ROBUST POLYNOMIAL CONTROLLER DESIGN

A Thesis submitted for the degree of Doctor of Philosophy

by

Kevin Wellstead

September 1991

Brunel University
Control Engineering Centre
Department of Electrical Engineering and Electronics
Uxbridge
Middlesex
UB8 3PH
U.K.

MEMORANDUM

Statement of Originality

The accompanying thesis is based on work carried out by the author at Brunel University between October 1988 and September 1991.

All work and ideas in this thesis are original, unless otherwise acknowledged in the text or by references. The work has not been submitted for another Degree in this university, nor for the award of a Degree or Diploma at any other Institution.

The main contribution of this work is the proposal of an alternative approach to the design of robust polynomial control systems. It utilises state space techniques by transforming the system to state space form, performing the design and transforming the resulting controller back to polynomial form.

During the period of this research two particular aspects have been reported in the technical literature. The references for these articles are

1) 'On Finding Polynomial Controllers with Reduced Pole Sensitivity using State Space Methods'
   Wellstead, K.D. and Daley, S.
   IEE International Conference, Control'91, Edinburgh, U.K., v1, pp 677-681
   IEE, London, U.K. (Conf Publ No. 332)

2) 'A Parametric Design Approach for Observer Based Fault Detection'
   Wang, H., Daley, S. and Wellstead, K.D.
   Eigth International Conference on Systems Engineering, Coventry, U.K., pp 248-255
   ISBN 0905949 10 2

ROBUST POLYNOMIAL CONTROLLER DESIGN

Kevin Wellstead

1991

Brunel University, Control Engineering Centre, Department of Electrical Engineering and Electronics, Uxbridge, Middlesex, UB8 3PH. U.K.

ABSTRACT

The work presented in this thesis was motivated by the desire to establish an alternative approach to the design of robust polynomial controllers. The procedure of pole-placement forms the basis of the design and for polynomial systems this generally involves the solution of a diophantine equation. This equation has many possible solutions which leads directly to the idea of determining the most appropriate solution for improved performance robustness.

A thorough review of many of the aspects of the diophantine equation is presented, which helps to gain an understanding of this extremely important equation. A basic investigation into selecting a more robust solution is carried out but it is shown that, in the polynomial framework, it is difficult to relate decisions in the design procedure to the effect on performance robustness. This leads to the approach of using a state space based design and transforming the resulting output feedback controller to polynomial form.

The state space design is centred around parametric output feedback which explicitly represents a set of possible feedback controllers in terms of arbitrary free parameters. The aim is then to select these free parameters such that the closed-loop system has improved performance robustness. Two parametric methods are considered and compared, one being well established and the other a recently proposed scheme. Although the well established method performs slightly better for general systems it is shown to fail when applied to this type of problem.

For performance robustness, the shape of the transient response in the presence of model uncertainty is of interest. It is well known that the eigenvalues and eigenvectors play an important role in determining the transient behaviour and as such the sensitivities of these factors to model uncertainty forms the basis on which the free parameters are selected. Numerical optimisation is used to select the free parameters such that the sensitivities are at a minimum.

It is shown both in a simple example and in a more realistic application that a significant improvement in the transient behaviour in the presence of model uncertainty can be achieved using the proposed design procedure.

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

To my Mother and Father

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Steve Daley, for all his help and encouragement over the three years of research culminating in this thesis. His support has been invaluable to the successful completion of this work.

Thanks must also go to John Marsh for his help with a number of theoretical aspects particularly in the field of linear algebra, and also to Tom Owens whose comments have helped tremendously both in carrying out the actual research and in the production of this thesis.

My colleague Hong Wang deserves thanks for a number of enlightening and fruitful discussions on many aspects of control systems design, and for his assistance in a number of areas I would also like to thank Bruce Baxter.

In research probably the most baffling problems are those associated with the equipment being used, particularly these days in the field of computing! Such problems can be frustrating and time consuming to deal with. To this end I am extremely grateful to John Philips for all his patience and help in solving the never ending list of problems I presented him with, and for doing so with good humour.

For their help and advice regarding many aspects of the university's computing facilities I would like to thank all those in the Computer Centre and in particular Raghbir Pank for his assistance with the NAG Fortran Library.

Dilip Dholiwar and the GEC Alsthom Engineering Research Centre also deserve thanks for supplying the data for the simulation of the hydraulic test rig.

This work was funded by the U.K. Science and Engineering Research Council.

CHAPTER 1

INTRODUCTION

1.1 Historical Background

The problem of designing accurate control systems in the presence of significant plant uncertainties is classical, Dorato (1987). This problem has been dealt with as far back as the 1920's when Black (1927) proposed using feedback with large loop gains to overcome the problem of significant variations in vacuum tube characteristics in the design of a vacuum tube amplifier. Dorato (1987) details the development of robust control theory from this early proposal and the classical work of Nyquist (1932) and Bode (1945) through to the late 1980's.

In the 1960's and 1970's much attention was focused on the state variable approach and in particular the linear quadratic Gaussian (LQG) method for optimal control. Kalman (1964) and Safonov and Athans (1977) showed that the optimal LQG state feedback control laws had some very strong robustness properties with infinite gain margins and 60-deg phase margins. However, in practice it is often necessary to employ Kalman filter theory to obtain an optimal estimate of the state vector which is then taken as an exact measurement in the LQG design. Doyle (1978) showed that when an estimate of the state vector is used, the design can exhibit arbitrarily poor stability margins and the robustness properties vanish.

LQG/LTR (linear quadratic Guassian/loop transfer recovery), Doyle and Stein (1979, 1981), provides a means of overcoming these problems by designing the Kalman filter such that the full state feedback properties are 'recovered'. One drawback is the inability of the method to deal with non-minimum phase systems as the procedure involves cancelling some of the filter poles with plant zeros.

Of particular importance to the shaping of robust methods today are three major discoveries in the late 1970's and early 1980's (Morari and Zafiriou, 1989). Youla *et al* (1976) showed that it is possible to parameterise all stabilising controllers for a particular system in a very effective manner, which guarantees that the resulting feedback controller automatically yields a closed-loop stable system. This effectively gives rise to a set of possible controllers which greatly simplifies the search for a more robust one. Zames (1981) postulated that measuring performance in terms of the ∞-norm rather than the traditional 2-norm might be closer to practical needs. This helped to establish the H_∞ optimal control approach to robust controller design. The work of Doyle in a number of papers in the early 1980's (Doyle and Stein, 1981; Doyle and Wall, 1982; Doyle, 1982) is quite important in the development of robust control theory. He argued that model uncertainty is often described very effectively in terms of norm-bounded perturbations.

He developed the structured singular value approach for testing 'robust stability' (i.e., stability in the presence of model uncertainty) and 'robust performance' (i.e., performance in the presence of model uncertainty), and is probably the primary motivation for the modern ∞-norm objective.

Other techniques for robust design include representing the model uncertainty stochastically as in Wonham (1967) and the game theoretic or minimax approach which basically represents the uncertainty as a factor that maximises a performance measure which is being minimised by the control variable (for example Ragade and Sarma, 1967; Bertsekas and Rhodes, 1973). The minimax approach can, however, become quite complicated for relatively simple design problems. It is also worth mentioning quantitative feedback theory (Horowitz, 1979, 1982; Horowitz and Sidi, 1980) which is based on loop gain shaping and the use of templates to represent the model uncertainty, each of which contains the set of possible plant transfer function values at a particular frequency. Other authors (for example Gourishankar and Ramar, 1976; Owens and O'Reilly, 1989) have suggested that the design be based on the sensitivities of the eigenvalues and eigenvectors. The conditioning of the matrix of eigenvectors has also been suggested as a good basis on which to design robust controllers (Kautsky *et al*, 1985; Byers and Nash, 1989). Further information on these methods and other alternative approaches to the robust control problem can be found in Dorato (1987), Maciejowski (1989) and Morari and Zafiriou (1989).

A discussion of a number of preliminary points regarding some basic definitions in the general robust control problem is presented next, followed by more specific information on the type of system being considered and the problem of interest. This naturally leads to a discussion of the objectives of this work and an outline of the thesis.

1.2 Preliminaries

Robust design attempts to take account of uncertainty in the model and disturbances on the system. Model uncertainty arises due to the difference between the real plant and the model being used for the design of the controller. When modelling a system it is often necessary to make certain assumptions such that the problem can be simply defined and a model easily obtained. Examples of such assumptions are linearity, the order of the model, the time delay, noise characteristics and the time invariance of parameters. The errors introduced by such assumptions can give rise to model uncertainty.

Model uncertainty can generally be split into two categories, unstructured and structured. To help understand the difference between the two, consider the typical feedback control system shown in figure 1.1 where P is the plant, C is the controller, $w(t)$ is the demand signal, $e(t)$ is the error, $u(t)$ is the input and $y(t)$ the output.

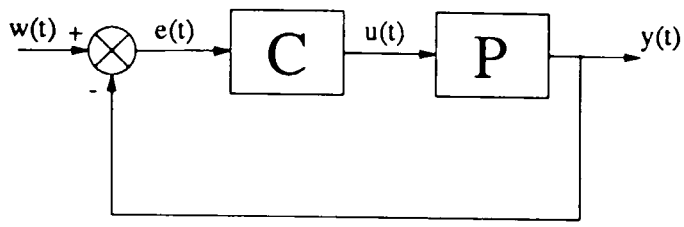


Figure 1.1 - Block Diagram of a Typical Feedback Control System

P can be expressed as

$$P = M + \Delta M \tag{1.1}$$

where M represents the derived model of the plant and ΔM the modelling error or uncertainty due to the violation of certain assumptions as outlined above.

An unstructured description of the model uncertainty essentially bounds the magnitude of possible perturbations, i.e.

$$\| \Delta M \| \leq \mu < +\infty \tag{1.2}$$

but does not trace the origins of the perturbations to specific elements of the plant. A structured description can be represented as

$$\Delta M = K\varepsilon \tag{1.3}$$

and attempts to specify some information, using K, regarding which elements of the plant are subject to perturbations. ε represents the unknown magnitude of the perturbations.

Clearly the unstructured approach may lead to controller designs which are unnecessarily conservative as it can include perturbations which do not actually occur in the plant. A structured approach on the other hand has the drawback that it does not deal with perturbations that affect the order of the plant (Maciejowski, 1989).

The robustness problem itself can primarily be split into two types, robust stability and robust performance. The stability problem is concerned with ensuring that the closed-loop system remains stable in the face of model uncertainty, whereas robust performance is concerned with how the closed-loop system behaves subject to model uncertainty.

1.3 Definition of the Problem

This work is concerned with discrete single-input single-output (SISO) systems in input-output (or polynomial) form. The open-loop system is as shown in figure 1.2

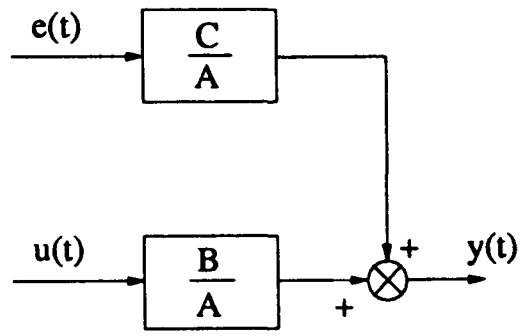


Figure 1.2 - Block Diagram of the Open-Loop System

which can be expressed as

$$A_p(z^{-1})y(t) = B_p(z^{-1})u(t) + C_p(z^{-1})e(t) \tag{1.4}$$

where

$$A_p(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{n_a} z^{-n_a} \tag{1.5}$$

$$B_p(z^{-1}) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{n_b} z^{-n_b} \tag{1.6}$$

$$C_p(z^{-1}) = 1 + c_1 z^{-1} + c_2 z^{-2} + \cdots + c_{n_c} z^{-n_c} \tag{1.7}$$

and z^{-1} can be interpreted as the backward shift operator. The signals $y(t)$, $u(t)$ and $e(t)$ are the sampled system output, the control input and a white noise sequence respectively. $C_p(z^{-1})$ is a colouring polynomial for the signal $e(t)$, used to characterise the disturbance more accurately.

For this type of system, the problem considered is that of performance robustness. To ensure that the problem remains tractable it is assumed that the orders of the system polynomials $A_p(z^{-1})$ and $B_p(z^{-1})$ are fairly accurate and that information is available on which coefficients are perturbed, thus the problem is one of structured model uncertainty. For the general polynomial system this can be expressed as

$$\hat{A}_p(z^{-1}) = 1 + (a_1 + \Delta a_1)z^{-1} + (a_2 + \Delta a_2)z^{-2} + \cdots + (a_{n_a} + \Delta a_{n_a})z^{-n_a} \tag{1.8}$$

$$\hat{B}_p(z^{-1}) = (b_0 + \Delta b_0) + (b_1 + \Delta b_1)z^{-1} + \cdots + (b_{n_b} + \Delta b_{n_b})z^{-n_b} \tag{1.9}$$

where $a_1, \cdots, a_{n_a}, b_0, \cdots, b_{n_b}$ are the known nominal values of the coefficients and $\Delta a_1, \cdots, \Delta a_{n_a}$, $\Delta b_0, \cdots, \Delta b_{n_b}$ are the unknown errors or variations in the coefficients, some of which may be zero.

The concept of pole-placement for controller design has its roots in classical control theory and the idea of placing poles in certain locations to achieve a desired closed-loop behaviour is intuitively appealing. The methods for performing such a design are generally quite straightforward and all of these points help to explain why pole-placement has become very popular in industry for controller design. On the basis of this the approach of pole-placement is adopted as the design procedure for this work.

Before continuing with details of the objectives of this thesis and an outline of the various chapters, it is useful to review the pole-placement design procedure for polynomial systems.

1.4 Pole-Placement Design for Polynomial Systems

Following Wellstead and Sanoff (1981), servo and regulatory control can be applied to the system in (1.4) using the control law:

$$F_p(z^{-1})u(t) = H_p(z^{-1})w(t) - G_p(z^{-1})y(t) \tag{1.10}$$

where

$$F_p(z^{-1}) = f_0 + f_1 z^{-1} + f_2 z^{-2} + \cdots + f_{n_f} z^{-n_f} \tag{1.11}$$

$$G_p(z^{-1}) = g_0 + g_1 z^{-1} + g_2 z^{-2} + \cdots + g_{n_g} z^{-n_g} \tag{1.12}$$

$$H_p(z^{-1}) = h_0 + h_1 z^{-1} + h_2 z^{-2} + \cdots + h_{n_h} z^{-n_h} \tag{1.13}$$

and $w(t)$ is the demand signal.

Note that the pole-placement design assumes the time delay, t_d is incorporated in $B_p(z^{-1})$, hence $n_b = \tilde{n}_b + t_d$ where $\tilde{n}_b$ is the true order of $B_p(z^{-1})$. This will lead to some of the leading coefficients of $B_p(z^{-1})$ being zero. Also, due to sampling, the time delay will always be at least one, so b_0 will be equal to zero.

This gives rise to the closed-loop system as shown in figure 1.3

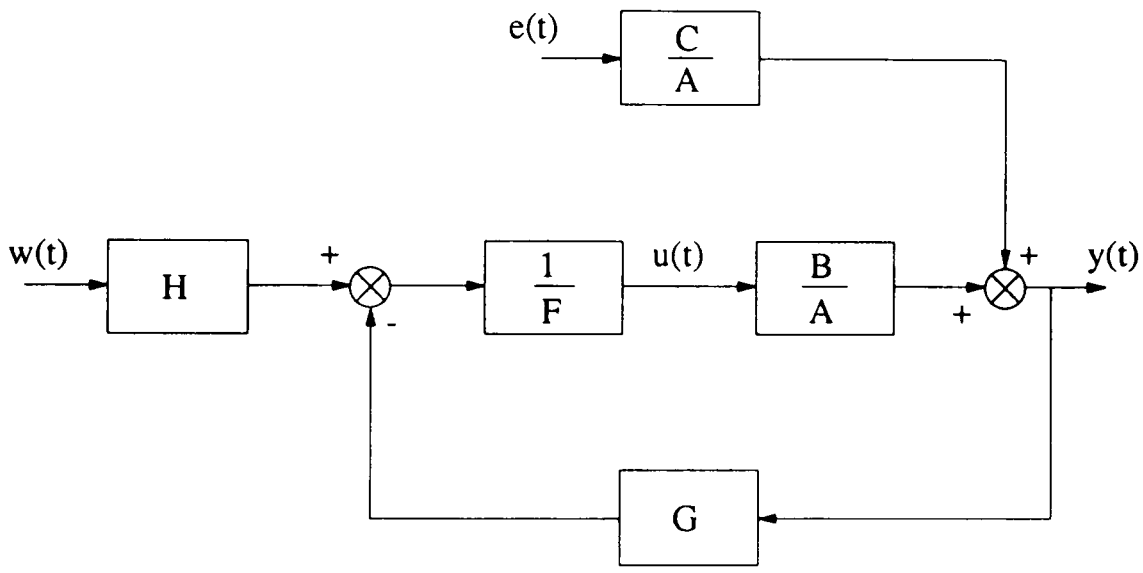


Figure 1.3 - Block Diagram of the Closed-Loop System

which can be expressed as

$$y(t) = \frac{B_p(z^{-1})H_p(z^{-1})}{A_p(z^{-1})F_p(z^{-1}) + B_p(z^{-1})G_p(z^{-1})} w(t) + \frac{F_p(z^{-1})C_p(z^{-1})}{A_p(z^{-1})F_p(z^{-1}) + B_p(z^{-1})G_p(z^{-1})} e(t) \tag{1.14}$$

If $T_p(z^{-1}) = 1 + t_1 z^{-1} + t_2 z^{-1} + \cdots + t_{n_t} z^{-n_t}$ specifies the desired closed-loop pole positions then $F_p(z^{-1})$ and $G_p(z^{-1})$ are obtained from the solution to the diophantine equation

$$A_p(z^{-1})F_p(z^{-1}) + B_p(z^{-1})G_p(z^{-1}) = C_p(z^{-1})T_p(z^{-1}) \tag{1.15}$$

where $C_p(z^{-1})$ is included on the right hand side (RHS) of the equation to minimise the variance of the disturbance. To explain, consider the disturbance term

$$d(t) = C_p(z^{-1})e(t) \tag{1.16}$$

The variance of $d(t)$, $E[d^2(t)]$ can be expressed as

$$\begin{aligned} E[d^2(t)] = E[e^2(t) + c_1^2 e^2(t-1) + c_2^2 e^2(t-2) + \cdots \\ \cdots + c_1 e(t)e(t-1) + c_2 e(t)e(t-2) + \cdots \\ \cdots + c_1 c_2 e(t-1)e(t-2) + \cdots] \end{aligned} \tag{1.17}$$

But as $e(t)$ is an uncorrelated sequence

$$E[e(t-\alpha)e(t-\beta)] = 0 \qquad \text{for} \quad \alpha \neq \beta \tag{1.18}$$

Therefore

$$E[d^2(t)] = E[1 + c_1^2 + c_2^2 + \cdots]\sigma_e^2 \tag{1.19}$$

where σ_e^2 is the variance of $e(t)$.

Clearly if $C_p(z^{-1})$ can be removed from the disturbance term, the variance will be minimised. This can be achieved by forcing the denominator of the closed-loop system to contain $C_p(z^{-1})$ as a factor, hence the form of the diophantine equation (1.15).

This gives the closed-loop system as

$$y(t) = \frac{B_p(z^{-1})H_p(z^{-1})}{C_p(z^{-1})T_p(z^{-1})}w(t) + \frac{F_p(z^{-1})}{T_p(z^{-1})}e(t) \tag{1.20}$$

where the denominator of the demand signal term still contains $C_p(z^{-1})$ which can be removed using the precompensator, $H_p(z^{-1})$, by incorporating it as a factor, i.e.

$$H_p(z^{-1}) = H_p'(z^{-1})C_p(z^{-1}) \tag{1.21}$$

Essentially the precompensator term, $H_p'(z^{-1})$ is used to ensure that the output $y(t)$ tracks the command input $w(t)$ in the steady state. Considering the response of the closed-loop system to a step input, for zero steady state error

$$\frac{B_p(1)H_p'(1)C_p(1)}{A_p(1)F_p(1) + B_p(1)G_p(1)} = 1 \tag{1.22}$$

The form of $H_p'(1)$ which ensures that this equation is satisfied is not unique. If $A_p(z^{-1})$ is forced to contain a factor of $(1 - z^{-1})$ by cascading a digital integrator with the open-loop system, then $A_p(1) = 0$ and hence $H_p'(1)$ can be obtained from

$$H_p'(1) = \frac{G_p(1)}{C_p(1)} \tag{1.23}$$

and as this represents a scalar value, $n_{h'} = 0$. Therefore $H_p(z^{-1}) = H_p'(1)C_p(z^{-1})$ and $n_h = n_c$.

However in practice the model parameters $A_p(z^{-1})$, $B_p(z^{-1})$ and $C_p(z^{-1})$ are generally not accurately known and estimates are used. Hence $F_p(z^{-1})$ and $G_p(z^{-1})$ are obtained from

$$\hat{A}_p(z^{-1})F_p(z^{-1}) + \hat{B}_p(z^{-1})G_p(z^{-1}) = \hat{C}_p(z^{-1})T(z^{-1}) \tag{1.24}$$

where $\hat{A}_p(z^{-1})$, $\hat{B}_p(z^{-1})$ and $\hat{C}_p(z^{-1})$ are estimates of the model parameters. $H_p(z^{-1})$ is then calculated as above using the estimates of the model parameters.

1.5 Objective of the Thesis

From a robustness point of view the diophantine equation is extremely interesting due to the large number of possible solutions, all of which lead to a stabilising controller that places the closed-loop poles in the desired locations. Particular solutions may however yield a closed-loop system with improved robustness properties.

The robust design problem can now be stated as the determination of suitable $F_p(z^{-1})$ and $G_p(z^{-1})$ polynomials which satisfy the diophantine equation (1.15) and which minimise the effect of $\Delta a_1, \cdots, \Delta a_{n_a}, \Delta b_0, \cdots, \Delta b_{n_b}$ on the transient response of the closed-loop system.

Uncertainty in the $C_p(z^{-1})$ polynomial is not considered as it does not affect the transient behaviour of the closed-loop system. It is incorporated, however, in the precompensator $H_p(z^{-1})$ which is selected to achieve zero steady state error. The presence of uncertainty does not represent a problem for steady state tracking if the procedure for selecting the precompensator outlined in the previous section is used. Considering the expression for the precompensator, in the steady state

$$H_p(1) = \frac{C_p(1)G_p(1)}{C_p(1)} = G_p(1) \tag{1.27}$$

and it is clear that good steady state tracking will always be maintained as $H_p(1)$ is independent of any uncertainty in $C_p(z^{-1})$.

Now consider how to solve this problem and obtain the robust $F_p(z^{-1})$ and $G_p(z^{-1})$ polynomials. Section 1.1 gave an indication of various approaches to the solution of the robust control problem and it was noted that a major development was the Youla parameterisation which effectively gives rise to a set of possible controllers, allowing the most robust one to be found. Obtaining a solution to the diophantine equation represents a similar situation where there are a set of controllers and the problem becomes one of searching for the most robust controller.

The concept of searching for a robust controller is quite natural in robust design and is easily formulated in terms of an optimisation problem. Indeed many robust techniques involve some form of optimisation in the design of a suitable controller. The rapid development in computing technology over recent years opens up the possibility of solving the optimisation problem numerically, Maciejowski (1989).

This thesis presents an alternative approach to the solution of the robust design problem as outlined above, based on the theme of utilising modern computing technology to conduct the search for a robust controller, in the form of a numerical optimisation problem.

1.6 Outline of the Thesis

Chapter 2: The diophantine equation is clearly extremely important in the design of a controller for systems in input-output form. This chapter discusses a number of the aspects of this equation to help gain a better understanding of the robust controller design problem. There are two main approaches to solving the equation and they are both reviewed, followed by a discussion of various problems that may be encountered when attempting to find a solution. A simple approach to finding a more robust controller is then developed but it is shown that the method has a number of shortcomings, which leads directly to the idea of a state space design.

Chapter 3: The link between polynomial and state space systems is established showing that, as would be expected, an output feedback state space design must be used. As the aim is to use optimisation techniques to select a more robust controller, a parametric design is used which effectively specifies a set of possible controllers. Two of the main parametric output feedback methods are reviewed and a comparison made of their performance on some test examples. The method which performs better is however not used as the structure of the type of problem being considered here causes it some difficulty. This is discussed more fully when the overall design is applied to an example in chapter 5.

Chapter 4: After determining the set of possible controllers using parametric design, the problem becomes one of how to select the free parameters such that the resulting controller yields a closed-loop system with improved performance robustness. This issue is addressed in this chapter, which first introduces how to quantify mathematically the effect of errors in the model. A mathematical description of the output is then obtained using modal decomposition and from this a number of possible cost functions are derived for use with numerical optimisation algorithms. A general introduction into such algorithms is then given.

Chapter 5: The previous chapters develop the overall robust design technique, this chapter applies the method to an example. With the application of the method arises questions and problems associated with its implementation on a computer and a small discussion of some of the most important points is given. It is then shown why one of the parametric design methods cannot be used on the this type of problem. A comprehensive set of results is then obtained which helps to illustrate the relative benefits of each of the proposed cost functions and the typical improvement that can be achieved with this robust design approach.

Chapter 6: The application of the method to a more realistic problem is considered in this chapter. Daley (1987) considered the application of self-tuning control to a hydraulic rig to help overcome problems associated with varying supply pressure and load. From the basic physical equations of the plant a nonlinear continuous time simulation of the rig is set up and a robust controller designed from a model obtained using system identification techniques. It is shown

that the robust controller performs well compared to the controller obtained from the minimum order solution to the diophantine equation. Also the performance compares favourably with that of the self-tuning controller of Daley (1987).

Chapter 7: The conclusions drawn from the preceding chapters are presented here. The chapter brings together and highlights both the advantages and the disadvantages of this type of approach to designing robust controllers. There are still a number of problems with the method and a discussion of these follows, leading onto some suggestions for future work.

REFERENCES

Bertsekas, D.P. and Rhodes, I.B. (1973)
'Sufficiently Informative Functions and the Minimax Feedback Control of Uncertain Dynamic Systems'
IEEE Transactions on Automatic Control, v18, n4, pp 117-124

Black, H.S. (1927)
'Stabilized Feedback Amplifiers'
U.S. Patent 2,102,671

Bode, H.W. (1945)
'Network Analysis and Feedback Amplifier Design'
Van Nostrand, Wokingham, U.K.

Byers, R. and Nash, S.G. (1989)
'Approaches to Robust Pole Assignment'
International Journal of Control, v49, n1, pp 97-117

Daley, S. (1987)
'Application of a Fast Self-Tuning Control Algorithm to a Hydraulic Test Rig'
Proceedings of the Institute of Mechanical Engineers, v201, nC4, pp 285-295

Dorato, P. (1987)
'A Historical Review of Robust Control'
IEEE Control Systems Magazine, v7, n2, pp 44-47

Doyle, J.C. (1978)
'Guaranteed Margins for LQG Regulators'
IEEE Transactions on Automatic Control, v23, n4, pp 756-757

Doyle, J.C. (1982)
'Analysis of Feedback Systems with Structured Uncertainties'
IEE Proceedings, v129, Pt D, n6, pp 242-250

Doyle, J.C. and Stein, G. (1979)
'Robustness with Observers'
IEEE Transactions on Automatic Control, v24, n8, pp 607-611

Doyle, J.C. and Stein, G. (1981)
'Multivariable Feedback Design: Concepts for a Classical/Modern Synthesis'
IEEE Transactions on Automatic Control, v26, n2, pp 4-16

Doyle, J.C. and Wall, J.E. (1982)
'Performance and Robustness Analysis for Structured Uncertainty'
Proceedings of the 21st IEEE Conference on Decision and Control, Orlando, Florida, U.S.A. v2, pp 629-636
IEEE, New York, U.S.A.

Gourishankar, V. and Ramar, K. (1976)
'Pole Assignment with Minimum Eigenvalue Sensitivity to Plant Parameter Variations'
International Journal of Control, v23, n4, pp 493-504

Horowitz, I. (1979)
'Quantitative Synthesis of Uncertain Multiple Input-Output Feedback Systems'
International Journal of Control, v30, n1, pp 81-106

Horowitz, I. (1982)
'Quantitative Feedback Theory'
IEE Proceedings, v129, Pt D, n6, pp 215-226

Horowitz, I. and Sidi, M. (1980)
'Practical Design of Multivariable Feedback Systems with Large Plant Uncertainty'
International Journal of Systems Science, v11, pp 851-875

Kalman, R.E. (1964)
'When is a Linear Control System Optimal?'
Transactions of the ASME, Ser D, Journal of Basic Engineering, v86, pp 51-60

Kautsky, J., Nichols, N.K. and Van Dooren, P. (1985)
'Robust Pole Assignment in Linear State Feedback'
International Journal of Control, v41, n5, pp 1129-1155

Maciejowski, J.M. (1989)
'Multivariable Feedback Design'
Addison-Wesley, Wokingham, U.K.

Morari, M. and Zafiriou, E. (1989)
'Robust Process Control'
Prentice-Hall, London, U.K.

Nyquist, H. (1932)
'Regeneration Theory'
Bell System Technical Tour, v2, pp 126-147

Owens, T.J. and O'Reilly, J. (1989)
'Parametric State Feedback Control for Arbitrary Eigenvalue Assignment with Minimum Sensitivity'
IEE Proceedings, Pt D, v136, n6, pp 307-312

Ragade, R.K. and Sarma, I.G. (1967)
'A Game Theoretic Approach to Optimal Control in the Presence of Uncertainty'
IEEE Transactions on Automatic Control, v12, n8, pp 395-402

Safonov, M.G. and Athans, M. (1977)
'Gains and Phase Margin for Multiloop LQG Regulators'
IEEE Transactions on Automatic Control, v22, n4, pp 173-179

Wellstead, P.E. and Sanoff, S.P. (1981)
'Extended Self-Tuning Algorithm'
International Journal of Control, v34, n3, pp 433-455

Wonham, W.M. (1967)
'Optimal Stationary Control of a Linear System with State-Dependent Noise'
SIAM Journal on Control, v5, n3, pp 486-500

Youla, D.C., Jabr, H.A. and Bongiorno, J.J. (1976)
'Modern Wiener-Hopf Design of Optimal Controllers - Part II: The Multivariable Case'
IEEE Transactions on Automatic Control, v21, n4, pp 75-93

Zames, G. (1981)
'Feedback and Optimal Sensitivity: Model Reference Transformations, Multiplicative Seminorms and Approximate Inverses'
IEEE Transactions on Automatic Control, v26, n2, pp 301-320

CHAPTER 2

THE DIOPHANTINE EQUATION

2.1 Introduction

Section 1.4 detailed the pole-placement design procedure for polynomial systems. From this it is clear that the diophantine equation

$$A_p F_p + B_p G_p = C_p T_p \tag{2.1}$$

is extremely important, as the whole design centres around obtaining its solution, which of course gives the F_p and G_p controller polynomials. Once a solution has been found the third controller polynomial, the precompensator H_p is easily obtained. As the solution of this equation is such an important part of the design stage it is useful to gain an understanding of the conditions under which solutions exist, the range of possible solutions and the approaches that can be used to obtain a solution.

There are basically two approaches to the solution of the equation and these are discussed more fully in the following two sections. Various problems associated with finding a solution are then discussed, followed by details on some work carried out on obtaining more robust solutions to the diophantine equation. However, before proceeding with these topics it is useful to present two theorems (Kucera, 1979) which clarify the conditions for the existence of solutions and the range of possible solutions.

THEOREM 2.1:

The equation (2.1) has a solution if and only if the greatest common divisor of A_p and B_p is a factor of the right hand side (RHS), $C_p T_p$.

PROOF:

STEP 1 - Let F_0 and G_0 be a solution to the diophantine equation and the greatest common divisor of A_p and B_p be g_p.

Then

$$A_p = g_p A_0 \tag{2.2}$$

$$B_p = g_p B_0 \tag{2.3}$$

and

$$g_p(A_0F_0 + B_0G_0) = C_pT_p \tag{2.4}$$

Hence g_p is a factor of C_pT_p.

STEP 2 - Assume $C_pT_p = g_pC_0T_0$

It is well known that two polynomials, P_p and Q_p, always exist such that

$$A_pP_p + B_pQ_p = g_p \tag{2.5}$$

Multiplying by C_0T_0 gives

$$A_p[P_pC_0T_0] + B_p[Q_pC_0T_0] = C_pT_p \tag{2.6}$$

Hence the solution of (2.1)

[]

This theorem basically outlines the conditions for a solution to exist to the diophantine equation. Its importance will become clear later when the problems associated with this equation are discussed.

THEOREM 2.2:

Let F_0 and G_0 be a solution to equation (2.1). The general form of the solution is

$$F_p = F_0 - B_0X_p \tag{2.7}$$

$$G_p = G_0 + A_0X_p \tag{2.8}$$

where A_0 and B_0 are as defined in theorem 2.1 and X_p is some polynomial.

PROOF:

Clearly

$$A_pF_0 + B_pG_0 = C_pT_p \tag{2.9}$$

and

$$A_pF_p + B_pG_p = C_pT_p \tag{2.10}$$

therefore

$$A_pF_0 + B_pG_0 = A_pF_p + B_pG_p \tag{2.11}$$

$$-A_p(F_p - F_0) = B_p(G_p - G_0) \tag{2.12}$$

From theorem 2.1, $A_p = g_pA_0$ and $B_p = g_pB_0$. The polynomials A_0 and B_0 are coprime and satisfy $A_pB_0 = B_pA_0$, so B_0 must be a divisor of $-(F_p - F_0)$ and A_0 a divisor of $(G_p - G_0)$, i.e.

$$(F_p - F_0) = -B_0X_p \tag{2.13}$$

$$(G_p - G_0) = A_0X_p \tag{2.14}$$

for some polynomial X_p, hence the general form for the solution to (2.1).

□

Theorem 2.2 highlights the fact that there are infinitely many solutions to the diophantine equation (2.1).

The equation can be solved by either matrix methods or by polynomial methods (Kucera, 1979; Clarke, 1982; Mohtadi, 1988; Astrom and Wittenmark, 1989). A review of each approach is given, followed by a discussion of problems associated with the equation and various suggested methods to help overcome these problems. The chapter finishes with a novel investigation aimed at obtaining a more robust solution to the diophantine equation.

2.2 Solution via Polynomial Methods

The polynomial solution outlined here follows that of Kucera (1979), although many authors have presented similar derivations.

From theorem 2.2, the general form of the solution is

$$F_p = F_0 - B_0X_p \tag{2.15}$$

$$G_p = G_0 + A_0X_p \tag{2.16}$$

where X_p is some polynomial.

There are many ways to calculate this solution, one of which is to use an extended Euclidean algorithm which calculates a greatest common divisor (GCD), g_p of A_p and B_p, along with two pairs of coprime polynomials P_p, Q_p and R_p, S_p satisfying

$$A_p P_p + B_p Q_p = g_p \tag{2.17}$$

$$A_p R_p + B_p S_p = 0 \tag{2.18}$$

Also

$$C_0 T_0 = \frac{C_p T_p}{g_p} \tag{2.19}$$

Hence the general solution is

$$F_p = P_p C_0 T_0 + R_p X_p \tag{2.20}$$

$$G_p = Q_p C_0 T_0 + S_p X_p \tag{2.21}$$

Note that $C_0 T_0$ must be a finite polynomial, which forms a useful check on the existence of a solution.

A special solution is the minimum degree solution with respect to (w.r.t) F_p or G_p. It is calculated using the polynomial division algorithm to find (in the case of the minimum degree solution w.r.t F_p)

$$F_0 = B_0 u_p + v_p \tag{2.22}$$

where u_p is the quotient and v_p the remainder. Then

$$F_p = v_p - B_0 (X_p - u_p) \tag{2.23}$$

and the minimum degree solution is obtained by putting $X_p = u_p$, therefore

$$F_p = v_p \tag{2.24}$$

$$G_p = G_0 + A_0 u_p \tag{2.25}$$

This is a unique solution and may not necessarily be the same as the minimum degree solution w.r.t G_p.

Appendix A contains details of the extended Euclidean algorithm and the polynomial division algorithm.

2.3 Solution via Matrix Methods

The equation (2.1) can be transformed to a matrix equation of the form $A_c\underline{x} = \underline{b}$ and matrix methods used to obtain a solution.

Expanding (2.1) gives

$$A_pF_p = f_0 + (a_1f_0 + f_1)z^{-1} + (a_2f_0 + a_1f_1 + f_2)z^{-2} + \cdots + a_{n_a}f_{n_f}z^{-(n_a+n_f)} \tag{2.26}$$

$$B_pG_p = b_0g_0 + (b_1g_0 + b_0g_1)z^{-1} + (b_2g_0 + b_1g_1 + b_0g_2)z^{-2} + \cdots$$
$$\cdots + b_{n_b}g_{n_g}z^{-(n_b+n_g)} \tag{2.27}$$

$$C_pT_p = 1 + (c_1 + t_1)z^{-1} + (c_2 + c_1t_1 + t_2)z^{-2} + \cdots + c_{n_c}t_{n_t}z^{-(n_c+n_t)} \tag{2.28}$$

Assuming

$$n_a + n_f = n_b + n_g \tag{2.29}$$

which can always be achieved by padding with zero terms if necessary, the diophantine equation can be represented as

$$\begin{aligned}&[f_0 + b_0g_0 \quad a_1f_0 + f_1 + b_0g_1 + b_1g_0 \quad a_2f_0 + a_1f_1 + f_2 + b_0g_2 + b_1g_1 + b_2g_0 \quad \cdots \\ &\quad \cdots \quad a_{n_a}f_{n_f} + b_{n_b}g_{n_g}] \cdot \left[z^0 \quad z^{-1} \quad z^{-2} \quad \cdots \quad z^{-(n_a+n_f)}\right]^T = \\ &[1 \quad c_1 + t_1 \quad c_2 + c_1t_1 + t_2 \quad \cdots \quad c_{n_c}t_{n_t}] \cdot \left[z^0 \quad z^{-1} \quad z^{-2} \quad \cdots \quad z^{-(n_c+n_t)}\right]^T\end{aligned} \tag{2.30}$$

If $n_c + n_t \le n_a + n_f$ then the equation can be expressed as

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & b_0 & 0 & 0 & \dots & 0 \\ a_1 & 1 & 0 & & \cdot & b_1 & b_0 & 0 & & \cdot \\ a_2 & a_1 & 1 & & \cdot & b_2 & b_1 & b_0 & & \cdot \\ \cdot & a_2 & a_1 & & \cdot & \cdot & b_2 & b_1 & & \cdot \\ \cdot & \cdot & a_2 & & 1 & \cdot & \cdot & b_2 & & b_0 \\ a_{n_a} & \cdot & \cdot & & a_1 & b_{n_b} & \cdot & \cdot & & b_1 \\ 0 & a_{n_a} & \cdot & & a_2 & 0 & b_{n_b} & \cdot & & b_2 \\ 0 & 0 & a_{n_a} & & \cdot & 0 & 0 & b_{n_b} & & \cdot \\ \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & & \cdot \\ 0 & 0 & 0 & \dots & a_{n_a} & 0 & 0 & 0 & \dots & b_{n_b} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_{n_f} \\ g_0 \\ g_1 \\ \cdot \\ \cdot \\ g_{n_g} \end{bmatrix} = \begin{bmatrix} 1 \\ c_1 + t_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ c_{n_c} t_{n_t} \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \tag{2.31}$$

which clearly is of the form $A_s \underline{x} = \underline{b}$ where A_s is a sylvester matrix of the coefficients of A_p and B_p, $\underline{x}$ is a vector of unknown controller polynomial coefficients and $\underline{b}$ is a vector containing the coefficients of $C_p T_p$.

If n_f is set to $n_b - 1$ and n_g to $n_a - 1$ then this will give rise to the minimum order solution and A_s will be square. The set of equations can then easily be solved by inverting A_s or preferably, from a numerical point of view, by one of a number of algorithms to solve a set of linear equations such as Crouts factorisation method, NAG (1990). If n_f or n_g are set to higher values then A_s will no longer be square and it is necessary to arbitraily set some of the unknowns to find a solution. Section 2.5.3 discusses this aspect in greater depth later on.

2.4 Problems Associated with Finding a Solution

Theorem 2.1 gives a good indication of when problems will arise with finding a solution to the diophantine equation. If A_p and B_p have an exact common factor which is not a factor of the RHS, then no solution exists.

In practice, however it is more likely that a near common factor will be encountered. There are two principle ways that such a factor can arise (Mohtadi, 1988).

1) As the sample rate increases the poles and zeros of a discrete system tend to map to a region close to the (1,0) point in the z-plane (Astrom *et al*, 1984), obviously leading to common factors.

2) It is possible to overparameterise real systems during identification if slow sample rates are used in conjunction with high order models resulting in a possible common factor.

Although a solution can generally be obtained in the presence of a near common factor, it tends to be a poor one in terms of numerical robustness. To help understand the problems that can arise with such a factor, consider a simple example.

Example 2.4.1:

$$A_p = (1 + \hat{a}z^{-1})(1 + 2z^{-1}) \tag{2.32}$$

$$B_p = (1 + z^{-1})(3z^{-1}) \tag{2.33}$$

$$C_pT_p = 1 + z^{-1} + 2z^{-2} + z^{-3} \tag{2.34}$$

where $\hat{a}$ is selected as 0.9999 and 1.0001. The following solutions were obtained using Pro-Matlab version 3.5e.

Matrix solution:

$\hat{a} = 0.9999$ $\quad F_p = 1 + 1.0001e4z^{-1}$ (2.35)

$$G_p = -3.3343e3 - 6.6663e3z^{-1} \tag{2.36}$$

$\hat{a} = 1.0001$ $\quad F_p = 1 - 9.999e3z^{-1}$ (2.37)

$$G_p = 3.3323e3 + 6.667e3z^{-1} \tag{2.38}$$

Polynomial solution:

The general solution is used with the arbitrary polynomial set to 1.

$\hat{a} = 0.9999$ $\quad F_p = 1 - 4.9999z^{-1} - 1.0008e4z^{-2} - 1.0009e4z^{-3} -$

$$2.0011e4z^{-4} - 1.0004e4z^{-5} \tag{2.39}$$

$$G_p = 1 + 0.334e4z^{-1} + 1.0008e4z^{-2} + 1.3343e4z^{-3} +$$

$$1.6674e4z^{-4} + 0.6669e4z^{-5} \tag{2.40}$$

$\hat{a} = 1.0001$ $\quad F_p = 1 - 5.0001z^{-1} + 0.9992e4z^{-2} + 0.9991e4z^{-3} +$

$$1.9989e4z^{-4} + 0.9996e4z^{-5} \tag{2.41}$$

$$G_p = 1 - 0.3327e4z^{-1} - 0.9992e4z^{-2} - 1.3324e4z^{-3} -$$

$$1.6659e4z^{-4} - 0.6665e4z^{-5} \tag{2.42}$$

Clearly the presence of a near common factor causes the value of some coefficients to be quite large which is undesirable. Of more significance though is the dramatic change in the values with a small change in $\hat{a}$ around the nominal value of 1 (for an exact common factor).

A second problem with finding a solution occurs when the coefficients of B_p become small. The reason for this in the case of the matrix approach is that columns of A_s are close to zero and the matrix becomes ill-conditioned. For the polynomial approach large multipliers appear in the extended Euclidean algorithm resulting in large values for some of the coefficients of the resulting polynomials. This is an important problem as the magnitude of the coefficients of B_p is dependent on the sample rate (Mohtadi, 1988).

Intuitively, for the polynomial solution, a simple approach to overcoming the problem of a common factor is to force the RHS to contain it as a factor and find a solution to

$$A_pF_p + B_pG_p = g_pC_pT_p \tag{2.43}$$

and then g_p can be dealt with in the same way as C_p in the precompensator. For example consider the closed-loop system

$$\frac{y(t)}{u(t)} = \frac{B_pH_p}{A_pF_p + B_pG_p} = \frac{B_pH_p}{g_pC_pT_p} \tag{2.44}$$

To eliminate g_p and C_p from the denominator, H_p must contain g_pC_p as a factor, i.e.

$$H_p = g_pC_pH_p' \tag{2.45}$$

and H_p' can be calculated as before for zero steady state error.

