaI);
      N := i - 1;

    END {FindGCD} ;

  PROCEDURE DeduceEandF(VAR Beta {F} ,
                        Gamma {E} : Polynomial);

    VAR
      i: INTEGER;
      BetaQ, OldBeta: Polynomial;

    BEGIN {DeduceEandF}
```

```
  {--
c       Multiply quotients by -1
c --}
        FOR i := 1 TO N DO
          PolScalarMultiply(Quotient[i], - 1.0,
                            Quotient[i]);

        IF N < 1 THEN
          BEGIN
          PolZero(Beta, - 1);
          PolUnity(Gamma, 0);
          Gamma.Coeff[0] := 1.0 / A.Coeff[0];
          END
        ELSE
          BEGIN
          PolEquate(Beta, Quotient[N]); {beta n = -q n}
          PolUnity(Gamma, 0); {gamma n = 1}
          END;

        FOR i := N - 1 DOWNTO 1 DO
          BEGIN {I-2.4.12}
          PolEquate(OldBeta, Beta);
          PolMultiply(BetaQ, Beta, Quotient[i]);
          PolAdd(Beta, Gamma, BetaQ);
          PolEquate(Gamma, OldBeta);
          END;

      END {DeduceEandF} ;

    PROCEDURE NormaliseGCD;

      VAR
        GCD0: REAL;

      BEGIN {NormaliseGCD}
        GCD0 := GCD.Coeff[0];
        PolScalarMultiply(E, 1 / GCD0, E);
        PolScalarMultiply(F, 1 / GCD0, F);
        PolScalarMultiply(GCD, 1 / GCD0, GCD);
      END {NormaliseGCD} ;

    BEGIN {PolEuclid}

      FindGCD(A, B);
      DeduceEandF(F, E);

  {--    Tidy up
c --}
      PolTruncate(E);
      PolTruncate(F);
      NormaliseGCD;

    END {PolEuclid} ;

  PROCEDURE PolDioRecursion(VAR E, F: Polynomial;
                            A, B: Polynomial);

  {-- Diophantine recursion algorithm.
```

```

  Given E and F solving
     EA + FB = s^k
  this algorithm finds new values of E and F solving
     EA + FB = s^(k+1)

  The solution is such that F/A is strictly proper
--}

  VAR
    MarkovParameter: REAL;

  BEGIN { PolDioRecursion }

    IF F.Deg < A.Deg - 1 THEN
      BEGIN
      MarkovParameter := 0.0;
      PolsMultiply(E, E);
      PolsMultiply(F, F);
      END
    ELSE
      BEGIN
      MarkovParameter := F.Coeff[0] / A.Coeff[0];

      PolsMultiply(E, E);
      PolWeightedAdd(E, 1.0, E, MarkovParameter, B);

      PolsMultiply(F, F);
      PolWeightedAdd(F, 1.0, F, - MarkovParameter, A);
      PolTruncate(F);
      END;

    PolTruncate(E); {Just in case the order is less than
                     usual}

  END { PolDioRecursion } ;

PROCEDURE PolDiophantine(VAR E, F: Polynomial;
                         VAR GCD: Polynomial;
                         A, B, PC: Polynomial);

{-- This procedure gives the controller
  polynomials E and F
  for pole placement:

     PC = EA + FB

  A is the open-loop system denominator polynomial
  B is the open-loop system numerator polynomial
  PC is the closed-loop system denominator polynomial

     Observer :  phi* = E/C u + F/C y

     Control law: phi* = w
--}

  VAR
    Ek, Fk, E0, F0: Polynomial;
```

```
    i: INTEGER;

  BEGIN {PolDiophantine}

    PolEuclid(GCD, E0, F0, A, B);
    PolEquate(E, E0);
    PolEquate(F, F0);

    WITH PC DO
      BEGIN
      PolEquate(Ek, E);
      PolEquate(Fk, F);
      PolScalarMultiply(E, Coeff[Deg], E);
      PolScalarMultiply(F, Coeff[Deg], F);

      FOR i := 1 TO Deg DO
        BEGIN
        PolDioRecursion(Ek, Fk, A, B);
        PolWeightedAdd(E, 1.0, E, Coeff[Deg - i], Ek);
        PolWeightedAdd(F, 1.0, F, Coeff[Deg - i], Fk);
        END;
      END {WITH PC} ;

  END {PolDiophantine} ;

PROCEDURE PolZeroCancellingEmulator
    (VAR E, F: Polynomial;
     P, C, A: Polynomial;
     ZMinus, ZPlus: Polynomial;
     VAR GCDAZ: Polynomial {GCD of A and Z-} );

{-- Given open loop system
              y = B(s)/A(s) u + C(s)/A(s) v
    finds emulator polynomials for
                     phi = P(s)/(Zplus.Zminus) y
--}

  VAR
    AZPlus, PC: Polynomial;

  BEGIN {PolZeroCancellingEmulator}
    PolMultiply(AZPlus, A, ZPlus);
    PolMultiply(PC, P, C);
    PolDiophantine(E, F, GCDAZ, AZPlus, ZMinus, PC);
  END {PolZeroCancellingEmulator} ;

PROCEDURE PolPade(VAR Pade: Polynomial;
                  Deg: INTEGER;
                  Delay: REAL);

{--   Pade is nth degree Denominator of Pade
      approximation to delay
--}

  VAR
    i: INTEGER;

  BEGIN {PolPade}
```

```
    Pade.Deg := Deg;

    WITH Pade DO
      BEGIN
      Coeff[Deg] := 1.0;
      FOR i := 1 TO Deg DO
        Coeff[Deg - i] := Delay / i * (Deg - i + 1) / (2 *
                         Deg - i + 1) * Coeff[Deg - i +
                         1];
      END;

  END {PolPade} ;

{----------------------------------------------}
{--    Emulator design for self-tuning        --}
{----------------------------------------------}

PROCEDURE SetDesignKnobs(VAR DesignKnobs: TypeDesignKnobs;
                        A, B: Polynomial;
                        ZeroAtOrigin, ZHasFactorB,
                         ContinuousTime: BOOLEAN);

  VAR
    Gain: REAL;

  PROCEDURE DesignP(A, B: Polynomial);

    VAR
      AA, BB, PP: Polynomial;

    BEGIN {DesignP}
      WITH DesignKnobs DO
        BEGIN
        IF ZeroAtOrigin THEN
          BEGIN
          PolsDivide(A, A);
          PolsDivide(B, B);
          END;

        PolSquare(AA, A);
        PolSquare(BB, B);
        PolWeightedAdd(PP, 1.0, BB, LQWeight, AA);
        PolSqrt(P, PP);

        PolScalarMultiply(P, 1 / P.Coeff[P.Deg], P);
        END;
    END {DesignP} ;

  BEGIN {SetDesignKnobs}
    WITH DesignKnobs DO
      BEGIN

      IF NOT ZHasFactorB THEN
        BEGIN {Set B+=B; B- = 1}
        PolUnity(BMinus, 0);
        PolEquate(BPlus, B);
        END
      ELSE {Set B- = B without s term; B+ = rest}
```

```
        BEGIN
        PolUnity(BPlus, 0);
        IF ZeroAtOrigin THEN
          BEGIN
          PolsDivide(BMinus, B);
          PolsMultiply(BPlus, BPlus);
          END
        ELSE
          BEGIN
          PolEquate(BMinus, B);
          END;

        { Normalise B-, and adjust B+ }
        Gain := PolGain(BMinus, ContinuousTime);
        PolScalarMultiply(BPlus, Gain, BPlus);
        PolScalarMultiply(BMinus, 1.0 / Gain, BMinus);
        END;

      PolMultiply(ZMinus, BMinus, ZMinusPlus);
      PolMultiply(Z, ZMinus, ZPlus);

      IF LQ THEN DesignP(A, B);

      END {WITH DesignKnobs} ;
    END {SetDesignKnobs} ;

  PROCEDURE PolInitialConditions
      (VAR InitialCondition, ED: Polynomial;
       A, D, E: Polynomial;
       DesignKnobs: TypeDesignKnobs);

{--    Computes the initial condition for an emulator
       given the initial condition D of the system B/A.

       ED is the unrealisable part of the
       initial condition
--}

    VAR
      Rem, AZPlusPlus, ZPlusPlus, FD, EDD, EDC,
      GCDAZ: Polynomial;

    BEGIN {PolInitialConditions}
      WITH DesignKnobs DO
        BEGIN
        PolMultiply(ZPlusPlus, ZPlus, ZMinusPlus);

        IF BMinus.Deg = 0 THEN
          BEGIN
          PolMultiply(AZPlusPlus, A, ZPlusPlus);
          PolDerivativeEmulator(ED, FD, P, D, AZPlusPlus);
          END
        ELSE
          PolZeroCancellingEmulator(ED, FD, P, D, A, BMinus,
                                    ZPlusPlus, GCDAZ);

        PolMultiply(EDD, E, D);
        PolMultiply(EDC, ED, C);
```

```
      PolWeightedAdd(InitialCondition, 1.0, EDD, - 1.0,
                 EDC);

      PolDivide(InitialCondition, Rem, InitialCondition,
              BMinus);
      PolTruncate(InitialCondition);
      END;
    END; {PolInitialConditions}

  PROCEDURE PolEmulator(VAR F, G, InitialCondition,
   {Emulator numerators}
                     FFilter, GFilter {Emulator
                               denominators}
                     : Polynomial;
                     VAR E, ED: {error} Polynomial;
                     VAR GCDAZ: Polynomial {GCD of A and
                      Z-} ;
                     A: Polynomial {System denominator} ;
                     D: Polynomial {Initial condition} ;
                     DesignKnobs: TypeDesignKnobs);

{-- Given open loop system
              y = B(s)/A(s) u + C(s)/A(s) v
    finds emulator polynomials for
                     phi = P(s)/(Zplus.Zminus) y
--}

    VAR
      Junk, AZPlus: Polynomial;

    PROCEDURE FindEandF;

      BEGIN {FindEandF}
        WITH DesignKnobs DO
          IF ZMinus.Deg = 0 THEN
            BEGIN
            PolMultiply(AZPlus, A, ZPlus);
            PolDerivativeEmulator(E, F, P, C, AZPlus);
            END
          ELSE
            BEGIN
            PolZeroCancellingEmulator(E, F, P, C, A, ZMinus,
                                ZPlus, GCDAZ);
            END;
      END {FindEandF} ;

    BEGIN {PolEmulator}
      WITH DesignKnobs DO
        BEGIN
        FindEandF;

        IF GCDAZ.Deg > 0 THEN {Move factor from Z- to Z+,
                          and try again}
          BEGIN
          PolMultiply(ZPlus, GCDAZ, ZPlus);
          PolDivide(ZMinus, Junk, GCDAZ, ZMinus);
          FindEandF;
          END;
```

```

{--
        Compute I = (E.D - ED.C)/ Z-
--}
      PolInitialConditions(InitialCondition, ED, A, D, E,
                           DesignKnobs);

{--
        Derive the remaining emulator polynomials
--}
      PolMultiply(G, E, BPlus);
      PolMultiply(GFilter, C, ZMinusPlus);
      PolMultiply(FFilter, C, ZPlus);
      END {WITH DesignKnobs} ;
    END {PolEmulator} ;

  PROCEDURE PolDelayEmulator(VAR F, G, InitialCondition,
                             FFilter, GFilter: Polynomial;
                             VAR E, ED: Polynomial;
                             VAR GCDAZ, PadeDenominator,
                              PadeNumerator: Polynomial;
                             A, D: Polynomial;
                             DesignKnobs: TypeDesignKnobs;
                             Delay: REAL;
                             PadeOrder: INTEGER);

{--    Computes PadeOrder Pade approximation to e^-sT
       where T is the time delay.

       Multiplies P polynomial by PadeDenominator.
       Multiplies B and Z- polynomials by Pade Numerator

       Resultant system (A,B,C) has approximate time delay.
       Resultant reference model Z/P also has approximate
       delay.