Clearly this has the disadvantage of increasing the order of the precompensator polynomial but has the advantage of being extremely easy to implement. When calculating C_0T_0, if a remainder is left then g_p is not a factor of C_pT_p and the above procedure must be carried out giving $C_0T_0 = C_pT_p$ and hence a solution.

The procedure does assume that the common factor is exact which will generally not be the case. In example 2.4.1 with a near common factor the extended Euclidean algorithm returned $g_p = 1$ and hence failed to detect its presence. Clearly the above procedure is worthless in such a case.

For the matrix approach the only complete solution is to isolate the offending common factor in A_p and B_p and remove it from the polynomials before constructing the simultaneous equations (Tuffs, 1984). Again this relies on having an exact common factor so is not appropriate for practical problems.

It is possible to formulate a recursive solution to (2.1) by introducing an arbitrary signal $\xi(t)$ to produce a regression model (Edmunds, 1976; Alix *et al*, 1982)

$$ct(t) = F_p a(t) + G_p b(t - t_d) \tag{2.46}$$

where $ct(t) = C_p T_p \xi(t)$, $a(t) = A_p \xi(t)$ and $b(t) = B_p \xi(t)$.

This can then be used as the basis of a recursive estimator with a parameter vector $[F_p \quad G_p]$, a measurement $ct(t)$ and a data vector

$$[a(t) \quad a(t-1) \quad \cdots \quad b(t-t_d) \quad b(t-t_d-1) \quad \cdots \quad] \tag{2.47}$$

The problem of a common factor is also present in this framework and appears as linear dependence in the data vector. Theorem 2.1 showed that no solution exists unless the common factor is also a factor of the RHS so it is reasonable to expect the estimator to experience some difficulty in converging to a solution as in fact none exists.

Another approach (Lawson and Hanson, 1974; Tuffs, 1984) is to examine the 'pseudo-rank'[1] of the A_s matrix and if a rank deficiency is detected, calculate a 'minimum-norm' solution, $\bar{x}$ which minimises the Euclidean length of $\underline{\bar{b}} = \underline{b} - A_s \underline{\bar{x}}$. Such a solution is numerically very robust. The major drawback here is that it has not been proved that the closed-loop system is stable under all conditions.

Berger (1988) suggests splitting the desired closed-loop poles into two parts J_p and K_p. The first part, J_p is chosen to satisfy the desired design criteria whilst K_p, the second part, is initially set to zero but can be adjusted to improve the conditioning of the set of linear equations. It is necessary to specify bounds for the coefficients of K_p.

For the problem of small B_p coefficients due to rapid sampling (which can also give rise to a common factor), Middleton and Goodwin (1986) have proposed a method which involves replacing the z^{-1} operator with the δ operator which is defined as

1 Lawson and Hanson (1974) define the pseudo-rank of a matrix A to be the rank of a matrix $\bar{A}$ that replaces A as a result of a specific computational algorithm.

$$\delta = \frac{z-1}{\Theta} \tag{2.48}$$

where Θ is the sampling interval. It is claimed that the use of this operator gives rise to a number of benefits including improved finite word length characteristics and an improvement in the conditioning of the sylvester matrix. Of course the implementation of control strategies in the δ operator are more complex than those in the more common shift operator.

There are many other discussions of the diophantine equation and its properties in the literature together with a number of proposed methods for overcoming the problems associated with finding a solution. Such a proliferation of methods indicates that no one approach can deal with all the shortcomings of this equation and that the calculation of its solution should be carried out with some care.

2.5 Obtaining a More Robust Solution

Putting aside the problems associated with the equation, another interesting aspect is the number of possible solutions as highlighted by theorem 2.2. As a first step towards the goal of designing polynomial controllers with improved performance robustness, it would be interesting to investigate the robustness properties of various solutions to the diophantine equation.

The matrix approach appears to be the more popular method of solving the equation. This is probably due to a greater general familiarity with the theory of matrices, the fact that the matrix representation of the equation is of a standard form and lastly because the matrix method is more easily implemented on a computer. Thus it seems appropriate to base the investigation on this approach. It is assumed that there are no problems with common factors or small coefficients and so standard matrix analysis is used to solve the equation.

Based on this method for solving the equation, the effect of errors (or perturbations) in the model parameters is considered which helps to establish a suitable robustness criteria. It will be seen that vector and matrix norms play an important role in the evaluation of this criteria and so a brief discussion and definition of them is included, followed by some comments on how to select alternative solutions. A set of results using the proposed robustness criteria are presented for a number of examples and conclusions drawn about the suitability of such an approach for improving performance robustness.

2.5.1 The Effect of Parameter Perturbations

If the parameters are subject to perturbations then the matrix equation (2.31) can be represented as

$$(A_s + \Delta A_s)(\underline{x} + \Delta \underline{x}) = (\underline{b} + \Delta \underline{b}) \tag{2.49}$$

where A_s, $\underline{x}$ and $\underline{b}$ represent the true values and ΔA_s, $\Delta \underline{x}$ and $\Delta \underline{b}$ the errors.

It is necessary to establish some sort of measure to gauge the robustness of various solutions. The matrix form of the diophantine equation is of a standard form on which much work has been carried out. Perturbation theory for linear systems can be used to help establish the appropriate criteria.

A suitable measure of robustness could be obtained by computing an upper bound for the relative perturbation $\| \Delta \underline{x} \| / \| \underline{x} \|$. There are a number of such bounds, the following is taken from a derivation in Lancaster and Tismenetsky (1985).

Subtracting $A_s \underline{x} = \underline{b}$ from (2.49) gives

$$(A_s + \Delta A_s)\Delta \underline{x} + \Delta A_s \underline{x} = \Delta \underline{b} \tag{2.50}$$

or

$$A_s(I + A_s^{-1}\Delta A_s)\Delta \underline{x} = \Delta \underline{b} - \Delta A_s \underline{x} \tag{2.51}$$

Lancaster and Tismenetsky (1985) show that the existence of $(I + A_s^{-1}\Delta A_s)^{-1}$ is implied if

$$\rho = \| A_s^{-1} \| \, \| \Delta A_s \| < 1 \tag{2.52}$$

where the particular norm used must satisfy

$$\| I \| = 1 \tag{2.53}$$

Also

$$\| (I + A_s^{-1}\Delta A_s)^{-1} \| < (1 - \rho)^{-1} \tag{2.54}$$

Thus

$$\Delta\underline{x} = (I + A_s^{-1}\Delta A_s)^{-1}A_s^{-1}\Delta\underline{b} - (I + A_s^{-1}\Delta A_s)^{-1}A_s^{-1}\Delta A_s\underline{x} \tag{2.55}$$

and if $\|\cdot\|_v$ is any vector norm compatible with $\|\cdot\|$,

$$\|\Delta\underline{x}\|_v \le \frac{\|A_s^{-1}\|}{1-\rho}\|\Delta\underline{b}\|_v + \frac{\rho}{1-\rho}\|\underline{x}\|_v \tag{2.56}$$

$A_s\underline{x} = \underline{b}$ implies that $\|\underline{b}\|_v \le \|A_s\|\,\|\underline{x}\|_v$ hence

$$\frac{1}{\|\underline{x}\|_v} \le \frac{\|A_s\|}{\|\underline{b}\|_v} \tag{2.57}$$

Thus

$$\frac{\|\Delta\underline{x}\|_v}{\|\underline{x}\|_v} \le \frac{\|A_s\|\,\|A_s^{-1}\|}{1-\rho}\cdot\frac{\|\Delta\underline{b}\|_v}{\|\underline{b}\|_v} + \frac{\rho}{1-\rho} \tag{2.58}$$

Define $\kappa(A_s) = \|A_s\|\,\|A_s^{-1}\|$ to be the condition number of A_s and note that

$$\rho = \|A_s^{-1}\|\,\|\Delta A_s\| = \frac{\kappa(A_s)\|\Delta A_s\|}{\|A_s\|} \tag{2.59}$$

Thus

$$\frac{\|\Delta\underline{x}\|_v}{\|\underline{x}\|_v} \le \frac{\kappa(A_s)}{1-\kappa(A_s)\|\Delta A_s\|/\|A_s\|}\left[\frac{\|\Delta\underline{b}\|_v}{\|\underline{b}\|_v} + \frac{\|\Delta A_s\|}{\|A_s\|}\right] = U_b \tag{2.60}$$

It would seem reasonable to suggest that the solution which minimises the upper bound, U_b is the most robust solution as this minimises the maximum possible variation in $\underline{x}$.

However, these results are really only valid for small perturbations in A_s and $\underline{b}$. As the relative perturbation of A_s increases, there will come a point when the evaluation of U_b will return a negative value, due to $\kappa(A_s)\|\Delta A_s\|/\|A_s\|$ becoming > 1. The relative perturbation of $\underline{x}$ should always be positive, thus the value of U_b will be invalid.

At this point it is worth noting that the essential difference between sensitivity and robustness analysis is that sensitivity based results are concerned with small perturbations, whereas more significant parameter variations are considered with robust design techniques. The above perturbation theory gives rise to a sensitivity result, hence the upper bound becoming invalid for larger changes. However, sensitivity analysis can provide a useful insight into appropriate robustness measures.

Upon closer examination of the expression for U_b it can be seen that the condition number of A_s has a large influence on the upper bound of the relative perturbation of $\underline{x}$. Based on this observation and the knowledge of the benefits of achieving well conditioned matrices, it would seem reasonable to suggest that the conditioning of the sylvester matrix would be a useful measure of robustness for systems with not necessarily small parameter perturbations.

2.5.2 Matrix and Vector Norms

The upper bound, U_b of the relative perturbation and the condition number depend on vector and matrix norms, of which there are many. The most commonly used matrix norms are P or Holder norms and the Frobenius or Euclidean norm.

The P norm is defined as

$$\|A\|_p = \sup_{\underline{x}\neq 0} \frac{\|A\underline{x}\|_p}{\|\underline{x}\|_p} \tag{2.61}$$

for any $\underline{x}$, where $\|\underline{x}\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}$ and p is generally taken as $p = 1, 2$ or ∞.

The Frobenius norm is defined as

$$\|A\|_F = \left[\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2\right]^{1/2} \tag{2.62}$$

However because of the assumption in (2.53) that $\|I\| = 1$ the Frobenius norm cannot be used as $\|I\|_F = n^{1/2}$. Essentially the aim here is to find the minimum condition number and it is well known (Golub and Van Loan, 1983; Horn and Johnson, 1985) that if a point is a minimum with respect to one norm it will also be a minimum with respect to another norm. This equivalence of norms means that any one of the p norms could be used to calculate the condition number and select the most robust solution.

2.5.3 Selecting the Order of the Controller Polynomials

A common approach to solving equation (2.31) is to use the minimum order solution (Kucera, 1979; Wellstead and Sanoff, 1981; Clarke, 1982) where

$$n_f = n_b - 1 \tag{2.63}$$

$$n_g = n_a - 1 \tag{2.64}$$

although a number of other authors have proposed using alternative solutions (for example Astrom and Wittenmark, 1980; McDermott and Mellichamp, 1984; Warwick *et al*, 1985). The choice of which solution is 'best' is still an area of on-going research and of course will depend on the design objectives.

To select other solutions the orders of F_p and G_p will have to be changed. For the matrix equation this will mean changing the dimension of the matrix and vectors. Thus it is necessary to understand the conditions under which n_f and n_g can be selected.

In equation (2.31), the number of columns containing A_p coefficients = the number of F_p coefficients = $n_f + 1$, and the number of columns containing B_p coefficients = the number of G_p coefficients = $n_g + 1$.

$$\text{i.e.} \qquad N_c = N_u = n_f + n_g + 2 \tag{2.65}$$

where N_c = the number of columns and N_u = the number of unknowns.

The number of A_p coefficients = $n_a + 1$, and B_p coefficients = $n_b + 1$. Again examining equation (2.31) it is clear that the coefficients are moved down by one row for each successive column, therefore the number of extra rows created is n_f for the A_p coefficients and n_g for the B_p coefficients.

i.e. $$N_r = N_e = \max\{n_a + n_f + 1, n_b + n_g + 1\} \tag{2.66}$$

where N_r = the number of rows and N_e = the number of equations.

From equation (2.29) it is clear that the two values in the brackets will be equal, so either can be used.

As the number of rows and columns are affected there are three possible situations that could arise, assuming that A_s is of full rank:

i) The number of rows = the number of columns

$$n_a + n_f + 1 = n_f + n_g + 2 \qquad \text{and} \qquad n_b + n_g + 1 = n_f + n_g + 2$$

$$n_g = n_a - 1 \qquad\qquad n_f = n_b - 1$$

and a solution can be easily obtained. This case only occurs for the minimum order solution.

ii) The number of rows < the number of columns

This corresponds to the case where the number of unknowns > the number of equations, thus by setting some of the unknowns arbitrarily it is possible to easily obtain a solution.

iii) The number of rows > the number of columns

In this case there are more equations than unknowns, which can lead to problems of inconsistency where a set of values for the unknowns is obtained which do not satisfy all of the equations. To guarantee that such problems do not arise it is necessary to investigate the conditions under which this case will never occur.

Limits for n_f and n_g such that the number of rows ≤ the number of columns are

$$n_a + n_f + 1 \le n_f + n_g + 2 \qquad \text{and} \qquad n_b + n_g + 1 \le n_f + n_g + 2$$

$$n_g \ge n_a - 1 \qquad\qquad n_f \ge n_b - 1$$

As $n_g = n_a - 1$ and $n_f = n_b - 1$ are the minimum order solution, these conditions will always be fulfilled and case iii) can never occur.

Suitable choices for n_f and n_g can be deduced from the conditions mentioned above. To summarise, all of the following constraints must be satisfied :

a) $n_a + n_f = n_b + n_g$

b) $n_g \geq n_a - 1$

c) $n_f \geq n_b - 1$

d) if $N_u > N_e$ then some of the unknowns must be arbitrarily assigned

The first constraint means that if the order of F_p is increased then the order of G_p must be increased by the same amount.

2.5.4 Simulation Results

Three examples are considered

1) A non-minimum phase system (taken from Wellstead and Sanoff, 1981)

$$(1 - 1.6z^{-1} + 0.6z^{-2})y(t) = (z^{-1} + 1.5z^{-2})u(t) + (1 - 0.4z^{-1})e(t) \quad (2.67)$$

and it is desired to have a closed loop pole at 0.8

It is assumed that the A_p, B_p and C_p polynomial coefficients are subject to uniformly distributed random perturbations of 2%, 5% and then 15%.

2) A system proposed by Berger (1984)

$$(1 - 2z^{-1} + z^{-2})y(t) = (z^{-1} + 0.1z^{-2})u(t) \quad (2.68)$$

and the desired closed loop poles are all assumed to be zero.

The A_p, B_p polynomial coefficients are now assumed to be subject to normally distributed random perturbations with variances of firstly 0.01% and 0.02% respectively and then 5% and 10% respectively.

3) A hydraulic rig (taken from Daley, 1987)

$$(1 - 0.54666z^{-1})y(t) = (1.28621z^{-1})u(t) \quad (2.69)$$

and the desired closed loop pole positions are $0.75 \pm j0.2$

The A_p, B_p polynomial coefficients are time-varying with respect to the supply pressure. The relative size of the variations are 10% and 125% respectively.

A digital integrator is cascaded with each system to achieve the desired steady state performance as outlined in section 1.2, hence two closed loop poles being specified for example 3 which is a first order system.

The minimum order solution is unique, however when n_f and n_g are increased beyond their minimum order values, a set of solutions is obtained. The size of the set increases as n_f and n_g increase. Consider the case where n_f and n_g are increased by one from their minimum order values. This particular set of solutions will be referred to as the minimum order + 1 solutions. Section 2.5.3 outlined how the size of equation (2.31) was affected by changes in n_f and n_g. From this work, in particular equations (2.65) and (2.66), it is clear that the number of unknowns in equation (2.31) increases by two whereas the number of individual equations (or rows of the matrix A_s) only increases by one. In order to solve equation (2.31) for a particular solution it is necessary to arbitrarily set one of the unknowns. The mathematics involved can be greatly simplified if the unknown is set to zero as this is equivalent to deleting one of the columns from the sylvester matrix A_s. It is important to ensure that the resulting square sylvester matrix is of full rank, else the solution will suffer from numerical problems as outlined in section 2.4.

If n_f and n_g are increased by two from their minimum order values then the minimum order + 2 set of solutions is obtained. In this case it is necessary to arbitrarily set two of the unknowns, and if zero is again used this translates to deleting two columns from the sylvester matrix.

It is possible to continue increasing n_f and n_g resulting in even larger sets of solutions. However, for the purposes of this investigation only solutions up to and including the minimum order + 2 set will be used, as this should give a sufficient indication of the performance of the robustness measures and the suitability of this approach.

The levels of perturbation specified for examples 1 and 3 define the approximate level of random perturbation required, and for example 2 the actual variance of the perturbation is given. On the basis of this, four sets of random perturbations are generated for each example to allow a better investigation into the correlation between the measures of robustness (condition number and upper bound) and the true relative perturbation. This results in four plots appearing in each figure corresponding to the four sets of random perturbations.

For the minimum order + 1 solutions the x-axis on the graphs corresponds to which column was deleted from the sylvester matrix. However, as f_0 is fixed at 1, column 1 was not actually deleted and in its place are the results for the minimum order solution. For the minimum order + 2 solutions the x-axis can no longer be used to indicate which columns are deleted as it is now necessary to remove two. Instead all possible solutions are shown in no particular order except that the minimum order solution is still first. The 2-norm was used for calculating the condition number and upper bound throughout. The graphs are located at the end of the chapter.

The 2% perturbation results for example 1 are shown in figures 2.1 and 2.2. The upper bound and the condition number agree quite well with the lowest minima indicating the best solutions. From the graph of the true relative perturbation it appears that the measures are generally selecting good solutions as regards robustness. Figures 2.3 and 2.4 show the 5% perturbation results. The upper bound is only valid for the minimum order + 1 solutions where there is good agreement with the condition number on which solutions are better. The upper bound for the minimum order + 2 solutions demonstrates the effect of higher levels of perturbation, highlighting its inadequacy as a robustness measure. Comparing the condition number and the true relative perturbation, it can again be seen that generally the condition number selects the better solutions. When the level of perturbation is increased still further to 15% (figures 2.5 and 2.6) the upper bound becomes totally invalid for all solutions. Again the results show a good correlation between the condition number and the true relative perturbation.

Moving on to the second example, the results for a low level of perturbation are shown in figures 2.7 and 2.8. Even at this level of perturbation the upper bound is not valid for all solutions and so should be ignored. Comparing the condition number and the true relative perturbation it can be seen that the correlation between the two is not as good as for the first example. Figures 2.9 and 2.10 show the results with a higher level of perturbation and again the same conclusions can be drawn when comparing the condition number and the true relative perturbation.

Lastly in figures 2.11 and 2.12 the results for the third example are given. As would be expected, due to the high level of perturbation for this example, the upper bound is again invalid. There is a slightly better correlation between the condition number and the true relative perturbation than for example 2, but still not as good as for example 1.

2.6 Conclusions

Having established that the diophantine equation is important in the calculation of a pole-placement controller, this chapter has outlined some important points regarding the equation and obtaining a solution to it. Such an understanding is useful when considering the problem of robustness.

There are two approaches to solving the equation, polynomial methods and matrix methods. Neither appears to have any distinct advantages although the matrix approach seems to be more popular, possibly due to the greater general familiarity with matrix theory, the fact that the matrix representation of the equation is of a standard form and also because the matrix approach is easier to implement on a computer.

A solution to the equation only exists if A_p and B_p are coprime or if the right hand side contains their common factor. The consequence of this statement only becomes apparent when it is understood how common factors can arise for discrete time systems. It is clear that the sample time plays an important role in the occurrence of such factors and so should be chosen carefully.

With exact common factors it is possible to easily detect and overcome their presence, however it is more likely that near common factors will be present which show themselves as ill-conditioning of the matrix equation. Many techniques have been proposed for the case of near common factors but no one method appears to have totally overcome the range of possible problems that could be encountered.

The occurrence of small B_p polynomial coefficients also causes problems when trying to solve the equation, which is important as the magnitude of the B_p coefficients is also a function of sample time. Obtaining a solution clearly suffers from a number of problems but putting them aside, another interesting aspect is the number of possible solutions to the equation.

Any solution will meet the design objective by placing the poles in their desired locations, but different solutions may have interesting properties from the point of view of additional design goals. The goal in this case is to find controllers where the closed-loop transient response is robust to changes in the open-loop model parameters.

From the derivation of an upper bound on the relative perturbation of the solution to the matrix form of the equation, it can be seen that the conditioning of the sylvester matrix is important. An investigation into obtaining better conditioned matrices by changing the order of F_p and G_p is then presented. Although the correlation between the conditioning and the true relative perturbation was not perfect, it is clear that the commonly used minimum order solution is not necessarily the best in this sense.

However this is really only addressing the problem of numerical robustness in the sense that the controller polynomial coefficients will be less affected by changes in the model polynomial coefficients. This is certainly a desirable property to achieve but its effect on the stated goal is difficult to assess. The transient response is dependent on a number of factors, one of which is the poles of the system, i.e. the roots of the characteristic polynomial. Minimising the change in the characteristic polynomial's coefficients does not necessarily minimise the change in the roots (or pole positions). The reason for this is that the relationship between the roots of a polynomial and its coefficients is not a simple one.

It is clear that an alternative approach is needed which can investigate the effect of model parameter perturbations on the factors that directly effect the transient response. There is a growing interest in parameter space methods (Siljak, 1989) which, in the algebraic framework,

relates the changes in the coefficients of a polynomial to changes in the roots. However the transient response is not solely dependent on the pole positions so this approach would not enable a full investigation into transient response robustness.

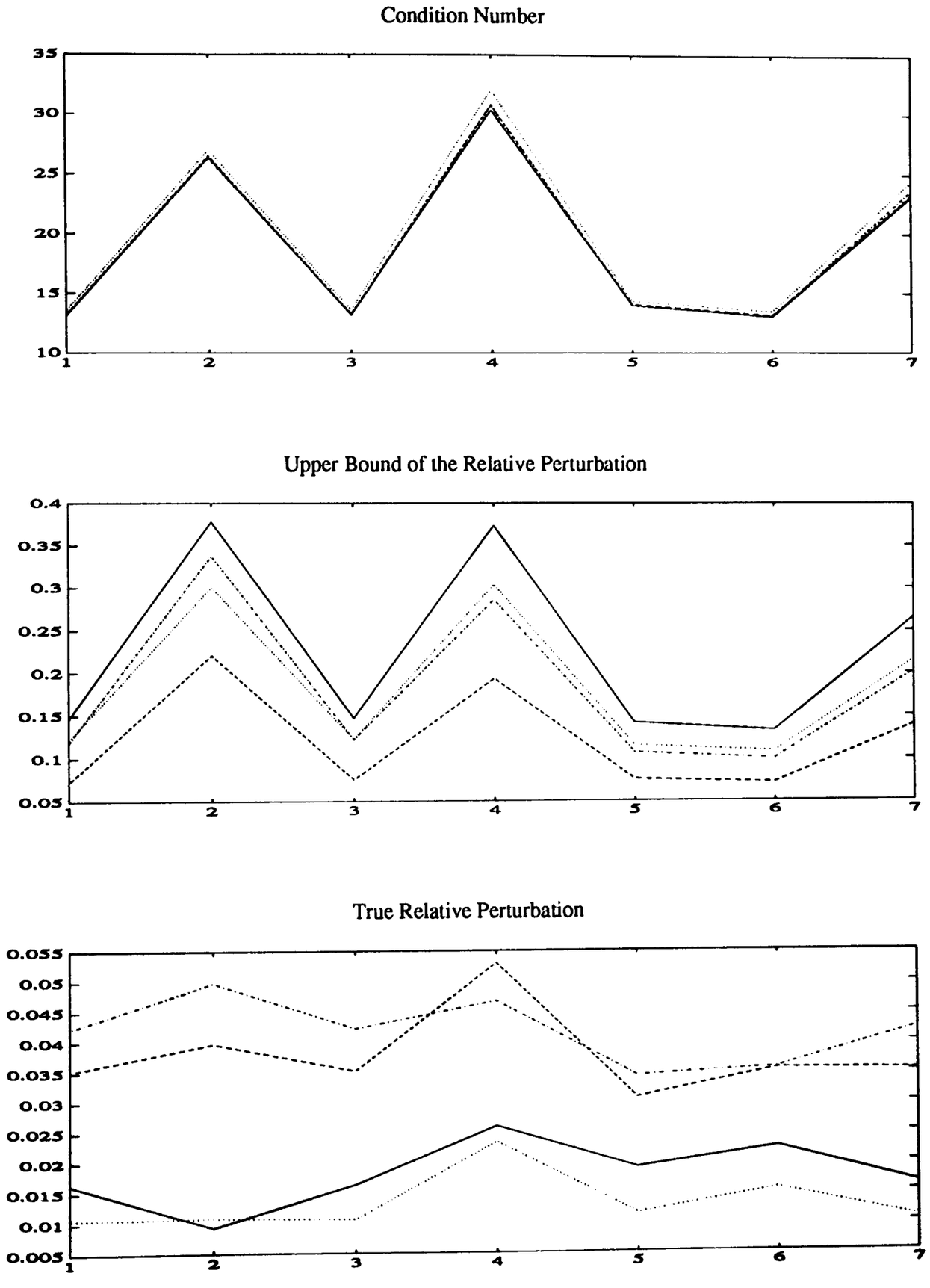
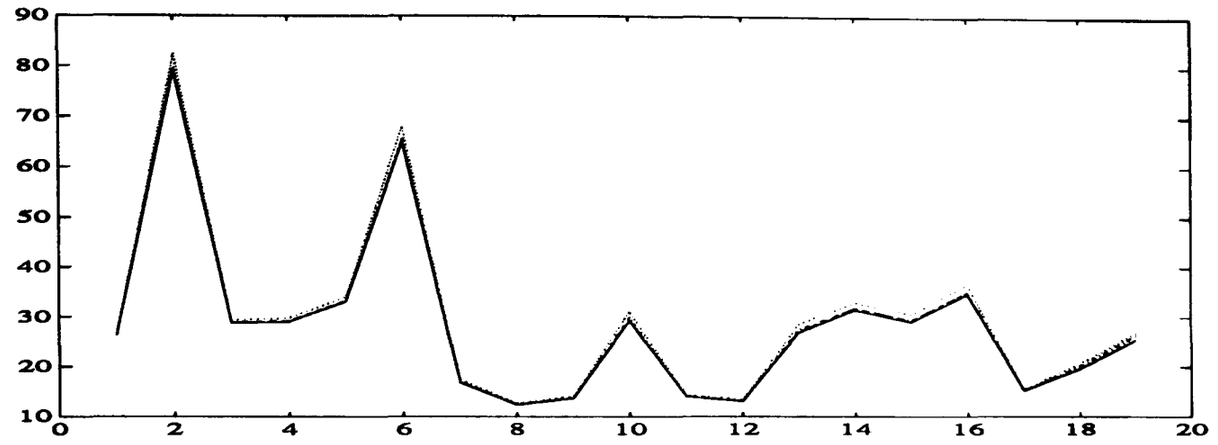
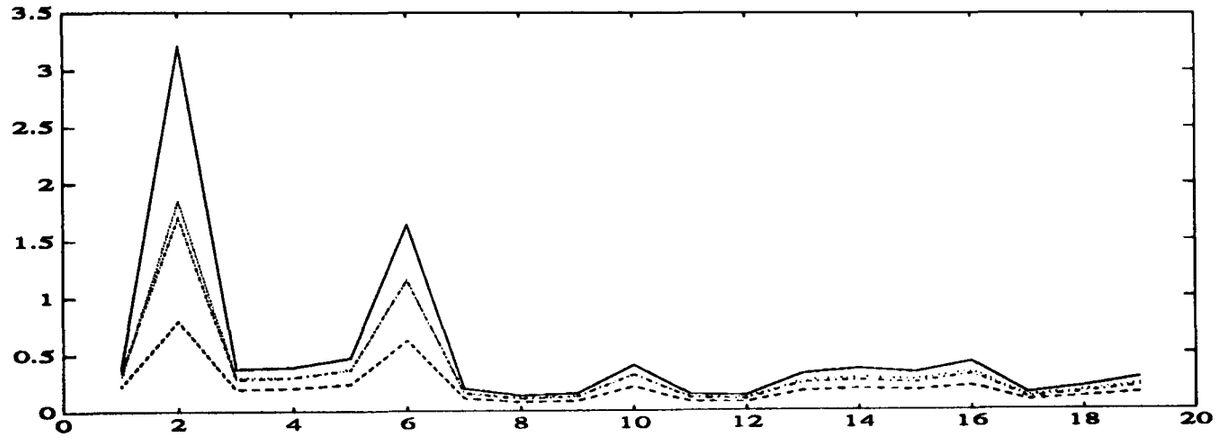


Figure 2.1 - Min Order + 1 Solutions for Example 1 (2% Perturbation)

Condition Number



Upper Bound of the Relative Perturbation



True Relative Perturbation

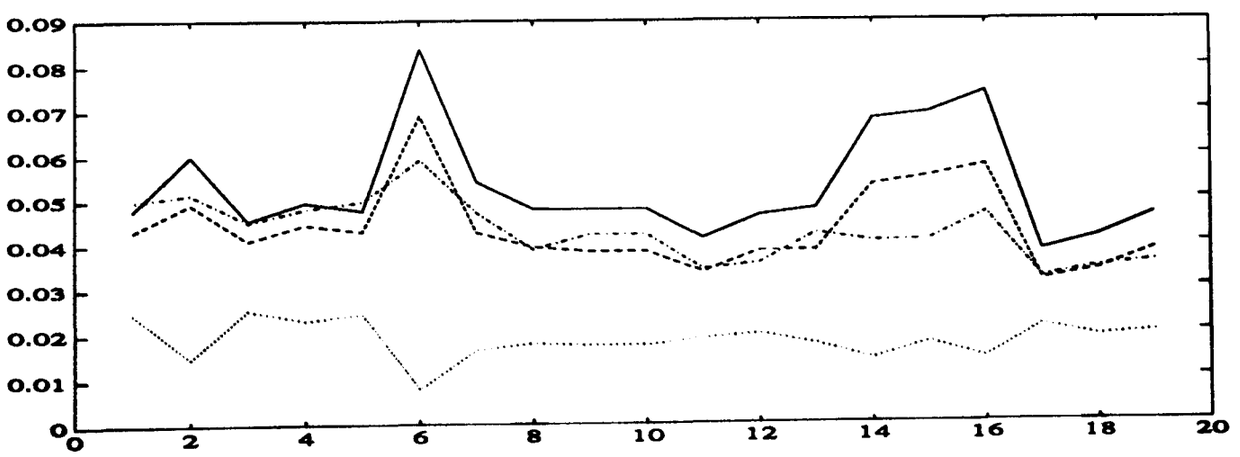


Figure 2.2 - Min Order + 2 Solutions for Example 1 (2% Perturbation)

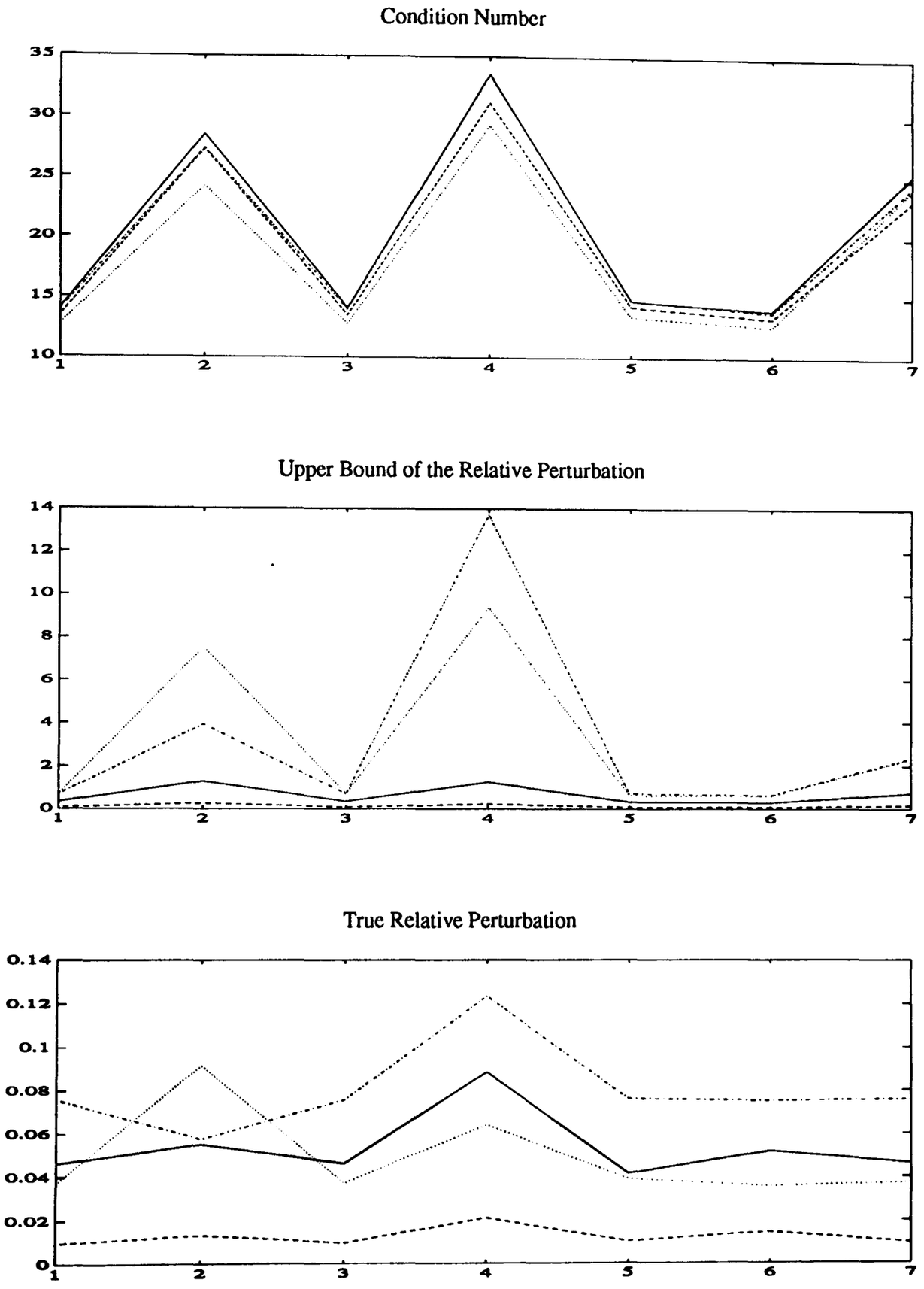


Figure 2.3 - Min Order + 1 Solutions for Example 1 (5% Perturbation)

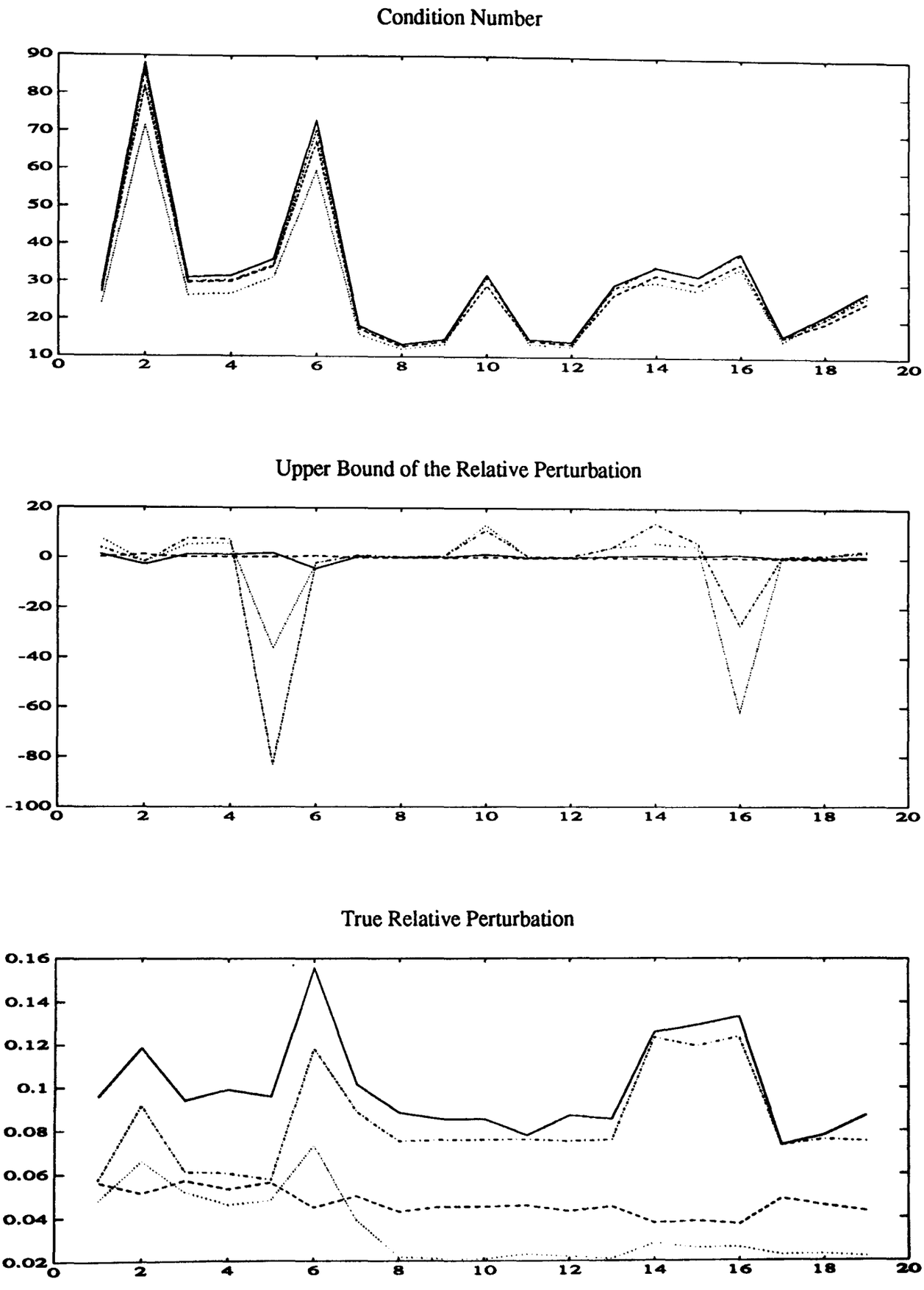


Figure 2.4 - Min Order + 2 Solutions for Example 1 (5% Perturbation)

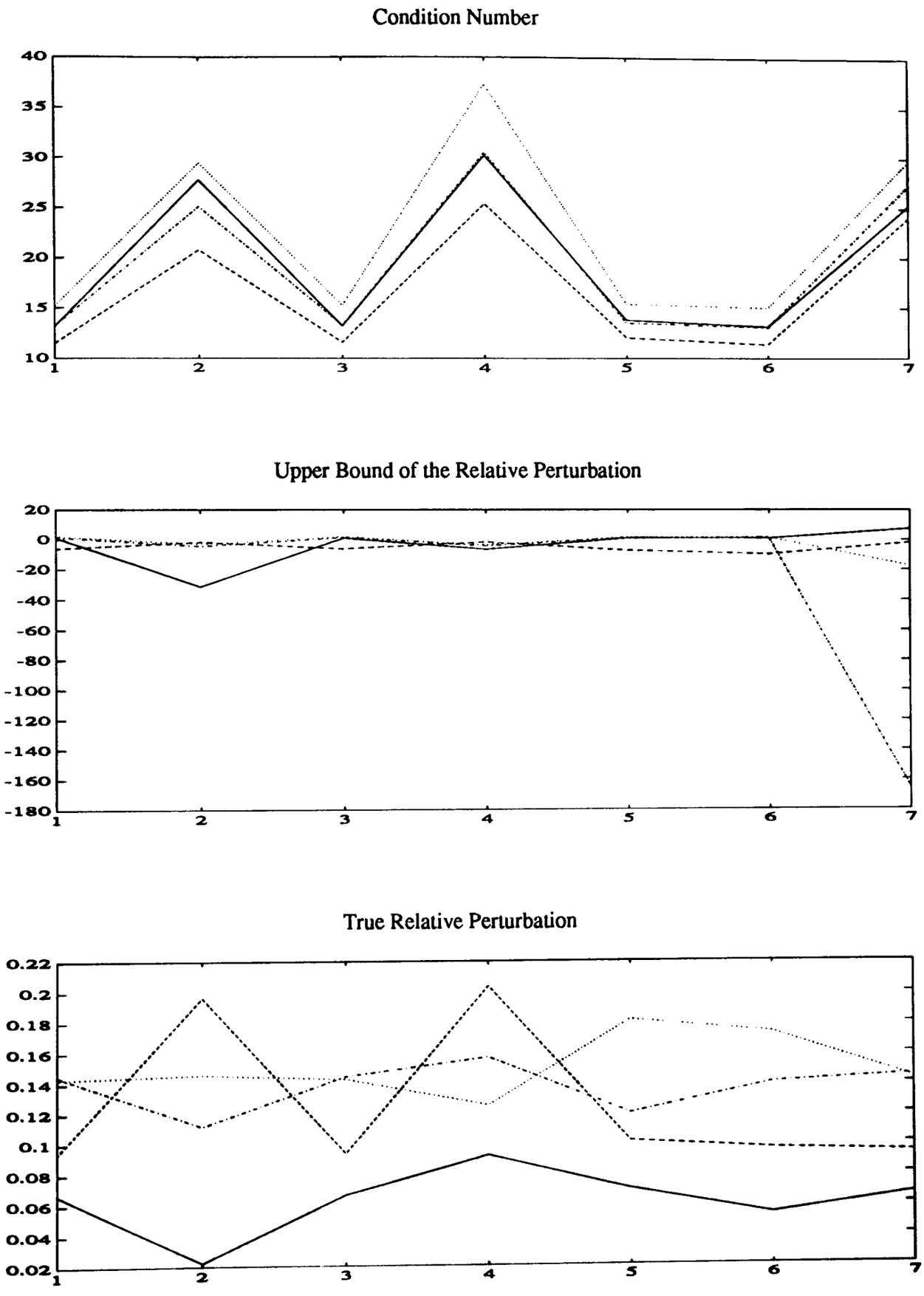


Figure 2.5 - Min Order + 1 Solutions for Example 1 (15% Perturbation)

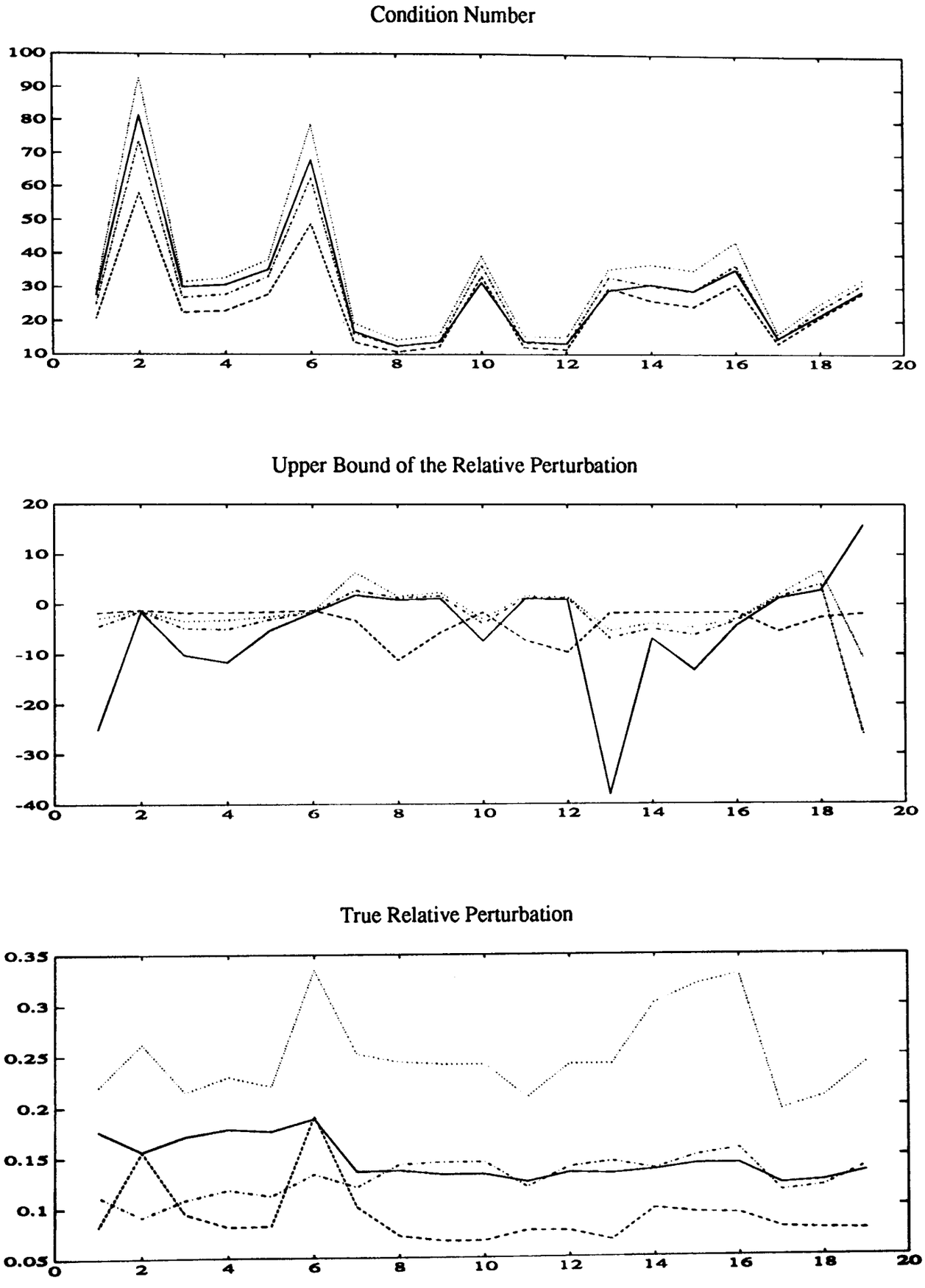


Figure 2.6 - Min Order + 2 Solutions for Example 1 (15% Perturbation)

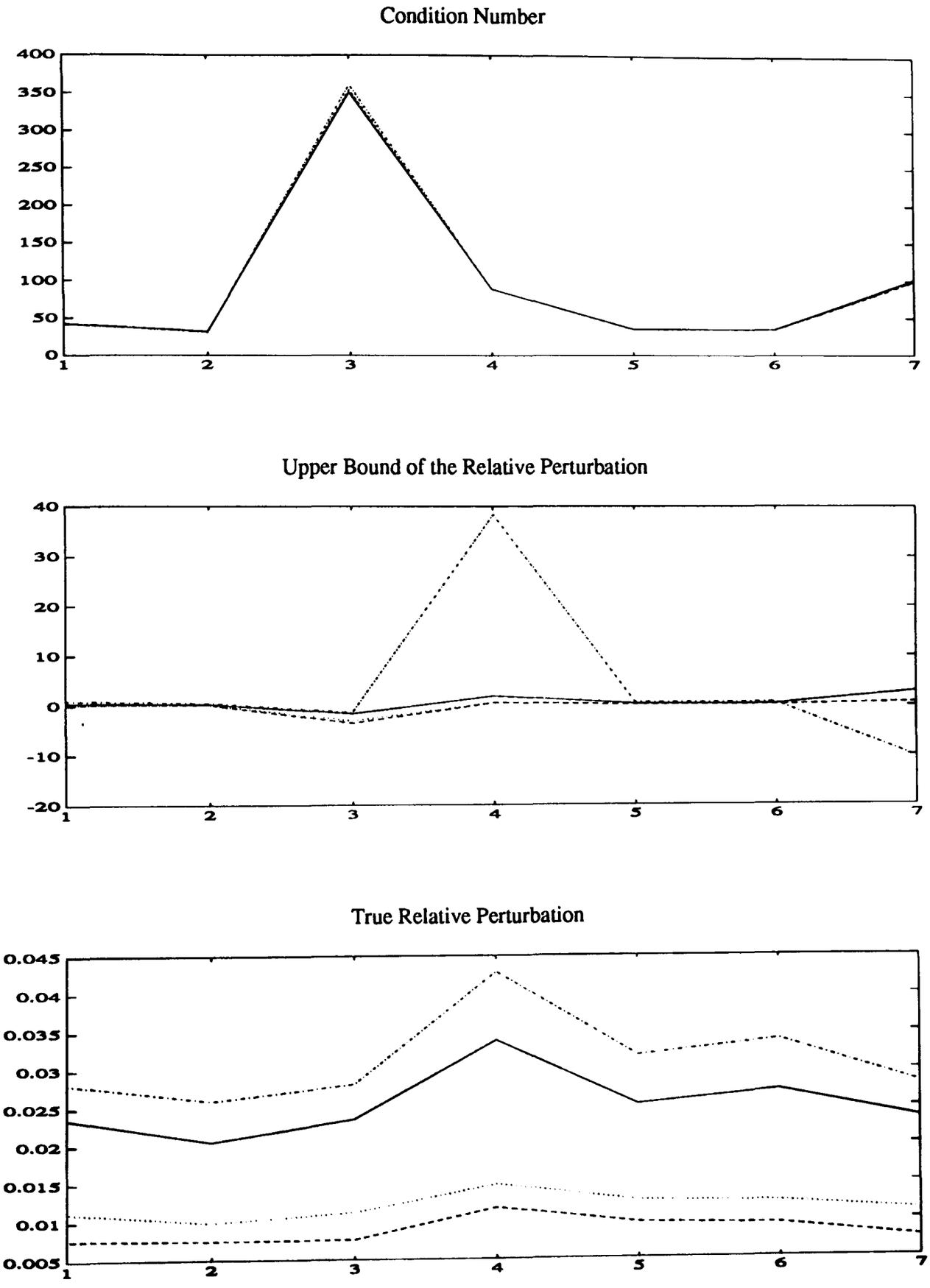


Figure 2.7 - Min Order + 1 Solutions for Example 2 (0.01% & 0.02% Perturbation)

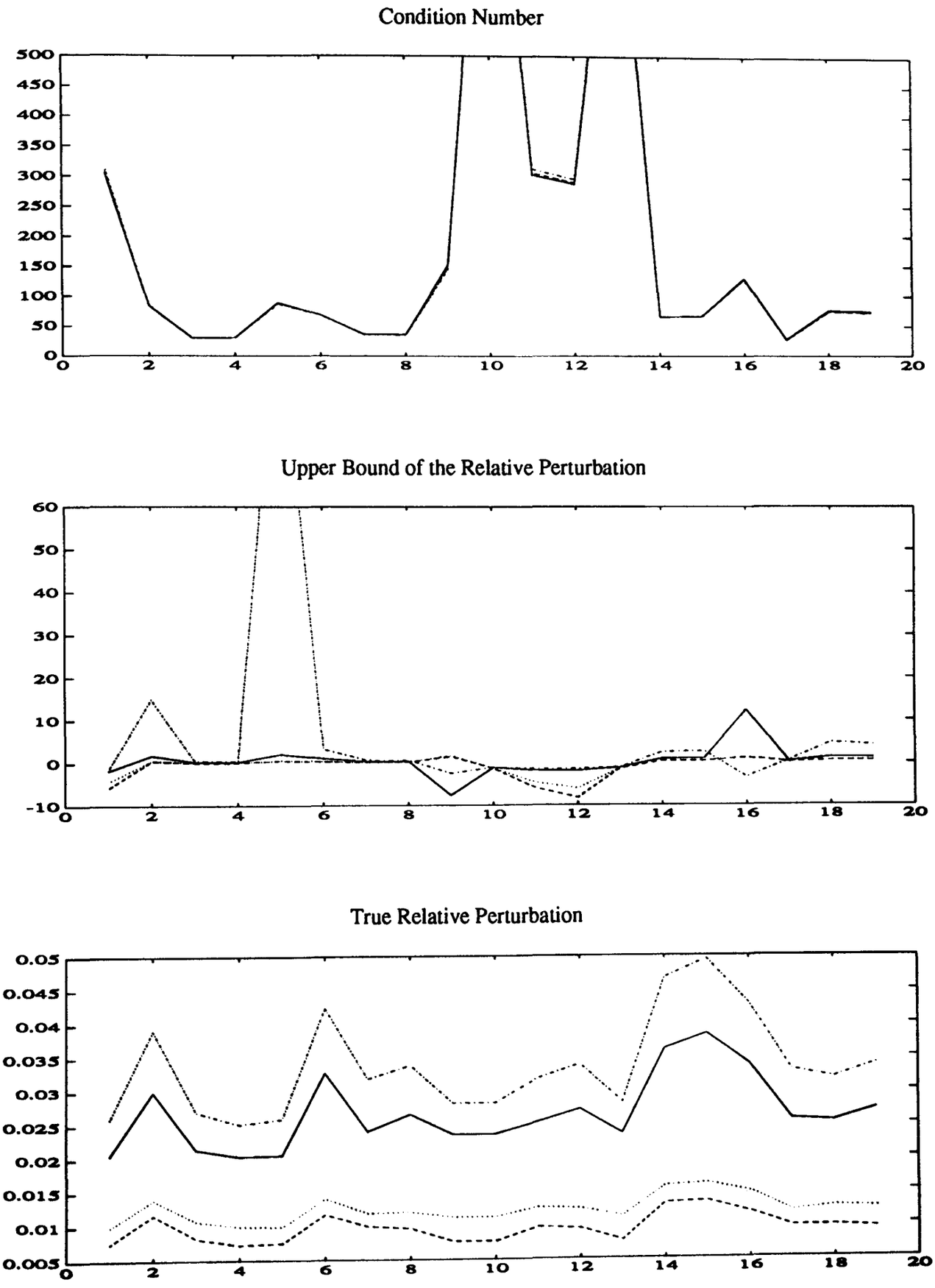


Figure 2.8 - Min Order + 2 Solutions for Example 2 (0.01% & 0.02% Perturbation)

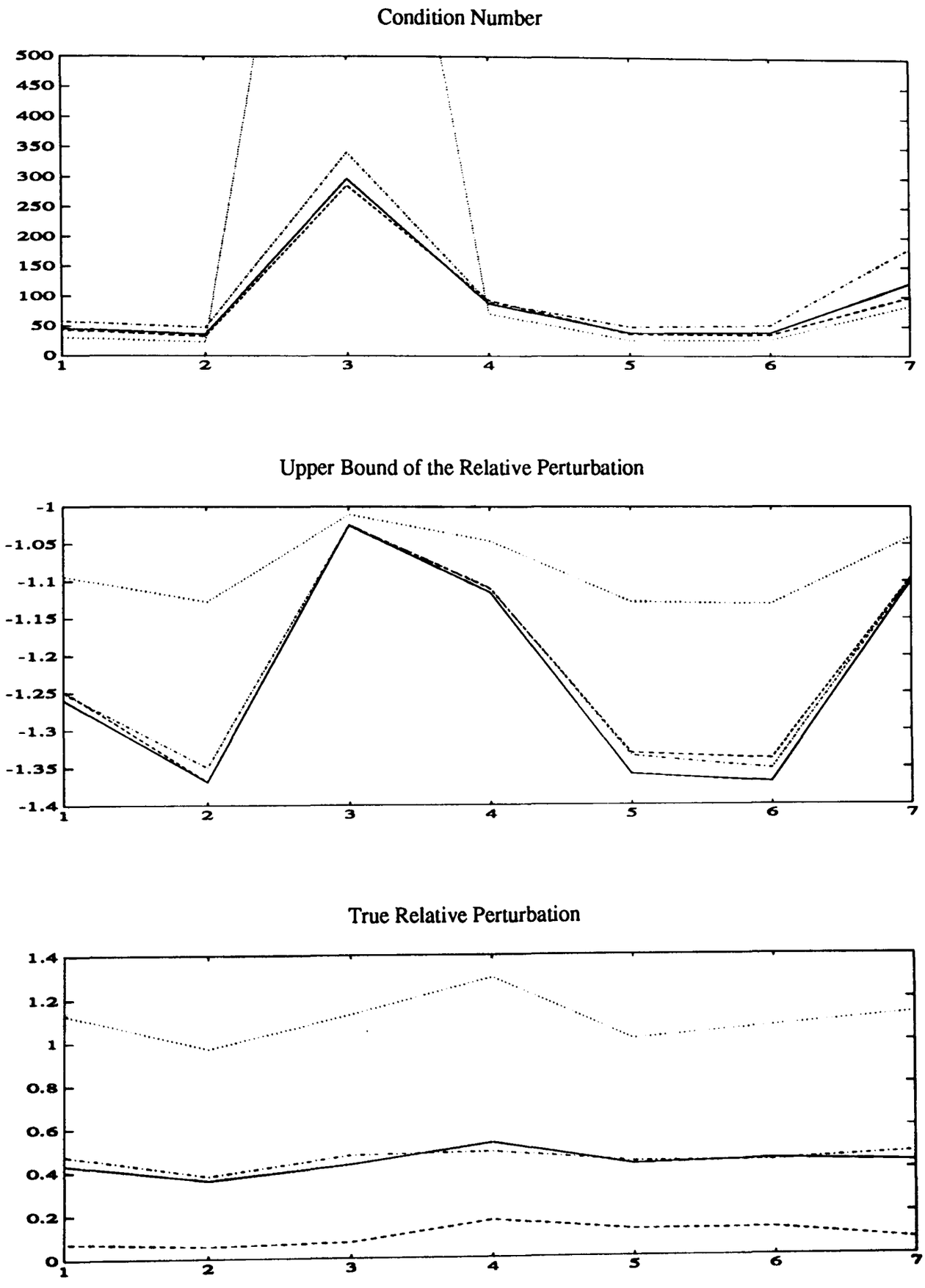


Figure 2.9 - Min Order + 1 Solutions for Example 2 (5% & 10% Perturbation)

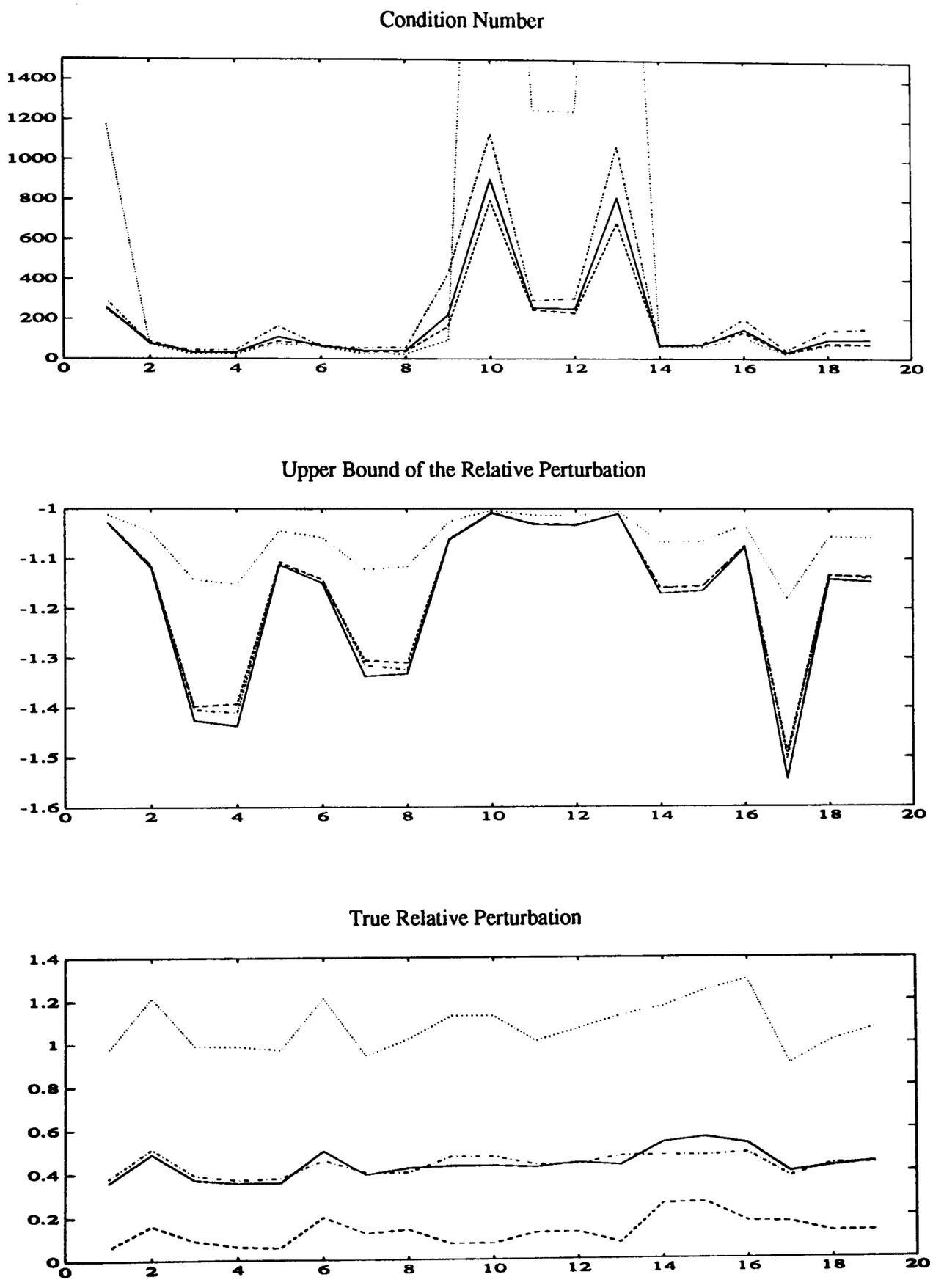


Figure 2.10 - Min Order + 2 Solutions for Example 2 (5% & 10% Perturbation)

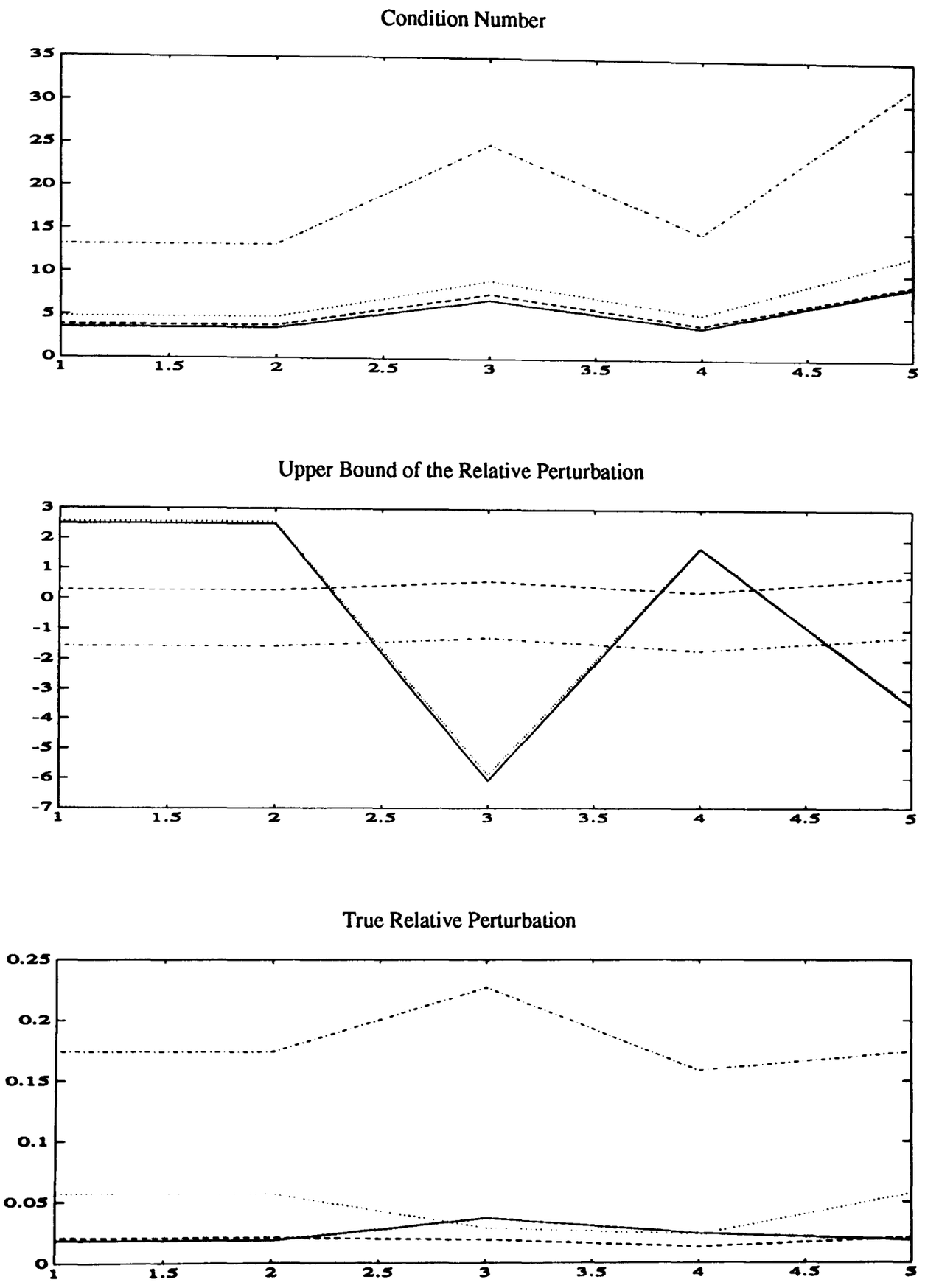


Figure 2.11 - Min Order + 1 Solutions for Example 3

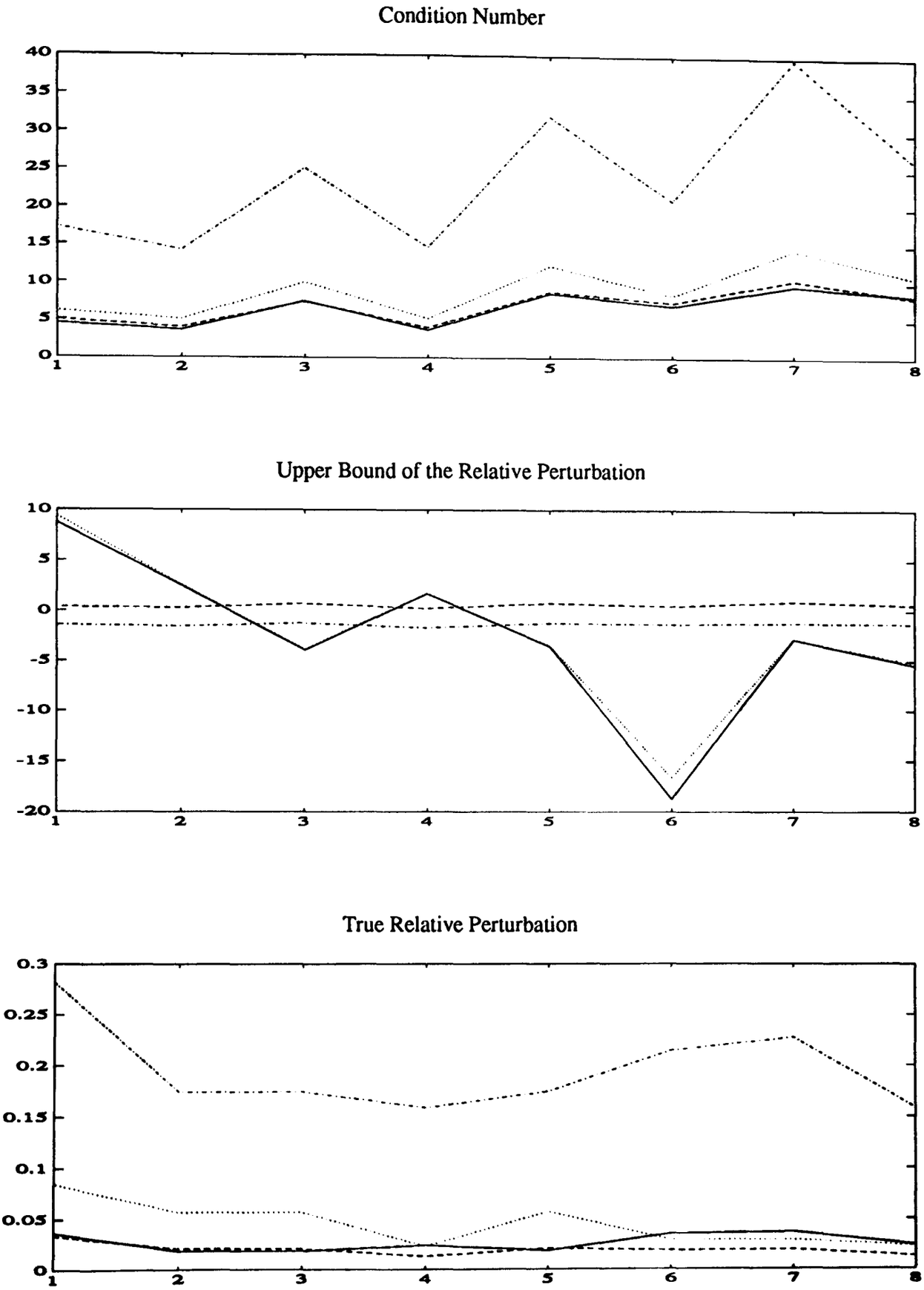


Figure 2.12 - Min Order + 2 Solutions for Example 3

REFERENCES

Astrom, K.J., Hagander, P. and Sternby, J. (1984)
'Zeros of sampled systems'
Automatica, v20, n1, pp 31-38

Astrom, K.J. and Wittenmark, B. (1980)
'Self-Tuning Controllers based on Pole-Zero Placement'
IEE Proceedings, v127, Pt D, n3, pp 120-130

Astrom, K.J. and Wittenmark, B. (1989)
'Adaptive Control'
Addison-Wesley, Wokingham, U.K.

Alix, F., Dion, J.M., Dugard, L. and Landau, I.D. (1982)
'Adaptive Control of Non Minimum Phase Systems Comparison of Several Algorithms and Improvements'
Ricerche Di Automatica, v13, n1, pp 173-189

Berger, C.S. (1984)
'Robust Controller Design by Minimisation of the variation of the Coefficients of the Closed-Loop Characteristic Equation'
IEE Proceedings, v131, Pt D, n3, pp 103-106

Berger, C.S. (1988)
'Robust Pole-Placement Algorithm for Adaptive Control'
IEE Proceedings, v135, Pt D, n6, pp 493-498

Clarke, D.W. (1982)
'Model Following and Pole-Placement Self-Tuners'
Optimal Control Applications and Methods, v3, pp 323-335

Daley, S. (1987)
'Application of a Fast Self-Tuning Control Algorithm to a Hydraulic Test Rig'
Proceedings of the Institution of Mechanical Engineers, v201, n4, pp 285-295

Edmunds, J.M. (1976)
'Digital Adaptive Pole-Shifting Regulators'
PhD Thesis, Control Systems Centre, UMIST, U.K.

Golub, G.H. and Van Loan, C.F. (1983)
'Matrix Computations'
North Oxford Academic Publishing, Oxford, U.K.

Horn, R.A. and Johnson, C.A.(1985)
'Matrix Analysis'
Cambridge University Press, Cambridge, U.K.

Kucera, V. (1979)
'Discrete Linear Control: The Polynomial Equation Approach'
Wiley, Chichester, U.K.

Lancaster, P. and Tismenetsky, M. (1985)
'The Theory of Matrices'
Academic Press, London, U.K.

Lawson, C.L. and Hanson, R.J. (1974)
'Solving Least-Squares Problems'
Prentice-Hall, London, U.K.

McDermott, P.E. and Mellichamp, D.A. (1984)
'An Auto-Pole-Placement Self-Tuning Controller'
International Journal of Control, v40, n6, pp 1131-1147

Middleton, R.H. and Goodwin, G.C. (1986)
'Improved Finite Word Length Characteristics in Digital Control Using Delta Operators'
IEEE Transactions on Automatic Control, v31, n11, pp 1015-1021

Mohtadi, C. (1988)
'Numerical Algorithms in Self-Tuning Control'
Ch 4, 'Implementation of Self-Tuning Controllers', Peregrinus, ed K.Warwick

NAG (1990)
Fortran Library Manual Mk 14
Numerical Algorithms Group Ltd, Oxford, U.K.

Siljak, D.D. (1989)
'Parameter Space Methods for Robust Control Design: A Guided Tour'
IEEE Transactions on Automatic Control, v34, n7, pp 674-688

Tuffs, P.S (1984)
'Self-Tuning Control: Algorithms and Applications'
PhD Thesis, Dept of Eng Science, University of Oxford, U.K.

Warwick, K., Farsi, M. and Karam, K.Z. (1985)
'A Simplified Pole-Placement Self-Tuning Controller'
IEE International Conference on Control'85, Cambridge, U.K. v1 pp 1-6
IEE, London, U.K. (Conf Publ No. 252)

Wellstead, P.E. and Sanoff, S.P. (1981)
'Extended Self-Tuning Algorithm'
International Journal of Control, v34, n3, pp 433-455

CHAPTER 3

STATE SPACE DESIGN FOR POLYNOMIAL SYSTEMS

3.1 Introduction

The previous chapter presented an initial investigation into obtaining controllers with improved performance robustness by utilising alternative solutions to the diophantine equation. The major conclusion was that although extra design freedom exists and it is easy to select alternative controllers, the specification of a suitable robustness measure, on which to base the choice of controller, was quite difficult. The main reason for this is that in the polynomial framework there is no simple approach to assessing the effect of model uncertainty on the response of the closed-loop system.

In a state space framework it is well known that the transient behaviour of a system is governed by the eigenvalues and eigenvectors. Further, this description readily allows access to these factors and hence provides a good basis for an investigation into improving performance robustness. All of these points highlight the main reasons for turning to a state space based approach for the design of robust polynomial controllers. Of course the polynomial description of a system only provides a relationship between the input and output, so if state space methods are to be used to design the controller polynomials it will be necessary to use an output feedback approach.

The state space design needs to provide extra degrees of freedom which can then be utilised as in the polynomial design outlined in the previous chapter to improve performance robustness. There are a number of options and it is worth noting that any robust state space design method could be used to illustrate the overall robust polynomial controller design procedure. Due to the close link between the eigenstructure (eigenvalues and eigenvectors) and the transient response it is natural to consider the techniques of eigenstructure assignment and parametric methods for the state space design.

Of course it is necessary to define a suitable robustness measure on which to base the selection of the extra design freedom. There are a number of possibilities for such a measure, but as already mentioned the eigenvalues and eigenvectors are very important in determining the shape of the transient response and so it is reasonable to expect that they will play an important role in the definition of a suitable measure. A complete discussion of this issue is presented in the following chapter.

In this chapter a more detailed explanation of the relationship between the eigenvalues/eigenvectors and the transient response is given using modal decomposition, hence emphasising the reasons for turning to a state space based design. A discussion of the overall design with the transformation of the polynomial system to state space form and of the controller back to polynomial form is then presented. An introduction to eigenstructure techniques is then given, leading onto a discussion of parametric methods and in particular a review and comparison of two specific parametric output feedback methods, followed by some conclusions on their suitability.

3.2 Modal Decomposition

To understand the factors that effect the transient response, consider the modal decomposition of the discrete linear state space system (Kailath, 1980 and Ogata, 1987)

$$\underline{x}(k+1) = A\underline{x}(k) + B\underline{u}(k) \tag{3.1}$$

$$\underline{y}(k) = C\underline{x}(k) + D\underline{u}(k) \tag{3.2}$$

Assuming the initial state is $\underline{x}(0)$ and the initial input is $\underline{u}(0)$, from equation (3.1)

$$\underline{x}(1) = A\underline{x}(0) + B\underline{u}(0) \tag{3.3}$$

$$\underline{x}(2) = A\underline{x}(1) + B\underline{u}(1)$$

$$= A^2\underline{x}(0) + AB\underline{u}(0) + B\underline{u}(1) \tag{3.4}$$

$$\underline{x}(3) = A\underline{x}(2) + B\underline{u}(2)$$

$$= A^3\underline{x}(0) + A^2B\underline{u}(0) + AB\underline{u}(1) + B\underline{u}(2) \tag{3.5}$$

$$\vdots$$

$$\underline{x}(k) = A^k\underline{x}(0) + \sum_{j=0}^{k-1} A^{k-j-1}B\underline{u}(j) \tag{3.6}$$

It is well known that

$$A\underline{v}_i = \lambda_i\underline{v}_i \tag{3.7}$$

where λ_i are the eigenvalues and $\underline{v}_i$ the right eigenvectors of the matrix A, so clearly

$$AV = V\Lambda \tag{3.8}$$

where $V = [\underline{v}_1 \cdots \underline{v}_n]$ and $\Lambda = \text{diag}[\lambda_1 \cdots \lambda_n]$. Therefore

$$\Lambda = V^{-1}AV \tag{3.9}$$

Clearly A^k can be expressed by

$$A^k = V(V^{-1}AV)^k V^{-1} \tag{3.10}$$

and substituting (3.9) gives

$$A^k = V\Lambda^k V^{-1} \tag{3.11}$$

and as

$$V^{-1} = W = \begin{bmatrix} \underline{w}_1^T \\ \cdot \\ \cdot \\ \underline{w}_n^T \end{bmatrix} \tag{3.12}$$

where $\underline{w}_i^T$ are the left eigenrows of the matrix A,

$$A^k = [\underline{v}_1 \quad \cdot \quad \cdot \quad \underline{v}_n] \begin{bmatrix} \lambda_1 & 0 & \cdot & 0 \\ 0 & \cdot & & \cdot \\ \cdot & & \cdot & 0 \\ 0 & \cdot & 0 & \lambda_n \end{bmatrix} \begin{bmatrix} \underline{w}_1^T \\ \cdot \\ \cdot \\ \underline{w}_n^T \end{bmatrix} \tag{3.13}$$

Thus

$$A^k = \sum_{i=1}^{n} \underline{v}_i \underline{w}_i^T \lambda_i^k \tag{3.14}$$

Clearly equation (3.6) becomes

$$\underline{x}(k) = \sum_{i=1}^{n} \underline{v}_i \underline{w}_i^T \left(\lambda_i^k \underline{x}(0) + \sum_{j=0}^{k-1} \lambda_i^{k-j-1} B \underline{u}(j) \right) \tag{3.15}$$

and the output can be expressed as

$$\underline{y}(k) = C \sum_{i=1}^{n} \underline{v}_i \underline{w}_i^T \left(\lambda_i^k \underline{x}(0) + \sum_{j=0}^{k-1} \lambda_i^{k-j-1} B \underline{u}(j) \right) + D \underline{u}(k) \tag{3.16}$$

Clearly the eigenvalues and eigenvectors are extremely important in determining the shape of the transient response. The eigenvalues affect the rate of decay and the eigenvectors the gain associated with each mode.

Having shown the importance of the eigenvalues and eigenvectors, it is necessary to establish the link between the two representations which will then allow a state space approach to be used in the design of the controller polynomials.

3.3 The Link between Polynomial and State Space Representations

The single-input single-output (SISO) polynomial system defined in equation (1.4) can be expressed in observable canonical form, Ogata (1987), as

$$\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{bmatrix}_{k+1} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & -a_n \\ 1 & 0 & \cdots & 0 & 0 & -a_{n-1} \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ 0 & 0 & \cdots & 1 & 0 & -a_2 \\ 0 & 0 & \cdots & 0 & 1 & -a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{bmatrix}_k + \begin{bmatrix} b_n - a_n b_0 \\ b_{n-1} - a_{n-1} b_0 \\ \cdot \\ \cdot \\ b_2 - a_2 b_0 \\ b_1 - a_1 b_0 \end{bmatrix} u(k) \tag{3.17}$$

$$y(k) = [0 \quad 0 \quad \cdot \quad \cdot \quad \cdot \quad 0 \quad 1] \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} + b_0 u(k) \tag{3.18}$$

where there are n states, $m(=1)$ inputs and $r(=1)$ outputs. Note that it is assumed that $n_a = n_b = n$ which can always be achieved by padding with zero terms if necessary. Also the time delay, t_d will always be ≥ 1 due to sampling and assuming the system is strictly proper, b_0 will always be 0 as the time delay is included in $B_p(z^{-1})$. Hence in the analysis it is unnecessary to consider a D matrix in the state space description.

It is well known that the poles of a closed-loop system may be arbitrarily placed by using state feedback if and only if the system is controllable (Friedland, 1986). For output feedback however it is also required that $r + m > n$, which can be achieved by adding a dynamic compensator of suitable order. Following Brasch and Pearson (1970), for convenience the compensator dynamics will be taken to be a number of integrators. It is also assumed that every state in the compensator is observable and controllable. Therefore the new system with dynamic compensator of order p can be represented as

$$\begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix}_{k+1} = \begin{bmatrix} A & 0_{n\times p} \\ 0_{p\times n} & 0_{p\times p} \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix}_k + \begin{bmatrix} \underline{b} & 0_{n\times p} \\ 0_{p\times 1} & I_p \end{bmatrix} \begin{bmatrix} u_1 \\ \underline{u}_2 \end{bmatrix}_k \tag{3.19}$$

$$\begin{bmatrix} y_1 \\ \underline{y}_2 \end{bmatrix}_k = \begin{bmatrix} \underline{c}^T & 0_{1\times p} \\ 0_{p\times n} & I_p \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix}_k \tag{3.20}$$

where the dimension of $\underline{x}_1$ is $(n \times 1)$, $\underline{x}_2$ is $(p \times 1)$, u_2 and y_2 are $(p \times 1)$. y_1 and u_1 are scalars. $0_{i\times j}$ indicates a zero block of dimension $i \times j$ and I_p indicates a $p \times p$ identity block.

Output feedback can then be applied via the control law

$$\underline{u}(k) = K_y \underline{y}(k) \tag{3.21}$$

In state space form, the problem is then to determine the controller gain matrix K_y which places the closed-loop eigenvalues in the desired locations, and yields a closed-loop system where the transient response is robust to changes in the elements of the A and B matrices.

Once K_y has been obtained via a suitable state space design method it is necessary to interpret it in a polynomial framework. Note that the original input and output are u_1 and y_1 and that the relationship between $\underline{u}_2$ and $\underline{y}_2$ is

$$\underline{y}_2(k) = \underline{u}_2(k-1) \tag{3.22}$$

The control law can also be partitioned in a similar way

$$\begin{bmatrix} u_1 \\ \underline{u}_2 \end{bmatrix}_k = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} y_1 \\ \underline{y}_2 \end{bmatrix}_k \tag{3.23}$$

where the sub blocks of K_y are of appropriate dimension. Rewriting gives

$$u_1(k) = K_{11}y_1(k) + K_{12}\underline{y}_2(k) \tag{3.24}$$

$$\underline{u}_2(k) = K_{21}y_1(k) + K_{22}\underline{y}_2(k) \tag{3.25}$$

From (3.22) and (3.25)

$$\underline{y}_2(k) = (I - K_{22}z^{-1})^{-1}K_{21}z^{-1}y_1(k) \tag{3.26}$$

Hence (3.24) becomes

$$u_1(k) = \left(K_{11} + K_{12}(I - K_{22}z^{-1})^{-1}K_{21}z^{-1}\right)y_1(k) \tag{3.27}$$

Which gives

$$\frac{y_1(k)}{u_1(k)} = \frac{1}{K_{11} + K_{12}(zI - K_{22})^{-1}K_{21}} = -\frac{F_p(z^{-1})}{G_p(z^{-1})} \tag{3.28}$$

Thus giving the expressions for $F_p(z^{-1})$ and $G_p(z^{-1})$. The precompensator polynomial, $H_p(z^{-1})$ is calculated as before for zero steady state error.

3.4 State Space Design

Having shown in the previous sections how closely related the eigenvalues and eigenvectors are to the transient response, and how state space methods can be used in the design of polynomial controllers, the next step is to consider the actual state space design to be used.

The eigenvalues and eigenvectors are clearly very important as regards the transient response, hence it is desirable to consider the placement of not only the eigenvalues but the eigenvectors as well. Such a design procedure is more commonly known as eigenstructure assignment (Burrows, 1990).

Consider the linear time invariant state space representation of a system

$$\underline{\dot{x}} = A\underline{x} + B\underline{u} \tag{3.29}$$

$$\underline{y} = C\underline{x} \tag{3.30}$$

where there are n states, m inputs, r outputs and A, B, C are matrices of appropriate dimensions. The D matrix will be zero due to the nature of the transformation from polynomial to state space form and the time delay of the system, as outlined in section 3.3. Output feedback can be applied to the system via the control law

$$\underline{u} = K\underline{y} \tag{3.31}$$

Eigenstructure assignment can then be described as the assignment of the closed-loop eigenvalues and eigenvectors (either left or right) using the control law (3.31).

Srinathkumar (1978) discusses eigenstructure assignment using output feedback and concludes that $\min(n, m+r-1)$ eigenvalues may be arbitraily placed as well as $(r-1)$ eigenvectors partially assigned with m entries in each vector arbitraily chosen. Clearly for all n eigenvalues to be arbitraily assigned using output feedback, $r+m > n$. This result was originally obtained by Kimura (1975).

Whilst the choice of a closed-loop eigenvalue may be arbitrary, under certain assumptions, the corresponding closed-loop eigenvector is constrained to lie in a subspace of the full state space. This subspace is termed the allowable eigenvector subspace and Burrows (1990) discusses in detail various approaches to determining it and the corresponding controller gain matrix.

Of course having a technique for assigning the whole eigenstructure does open up the question of what is the most suitable position for the eigenvalues and eigenvectors? The concept of placing the eigenvalues (or poles) of a system is extremely well known and the effect on the characteristics of a system for various eigenvalue locations is clearly understood. By selecting appropriate eigenvectors it is hoped to achieve a system with better robustness properties. However, rather than concentrating on the specific values for the eigenvectors it is conceptually more appealing to consider the actual design goal as is the case in parametric methods.

The design procedure is the same as the eigenstructure assignment techniques but the objectives are slightly different. With parametric methods the actual position of the eigenvectors is not of direct concern as the aim is to satisfy additional design objectives, which of course is exactly the situation in this case and as such parametric methods will be investigated further as the basis of the state space design.

There are a number of approaches to the problem of parametric output feedback design. Kalsi (1990) compared the approaches of Roppenecker and O'Reilly (1989) with that of Fahmy and O'Reilly (1988a). Roppenecker and O'Reilly (1989) select the free parameters to ensure that the closed-loop right eigenvectors are orthogonal. They express the controller gain matrix, K, in terms of r free parameter vectors, where r is the number of outputs, and show that the choice of the first $r-1$ vectors is arbitrary as regards the orthogonality condition. They then propose a method for calculating the remaining free parameter vector such that the orthogonality condition holds. However, it may not always be possible to obtain a value for this vector which then requires adjustments to be made to the initial choice of the first $r-1$ vectors and the procedure repeated until a solution is found. The method of Fahmy and O'Reilly (1988a) is essentially a multi-stage design where part of the eigenstructure is assigned with successive feedback loops. Such a procedure requires the previously assigned eigenstructure to be protected against further feedback loops. Fahmy and O'Reilly (1988a) propose four procedures to implement the design: partial eigenvalue/right eigenvector assignment, partial eigenvalue/left eigenvector assignment and two procedures to protect the eigenvalues and eigenvectors (the choice of which procedure depends on whether it is required to protect the left or right eigenvectors).

Kalsi (1991) applied each method to a number of test examples and compared how accurately the closed-loop eigenstructure was assigned. It was concluded that the method of Roppenecker and O'Reilly (1989) had quite poor numerical properties whereas the method of Fahmy and O'Reilly (1988a) assigned the desired eigenstructure quite accurately. The problem of multiple eigenvalues was considered in a third method, Fahmy and O'Reilly (1988b).