       Then does pole/Zero placement on modified system
--}

    BEGIN {PolDelayEmulator}
      WITH DesignKnobs DO
        BEGIN
{--    Compute Pade numerator and denoninator
--}
       PolPade(PadeDenominator, PadeOrder, Delay);
       PolOfMinusS(PadeNumerator, PadeDenominator);

{--    Adjust design polynomials
       (N.B. DesignKnobs is passed by value)
--}
       PolMultiply(P, P, PadeDenominator);
       PolMultiply(ZMinus, ZMinus, PadeNumerator);

       PolEmulator(F, G, InitialCondition, FFilter,
                   GFilter, E, ED, GCDAZ, A, D,
                   DesignKnobs);

       PolMultiply(GFilter, GFilter, PadeDenominator);
```

```
      END {WITH DesignKnobs} ;
   END {PolDelayEmulator} ;

PROCEDURE DesignEmulator(VAR STCKnobs: TypeSTCKnobs;
                         VAR STCState: TypeSTCState);

{ Designs the emulator coefficients in terms of
  the system and design polynomials }

  VAR
   i: INTEGER;
   ZPlusZMinusPlus: Polynomial;

  BEGIN {DesignEmulator}
   WITH STCKnobs, STCState, SystemKnobs, DesignKnobs,
        EmKnobs, ErrorPolynomials DO
     BEGIN
     PolDelayEmulator(F, G, InitialCondition, FFilter,
                      GFilter, E, ED, GCDAZ,
                      PadeDenominator, PadeNumerator, A,
                      D, DesignKnobs, Delay, PadeOrder);

     PolMultiply(ZPlusZMinusPlus, ZPlus, ZMinusPlus);
     FOR i := 1 TO NumberInteractions DO
       BEGIN
       PolMultiply(GInteraction[i], E, BInteraction[i]);
       PolZero(EI[i], 0);
       END;
     END;
  END {DesignEmulator} ;

{-----------------------------------------------}
{-- Input output  procedures                  --}
{-----------------------------------------------}

PROCEDURE WriteTitle(Title: TypeTitle);

  BEGIN {WriteTitle}
   WriteLn(Pretty, Title, Pretty);
   WriteLn(OutLog, Pretty, Title, Pretty);
   IF NOT Eof(InLog) THEN ReadLn(InLog);
  END {WriteTitle} ;

PROCEDURE WriteLoopTitle(Loop, Loops: INTEGER);

  VAR
   LoopTitle: TypeTitle;
   i: INTEGER;

  BEGIN {WriteLoopTitle}
   IF Loops > 1 THEN
     BEGIN
     IF Loop = 1 THEN LoopTitle := 'LOOP 1          '
     ELSE
       BEGIN
       LoopTitle := 'LOOP 1        ';
       FOR i := 2 TO Loops DO
         LoopTitle[6] := Succ(LoopTitle[6]);
```

```
        END;
      WriteLn;
      WriteTitle(LoopTitle);
      END;
  END {WriteLoopTitle} ;

PROCEDURE Skip(VAR F: TEXT);

  VAR
    Ch: CHAR;

  BEGIN {Skip}
    WHILE (F^ = ' ') AND NOT Eoln(F) DO Read(F, Ch);
  END {Skip} ;

PROCEDURE GetSymbol(VAR F: TEXT;
                    VAR ChangeChar: CHAR);

  BEGIN
    ChangeChar := ' ';
    IF NOT Eof(F) THEN
      IF NOT Eoln(F) THEN Read(F, ChangeChar);
    IF ChangeChar <> ChangeSymbol THEN ChangeChar := ' ';
  END;

FUNCTION NameMatched(i: INTEGER;
                     VAR Name: TypeName): BOOLEAN;

  VAR
    Ch: CHAR;

  BEGIN {NameMatched}
    IF (i > LengthName) THEN NameMatched := TRUE
    ELSE
      BEGIN
      IF Eoln(InLog) THEN Ch := ' '
      ELSE Read(InLog, Ch);

      IF (Ch = ' ') AND (InLog^ IN ['-', '0'..'9']) THEN
        NameMatched := TRUE
      ELSE
        BEGIN
        IF NOT (Ch = Name[i]) THEN NameMatched := FALSE
        ELSE NameMatched := NameMatched(i + 1, Name);
        END;

      END;

  END {NameMatched} ;

FUNCTION NewValue(All: BOOLEAN;
                  VAR ChangeChar: CHAR): BOOLEAN;

  VAR
    NV: BOOLEAN;

  BEGIN {NewValue}
    NV := NOT Eoln(Input);
```

```
    IF NV THEN NewValue := NOT (Input^ = ' ')
    ELSE NewValue := FALSE;

    IF All THEN
      BEGIN
      IF NV THEN ChangeChar := ChangeSymbol
      ELSE ChangeChar := ' ';
      END;

  END {NewValue} ;

PROCEDURE GetComment(VAR F: TEXT;
                     VAR Comment: TypeComment);

  BEGIN {GetComment} ;
    Skip(F);
    WITH Comment DO
      BEGIN
      Length := 0;
      WHILE NOT Eoln(F) AND (Length < LengthComment) DO
        BEGIN
        Length := Length + 1;
        Read(F, Str[Length]);
        END;
      END;
  END {GetComment} ;

PROCEDURE PutComment(VAR F: TEXT;
                     Comment: TypeComment);

  VAR
    i: INTEGER;

  BEGIN {PutComment} ;
    Write(F, ' ');
    WITH Comment DO
      FOR i := 1 TO Length DO
        BEGIN
        Write(F, Str[i]);
        END;
  END {PutComment} ;

PROCEDURE EnterReal(VAR x: REAL;
                    Default: REAL;
                    Name: TypeName;
                    All: BOOLEAN);

  VAR
    ValueFromFile: BOOLEAN;
    Comment: TypeComment;
    ChangeChar: CHAR;

  BEGIN {EnterReal}

    x := Default;
    GetSymbol(InLog, ChangeChar);
    ValueFromFile := FALSE;
```

```
    Comment.Length := 0;

    IF NOT Eof(InLog) THEN
      BEGIN
      IF NameMatched(1, Name) THEN
        BEGIN
        Read(InLog, x);
        ValueFromFile := TRUE;
        GetComment(InLog, Comment);
        END;
      ReadLn(InLog);
      END;

    IF All OR (ChangeChar = ChangeSymbol) OR
       NOT ValueFromFile THEN
      BEGIN
      Write(Name, ' = ', x: fw: dp);
      PutComment(Output, Comment);
      WriteLn(Output, ' := ');
      IF NewValue(All, ChangeChar) THEN
        BEGIN
        Read(Input, x);
        IF NOT Eoln(Input) THEN
          GetComment(Input, Comment);
        END;
      ReadLn(Input);
      END;

    Write(OutLog, ChangeChar, Name, ' ', x: fw: dp, ' ');
    PutComment(OutLog, Comment);
    WriteLn(OutLog, ' ');
  END {EnterReal} ;

PROCEDURE EnterInteger(VAR x: INTEGER;
                       Default: INTEGER;
                       Name: TypeName;
                       All: BOOLEAN);

  VAR
    ValueFromFile: BOOLEAN;
    Comment: TypeComment;
    ChangeChar: CHAR;

  BEGIN {EnterInteger}
    x := Default;
    GetSymbol(InLog, ChangeChar);
    ValueFromFile := FALSE;
    Comment.Length := 0;

    IF NOT Eof(InLog) THEN
      BEGIN
      IF NameMatched(1, Name) THEN
        BEGIN
        Read(InLog, x);
        ValueFromFile := TRUE;
        GetComment(InLog, Comment);
        END;
      ReadLn(InLog);
```

```
      END;

    IF All OR (ChangeChar = ChangeSymbol) OR
       NOT ValueFromFile THEN
      BEGIN
      Write(Name, ' = ', x: fw);
      PutComment(Output, Comment);
      WriteLn(Output, ' := ');
      IF NewValue(All, ChangeChar) THEN
        BEGIN
        Read(Input, x);
        IF NOT Eoln(Input) THEN
          GetComment(Input, Comment);
        END;
      ReadLn(Input);
      END;

    Write(OutLog, ChangeChar, Name, ' ', x: fw, ' ');
    PutComment(OutLog, Comment);
    WriteLn(OutLog, '  ');
  END {EnterInteger} ;

PROCEDURE EnterBoolean(VAR x: BOOLEAN;
                       Default: BOOLEAN;
                       Name: TypeName;
                       All: BOOLEAN);

  VAR
    Ch: CHAR;
    ValueFromFile: BOOLEAN;
    Comment: TypeComment;
    ChangeChar: CHAR;

  PROCEDURE ReadBoolean(VAR F: TEXT);

    BEGIN {ReadBoolean}
      Ch := ' ';
      WHILE (Ch = ' ') AND NOT Eoln(F) DO Read(F, Ch);
      x := Ch IN ['T', 't'];
      WHILE NOT (F^ = ' ') AND NOT Eoln(F) DO
        Read(F, Ch);
    END {ReadBoolean} ;

  BEGIN {EnterBoolean}
    x := Default;
    GetSymbol(InLog, ChangeChar);
    ValueFromFile := FALSE;
    Comment.Length := 0;

    IF NOT Eof(InLog) THEN
      BEGIN
      IF NameMatched(1, Name) THEN
        BEGIN
        ReadBoolean(InLog);
        GetComment(InLog, Comment);
        ValueFromFile := TRUE;
        END;
      ReadLn(InLog);
```

```
      END;

    IF All OR (ChangeChar = ChangeSymbol) OR
       NOT ValueFromFile THEN
      BEGIN
      Write(Name, ' = ', x);
      PutComment(Output, Comment);
      WriteLn(Output, ' := ');
      IF NewValue(All, ChangeChar) THEN
        BEGIN
        ReadBoolean(Input);
        IF NOT Eoln(Input) THEN
          GetComment(Input, Comment);
        END;
      ReadLn(Input);
      END;

    IF x THEN Write(OutLog, ChangeChar, Name, ' TRUE ')
    ELSE Write(OutLog, ChangeChar, Name, ' FALSE ');
    PutComment(OutLog, Comment);
    WriteLn(OutLog, ' ');
  END {EnterBoolean} ;

PROCEDURE EnterPolynomial(VAR x: Polynomial;
                          Default: Polynomial;
                          Name: TypeName;
                          All: BOOLEAN);

  VAR
    i: INTEGER;
    Factor: Polynomial;
    AnotherFactor: BOOLEAN;
    ValueFromFile: BOOLEAN;
    Comment: TypeComment;
    ChangeChar: CHAR;

  PROCEDURE GetPolynomial(VAR F: TEXT;
                          VAR x: Polynomial;
                          VAR AnotherFactor: BOOLEAN);

    VAR
      Ch: CHAR;

    BEGIN {GetPolynomial}
      WITH x DO
        BEGIN
        Deg := - 1;
        Skip(F);
        WHILE NOT Eoln(F) AND
            (F^ IN ['0'..'9', '+', '-']) DO
          BEGIN
          Deg := Deg + 1;
          Read(F, Coeff[Deg]);
          Skip(F);
          END;
        END;

      AnotherFactor := F^ = MultiplySymbol;
```

```
      IF AnotherFactor THEN Read(F, Ch);

    END {GetPolynomial} ;

  BEGIN {EnterPolynomial}
    WITH x DO
      BEGIN
      PolEquate(x, Default);
      AnotherFactor := FALSE;
      ValueFromFile := FALSE;
      CommentLength := 0;
      GetSymbol(InLog, ChangeChar);

      IF NOT Eof(InLog) THEN
        BEGIN
        IF NameMatched(1, Name) THEN
          BEGIN
          GetPolynomial(InLog, x, AnotherFactor);
          ValueFromFile := TRUE;
          GetComment(InLog, Comment);
          END;
        ReadLn(InLog);
        END;

      IF All OR (ChangeChar = ChangeSymbol) OR
         NOT ValueFromFile THEN
        BEGIN
        Write(Name, ' = ');
        FOR i := 0 TO Deg DO Write(Coeff[i]: fw: dp);
        IF AnotherFactor THEN
          Write(Output, ' ', MultiplySymbol);
        PutComment(Output, Comment);
        WriteLn(' := ');
        IF NewValue(All, ChangeChar) THEN
          BEGIN
          GetPolynomial(Input, x, AnotherFactor);
          IF NOT Eoln(Input) THEN
            GetComment(Input, Comment);
          END;
        ReadLn(Input);
        END;

      Write(OutLog, ChangeChar, Name);
      FOR i := 0 TO Deg DO
        Write(OutLog, ' ', Coeff[i]: fw: dp);
      IF AnotherFactor THEN
        Write(OutLog, ' ', MultiplySymbol);
      PutComment(OutLog, Comment);
      WriteLn(OutLog);

      IF AnotherFactor THEN
        BEGIN
        WriteLn('Next factor ...');
        EnterPolynomial(Factor, Default, Name, All);
        PolMultiply(x, x, Factor);
        END;
      END;

```

```
  END {EnterPolynomial} ;

PROCEDURE WriteParameters(VAR LoopVAR: TypeLoopVAR);
{LoopVAR should really be passed by value, but it would
 take a lot of stack}

  VAR
   i: INTEGER;