Although the method of Fahmy and O'Reilly (1988a) accurately assigns the eigenstructure, the protection part of the design can sometimes cause difficulties. The protection may not be exact or may be extended to include additional eigenvalues that have not yet been correctly assigned.

Daley (1990) has proposed a new scheme which restricts the free parameters of a state feedback approach to yield the output feedback controller. This involves placing constraints on the free parameters which could make their choice quite difficult. However, Daley (1990) has shown that the constraints will be satisfied if the free parameters are selected from the null spaces of various matrices which are relatively simple to generate.

To assess the numerical behaviour of this new method a similar procedure to that of Kalsi (1990) can be adopted and a comparison made with the method of Fahmy and O'Reilly (1988a), which is known to perform well.

This section first reviews the two methods highlighting the major points of each. They are then applied to a number of test examples to see how accurately the closed-loop eigenvalues are placed and whether any problems are experienced in the design of a suitable controller.

3.4.1 The Parametric Output Feedback Method of Fahmy and O'Reilly

Consider a discrete linear time-invariant multivariable system described by

$$\underline{x}_{k+1} = A\underline{x}_k + B\underline{u}_k \tag{3.32}$$

$$\underline{y}_k = C\underline{x}_k \tag{3.33}$$

where $\underline{x} \in R^n$, $\underline{u} \in R^m$ and $\underline{y} \in R^r$, n is the number of states, m the number of inputs and r the number of outputs, with $r + m > n$. A, B and C are matrices of appropriate dimensions.

Fahmy and O'Reilly (1988a) propose a multi-stage approach to the problem of parametric output feedback design, where successive output feedback loops are applied to assign the whole of the eigenstructure. It is necessary to protect previously assigned eigenvalues and eigenvectors from the effect of subsequent output feedback loops.

The eigenstructure is assigned using two procedures, partial eigenvalue-right eigenvector assignment and partial eigenvalue-left eigenvector assignment. The procedures are used in sequence, hence the multi-stage nature of this design approach. After the first stage the partially assigned eigenstructure should be protected using either an input reduction matrix or an output reduction matrix.

First consider the two cases of partial eigenstructure assignment. In each case the application of output feedback in the form $u(t) = Ky(t)$ is considered to assign a part of the closed-loop system eigenstructure.

i) Partial eigenvalue - right eigenvector assignment

Consider a subset $\Lambda_s = \{\lambda_1, \cdots, \lambda_s\}$, where $s \leq r$ of the closed-loop eigenvalue set Λ_n. Note that the specified closed-loop eigenvalues should all be different from the open-loop eigenvalues. The matrices K and C are partitioned as

$$K = [K_{11} \quad K_{12}], \qquad C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} \tag{3.34}$$

where K_{11} is an $m \times s$ matrix and C_1 a $s \times n$ matrix. Fahmy and O'Reilly (1988a) obtain the equation

$$KCV_s = F_s \tag{3.35}$$

where V_s is the matrix of the first s right eigenvectors defined as

$$\underline{v}_i = \text{adj}[\lambda_i I - A]B\underline{f}_i \tag{3.36}$$

and F_s is defined as

$$F_s = [\underline{f}_1' \cdots \underline{f}_s'] \tag{3.37}$$

where $\underline{f}_i' = |\lambda_i I - A|\underline{f}_i$ and $\underline{f}_i$ are the free parameter vectors.

Equation (3.35) can be solved for K_{11}, giving

$$K_{11} = [F_s - K_{12}C_2V_s][C_1V_s]^{-1} \tag{3.38}$$

where K_{12} is arbitrary, and it is suggested that it is taken as a zero matrix as its role is cancelled out when considering the whole eigenstructure. If $s = r$ then

$$K_{11} = F_s[C_1 V_s]^{-1} \tag{3.39}$$

The free parameter vectors f_i are chosen under the following conditions

(i) $|C_1 V_s| \neq 0$

(ii) $f_i \in R^r$ for a real eigenvalue λ_i

(iii) $f_j = f_i \in R^r$ or $f_j = f_i^* \in C^r$ for a complex conjugate pair of eigenvalues λ_i. $\lambda_j = \lambda_i^*$

[]

ii) Partial eigenvalue - left eigenvector assignment

Take $s \leq m$. The matrices K and B are partitioned as

$$K = \begin{bmatrix} K_{11} \\ K_{12} \end{bmatrix}, \quad B = [B_1 \quad B_2] \tag{3.40}$$

where K_{11} is an $s \times r$ matrix and B_1 a $n \times s$ matrix. This gives the equation

$$W_s B K = G_s \tag{3.41}$$

where W_s is the matrix of the first s left eigenrows defined as

$$\underline{w}_i^T = \underline{g}_i^T C \,\mathrm{adj}[\lambda_i I - A] \tag{3.42}$$

and G_s is defined as

$$G_s = \begin{bmatrix} |\lambda_1 I - A| \underline{g}_1^T \\ \vdots \\ |\lambda_s I - A| \underline{g}_s^T \end{bmatrix} \tag{3.43}$$

with $\underline{g}_i^T$ being the free parameter vectors.

Setting the K_{12} to a zero matrix, equation (3.41) can be solved for K_{11}, giving

$$K_{11} = [W_s B_1]^{-1} G_s \tag{3.44}$$

and the free parameter vectors $\underline{g}_i^T$ can be chosen under similar conditions as those for $\underline{f}_i$.

[]

Using either i) or ii) K can be determined to partially assign the eigenstructure. The system is then

$$\underline{x}_{k+1} = (A + BKC)\underline{x}_k + B\underline{u}_k \tag{3.45}$$

$$\underline{y}_k = C\underline{x}_k \tag{3.46}$$

on which subsequent calculations are performed.

As previously mentioned it is necessary to protect parts of the eigenstructure which have already been assigned when considering the application of another feedback loop. This can be achieved in one of two ways.

iii) Protection of the assigned eigenstructure using an input reduction matrix

Consider the system in (3.32) and (3.33), an eigenvalue λ_i can be made uncontrollable by choosing an input reduction matrix $\bar{B}$ such that

$$\underline{w}_i^T B \bar{B} = 0 \tag{3.47}$$

The system then becomes

$$\underline{x}_{k+1} = A\underline{x}_k + B\tilde{B}\underline{u}_k \tag{3.48}$$

$$\underline{y}_k = C\underline{x}_k \tag{3.49}$$

and the eigenvalue will be invariant under output feedback. It is then shown, in Fahmy and O'Reilly (1988a), that the left eigenrow associated with this eigenvalue will also be invariant under output feedback.

If an output feedback matrix $\tilde{K}$ is then determined for this system, the true K will be given by

$$K = \tilde{B}\tilde{K} \tag{3.50}$$

[]

iv) Protection of the assigned eigenstructure using an output reduction matrix

Consider the system in (3.32) and (3.33), an eigenvalue λ_i can be made unobservable by choosing an output reduction matrix $\tilde{C}$ such that

$$\tilde{C}C\underline{v}_i = 0 \tag{3.51}$$

The system then becomes

$$\underline{x}_{k+1} = A\underline{x}_k + B\underline{u}_k \tag{3.52}$$

$$\underline{y}_k = \tilde{C}C\underline{x}_k \tag{3.53}$$

and the eigenvalue and associated right eigenvector will be invariant under output feedback.

If an output feedback matrix $\tilde{K}$ is then determined for this system, the true K will be given by

$$K = \tilde{K}\tilde{C} \tag{3.54}$$

[]

The approach for a two stage design is then as follows.

1. Divide the self-conjugate set of eigenvalues Λ_n into two sets of self-conjugate sets of eigenvalues, Λ_s and Λ_t. The upper bound on s is dependent on whether the right or left eigenvectors are to be assigned first, see i) and ii) above.
2. Use either method i) or ii) to determine an output feedback matrix K_1 that partially assigns the first s eigenvalues and eigenvectors.
3. If the left {right} eigenvectors were assigned then the eigenstructure must be protected as outlined in iii) {iv)}.
4. If method i) {ii)} was used to assign the first part of the eigenstructure then use the approach of ii) {i)} to determine K_2 which will assign the remaining eigenstructure.
5. The overall controller gain matrix K is then obtained as

$$K_y = K_1 + K_2 \tag{3.55}$$

3.4.2 The Parametric Output Feedback Method of Daley

Considering the system in (3.32) and (3.33), Daley (1990) shows that the application of output feedback, $\underline{u}_k = K_y \underline{y}_k$, can be achieved using state feedback, $\underline{u}_k = K_x \underline{x}_k$, by ensuring that $K_x = [K_y \quad : \quad 0]$, where K_x is an $m \times n$ matrix and K_y a $m \times r$ matrix. Note that C must be of the form $C = [I \quad : \quad 0]$, where I is a $r \times r$ identity matrix. This does not represent a severe restriction, as the required form for C can be easily achieved by a state transformation. In this case due to the structure of C in the observable canonical form, a simple re-ordering of the states of the system and augmented dynamic compensator will achieve the desired form for C.

The general form for the state space system is then

$$\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_{n+p} \end{bmatrix}_{k+1} = \begin{bmatrix} -a_1 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ \cdot & \cdot & & \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & & \cdot & \cdot \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -a_n & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -a_{n-1} & 0 & \dots & 0 & 1 & \dots & 0 & 0 \\ \cdot & \cdot & & \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & & \cdot & \cdot \\ -a_2 & 0 & \dots & 0 & 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_{n+p} \end{bmatrix}_k + \begin{bmatrix} b_1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 0 & 0 & \dots & 1 \\ b_n & 0 & \dots & 0 \\ b_{n-1} & 0 & \dots & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ b_2 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ u_{p+1} \end{bmatrix}_k \tag{3.56}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_{p+1} \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_{n+p} \end{bmatrix}_k \tag{3.57}$$

State feedback can be applied via the control law

$$\underline{u}(k) = K_x \underline{x}(k) \tag{3.58}$$

and the controller gain matrix, K_x can be parameterised as

$$K_x = [\underline{f}_1 \quad \underline{f}_2 \quad \cdots \quad \underline{f}_n][\underline{v}_1 \quad \underline{v}_2 \quad \cdots \quad \underline{v}_n]^{-1} = FV^{-1} \tag{3.59}$$

where F is a $m \times n$ matrix of free parameters and V is a $n \times n$ matrix of closed-loop right eigenvectors.

K_x will assign a specified set of distinct closed-loop eigenvalues, $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$, provided that the closed-loop eigenvectors satisfy

$$\underline{v}_i = (\lambda_i I - A)^{-1} B \underline{f}_i \tag{3.60}$$

and the pair $[A, B]$ is fully controllable. The inverse in (3.60) exists provided that the closed-loop eigenvalues are all different from the open-loop eigenvalues. If

$$F = [F_1 \quad : \quad F_2] \tag{3.61}$$

where the dimensions of F_1 and F_2 are $m \times r$ and $m \times (n-r)$ respectively and

$$V^{-1} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}^{-1} = W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \tag{3.62}$$

where W is the matrix of closed-loop left eigenrows and the dimensions of V_{11}, W_{11} are $r \times r$; V_{12}, W_{12} are $r \times (n-r)$; V_{21}, W_{21} are $(n-r) \times r$ and V_{22}, W_{22} are $(n-r) \times (n-r)$, then in order for

$$K_x = [K_y \quad : \quad 0] \tag{3.63}$$

F_1 and F_2 must be constrained such that

$$F_1 W_{12} + F_2 W_{22} = 0 \tag{3.64}$$

It is then shown in Daley (1990) that the constraint in (3.64) will always be satisfied if the free parameter vectors in F_2 satisfy

$$\left[I - F_1 V_{11}^{-1} [(\lambda_j I - A)^{-1} B]_r \right] f_j = 0 \tag{3.65}$$

where $j = r+1 \rightarrow n$ and $[\cdot]_r$ denotes the first r rows of the matrix. For there to be non-trivial solutions to equation (3.65) the dimension of the null-space of $\left[I - F_1 V_{11}^{-1} [(\lambda_j I - A)^{-1} B]_r \right]$ must be non-zero, which will be the case if

$$\zeta \Gamma = 0 \tag{3.66}$$

where

$$\zeta = \begin{bmatrix} \alpha_{11}\Delta S_{11} & \alpha_{12}\Delta S_{12} & \cdots & \alpha_{1r}\Delta S_{1r} \\ \alpha_{21}\Delta S_{21} & \alpha_{22}\Delta S_{22} & \cdots & \alpha_{2r}\Delta S_{2r} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha_{q1}\Delta S_{q1} & \alpha_{q2}\Delta S_{q2} & \cdots & \alpha_{qr}\Delta S_{qr} \end{bmatrix}, \quad \underline{\Gamma} = \begin{bmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ f_r \end{bmatrix} \tag{3.67}$$

$\alpha_{11}, \alpha_{12}, \cdots, \alpha_{qr}$ are arbitrary scalar parameters, $q = n - r$ and

$$\Delta S_{ik} = [((\lambda_{i+r} I - A)^{-1} - (\lambda_k I - A)^{-1})B]_r \tag{3.68}$$

Thus selecting the free parameters using (3.65) and (3.66) allows an output feedback controller gain matrix K_y to be determined using (3.59) and (3.60).

3.4.3 Examples used for the Comparison of the Methods

In Kalsi (1991) three examples are considered.

Example 1 - Topaloglu and Seborg (1975)

$$A = \begin{bmatrix} -3.0 & 2.0 & 0 \\ 4.0 & -5.0 & 1.0 \\ 0 & 0 & -3.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1.0 \\ 1.0 & 0 \\ 0 & 1.0 \end{bmatrix}, \quad C = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0 \end{bmatrix} \tag{3.69}$$

$n = 3, m = 2$ and $r = 2$. The desired closed-loop eigenvalues are $\{-10, -9, -8\}$.

Example 2 - Owens (1988)

$$A = \begin{bmatrix} -0.5 & 0 & 0 & 0 & 0 \\ 0 & -8.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 2.0 \end{bmatrix}, \quad B = \begin{bmatrix} 4.0 & 1.0 & 2.0 \\ 3.0 & 4.0 & 0 \\ 0 & 2.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 2.0 & 1.0 & 3.0 \end{bmatrix},$$

$$C = \begin{bmatrix} 4.0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 2.0 & 0 \\ 0 & 0 & 1.0 & 0 & 1.0 \end{bmatrix} \tag{3.70}$$

$n = 5$, $m = 3$ and $r = 3$. The desired closed-loop eigenvalues are $\{-1, -2, -3, -5 \pm j5\}$.

Example 3 - Sobel and Shapiro (1985)

$$A = \begin{bmatrix} -20 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -25 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 \\ -0.744 & -0.032 & 0 & -0.154 & -0.0042 & 1.54 & 0 \\ 0.337 & -1.12 & 0 & 0.249 & -1.0 & -5.2 & 0 \\ 0.02 & 0 & 0.0386 & -0.996 & -0.0003 & -0.117 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & -0.5 \end{bmatrix},$$

$$B = \begin{bmatrix} 20 & 0 \\ 0 & 25 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 0 & 1.0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.71}$$

$n = 7$, $m = 2$ and $r = 4$. The desired closed-loop eigenvalues are $\{-1.5 \pm j1.5, -2 \pm j, -17, -22, -0.7\}$.

For the purposes of this comparison a fourth example is also considered here

Example 4 - Fahmy and O'Reilly (1988a)

$$A = \begin{bmatrix} 1.0 & 0 & 1.0 & 1.0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 1.0 & 1.0 \end{bmatrix}, \quad B = \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \\ 1.0 & 1.0 \\ 0 & 1.0 \end{bmatrix}, \quad C = \begin{bmatrix} 1.0 & 1.0 & 0 & 1.0 \\ 0 & 0 & 1.0 & 0 \\ 1.0 & 0 & 1.0 & 0 \end{bmatrix} \tag{3.72}$$

$n = 4$, $m = 2$ and $r = 3$. The desired closed-loop eigenvalues are $\{-1 \pm j0.5, -3, -4\}$.

The approaches of sections 3.5.1 and 3.5.2 are used to design an output feedback matrix, K_y for each of the four examples. Although a direct comparison cannot really be made as Fahmy and O'Reilly (1988a) is a multi-stage design, it is possible to compare how accurately the closed-loop eigenvalues are placed. This will then give an indication of the numerical properties

of each method. It was found in Kalsi (1991) that the approach of Roppenecker and O'Reilly (1987) sometimes placed eigenvalues quite inaccurately and so it was concluded that the method may exhibit poor numerical properties.

For example 3, the condition of $r+m>n$ (required by both methods) is not satisfied so it is necessary to augment the system with a dynamical compensator of suitable order. In this case a 2nd order compensator will satisfy the condition, hence the system is now

$$A=\begin{bmatrix} -20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ -0.744 & -0.032 & 0 & -0.154 & -0.0042 & 1.54 & 0 & 0 & 0 \\ 0.337 & -1.12 & 0 & 0.249 & -1.0 & -5.2 & 0 & 0 & 0 \\ 0.02 & 0 & 0.0386 & -0.996 & -0.000295 & -0.117 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$B=\begin{bmatrix} 20 & 0 & 0 & 0 \\ 0 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}, \quad C=\begin{bmatrix} 0 & 0 & 0 & 1.0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix} \tag{3.73}$$

The additional closed-loop eigenvalues will be placed at {-30, -35}. Such a choice should ensure that the augmented eigenvalues have a minimal effect on the transient behaviour of the closed-loop system.

For examples 2,3 and 4, C is not in the required form for the method of Daley (1990). Hence it is necessary to apply a state transformation to each system.

In the case of example 2, the state transformation matrix was found to be

$$T=\begin{bmatrix} 0.25 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & -0.9 & 0 \\ 0 & 0 & 0.5 & 0 & -0.7 \\ 0 & 0.4 & 0 & 0.45 & 0 \\ 0 & 0 & 0.5 & 0 & 0.7 \end{bmatrix} \tag{3.74}$$

hence

$$A' = T^{-1}AT = \begin{bmatrix} -0.5 & 0 & 0 & 0 & 0 \\ 0 & -0.8 & 0 & 8.1 & 0 \\ 0 & 0 & 1.0 & 0 & 1.4 \\ 0 & 1.6 & 0 & -6.2 & 0 \\ 0 & 0 & 0.7143 & 0 & 1.0 \end{bmatrix},$$

$$B' = T^{-1}B = \begin{bmatrix} 16.0 & 4.0 & 8.0 \\ 5.0 & 6.0 & 2.0 \\ 2.0 & 3.0 & 4.0 \\ -2.2222 & -3.1111 & 0.4444 \\ 1.4286 & -0.7143 & 1.4286 \end{bmatrix},$$

$$C' = CT = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 \end{bmatrix} \tag{3.75}$$

For example 3

$$T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & -0.5 \\ 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & -0.5 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 \end{bmatrix} \tag{3.76}$$

giving

$$A' = T^{-1}AT = \begin{bmatrix} -0.577 & -0.0042 & 1.54 & 0 & 0 & 0 & -0.5978 & -0.032 & -0.4438 \\ 0.1245 & -1.0 & -5.2 & 0 & 0 & 0 & 0.3604 & -1.12 & 0.1114 \\ -0.498 & -0.0003 & -0.117 & 0.0386 & 0 & 0 & -0.484 & 0 & 0.512 \\ 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2115 & -0.0021 & 0.77 & 0 & 0 & 0 & -10.2989 & -0.016 & -10.2219 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -25.0 & 0 \\ -0.2115 & 0.0021 & -0.77 & 0 & 0 & 0 & -9.7011 & 0.016 & -9.7781 \end{bmatrix},$$

$$B' = T^{-1}B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \\ 14.2857 & 0 & 0 & 0 \\ 0 & 25.0 & 0 & 0 \\ 14.2857 & 0 & 0 & 0 \end{bmatrix},$$

$$C' = CT = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 \end{bmatrix} \tag{3.77}$$

and for example 4

$$T = \begin{bmatrix} 0 & -1.0 & 1.0 & 0 \\ 1.0 & 1.0 & -1.0 & -1.0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \tag{3.78}$$

which gives

$$A' = T^{-1}AT = \begin{bmatrix} 1.0 & 2.0 & 0 & 1.0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 1.0 & 1.0 & 1.0 \\ 0 & 1.0 & 0 & 1.0 \end{bmatrix}, \quad B' = T^{-1}B = \begin{bmatrix} 1.0 & 2.0 \\ 1.0 & 1.0 \\ 2.0 & 1.0 \\ 0 & 1.0 \end{bmatrix},$$

$$C' = CT = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \end{bmatrix} \tag{3.79}$$

3.4.4 Results for the Method of Fahmy and O'Reilly

In each example the free parameters f_i in eqn (3.37) were chosen randomly

Example 1

The output feedback matrix

$$K_y = \begin{bmatrix} -62.3913 & 89.0 \\ -105.0 & 159.0 \end{bmatrix} \tag{3.80}$$

was obtained which placed the closed-loop eigenvalues at

-9.99999999997717
-9.00000000003750
-7.99999999998551

Example 2

The output feedback matrix

$$K_y = \begin{bmatrix} 7.0749 & -3.2769 & -23.9639 \\ 7.5643 & -3.5401 & -25.5432 \\ -1.9588 & 1.5799 & 5.1844 \end{bmatrix} \tag{3.81}$$

was obtained which placed the closed-loop eigenvalues at

$-5.00000000000000 \pm j4.99999999999994$
-3.00000000000041
-1.99999999999951
-1.00000000000016

Example 3

The output feedback matrix

$$K_y = \begin{bmatrix} 3.1412 & -0.5958 & 0.8563 & 0.2503 & 5.7946 & -0.7882 \\ 0.5197 & -0.2009 & 24.2762 & 7.4052 & 31.2904 & -4.6199 \\ 6.6958 & 9.9175 & -143.9334 & -14.7714 & -143.9688 & 18.0386 \\ 32.3750 & 46.0288 & -698.4580 & -63.5818 & -693.0210 & 79.0398 \end{bmatrix} \tag{3.82}$$

was obtained which placed the closed-loop eigenvalues at

$-1.49999999999999 \pm j1.49999999999999$

$-2.00000000000000 \pm j0.99999999999998$

-16.99999999999997

-21.99999999999998

-0.70000000000001

-29.99999999999999

-35.00000000000068

Example 4

The output feedback matrix

$$K_y = \begin{bmatrix} -30.5551 & -66.3419 & 81.4301 \\ -52.9081 & -116.9301 & 143.7831 \end{bmatrix} \tag{3.83}$$

was obtained which placed the closed-loop eigenvalues at

$-1.00000000000235 \pm j0.49999999999770$

-4.00000000001050

-2.99999999998448

3.4.5 Results for the Method of Daley

Again the free parameters were chosen randomly for each example

Example 1

With the α's in ζ, which are arbitrary, set to 1, the following output feedback matrix was obtained

$$K_y = \begin{bmatrix} -10.9141 & 88.9999 \\ -105.0 & 1560.3744 \end{bmatrix} \tag{3.84}$$

which placed the closed-loop eigenvalues at

-10.00000000506724

-8.99999998539417

-8.00000000995767

and with the α's chosen randomly as

$$[\alpha_{11} \quad \alpha_{12}] = [0.0500 \quad 0.7615] \tag{3.85}$$

the output feedback matrix was

$$K_y = \begin{bmatrix} 79.2752 & 89.0 \\ -105.0 & -111.6407 \end{bmatrix} \tag{3.86}$$

giving the closed-loop eigenvalues at

-9.99999999983979
-9.00000000027615
-7.99999999988574

Example 2

With the α's set to 1, V_{11} was singular and so it was not possible to obtain a solution. However, selecting them randomly as

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix} = \begin{bmatrix} 0.7702 & 0.7702 & 0.8278 \\ 0.1253 & 0.1253 & 0.0159 \end{bmatrix} \tag{3.87}$$

gave a non-singular V_{11} and thus a solution. Note that the second column is the same as the first, this is due to the second column of ζ being the complex conjugate of the first. Also it appears that any set of randomly chosen α's gave rise to a solution.

The output feedback matrix

$$K_y = \begin{bmatrix} 0.6111 & -1.2109 & 1.3832 \\ 2.8451 & -3.5558 & 0.9548 \\ -0.3818 & -0.1552 & -1.6337 \end{bmatrix} \tag{3.88}$$

was obtained which placed the closed-loop eigenvalues at

$-5.00000000000115 \pm j4.99999999999876$

-2.99999999999987

-2.00000000000008

-0.99999999999996

Example 3

For this example it was not possible to find a set of α's which gave a non-singular V_{11}, hence no solution was found.

Example 4

With the α's set to 1, the following output feedback matrix was obtained

$$K_y = \begin{bmatrix} -33.9803 & 56.6120 & -63.2181 \\ 41.8266 & -73.1659 & 80.3171 \end{bmatrix} \tag{3.89}$$

which placed the closed-loop eigenvalues at

$-1.00000000000107 \pm j0.49999999999948$

-4.00000000000489

-2.99999999999250

and with the α's chosen randomly as

$$[\alpha_{11} \quad \alpha_{12} \quad \alpha_{13}] = [0.8459 \quad 0.8459 \quad 0.4121] \tag{3.90}$$

the output feedback matrix was

$$K_y = \begin{bmatrix} -17.5717 & 45.6026 & -58.4437 \\ 27.8993 & -77.5729 & 97.6308 \end{bmatrix} \tag{3.91}$$

placing the closed-loop eigenvalues at

$-1.00000000000074 \pm j0.49999999999965$

-4.00000000000432

-2.99999999999422

3.4.6 Discussion of the Results

Clearly the results show that the numerical properties of both methods are quite good with the closed-loop eigenvalues being placed extremely accurately.

The method of Fahmy and O'Reilly (1988a) was able to find a solution which gave a non-singular matrix of right eigenvectors for all the examples. However the method of Daley (1990) has some difficulty in obtaining a non-singular sub-block of the matrix of right eigenvectors in all cases. The rather unsatisfactory step of randomly selecting arbitrary scalars has to be done in example 2 in order for a solution to be obtained. For example 3 no solution was obtained at all.

The main problem with the method of Daley (1990) is clearly that of ensuring the invertibility of V_{11}.

This could be overcome by defining a vector sub-space that $\underline{\Gamma}$ must lie in such that V_{11} is invertible. $\underline{\Gamma}$ could then be selected from the intersection of this space and the null space of ζ, which is a requirement for non-trivial solutions to equation (3.65).

For V_{11}^{-1} to exist, its columns must be linearly independent, i.e

$$\omega_1 \underline{v}_1^{11} + \cdots + \omega_r \underline{v}_r^{11} \neq 0 \tag{3.92}$$

for all ω_i where at least one is non-zero. Note that $\underline{v}_i^{11}$ denotes the i'th column vector of V_{11}. Writing this equation in vector form

$$[\underline{v}_1^{11} \cdots \underline{v}_r^{11}][\omega_1 \cdots \omega_r]^T \neq 0 \tag{3.93}$$

From equation (3.60)

$$\underline{v}_i^{11} = [(\lambda_i I - A)^{-1} B]_r \underline{f}_i \tag{3.94}$$

Therefore

$$[S_r \underline{f}_1 \cdots S_r \underline{f}_r][\omega_1 \cdots \omega_r]^T \neq 0 \tag{3.95}$$

where $S_i = [(\lambda_i I - A)^{-1}B]_r$. This can be expressed in a more compact form

$$\zeta^{\blacklozenge}\underline{\Gamma}' \neq 0 \tag{3.96}$$

where

$$\zeta^{\blacklozenge} = [S_1 \cdots S_r] \quad \text{and} \quad \underline{\Gamma}' = [\omega_1 \underline{f}_1^T \cdots \omega_r \underline{f}_r^T]^T \tag{3.97}$$

Equation (3.96) is clearly of the standard form $A\underline{x} = \underline{b}$. For the case when $\underline{b} = 0$ it is well known that the set of solutions will form a basis of a vector subspace, generally known as the null space of A. However, the case of $\underline{b} \neq 0$ is of interest here and all that can now be stated is that the set of solutions will lie in some vector subspace but that they may not necessarily form a basis of that space.

For equation (3.96) this makes it difficult to define the set of possible $\underline{\Gamma}'$ vectors. It is possible to investigate this subspace further which may possibly lead to a way of defining the set of solution vectors to equation (3.96). However, a more significant problem remains, namely that of the arbitrary scalar weights. Recalling that equation (3.92) must be valid for all $\underline{w}_i$ $(i = 1 \rightarrow r)$, where at least one is non-zero, it is clear that even if the set of $\underline{\Gamma}'$ vectors could be found it would be extremely difficult to deduce the set of F_1 vectors using the form of $\underline{\Gamma}'$ in equation (3.97). No solution to this problem was found and so the idea of obtaining a set of F_1 vectors such that V_{11} is non-singular was not pursued further. It should be noted, however, that the problem of a singular V_{11} only caused problems occasionally and did not prevent the application of the method in most cases.

A second possible problem is in the determination of a space for the F_1 vectors such that there are non-trivial solutions to equation (3.65). The linear independence identity is used to ensure the dependence of a set of vectors, the columns of $\left[I - F_1 V_{11}^{-1}[(\lambda_j I - A)^{-1}B]_r\right]$. The only condition for dependence is that at least one set of scalar weightings (one or more non-zero) exists such that the weighted sum of the vectors is zero. This can be achieved by selecting a set and forcing the sum to be zero, hence the reason that the F_1 vectors must lie in a null space. Clearly this is actually selecting a subspace of the total space that the F_1 vectors may be in. This subspace may not necessarily include a set of F_1 vectors that yield a non-singular V_{11}.

3.5 Summary

This chapter has outlined the transformation of the polynomial system to state space form and detailed the interpretation of the output feedback matrix in terms of controller polynomials. Due to the importance of the eigenvalues and eigenvectors in determining the shape of the transient response it has been suggested that parametric methods be used to design the state space controller.

Two approaches to the problem of parametric output feedback design have been investigated and their performance assessed when applied to a number of design examples.

The method of Fahmy and O'Reilly (1988a) is a multi-stage design where successive output feedback loops are applied until the whole eigenstructure is assigned. It can be argued that this gives the method greater flexibility and a wider set of possible feedback matrices. However, it is difficult to know how to split the set of desired closed-loop eigenvalues. Good solutions were obtained for all the examples considered.

The second method, Daley (1990), can be thought of as a parametric state feedback design where the free parameters are constrained to effectively give output feedback. Problems arise when dealing with these constraints. It is necessary to ensure the invertibility of a sub-block of the matrix of right eigenvectors, which appears to be quite difficult for examples 2 and 3. By randomly selecting some arbitrary scalars in the design process it was possible to obtain a solution for example 2. However, this is an unsatisfactory way of approaching the problem as potentially many random choices may have to be made in order to find a solution.

The results of the comparison show that both methods appear to have good numerical properties, but the method of Daley (1990) did experience a number of difficulties. Although this does not stop the method being used it does suggest that the method of Fahmy and O'Reilly (1988a) would be more suitable. However when applied to a transformed polynomial system the method of Fahmy and O'Reilly (1988a) failed to find a solution at all. The reasons for this failure appear to be linked to the structure of the open-loop system matrices and a more detailed discussion is included in chapter five where the robust design is applied to an example.

As mentioned in the introduction there are a number of options when considering the state space design. In the area of eigenstructure assignment techniques a number of interesting methods have recently been proposed, unfortunately too late for consideration in this work.

Burrows and Patton (1988, 1989a, 1989b, 1990a, 1990b) have carried out much work in the area of robust eigenstructure assignment, utilising numerical minimisation methods to select appropriate sets of eigenvectors from prespecified regions of the complex plane on the basis of a number of robustness measures, including low norm control law and maximally orthogonal

eigenvectors. White (1991) has proposed a scheme which appears to be as numerically stable as Fahmy and O'Reilly (1988a) but which overcomes the problem of protection by assigning the desired eigenstructure in just one stage.

It would certainly be interesting to investigate these methods further and assess their impact on the robustness of the polynomial system. However, it is worth reiterating at this stage that any effective robust state space design procedure could be used as the aim is to illustrate and prove the concept of designing a polynomial controller via the state space domain.

The overall conclusion of this chapter is that the method of Daley (1990) is to be used as the basis of the state space design, which is assumed in all of the following chapters.

REFERENCES

Brasch, F.M., and Pearson, J.B. (1970)
'Pole Placement using Dynamic Compensator'
IEEE Transactions on Automatic Control, v15, pp 34-43

Burrows, S.P. (1990)
'Robust Control Design Techniques using Eigenstructure Assignment'
PhD Thesis, Dept of Electronics, University of York, U.K.

Burrows, S.P. and Patton, R.J. (1988)
'Robust Eigenstructure Assignment using the CTRL-C Design Package'
Sixth International Conference on Systems Engineering, Coventry, U.K., pp 868-875

Burrows, S.P. and Patton, R.J. (1989a)
'On Robustness Properties of Eigenstructure Assignment in the Control of Multivariable Uncertain Systems'
Sixth IFAC Symposium on Dynamic Modelling and Control of National Economies, Edinburgh, U.K., v2, pp 513-517

Burrows, S.P. and Patton, R.J. (1989b)
'Robust Eigenstructure Assignment for Flight Control using the CTRL-C Design Package'
AIAA Conference on Guidance, Navigation and Control, Boston, Mass, U.S.A., v2, pp 1489-1494

Burrows, S.P. and Patton, R.J. (1990a)
'Optimal Eigenstructure Assignment for Multiple Design Objectives'
1990 American Control Conference, San Diego, California, U.S.A., pp 1978-1683

Burrows, S.P. and Patton, R.J. (1990b)
'Robust Low Norm Output Feedback Design for Flight Control Systems'
AIAA Conference on Guidance, Navigation and Control, Boston, Mass, U.S.A.

Daley, S. (1990)
'On Eigenstructure Assignability using parametric output feedback'
Brunel University Control Engineering Centre Internal Report, August 1990

Fahmy, M.M. and O'Reilly, J. (1988a)
'Multistage Parametric Eigenstructure Assignment by Output Feedback Control'
International Journal of Control, v48, n1, pp 97-116

Fahmy, M.M. and O'Reilly, J. (1988b)
'Parametric Eigenstructure Assignment by Output Feedback Control: The Case of Multiple Eigenvalues'
International Journal of Control, v48, n4, pp 1519-1535

Friedland, B. (1986)
'Control System Design - An Introduction to State Space Methods'
McGraw Hill, London, U.K.

Kailath, T. (1980)
'Linear Systems'
Prentice Hall, London, U.K.

Kalsi, R.S. (1991)
'A Multi-Stage Parametric Approach to Output Feedback Controller Design'
Final Year Project, Dept of Elec Eng, Brunel University, Middlesex, U.K.

Kimura, H. (1975)
'Pole Assignment by Gain Output Feedback'
IEEE Transactions on Automatic Control, v20, n4, pp 509-516

Ogata, K. (1987)
'Discrete Time Control Systems'
Prentice Hall, London, U.K.

Roppenecker, G. and O'Reilly, J. (1989)
'Parametric Output Feedback Controller Design'
Automatica, v25, n2, pp 259-265

Sobel, K.M. and Shapiro, E.Y. (1985)
'Eigenstructure Assignment: A Tutorial - Part II Applications'
Proceedings of the 1985 American Control Conference, v1, pp 461-467
IEEE, New York, U.S.A. (Cat No. 85CH2119-6)

Srinathkumar, S. (1978)
'Eigenvalue/Eigenvector Assignment Using Output Feedback'
IEEE Transactions on Automatic Control, v23, n1, pp 79-81

Topaloglu, T. and Seborg, D.E. (1975)
'A Design Procedure for Pole Assignment using Output Feedback'
International Journal of Control, v22, n6, pp 741-748

White, B.A. (1991)
'Eigenstructure Assignment by Output Feedback'
International Journal of Control, v53, n6, pp 1413-1429

CHAPTER 4

SELECTING A ROBUST CONTROLLER

4.1 Introduction

The robust design procedure basically consists of transforming the polynomial system to state space form, carrying out a state space design and transforming the resulting controller back to polynomial form. The previous chapter outlined this procedure and presented details of the state space design, which is based on methods that explicitly represent a set of possible feedback controllers in terms of arbitrary free parameters, and as such are called parametric methods. The problem of how to select a controller from this set, such that the closed-loop system is more robust, is addressed in this chapter. Different controllers are effectively selected by changing the values of the free parameters and so the robust design reduces to the problem of selecting appropriate values for these parameters.

One of the reasons for turning to a state space based design was the ease with which the factors that effect the transient response can be investigated as highlighted by the discussion in section 3.2 on modal decomposition. However, it is necessary to have some way of mathematically quantifying the effect of model uncertainty on these factors.

Consider the graph in figure 4.1 which shows the value of a function $f(x)$ plotted against the variable x.

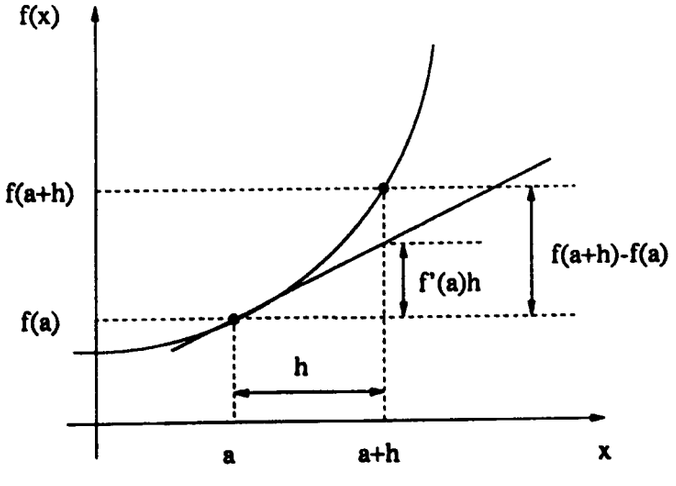


Figure 4.1 - Plot of a Function $f(x)$ against x

At $x = a$ the function value is $f(a)$ and when x is changed to $a + h$ the function value becomes $f(a + h)$. The gradient at $x = a$ is given by $f'(a)$ where

$$f'(a) = \left[\frac{df(x)}{dx} \right]_{x=a} \tag{4.1}$$

$f(a+h) - f(a)$ is called the *increment* (of $f(x)$ from $x = a$ to $x = a + h$) and is denoted by Δf

$$\Delta f = f(a+h) - f(a) \tag{4.2}$$

$f'(a)h$ is called the *differential* (at $x = a$ with increment h) and is denoted by df

$$df = f'(a)h \tag{4.3}$$

The differential is also sometimes referred to as the sensitivity, denoting that the expression evaluates how sensitive the function is to changes in the variables.

The increment is the actual change in the function due to a change in the variable whereas the differential is an estimate of the change in the function. The differential hence forms the basic mathematical tool needed to estimate the change in the factors that affect the transient response due to a change in the model parameters. Further information on differential calculus can be found in Salas and Hille (1990).

It is now possible to define a function consisting of expressions for the differentials of the factors of interest. This function is often referred to as a cost function, an objective function or a performance index. As it effectively represents the sensitivity to model uncertainty, it is desirable to obtain the lowest possible value for the function, which has clearly reduced the robust design to an optimisation problem as desired.

With complicated functions, as is the case here, there may be many local minima, at which the function has the lowest value with respect to the neighbourhood of possible points. The lowest local minimum is called the global minimum and clearly the ultimate goal of any optimisation procedure is to find this point. For complicated functions this represents an extremely difficult task and a more realistic target would simply be to aim for a good local minimum point which yields the desired level of improvement.

Having established how to quantify the effect of model uncertainty, the next section considers the problem of defining appropriate cost functions. Following this is a more detailed discussion of numerical optimisation which helps to classify the problem being considered allowing the selection of the most suitable optimisation algorithm.

4.2 Cost Functions

The previous chapter illustrated, through modal decomposition, how important the eigenvalues and eigenvectors are in determining the shape of the transient response. The way these factors are affected by structured model uncertainty will directly determine how the transient response is affected. This opens up a number of possibilities for evaluating performance robustness. Gourishankar and Ramar (1976), Owens (1988) and Owens and O'Reilly (1989) use eigenvalue sensitivity as the basis on which to select robust controllers. The sensitivity of the whole eigenstructure (eigenvalues and eigenvectors) is the approach used by Crossley and Porter (1969), Ling and Wang (1988). Other authors have investigated alternative approaches such as the conditioning of the matrix of right eigenvectors, V, which is claimed to improve the sensitivity of the transient response (Kautsky, Nichols and Van Dooren, 1985; Byers and Nash, 1989; Owens, 1991b). The cost function in this case should describe the change in the output of the system due to a change in some or all of the model parameters.

Before considering the details of some alternative cost functions it is necessary to quantify the error in the state space model. Chapter 3 described how the polynomial system is transformed to a state space system. From this it is clearly unnecessary to consider errors in the C matrix, as all its elements will be constant, hence the model uncertainty can be represented as

$$A = A_0 + \Delta A \tag{4.4}$$

$$B = B_0 + \Delta B \tag{4.5}$$

where ΔA and ΔB are the increments of A and B defined as

$$\Delta A = \sum_{i=1}^{u} P_i \varepsilon_i \tag{4.6}$$

$$\Delta B = \sum_{i=1}^{u} Q_i \varepsilon_i \tag{4.7}$$

P_i and Q_i represent the known information about the structure of the errors and ε_i the unknown magnitude of the errors. The increments can be approximated by the differentials

$$dA \approx \Delta A \tag{4.8}$$

$$dB \approx \Delta B \tag{4.9}$$

where

$$dA = \sum_{i=1}^{u} \frac{\partial A}{\partial \varepsilon_i} \Delta\varepsilon_i \tag{4.10}$$

$$dB = \sum_{i=1}^{u} \frac{\partial B}{\partial \varepsilon_i} \Delta\varepsilon_i \tag{4.11}$$

and $\Delta\varepsilon_i$ is the change (or increment) in ε. Clearly the partial derivatives are

$$\frac{\partial A}{\partial \varepsilon_i} = P_i \tag{4.12}$$

$$\frac{\partial B}{\partial \varepsilon_i} = Q_i \tag{4.13}$$

assuming the ε_i's are independent.

4.2.1 Eigenvalue Differential Cost Function

A number of authors have derived expressions for eigenvalue sensitivity, for example Crossley and Porter (1969), Ling and Wang (1988), Skelton (1988). In general all such expressions are derived from the equation

$$A_c \underline{v}_i = \lambda_i \underline{v}_i \tag{4.14}$$

where A_c denotes the closed-loop system matrix, λ_i and $\underline{v}_i$ its associated eigenvalues and right eigenvectors respectively. Similarly for the left eigenvectors, $\underline{w}_i$

$$A_c^T \underline{w}_i = \lambda_i \underline{w}_i \tag{4.15}$$

Following Porter and Crossley (1972), partially differentiating (4.14) with respect to ε_i gives

$$\frac{\partial A_c}{\partial \varepsilon_t}\underline{v}_i + A_c\frac{\partial \underline{v}_i}{\partial \varepsilon_t} = \frac{\partial \lambda_i}{\partial \varepsilon_t}\underline{v}_i + \lambda_i\frac{\partial \underline{v}_i}{\partial \varepsilon_t} \tag{4.16}$$

Pre-multiplying by $\underline{w}_i^T$

$$\underline{w}_i^T\frac{\partial A_c}{\partial \varepsilon_t}\underline{v}_i + \underline{w}_i^T A_c\frac{\partial \underline{v}_i}{\partial \varepsilon_t} = \underline{w}_i^T\frac{\partial \lambda_i}{\partial \varepsilon_t}\underline{v}_i + \underline{w}_i^T\lambda_i\frac{\partial \underline{v}_i}{\partial \varepsilon_t} \tag{4.17}$$

From (4.15)

$$\underline{w}_i^T A_c = \underline{w}_i^T\lambda_i \tag{4.18}$$

Which means that

$$\underline{w}_i^T A_c\frac{\partial \underline{v}_i}{\partial \varepsilon_t} = \underline{w}_i^T\lambda_i\frac{\partial \underline{v}_i}{\partial \varepsilon_t} \tag{4.19}$$

Hence (4.17) becomes

$$\underline{w}_i^T\frac{\partial A_c}{\partial \varepsilon_t}\underline{v}_i = \underline{w}_i^T\underline{v}_i\frac{\partial \lambda_i}{\partial \varepsilon_t} \tag{4.20}$$

For normalised eigenvectors $\underline{w}_i^T\underline{v}_i = 1$, hence

$$\frac{\partial \lambda_i}{\partial \varepsilon_t} = \underline{w}_i^T\frac{\partial A_c}{\partial \varepsilon_t}\underline{v}_i \tag{4.21}$$

Also as $A_c = A + BKC$

$$\frac{\partial A_c}{\partial \varepsilon_t} = \frac{\partial (A + BKC)}{\partial \varepsilon_t} = \frac{\partial A}{\partial \varepsilon_t} + \frac{\partial B}{\partial \varepsilon_t} KC$$

$$= P_t + Q_t KC \tag{4.22}$$

Hence

$$\frac{\partial \lambda_i}{\partial \varepsilon_t} = \underline{w}_i^T (P_t + Q_t KC) \underline{v}_i \tag{4.23}$$

and the eigenvalue differential can then be expressed as

$$d\lambda_{it} = \underline{w}_i^T (P_t + Q_t KC) \underline{v}_i \Delta\varepsilon_t \tag{4.24}$$

where $d\lambda_{it}$ denotes the differential of the i'th eigenvalue to the t'th error. As already stated the aim is to determine the value of the free parameters which yield the lowest value for this function. However the function is comprised of two parts, the known $\partial\lambda_i/\partial\varepsilon_t$ and the unknown $\Delta\varepsilon_t$. As there is no control over the value of $\Delta\varepsilon_t$, the cost function should only consist of the known partial derivative part, giving the eigenvalue differential cost function as

$$J = J_1 = \sum_{t=1}^{u} \sum_{i=1}^{n} \beta_{it} \left(\frac{\partial \lambda_i}{\partial \varepsilon_t} \right)^2 \tag{4.25}$$

The partial derivative is squared to ensure that the cost function remains positive. β_{it} are positive weights used to place importance on each of the eigenvalues.