  BEGIN {WriteParameters}
    WITH LoopVAR.STCState.SystemKnobs DO
      BEGIN
      WriteLn('-----------------------------------');
      WriteLn('       System polynomials');
      WriteLn('-----------------------------------');
      Write(Output, 'A        ');
      PolLineWrite(Output, A);
      Write(Output, 'B        ');
      PolLineWrite(Output, B);
      FOR i := 1 TO NumberInteractions DO
        BEGIN
        Write(Output, 'B[', i: 1, ']     ');
        PolLineWrite(Output, BInteraction[i]);
        END;
      Write(Output, 'D        ');
      PolLineWrite(Output, D);
      END;
  END {WriteParameters} ;

PROCEDURE WriteDesign(VAR LoopVAR: TypeLoopVAR);
{LoopVAR should really be passed by value, but it would
 take a lot of stack}

  VAR
   i: INTEGER;

  BEGIN {WriteDesign}
    WITH LoopVAR, STCState, EmKnobs, STCKnobs,
         DesignKnobs, SystemKnobs, ErrorPolynomials DO
      BEGIN
      WriteParameters(LoopVAR);
      WriteLn('-----------------------------------');
      WriteLn('       Design polynomials');
      WriteLn('-----------------------------------');
      Write(Output, 'B+       ');
      PolLineWrite(Output, BPlus);
      Write(Output, 'B-       ');
      PolLineWrite(Output, BMinus);
      Write(Output, 'C        ');
      PolLineWrite(Output, C);
      Write(Output, 'P        ');
      PolLineWrite(Output, P);
      Write(Output, 'Z+       ');
      PolLineWrite(Output, ZPlus);
      Write(Output, 'Z-       ');
      PolLineWrite(Output, ZMinus);
      Write(Output, 'Z-+      ');
      PolLineWrite(Output, ZMinusPlus);
```

```
      IF SystemKnobs.Delay > 0.0 THEN
        BEGIN
        Write(Output, 'Pade      ');
        PolLineWrite(Output, PadeDenominator);
        END;

      IF GCDAZ.Deg > 0 THEN
        BEGIN
        WriteLn('-----------------------------------');
        Write(Output, 'GCD of A and Z-');
        PolLineWrite(Output, GCDAZ);
        END;

      WriteLn('-----------------------------------');
      Write(Output, 'F         ');
      PolLineWrite(Output, F);
      Write(Output, 'F filter ');
      PolLineWrite(Output, FFilter);

      Write(Output, 'G         ');
      PolLineWrite(Output, G);
      Write(Output, 'G filter ');
      PolLineWrite(Output, GFilter);

      FOR i := 1 TO NumberInteractions DO
        BEGIN
        Write(Output, 'G[', i: 1, ']      ');
        PolLineWrite(Output, GInteraction[i]);
        END;

      Write(Output, 'I         ');
      PolLineWrite(Output, InitialCondition);

      Write(Output, 'E         ');
      PolLineWrite(Output, E);

      Write(Output, 'ED        ');
      PolLineWrite(Output, ED);

      WriteLn('-----------------------------------');

      END;
  END {WriteDesign} ;

{---------------------------------------------}
{-- Data filtering procedures               --}
{---------------------------------------------}

PROCEDURE StateVariableFilter
    (u {Signal to be filtered} : REAL;
     A: Polynomial;
     FilterKnobs: TypeFilterKnobs;
     VAR FilterState: TypeFilterState);

  PROCEDURE cStateVariableFilter
      (u {Signal to be filtered} : REAL;
       A: Polynomial;
       FilterKnobs: TypeFilterKnobs;
```

```
       VAR FilterState: TypeFilterState);

{--   A state variable filter:
      FilterState[i] := s^-i / a(s^-1) *u
--}

{--
    The input u is assumed to be a straight line between
    uOld and u within the interval.
    uOld is automatically updated but can be
    changed if required.
    For example it can be set equal to the current
    value of u if the input is constant within one
    sample
--}

     VAR
       k, Index: INTEGER;
       Sum, hk: REAL;
       Increment: StateVector;

     BEGIN { cStateVariableFilter }
       WITH FilterKnobs DO
         BEGIN
         IF ConstantBetweenSamples THEN
           FilterState.Old := u;

         IF A.Deg = 0 THEN
           FilterState.State[0] := u / A.Coeff[0]
         ELSE
           BEGIN
           FilterState.State[0] := 0.0;
           FOR Index := 0 TO A.Deg DO
             Increment[Index] := FilterState.State[Index];
           FOR k := 1 TO ApproximationOrder DO
             BEGIN
             Sum := 0.0;
             hk := SampleInterval / k;

             BEGIN {Matrix Multiplication}
             FOR Index := 1 TO A.Deg DO
               Sum := Sum - A.Coeff[Index] *
                      Increment[Index];

             FOR Index := A.Deg DOWNTO 2 DO
               Increment[Index] := hk * Increment[Index -
                                  1];

             Increment[0] := Sum / A.Coeff[0];
             END {Matrix Multiplication} ;

             IF k = 1 THEN
               Increment[0] := Increment[0] +
                               FilterState.Old /
                               A.Coeff[0];

             IF k = 2 THEN
               Increment[0] := Increment[0] + (u -
```

```
                          FilterState.Old) /
                          A.Coeff[0];

            Increment[1] := hk * Increment[0];

            FOR Index := 0 TO A.Deg DO
              FilterState.State[Index] := FilterState.
                                          State[Index] +
                                          Increment[Index]
                                          ;

            END;
          END;
        FilterState.Old := u;

        END {WITH FilterKnobs}
      END { of cStateVariableFilter } ;

    PROCEDURE dStateVariableFilter
        (u [Signal to be filtered} : REAL;
         A: Polynomial; {Now a discrete-time polynomial}
         FilterKnobs: TypeFilterKnobs;
         VAR FilterState: TypeFilterState);

{--     A state variable filter:
--}

      VAR
        Index: INTEGER;
        Sum: REAL;

      BEGIN { dStateVariableFilter }
        WITH FilterKnobs DO
          BEGIN

          FOR Index := A.Deg DOWNTO 1 DO
            FilterState.State[Index] := FilterState.State[
                                        Index - 1];

          Sum := u;
          FOR Index := 1 TO A.Deg DO
            Sum := Sum - A.Coeff[Index] *
                   FilterState.State[Index];

          FilterState.State[0] := Sum / A.Coeff[0];

          END {WITH FilterKnobs}
      END { of dStateVariableFilter } ;

    BEGIN { StateVariableFilter ]
      WITH FilterKnobs DO
        BEGIN
        IF ContinuousTime THEN
          cStateVariableFilter(u, A, FilterKnobs,
                               FilterState)
        ELSE
          dStateVariableFilter(u, A, FilterKnobs,
                               FilterState);
```

```
      END {WITH FilterKnobs}
    END { of StateVariableFilter } ;

  PROCEDURE FilterInitialise(VAR FilterState:
                             TypeFilterState;
                             ContinuousTime: BOOLEAN;
                             InitialValue: REAL);
   { Initialises the state of StateVariableFilter }

    VAR
     j: INTEGER;

    BEGIN { FilterInitialise}
      FOR j := 0 TO MaxState DO
        FilterState.State[j] := 0.0;

      IF ContinuousTime THEN
        FilterState.State[1] := InitialValue
      ELSE FilterState.State[0] := InitialValue;
    END { FilterInitialise} ;

  FUNCTION StateOutput(FilterState: TypeFilterState;
                       Numerator,
   Denominator: Polynomial): REAL;

    VAR
      i, RelativeDegree: INTEGER;
      Sum: REAL;
    { Computes a scalar system output from the system state }

    BEGIN {StateOutput}
      Sum := 0.0;

      RelativeDegree := Denominator.Deg - Numerator.Deg;

      IF RelativeDegree >= 0 THEN
        FOR i := 0 TO Numerator.Deg DO
          Sum := Sum + FilterState.State[i +
                 RelativeDegree] * Numerator.Coeff[i];

      StateOutput := Sum;
    END {StateOutput} ;

  FUNCTION Filter(u {Input to filter} : REAL;
                  Numerator, Denominator: Polynomial;
                  FilterKnobs: TypeFilterKnobs;
                  VAR FilterState: TypeFilterState): REAL;
   { Implements a continuous-time transfer function ]

    BEGIN { Filter }
      WITH FilterKnobs DO
        BEGIN
        StateVariableFilter(u, Denominator, FilterKnobs,
                            FilterState);
        Filter := StateOutput(FilterState, Numerator,
                              Denominator);
        END {FilterKnobs}
    END {of Filter} ;
```

```

FUNCTION Delayed(u: REAL;
                 Delay: INTEGER;
                 VAR State: TypeDelayState): REAL;
{ Implements a time delay with input u -
  Delay is measured in sample intervals }

  BEGIN {Delayed}
    WITH State DO
      BEGIN
      InPointer := (InPointer + 1) MOD (MaxDelay + 1);
      OutPointer := (InPointer - Delay + MaxDelay + 1) MOD
                    (MaxDelay + 1);
      Buffer[InPointer] := u;
      Delayed := Buffer[OutPointer];
      END;
  END { Delayed} ;

FUNCTION DelayFilter(u {Input to filter} : REAL;
                     Numerator, Denominator: Polynomial;
                     Delay: REAL;
                     FilterKnobs: TypeFilterKnobs;
                     VAR FilterState: TypeFilterState;
                     VAR DelFilterState: TypeDelayState):
 REAL;

{ Implements a time delay in series with a
  rational transfer function -
  the delay is on the input of the state filter so that
  a time varying delay is not correctly handled ; but
  the memory requirement is reduced}

  VAR
    uDelayed: REAL;

  BEGIN { DelayFilter }
    WITH FilterKnobs DO
      BEGIN
      uDelayed := Delayed(u, Round(Delay / FilterKnobs.
                                   SampleInterval),
                          DelFilterState);

      StateVariableFilter(uDelayed, Denominator,
                          FilterKnobs, FilterState);

      DelayFilter := StateOutput(FilterState, Numerator,
                                 Denominator);

      END {FilterKnobs}
  END {of DelayFilter} ;

PROCEDURE TimeDelayInitialise
    (VAR State: TypeDelayState;
     InitialValue: REAL);

  VAR
    j: INTEGER;

```

```
  BEGIN {TimeDelayInitialise}
    WITH State DO
      BEGIN
      FOR j := 0 TO MaxDelay DO
        Buffer[j] := InitialValue;
      InPointer := 0;
      END;
  END {TimeDelayInitialise} ;

FUNCTION TimeFor(Interval: INTEGER;
                 VAR Counter: INTEGER): BOOLEAN;

  BEGIN {TimeFor}
    TimeFor := Counter = 0;
    Counter := (Counter + 1) MOD Interval;
  END {TimeFor} ;

{-----------------------------------------------}
{--    CSTC initialisation procedures        --}
{-----------------------------------------------}

PROCEDURE SigGenInitialise(VAR SigGenKnobs:
                           TypeSigGenKnobs);

  BEGIN {SigGenInitialise}
    WITH SigGenKnobs DO
      BEGIN
      EnterReal(StepAmplitude, 0.0,
             'Step amplitude           ', All);
      EnterReal(SquareAmplitude, 0.0,
             'Square amplitude         ', All);
      EnterReal(CosAmplitude, 0.0,
             'Cos amplitude            ', All);
      EnterReal(Period, 10.0, 'Period                 ',
             All);
      END;
  END {SigGenInitialise} ;

PROCEDURE tSystemInitialise(STCKnobs: TypeSTCKnobs;
                            STCState: TypeSTCState;
                            VAR tSystemKnobs:
                            TypeSystemKnobs;
                            VAR tSystemState:
                            TypeSystemState;
                            ContinuousTime: BOOLEAN;
                            RunKnobs: TypeRunKnobs);

  VAR
    i: INTEGER;

  BEGIN {tSystemInitialise}
    WITH tSystemKnobs DO
      BEGIN

      PolEquate(A, STCState.SystemKnobs.A);
      PolEquate(B, STCState.SystemKnobs.B);
      NumberInteractions := STCState.SystemKnobs.
                            NumberInteractions;
```

```
FOR i := 1 TO NumberInteractions DO
  PolEquate(BInteraction[i],
       STCState.SystemKnobs.BInteraction[i]);

PolEquate(D, STCState.SystemKnobs.D);
Delay := STCState.SystemKnobs.Delay;
IF STCKnobs.IntegralAction THEN
  BEGIN
  PolsDivide(A, A);
  PolsDivide(B, B);
  FOR i := 1 TO NumberInteractions DO
    PolsDivide(BInteraction[i], BInteraction[i]);
  PolsDivide(D, D);
  END;

IF NOT RunKnobs.CorrectSystem THEN
  BEGIN
  WriteTitle('Actual system  ');
  EnterPolynomial(A, A, 'A (system denominator)   ',
           All);
  EnterPolynomial(B, B, 'B (system numerator)     ',
           All);
  IF RunKnobs.Loops > 1 THEN
    EnterInteger(NumberInteractions,
           RunKnobs.Loops - 1,
           'Number of interactions   ', All)
  ELSE NumberInteractions := 0;

  FOR i := 1 TO NumberInteractions DO
    EnterPolynomial(BInteraction[i],
             STCState.SystemKnobs.
             BInteraction[i],
             'Interaction polynomial   ',
             All);

  EnterPolynomial(D, STCState.SystemKnobs.D,
           'D (initial conditions)   ', All);

  EnterReal(Delay, 0.0, 'Time delay                ',
        All);
  EnterInteger(Lags, 0, 'Number of lags            ',
        All);
  IF Lags > 0 THEN
    BEGIN
    EnterReal(LagTimeConstant, 0.0,
         'Lag time constant        ', All);
    EnterBoolean(Interactive, FALSE,
           'Interactive lags         ', All);

    END;
  END;

IF NOT ContinuousTime THEN {Multiply by forward
                  shift}
  WITH tSystemKnobs DO PolsMultiply(B, B);

FilterInitialise(tSystemState.FilterState,
         ContinuousTime, 0.0);
```

```
    FilterInitialise(tSystemState.ICState,
                     ContinuousTime, 1.0 / A.Coeff[0]);
    TimeDelayInitialise(tSystemState.DelayState, 0.0);
    FOR i := 0 TO MaxLags DO
      tSystemState.LagState[i] := 0.0;
    END;
  END {tSystemInitialise} ;

PROCEDURE ModelKnobsInitialise
    (STCKnobs: TypeSTCKnobs;
    tSystemKnobs: TypeSystemKnobs;
    VAR ModelKnobs: TypeSystemKnobs;
    ContinuousTime: BOOLEAN);

  BEGIN {ModelKnobsInitialise}
    WITH STCKnobs, ModelKnobs, DesignKnobs DO
      BEGIN
      IF ZHasFactorB THEN
        BEGIN
        PolUnitGain(tSystemKnobs.B, ContinuousTime);
        PolMultiply(B, tSystemKnobs.B, ZMinusPlus)
        END
      ELSE PolEquate(B, ZMinusPlus);
      PolMultiply(B, B, ZPlus);
      PolEquate(A, P);
      END;

  END {ModelKnobsInitialise} ;

PROCEDURE ModelInitialise(STCKnobs: TypeSTCKnobs;
                          STCState: TypeSTCState;
                          tSystemKnobs: TypeSystemKnobs;
                          VAR ModelKnobs: TypeSystemKnobs;
                          VAR ModelState: TypeSystemState;
                          ContinuousTime: BOOLEAN);

  BEGIN {ModelInitialise}
    ModelKnobsInitialise(STCKnobs, tSystemKnobs,
                         ModelKnobs, ContinuousTime);
    WITH STCKnobs, ModelKnobs, DesignKnobs DO
      BEGIN
      PolEquate(D, STCState.ErrorPolynomials.ED);
      Delay := tSystemKnobs.Delay;
      FilterInitialise(ModelState.FilterState,
                       ContinuousTime, 0.0);
      FilterInitialise(ModelState.ICState, ContinuousTime,
                       1.0 / A.Coeff[0]);

      TimeDelayInitialise(ModelState.DelayState, 0.0);
      END;
  END {ModelInitialise} ;

PROCEDURE SystemInitialise(VAR STCKnobs: TypeSTCKnobs;
                           VAR STCState: TypeSTCState;
                           RunKnobs: TypeRunKnobs);

  VAR
    i: INTEGER;
```

```
  One, Zero: Polynomial;

BEGIN {SystemInitialise}
  PolZero(Zero, 0);
  PolUnity(One, 0);
  WITH STCKnobs, STCState.SystemKnobs DO
    BEGIN
    WriteTitle('Assumed system ');
    PolZero(A, 1);
    A.Coeff[0] := 1.0;
    EnterPolynomial(A, A, 'A (system denominator)    ',
                    All);
    PolZero(B, 0);
    B.Coeff[0] := 1.0;
    EnterPolynomial(B, B, 'B (system numerator)      ',
                    All);

    IF A.Coeff[0] <> 1.0 THEN
      BEGIN
      WriteLn('Normalising A and B so that a0 = 1');
      PolNormalise(A, B, 0);
      Write('A ');
      PolLineWrite(Output, A);
      Write('B ');
      PolLineWrite(Output, B);
      END;

    EnterInteger(NumberInteractions, RunKnobs.Loops - 1,
                 'Number of interactions    ', All);
    IF NumberInteractions > MaxNumberInteractions THEN
      NumberInteractions := MaxNumberInteractions;
    FOR i := 1 TO NumberInteractions DO
      EnterPolynomial(BInteraction[i], Zero,
                      'Interaction polynomial    ', All);
    PolZero(D, A.Deg - 1);
    EnterPolynomial(D, D, 'D (initial conditions)    ',
                    All);

    IF IntegralAction THEN
      WITH STCState.SystemKnobs DO
        BEGIN
        IF All THEN
          WriteLn('Augmenting A, B and D with s');
        PolsMultiply(A, A);
        PolsMultiply(B, B);
        FOR i := 1 TO NumberInteractions DO
          PolsMultiply(BInteraction[i],
                       BInteraction[i]);
        PolsMultiply(D, D);
        END;

    EnterReal(Delay, 0.0, 'Time delay                ',
              All);

    END;
  END {SystemInitialise} ;

PROCEDURE DesignInitialise(VAR STCKnobs: TypeSTCKnobs;
```

```
                        VAR STCState: TypeSTCState;
                        ContinuousTime: BOOLEAN);

VAR
  One, Zero: Polynomial;

BEGIN {DesignInitialise}
  PolZero(Zero, 0);
  PolUnity(One, 0);
  WITH STCKnobs, STCState.SystemKnobs, DesignKnobs,
      STCState.EmKnobs DO
    BEGIN
    WriteTitle('Emulator design');
    EnterBoolean(ZHasFactorB, FALSE,
              'Z has factor B           ', All);
    EnterPolynomial(ZMinusPlus, One,
              'Z-+ (Z- not including B) ', All);

    EnterPolynomial(ZPlus, One,
              'Z+ (nice model numerator)', All);
    WITH ZPlus DO
      IF ContinuousTime AND (Coeff[Deg] <> 1.0) THEN
        WriteLn('WARNING:  Z+ does not have unit gain ')
          ;

    EnterBoolean(LQ, FALSE, 'Linear-quadratic poles    ',
              All);

    IF LQ THEN
      EnterReal(LQWeight, 0,
              'Linear-quadratic weight  ', All)
    ELSE
      BEGIN
      PolUnity(P, 1);
      P.Coeff[0] := 1.0;
      EnterPolynomial(P, P, 'P (model denominator)     ',
              All);
      WITH P DO
        IF ContinuousTime AND (Coeff[Deg] <> 1.0) THEN
          WriteLn('WARNING: P does not have unit gain ')
            ;
      END;

    SetDesignKnobs(DesignKnobs, A, B, IntegralAction,
              ZHasFactorB, ContinuousTime);

    EnterPolynomial(C, P, 'C (emulator denominator) ',
              All);
    EnterInteger(PadeOrder, 0,
              'Pade approximation order ', All);
    IF (PadeOrder > 0) AND
       (STCState.SystemKnobs.Delay = 0) THEN
      BEGIN
      PadeOrder := 0;
      WriteLn(
'Setting Pade approximation order to zero as delay is zero'
          );
      END;
```

```

    EnterReal(Small, 0.0001,
            'Small positive number     ', All);

    END;
  END {DesignInitialise} ;

PROCEDURE InitFilterKnobs(VAR FilterKnobs: TypeFilterKnobs
                       );

  BEGIN {InitFilterKnobs}
   WITH FilterKnobs DO
    BEGIN
    WriteTitle('Filters          ');
    EnterReal(SampleInterval, 0.1,
            'Sample Interval           ', All);
    EnterInteger(ApproximationOrder, 5,
            'Approximation Order       ', All);
    EnterBoolean(ContinuousTime, TRUE,
            'Continuous-time?          ', All);
    END;
  END {InitFilterKnobs} ;

PROCEDURE STCInitialise(VAR LoopVAR: TypeLoopVAR;
                     FilterKnobs: TypeFilterKnobs;
                     RunKnobs: TypeRunKnobs);

  VAR
   One, Zero: Polynomial;

  PROCEDURE KnobsInitialise(FilterKnobs: TypeFilterKnobs;
                         RunKnobs: TypeRunKnobs;
                         VAR PutDataKnobs:
                         TypePutDataKnobs;
                         VAR STCKnobs: TypeSTCKnobs;
                         VAR STCState: TypeSTCState);

   PROCEDURE TunerInitialise
       (SampleInterval: REAL;
        VAR Knobs: TypeTunerKnobs;
        VAR State: TypeTunerState);

    VAR
     i: INTEGER;

    BEGIN {TunerInitialise}
     WITH Knobs DO
      BEGIN
      EnterReal(InitialVariance, 1E5,
              'Initial Variance        ', All);
      EnterReal(ForgetTime, 1000.0,
              'Forget time             ', All);
      ForgetFactor := 1 - (SampleInterval /
                   ForgetTime);

      EnterReal(DeadBand, 0.0,
              'Dead band               ', All);

```

```
          EnterBoolean(On, TRUE,
                       'Estimator on          ', All);

          EnterInteger(TuneInterval, 1,
                       'Tune interval         ', All);
          END;

        WITH State, Knobs DO
          BEGIN
          FOR i := 1 TO MaxParameters DO
            BEGIN
            TuningGain[i] := 0.0;
            Variance[i] := InitialVariance;
            END;

          FOR i := 1 TO MaxUFactor DO UFactor[i] := 0.0;

          EstimationError := 0.0;
          Sigma := 0.0;
          Sigma1 := 0.0;
          TuneCounter := 0;
          END;
      END {TunerInitialise} ;

    PROCEDURE TuneEmInitialise
        (SampleInterval: REAL;
         VAR TunerKnobs: TypeTunerKnobs;
         VAR TunerState: TypeTunerState);

      BEGIN {TuneEmInitialise}
        TunerInitialise(SampleInterval, TunerKnobs,
                        TunerState);
      END {TuneEmInitialise} ;

    PROCEDURE IdentifyInitialise
        (SampleInterval: REAL;
         VAR STCKnobs: TypeSTCKnobs;
         VAR STCState: TypeSTCState);

      VAR
        i: INTEGER;
        Cs: Polynomial;

      BEGIN {IdentifyInitialise}
        WITH STCKnobs, STCState DO
          BEGIN
          WriteTitle('Identification ');
          TunerInitialise(SampleInterval, IdentifyKnobs,
                          IdentState);

          WITH SysEmKnobs, STCState.SystemKnobs DO
            BEGIN
            EnterPolynomial(Cs, A,
                            'Cs (emulator denominator)',
                            All);
            IF Cs.Coeff[0] <> 1.0 THEN
              BEGIN
              WriteLn('Normalising Cs so that c0 = 1');
```

```
          PolScalarMultiply(Cs, 1.0 / Cs.Coeff[0],
                         Cs);
          Write('Cs ');
          PolLineWrite(Output, Cs);
          END;

        EnterBoolean(IdentifyingRational, TRUE,
                     'Identifying rational part',
                     All);
        EnterBoolean(IdentifyingDelay, FALSE,
                     'Identifying delay         ',
                     All);

        PolEquate(GFilter, Cs);
        PolEquate(FFilter, Cs);
        PolEquate(G, B);

        FOR i := 1 TO NumberInteractions DO
          PolEquate(GInteraction[i], BInteraction[i]);

        PolMinus(F, Cs, A);
        PolRemove(F, 1); {Highest coeff should be
                          zero}

        PolEquate(InitialCondition, D);
        END;
      END;
    END {IdentifyInitialise} ;

  PROCEDURE ControlInitialise
      (VAR ControlKnobs: TypeControlKnobs);

    BEGIN {ControlInitialise}
      WITH ControlKnobs DO
        BEGIN
        WriteTitle('Controller      ');

        EnterPolynomial(qNumerator, Zero,
                       'Q numerator               ',
                       All);
        EnterPolynomial(qDenominator, One,
                       'Q denominator             ',
                       All);
        EnterPolynomial(rNumerator, One,
                       'R numerator               ',
                       All);
        EnterPolynomial(rDenominator, One,
                       'R denominator             ',
                       All);
        END;
    END {ControlInitialise} ;

  PROCEDURE PutDatInitialise
      (VAR PutDataKnobs: TypePutDataKnobs);