4.2.2 Eigenstructure Differential Cost Function

The previous section evaluated an expression for the eigenvalue differential, however from the modal decomposition it is clear that the differential of the whole eigenstructure (eigenvalues and eigenvectors) should be considered. This has also been dealt with by a number of authors, for example Crossley and Porter (1969), Porter and Crossley (1972), Ling and Wang (1988) and again are generally derived from equations (4.14) and (4.15). However, for the parametric method the right eigenvectors are expressed as

$$\underline{v}_i = (\lambda_i I - A)^{-1} B \underline{f}_i \tag{4.26}$$

which can be used to evaluate an expression for the eigenvector differential. Partially differentiating (4.26) gives

$$\begin{aligned}\frac{\partial \underline{v}_i}{\partial \varepsilon_t} &= \frac{\partial (\lambda_i I - A)^{-1}}{\partial \varepsilon_t} B \underline{f}_i + (\lambda_i I - A)^{-1} \frac{\partial B}{\partial \varepsilon_t} \underline{f}_i \\ &= -(\lambda_i I - A)^{-1} \frac{\partial (\lambda_i I - A)}{\partial \varepsilon_t} (\lambda_i I - A)^{-1} B \underline{f}_i + (\lambda_i I - A)^{-1} Q_t \underline{f}_i \\ &= -(\lambda_i I - A)^{-1} \left(\frac{\partial \lambda_i}{\partial \varepsilon_t} I - \frac{\partial A}{\partial \varepsilon_t} \right) (\lambda_i I - A)^{-1} B \underline{f}_i + (\lambda_i I - A)^{-1} Q_t \underline{f}_i \end{aligned} \tag{4.27}$$

Substituting $\partial \lambda_i / \partial \varepsilon_t$ from (4.23) gives

$$\frac{\partial \underline{v}_i}{\partial \varepsilon_t} = -(\lambda_i I - A)^{-1} (\underline{w}_i^T (P_t + Q_t K C) \underline{v}_i I - P_t) (\lambda_i I - A)^{-1} B \underline{f}_i + (\lambda_i I - A)^{-1} Q_t \underline{f}_i \tag{4.28}$$

As $W = V^{-1}$, where W is the matrix of left eigenrows and V is the matrix of right eigenvectors, it is relatively simple to find an expression for the partial derivative of the left eigenrows.

$$\frac{\partial \underline{w}_i^T}{\partial \varepsilon_t} = \frac{\partial (\underline{e}_i^T V^{-1})}{\partial \varepsilon_t} = -\underline{e}_i^T V^{-1} \frac{\partial V}{\partial \varepsilon_t} V^{-1} = -\underline{e}_i^T W \frac{\partial V}{\partial \varepsilon_t} W \tag{4.29}$$

where $\underline{e}_i$ is the i'th column of the unit matrix.

The right eigenvector differential can then be expressed as

$$d\underline{v}_{it} = (-(\lambda_i I - A)^{-1} (\underline{w}_i^T (P_t + Q_t K C) \underline{v}_i I - P_t) (\lambda_i I - A)^{-1} B \underline{f}_i + (\lambda_i I - A)^{-1} Q_t \underline{f}_i) \Delta \varepsilon_t \tag{4.30}$$

where $d\underline{v}_{it}$ denotes the differential of the i'th right eigenvector to the t'th error. Similarly the left eigenrow differential is

$$d\underline{w}_{it}^T = -\underline{e}_i^T W \frac{\partial V}{\partial \varepsilon_t} W \Delta\varepsilon_t \tag{4.31}$$

The differentials are vector values and as with the eigenvalue differential consist of a known and unknown part. Using only the known partial derivative part and ensuring that the cost function is a positive scalar value gives the eigenstructure differential cost function as

$$J_2 = \sum_{t=1}^{u} \sum_{i=1}^{n} \eta_{it} \left\| \frac{\partial \underline{v}_i}{\partial \varepsilon_t} \right\| + \eta_{it} \left\| \frac{\partial \underline{w}_i^T}{\partial \varepsilon_t} \right\| \tag{4.32}$$

where $\|\cdot\|$ denotes any vector norm, and η_{it} are positive weights. As it is necessary to also consider the eigenvalue differential to ensure the eigenvalue sensitivities do not become too large, the overall cost function will be

$$J = J_1 + \sigma J_2 \tag{4.33}$$

where σ is also a positive weight.

4.2.3 Transient Response Differential Cost Function

The previous cost functions merely consisted of expressions for the eigenvalue and eigenvector sensitivities, as it is known that they are important in determining the sensitivity of the transient response. However, as an expression for the transient response was actually derived in chapter 3, it would be interesting to directly determine its differential and hopefully define a transient response differential cost function. It should be noted that other authors have also looked at this problem, for example Skelton (1988) defines an expression for output sensitivity by differentiating the state equations.

The differential of the transient response is given by

$$d\underline{y}(k) = \frac{\partial \underline{y}(k)}{\partial \varepsilon_t} \Delta\varepsilon_t \tag{4.34}$$

and from chapter 3

$$\underline{y}(k) = C \sum_{i=1}^{n} \underline{v}_i \underline{w}_i^T \left(\lambda_i^k \underline{x}(0) + \sum_{j=0}^{k-1} \lambda_i^{k-j-1} B \underline{u}(j) \right) \tag{4.35}$$

hence the partial derivative is

$$\frac{\partial \underline{y}(k)}{\partial \varepsilon_t} = C \sum_{i=1}^{n} \frac{\partial \underline{v}_i \underline{w}_i^T}{\partial \varepsilon_t} \left(\lambda_i^k \underline{x}(0) + \sum_{j=0}^{k-1} \lambda_i^{k-j-1} B \underline{u}(j) \right) + \underline{v}_i \underline{w}_i^T \frac{\partial \left(\lambda_i^k \underline{x}(0) + \sum_{j=0}^{k-1} \lambda_i^{k-j-1} B \underline{u}(j) \right)}{\partial \varepsilon_t} \tag{4.36}$$

which gives

$$\frac{\partial \underline{y}(k)}{\partial \varepsilon_t} = C \sum_{i=1}^{n} \frac{\partial \underline{v}_i \underline{w}_i^T}{\partial \varepsilon_t} \left(\lambda_i^k \underline{x}(0) + \sum_{j=0}^{k-1} \lambda_i^{k-j-1} B \underline{u}(j) \right) + \underline{v}_i \underline{w}_i^T \left(k \lambda_i^{k-1} \frac{\partial \lambda_i}{\partial \varepsilon_t} \underline{x}(0) + \sum_{j=0}^{k-1} \left((k-j-1) \lambda_i^{k-j-2} \frac{\partial \lambda_i}{\partial \varepsilon_t} B \underline{u}(j) + \lambda_i^{k-j-1} \frac{\partial B}{\partial \varepsilon_t} \underline{u}(j) \right) \right) \tag{4.37}$$

This is quite a complicated expression, the evaluation of which is dependent on particular values of k, the discrete time sequence. It is clearly not very useful as regards defining a cost function, however upon closer examination it can be seen that the eigenvectors of the system always appear in the form of $\underline{v}_i \underline{w}_i^T$. This does suggest that a more suitable representation of the eigenvector differential would be obtained by considering the term $\underline{v}_i \underline{w}_i^T$.

Partially differentiating this term gives

$$\frac{\partial \underline{v}_i \underline{w}_i^T}{\partial \varepsilon_t} = \frac{\partial \underline{v}_i}{\partial \varepsilon_t} \underline{w}_i^T + \underline{v}_i \frac{\partial \underline{w}_i^T}{\partial \varepsilon_t} \tag{4.38}$$

and the transient response differential cost function can be defined as

$$J = J_1 + \sigma J_3 \tag{4.39}$$

where

$$J_3 = \sum_{t=1}^{u} \sum_{i=1}^{n} \gamma_{it} \left\| \frac{\partial \underline{v}_i}{\partial \varepsilon_t} \underline{w}_i^T + \underline{v}_i \frac{\partial \underline{w}_i^T}{\partial \varepsilon_t} \right\| \tag{4.40}$$

and now $\|\cdot\|$ denotes any matrix norm and σ, γ_{it} are positive weights.

4.2.4 Conditioning Cost Function

Wilkinson (1965) showed that the conditioning of the matrix of eigenvectors is related to the sensitivity of the eigenvalues. This can be seen from the Bauer-Fike theorem, Golub and Van Loan (1983), which states that if μ is an eigenvalue of $A + E$ and $V^{-1}AV = D = \text{diag}(\lambda_1 \cdots \lambda_n)$ then

$$\min_{\lambda \in \lambda(A)} |\lambda - \mu| \leq \kappa_F(V) \|E\|_F \tag{4.41}$$

where $\|\cdot\|_F$ denotes the Frobenius norm and $\kappa_F(V) = \|V\|_F \|V^{-1}\|_F$.

The theorem basically states that the eigenvalues of a matrix move at a rate no greater than $\kappa_F(V)$ (the Frobenius norm condition number of the matrix of right eigenvectors) per unit change in $\|E\|$ (the error in the matrix).

Kautsky *et al* (1985) propose a number of different measures of robustness based on the conditioning of the matrix of eigenvectors.

i) $r_1 = \|\underline{c}\|_\infty \equiv \max_j \{c_j\}$

ii) $r_2 = \kappa_2(V)$

iii) $r_3 = n^{-1/2} \|V^{-1}\|_F$

iv) $r_4 = n^{-1/2} \left(\sum_j \sin^2 \theta_j \right)^{1/2}$

where $\|\cdot\|_\infty$ denotes the infinity norm, $\|\cdot\|_F$ the Frobenius norm and $\kappa_2(V)$ the 2-norm condition number of the matrix V

$$\kappa_2(V) \equiv \|V\|_2 \|V^{-1}\|_2 \tag{4.42}$$

$\underline{c} = [c_1, \cdots c_n]$ is the vector of condition numbers, which assuming that the right eigenvectors are normalised such that $\|\underline{v}_i\|_2 = 1$, are defined as

$$c_i = \|\underline{w}_i^T\|_2 \tag{4.43}$$

and lastly θ_j are the angles between the eigenvectors $\underline{v}_j$ and certain corresponding orthonormal vectors $\underline{\hat{v}}_j$.

All these measures attain their minimum simultaneously when the assigned eigenvalues are as insensitive as possible. Kautsky *et al* (1985) go on to propose two further weighted measures

$$\text{v)} \quad r_3(D) = \frac{\|DV^{-1}\|_F}{\|D\|_F} \equiv \frac{\left(\sum_j d_j^2 c_j^2\right)^{1/2}}{\left(\sum_j d_j^2\right)^{1/2}}$$

$$\text{vi)} \quad r_4(D) = \frac{\left(\sum_j d_j^2 \sin^2\theta_j\right)^{1/2}}{\left(\sum_j d_j^2\right)^{1/2})}$$

A suitable set of weights is given by $d_j^{-1} = (1 - |\lambda_j|)$ for the discrete time case. Using these weights has the effect of minimising an upper bound on the stability margin of the closed-loop system.

Byers and Nash (1989) also propose a number of measures of robustness

$$\text{vii)} \quad r_5 = \kappa_F^2[V]$$

$$\text{viii)} \quad r_6 = \frac{-1}{r_5}$$

$$\text{ix)} \quad r_7 = \|V\|_F^2 + \|V^{-1}\|_F^2$$

$$\text{x)} \quad r_8 = \frac{-1}{r_7}$$

$$\text{xi)} \quad r_9 = \log[r_7]$$

and state that although no cost function is uniformly better than the others, functions r_8 and r_9 generally perform well. r_9 is recommended for general use, particularly if ill-conditioned problems are to be encountered. However, a recent paper, Owens (1991a) suggests that r_9 may not be effective in certain cases.

From the literature it is certainly very difficult to judge the suitability of any particular conditioning based cost function. Owens (1991b) does discuss a number of the possible functions and gives some insight into their relative benefits. From this and remembering the Bauer-Fike theorem, a good initial choice would seem to be

$$J = J_4 = \kappa_F(V) = \| V \|_F \| V^{-1} \|_F \tag{4.44}$$

Owens (1991b) also reports that considering $(\kappa_F(V))^2$ resulted in a faster convergence to the solution for the example considered.

4.3 Optimisation Techniques

The Numerical Algorithms Group (NAG) library of FORTRAN 77 routines contains a number of quite sophisticated optimisation algorithms. It was felt that using the routines in this library would greatly simplify the implementation of the robust design method and further details of the reasons for this decision can be found in the following chapter.

The aim of this section is to introduce the concepts behind numerical optimisation and some of the terminology used. This will then allow the problem being considered to be classified which will aid in the selection of an appropriate optimisation routine. The section finishes with a brief introduction to some of the main types of algorithms. Because the NAG routines are to be used the following is biased to their approach and is largely taken from NAG (1990).

4.3.1 Introduction and Terminology

The solution of optimisation problems by a single, all-purpose, method can be cumbersome and inefficient. For this reason such problems are classified into particular categories for which various algorithms are best suited. The problem can be characterised by the properties of the cost function and the constraint functions, for example

Properties of cost Function

Nonlinear

Sums of squares of nonlinear functions

Quadratic

Sums of squares of linear functions

Linear

Properties of Constraints

Nonlinear

Sparse linear

Linear

Bounds

None

It is necessary to express these types of problems mathematically for numerical optimisation. Firstly, consider unconstrained problems where there are no restrictions on the value of the variables. Mathematically the problem can be stated as

$$\underset{\underline{x}}{\text{minimise}} \quad F(\underline{x}) \tag{4.45}$$

where $\underline{x} \in \mathbf{R}^n$ are the variables and $F(\underline{x})$ the function.

The NAG library makes special provision for problems which can be expressed as the sum of squared functions, often referred to as a least squares problem. The mathematical statement of this problem is

$$\underset{\underline{x}}{\text{minimise}} \quad \left\{ \underline{f}^T \underline{f} = \sum_{i=1}^{m} f_i^2(\underline{x}) \right\} \tag{4.46}$$

where the i'th element of the m-vector $\underline{f}$ is the function $f_i(\underline{x})$, which is often referred to as a residual.

Now consider the problem subject to constraints of some kind. As indicated above there are a number of categories of constraints, probably the most straightforward of which are bounds on the variables. For the problem of (4.45) the variables can be bounded by

$$l_i \leq x_i \leq u_i \tag{4.47}$$

where x_i are the respective elements of the vector $\underline{x}$ and l_i the lower bounds and u_i the upper bounds on the value of x_i. This does assume that bounds exist on all the variables, but by allowing $u_i = \infty$ and $l_i = -\infty$ all the variables need not be restricted.

Next consider linear constraints, which are defined as linear functions in more than one of the variables, e.g. $3x_1 + 2x_2 \geq 4$. Mathematically such constraints can be described by

Equality constraints	$\underline{a}_i^T \underline{x} = b_i$	$i = 1, 2, \cdots, m_1$
Inequality constraints	$\underline{a}_i^T \underline{x} \geq b_i$	$i = m_1 + 1, \cdots, m_2$
	$\underline{a}_i^T \underline{x} \leq b_i$	$i = m_2 + 1, \cdots, m_3$
Range constraints	$s_j \leq \underline{a}_i^T \underline{x} \leq t_j$	$i = m_3 + 1, \cdots, m_4$
		$j = 1, \cdots, m_4 - m_3$
Bound constraints	$l_i \leq x_i \leq u_i$	$i = 1, 2, \cdots, n$

where each $\underline{a}_i$ is a vector of length n and b_i, s_j and t_j are constant scalars. Also note that any of the categories above may be null.

If $F(\underline{x})$ is a linear function, the linearly constrained problem is termed a linear programming problem and if $F(\underline{x})$ is a quadratic function, the problem is termed a quadratic programming problem.

Lastly, in the discussion of the characterisation of the problem, consider the case of (4.45) subject to nonlinear constraints, e.g. $x_1^2 + x_3 \geq 0$. The above mathematical description of the linear constraints still applies but now the following constraints must also be considered

Equality constraints	$c_i(\underline{x}) = 0$	$i = 1, 2, \cdots, m_5$
Inequality constraints	$c_i(\underline{x}) \geq 0$	$i = m_5 + 1, \cdots, m_6$
Range constraints	$v_j \leq c_i(\underline{x}) \leq w_j$	$i = m_6 + 1, \cdots, m_7$
		$j = 1, 2, \cdots, m_7 - m_6$

where each c_i is a nonlinear function and v_j, w_j are constant scalars. Again any category may be null.

The problem facing the optimisation algorithms is to find a minimum of the given function $F(\underline{x})$ subject to constraints of one of the above types. It is worth mentioning here that the function is likely to have many minima, each called a local minimum because the function has the lowest value at that point with respect to the neighbourhood of possible points. The lowest of these local minima is termed the global minimum for obvious reasons. In the face of this, all that can be expected of the algorithms is that they find a local minimum point, and by starting the algorithm in a number of places it is hoped to find a local minimum that yields a desirable result.

For an algorithm to find a minimum point (either local or global) it must have some way of determining whether or not a particular point is a minimum. Also it must have a method of determining the direction to move, such that a minimum point can be found. The next part of this discussion attempts to address this point by first reviewing some of the required mathematics and defining conditions for a minimum.

The vector of first partial derivatives of $F(\underline{x})$ is called the gradient vector and is denoted by $g(\underline{x})$, i.e.

$$g(\underline{x}) = \left[\frac{\partial F(\underline{x})}{\partial x_1}, \frac{\partial F(\underline{x})}{\partial x_2}, \cdots, \frac{\partial F(\underline{x})}{\partial x_n} \right]^T \tag{4.48}$$

The gradient vector is of importance in optimisation because it must be zero at an unconstrained minimum of any function with continuous first derivatives.

The matrix of second partial derivatives of the function is termed its Hessian matrix and is denoted by $G(\underline{x})$. Its (i,j)'th element is given by

$$\frac{\partial^2 F(\underline{x})}{\partial x_i \partial x_j} \tag{4.49}$$

If $F(\underline{x})$ has continuous second partial derivatives then $G(\underline{x})$ must be positive semi-definite at any unconstrained minimum of the function.

In nonlinear least squares problems, the matrix of first derivatives of the vector valued function $\underline{f}$ is termed the Jacobian matrix.

4.3.2 Classification of the Problem

At this stage it is possible to determine which category the problem of interest lies in. To do this it is necessary to first of all examine the cost functions being considered. The variables are obviously the free parameter vectors used to determine the eigenvectors and hence the output feedback matrix.

Consider the relationship between the variables which are the free parameter vectors $\underline{f}_i$, introduced by the parametric design approach, and the value of the function, which is made up of terms dependent on the closed-loop eigenvalues and eigenvectors. The relationship is clearly a very complicated one and it is not a straightforward task to determine the character of it. The closed-loop system matrix is given by

$$A_c = A + BK_x \tag{4.50}$$

for state feedback. Consider the expression for the state feedback matrix

$$K_x = FV^{-1} \tag{4.51}$$

and the expression for the right eigenvectors

$$\underline{v}_i = (\lambda_i I - A)^{-1} B \underline{f}_i \tag{4.52}$$

where the definition of the terms involved can be found in chapter 3, which discusses the parametric design methods in detail. Note that it was concluded in that chapter that the method of Daley (1990) will form the basis of the state space design.

Because of the dependency of the right eigenvectors $\underline{v}_i$ on the free parameter vectors $\underline{f}_i$, nonlinear terms will be present in the evaluation of K_x. Hence the problem is of a nonlinear nature.

Because the parametric design approach of Daley (1990) is forcing the state feedback matrix to be in the form $[K_y : 0]$, such that output feedback can be used, the free parameter vectors $\underline{f}_i$ are subject to constraints which from chapter 3 were shown to be

$$F_1 W_{12} + F_2 W_{22} = 0 \tag{4.53}$$

Again because of the dependency of the eigenvectors on the free parameter vectors, this equation is of a nonlinear nature. However Daley (1990) did reduce this constraint equation to two further equations

$$\left[I - F_1 V_{11}^{-1}[(\lambda_i I - A)^{-1}B]_r\right]\underline{f}_i = 0\,, \qquad i = r+1 \rightarrow n \tag{4.54}$$

$$\zeta\underline{\Gamma} = 0 \tag{4.55}$$

This has not removed the non-linearity, but does give an indication of how the actual constraints can effectively be removed. $\underline{\Gamma}$ contains all the free parameter vectors associated with F_1 which can be chosen freely provided that $\underline{\Gamma}$ lies in the null space of the matrix ζ. Once F_1 is determined, the remaining free parameter vectors can be selected subject to equation (4.54). This also requires the vectors to be selected from a null space. All the null spaces involved will have a set of basis vectors, and any vector that lies in these null spaces will simply be a linear combination of the basis vectors. For example consider a null space described by the set of basis vectors

$$\{\underline{x}_1 \quad \underline{x}_2 \quad \underline{x}_3 \quad \underline{x}_4 \quad \underline{x}_5\} \tag{4.56}$$

To select a vector $\underline{y}$ which lies in this null space, it must satisfy

$$\underline{y} = \alpha_1\underline{x}_1 + \alpha_2\underline{x}_2 + \alpha_3\underline{x}_3 + \alpha_4\underline{x}_4 + \alpha_5\underline{x}_5 \tag{4.57}$$

where the scalar values, α_i are completely arbitrary.

This clearly demonstrates that to effectively remove the constraints from the problem, the scalar multipliers of the basis vectors of the null spaces should be used as the free parameters.

If this approach is taken the robust design can then be classified as an unconstrained nonlinear optimisation problem. Examining the eigenvalue differential cost function, J_1 it is easily seen that it consists of the sum of a number of squared terms. This clearly fits into the description of the least squares type of problem. However to gain any real benefit from the least squares formulation in the NAG library it is necessary to ensure that the number of squared terms is greater than the number of variables. The cost functions J_2 and J_3 consist of a number of terms

which are norms of vectors or matrices. It is possible to force these into a least squares framework by squaring the norms, but such a step would only really be necessary if computational efficiency (the major advantage of the least squares framework) becomes critical.

4.3.3 Common Optimisation Methods

All the algorithms in the NAG library generate an iterative sequence $\underline{x}(k)$ that converges to the solution $\underline{x}^*$ in the limit. The sequence is usually constructed by satisfying

$$\underline{x}(k+1) = \underline{x}(k) + \alpha(k)\underline{p}(k) \tag{4.58}$$

where the vector $\underline{p}(k)$ is termed the direction of search and $\alpha(k)$ is the step length.

The step length is chosen such that $F(\underline{x}(k+1)) < F(\underline{x}(k))$ and is computed by performing a one-dimensional optimisation. The NAG library uses two techniques for one-dimensional optimisation, one fits a quadratic polynomial using only function evaluations and the other uses additional information about the gradient to fit a cubic polynomial.

The major differences between the various methods arise due to the need to use varying levels of information about the derivatives of $F(\underline{x})$ in defining the search direction, further details can be found in Gill and Murray (1981). Four common algorithms for unconstrained minimisation are

i) ***Newton-Type Methods (Modified Newton Methods)***

Newton-type methods use the Hessian matrix $G(\underline{x}(k))$, or a finite difference approximation to $G(\underline{x}(k))$, to define the search direction. Newton-type methods are the most powerful methods available for general problems and will find the minimum of a quadratic function in one iteration.

ii) ***Quasi-Newton Methods***

Quasi-Newton methods approximate the Hessian $G(\underline{x}(k))$ by a matrix $B(\underline{x}(k))$ which is modified at each iteration to include information obtained about the curvature of $F(\underline{x})$ along the latest search direction. Although not as robust as Newton-type methods, quasi-Newton methods can be more efficient because $G(\underline{x}(k))$ is not computed, or approximated by finite differences. Quasi-Newton methods minimise a quadratic function in n iterations.

iii) ***Conjugate-Gradient Methods***

Unlike Newton-type and quasi-Newton methods, conjugate gradient methods do not require storage of an n by n matrix and so are ideally suited to solve large problems. Conjugate-gradient type methods are not usually as reliable or efficient as Newton-type, or quasi-Newton methods.

iv) ***Downhill Simplex Method***

This method is due to Nelder and Mead (1965) and is a completely self-contained approach that does not use one-dimensional minimisation, unlike the previous methods. It requires only function evaluations, so no derivative information is needed. The method is not very efficient in terms of the number of function evaluations that it requires but is numerically quite robust.

4.4 Summary

The previous chapter discussed the state space design where it was decided to use parametric methods which explicitly represent a set of possible feedback controllers in terms of arbitrary free parameters. This chapter has been concerned with the problem of selecting the free parameters such that the resulting controller yields a closed-loop system with improved performance robustness.

To achieve this a suitable function relating the sensitivity of the closed-loop system to structured model uncertainty can be defined, and the free parameters selected such that this function is minimised. This has clearly reduced the robust design to an optimisation problem as desired, which can then be solved using numerical methods. Of course it can only be expected that a local minimum point is found and by starting the optimisation procedure from a number of points it is hoped to find a local minimum which yields the desired level of robustness.

For performance robustness the effect of model uncertainty on the shape of the transient response is of interest. Through modal decomposition it was shown that the sensitivity of the transient response is strongly related to the sensitivity of the eigenvalues and eigenvectors, leading to the definition of a number of cost functions. One further cost function was also included which is based on the conditioning of the matrix of right eigenvectors.

Having established suitable cost functions, the fundamentals of numerical optimisation were discussed in more detail, showing how problems can be categorised to help in the selection of an appropriate optimisation algorithm. The problem being considered can be classified as an unconstrained nonlinear problem, although for the design method of Daley (1990), some re-arrangement of the problem had to be carried out. This involved re-defining the free parameters to be scalar multipliers of the basis vectors of the null space of a matrix.

REFERENCES

Byers, R. and Nash, S.G. (1989)
'Approaches to Robust Pole Assignment'
International Journal of Control, v49, n1, pp 97-117

Crossley, T.R. and Porter, B. (1969)
'Eigenvalue and Eigenvector Sensitivities in Linear Systems Theory'
International Journal of Control, v10, n2, pp 163-170

Daley, S. (1990)
'On Eigenstructure Assignability using parametric output feedback'
Brunel University Control Engineering Centre Internal Report, August 1990

Gill, P.E. and Murray, W. (1981)
'Practical Optimization'
Academic Press, London, U.K.

Golub, G.H. and Van Loan, C.F. (1983)
'Matrix Computations'
North Oxford Academic Publishing, Oxford, U.K.

Gourishankar, V. and Ramar, K. (1976)
'Pole Assignment with Minimum Eigenvalue Sensitivity to Plant Parameter Variations'
International Journal of Control, v23, n4, pp 493-504

Kautsky, J., Nichols, N.K. and Van Dooren, P. (1985)
'Robust Pole Assignment in Linear State Feedback'
International Journal of Control, v41, n5, pp 1129-1155

Ling, Y. and Wang, B. (1988)
'First and Second Order Eigensensitivities of Matrices with Distinct Eigenvalues'
International Journal of Systems Science, v19, n7, pp 1053-1067

NAG (1990)
'Fortran Library Manual Mk 14'
Numerical Algorithms Group Ltd, Oxford, U.K.

Nelder, J.A. and Mead. R. (1965)
'A simplex Method for Function Minimisation'
Computer Journal, v7, pp 308-313

Owens, T.J. (1988)
'Parametric Output Feedback Control with Response Insensitivity'
International Journal of Control, v48, n3, pp 1213-1239

Owens, T.J. and O'Reilly, J. (1989)
'Parametric State Feedback Control for Arbitrary Eigenvalue Assignment with Minimum Sensitivity'
IEE Proceedings, Pt D, v136, n6, pp 307-312

Owens, T.J. (1991a)
'A Framework for Optimal Control by Nonlinear Programming'
International Symposium on Applied Mathematical Programming and Modelling, Brunel University, Middlesex, U.K.

Owens, T.J. (1991b)
'Parametric State Feedback Control for Arbitrary Eigenvalue Assignment with Minimum Condition Number'
1991 European Control Conference, Grenoble, France. v2, pp 1292-1294
Hermes, Paris, France.

Porter, B. and Crossley, R. (1972)
'Modal Control, Theory and Applications'
Taylor and Francis Ltd, London, U.K.

Sallas, S.L. and Hille, E. (1990)
'Calculus - One Variable'
John Wiley and Sons, Chichester, U.K.

Skelton, R.E. (1988)
'Dynamic Systems Control'
John Wiley and Sons, Chichester, U.K.

Wilkinson, J.H. (1965)
'The Algebraic Eigenvalue Problem'
Clarendon Press, Oxford, U.K.

CHAPTER 5

IMPLEMENTATION AND APPLICATION OF THE ROBUST DESIGN PROCEDURE

5.1 Introduction

The problem being considered is that of designing a polynomial controller for a system in input-output form, such that the closed-loop system is robust to changes in the parameters of the plant being controlled. The preceding chapters have defined this problem and proposed a possible approach to solving it, involving transforming to state space form and performing a parametric output feedback design. The extra degrees of freedom in the design are then chosen to satisfy some robustness criteria using the techniques of numerical optimisation.

There are still a number of unanswered questions though, involving the choice of cost function and optimisation technique. General answers are not easily provided as to a certain extent they are dependent on the actual plant being considered. To help illustrate how to make suitable choices, the application of the proposed design method to an example is considered in this chapter.

At this stage a secondary set of problems comes to light, namely those related to the implementation of the method. This is an important aspect, as the way the method is implemented will have a significant effect on its performance mainly in terms of speed. A discussion of the major points regarding the implementation is presented which highlights the computational requirements of the method as well as detailing some of the main problems encountered and how they were overcome.

As previously mentioned in chapter three the state space parametric method of Fahmy and O'Reilly (1988) experienced some difficulties when applied to transformed polynomial systems. The problems encountered with this approach when applied to the example being considered are also discussed in this chapter leading to the conclusion that the state space design should be based on the parametric method of Daley (1990).

A comprehensive set of results is presented for each of the cost functions which aims at highlighting how the weights are chosen and the typical improvement that can be achieved with this approach. The chapter finishes with a discussion of these results and conclusions about the performance of the proposed cost functions.

5.2 Implementation of the Robust Design Procedure

Pro-Matlab v3.5 is a very flexible interactive package that allows easy development of ideas and designs. It contains a large number of quite complex inbuilt functions largely relating to matrix operations, such as inversion, singular value decomposition and factorisations. A number of other functions for particular tasks are available through optional toolboxes, for example in the areas of control system design and system identification. The package is installed on a Sun 3/60 workstation running under the UNIX operating system in a windows environment.

To help illustrate the programs being used a number of flowcharts have been included which can be found at the end of the chapter. Appendix B contains listings of the actual programs for more detailed information.

Figure 5.1 outlines the overall design from specifying a polynomial system to obtaining the robust controller polynomials. The loop is indicative of the optimisation procedure where a local minimum value of the cost function is sought. Figure 5.2 expands on the parametric design stage and as can be seen a significant level of computation is required for this part, which suggests that it will largely determine the time taken to perform one iteration of the optimisation phase.

Such a design is very easy to implement in Pro-Matlab due to the number of inbuilt functions. However Pro-Matlab only has the downhill simplex method of optimisation available[1] which, although numerically robust, does not perform well on complex problems.

The Numerical Algorithms Group (NAG) library contains a number of routines for numerical optimisation based on various algorithms, as outlined in chapter 4. The library routines are however written in Fortran 77.

This suggests a number of possible options; 1) Perform the whole design in Pro-Matlab; 2) Attempt to link Pro-Matlab and Fortran 77 and lastly 3) Perform the design entirely in Fortran 77.

Considering each of these options in turn, firstly 1). This would involve writing all of the required optimisation routines, which although possible has a number of drawbacks. For numerical optimisation, speed is certainly an extremely important factor, which will be effected by the design of the algorithm (i.e how quickly it converges) and the actual implementation on the computer. This is really heading into the realms of mathematicians and computer scientists and it is felt that the development of satisfactory algorithms would have become a research topic in its own right, and so is outside the scope of this work. Also for speed considerations it should be remembered that as Pro-Matlab is an interactive package all commands are interpreted and it is well known that interpreted computer languages are considerably slower than compiled

1 The Mathworks Inc, suppliers of Pro-Matlab, have recently introduced an 'optimisation toolbox' which contains a number of more complex numerical optimisation algorithms.

languages, such as Fortran 77. Pro-Matlab itself is written in the language C and as such provides an easy interface to routines written in this language. Hence the optimisation routines could be written in C, thus increasing the speed with which they run.

This leads to the conclusion that if a compiled language and Pro-Matlab are to be interfaced, why not use Fortran 77 (option 2) and gain access to the NAG library routines. This avoids having to write the optimisation routines but presents another problem. Because Fortran 77 and C (the language Pro-Matlab is written in) are very different in the way they store and reference data, an interface between Fortran 77 and Pro-Matlab is not a simple proposition. Pro-Matlab does provide facilities for setting up such an interface but it was found to be quite limited and not particularly useful for this type of application.

Of course, option 2 could be tackled by not providing such a rigid link between the two. A more flexible link could be established by simply writing the required data to a file. This does require Pro-Matlab and Fortran 77 to be set up to read the same type of file format, which presents another difficulty as Pro-Matlab has its own special format (needed because the basic data structure is a matrix). This approach also suffers from speed problems as writing to a file is extremely time consuming in relation to processor time.

The third option is really quite similar to the first except that now the optimisation routines are available (from the NAG library) but many of the necessary matrix functions are not. Some of the algorithms to perform these functions can be almost as complicated to implement as the numerical optimisation algorithms. Certainly speed considerations would not be as much of a problem but accuracy would be. Pro-Matlab is significantly more accurate than Fortran 77, which can be verified by some simple tests, such as inverting a matrix. This is quite logical as the version of Fortran 77 on the sun workstation only works up to double precision whereas Pro-Matlab works to a much higher level of precision.

Clearly the best approach is to try and establish a link between Pro-Matlab and Fortran 77 in order that the maximum benefit can be gained from the facilities available in both.

5.2.1 The Link Between Pro-Matlab and Fortran 77

From the introductory discussion on the implementation it is clear that the best way to proceed is to establish a link via data files. However, because file access is slow, all calculations associated with the optimisation should be performed entirely in Fortran 77. This will entail developing some routines to perform functions which are available in Pro-Matlab, but will lead to a much faster implementation of the design. The basic scheme is then to perform as much of the pre-optimisation work as possible in Pro-Matlab, transfer all required data via a file to the Fortran 77 routine and after a sub-optimal solution is found transfer the results back to Pro-Matlab for all post-optimisation work.

Obviously a common data file format is required and as Pro-Matlab provides a set of Fortran 77 routines to read and write Pro-Matlab format files, the simplest way to proceed would be to use the Pro-Matlab file format. The two routines are called LOADMAT and SAVEMAT and the argument list for both routines is the same and consists of

type - Matrix type flag; considering the type flag as a decimal integer, the ones decimal place is used to indicate numeric or textual interpretation of the matrix data (0 for numeric and 1 for textual); the 1000's decimal place is used to indicate the machine format for the matrix data (0 for Intel 8086 based machines, 1 for Motorola 68000 based machines and a 2 for Vax d format). A flag of 1000 indicates numeric data in a 68000 machine format and a flag of 1001 indicates textual interpretation of the 68000 machine format data.

mrows - Number of rows in the matrix.

ncols - Number of columns in the matrix.

imagf - Imaginary flag; 0 for no imaginary part or 1 for an imaginary part.

namlen - Number of characters in the matrix name plus 1 (for zero byte string terminator).

name - Character array holding the matrix name.

rpart - Real part of the matrix (mrows x ncols double precision elements stored column wise).

ipart - Imaginary part of the matrix (only used if imagf = 1).

lunit - Logical Fortran 77 unit.

irec - Direct access record counter (set to 1 to start at the beginning of the file).

flag - Read/Write status flag; 0 - good read/write, 1 - end of file, 0 - error during read/write.

LOADMAT reads a double precision matrix from a Pro-Matlab format file and successive calls to the routine will allow all matrices to be read until the end of the file has been reached. Before calling the routine only the logical unit number, *lunit*, must be specified. A logical unit number is assigned using the OPEN statement in Fortran 77. When opening the file it must be specified as unformatted and direct access. More information on file handling in Fortran 77 can be obtained from Edgar (1989). All the remaining arguments are returned by the routine.

SAVEMAT is used for the reverse process where successive calls will write a double precision matrix to the specified file (denoted by the logical unit number). However, before calling this routine it is necessary to specify a number of the arguments, namely:- *type*, *mrows*, *ncols*, *imagf*, *namlen*, *name*, *rpart*, *ipart*, *lunit*, *irec*. The only argument returned by the routine is *flag*, to indicate the success of the write operation.

Now consider what data will actually be passed using the data files. To answer this it is necessary to take a closer look at the parametric design method. The flowchart in figure 5.2 will help here. The first step is to calculate $\underline{\Gamma}$, which is dependent on the null space of ζ. This null space does not depend on the free parameters, which are the scalar multipliers of the null space vectors. Hence the null space can be calculated in Pro-Matlab and passed to the Fortran 77 optimisation routine using the file. The next step which requires additional data is the calculation of the right eigenvectors $\underline{v}_i$. Recall that

$$\underline{v}_i = (\lambda_i I - A)^{-1} B \underline{f}_i \tag{5.1}$$

If $(\lambda_i I - A)^{-1} B$ is calculated in Pro-Matlab for all i and stored in the file, then the calculation of the eigenvectors during optimisation will only involve one multiplication of a matrix and a vector.

From the flowchart it is apparent that this is all the data that can be calculated prior to optimisation as all other values are dependent on the free parameters. However, it is also necessary to consider the calculation of the cost functions to determine what additional data is required.

Consider the eigenvalue differential cost function

$$\frac{\partial \lambda_i}{\partial \varepsilon_t} = \underline{w}_i^T (P_t + Q_t K C) \underline{v}_i \tag{5.2}$$

for this expression, $\underline{w}_i^T$, $\underline{v}_i$ and K will all be calculated as a result of performing the parametric design at each iteration. The only information missing is C, but as it will always be in the form $[I : 0]$ there is no need to store it in the data file.

Now consider the eigenstructure differential cost function

$$\frac{\partial \underline{v}_i}{\partial \varepsilon_t} = -(\lambda_i I - A)^{-1} (\underline{w}_i^T (P_t + Q_t K C) \underline{v}_i I - P_t)(\lambda_i I - A)^{-1} B \underline{f}_i + (\lambda_i I - A)^{-1} Q_t \underline{f}_i \tag{5.3}$$

This requires $(\lambda_i I - A)^{-1}$ to be calculated in Pro-Matlab for all i and stored in the file. Of course it is possible to store just $(\lambda_i I - A)^{-1}$ and B separately and explicitly calculate $(\lambda_i I - A)^{-1}B$ in Fortran 77 when it is needed. However, as speed is of much greater concern than storage in this case, both $(\lambda_i I - A)^{-1}$ and $(\lambda_i I - A)^{-1}B$ will be calculated and stored for all i.

The remaining cost functions require no further data, however to assist in the construction of the Fortran 77 routines a number of other variables are stored in the file which give information on the dimensions of the matrices involved and which eigenvalues are complex conjugates.

It was also decided to store all the necessary weights and initial values for the free parameters in another file so as to allow changes to be easily made.

Once the optimisation has been performed in Fortran 77, the only data required to continue with the conversion of the controller back to polynomial form is the value of K_y. A number of other variables, however, are also returned to aid in the analysis of the solution.

To determine what routines need to be written in Fortran 77, it is again necessary to consider the parametric design and the cost functions. To evaluate the functions is relatively simple once the parametric design has been carried out. The design relies on routines being available to calculate the null space of a matrix, in order to determine the vectors in F_2, and to calculate the inverse of real and complex matrices. Although Pro-Matlab has such functions, they are not directly available in the NAG library and the following sections discuss how to implement the required routines.

5.2.2 Calculation of the Null Space of a Matrix in Fortran 77

The parametric design method requires the calculation of null spaces of matrices. Pro-Matlab has a function to perform this task but as the design is carried out during optimisation, it is necessary to write a Fortran 77 routine to calculate the null space of a matrix. The algorithm is based on the singular value decomposition (SVD).

Consider the SVD of a $m \times n$ matrix A, where $m \geq n$

$$A = U_1 D U_2^T \tag{5.4}$$

where D is a $m \times n$ matrix, U_1 and U_2^T are orthonormal matrices of dimension $m \times m$ and $n \times n$ respectively. If rank$\{A\} = r \leq n$ then D is defined as

$$D = \begin{bmatrix} V & 0 \\ 0 & 0 \end{bmatrix} \tag{5.5}$$

where V is a $r \times r$ diagonal matrix of the singular values of A.

Let $U_1 = [U_{11} \quad U_{12}]$ and $U_2 = [U_{21} \quad U_{22}]$ where the dimension of the sub-blocks are U_{11} - $m \times r$, U_{12} - $m \times (m-r)$, U_{21} - $n \times r$, U_{22} - $n \times (n-r)$. Then

U_{22} is a basis for the null space of A.
U_{12} is a basis for the null space of A^T.
U_{11} is a basis for the range space of A.
U_{21} is a basis for the range space of A^T.

Of course the next problem is to calculate the SVD of the matrix A. This can be achieved by finding the matrix of right eigenvectors of the matrix AA^T which will be equal to U_1 and the matrix of right eigenvectors of the matrix A^TA which will be equal to U_2. The NAG library routine F02ABF can be used to calculate the eigenvectors of a matrix. Details of the routine can be found in appendix C.

5.2.3 Calculation of an Accurate Inverse of a Matrix in Fortran 77

It was found in practice that in the calculation of the F_2 vectors from the null space of

$$I - F_1 V_{11}^{-1} [(\lambda_{j+r} I - A)^{-1} B]_r \tag{5.6}$$

(where $j = 1 \rightarrow n-r$) the calculation of the inverse of V_{11} was extremely important. In Pro-Matlab, which has greater precision, there were no problems in calculating an accurate inverse, but in Fortran 77, with only double precision, this was not the case. It is recommended in NAG (1990) that when finding the inverse of a real matrix A_r, the equation

$$A_r X_r = I \tag{5.7}$$

be solved for X_r, which is the inverse. I is the identity matrix of the same order as A_r. The algorithms for solving such a set of real linear equations are faster in execution and numerically

more stable and accurate. Hence for real matrices this equation is simply solved using two NAG routines. F03AFF factorises the matrix A_r into upper and lower triangular form and F04AHF uses this form to solve the set of equations using backward substitution with correction. Details of these routines are in appendix C.