    BEGIN {PutDatInitialise}
      WITH PutDataKnobs DO
        BEGIN
```

```
        EnterReal(Max, 1E5, 'Maximum control signal   ',
                All);
        EnterReal(Min, - 1E5,
                'Minimum control signal   ', All);
        EnterBoolean(Switched, FALSE,
                'Switched control signal  ', All);
        END;
      END {PutDatInitialise} ;

    BEGIN {KnobsInitialise}
      WITH STCKnobs, STCState DO
        BEGIN
        SystemInitialise(STCKnobs, STCState, RunKnobs);

        IF SelfTuning OR Auto THEN
          BEGIN
          DesignInitialise(STCKnobs, STCState,
                         FilterKnobs.ContinuousTime);
          DesignEmulator(STCKnobs, STCState);
          WriteDesign(LoopVAR);
          END;

        IF SelfTuning THEN
          WITH DesignKnobs DO
            BEGIN
            WriteTitle('STC type        ');
            EnterBoolean(Explicit, FALSE,
                      'Explicit self-tuning     ',
                      All);
            EnterBoolean(UsingLambda, TRUE,
                      'Using lambda filter      ',
                      All);
            EnterBoolean(IdentifyingSystem, FALSE,
                      'Identifying system       ',
                      All);
            END;

        IF SelfTuning OR IdentifyingSystem THEN
          EnterBoolean(TuningInitialConditions, FALSE,
                    'Tuning initial conditions', All);

        IF IdentifyingSystem THEN
          BEGIN
          IdentifyInitialise(FilterKnobs.SampleInterval,
                         STCKnobs, STCState);
          END;

        IF SelfTuning AND NOT Explicit THEN
          BEGIN
          WriteTitle('Tuner           ');
          TuneEmInitialise(FilterKnobs.SampleInterval,
                         TunerKnobs, TunerState);
          END;

        IF Auto THEN ControlInitialise(ControlKnobs);

        PutDatInitialise(PutDataKnobs);

```

```
      IF NOT Auto THEN
        BEGIN
        WITH ControlKnobs DO
          BEGIN
          PolEquate(qNumerator, One);
          PolEquate(qDenominator, One);
          PolEquate(rNumerator, One);
          PolEquate(rDenominator, One);
          END;
        END;
      END;
  END {KnobsInitialise} ;

PROCEDURE StateInitialise(STCKnobs: TypeSTCKnobs;
                          VAR STCState: TypeSTCState;
                          ContinuousTime: BOOLEAN);

  VAR
    i: INTEGER;

  PROCEDURE InitEmulator(EmKnobs: TypeEmKnobs;
                         VAR EmState: TypeEmState;
                         ContinuousTime: BOOLEAN);

    VAR
      i: INTEGER;

    BEGIN {InitEmulator}
      WITH EmState, EmKnobs DO
        BEGIN
        FilterInitialise(uState, ContinuousTime, 0.0);
        FilterInitialise(yState, ContinuousTime, 0.0);
        FilterInitialise(ICState, ContinuousTime,
                         1.0 / FFilter.Coeff[0]);

        FOR i := 1 TO MaxNumberInteractions DO
          BEGIN
          FilterInitialise(InterState[i],
                           ContinuousTime, 0.0);
          FilterInitialise(InterFState[i],
                           ContinuousTime, 0.0);
          END;
        END;
    END {InitEmulator} ;

  BEGIN {StateInitialise}
    WITH STCKnobs, STCState DO
      BEGIN
      Phi := 0.0;
      PhiHat := 0.0;

      IF IdentifyingSystem THEN
        InitEmulator(SysEmKnobs, SysEmState,
                     ContinuousTime);

      IF Auto THEN
        InitEmulator(EmKnobs, EmState, ContinuousTime);

```

```
        IF SelfTuning AND UsingLambda THEN
          InitEmulator(EmKnobs, LambdaEmState,
                       ContinuousTime);

        WITH STCKnobs, STCState, DesignKnobs,
            ControlKnobs DO
          BEGIN
          FilterInitialise(qState, ContinuousTime, 0.0);
          FilterInitialise(wState, ContinuousTime, 0.0);
          FilterInitialise(PhicState, ContinuousTime,
                           0.0);
          FilterInitialise(uLambdaState, ContinuousTime,
                           0.0);
          FilterInitialise(yLambdaState, ContinuousTime,
                           0.0);
          FOR i := 1 TO MaxNumberInteractions DO
            FilterInitialise(iLambdaState[i],
                             ContinuousTime, 0.0);
          TimeDelayInitialise(uDelayState, 0.0);
          TimeDelayInitialise(yDelayState, 0.0);
          TimeDelayInitialise(EmState.DelFiltState, 0.0);
          TimeDelayInitialise(LambdaEmState.DelFiltState,
                              0.0);
          TimeDelayInitialise(SysEmState.DelFiltState,
                              0.0);
          END;

        END;
      END {StateInitialise} ;

    BEGIN {STCInitialise}
      PolZero(Zero, 0);
      PolUnity(One, 0);

      WITH LoopVAR, STCKnobs DO
        BEGIN
        WriteTitle('Control action ');
        EnterBoolean(Auto, TRUE,
                     'Automatic controller mode', All);
        EnterBoolean(IntegralAction, TRUE,
                     'Integral action          ', All);
        KnobsInitialise(FilterKnobs, RunKnobs, PutDataKnobs,
                        STCKnobs, STCState);
        StateInitialise(STCKnobs, STCState,
                        FilterKnobs.ContinuousTime);
        END;
    END {STCInitialise} ;

  {---------------------------------------------}
  {--    System simulation procedures         --}
  {---------------------------------------------}

  FUNCTION SigGen(SigGenKnobs: TypeSigGenKnobs;
                  Time: REAL): REAL;

    VAR
      Sig, NormalisedTime: REAL;

```

```
  BEGIN {SigGen}
   WITH SigGenKnobs DO
    BEGIN
    Sig := StepAmplitude;
    NormalisedTime := Time / Period;

    IF ((NormalisedTime - Trunc(NormalisedTime)) <
      0.5) THEN
      Sig := Sig + SquareAmplitude
    ELSE Sig := Sig - SquareAmplitude;

    IF NOT (CosAmplitude = 0.0) THEN
      Sig := Sig + CosAmplitude * Sin(TwoPi * Time /
                                   Period);

    SigGen := Sig;
    END;
  END {SigGen} ;

FUNCTION System(u: REAL;
                Knobs: TypeSystemKnobs;
                FilterKnobs: TypeFilterKnobs;
                VAR State: TypeSystemState): REAL;

  VAR
   y, uD: REAL;

  BEGIN {System}
   WITH Knobs, FilterKnobs, State DO
    BEGIN
    uD := Delayed(u, Round(Delay / SampleInterval),
              DelayState);
    y := Filter(uD, B, A, FilterKnobs, FilterState);

    IF D.Deg >= 0 THEN
      y := y + Filter(0.0, D, A, FilterKnobs, ICState);

    System := y;
    END;
  END {System} ;

FUNCTION MultiLag(u: REAL;
                  Lags: INTEGER;
                  TimeConstant: REAL;
                  Interactive: BOOLEAN;
                  FilterKnobs: TypeFilterKnobs;
                  VAR State: TypeLagState): REAL;

  VAR
   i: INTEGER;

  BEGIN {MultiLag}
   State[0] := u;
   WITH FilterKnobs DO
    FOR i := 1 TO Lags DO
     IF Interactive THEN
       State[i] := State[i] + (State[i - 1] + State[i +
                  1] - 2 * State[i]) *
```

```
                    SampleInterval * Sqr(Lags) /
                    TimeConstant
        ELSE
          State[i] := State[i] + (State[i - 1] -
                    State[i]) * SampleInterval * Lags /
                    TimeConstant;

    MultiLag := State[Lags];

    IF Interactive THEN State[Lags + 1] := State[Lags];
  END {MultiLag} ;

{---------------------------------------------}
{--    Self-tuner input/output procedures     --}
{---------------------------------------------}

PROCEDURE GetData(VAR ThisLoopVAR: TypeLoopVAR;
                  VAR LoopVAR: LoopVARs;
                  VAR InData: TEXT;
                  VAR Time: REAL;
                  RunKnobs: TypeRunKnobs;
                  FilterKnobs: TypeFilterKnobs);

  PROCEDURE GetDataFromFile(VAR InData: TEXT);

    VAR
      i: INTEGER;

    BEGIN {GetDataFromFile}
      WITH ThisLoopVAR DO
        BEGIN
        Read(InData, Time, u, y);
        FOR i := 1 TO tSystemKnobs.NumberInteractions DO
          IF NOT Eoln(InData) THEN
            Read(InData, Interaction[i])
          ELSE Interaction[i] := 0.0;

        IF NOT Eoln(InData) THEN Read(InData, w)
        ELSE w := 0.0;

        ReadLn(InData);
        END;
    END {GetDataFromFile} ;

  PROCEDURE Simulate;

    VAR
      uD: REAL;
      j, Loop: INTEGER;

    BEGIN {Simulate}
      WITH ThisLoopVAR, RunKnobs DO
        BEGIN
        InDist := SigGen(InDisturbKnobs, Time);
        OutDist := SigGen(OutDisturbKnobs, Time);

        IF NOT Cascade OR (ThisLoop = 1) THEN uD := u
        ELSE uD := LoopVAR[ThisLoop - 1].y;
```

```

        uD := uD + InDist;

        WITH tSystemKnobs, tSystemState DO
          uD := MultiLag(uD, Lags, LagTimeConstant,
                         Interactive, FilterKnobs,
                         LagState);

        y := System(uD, tSystemKnobs, FilterKnobs,
                    tSystemState) + OutDist;

        j := 0;
        FOR Loop := 1 TO Loops DO
          IF NOT (Loop = ThisLoop) THEN
            WITH tSystemKnobs DO
              BEGIN
              j := j + 1;
              y := y + Filter(Interaction[j],
                              BInteraction[j], A,
                              FilterKnobs,
                              InteractionState[j + 1]);
              END;

        IF NOT Cascade OR (ThisLoop = Loops) THEN
          w := SigGen(SetPointKnobs, Time)
        ELSE w := LoopVAR[ThisLoop + 1].u;

        END;
    END {Simulate} ;

  BEGIN {GetData}
    IF RunKnobs.ExternalData THEN GetDataFromFile(InData)
    ELSE Simulate;
  END {GetData} ;

PROCEDURE PutData(VAR u: REAL;
                  PutDataKnobs: TypePutDataKnobs);

  BEGIN {PutData}
    WITH PutDataKnobs DO
      BEGIN
      IF u > Max THEN u := Max
      ELSE IF u < Min THEN u := Min;

      IF Switched THEN
        IF Abs(u - Min) < Abs(u - Max) THEN u := Min
        ELSE u := Max;
      END;
  END {PutData} ;

{---------------------------------------------}
{--   High-gain (emulator-free) control     --}
{---------------------------------------------}

PROCEDURE HighGainControl(VAR u: REAL;
                          w, y: REAL;
                          FilterKnobs: TypeFilterKnobs;
                          VAR STCKnobs: TypeSTCKnobs;
```

```
                        VAR STCState: TypeSTCState);

   BEGIN {HighGainControl}
     WITH STCKnobs, STCState, DesignKnobs, FilterKnobs,
          ControlKnobs DO
       BEGIN
       Phi := Filter(y, P, Z, FilterKnobs, PhicState);

       u := Filter(w - Phi, qDenominator, qNumerator,
                   FilterKnobs, qState);

       END;
   END {HighGainControl} ;

 {----------------------------------------------}
 {--    Self-tuning  control                  --}
 {----------------------------------------------}

 PROCEDURE SelfTuningControl(VAR u: REAL
  { The control signal } ;
                          w, y: REAL
                      { The setpoint and system output }
                          ;
                          Interaction: TypeInteraction
                          {Interaction terms} ;
                          FilterKnobs: TypeFilterKnobs
                       { The digital filter parameters }
                          ;
                          ExternalData: BOOLEAN;
                          VAR PutDataKnobs:
                          TypePutDataKnobs
                     { The control signal limits etc.  }
                          ;
                          VAR STCKnobs: TypeSTCKnobs
                      { The user defined STC variables }
                          ;
                          VAR STCState: TypeSTCState
                       { The internal state of the STC }
                          );

   VAR
     One: Polynomial; { A unit polynomial of zero order }

 { The continuous-time self-tuning controller implementing
   many possible algorithms }

     {----------------------------------------------}
     {--    Emulator-based control procedures     --}
     {----------------------------------------------}

   FUNCTION Emulator(y, u: REAL;
                     Interaction: TypeInteraction;
                     NumberInteractions: INTEGER;
                     F, FFilter, G, GFilter,
                      InitialCondition: Polynomial;
                     GInteraction: InterPolynomial;
                     InputDelay: REAL;
                     FilterKnobs: TypeFilterKnobs;
```

```
                VAR EmState: TypeEmState): REAL;

{ Implements a tuneable emulator including
initial condition and interaction terms }

    VAR
     i: INTEGER;
     Em: REAL;

    BEGIN {Emulator}
     WITH FilterKnobs, EmState DO
      BEGIN
      Em := 0.0;

    {--    System input component of emulator output --}
      IF InputDelay > 0.0 THEN
        Em := Em + DelayFilter(u, G, GFilter,
                        InputDelay, FilterKnobs,
                        uState, DelFiltState)
      ELSE
        Em := Em + Filter(u, G, GFilter, FilterKnobs,
                     uState);

    {--   System output component of emulator output --}
      Em := Em + Filter(y, F, FFilter, FilterKnobs,
                   yState);

 {--   Initial condition  component of emulator output --}
      Em := Em + Filter(0.0, InitialCondition, FFilter,
                   FilterKnobs, ICState);

    {--    Interaction  component of emulator output --}
      FOR i := 1 TO NumberInteractions DO
        Em := Em + Filter(Interaction[i],
                     GInteraction[i], FFilter,
                     FilterKnobs, InterState[i]);
      END;

     Emulator := Em;
    END { Emulator} ;

  FUNCTION Control(y, w: REAL;
              Interaction: TypeInteraction;
              STCKnobs: TypeSTCKnobs;
              VAR STCState: TypeSTCState;
              FilterKnobs: TypeFilterKnobs): REAL;

   FUNCTION ImplicitSolution
       (y, w: REAL;
        Interaction: TypeInteraction;
        Em0State, Em1State: TypeEmState;
        Q0State, Q1State: TypeFilterState;
        STCKnobs: TypeSTCKnobs;
        VAR STCState: TypeSTCState;
        FilterKnobs: TypeFilterKnobs): REAL;

    VAR
     PhiQ0Hat, PhiQ1Hat, u: REAL;
```

```

    BEGIN {ImplicitSolution}
      WITH STCKnobs, STCState, EmKnobs, DesignKnobs,
           SystemKnobs, FilterKnobs, ControlKnobs DO
        BEGIN
        PhiQ0Hat := Emulator(y, 0.0, Interaction,
                             NumberInteractions, F,
                             FFilter, G, GFilter,
                             InitialCondition,
                             GInteraction, 0.0,
                             FilterKnobs,
                    Em0State) + Filter(0.0, qNumerator,
                                       qDenominator,
                                       FilterKnobs,
                                       Q0State);

        PhiQ1Hat := Emulator(y, 1.0, Interaction,
                             NumberInteractions, F,
                             FFilter, G, GFilter,
                             InitialCondition,
                             GInteraction, 0.0,
                             FilterKnobs,
                    Em1State) + Filter(1.0, qNumerator,
                                       qDenominator,
                                       FilterKnobs,
                                       Q1State);

        u := (w - PhiQ0Hat) / (PhiQ1Hat - PhiQ0Hat);

        ImplicitSolution := u;

        END;
    END {ImplicitSolution} ;

  BEGIN {Control}
    WITH STCKnobs, ControlKnobs, STCState DO
      IF qNumerator.Deg = qDenominator.Deg THEN
        Control := ImplicitSolution(y, w, Interaction,
                                    EmState, EmState,
                                    qState, qState,
                                    STCKnobs, STCState,
                                    FilterKnobs)
      ELSE
        WITH FilterKnobs DO
          Control := Filter(w - PhiHat, qDenominator,
                            qNumerator, FilterKnobs,
                            qState);
  END {Control} ;

{-----------------------------------------------}
{--   Emulator tuning procedures              --}
{-----------------------------------------------}

PROCEDURE SetData(VAR DataVector: TypeDataVector;
                  State: TypeEmState;
                  Knobs: TypeEmKnobs;
                  TuningInitialConditions,
                   IntegralAction: BOOLEAN;
```

```
                 NumberInteractions: INTEGER);

  VAR
   i, k, j: INTEGER;
   Integrating: INTEGER; { Set to one if Integral
                        action, else zéro }

  BEGIN {SetData}
   IF IntegralAction THEN Integrating := 1
   ELSE Integrating := 0;

   j := 0;
   WITH State, Knobs, DataVector DO
    BEGIN
    IF TuningInitialConditions THEN
     WITH ICState DO
      FOR k := FFilter.Deg -
       InitialCondition.Deg TO FFilter.Deg DO
        BEGIN
        j := j + 1;
        Data[j] := State[k];
        END;

    WITH uState DO
     FOR k := GFilter.Deg - G.Deg TO GFilter.Deg -
      Integrating DO
       BEGIN
       j := j + 1;
       Data[j] := State[k];
       END;

    WITH yState DO
     FOR k := FFilter.Deg - F.Deg TO FFilter.Deg -
      Integrating DO
       BEGIN
       j := j + 1;
       Data[j] := State[k];
       END;

    FOR i := 1 TO NumberInteractions DO
     WITH InterState[i] DO
      FOR k := GFilter.Deg -
       GInteraction[i].Deg TO GFilter.Deg -
       Integrating DO
        BEGIN
        j := j + 1;
        Data[j] := State[k];
        END;

    NumberOfParameters := j;
    END;
  END {SetData} ;

PROCEDURE TuneEmulator(VAR Knobs: TypeEmKnobs;
                 State: TypeTunerState;
                 TuningInitialConditions,
                  IntegralAction: BOOLEAN;
                 NumberInteractions: INTEGER);
```

```
    VAR
      i, k, j: INTEGER;
      Integrating: INTEGER; { Set to one if Integral
                              action, else zero }

    BEGIN {TuneEmulator}
      IF IntegralAction THEN Integrating := 1
      ELSE Integrating := 0;
      j := 0;

      WITH Knobs, State DO
        BEGIN
        IF TuningInitialConditions THEN
          WITH InitialCondition DO
            FOR k := 0 TO Deg DO
              BEGIN
              j := j + 1;
              Coeff[k] := Coeff[k] - (TuningGain[j] /
                          Sigma1) * EstimationError;

              END;

        WITH G DO
          FOR k := 0 TO Deg - Integrating DO
            BEGIN
            j := j + 1;
            Coeff[k] := Coeff[k] - (TuningGain[j] /
                        Sigma1) * EstimationError;
            END;

        WITH F DO
          FOR k := 0 TO Deg - Integrating DO
            BEGIN
            j := j + 1;
            Coeff[k] := Coeff[k] - (TuningGain[j] /
                        Sigma1) * EstimationError;
            END;

        FOR i := 1 TO NumberInteractions DO
          WITH GInteraction[i] DO
            FOR k := 0 TO Deg - Integrating DO
              BEGIN
              j := j + 1;
              Coeff[k] := Coeff[k] - (TuningGain[j] /
                          Sigma1) * EstimationError;
              END;

        END;
      END {TuneEmulator} ;

    PROCEDURE UpdateLeastSquaresGain
        (VAR TunerState: TypeTunerState;
         TunerKnobs: TypeTunerKnobs;
         DataVector: TypeDataVector);

      VAR
        j, UIndex1, UIndex2: INTEGER;
```

```
     fj, bj, OldSigma1, Lambda: REAL;
     i: INTEGER;
     UFac: REAL;

   FUNCTION UTX(j: INTEGER): REAL; { computes jth element
                                   of UT * X }

     VAR
      i: INTEGER;
      Sum, UFac: REAL;

     BEGIN { UTX }

      Sum := 0.0;

      WITH TunerState, TunerKnobs, DataVector DO
       BEGIN
       FOR i := 1 TO j DO
         BEGIN

         IF i = j THEN { use unit Diagonal term }
           UFac := 1.0
         ELSE
           BEGIN
           UIndex1 := UIndex1 + 1;
           UFac := UFactor[UIndex1]
           END;

         Sum := Sum + Data[i] * UFac;

         END;

       UTX := Sum
       END

   END;
  { of UTX }

  BEGIN { UpdateLeastSquaresGain }

   WITH TunerState, TunerKnobs, DataVector DO
    BEGIN

    UIndex1 := 0;
    UIndex2 := 0;
    Sigma1 := 1.0;

    FOR j := 1 TO NumberOfParameters DO
      BEGIN
      OldSigma1 := Sigma1;
      fj := UTX(j);

      bj := Variance[j] * fj;
      fj := fj / ForgetFactor;
      Sigma1 := OldSigma1 + fj * bj;
      Variance[j] := OldSigma1 / Sigma1 *
                  Variance[j] / ForgetFactor;
      TuningGain[j] := bj; { jth element of normalised
```

```
                         Kalman TuningGain }

        Lambda := - fj / OldSigma1;

{--     Update jth column of UFactor
        and 1-jth element of TuningGain
--}
        FOR i := 1 TO j - 1 DO
          BEGIN
          UIndex2 := UIndex2 + 1;
          UFac := UFactor[UIndex2];
          UFactor[UIndex2] := UFac + Lambda *
                              TuningGain[i];
          TuningGain[i] := TuningGain[i] + UFac * bj;
          END;
        END;

      { Set Sigma1 to value required in 'Tune' }
      Sigma1 := (Sigma1 - 1.0) * ForgetFactor +
              ForgetFactor;

      { Set Sigma to XT P X }
      Sigma := (Sigma1 - ForgetFactor) / Sigma1;

      END;

    END { of UpdateLeastSquaresGain } ;

  PROCEDURE IdentifySystem(y, u: REAL;
                        Interaction: TypeInteraction;
                        FilterKnobs: TypeFilterKnobs;
                        VAR STCKnobs: TypeSTCKnobs;
                        VAR STCState: TypeSTCState);

    VAR
      yHat: REAL;
      i: INTEGER;
      DataVector: TypeDataVector;

    PROCEDURE TuneDelay(VAR Delay: REAL;
                     State: TypeTunerState;
                     NumberOfParameters: INTEGER);

      BEGIN {TuneDelay}
        WITH State DO
          Delay := Delay -
                (TuningGain[NumberOfParameters] /
                Sigma1) * EstimationError;
        IF Delay < 0.0 THEN Delay := 0.0; {Negative
          delays are not allowed}
      END {TuneDelay} ;

    PROCEDURE SetDelayData(VAR DataVector: TypeDataVector;
                        State: TypeEmState;
                        Knobs: TypeEmKnobs);

      VAR
        sG: Polynomial;
```

```
    BEGIN {SetDelayData}

      WITH DataVector, Knobs, State DO
        BEGIN
        NumberOfParameters := NumberOfParameters + 1;
        PolsMultiply(sG, G);
        Data[NumberOfParameters] := - StateOutput(uState
                                   , sG, GFilter);
        END
    END {SetDelayData} ;

  BEGIN {IdentifySystem}

    WITH STCKnobs, STCState, SysEmState, SysEmKnobs,
        SystemKnobs, IdentState DO
      BEGIN

      yHat := Emulator(y, u, Interaction,
                  NumberInteractions, F, FFilter,
                  G, GFilter, InitialCondition,
                  GInteraction, SystemKnobs.Delay,
                  FilterKnobs, SysEmState);

      EstimationError := yHat - y;

      WITH IdentifyKnobs DO
        IF (Abs(EstimationError) >= DeadBand) AND On AND
          TimeFor(TuneInterval, TuneCounter) THEN
          BEGIN

          IF IdentifyingRational THEN
            SetData(DataVector, SysEmState, SysEmKnobs,
                    TuningInitialConditions,
                    IntegralAction,
                    SystemKnobs.NumberInteractions)
          ELSE DataVector.NumberOfParameters := 0;

          IF IdentifyingDelay THEN
            SetDelayData(DataVector, SysEmState,
                    SysEmKnobs);

          UpdateLeastSquaresGain(IdentState,
                            IdentifyKnobs,
                            DataVector);

          IF IdentifyingRational THEN
            TuneEmulator(SysEmKnobs, IdentState,
                    TuningInitialConditions,
                    IntegralAction,
                    SystemKnobs.NumberInteractions
                    );
          IF IdentifyingDelay THEN
            TuneDelay(SystemKnobs.Delay, IdentState,
                  DataVector.NumberOfParameters);

          END;
      END;
```

```
  WITH STCKnobs, STCState, SysEmKnobs, IdentState,
      STCState.SystemKnobs DO
    BEGIN
    PolMinus(A, FFilter, F);
    PolEquate(B, G);
    PolTruncate(B);
    PolEquate(D, InitialCondition);

    FOR i := 1 TO NumberInteractions DO
      PolEquate(BInteraction[i], GInteraction[i]);
    END;

 END {IdentifySystem} ;

PROCEDURE TunePhiEmulator(y: REAL;
                    FilterKnobs: TypeFilterKnobs;
                    VAR STCKnobs: TypeSTCKnobs;
                    VAR STCState: TypeSTCState);