For complex matrices, such as V_{11}, the following procedure is used to ensure an accurate inverse is found. Consider the complex equation

$$A_c X_c = I \tag{5.8}$$

where A_c is the complex matrix and X_c its unknown inverse.

If the equation is transformed to a real equation of the same form, then NAG routines for solving a set of real equations can be used to find the inverse. Consider equation (5.8) in terms of its real and imaginary parts

$$(A_r + jA_i)(X_r + jX_i) = I \tag{5.9}$$

where I is the same dimension as the original complex A_c. Combining real and imaginary parts the equation can be written as

$$(A_r X_r - A_i X_i) + j(A_i X_r + A_r X_i) = I \tag{5.10}$$

which gives rise to two real equations

$$A_r X_r - A_i X_i = I \tag{5.11}$$

$$A_i X_r + A_r X_i = 0 \tag{5.12}$$

or, alternatively, in vector form

$$[A_r \quad -A_i]\begin{bmatrix} X_r \\ X_i \end{bmatrix} = I \tag{5.13}$$

$$[A_i \quad A_r]\begin{bmatrix} X_r \\ X_i \end{bmatrix} = 0 \tag{5.14}$$

which can be combined into one real matrix equation

$$\begin{bmatrix} A_r & -A_i \\ A_i & A_r \end{bmatrix}\begin{bmatrix} X_r \\ X_i \end{bmatrix} = \begin{bmatrix} I \\ 0 \end{bmatrix} \tag{5.20}$$

This equation is clearly of the correct form and can be solved for the real and imaginary parts of X_c.

5.3 Application of the Robust Design Procedure

Having established how the method is implemented on a computer, it is possible to proceed with an application to a simple example. The definition of this example is presented first including the calculation of a controller from the minimum order solution of the diophantine equation, which can be used to assess the performance of the robust control schemes. With the definition of the model it is also natural to consider the transformation to state space form and the definition of the model uncertainty.

A slight detour is then taken to consider the details of the problems experienced with the parametric method of Fahmy and O'Reilly (1988) when applied to this example. The conclusion of this work is that the method of Daley (1990) should be used as the basis of the state space design. This is followed by a discussion of the design in the state space framework to assess the number of free parameters, which naturally leads to the selection of an appropriate optimisation routine. As there are a large number of results, they are presented in a rather compact form and an explanation of this layout is given before the results themselves are presented for each cost function.

5.3.1 Definition of the System and Preliminary Work

The following non-minimum phase system is to be considered (taken from Wellstead and Sanoff, 1981)

$$(1-1.6z^{-1}+0.6z^{-2})y(k)=(z^{-1}+1.5z^{-2})u(k)+(1-0.4z^{-1})e(k) \tag{5.16}$$

The open-loop poles are located at $z=1$ and $z=0.6$, and it is desired to place the closed-loop poles at $z=0.75\pm j0.2$.

As outlined in chapter 1, for zero steady state error it is necessary to ensure that the system has integral action. In this case it is easily verified that the $A_p(z^{-1})$ polynomial is of the form

$$A_p(z^{-1})=(1-z^{-1})(1-0.6z^{-1}) \tag{5.17}$$

and so the system already has integral action. It is worth noting that the integral action should be invariant under parameter variation else the steady state error would vary with coefficient changes. Hence if a system exhibits integral action which is not structural then an integrator should still be cascaded with the system. For this example it is assumed that only the $(1-0.6z^{-1})$ part of $A_p(z^{-1})$ is time varying, hence there is no need to cascade an integrator in this case.

It is also assumed that the $A_p(z^{-1})$ and $B_p(z^{-1})$ coefficients vary by the same amount and that the variation will be of the order of +50% of their nominal magnitude.

For the purposes of a comparison the controller obtained from the minimum order solution of the diophantine equation can be calculated. The minimum order controller polynomials are

$$F_p(z^{-1})=1.0-0.3466z^{-1} \tag{5.18}$$

$$G_p(z^{-1})=0.0466-0.0220z^{-1} \tag{5.19}$$

$$H_p(z^{-1})=0.0410-0.0164z^{-1} \tag{5.20}$$

and the corresponding pole positions for the nominal and perturbed closed-loop system are

Original	Perturbed	Distance Moved
$0.75\pm j0.2$	$0.8967\pm j0.3727$	$0.1467\pm j0.1727$
0.4	0.3834	-0.0166

Table 5.1 - Pole Positions for the Perturbed Closed-Loop System with the Minimum Order Controller

Figure 5.3 shows the transient response of the nominal and perturbed closed-loop system which is clearly significantly affected by parameter variations.

Now consider the robust design and hence the transformation of the system to state space form. Following the procedure outlined in chapter 3

$$A = \begin{bmatrix} 1.6 & \cdots & 1.0 \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ -0.6 & \cdots & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1.0 & . & \cdots & 0 \\ . & 1.0 & & . \\ . & & . & . \\ . & & & 1.0 \\ 1.5 & . & \cdots & 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 1.0 & . & . & 0 & 0 & \cdots & 0 \\ . & 1.0 & & . & . & & . \\ . & & . & . & . & & . \\ 0 & . & . & 1.0 & 0 & \cdots & 0 \end{bmatrix} \qquad (5.21)$$

where the actual dimensions are dependent on the value of p. Notice that the system states have been re-arranged such that C is in the correct form for the parametric design procedure of Daley (1990).

It is then possible to easily define the structured model uncertainty as

$$\Delta A = \begin{bmatrix} -p_1 & \cdot & \cdot & 0 \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ p_2 & \cdot & \cdot & 0 \end{bmatrix} \varepsilon_1 \qquad (5.22)$$

$$\Delta B = \begin{bmatrix} q_1 & \cdot & \cdot & 0 \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ q_2 & \cdot & \cdot & 0 \end{bmatrix} \varepsilon_1 \qquad (5.23)$$

and as the variation in the coefficients will be of the same order it is only necessary to set the p's and q's to 1. Note that u, the number of error terms, is equal to 1. The form of ΔA with $-p_1$ and $+p_2$ is due to the integral action in $A_p(z^{-1})$.

5.3.2 Problems Associated with the Parametric Method of Fahmy and O'Reilly

The parametric design method of Fahmy and O'Reilly (1988) has consistently had problems calculating an output feedback controller for transformed polynomial systems. The problem appears to be linked with the structure of the open-loop system matrices and the following outlines how the method fails.

Considering the given example, but not transformed such that $C = [I : 0]$ (which is unnecessary for this method), with $p = 3$ the system matrices are

$$A = \begin{bmatrix} 0 & -0.6 & 0 & 0 & 0 \\ 1 & 1.6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1.5 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5.24)$$

and the desired closed-loop eigenvalues are $[0.75 \pm j0.2 \quad -0.01 \quad -0.02 \quad -0.03]$.

First stage - right eigenvector assignment

Choose $s = 3$, hence require three free parameter vectors which are chosen randomly as

$$F_s = [f_1 \quad f_2 \quad f_3] = \begin{bmatrix} 0.0329 & 0.0329 & -0.2530 \\ 0.8892 & 0.8892 & 5.2730 \\ -10.7023 & -10.7023 & -6.1485 \\ 0.7486 & 0.7486 & 9.6141 \end{bmatrix} \qquad (5.25)$$

The first three right eigenvectors are then calculated as

$$V_s = [\underline{v}_1 \quad \underline{v}_2 \quad \underline{v}_3] = \begin{bmatrix} 0.71 - j0.31 & 0.71 + j0.31 & 1.24 \\ -0.91 + j0.15 & -0.91 - j0.15 & -0.61 \\ 1.11 - j0.30 & 1.11 + j0.30 & -527.30 \\ -13.32 + j3.55 & -13.32 - j3.55 & 614.85 \\ 0.93 - j0.25 & 0.93 + j0.25 & -961.41 \end{bmatrix} \qquad (5.26)$$

giving

$$K_{11} = \begin{bmatrix} -0.0946 & 0.0059 & 0.0045 \\ -2.5596 & 0.1329 & 0.1200 \\ 30.8060 & -1.7202 & -1.4541 \\ -2.1547 & 0.1010 & 0.1001 \end{bmatrix} \tag{5.27}$$

Intermediate stage - eigenvalue/eigenvector protection

To protect the eigenvalues placed in the first stage it is necessary to calculate an output reduction matrix $\tilde{C}$ which satisfies

$$\tilde{C}CV_s = 0 \tag{5.28}$$

Transposing this equation gives

$$(CV_s)^T\tilde{C}^T = 0 \tag{5.29}$$

and $\tilde{C}$ can be obtained from the null space of $(CV_s)^T$.

$$(CV_s)^T = \begin{bmatrix} -0.91 - j0.15 & 1.11 + j0.30 & -13.32 - j3.55 & 0.93 + j0.25 \\ -0.91 + j0.15 & 1.11 - j0.30 & -13.32 + j3.55 & 0.93 - j0.25 \\ -0.61 & -527.30 & 614.85 & -961.41 \end{bmatrix} \tag{5.30}$$

giving

$$\tilde{C} = [-6.4858e-15 \quad -0.8870 \quad -0.0415 \quad 0.4599] \tag{5.31}$$

Note that the first element of this vector is close to zero, which appeared to be the case for all other examples considered. This suggests that something about the structure of A, B and C causes this to happen.

Now

$$A_1 = A + BK_1C = \begin{bmatrix} 0 & -0.7418 & 0.0088 & 0.0068 & 0 \\ 1.0 & 1.5054 & 0.0059 & 0.0045 & 0 \\ 0 & -2.5596 & 0.1329 & 0.1200 & 0 \\ 0 & 30.8060 & -1.7702 & -1.4546 & 0 \\ 0 & -2.1547 & 0.1010 & 0.1001 & 0 \end{bmatrix} \tag{5.32}$$

$$C_1 = \tilde{C}C = [0 \quad -6.4858e-15 \quad -0.8870 \quad -0.0415 \quad 0.4599] \tag{5.33}$$

and the second stage design is based on A_1, B and C_1. The leading zeros in C_1 are a result of the combination of the zero in $\tilde{C}$ and the structure of the first two columns of C.

Second stage - left eigenvector assignment

This stage assigns the remaining two eigenvalues and their associated left eigenvectors. The two free parameters, which are scalars, are chosen as

$$G_t = \begin{bmatrix} 3.4949 \\ -12.5092 \end{bmatrix} \tag{5.34}$$

giving the eigenvectors as

$$W_t^T = [\underline{w}_4 \quad \underline{w}_5] = \begin{bmatrix} -0.3523e-13 & 0.2266e-13 \\ 0.7041e-15 & -0.6715e-15 \\ 154.9955 & -369.8497 \\ 7.2569 & -17.3164 \\ -80.3684 & 191.7748 \end{bmatrix} \tag{5.35}$$

The effect of the leading zeros in C_1 can clearly be seen. Recall that

$$\tilde{K}_{21} = [W_t B_1]^{-1} G_t \tag{5.36}$$

and the matrix W_tB_1 is

$$W_r B_1 = \begin{bmatrix} -0.5213e-13 & 154.9955 \\ 0.3332e-13 & -369.8497 \end{bmatrix} \tag{5.37}$$

which is certainly close to singular and is the cause of the failure of the method.

The problem does appear to be closely related to the structure of C, particularly the first two columns. To eliminate the zeros in these columns a state transformation could be used.

Selecting

$$T = \begin{bmatrix} 0.9103 & 0.3282 & 0.2470 & 0.0727 & 0.7665 \\ 0.7622 & 0.6326 & 0.9826 & 0.6316 & 0.4777 \\ 0.2625 & 0.7564 & 0.7227 & 0.8847 & 0.2378 \\ 0.0475 & 0.9910 & 0.1534 & 0.2727 & 0.2749 \\ 0.7361 & 0.3653 & 0.6515 & 0.4364 & 0.3593 \end{bmatrix} \tag{5.38}$$

the transformed system is

$$A' = T^{-1}AT = \begin{bmatrix} -12.0329 & -7.5397 & -10.2049 & -6.0669 & -8.6642 \\ -12.8911 & -8.1191 & -11.0231 & -6.5661 & -9.2630 \\ 11.8317 & 7.5256 & 10.2769 & 6.1440 & 8.4681 \\ 0.8081 & 0.4876 & 0.6447 & 0.3776 & 0.5905 \\ 15.3245 & 9.4643 & 12.6977 & 7.5067 & 11.0975 \end{bmatrix} \tag{5.39}$$

$$B' = T^{-1}B = \begin{bmatrix} -6.2748 & 0.0613 & 0.9066 & 7.6603 \\ -5.9431 & 0.8272 & 2.0939 & 5.7442 \\ 4.0784 & -1.7437 & -0.3933 & -3.8486 \\ 0.7715 & 1.8398 & -1.2741 & -1.3003 \\ 10.5666 & -0.0395 & -1.7258 & -10.1938 \end{bmatrix} \tag{5.40}$$

$$C' = CT = \begin{bmatrix} 0.7622 & 0.6326 & 0.9826 & 0.6316 & 0.4777 \\ 0.2625 & 0.7564 & 0.7227 & 0.8847 & 0.2378 \\ 0.0475 & 0.9910 & 0.7534 & 0.2727 & 0.2749 \\ 0.7361 & 0.3653 & 0.6515 & 0.4364 & 0.3593 \end{bmatrix} \tag{5.41}$$

Performing the design using the same free parameters, it is found that K_{11} for the first stage is the same as previously. Also $(CV_s)^T$ is the same leading to

$$\tilde{C} = [-6.4858e-15 \quad -0.8870 \quad -0.0415 \quad 0.4599] \tag{5.42}$$

as before and

$$C_1 = [0.1038 \quad -0.5441 \quad -0.3726 \quad -0.5953 \quad -0.0571] \tag{5.43}$$

In the second stage

$$W_t^T = \begin{bmatrix} -18.1343 & 43.279 \\ 95.0704 & -226.8566 \\ 65.1146 & -155.3763 \\ 104.0310 & -248.2383 \\ 9.9754 & -23.8032 \end{bmatrix} \tag{5.44}$$

which appears to have removed the problem of leading zeros. However

$$W_t B_1 = \begin{bmatrix} 0.1335e-11 & 154.9955 \\ -0.3024e-11 & -369.8497 \end{bmatrix} \tag{5.45}$$

and again the problem of singularity stops a solution being obtained.

The fact that the problem is related to the structure of C can be verified by changing some of its elements. It was found that only two elements needed to be changed in order for a solution to be obtained, and as would be expected these are in the first two columns. For example with

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0.219 & 0.047 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.46}$$

a solution is obtained where

$$K_y = \begin{bmatrix} -10.4063 & 30.9594 & 2.4719 & -15.3845 \\ -2.6056 & 0.1382 & 0.1248 & 0.0002 \\ 31.4037 & -1.7946 & -1.5172 & 0 \\ -2.1898 & 0.1054 & 0.1038 & 0 \end{bmatrix} \tag{5.47}$$

It should be pointed out that a number of different strategies for overcoming the encountered problems were tried. These included specifying different desired closed-loop eigenvalues, assigning the left eigenvectors first and changing the split between the two stages. However no approach managed to ensure that a solution could be obtained. Another possibility is to use a different canonical form in the transformation from polynomial to state space representations. This however does not alleviate the problem as a similar structure for the C matrix is obtained when the dynamic compensator is augmented with the system.

The most probable reason for the failure of the method is overprotection. To explain, consider the equation

$$\tilde{C}CV_s = 0 \tag{5.48}$$

where V_s represents the matrix of the first s right eigenvectors assigned in the first stage. Recall from chapter three that $\tilde{C}$ must be chosen such that this equation is satisfied for the first s right eigenvectors to be protected from subsequent feedback loops. This equation can also be interpreted as requiring V_s to lie in the null space of $\tilde{C}C$. If the dimension of this null space is greater than s then other right eigenvectors may be protected leading to overprotection and the method would fail in the second stage, as is the case here.

A complete analysis of this problem and an investigation into ways of overcoming it is really outside the scope of this work, hence the parametric method of Daley (1990) will be used as the basis of the state space design.

5.3.3 Determining the Number of Free Parameters

Chapter four outlined how to avoid subjecting the optimisation procedure to constraints by effectively re-defining the free parameters as the scalar multipliers of the basis vectors of various null spaces. Hence to determine the number of free parameters it is necessary to obtain the dimensions of the null spaces involved and this section outlines how this is achieved.

Consider the system as outlined in section 5.3.1. There will be $p+2$ states, $p+1$ inputs and $p+1$ outputs, hence

$$n = p + 2 \tag{5.49}$$

$$r = p + 1 \tag{5.50}$$

$$m = p + 1 \tag{5.51}$$

Clearly

$$q = n - r = p + 2 - p - 1 = 1 \tag{5.52}$$

so

$$F_1 = [f_1 \cdots f_r] \tag{5.53}$$

$$F_2 = [f_n] \tag{5.54}$$

Recall that the whole of F_1 is selected from the null space of ζ, where

$$\zeta = \begin{bmatrix} \alpha_{11}\Delta S_{11} & \cdots & \alpha_{1r}\Delta S_{1r} \\ \vdots & & \vdots \\ \alpha_{q1}\Delta S_{q1} & \cdots & \alpha_{qr}\Delta S_{qr} \end{bmatrix} \tag{5.55}$$

and

$$\Delta S_{ik} = [((\lambda_{i+r} I - A)^{-1} - (\lambda_k I - A)^{-1})B]_r \tag{3.56}$$

In this case the α's are all chosen as 1 and the dimension of ζ is clearly $(r \times rm)$ as $q = 1$ and the dimension of ΔS_{ik} is $(r \times m)$.

Section 5.2 discussed details of the implementation which included a section on how to calculate the null space of a matrix using its singular value decomposition. From this the dimension of the null space of ζ can be deduced as $(rm \times (rm - r))$, assuming full rank. Thus the number of free parameters used in the calculation of F_1 is $r(m - 1)$.

The remaining vectors in F_2 are each obtained by calculating the null space of Z_i, $(i = r + 1 \rightarrow n)$. In this case there is only one F_2 vector and it lies in the null space of

$$Z_n = I - F_1 V_{11}^{-1}[(\lambda_n I - A)^{-1} B]_r \tag{5.57}$$

The dimension of Z_n is obviously $(r \times r)$, so the existence of the null space is dependent on the rank of the matrix. Daley (1990) has shown that this null space will exist and that the rank of Z_n will be deficient by at least 1. Hence the dimension of the null space will be $(r \times 1)$ and only one free parameter is needed for the calculation of the F_2 vector.

This brings the total number of free parameters, N_f to

$$N_f = r(m-1)+1 \tag{5.58}$$

5.3.4 Selecting the Optimisation Routine

This section considers the choice of the optimisation algorithm. For simplicity and to allow a quick assessment of the robust design approach it was decided to use algorithms that require no derivative information. It is recognised that derivative information will generally improve the efficiency of optimisation in all areas, and algorithms that utilise such information could be used if unsatisfactory results are obtained from the function value only algorithms.

Most routines of this type in the NAG library are based on the quasi-Newton method, which really only leaves the question of whether the problem is of a least squares type. The only cost function which easily fits this description is the eigenvalue differential cost function, J_1. Note that for this example $u = 1$ so there are only n residual terms in the function. For any real benefit to be gained by describing the problem in a least squares manner, the number of residuals, N_{re} should be greater than the number of variables (or free parameters), N_f, i.e.

$$N_{re} > N_f \quad \text{or} \quad r(m-1)+1 > n \tag{5.59}$$

For $p = 1, N_f = 3$ and $N_{re} = 3$ and as p increases N_f increases at a higher rate than N_{re}. Thus there is no justification in expressing the problem in a least squares framework, hence it was decided to use NAG routine E04JAF to perform the optimisation. Further details of this routine can be found in appendix C.

5.3.5 Layout of the Results

All the results will be laid out in the following format

x) Initial comments

$\underline{X}^T = [\cdots]$

$\underline{W}^T = [\cdots]$

$J_{orig} = \cdots$ $J_{opt\ y} = \cdots$

Eigenvalue/Eigenvector Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02	...
orig	...	...	...	
opt	...	...	...	
orig	...	...	...	
opt	...	...	...	

Pole Positions:

Original	Perturbed	Distance Moved
...	...	...

Controller Polynomials:

$F_p(z^{-1}) = \cdots$
$G_p(z^{-1}) = \cdots$
$H_p(z^{-1}) = \cdots$

Final comments

Where

x specifies a reference number for the particular set of results. Hence when discussing results only this number need be quoted.

Initial comments

These comments briefly introduce the set of results explaining what they represent.

$\underline{X}^T$ is a vector that specifies the initial values for the free parameters, where the first $r(m-1)$ elements are used in the calculation of F_1, and the remaining element in the calculation of F_2.

$\underline{W}^T$ is a vector that specifies the value of the weights used. Clearly the interpretation is dependent on the cost function being used.

$\underline{W}^T(1 \rightarrow n)$ - specifies the n eigenvalue sensitivity weights β_i

$\underline{W}^T(n+1 \rightarrow 2n)$ - for the eigenstructure differential cost function, specifies the weights η_i, and for the transient response differential cost function specifies the weights γ_i

$\underline{W}^T(2n+1)$ - specifies the weight σ

For the conditioning cost function no weights need to be specified.

J_{orig} specifies the value of the cost function at the starting point.

$J_{opt\,y}$ specifies the value of the cost function at the final point. The value of y is used to indicate the conditions under which the routine terminated.

$y = 1$ - indicates that a minimum point was found.

$y = 2$ - indicates that not all the conditions for a minimum were satisfied but that no lower point could be found.

Eigenvalue/Eigenvector Sensitivities:

The eigenvalue sensitivities are given in the first two rows followed by the eigenvector sensitivities. For each entry there are two numbers. The upper entry corresponds to the sensitivity at the starting point and the lower one to the sensitivity at the final point. For the eigenvalue differential cost function results there will clearly not be entries for the eigenvector sensitivities and for the conditioning cost function results this table will not be given.

Pole Positions:

The first column of this table corresponds to the closed-loop pole positions for the nominal system, the second column to the closed-loop pole positions for the perturbed system and the third to how far the poles have actually moved. This table is only given for relatively good results and always for the conditioning cost function results.

Controller Polynomials:

The three controller polynomials are only specified for particularly good sets of results.

Final comments

These contain any conclusions that can be drawn and outline the reasons behind the decisions for the next set of results, e.g. how the weights should be changed.

5.3.6 Results for the Eigenvalue Differential Cost Function

The following results were obtained for the system described in (5.16), using the eigenvalue differential cost function. The desired closed-loop pole positions are $0.75 \pm j0.2$ and the maximum level of perturbation was assumed to be +50% of the nominal values of the $A_p(z^{-1})$ and $B_p(z^{-1})$ polynomial coefficients. The value of p represents the order of the controller.

$p = 1$

1) The starting point is randomly selected and as no information is available on suitable weights they are all set to 1.

$\underline{X}^T = [0.3586 \quad 0.8694 \quad -0.2330]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 1.0]$

$J_{orig} = 1.792632$ $\qquad$ $J_{opt\,1} = 1.792632$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01
orig	0.9463	0.4071e-1
opt	0.9463	0.4071e-1

No improvement in the sensitivities so try a different starting point.

2) Again the starting point is randomly selected and the weights are all set to 1.

$\underline{X}^T = [3.8833 \quad 66.1931 \quad -93.0856]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 1.0]$

$J_{orig} = 1.792632$ $\qquad$ $J_{opt\,1} = 1.792632$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01
orig	0.9463	0.4071e-1
opt	0.9463	0.4071e-1

Again no improvement at all and note that the sensitivities are the same as in 1). A number of other starting points were tried with the same result and the weights were also changed but did not effect the outcome. It appears that with $p = 1$ no significant improvement in robustness can be achieved, so try increasing p.

$p = 2$

3) As with the previous results the starting point is randomly chosen and the weights all set to 1.

$\underline{X}^T = [-0.0101 \quad -0.0392 \quad 0.0012 \quad 0.0011 \quad -0.0226 \quad -0.0129 \quad -0.0351]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 1.0 \quad 1.0]$

$J_{orig} = 8.021354$ $\qquad$ $J_{opt\,2} = 1.789106$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02
orig	0.9824	1.7910	1.6980
opt	0.9452	0.6246e-1	0.4555e-1

Clearly with the $p = 2$ case there is scope for improvement, however it is necessary to adjust the weights to concentrate more on the dominant eigenvalue sensitivities. A different starting point is also used to see if a lower initial function value can be obtained.

4) The starting point is again randomly selected and the weights adjusted to place more emphasis on the dominant eigenvalues.

$\underline{X}^T = [0.0022 \quad -0.0242 \quad -0.0356 \quad -0.0037 \quad -0.0170 \quad 0.0089 \quad 0.0027]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 0.1 \quad 0.1]$

$J_{orig} = 1.783400$ $\qquad$ $J_{opt\,2} = 1.782130$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02
orig	0.9437	0.8548e-1	0.1224
opt	0.9428	0.1304	0.1660

Note that the initial function value is lower than the final function value of 3). Other starting points were tried but no lower initial value was obtained, hence the values used here represent a good starting point. Examining the sensitivities it is again clear that the weights need to be adjusted to concentrate even more on the dominant eigenvalue sensitivities.

5) Using the same starting point as 4), the weights were continually adjusted through a number of iterations leading to the result presented here.

$\underline{X}^T = [0.0022 \quad -0.0242 \quad -0.0356 \quad -0.0037 \quad -0.0170 \quad 0.0089 \quad 0.0027]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 0.0001 \quad 0.0001]$

$J_{orig} = 1.781173$ $\qquad$ $J_{opt\,2} = 0.527665$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02
orig	0.9437	0.8548e-1	0.1224
opt	0.4023	32.39	31.47

A significant improvement in the dominant eigenvalue sensitivities has been achieved but at the expense of the sensitivities of the other poles. At this stage it is useful to determine the pole positions for the perturbed closed-loop system to assess how good this result is.

Pole Positions:

Original	Perturbed	Distance Moved
$0.75 \pm j0.2$	$0.6981 \pm j0.3395$	$-0.0519 \pm j0.1395$
-0.01	0.2315	0.2415
-0.02	0.0013	0.0213

One of the controller poles has become quite significant and so this solution is not particularly good even though the dominant pole movement has been very much reduced. By adjusting the weights it should be possible to overcome this problem.

6) For the same starting point as 4), the results presented here are the best compromise that could be achieved between reducing the dominant eigenvalue sensitivity and increasing the sensitivities of the other poles.

$\underline{X}^T = [0.0022 \quad -0.0242 \quad -0.0356 \quad -0.0037 \quad -0.0170 \quad 0.0089 \quad 0.0027]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 0.0005 \quad 0.0005]$

$J_{orig} = 1.781181$ $\qquad$ $J_{opt\,2} = 1.023105$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02
orig	0.9437	0.8548e-1	0.1224
opt	0.5692	19.64	19.10

Pole Positions:

Original	Perturbed	Distance Moved
$0.75 \pm j0.2$	$0.7869 \pm j0.3611$	$0.0369 \pm j0.1611$
-0.0099	0.0967	0.1066
-0.0201	0.0025	0.0226

Controller Polynomials:

$F_p(z^{-1}) = 1.0 - 0.0640z^{-1} - 0.2027z^{-2}$

$G_p(z^{-1}) = 0.1940 - 0.2329z^{-1} + 0.0812z^{-2}$

$H_p(z^{-1}) = 0.0704 - 0.0282z^{-1}$

The transient response of the closed-loop system with this controller is shown in figure 5.4 which is clearly a significant improvement over the response for the minimum order controller shown in figure 5.3. Of course it is natural to consider what level of improvement can be achieved by increasing p further.

$p = 3$

7) Repeating the same procedure as above a good starting point was found and the weights adjusted to achieve a satisfactory compromise but with a significant level of improvement.

$\underline{X}^T = [-0.0724 \quad -0.6805 \quad -1.8138 \quad -2.9852 \quad -4.5878 \quad -2.7362 \quad -1.7356$
$\quad -3.1889 \quad 0.8057 \quad 3.4629 \quad 0.8311 \quad -1.8653 \quad -0.8210]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 0.0 \quad 0.0005 \quad 0.0005]$

$J_{orig} = 1.583238$ $\qquad$ $J_{opt\,1} = 1.163915$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02	-0.03
orig	0.8877	0.3603e-1	2.6900	2.6830
opt	0.6385	2.298	19.98	17.26

Pole Positions:

Original	Perturbed	Distance Moved
$0.75 \pm j0.2$	$0.8094 \pm j0.3646$	$0.0594 \pm j0.1646$
-0.0098	-0.0079	0.0019
-0.0201	$0.0222 \pm j0.0193$	$0.0423 \pm j0.0193$
-0.0301		$0.0523 \pm j0.0193$

This is quite a good result but comparing it with 6), the best result for the $p = 2$ case, it can be seen that the dominant pole sensitivities are not as good here. However it was expected that with $p = 3$ a better result would be obtained. Try a different starting point.

8) A number of alternative starting points were tried and the procedure for adjusting the weights carried out in each case. The best result obtained is presented here.

$$\underline{X}^T = [0.5634 \quad -0.2503 \quad 1.0725 \quad -1.8525 \quad -1.8031 \quad -3.0674 \quad 2.3256$$
$$-1.4839 \quad -0.9125 \quad -4.3826 \quad -6.6312 \quad -0.6958 \quad -6.5606]$$

$$\underline{W}^T = [1.0 \quad 1.0 \quad 0.002 \quad 0.0001 \quad 0.002]$$

$J_{orig} = 1.798684$ $\qquad$ $J_{opt\,1} = 1.017731$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02	-0.03
orig	0.9483	0.1555	0.4801	0.2811
opt	0.5661	9.5444	0.8613	9.8610

Pole Positions:

Original	Perturbed	Distance Moved
$0.75 \pm j0.2$	$0.7861 \pm j0.3603$	$0.0361 \pm j0.1603$
-0.01	-0.0195	-0.0095
-0.02	0.0044	0.0244
-0.03	0.0846	0.1146

Controller Polynomials:

$$F_p(z^{-1}) = 1.0 - 0.0366z^{-1} - 0.2071z^{-2} - 0.0040z^{-3}$$
$$G_p(z^{-1}) = 0.1966 - 0.2326z^{-1} + 0.0779z^{-2} + 0.0016z^{-3}$$
$$H_p(z^{-1}) = 0.0725 - 0.0290z^{-1}$$

Note that a better result was obtained from what would be classed as a worse starting point as the initial function value is higher than for 7). This illustrates the fact that these functions probably have many local minima and it is extremely important to try a number of alternative starting points. The transient response for the closed-loop system with this controller is shown in figure 5.5. Clearly this result is only a slight improvement over the $p = 2$ case which suggests that this type of result represents the best level of improvement that can be obtained. To verify consider increasing p again.

$p = 4$

9) After trying a number of randomly chosen starting points and suitable adjustment of the weights, this result was the best obtained.

$$\underline{X}^T = [-1.3462 \quad -3.7481 \quad -3.6908 \quad -0.6515 \quad -1.0806 \quad -3.9009 \quad -3.4066$$
$$3.4978 \quad -3.1640 \quad -1.5535 \quad -1.6996 \quad -0.0319 \quad 0.2454 \quad -1.4202$$
$$-1.6027 \quad 0.3568 \quad -1.4492 \quad -0.0576 \quad -2.1061 \quad -0.7171 \quad -1.1195]$$

$$\underline{W}^T = [1.0 \quad 1.0 \quad 0.003 \quad 0.002 \quad 0.002 \quad 0.003]$$

$J_{orig} = 2.694211$ $\qquad$ $J_{opt\,1} = 0.808537$

Eigenvalue Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02	-0.03	-0.04
orig	1.1430	0.9609	2.7600	1.1640	4.5700
opt	0.4816	7.0310	3.8370	3.1740	6.9960

Pole Positions:

Original	Perturbed	Distance Moved
$0.75 \pm j0.2$	$0.7530 \pm j0.3517$	$0.0030 \pm j0.1517$
-0.0099	0.1210	0.1309
-0.0199	-0.0173	0.0026
-0.03	-0.0322	-0.0022
-0.0401	0.0046	0.0447

Controller Polynomials:

$$F_p(z^{-1}) = 1.0 - 0.0359z^{-1} - 0.2681z^{-2} - 0.0131z^{-3} - 0.1481e-3z^{-4}$$
$$G_p(z^{-1}) = 0.2359 - 0.2870z^{-1} + 0.0913z^{-2} + 0.0051z^{-3} + 0.5933e-4z^{-4}$$
$$H_p(z^{-1}) = 0.0754 - 0.0302$$

On initial inspection this appears to be significantly better than both the $p = 2$ and the $p = 3$ cases. However one of the controller poles has moved considerably but even at 0.1210 it could be argued that it would still not significantly influence the shape of the response. The transient response associated with this controller is shown in figure 5.6 and clearly it is better than has been obtained previously, but only just. It certainly suggests that the benefits of considering controllers beyond third or fourth order is questionable.

5.3.7 Results for the Eigenstructure Differential Cost Function

This cost function contains the eigenvector differential and to help determine if this term leads to better solutions, the same starting points for the best results for the eigenvalue differential cost function are used. Also from the previous set of results it is clear that the case of $p = 1$ need not be considered.

$p = 2$

10) Using the same starting point as 6) and refining the weights lead to two sets of results presented here and in 11).

$\underline{X}^T = [0.0022 \quad -0.0242 \quad -0.0356 \quad -0.0037 \quad -0.0170 \quad 0.0089 \quad 0.0027]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 5.0e-4 \quad 5.0e-4 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0e-2 \quad 1.0e-4]$

$J_{orig} = 1.921558$ $\qquad$ $J_{opt\,2} = 1.789696$

Eigenvalue/Eigenvector Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02
orig	0.9437	0.8548e-1	0.1224
opt	0.9455	0.2709e-3	0.3976e-1
orig	2.6830e2	4.1390e2	4.5320e4
opt	7.7010	2.1200	1.7510e2

11) Second result

$\underline{X}^T = [0.0022 \quad -0.0242 \quad -0.0356 \quad -0.0037 \quad -0.0170 \quad 0.0089 \quad 0.0027]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 5.0e-4 \quad 5.0e-4 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0e-2 \quad 1.0e-6]$

$J_{orig} = 1.782585$ $\qquad$ $J_{opt\,2} = 1.317259$

Eigenvalue/Eigenvector Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02
orig	0.9437	0.8548e-1	0.1224
opt	0.7054	12.07	11.75
orig	2.6830e2	4.1390e2	4.5320e4
opt	2.5420e4	1.2840e5	9.4000e4

Examining the results in 10) and 11) it can be seen that there appears to be a conflict between the dominant eigenvalue sensitivities and their associated eigenvector sensitivities. When the dominant eigenvalue sensitivities are decreased the eigenvector sensitivities increase and vice versa. A number of alternative starting points were tried but all were subject to this conflict and no satisfactory result could be obtained.

A value of $p = 3$ was used with the same starting point as 8) to determine if this problem could be overcome by introducing more design freedom. However the conflict was still present and it is felt that increasing p further would not help in this situation.

Perhaps a better approach to the application of this cost function would be to start at the sub-optimal points found using the eigenvalue differential cost function. The aim would then be to try and reduce the eigenvector sensitivities without adversely affecting the eigenvalue sensitivities.

12) Using the sub-optimal point found in 6) as the starting point and again refining the weights as before.

$\underline{X}^T = [0.0086 \quad -0.0173 \quad -0.0420 \quad 0.0032 \quad -0.0234 \quad 0.0063 \quad -0.0018]$

$\underline{W}^T = [1.0 \quad 1.0 \quad 5.0e-4 \quad 5.0e-4 \quad 1.0 \quad 1.0 \quad 1.0e-4 \quad 1.0e-7 \quad 5.0e-10]$

$J_{orig} = 1.116970$ $\qquad$ $J_{opt\,2} = 1.098190$

Eigenvalue/Eigenvector Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02
orig	0.5692	19.64	19.10
opt	0.6052	17.55	17.06
orig	9.3860e7	1.4490e8	1.0570e10
opt	6.6140e7	1.0200e8	7.4690e9

This result is quite encouraging as some improvement has been made. However by examining the transient response of the closed-loop system with the controller for this result, figure 5.7, it can be seen that the level of improvement in the eigenvector sensitivities really needs to be much greater to have any significant effect on performance robustness. A higher value of p could achieve this.

$p = 3$

13) This time the sub-optimal point found in 8) is used as the starting point and appropriate weights selected.

$$\underline{X}^T = [133.8 \quad -25.6 \quad 395.1 \quad -365.1 \quad 12.1 \quad -1214.0 \quad 173.6 \quad -254.0 \quad -23.5 \quad -622.4 \quad -1672.8 \quad -205.3 \quad -929.4]$$

$$\underline{W}^T = [1.2 \quad 1.2 \quad 2.0e-3 \quad 1.0e-4 \quad 2.0e-3 \quad 1.0 \quad 1.0 \quad 1.0e-6 \quad 1.0e-6 \quad 1.0e-6 \quad 1.0e-4]$$

$J_{orig} = 1.639185$ $\qquad$ $J_{opt\,2} = 1.320894$

Eigenvalue/Eigenvector Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02	-0.03
orig	0.5661	9.5440	0.8613	9.8610
opt	0.5726	9.4594	0.6515	9.5769
orig	2.4360e3	4.5300e7	5.1630e6	1.0150e7
opt	8.3265e2	3.3762e7	3.1700e6	1.4671e7

This is a slightly better improvement than in 12) but by comparing the transient response shown in figure 5.8 with that in figure 5.5 (the result from the eigenvalue differential cost function), it can be seen that no real improvement has been achieved. Assuming this is the typical improvement that can be expected when p is increased, it would suggest that a very high order controller is needed to gain any real advantage over the controllers obtained using the eigenvalue differential cost function.

5.3.8 Results for the Transient Response Differential Cost Function

From the modal decomposition it was seen that for performance robustness a slightly different form for the eigenvector differential was more appropriate and was included in the transient response differential cost function. From the previous results it would seem reasonable to start from the sub-optimal points found using the eigenvalue differential cost function and only the case of $p = 3$ is considered.

14) Using the sub-optimal point found in 8) and after refinement of the weights the following result was obtained.

$$\underline{X}^T = [133.8 \quad -25.6 \quad 395.1 \quad -365.1 \quad 12.1 \quad -1214.0 \quad 173.6 \quad -254.0 \quad -23.5 \quad -622.4 \quad -1672.8 \quad -205.3 \quad -929.4]$$

$\underline{W}^T = [1.0 \quad 1.0 \quad 2.0e-3 \quad 1.0e-4 \quad 2.0e-3 \quad 1.0 \quad 1.0$

$1.0e-4 \quad 1.0e-4 \quad 1.0e-4 \quad 1.0e-7]$

$J_{orig} = 1.639185$ $\qquad$ $J_{opt\,2} = 1.320894$

Eigenvalue/Eigenvector Sensitivities:

	$0.75 \pm j0.2$	-0.01	-0.02	-0.03
orig	0.5661	9.5440	0.8613	9.8610
opt	0.5722	9.6320	0.3183	9.4160
orig	1.5640e6	4.7530e7	1.2000e8	1.1190e8
opt	8.9320e4	1.5950e7	3.0490e7	1.5060e7

This is very similar to the result in 13) and by examining the transient response in figure 5.9 it can be seen that no real improvement over that shown in figure 5.5 has been achieved. Again it is expected that a very high order controller would be required to make any significant improvement in the eigenvector sensitivities.

5.3.9 Results for the Conditioning Cost Function

This cost function is quite different to the others as it is not directly concerned with sensitivities. No weights are needed so the problem of iteratively refining them does not exist. On the basis of the previous results it was decided to restrict the investigation to the cases of $p = 2$ and $p = 3$.

$p = 2$

15) A number of randomly selected starting points were tried and this was the best result obtained.

$\underline{X}^T = [-0.0243 \quad -0.0113 \quad -0.0083 \quad -0.1591 \quad -0.0685 \quad -0.2518 \quad 0.0159]$

$J_{orig} = 58.251362$ $\qquad$ $J_{opt\,2} = 14.445608$

Pole Positions:

Original	Perturbed	Distance Moved
$0.75 \pm j0.2$	$0.8854 \pm j0.3750$	$0.1354 \pm j0.1750$
-0.01	-0.0098	0.0002
-0.02	-0.0200	0.0000

This is clearly not as good as the result obtained using the eigenvalue differential cost function for the $p = 2$ case, which can be verified by comparing the transient response in figure 5.10 with that in figure 5.4. Indeed the response is only slightly better than that for the minimum order controller shown in figure 5.3. A higher value of p may yield a more desirable result.

$p = 3$

16) Again a number of randomly selected starting points were tried and this represents the best result obtained.

$$\underline{X}^T = [-0.5621 \quad -0.9059 \quad 0.3577 \quad 0.3586 \quad 0.8694 \quad -0.2330 \quad 0.0388$$
$$0.6619 \quad -0.9309 \quad -0.8931 \quad 0.0594 \quad 0.3423 \quad -0.9846]$$

$J_{orig} = 1.847105e3$ $\qquad$ $J_{opt\,2} = 25.646076$

Pole Positions:

Original	Perturbed	Distance Moved
$0.75 \pm j0.2$	$0.8872 \pm j0.3751$	$0.1372 \pm j0.1751$
-0.01	-0.0082	0.0018
-0.02	-0.0239	-0.0039
-0.03	-0.0300	0.0000

This result is very similar to that in 15) and again the transient response shown in figure 5.11 is significantly worse than that in figure 5.5 (the best result from the eigenvalue differential cost function).

5.4 Summary and Discussion of the Results

This chapter has considered the application of the proposed robust design method to a simple polynomial system to help illustrate the design procedure and give an indication of the improvement that can be achieved.

With the application of the method a number of issues regarding its implementation arise. These aspects are important as they can have a significant effect on the performance of the design procedure. From the results it is clear that it is necessary to perform the optimisation, which involves carrying out the state space design, many times to help in the selection of appropriate weights. Because of this, it is desirable to implement the method such that the optimisation can be performed reasonably quickly without sacrificing accuracy. For these reasons a joint Pro-Matlab / Fortran 77 implementation is adopted.

The first step of the design is to transform the system to state space form which then allows the model uncertainty to be defined. In this case the uncertain parameters are of the same magnitude and vary by the same amount leading to a simple definition of the uncertainty.

The problems with the parametric method of Fahmy and O'Reilly (1988) were then discussed in depth for this example and it was concluded that the method of Daley (1990) should be used for the state space design. It was then possible to show how to determine the number of free parameters in the design and hence decide on an appropriate optimisation routine.

The results for each cost function were then presented for values of p (the order of the controller) in the range of 1 to 4. Note that the lowest value of p must be chosen such that $r+m>n$, which is a requirement of the parametric state space design, and that it may not necessarily always be 1.

There are a few interesting points to note about the results. Firstly, the refinement of the weights is relatively easy with typically up to 4-5 iterations needed to find a good solution. Their choice is quite straightforward if based on the associated sensitivities and if a model of the perturbed system is available such that the perturbed closed-loop pole positions can be examined.

Secondly, the cost functions which utilise eigenvector differential information, do not appear to produce results which are significantly better than those obtained using the eigenvalue differential cost function. Of course it is expected that the eigenvalue sensitivities will be the most important as they affect the rate of rise and decay of the transient response. However the eigenvector sensitivity information should have helped produce better solutions. Examining the results shows that a conflict between the eigenvalue and eigenvector sensitivities appears to arise. This conflict prevents good solutions from being obtained. The most probable reason for this is that the method does not have the freedom necessary to reduce all the sensitivities. This was verified by the results which showed that for higher values of p, where there is greater design freedom, there was a slightly better improvement. The problem, however, is that to yield significantly better results it is expected that a very high order controller would be required.

Lastly considering the transient responses of the various controllers it is clear that a significant improvement can be achieved using the eigenvalue differential cost function but the conditioning cost function only produced a slight improvement over the minimum order controller. This is almost certainly due to the fact that conditioning is quite a general criteria which, among other factors, minimises an upper bound on the sensitivities of all the eigenvalues. The results for the eigenvalue differential cost function with all the eigenvalues weighted evenly showed that very little improvement in their sensitivities could be achieved, which explains what is effectively happening with the conditioning cost function.

This highlights the fact that it is often necessary to sacrifice the sensitivity of certain poles such that the sensitivity of other poles can be improved. In this case the controller or added poles are placed close to zero and as such have little effect on the transient response compared with the two dominant poles. The sensitivity of the dominant poles can then be decreased by allowing the sensitivities of the added poles to become large, which is verified by examining the results. This trade off is quite important as it requires the choice of the weights to be made with some care so as not to allow the added poles to have a significant effect on the transient response when the system is subject to model uncertainty. It is really necessary to have some idea of the maximum level of parameter variations in order that the movement of the added poles subject to this maximum change can be checked to ensure that they do not become dominant. In this case the aim was to keep the added poles within the $z = 0.1$ circle for the maximum variation of 50%.

Figure 5.6 illustrates the best transient response behaviour when the system is subject to parameter variations. As the order of the controller, p is increased there are more free parameters in the design process, so it is expected that higher order controllers would yield better results which is verified here as the corresponding controller for figure 5.6 is fourth order.

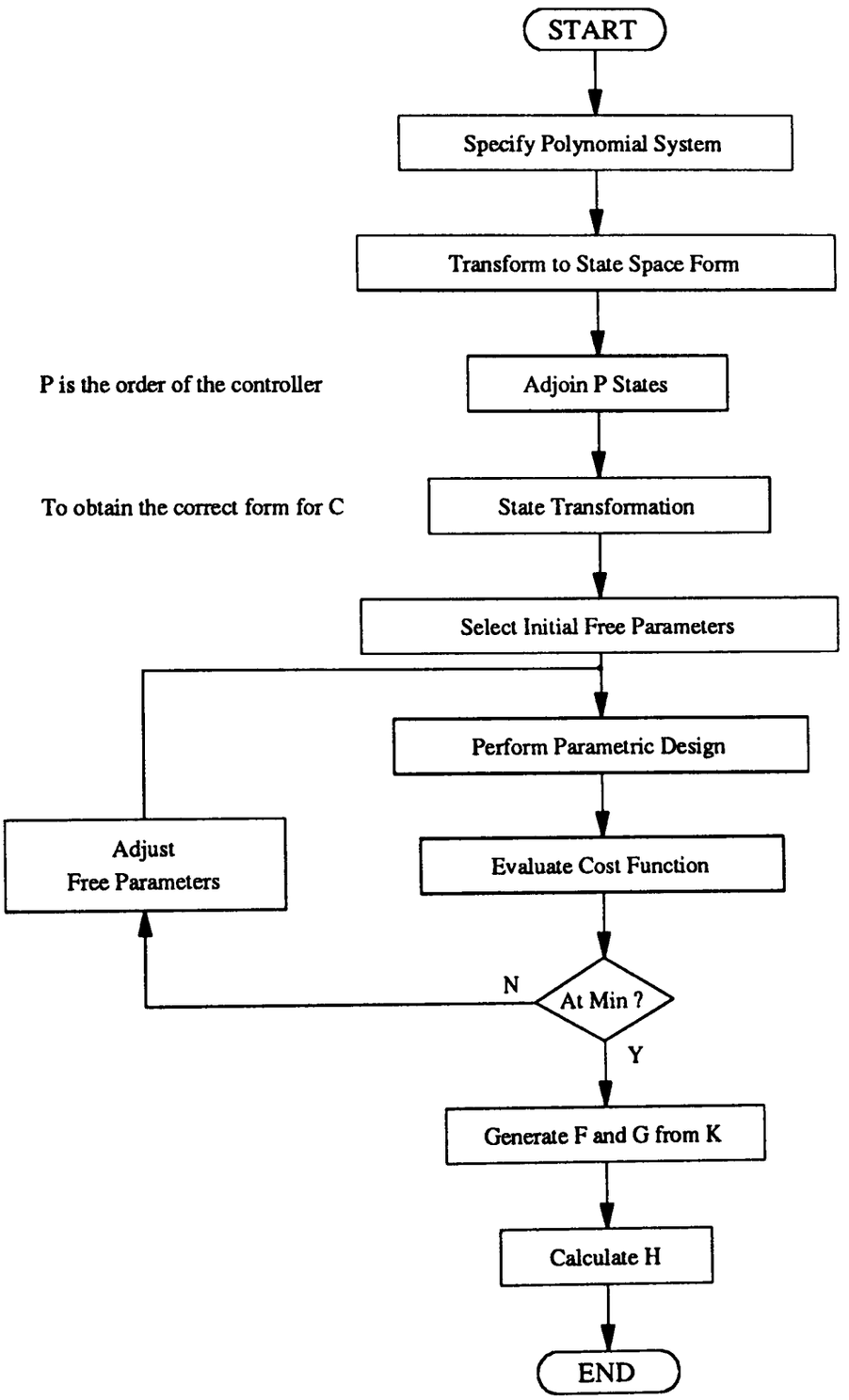


Figure 5.1 - Flowchart of the Robust Polynomial Controller Design Procedure

Note: F,G and H are controller polynomials in terms of the backward shift operator, z^{-1}.

K is the state space output feedback matrix.

The parametric design stage is expanded in figure 5.2.

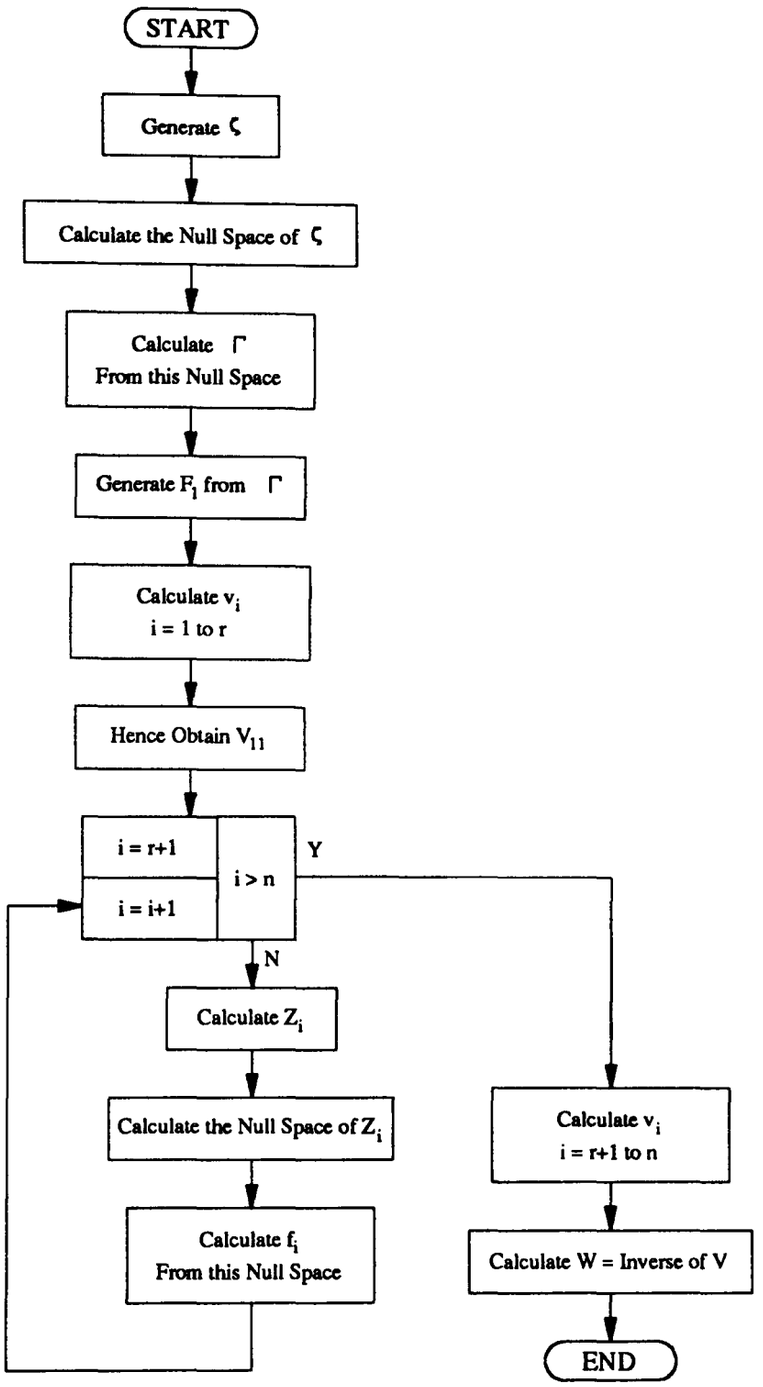


Figure 5.2 - Flowchart of the Parametric State Space Design Procedure

Note:

$$\zeta = \begin{bmatrix} \alpha_{11}\Delta S_{11} & \alpha_{12}\Delta S_{12} & \cdots & \alpha_{1r}\Delta S_{1r} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha_{q1}\Delta S_{q1} & \alpha_{q2}\Delta S_{q2} & \cdots & \alpha_{qr}\Delta S_{qr} \end{bmatrix}, \quad \underline{\Gamma} = \begin{bmatrix} \underline{f}_1 \\ \cdot \\ \cdot \\ \underline{f}_r \end{bmatrix}$$

α's are arbitrary scalars, $q = n - r$ and $\Delta S_{ik} = [((\lambda_{i+r} I - A)^{-1} - (\lambda_k I - A)^{-1})B]_r$

$V = [\underline{v}_1 \cdots \underline{v}_n]$

$\underline{v}_i = (\lambda_i I - A)^{-1} B \underline{f}_i$

$Z_i = I - F_1 V_{11}^{-1} [(\lambda_i I - A)^{-1} B]_r$

V_{11} = the 1st r rows of the 1st r right eigenvectors.

$F_1 = [\underline{f}_1 \cdots \underline{f}_r]$ and $F_2 = [\underline{f}_{r+1} \cdots \underline{f}_n]$

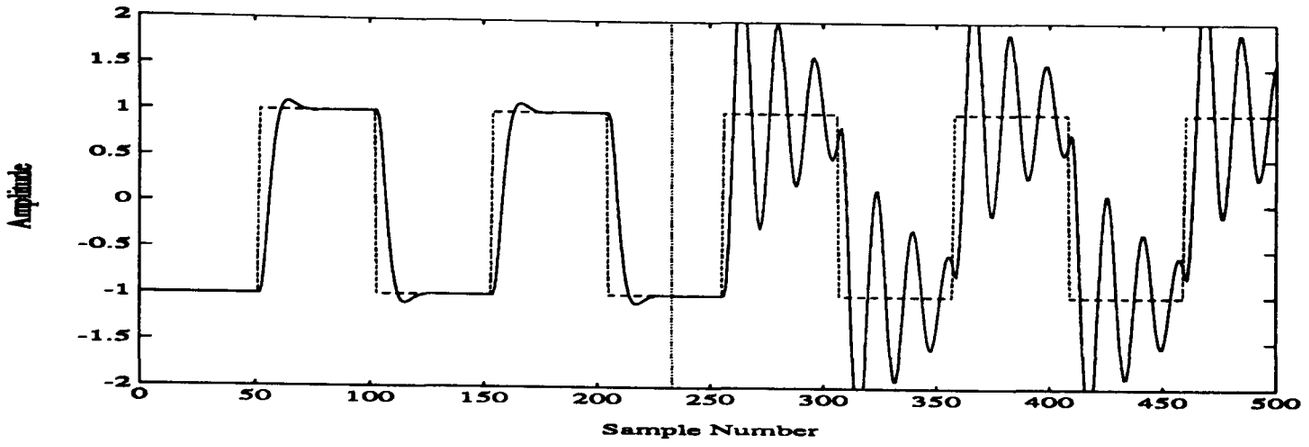


Figure 5.3 - Response of the Closed-Loop System with the Controller derived from the Diophantine Equation

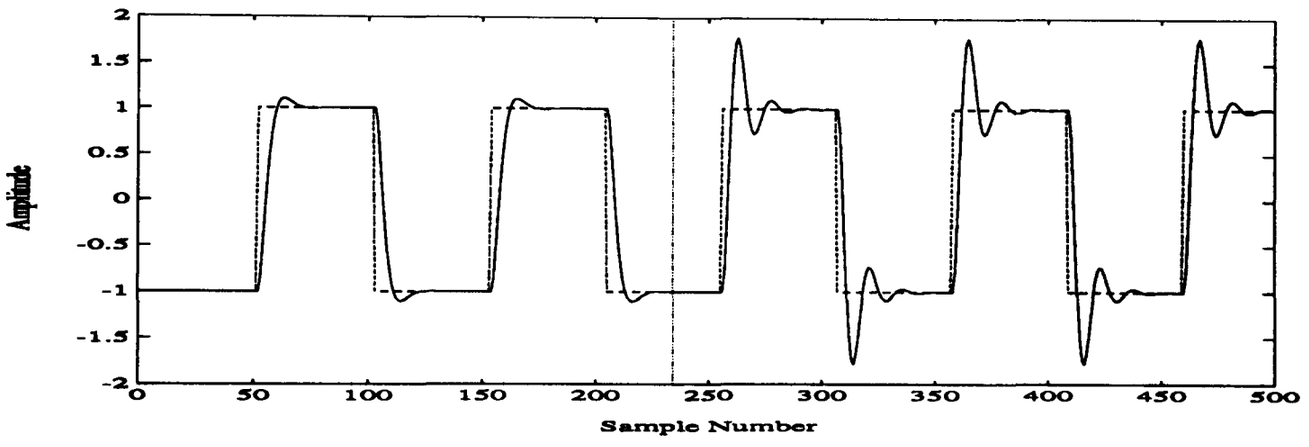


Figure 5.4 - Response of the Closed-Loop System with the p=2 Controller from Cost Function 1

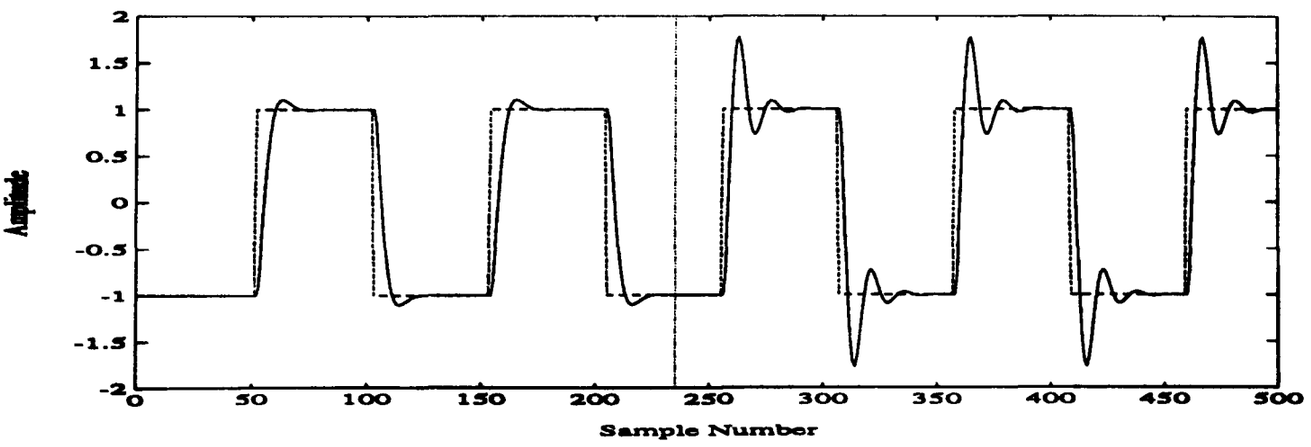


Figure 5.5 - Response of Closed-Loop System with the p=3 Controller from Cost Function 1

-------- Input _____ Output

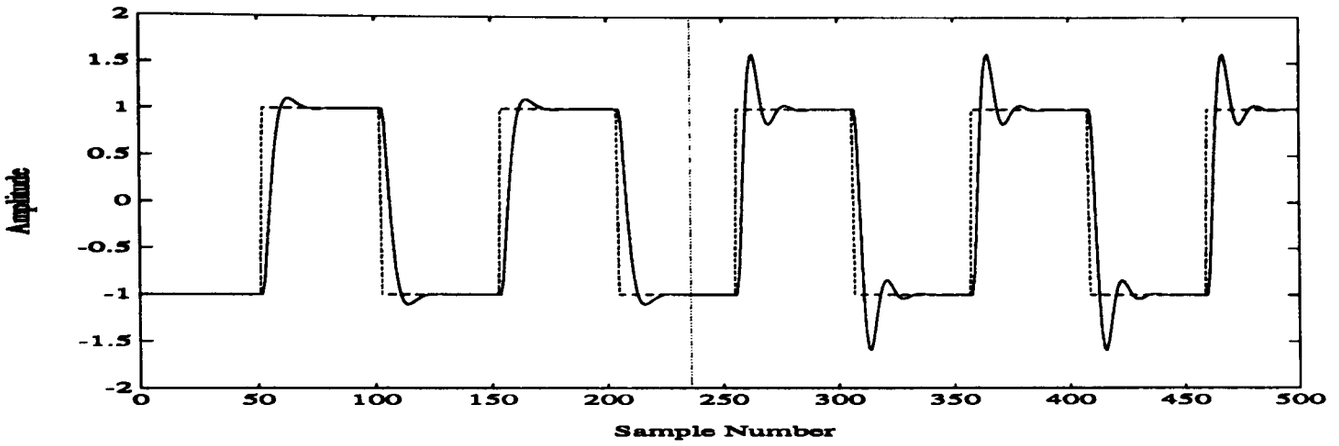


Figure 5.6 - Response of Closed-Loop System with the p=4 Controller from Cost Function 1

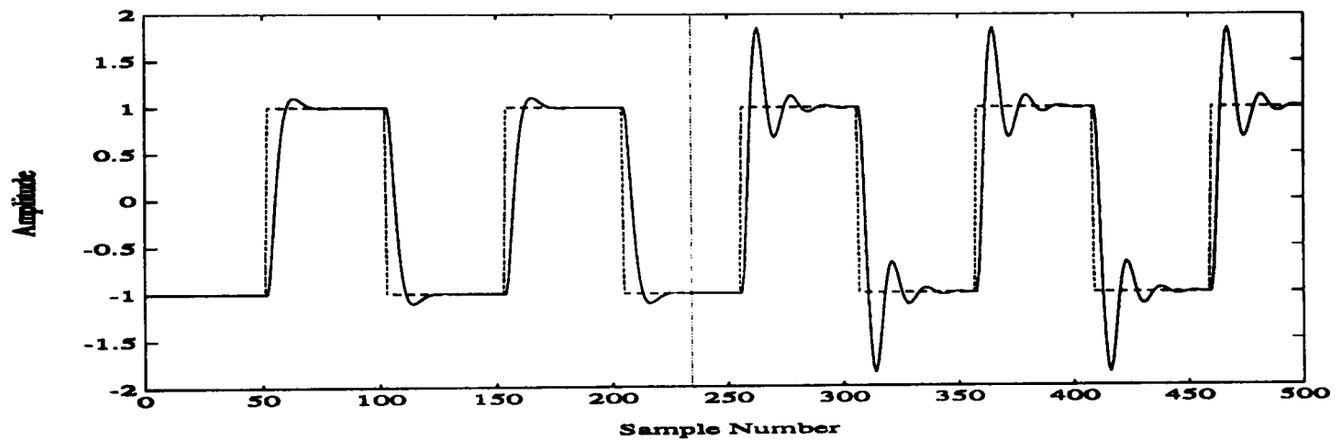


Figure 5.7 - Response of Closed-Loop System with the p=2 Controller from Cost Function 2

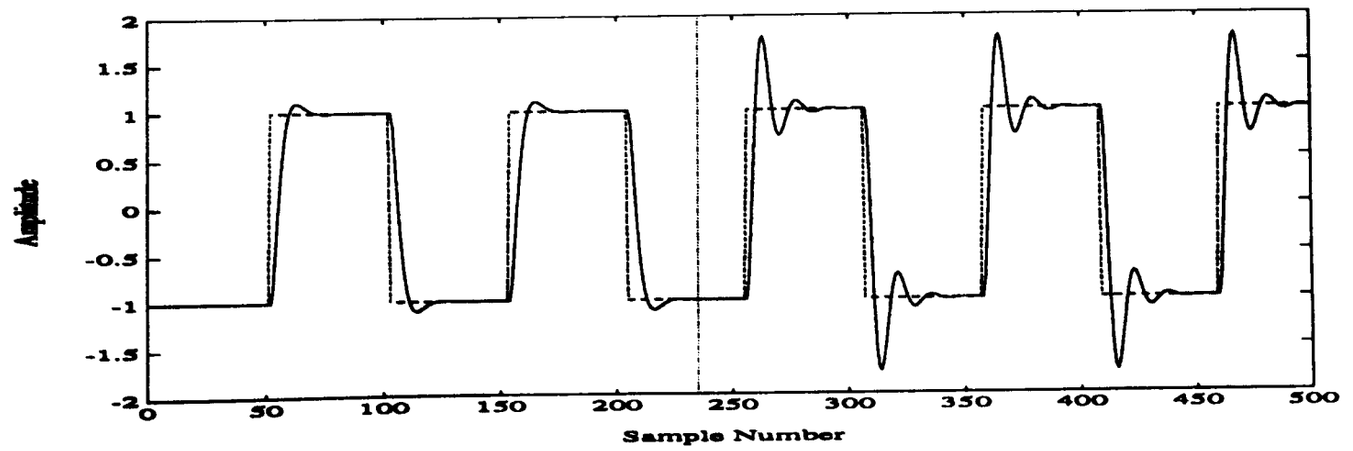


Figure 5.8 - Response of Closed-Loop System with the p=3 Controller from Cost Function 2

-------- Input _____ Output

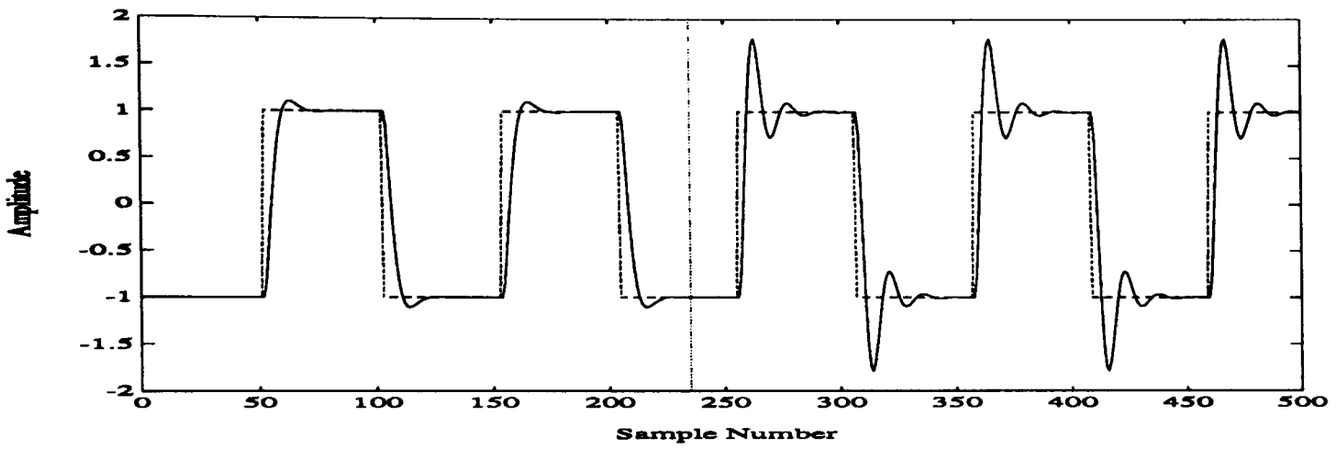


Figure 5.9 - Response of Closed-Loop System with the p=3 Controller from Cost Function 3

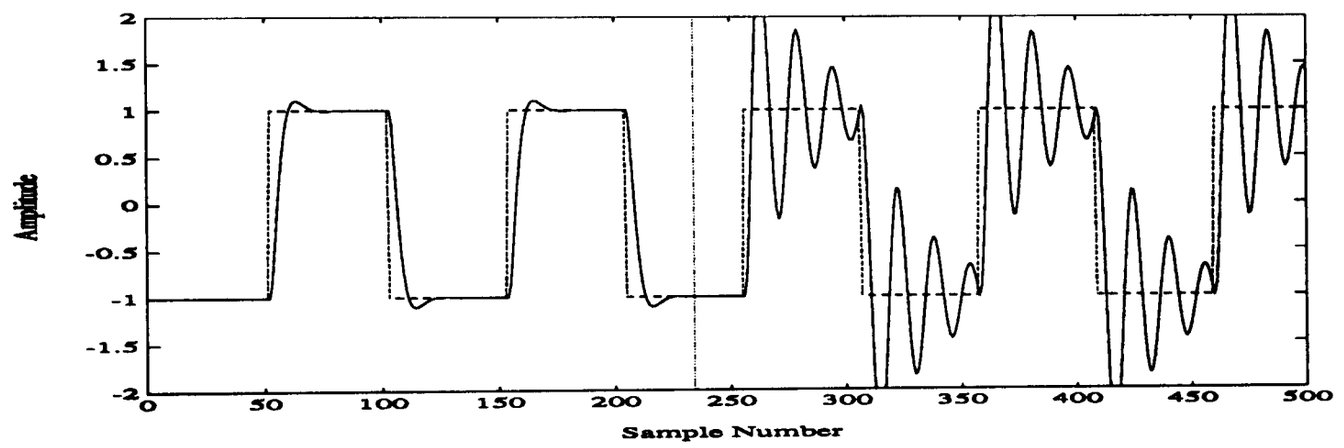


Figure 5.10 - Response of Closed-Loop System with the p=2 Controller from Cost Function 4

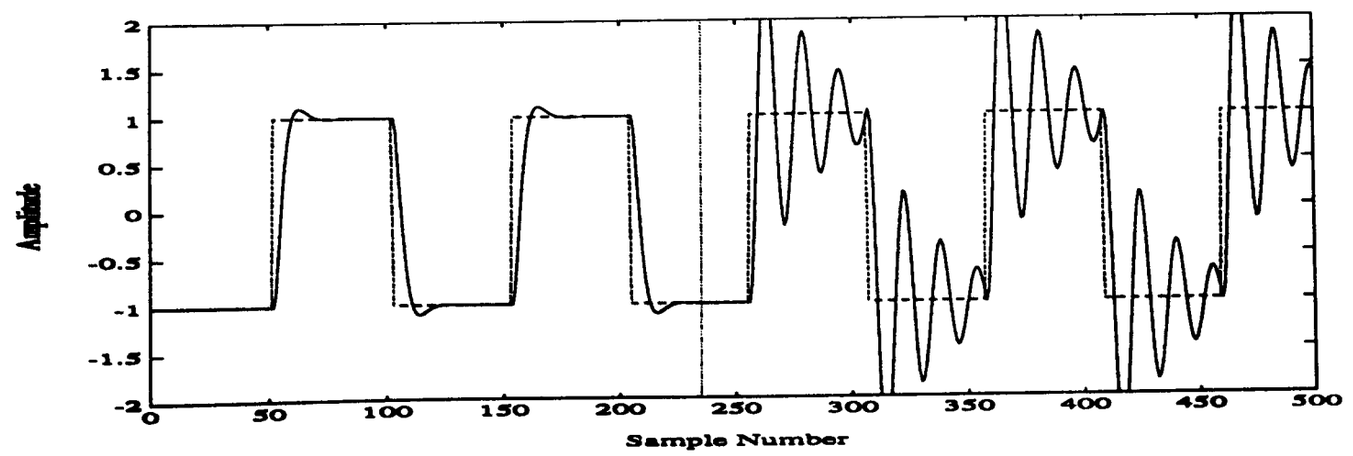


Figure 5.11 - Response of Closed-Loop System with the p=3 Controller from Cost Function 4

-------- Input _____ Output

REFERENCES

Daley, S. (1990)
'On Eigenstructure Assignability using parametric output feedback'
Brunel University Control Engineering Centre Internal Report, August 1990

Edgar, S.L. (1989)
'Advanced Problem Solving with FORTRAN 77'
Science Research Associates, Henley-on-Thames, England

Fahmy, M.M. and O'Reilly, J. (1988)
'Multistage Parametric Eigenstructure Assignment by Output Feedback Control'
International Journal of Control, v48, n1, pp 97-116

NAG (1990)
'Fortran Library Manual Mk 14'
Numerical Algorithms Group Ltd, Oxford, U.K.

PRO-MATLAB (1990)
'PRO-MATLAB User Guide'
Mathworks Inc, South Natick, Mass, U.S.A.

Wellstead, P.E. and Sanoff, S.P. (1981)
'Extended Self-Tuning Algorithm'
International Journal of Control, v34, n3, pp 433-455

CHAPTER 6

APPLICATION TO A HYDRAULIC RIG

6.1 Introduction

Hydraulic systems include some of the most powerful and fast-moving devices in engineering and are widely used in many industrial plants. The design of controllers for such systems can be quite difficult due to a number of factors including non-linearities (such as the relation between flowrate and pressure), variations in oil viscosity resulting from changing temperature and load dependent gain (Daley, 1990). Applying the proposed robust design procedure to a realistic industrial system of this type would provide useful information on the general applicability of the method. The hydraulic test rig considered in Daley (1987) would be a good system to consider as it was designed and built to be representative of real industrial plants.

The rig consists of a stiff shaft which is driven by a hydraulic motor and loaded with a hydraulic pump. The oil flow to the motor is controlled by an electrohydraulic servo-valve and the pressure differential across the pump can be changed to increase or decrease the loading on the shaft. A schematic of the hydraulic circuit of the rig is shown in figure 6.1.

Daley (1987) considered the application of self-tuning control to this system and assessed the performance of the closed-loop system when subjected to varying supply pressure and load. The results were compared to a more traditional proportional plus integral plus derivative (PID) controller and shown to be significantly better.

The aim here is to design a fixed term controller for the system which is robust to varying supply pressure and load, such that the performance is better than the PID controller and hopefully comparable to that of the self-tuning controller. Of course the self-tuning controller will in general be more flexible to variations in the plant parameters and perform well over a wide range of possible parameter values. However over a specified range, for which the robust fixed term controller is designed, it is hoped that the relative performance of the two controllers will be similar. Also, in this range the fixed term controller may even perform slightly better as there will be no tuning transients as with the self-tuning controller.

As it is not possible to perform tests on the real rig, it is necessary to construct an accurate simulation of the system. To do this, use can be made of commercially available simulation languages which contain the basic integration algorithms necessary to perform a continuous time

simulation, as well as an environment in which quite complex system equations can be specified. In this case the Advanced Continuous Simulation Language (ACSL) package is used and further details on the simulation can be found in the following section.

Using the simulation it is possible to show how the proposed method can be applied to a real plant, from system identification to controller design and implementation. Three robust fixed term controllers are developed, the first considering only variations in the supply pressure, the second only variations in the load and the third variations in both.

6.2 Nonlinear Simulation and Model Identification

A model of the rig can be developed by considering the equations for each part of the system. Figure 6.1 shows a schematic of the hydraulic circuit of the rig.

The flowrate, Q_v, through the valve can be approximated by the square root relationship of an orifice

$$Q_v = K_\theta X_s (P_s - P_m)^{0.5} \tag{6.1}$$

where X_s is the spool valve displacement, P_s is the supply pressure, P_m is the pressure differential across the motor and K_θ is the valve flow coefficient. For continuity of flow

$$Q_v = C_r \dot{\theta} + \left(\frac{V_t}{2B} \right) \dot{P}_m + K_1 P_m \tag{6.2}$$

where θ is the shaft position, C_r is the motor displacement, V_t is the total trapped volume, B is the oil bulk modulus and K_1 is a leakage coefficient. The motor torque is

$$T_m = P_m C_r \eta_m \tag{6.3}$$

where η_m is the efficiency of the motor. Neglecting static and coulomb friction

$$T_m = I\ddot{\theta} + D\dot{\theta} + T_p \tag{6.4}$$

where I is the total inertia of the pump, motor and shaft, D is the viscous friction coefficient and

$$T_p = P_p C_r \eta_p^{-1} \tag{6.5}$$

where P_p is the pressure differential across the pump and η_p is the efficiency of the pump.

If it is assumed that the dynamics of the servo-valve are much faster than the dynamics of the load, the servo and torque motor can be approximated by a pure gain term, i.e.

$$X_s \approx K_s u \tag{6.6}$$

where u is the input voltage to the torque motor. The output voltage is given by

$$y = K_t \dot{\theta} \tag{6.7}$$

where K_t is the tachometer constant.

These equations are very simple to simulate in ACSL and the program for the open-loop system can be found in appendix D. One problem however is the choice of suitable values for the many constants in the above equations. From the data supplied by the manufactures of some of the components of the rig, tests on the actual rig, Dholiwar (1991), and a process of trial and error, the following values were obtained

B = 7000.0e5 N/m^2	I = 1.08e-4 Kg m^2/rad
K_θ = 2.4e-6	D = 5.94e-4 Kg m^2/rad s
K_s = 0.0625 m/V	V_t = 3.51e-5 m^3
C_r = 9.56e-7 m^3/rad	K_1 = 2.12e-13 m^{4}s/Kg
η_m = 1.0	K_t = 8.0e-3 V s/rad
η_p = 1.0	

The nominal value of the supply pressure, P_s is taken as 68.96e5 N/m^2 (1000 lbf/in^2), and it is assumed that it could increase up to a maximum value of 137.93e5 N/m^2 (2000 lbf/in^2). The loading can be varied by changing the value of the pressure differential across the pump, P_p. Its nominal value is 22.98e5 N/m^2 (333 lbf/in^2) and the maximum load is assumed to correspond to a pressure differential of 44.83e5 N/m^2 (650 lbf/in^2).

Clearly from the above it would be possible to derive a model theoretically but as the aim is to show how the proposed method would be applied to a real plant, system identification techniques will be used. For the purposes of identifying a model of the open-loop system, a 1.5Hz square wave input is applied as shown in figure 6.2, and the input and output sampled at 83Hz (Daley, 1987). From this data and using the technique of instrumental variables (IV), the results in table 6.1 were obtained

No. of $A_p(z^{-1})$ parameters	No. of $B_p(z^{-1})$ parameters	Time Delay	$A_p(z^{-1})$ parameters	$B_p(z^{-1})$ parameters	V_N	Akaike's FPE
1	1	1	a_1 = -0.0152	b_0 = 2.5988	0.007059	0.007287
2	1	1	a_1 = -0.0356 a_2 = -0.0022	b_0 = 2.5391	0.007056	0.007401
2	2	1	a_1 = -0.9406 a_2 = 0.0954	b_0 = 1.3090 b_1 = -0.9005	0.007755	0.008263
2	2	2	a_1 = -4.9461 a_2 = 2.4478	b_0 = -8.9773 b_1 = 5.0231	0.08104	0.08636
3	2	1	a_1 = 3.3135 a_2 = 0.1688 a_3 = -0.1001	b_0 = 8.7602 b_1 = 2.8040	0.02635	0.02853

Table 6.1 - IV Estimation Results

The most appropriate model is indicated by the lowest value for the loss function, V_N and Akaike's final prediction error (FPE). Note that the loss function is defined as (Soderstrom and Stoica, 1989)

$$V_N = \frac{1}{N}\sum_{t=1}^{N} \varepsilon^2(t,\hat{\theta}_p) \tag{6.8}$$

where N is the number of data points and $\varepsilon(t,\theta_p)$ is the prediction error corresponding to the parameter vector θ_p, i.e.

$$\varepsilon(t,\theta_p) = y(t) - \hat{y}(t \mid t-1;\theta_p) \tag{6.9}$$

and $\hat{y}(t \mid t-1;\theta_p)$ denotes a prediction of $y(t)$ given the data up to and including time $t-1$, based on the model parameter vector θ_p. Akaike's FPE is then defined as (Soderstrom and Stoica, 1989; PRO-MATLAB, 1990)

$$\text{FPE} = V_N \frac{1+P/N}{1-P/N} \tag{6.10}$$

where P is the number of parameters in the model.

The first order model is the best fit, hence the design is based on

$$A_p(z^{-1}) = 1.0 - 0.0152z^{-1} \tag{6.11}$$

$$B_p(z^{-1}) = 2.5988z^{-1} \tag{6.12}$$

Note that the theoretical model (Daley, 1987) is second order. This suggests that in practice one of the open-loop poles of the system is significantly more dominant than the other.

To aid in the design process it is desirable to obtain the model of the system when subject to variations in the factors of interest. With the supply pressure at its maximum value of 137.93e5 N/m^2 (2000 lbf/in^2), the model becomes

$$A_p(z^{-1}) = 1.0 + 0.0026z^{-1} \tag{6.13}$$

$$B_p(z^{-1}) = 4.2164z^{-1} \tag{6.14}$$

and with the maximum load, corresponding to P_p = 44.83e5 N/m^2 (650 lbf/in^2)

$$A_p(z^{-1}) = 1.0 - 0.1066z^{-1} \tag{6.15}$$

$$B_p(z^{-1}) = 1.6903z^{-1} \tag{6.16}$$

and lastly when both the supply pressure and the load increase to their maximum values

$$A_p(z^{-1}) = 1.0 + 0.0094z^{-1} \tag{6.17}$$

$$B_p(z^{-1}) = 3.8097z^{-1} \tag{6.18}$$

Note that when the changes occur, a first order model may not be the most appropriate but as the design is to based on a first order model, the above help to give an indication of the typical variation in the parameters of the model.

The simulation of the closed-loop system is also quite straightforward and the corresponding ACSL program is in appendix D. The effect of implementing the controller on a digital computer has been taken into account by incorporating the controller polynomials in a discrete block. The sample rate for the input and output, from which the control signal is calculated, is again 83Hz.

6.3 Controller Design

Having established a suitable model of the hydraulic rig, it is possible to move onto the design of the control system. Generally the design of fixed term polynomial controllers is based on the minimum order solution of the diophantine equation. As such the robust polynomial controllers will be compared against the minimum order one and not a PID controller as in Daley (1987). However from the work of Daley (1987) it should be possible to draw general conclusions about the performance relative to the PID controller.

On the basis of the results obtained in the previous chapter, the eigenvalue differential cost function will be used throughout. The NAG library routine E04JAF is again used to perform the numerical optimisation.

This section is split into two subsections, the first dealing with the design of the minimum order controller and the second covering the design of the three robust controllers.

6.3.1 Minimum Order Polynomial Controller Design

As the system does not contain integral action it is necessary to cascade a digital integrator with it, hence the design is actually based on

$$A_p(z^{-1}) = 1.0 - 1.0152z^{-1} + 0.0152z^{-2} \tag{6.19}$$

$$B_p(z^{-1}) = 2.5988z^{-1} \tag{6.20}$$

and the desired closed-loop pole positions are chosen as $0.65 \pm j0.3$ for a good compromise between rise time and overshoot.

The minimum order controller is then

$$F_p(z^{-1}) = 1.0 \tag{6.21}$$

$$G_p(z^{-1}) = -0.1096 + 0.1914z^{-1} \tag{6.22}$$

$$H_p(z^{-1}) = 0.0818 \tag{6.23}$$

It is also useful, for the purposes of comparison, to gain an idea of how far the closed-loop poles move when changes in the system occur. Using the models from the previous section and the minimum order controller derived here, it is possible to deduce that

	Pole Positions	Distance Moved
Max increase in P_s only	$0.7298 \pm j0.5214$	0.2353
Max increase in P_p only	$0.6459 \pm j0.1136$	0.1864
Max increase in P_s and P_p	$0.7041 \pm j0.4733$	0.1815

Table 6.2 - Pole Positions for the Perturbed Closed-Loop System with the Minimum Order Controller

6.3.2 Robust Polynomial Controller Design

This section covers the design of three robust fixed term polynomial controllers. The first controller is designed assuming that only the supply pressure changes, the second assuming only the load changes and the third assuming that both change.

Again the design is based on the system with a cascaded digital integrator as shown in (6.19) and (6.20). The corresponding state space model is then

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} = \begin{bmatrix} 0 & -0.0152 \\ 1 & 1.0152 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} 0 \\ 2.5988 \end{bmatrix} u(k) \tag{6.24}$$

$$y(k) = [0 \quad 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k \tag{6.25}$$

The desired dominant closed-loop pole positions are again chosen as $0.65 \pm j0.3$.

The first step is to define the structured model uncertainty for each of the three designs. Examining the models derived in section 6.2, the following can be obtained

	Change in a_1	Change in b_0	Ratio of Change
Max increase in P_s only	0.0178	1.6176	1:90.87
Max increase in P_p only	-0.0914	-0.9085	1:9.94
Max increase in P_s and P_p	0.0246	1.2109	1:49.22

Table 6.3 - Ratio of the Changes in the Open-Loop Polynomial Coefficients

This information can help in the selection of P_t and Q_t, which represent the known structural information regarding the model uncertainty as outlined in chapter four.

However before specifying these, consider the choice of p, the order of the controller. It was found that for values of p beyond 3 the higher order coefficients of the controller polynomials were very small and so could be ignored. Indeed for the case of $p = 3$ the z^{-3} coefficient of $F_p(z^{-1})$ and $G_p(z^{-1})$ is often of the order 1e-10, hence there is little point in considering values of p beyond this case. Of course the idea of increasing p is to introduce more free parameters into the design process, thus it is desirable to use as high a value as possible, hence $p = 3$ is used throughout.

With this value of p it is necessary to specify the desired closed-loop positions of three additional poles. For all of the following they are taken as multiples of -0.0001 so as to have no real influence on the shape of the closed-loop transient behaviour.

Considering the design for the case where only the supply pressure is changing, the model uncertainty is then defined as

$$P_1 = \begin{bmatrix} -1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.26}$$

$$Q_1 = \begin{bmatrix} 90.87 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.27}$$

which clearly shows how to incorporate information about the relative changes in the $A_p(z^{-1})$ polynomial coefficients and the $B_p(z^{-1})$ polynomial coefficients. Note that effectively only one ε is being considered ($u = 1$), which is perfectly acceptable as from table 6.3 it can be seen that in each case the coefficients change in the same direction. As in the previous chapter the form of P_1 is due to the cascaded integrator.

As in the previous chapter the starting points are all randomly chosen as there is no information regarding a suitable starting point. Also all the weights are initially set to 1, again because no additional information is available on a more appropriate choice.