 VAR
  DataVector: TypeDataVector;

 BEGIN {TunePhiEmulator}

  WITH STCKnobs, STCState, EmKnobs, EmState,
      TunerState, SystemKnobs DO
    BEGIN

    WITH DesignKnobs, FilterKnobs DO
      Phi := Filter(y, P, Z, FilterKnobs, PhicState);

    EstimationError := PhiHat - Phi;

    WITH TunerKnobs DO
      IF (Abs(EstimationError) >= DeadBand) AND On AND
        TimeFor(TuneInterval, TuneCounter) THEN
        BEGIN
        SetData(DataVector, EmState, EmKnobs,
                TuningInitialConditions,
                IntegralAction, NumberInteractions);

        UpdateLeastSquaresGain(TunerState, TunerKnobs,
                               DataVector);
        TuneEmulator(EmKnobs, TunerState,
                     TuningInitialConditions,
                     IntegralAction,
                     NumberInteractions);
        END;
    END;
 END {TunePhiEmulator} ;

PROCEDURE TuneLambdaEmulator
    (y, u: REAL;
     Interaction: TypeInteraction;
     FilterKnobs: TypeFilterKnobs;
     LambdaNumerator, LambdaDenominator: Polynomial;
     ZLambda, PLambda: Polynomial;
```

```
      VAR STCKnobs: TypeSTCKnobs;
      VAR STCState: TypeSTCState);

 VAR
  uLambda, yLambda, PhiLambda, PhiLamHat: REAL;
  InterLambda: TypeInteraction;
  DataVector: TypeDataVector;
  i: INTEGER;

 BEGIN {TuneLambdaEmulator}

  WITH STCKnobs, STCState, EmKnobs, LambdaEmState,
     TunerState, SystemKnobs DO
   BEGIN

   WITH FilterKnobs DO
    BEGIN
    PhiLambda := Filter(y, PLambda, ZLambda,
                 FilterKnobs, PhicState);

    uLambda := DelayFilter(u, LambdaNumerator,
                    LambdaDenominator, Delay,
                    FilterKnobs,
                    uLambdaState,
                    uDelayState);
    yLambda := DelayFilter(y, LambdaNumerator,
                    LambdaDenominator, Delay,
                    FilterKnobs,
                    yLambdaState,
                    yDelayState);

    FOR i := 1 TO SystemKnobs.NumberInteractions DO
     InterLambda[i] := Filter(Interaction[i],
                     LambdaNumerator,
                     LambdaDenominator,
                     FilterKnobs,
                     iLambdaState[i]);

    WITH DesignKnobs DO
     PhiLamHat := Emulator(yLambda, uLambda,
                    InterLambda,
                    NumberInteractions, F,
                    FFilter, G, GFilter,
                    InitialCondition,
                    GInteraction, 0.0,
                    FilterKnobs,
                    LambdaEmState);
   END;

  EstimationError := PhiLamHat - PhiLambda;

  WITH TunerKnobs DO
   IF (Abs(EstimationError) >= DeadBand) AND On AND
     TimeFor(TuneInterval, TuneCounter) THEN
    BEGIN
    SetData(DataVector, LambdaEmState, EmKnobs,
         TuningInitialConditions,
         IntegralAction, NumberInteractions);
```

```
          UpdateLeastSquaresGain(TunerState, TunerKnobs,
                                 DataVector);
          TuneEmulator(EmKnobs, TunerState,
                       TuningInitialConditions,
                       IntegralAction,
                       NumberInteractions);

        END;
      END;
    END {TuneLambdaEmulator} ;

{----------------------------------------------}
{--    Self-tuning  control: procedure body   --}
{----------------------------------------------}

BEGIN {SelfTuningControl}
  WITH STCKnobs, STCState, ControlKnobs DO
    BEGIN

    PolUnity(One, 0);

    IF NOT FilterKnobs.ContinuousTime THEN
      IF NOT ExternalData THEN
        BEGIN
        IF NOT Auto THEN u := w
        ELSE
          u := Control(y, w, Interaction, STCKnobs,
                       STCState, FilterKnobs);
        PutData(u, PutDataKnobs);
        END;

    IF IdentifyingSystem THEN
      BEGIN
      IdentifySystem(y, u, Interaction, FilterKnobs,
                     STCKnobs, STCState);
      IF SelfTuning THEN
        WITH STCState.SystemKnobs DO
          SetDesignKnobs(DesignKnobs, A, B,
                         IntegralAction, ZHasFactorB,
                         FilterKnobs.ContinuousTime);
      END;

    IF SelfTuning THEN
      BEGIN
      IF Explicit THEN
        DesignEmulator(STCKnobs, STCState)
      ELSE IF UsingLambda THEN
        WITH DesignKnobs DO
          TuneLambdaEmulator(y, u, Interaction,
                             FilterKnobs, Z, P, One,
                             One, STCKnobs, STCState)
      ELSE
        TunePhiEmulator(y, FilterKnobs, STCKnobs,
                        STCState);
      END;

    IF FilterKnobs.ContinuousTime THEN
      IF NOT ExternalData THEN
```

```
          BEGIN
          IF NOT Auto THEN u := w
          ELSE
            u := Control(y, w, Interaction, STCKnobs,
                         STCState, FilterKnobs);
          PutData(u, PutDataKnobs);
          END;

      IF SelfTuning OR Auto THEN
        WITH DesignKnobs, EmKnobs, SystemKnobs DO
          BEGIN
          PhiHat := Emulator(y, u, Interaction,
                             NumberInteractions, F,
                             FFilter, G, GFilter,
                             InitialCondition,
                             GInteraction, 0.0,
                             FilterKnobs, EmState);

          END;

      WITH ControlKnobs DO
        IF Auto AND
           (qNumerator.Deg = qDenominator.Deg) THEN {Update Q filter}
          WITH STCState DO
            StateVariableFilter(u, qDenominator,
                                FilterKnobs, qState);

      END;
  END {SelfTuningControl} ;

{-----------------------------------------------}
{--    Simulation initialisation             --}
{-----------------------------------------------}

PROCEDURE RunInitialise;

  VAR
    Loop, i: INTEGER;

  BEGIN {RunInitialise}
    WITH RunKnobs DO
      BEGIN
      WriteTitle('Data Source    ');
      EnterBoolean(ExternalData, FALSE,
                   'External data           ', All);

      EnterReal(LastTime, 10.0,
                'Last time               ', All);

      EnterInteger(PrintInterval, 1,
                   'Print interval          ', All);

      IF ExternalData THEN
        BEGIN
        WriteTitle('Real data      ');
        Reset(InData, 'indata.dat');
        END;
```

```

    FOR Loop := 1 TO Loops DO
      WITH LoopVAR[Loop] DO
        BEGIN
        ThisLoop := Loop;
        u := 0.0;
        y0 := 0.0;
        w := 0.0;
        InDist := 0.0;
        LoopInteraction[Loop] := 0.0;
        FOR i := 1 TO Loops DO Interaction[i] := 0.0;
        END;

    END;
  END {RunInitialise} ;

PROCEDURE SimulationInitialise
    (VAR ThisLoopVAR: TypeLoopVAR;
     FilterKnobs: TypeFilterKnobs;
     RunKnobs: TypeRunKnobs);

  BEGIN {SimulationInitialise}
    WITH ThisLoopVAR, STCKnobs, ControlKnobs, RunKnobs DO
      IF NOT ExternalData THEN
        BEGIN
        WriteTitle('Simulation      ');
        IF NOT Cascade OR (Loop = Loops) THEN
          BEGIN
          WriteTitle('Setpoint        ');
          SigGenInitialise(SetPointKnobs);
          END;
        WriteTitle('In Disturbance ');
        SigGenInitialise(InDisturbKnobs);
        WriteTitle('Out Disturbance');
        SigGenInitialise(OutDisturbKnobs);

        tSystemInitialise(STCKnobs, STCState,
                          tSystemKnobs, tSystemState,
                          FilterKnobs.ContinuousTime,
                          RunKnobs);
        IF Auto THEN
          ModelInitialise(STCKnobs, STCState,
                          tSystemKnobs, ModelKnobs,
                          ModelState,
                          FilterKnobs.ContinuousTime);

        END;
  END {SimulationInitialise} ;

{----------------------------------------------}
{--  Execution of the simulation and control  --}
{----------------------------------------------}

PROCEDURE Run;

  VAR
    Loop, OtherLoop, j: INTEGER;
    ReportCount: INTEGER;
```

```
  FirstTime, ReportInterval: REAL;

 PROCEDURE WriteData(VAR ThisLoopVAR: TypeLoopVAR);

  VAR
   i: INTEGER;

  BEGIN {WriteData}
   WITH ThisLoopVAR, STCState, STCKnobs, SystemKnobs DO
    BEGIN
    Write(OutData, Time: fw: dp, ' ', u: fw: dp, ' ',
         y: fw: dp, ' ');
    FOR i := 1 TO tSystemKnobs.NumberInteractions DO
      Write(OutData, Interaction[i]: fw: dp, ' ');
    Write(OutData, w: fw: dp, ' ', y0: fw: dp, ' ',
         PhiHat: fw: dp, ' ', Phi: fw: dp);

    IF IdentifyingSystem AND
       NOT UsingHighGainControl THEN
      WITH STCState.SystemKnobs, IdentState DO
       BEGIN
       Write(OutSysPar, Time: fw: dp, ' ',
            EstimationError: fw: dp, ' ', Sigma: fw:
            dp, ' ');
       Write(OutSysPar, Delay: fw: dp, ' ');

       PolWrite(OutSysPar, B);
       PolWrite(OutSysPar, A);
       FOR i := 1 TO NumberInteractions DO
         PolWrite(OutSysPar, BInteraction[i]);
       IF TuningInitialConditions THEN
         PolWrite(OutSysPar, D);
       END;

    IF SelfTuning AND NOT UsingHighGainControl THEN
      WITH EmKnobs, TunerState DO
       BEGIN
       Write(OutEmPar, Time: fw: dp, ' ',
            EstimationError: fw: dp, ' ', Sigma: fw:
            dp, ' ');
       PolWrite(OutEmPar, F);
       PolWrite(OutEmPar, G);
       FOR i := 1 TO NumberInteractions DO
         PolWrite(OutEmPar, GInteraction[i]);

       IF TuningInitialConditions THEN
         PolWrite(OutEmPar, InitialCondition);
       END;

    END;

  END {WriteData} ;

 PROCEDURE WriteLnData;

  BEGIN {WriteLnData}
   WriteLn(OutData);
   WriteLn(OutSysPar);
```

```
    WriteLn(OutEmPar);
  END {WriteLnData} ;

PROCEDURE OneTimeStep(VAR ThisLoopVAR: TypeLoopVAR);

  BEGIN {OneTimeStep}

    WITH ThisLoopVAR, RunKnobs, FilterKnobs DO
      BEGIN
      GetData(ThisLoopVAR, LoopVAR, InData, Time,
              RunKnobs, FilterKnobs);

      WITH STCKnobs.ControlKnobs, STCState DO
        wf := Filter(w, rNumerator, rDenominator,
                     FilterKnobs, wState);

      WITH STCKnobs DO
        IF Auto THEN
          y0 := System(wf, ModelKnobs, FilterKnobs,
                       ModelState);

      IF UsingHighGainControl THEN
        BEGIN
        HighGainControl(u, wf, y, FilterKnobs, STCKnobs,
                        STCState);
        PutData(u, PutDataKnobs);
        END
      ELSE
        SelfTuningControl(u, wf, y, Interaction,
                          FilterKnobs, ExternalData,
                          PutDataKnobs, STCKnobs,
                          STCState);

      IF PrintNow THEN WriteData(ThisLoopVAR);

      END;
  END {OneTimeStep} ;

PROCEDURE Splice(VAR ThisLoopVAR: TypeLoopVAR);

  VAR
    k, j: INTEGER;

  BEGIN {Splice}
    WITH ThisLoopVAR, STCKnobs DO
      IF TuningInitialConditions THEN
        WITH STCState, SysEmState, SysEmKnobs,
             IdentState DO
          BEGIN
          WriteLn('Splicing data');
          j := 0;
          FOR k := FFilter.Deg -
           InitialCondition.Deg TO FFilter.Deg DO
            BEGIN
            j := j + 1;
            Variance[j] := Variance[j] + IdentifyKnobs.
                           InitialVariance;
            END;
```

```

          FilterInitialise(ICState,
                           FilterKnobs.ContinuousTime,
                           1.0 / FFilter.Coeff[0]);
          END
        ELSE WriteLn('Not splicing data');
      END {Splice} ;

    FUNCTION NoMore: BOOLEAN;

      VAR
        More: BOOLEAN;
        Loop: INTEGER;

      PROCEDURE PreventBump;
      {Preserves continuity in system output
        when B changes -
        Initial conditions are ignored}

        VAR
          yNew: REAL;
          i: INTEGER;

        BEGIN {PreventBump}
          WITH LoopVAR[Loop], tSystemKnobs, tSystemState DO
            BEGIN
            yNew := StateOutput(FilterState, B, A);
            IF yNew <> 0.0 THEN
              WITH FilterState DO
                FOR i := 0 TO A.Deg DO
                  State[i] := State[i] * y / yNew;
            END;
        END {PreventBump} ;

      BEGIN {NoMore}
        WITH RunKnobs DO
          BEGIN
          IF ExternalData THEN More := NOT Eof(InData)
          ELSE More := TRUE;

          IF More THEN
            BEGIN
            WriteLn('Time now is ', Time: fw: dp);
            EnterBoolean(More, FALSE,
                         'More time                ', All);
            IF More THEN
              BEGIN
              EnterReal(ExtraTime, 10.0,
                        'Extra time               ', All);
              LastTime := LastTime + ExtraTime;

              FOR Loop := 1 TO Loops DO
                WITH LoopVAR[Loop], tSystemKnobs,
                     STCKnobs DO
                  BEGIN
                  WriteLoopTitle(Loop, Loops);
                  IF NOT ExternalData THEN
                    BEGIN
```

```
                WriteTitle('Actual system  ');
                EnterPolynomial(A, A,
                             'A (system denominator)   '
                              , All);
                EnterPolynomial(B, B,
                             'B (system numerator)     '
                              , All);
                EnterReal(Delay, 0.0,
                           'Time delay             ',
                           All);
                PreventBump;
                END;
              IF NOT Explicit THEN
                WITH TunerKnobs DO
                  EnterBoolean(On, TRUE,
                             'Estimator on            '
                              , All);
              IF IdentifyingSystem THEN
                WITH IdentifyKnobs DO
                  EnterBoolean(On, TRUE,
                             'System estimator on     '
                              , All);
              END;
          END;
        END;
      END;
    NoMore := NOT More;
  END { NoMore } ;

BEGIN {Run}

  Time := FilterKnobs.SampleInterval;
  PrintCounter := 0;

  WITH RunKnobs DO
    IF ExternalData THEN

      REPEAT
        WriteLn('Processing data in file ...');
        WHILE NOT Eof(InData) AND (Time < LastTime) DO
          BEGIN
          PrintNow := TimeFor(PrintInterval,
                              PrintCounter);
          IF Eoln(InData) THEN
            BEGIN
            Splice(LoopVAR[1]);
            ReadLn(InData);
            END
          ELSE OneTimeStep(LoopVAR[1]);

          IF PrintNow THEN WriteLnData;
          Time := Time + FilterKnobs.SampleInterval;
          END;
      UNTIL NoMore
    ELSE
      REPEAT
        ReportCount := 1;
        FirstTime := Time;
```

```
          ReportInterval := (LastTime - FirstTime) /
                             ProgressReports;
          FirstTime := FirstTime -
                       FilterKnobs.SampleInterval;
          WriteLn('Simulation running:');
          WHILE (Time < LastTime) DO
            BEGIN
            IF (Time - FirstTime) >=
               ReportCount * ReportInterval THEN
              BEGIN
              Write('     ');
              Write((100 * ReportCount) DIV
                    ProgressReports: 3);
              WriteLn('% complete');
              ReportCount := ReportCount + 1;
              END;

            PrintNow := TimeFor(PrintInterval,
                                PrintCounter);
            FOR Loop := 1 TO Loops DO
              OneTimeStep(LoopVAR[Loop]);

            FOR Loop := 1 TO Loops DO
              IF OutputCoupled THEN
                LoopInteraction[Loop] := LoopVAR[Loop].y
              ELSE
                LoopInteraction[Loop] := LoopVAR[Loop].u;

            FOR Loop := 1 TO Loops DO
              BEGIN
              j := 0;
              FOR OtherLoop := 1 TO Loops DO
                IF NOT (OtherLoop = Loop) THEN
                  BEGIN
                  j := j + 1;
                  LoopVAR[Loop].Interaction[j] :=
                   LoopInteraction[OtherLoop];
                  END;
              END;
            IF PrintNow THEN WriteLnData;
            Time := Time + FilterKnobs.SampleInterval;
            END;
        UNTIL NoMore;

    WriteLnData; {Once more for luck}

  END {Run} ;

{---------------------------------------------}
{--    Selection of appropriate chapter      --}
{---------------------------------------------}

FUNCTION Chapter(VAR All: BOOLEAN): INTEGER;

  VAR
    What: INTEGER;
    Ch: CHAR;

```

```
  BEGIN {Chapter}
    WriteLn;
    WriteLn(Pretty, 'C S T C ', Version, Pretty);
    WriteLn;

    WriteLn('Enter all variables (y/n, default n)?');
    IF Eoln(Input) THEN All := FALSE
    ELSE
      BEGIN
      Read(Input, Ch);
      All := Ch IN ['y', 'Y'];
      END;
    ReadLn(Input);

    EnterInteger(What, 1, 'Chapter                ',
                 All);
    WriteLn;
    Chapter := What;
  END {Chapter} ;

{----------------------------------------------}
{--    Body of CSTC                          --}
{----------------------------------------------}

BEGIN {CSTC}
  Reset(InLog, 'inlog.dat');
  Rewrite(OutLog, 'outlog.dat');

  Rewrite(OutData, 'outdata.dat');
  Rewrite(OutEmPar, 'outempar.dat');
  Rewrite(OutSysPar, 'outsyspar.dat');

  PolZero(Zero, 0);
  PolUnity(One, 0);

  WITH RunKnobs DO
    BEGIN
    Loops := 1;
    Cascade := FALSE;
    FilterKnobs.ConstantBetweenSamples := FALSE;

    CASE Chapter(All) OF
      1:
        WITH LoopVAR[1], STCKnobs, STCState DO
          BEGIN
          UsingHighGainControl := FALSE;
          tSystemKnobs.NumberInteractions := 0;
          SystemKnobs.NumberInteractions := 0;
          IdentifyingSystem := FALSE;
          CorrectSystem := TRUE;
          SelfTuning := FALSE;
          RunInitialise;
          InitFilterKnobs(FilterKnobs);
          STCInitialise(LoopVAR[1], FilterKnobs,
                        RunKnobs);
          SimulationInitialise(LoopVAR[1], FilterKnobs,
                               RunKnobs);
          EnterBoolean(FilterKnobs.ConstantBetweenSamples,
```

```
                    FALSE, 'Constant between samples ',
                    All);
        Run;
        END;
    2:
      WITH LoopVAR[1], STCKnobs, STCState DO
        BEGIN
        EnterBoolean(FilterKnobs.ContinuousTime, TRUE,
                    'Continuous-time?         ', All);
        EnterBoolean(STCKnobs.IntegralAction, TRUE,
                    'Integral action          ', All);
        tSystemKnobs.NumberInteractions := 0;
        SystemKnobs.NumberInteractions := 0;
        SystemInitialise(STCKnobs, STCState, RunKnobs);
        DesignInitialise(STCKnobs, STCState,
                         FilterKnobs.ContinuousTime);
        DesignEmulator(STCKnobs, STCState);
        WriteDesign(LoopVAR[1]);
        END;
    3:
      WITH LoopVAR[1], STCKnobs, STCState DO
        BEGIN
        UsingHighGainControl := FALSE;
        tSystemKnobs.NumberInteractions := 0;
        SystemKnobs.NumberInteractions := 0;
        IdentifyingSystem := FALSE;
        CorrectSystem := TRUE;
        SelfTuning := FALSE;
        RunInitialise;
        InitFilterKnobs(FilterKnobs);
        STCInitialise(LoopVAR[1], FilterKnobs,
                      RunKnobs);
        SimulationInitialise(LoopVAR[1], FilterKnobs,
                             RunKnobs);
        Run;
        END;
    4:
      WITH LoopVAR[1], STCKnobs, STCState DO
        BEGIN
        UsingHighGainControl := FALSE;
        tSystemKnobs.NumberInteractions := 0;
        SystemKnobs.NumberInteractions := 0;
        IdentifyingSystem := FALSE;
        CorrectSystem := FALSE;
        SelfTuning := FALSE;
        RunInitialise;
        InitFilterKnobs(FilterKnobs);
        STCInitialise(LoopVAR[1], FilterKnobs,
                      RunKnobs);
        SimulationInitialise(LoopVAR[1], FilterKnobs,
                             RunKnobs);
        Run;
        END;
    5:
      WITH LoopVAR[1], STCKnobs, STCState DO
        BEGIN
        UsingHighGainControl := FALSE;
        tSystemKnobs.NumberInteractions := 0;
```

```
        SystemKnobs.NumberInteractions := 0;
        Small := 0.000001;
        IdentifyingSystem := TRUE;
        CorrectSystem := FALSE;
        SelfTuning := FALSE;
        RunInitialise;
        InitFilterKnobs(FilterKnobs);
        STCInitialise(LoopVAR[1], FilterKnobs,
                      RunKnobs);
        SimulationInitialise(LoopVAR[1], FilterKnobs,
                             RunKnobs);
        Run;
        WriteParameters(LoopVAR[1]);
        END;

    6, 7:
      WITH LoopVAR[1], STCKnobs, STCState DO
        BEGIN
        UsingHighGainControl := FALSE;
        tSystemKnobs.NumberInteractions := 0;
        SystemKnobs.NumberInteractions := 0;
        CorrectSystem := FALSE;
        SelfTuning := TRUE;
        RunInitialise;
        InitFilterKnobs(FilterKnobs);
        STCInitialise(LoopVAR[1], FilterKnobs,
                      RunKnobs);
        SimulationInitialise(LoopVAR[1], FilterKnobs,
                             RunKnobs);
        Run;
        WriteDesign(LoopVAR[1]);
        END;

    8:
      WITH LoopVAR[1], STCKnobs, STCState DO
        BEGIN
        tSystemKnobs.NumberInteractions := 0;
        SystemKnobs.NumberInteractions := 0;
        CorrectSystem := FALSE;
        SelfTuning := TRUE;
        UsingHighGainControl := TRUE;
        RunInitialise;
        InitFilterKnobs(FilterKnobs);
        STCInitialise(LoopVAR[1], FilterKnobs,
                      RunKnobs);
        SimulationInitialise(LoopVAR[1], FilterKnobs,
                             RunKnobs);
        Run;
        END;

    9:
      BEGIN
      Cascade := TRUE;
      OutputCoupled := FALSE;
      EnterInteger(Loops, 2,
                'Number of loops           ', All);
      RunInitialise;
      InitFilterKnobs(FilterKnobs);
```

```
    FOR Loop := 1 TO Loops DO
      WITH LoopVAR[Loop], STCKnobs DO
        BEGIN
        WriteLoopTitle(Loop, Loops);
        CorrectSystem := FALSE;
        EnterBoolean(SelfTuning, TRUE,
                     'Self-tuning control      ',
                     All);
        EnterBoolean(UsingHighGainControl, FALSE,
                     'Using high-gain control  ',
                     All);
        STCInitialise(LoopVAR[Loop], FilterKnobs,
                      RunKnobs);
        SimulationInitialise(LoopVAR[Loop],
                             FilterKnobs, RunKnobs);
        END;

    Run;
    FOR Loop := 1 TO Loops DO
      BEGIN
      WriteLoopTitle(Loop, Loops);
      WriteDesign(LoopVAR[Loop]);
      END;
    END;

  10:
    BEGIN
    EnterInteger(Loops, 2,
                 'Number of loops          ', All);
    EnterBoolean(OutputCoupled, FALSE,
                 'Output coupled           ', All);

    RunInitialise;
    InitFilterKnobs(FilterKnobs);
    FOR Loop := 1 TO Loops DO
      WITH LoopVAR[Loop], STCKnobs DO
        BEGIN
        WriteLoopTitle(Loop, Loops);
        CorrectSystem := FALSE;
        EnterBoolean(SelfTuning, TRUE,
                     'Self-tuning control      ',
                     All);
        EnterBoolean(UsingHighGainControl, FALSE,
                     'Using high-gain control  ',
                     All);
        STCInitialise(LoopVAR[Loop], FilterKnobs,
                      RunKnobs);
        SimulationInitialise(LoopVAR[Loop],
                             FilterKnobs, RunKnobs);
        END;

    Run;

    FOR Loop := 1 TO Loops DO
      BEGIN
      WriteLoopTitle(Loop, Loops);
      WriteDesign(LoopVAR[Loop]);
      END;
```

```
        END;

      END {CASE} ;
    END {WITH RunKnobs} ;
END.
```

Variable Index

Procedure Index