From the randomly chosen starting point

$$\underline{X}^T = [-0.5621 \quad -0.9059 \quad 0.3577 \quad 0.3586 \quad 0.8694 \quad -0.2330 \quad 0.0388$$
$$0.6619 \quad -0.9309 \quad -0.8931 \quad 0.0594 \quad 0.3423 \quad -0.9846]$$

with the weights

$$\underline{W}^T = [1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0]$$

the following result was obtained

$J_{orig} = 6.827726e2$ $\qquad$ $J_{opt\,1} = 1.230418e2$

Eigenvalue sensitivities:

	$0.65 \pm j0.3$	-0.0001	-0.0002	-0.0003
Original	18.4766	0.0234	0.0732	0.0099
Optimal	7.6139	1.2732	1.1924	2.0139

Table 6.4 - Eigenvalue Sensitivities for the Robust P_1 design

Pole Positions:

	Pole Positions	Distance Moved
Max increase in P_s only Controller designed for this case	$0.5504 \pm j0.3593$ 1.2813e-7 -0.0002 -0.0412	0.1159 1.0013e-4 0.0000 0.0409
Max increase in P_p only	$0.7027 \pm j0.2589$ 1.5094e-6 -0.0002 0.1104	0.0668 1.0151e-4 0.0000 0.1107
Max increase in P_s and P_p	$0.5771 \pm j0.3453$ 4.0011e-7 -0.0002 -0.0456	0.0858 1.0040e-4 0.0000 0.0453

Table 6.5 - Pole Positions for the Perturbed Closed-Loop System with the Robust P_s Controller

and the corresponding controller polynomials are

$$F_p(z^{-1}) = 1 - 0.6412z^{-1} - 1.4714e-4z^{-2} + 2.0230e-10z^{-3} \tag{6.28}$$

$$G_p(z^{-1}) = 0.1374 - 0.0594z^{-1} + 0.0038z^{-2} + 8.8239e-7z^{-3} \tag{6.29}$$

$$H_p(z^{-1}) = 0.0818 \tag{6.30}$$

Next consider the design for the case where only the load is changing, the model uncertainty is then defined as

$$P_1 = \begin{bmatrix} -1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.31}$$

$$Q_1 = \begin{bmatrix} 9.94 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.32}$$

Using the randomly chosen starting point

$$\underline{X}^T = [-0.2332 \quad -0.8663 \quad -0.1650 \quad 0.3735 \quad 0.1780 \quad 0.8609 \quad 0.6923$$
$$0.0539 \quad -0.8161 \quad 0.3078 \quad -0.1680 \quad 0.4024 \quad 0.8206]$$

and the weights

$$\underline{W}^T = [1.0 \quad 1.0 \quad 0.5 \quad 0.5 \quad 0.5]$$

the following result was obtained

$J_{orig} = 4.103985$ $\qquad$ $J_{opt\,1} = 1.828735$

Eigenvalue sensitivities:

	$0.65 \pm j0.3$	-0.01	-0.02	-0.03
Original	1.4324	0.0071	0.0111	0.0248
Optimal	0.7359	0.5864	0.5451	0.9220

Table 6.6 - Eigenvalue Sensitivities for the Robust P_p design

Pole Positions:

	Pole Positions	Distance Moved
Max increase in P_s only	$0.6077 \pm j0.4286$ 1.9144e-7 -0.0002 -0.0215	0.1373 1.0019e-4 0.0000 0.0212
Max increase in P_p only Controller designed for this case	$0.6800 \pm j0.2190$ 2.2770e-6 -0.0002 0.0804	0.0863 1.0227e-4 0.0000 0.0807
Max increase in P_s and P_p	$0.6174 \pm j0.3968$ 5.9893e-7 -0.0002 -0.0256	0.1021 1.0059e-4 0.0000 0.0253

Table 6.7 - Pole Positions for Perturbed Closed-Loop System with the Robust P_p Controller

and the corresponding controller polynomials for this case are

$$F_p(z^{-1}) = 1 - 0.4256z^{-1} - 9.7342e-5z^{-2} + 2.0230e-10z^{-3} \tag{6.33}$$

$$G_p(z^{-1}) = 0.0544 - 0.0248z^{-1} + 0.0026z^{-2} + 5.9111e-7z^{-3} \tag{6.34}$$

$$H_p(z^{-1}) = 0.0818 \tag{6.35}$$

Lastly consider the design for the case where both the supply pressure and the load are changing, the model uncertainty is now defined as

$$P_1 = \begin{bmatrix} -1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.36}$$

$$Q_1 = \begin{bmatrix} 49.22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.37}$$

This time starting at the random point

$$\underline{X}^T = [0.0344 \quad 0.1059 \quad 0.0924 \quad -0.2340 \quad 0.8859 \quad -0.7027 \quad -0.1067$$
$$0.0618 \quad -0.0646 \quad -0.3432 \quad -0.4841 \quad -0.2061 \quad 0.1505]$$

with the weights

$$\underline{W}^T = [1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0]$$

the following result was obtained

$J_{orig} = 68.924139$ $\qquad$ $J_{opt\,1} = 34.918173$

Eigenvalue sensitivities:

	$0.65 \pm j0.3$	-0.01	-0.02	-0.03
Original	5.4419	0.0335	1.6903	2.6149
Optimal	3.8851	1.0409	0.9747	1.6423

Table 6.8 - Eigenvalue Sensitivities for the Robust P_s and P_p design

Pole Positions:

	Pole Positions	Distance Moved
Max increase in P_s only	$0.5539 \pm j0.3644$ 1.3088e-7 -0.0002 -0.0396	0.1156 1.0013e-4 0.0000 0.0393
Max increase in P_p only	$0.7012 \pm j0.2566$ 5.9893e-7 -0.0002 -0.0256	0.0671 1.0059e-4 0.0000 0.0253
Max increase in P_s and P_p Controller designed for this case	$0.5796 \pm j0.3489$ 4.1664e-7 -0.0002 -0.0441	0.0857 1.0041e-4 0.0000 0.0438

Table 6.9 - Pole Positions for Perturbed Closed-Loop System with the Robust P_s and P_p Controller

and the corresponding controller polynomials for this case are

$$F_p(z^{-1}) = 1 - 0.6274z^{-1} - 1.4397e-4z^{-2} + 2.0230e-10z^{-3} \tag{6.38}$$

$$G_p(z^{-1}) = 0.1321 - 0.0540z^{-1} + 0.0037z^{-2} + 8.6386e-7z^{-3} \tag{6.39}$$

$$H_p(z^{-1}) = 0.0818 \tag{6.40}$$

6.4 Discussion of the Results and Conclusions

The response of the closed-loop system with the minimum order controller can be seen in figure 6.3. This clearly shows that the transient behaviour is very susceptible to changes in the supply pressure and load.

The response of the closed-loop system with each of the three robust controllers is shown in figures 6.4 to 6.6. As would be expected (because the controller polynomials are very similar) the response for the robust P_s controller is almost identical to the response for the robust P_s and P_p controller. The response for the robust P_p controller is certainly a significant improvement over that for the minimum order controller but not as good as for the other two robust controllers.

This suggests that the best approach to the robust design problem is to assume that the $B_p(z^{-1})$ parameters change significantly more than the $A_p(z^{-1})$ parameters, which was the basis on which the robust P_s and the robust P_s and P_p controllers were designed. Although, as is highlighted by the similarity of the polynomials for the two robust controllers, the actual ratio is not critical, which can be seen by comparing the defined P_t and Q_t in (6.26) and (6.27) with (6.36) and (6.37).

An interesting point is the typical value of the weights selected in this case as compared with the weights for the example in the previous chapter. For that example it was necessary to choose the weights to place a heavy bias on the dominant pole sensitivities. Here, however, the weights did not need to be significantly changed and were similar for all of the closed-loop poles. This suggests that the dominant closed-loop poles for the hydraulic rig are particularly sensitive to model uncertainty.

Examining the distance moved by the poles for each controller (tables 6.3, 6.5, 6.7 and 6.9) shows that there is a high correlation between how far the poles move and the actual transient behaviour of the closed-loop system.

Again the trade-off between the sensitivities of the additional controller poles and the sensitivities of the dominant poles can be clearly seen (tables 6.4, 6.6 and 6.8). This certainly seems to be a characteristic of this type of approach to robust design. However, in each case it can be seen that the additional poles remain close to the origin for the maximum parameter changes and so do not affect the closed-loop response.

The results show that this approach can lead to a significant improvement in performance robustness for practical systems, when compared against the minimum order controller. From the work of Daley (1987) it is also apparent that both the minimum order and the robust controllers are an improvement over the PID controller. Further, the level of improvement is comparable to that obtained via self-tuning control with the added benefit of much reduced on-line computation.

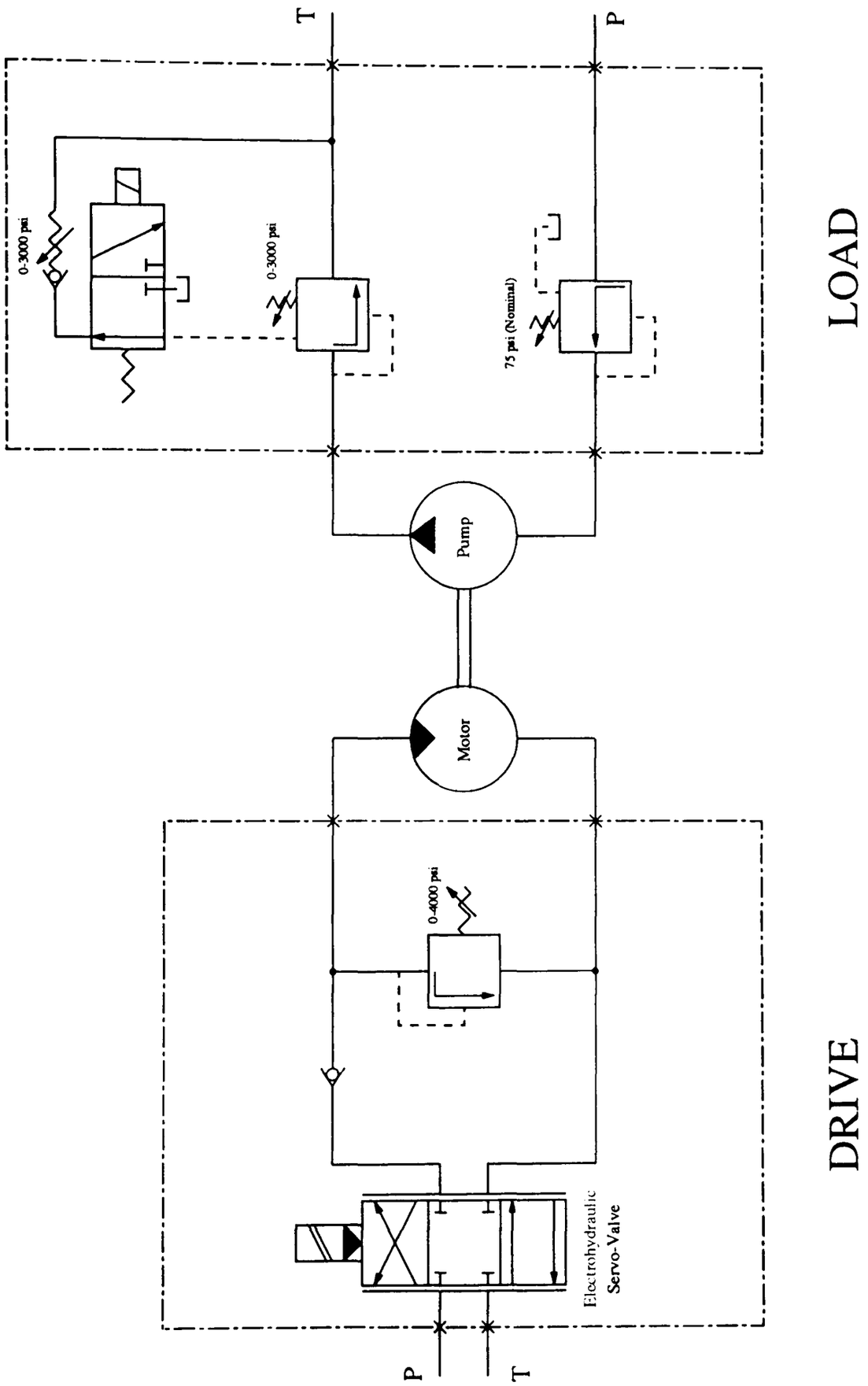


Figure 6.1 - Schematic of the Hydraulic Circuit of the Rig

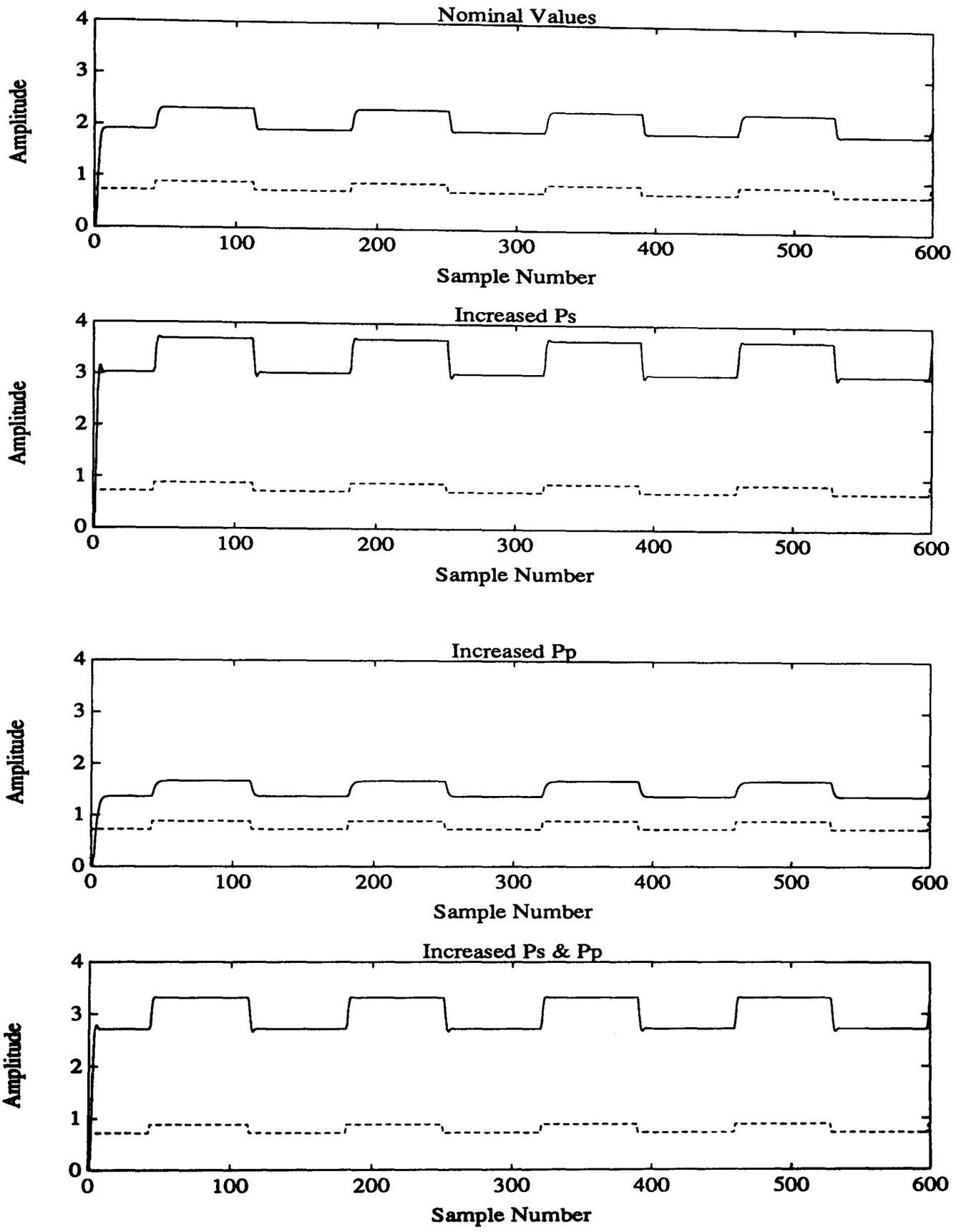


Figure 6.2 - Response of the Open-Loop System

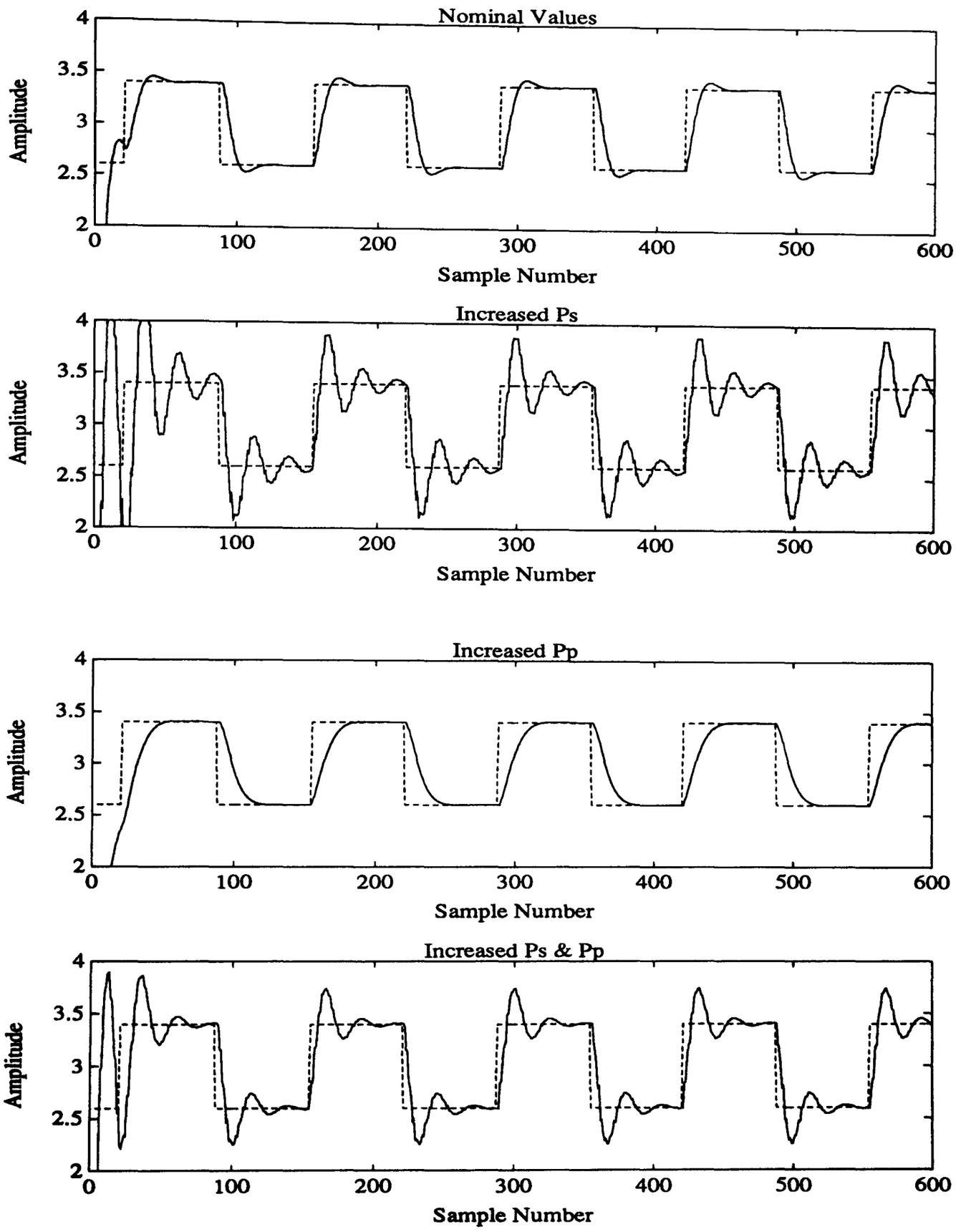


Figure 6.3 - Response of the Closed-Loop System with the Minimum Order Controller

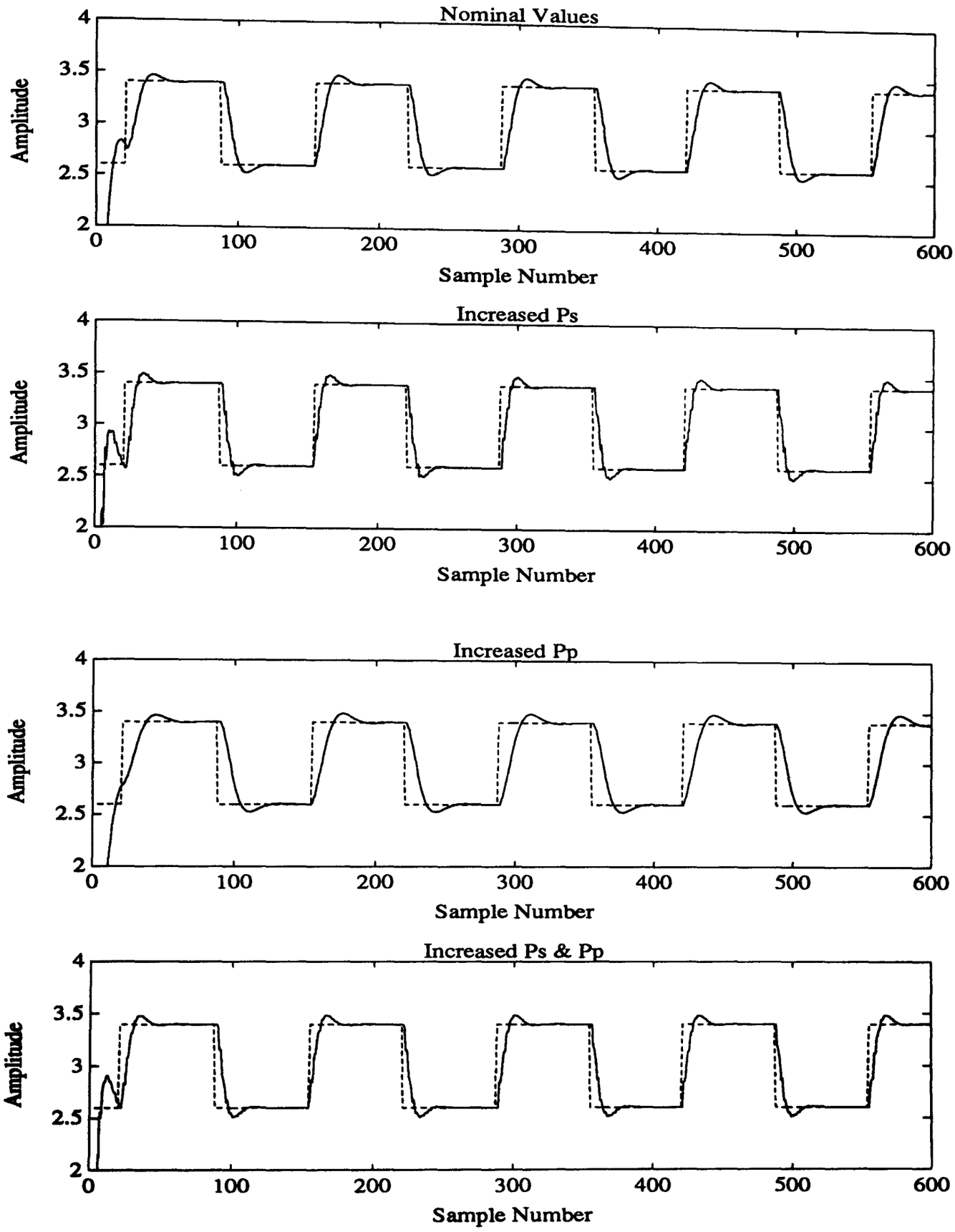


Figure 6.4 - Response of the Closed-Loop System with the Robust P_s Controller

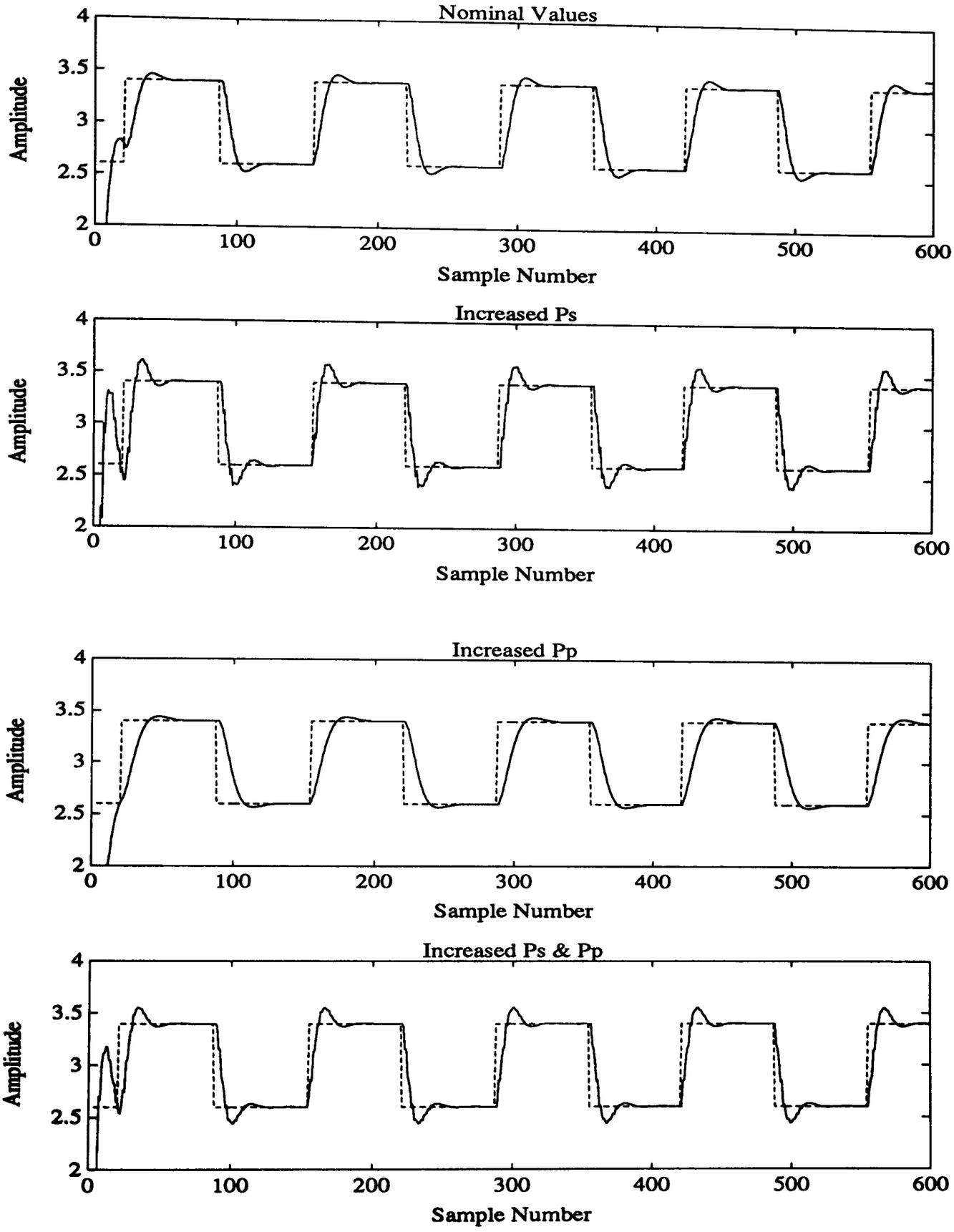


Figure 6.5 - Response of the Closed-Loop System with the Robust P_p Controller

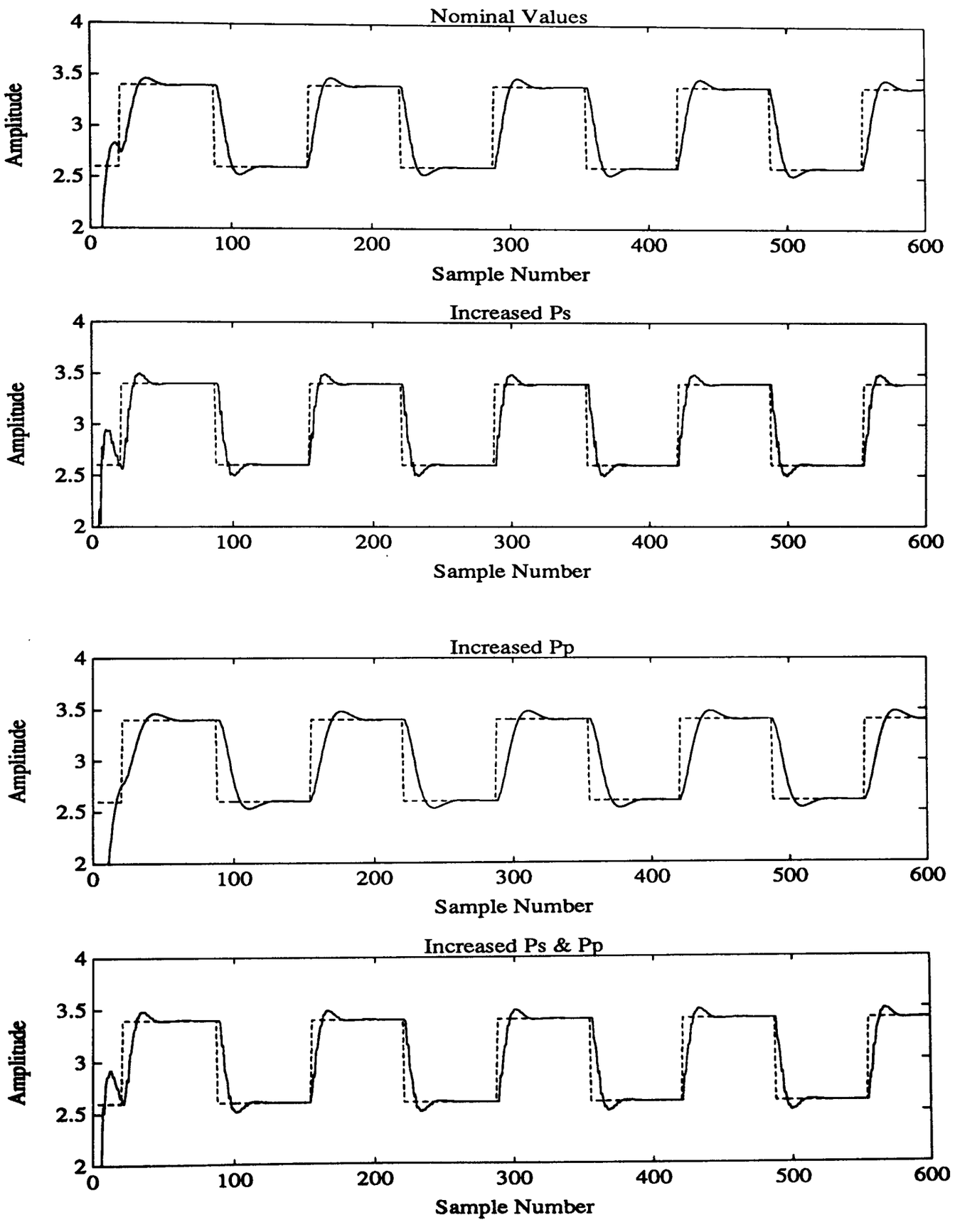


Figure 6.6 - Response of the Closed-Loop System with the Robust P_s and P_p Controller

REFERENCES

ACSL (1987)
'Advanced Continuous Simulation Language Reference Manual'
Mitchell and Gauthier Associates, Concord, Mass, U.S.A.

Daley, S. (1987)
'Application of a Fast Self-Tuning Control Algorithm to a Hydraulic Test Rig'
Proceedings of the Institute of Mechanical Engineers, v201, nC4, pp 285-295

Daley, S. (1990)
'Self-Tuning Control of a Hydraulic Test Rig using a PC'
The GEC Journal of Research, v7, n3, pp 157-167

Dholiwar, D. (1991)
Personal Communication

PRO-MATLAB (1990)
'PRO-MATLAB User Guide'
Mathworks Inc, South Natick, Mass, U.S.A.

Soderstrom, T. and Stoica, P. (1989)
'System Identification'
Prentice Hall, London, U.K.

CHAPTER 7

CONCLUSIONS

7.1 Summary and General Discussion

The problem of designing pole-placement controllers for polynomial systems, such that the closed-loop system exhibits an improved level of performance robustness has been considered. It was assumed that the system was subject to structured model uncertainty where only the coefficients of the open-loop system polynomials were perturbed.

The concept of searching a set of controllers for the most robust one is quite natural in robust design and can be easily cast in the form of an optimisation problem. Robust methods, in general, involve some form of optimisation and with the advent of more powerful computer hardware over recent years, it is only natural to consider whether numerical methods could be usefully employed to solve such problems. The work presented in this thesis was based on this theme.

The design of pole-placement controllers for polynomial systems involves the solution of a polynomial equation, often referred to as a diophantine equation. It has been shown that this equation is very important in the design of such controllers, and a thorough discussion of a number of the major points regarding the equation and finding a solution to it has been presented. The two approaches to solving the equation were reviewed and it can be argued that neither has any real advantage over the other. It was noted, however, that the use of matrix techniques in finding a solution seems to be the most popular approach. This is probably due to a greater general familiarity with matrix theory, the fact that the matrix representation of the equation is of a standard form and also because the matrix approach is easier to implement on a computer.

The conditions for the existence of a solution were established and the violation of these conditions was shown to be dependent on the sample time, hence it was suggested that the sample time be chosen with some care. Many techniques have been proposed for dealing with these violations but it appears that none are totally satisfactory.

From a robustness point of view an interesting property of the equation is the large number of possible solutions, although generally, the minimum order solution is used. In order to assess the merits of other solutions, with regard to performance robustness, a preliminary investigation was carried out. The main conclusion of this work was that in the polynomial framework it is difficult to relate the design of the controller to performance robustness. In the state space framework, however, it is well known that the transient response of a system is dependent on

the eigenvalues and eigenvectors. With all of the above points in mind it was decided to turn to a state space based approach for the problem of designing polynomial controllers with improved performance robustness.

The transformation of a polynomial system to state space form is quite straightforward allowing a state space design to be carried out. It was shown that the resulting output feedback matrix can be easily related to the controller polynomials normally obtained from the solution to the diophantine equation, thus completing the link between the two representations.

For the state space design it was decided to use parametric methods which explicitly represent a set of possible feedback controllers in terms of arbitrary free parameters, allowing the problem to be easily cast in an optimisation framework. Two parametric methods have been considered, one a well established approach (Fahmy and O'Reilly, 1988) and the other a newly proposed scheme (Daley, 1990). Both methods were briefly reviewed and then applied to a number of examples to assess their relative performance. Results showed that the newly proposed scheme did experience a number of difficulties in obtaining a solution for some of the examples considered, which suggested that the well established approach would be better suited in this case. However when applied to a transformed polynomial system the method of Fahmy and O'Reilly (1988) failed to find a solution. Because of this the newly proposed scheme of Daley (1990) was used as the basis of the state space design.

The extra freedom in the design, represented by the free parameters, can be utilised to achieve the goal of improved performance robustness. In order to select appropriate values for the free parameters, it is necessary to define suitable functions relating the sensitivity of the closed-loop system to structured model uncertainty. Having established the importance of the eigenvalues and eigenvectors, their sensitivities were used as the basis of these functions. The conditioning of the matrix of eigenvectors was also used as the basis for defining a suitable function. These functions are often termed cost functions, objective functions or performance indices. The aim is then to select the free parameters such that the cost function being used is at a minimum. This clearly completes the formulation of the robust design as an optimisation problem where numerical techniques can then be used to search for the minimum.

The parametric method of Daley (1990) entails placing certain restrictions on the free parameters which requires performing the optimisation subject to constraints. It was shown however that suitable re-arrangement of the problem allowed the issue of constraints to be avoided altogether.

The overall proposed robust polynomial controller design can then be summarised as

1) Transform the polynomial system to observable canonical state space form.

2) Define the structured model uncertainty.

3) Calculate a robust output feedback matrix which involves

a) Defining a suitable cost function consisting of the eigenvalue/eigenvector sensitivities.

b) Performing a parametric output feedback design.

c) Carrying out a numerical optimisation to determine the set of free parameters that yield a desirable local minimum value for the cost function.

4) Transform the robust output feedback matrix to controller polynomial form.

This approach to the design of robust polynomial controllers was applied to a relatively simple example and a simulation of a hydraulic test rig. The results highlighted a number of interesting points.

When using a cost function consisting of expressions for both the eigenvalue and the eigenvector sensitivities, a conflict appears to arise between the two. The results indicate that a significant increase in design freedom is necessary to yield any sort of desirable result, but of course this will lead to very high order controllers.

The conditioning based cost function did not prove very effective for the design of robust controllers in this case. It is felt that the main reason for this is the general nature of this type of cost function, where effectively the sensitivity of all the eigenvalues are equally important. Results have indicated that to achieve any significant improvement in robustness it is necessary to sacrifice the sensitivity of some eigenvalues in favour of others. Of course the eigenvalues whose sensitivities are allowed to increase should be placed such that their influence on the transient behaviour is minimal, i.e. close to the origin in the z-plane.

The most promising results were obtained using a cost function based solely on the sensitivities of the closed-loop eigenvalues. The results presented here indicate that a significant improvement in performance robustness can be achieved with this type of approach.

Although a direct comparison with other methods was not carried out it is possible to draw some general conclusions. The overall design is centred around the idea of casting the robust design into an optimisation framework and using numerical methods to solve this problem. Utilising the facilities available through a modern workstation it was found that a solution could generally be found in a matter of minutes and indeed, with the rapid advances being made in computing technology it is expected that this time could be significantly reduced in the near future. Such an approach allows quite complex non-linear cost functions to be used which gives greater freedom to define the most appropriate function for the specified design goal. On the basis of this it seems reasonable to suggest that this type of approach will be increasingly used in the future.

Daley (1987) applied self-tuning control to the hydraulic rig used in chapter six. Although a direct comparison cannot be made, it is clear that the performance of the fixed term robust controller compares favourably with that of the self-tuning controller discussed in Daley (1987). Over the range of parameter perturbations for which the robust controller was designed, its advantages are clear. The controller polynomials in terms of the backward shift operator are extremely easy to implement on a digital computer, requiring only a few simple calculations to obtain the control signal. With system identification and controller design being carried out on-line, the self-tuning controller has a heavy computational burden which will place a limit on the maximum sample rate as these calculations need to be carried out in the sample interval. The self-tuning controller has a number of other drawbacks such as tuning transients when abrupt parameter changes occur and problems with the identification algorithm during long periods of poor excitation.

There are a number of problems associated with the design of robust polynomial controllers and the next section discusses these in greater depth, which is then followed by some brief remarks on the most important aspects of the work presented in this thesis.

7.2 Problems and Future Work

In the design of pole-placement polynomial controllers, the diophantine equation plays an important role. It was shown, however, that some difficulties may be experienced in obtaining a solution of this equation. It appears that no complete method for overcoming all of the possible problems has been proposed and probably the best way to avoid any difficulties is to by-pass the solution of the equation altogether. The state space approach, although not specifically aimed at this problem, has the advantageous by-product of not needing to solve the diophantine equation.

The polynomial description also has the disadvantage that it is difficult to relate the design of controllers to the effect on performance robustness. This was one of the main reasons for turning to a state space approach for robust controller design. However, if a satisfactory robustness criteria could be set up in the polynomial framework, then a polynomial based design could be used. Kharitonov's theorem (Siljak, 1989) may prove useful for such a purpose as it basically relates changes in a polynomial's coefficients to changes in its roots and if applied to the characteristic polynomial may help to establish a suitable measure.

The state space design also has a number of problems. For the parametric output feedback method of Daley (1990) the singularity of V_{11} is a significant problem which needs to be addressed. V_{11} is dependent on a set of free parameter vectors which are effectively selected from a vector space. It has been suggested that a simple way forward would be to define a second vector space for the free parameter vectors such that V_{11} is always non-singular. The intersection of the two

spaces would then provide an appropriate space for the selection of the free parameter vectors. So far, however, it has not been possible to define this second vector space and so the problem remains unsolved.

The method of Fahmy and O'Reilly (1988) experienced even more severe problems in that no solution could be found at all. The reason appears to be related to the structure of the open-loop system matrices and it is suggested that overprotection of the right eigenvectors could be the cause of the problem. This matter certainly needs to be investigated further hopefully leading to a proposal which will allow the method to be used.

It is worth mentioning at this stage that the problem of output feedback itself remains an unresolved one as no method at present can guarantee that a controller can be obtained for all systems. As such the problems mentioned above may not necessarily be solvable and so it may be useful to conduct a wider search and assessment of possible state space methods. Section 3.5 in chapter 3 discusses some eigenstructure techniques which could possibly be used but unfortunately came to light too late to be incorporated in this work.

The design centres around the specification of suitable cost functions. The derivation presented in chapter four was based entirely on differential calculus. Such an approach is really only valid for small variations but does provide an approximation for larger changes. Results presented in this thesis would indicate that such an approximation is satisfactory as significant improvements in performance robustness can be achieved. The main problem, however, is that there is no information on how good this approximation is and as such it would be desirable to formulate more appropriate cost functions for large changes in the model parameters.

The minimisation of these functions also has some limitations which it is desirable to overcome. Numerical methods will only find a local minimum, which may prove to be the global minimum, but this cannot be guaranteed. Although this does not represent a significant drawback as a local minimum may yield the level of improvement sought, it would be desirable to obtain the global minimum as then it is known that no better solution exists, for the particular cost function being used. Such a task represents a major undertaking and would certainly involve a radically different algorithm. One approach which seems interesting is that of genetic optimisation, Goldberg (1989). Here the basic rules of genetics, reproduction, crossover and mutation are implemented on a computer and applied to a large population of starting points. The aim is to emulate the procedure of natural selection to find the best solution. Fundamentally the algorithm is working on patterns in the data and not based on gradient information as with many of the traditional schemes. Although this approach can still not guarantee finding the global minimum, due to the wide spread of its search the algorithm is likely to find a very good solution which could easily be missed by more conventional algorithms. One possible problem could be the actual implementation of the algorithm and it is also likely to take considerably longer than conventional algorithms to perform the search.

7.3 Concluding Remarks

The work presented in this thesis was motivated by the desire to establish an alternative approach to the design of robust polynomial controllers. In the pursuit of this goal a number of contributions have been made.

- The diophantine equation is extremely important in the design of pole-placement controllers for polynomial systems. A thorough discussion of many aspects of this equation was presented.

- A basic investigation into selecting a more robust solution to the diophantine equation has been conducted. The conclusion however was that in the polynomial framework it is difficult to relate decisions in the design procedure to the effect on performance robustness.

- An alternative robust design procedure was presented. It utilises state space techniques by transforming the system to state space form, performing the design and transforming the resulting controller back to polynomial form. Results have shown that a significant improvement in performance robustness can be achieved with such an approach.

- Two state space parametric output feedback methods were reviewed. One is a well established approach and the other a newly proposed scheme. It was shown that although the well established method performs better on general state space systems, it fails when applied to transformed polynomial systems.

- The design of the robust controller is based on the sensitivities of the eigenvalues and eigenvectors. Results have shown that there appears to be a conflict when attempting to minimise the sensitivities of both. It was suggested, however, that increasing the design freedom could yield more desirable results.

- The conditioning of the matrix of right eigenvectors was also used as the basis on which to design a robust controller. Again results have indicated that only a slight improvement in performance robustness can be achieved with such a measure.

- For this type of approach to robust polynomial controller design it was found that using only the sensitivities of the eigenvalues lead to a significant improvement in performance robustness. It was noted, however, that decreasing the sensitivities of some eigenvalues tends to lead to an increase in the sensitivities of others.

REFERENCES

Daley, S. (1990)
'On Eigenstructure Assignability using parametric output feedback'
Brunel University Control Engineering Centre Internal Report, August 1990

Fahmy, M.M. and O'Reilly, J. (1988)
'Multistage Parametric Eigenstructure Assignment by Output Feedback Control'
International Journal of Control, v48, n1, pp 97-116

Goldberg, D.E. (1989)
'Genetic Algorithms in Search, Optimisation and Machine Learning'
Addison-Wesley, Wokingham, U.K.

Siljak, D.D. (1989)
'Parameter Space Methods for Robust Control Design: A Guided Tour'
IEEE Transactions on Automatic Control, v34, n7, pp 674-688

White, B.A. (1991)
'Eigenstructure Assignment by Output Feedback'
International Journal of Control, v53, n6, pp 1413-1429

APPENDIX A

ALGORITHMS FOR THE POLYNOMIAL SOLUTION OF THE DIOPHANTINE EQUATION

A.1 Introduction

For the polynomial solution of the diophantine equation two algorithms are used: the extended Euclidean algorithm and the division of polynomials algorithm. The version of the algorithms shown here follows those outlined in Kucera (1979).

A.2 Division of Polynomials Algorithm

Definition: Given two polynomials A_p and B_p with $B_p \neq 0$, this algorithm returns two polynomials U_p and V_p such that

$$A_p = B_p U_p + V_p \tag{A.1}$$

where U_p is the quotient and V_p the remainder.

Algorithm:

1) Set $U_p = 0, V_p = A_p$

2) If $\deg(V_p) < \deg(B_p)$, stop

3) $$\lambda = \frac{\text{leading coefficient of } V_p}{\text{leading coefficient of } B_p}$$

$$n = \deg(V_p) - \deg(B_p)$$

4) $V_p = V_p - \lambda(z^{-1})^n B_p$

5) $U_p = U_p + \lambda(z^{-1})^n$

6) Goto 2

A.3 Extended Euclidean Algorithm

Definition: Given two polynomials A_p and B_p with $A_p, B_p \neq 0$, this algorithm returns five polynomials g_p, P_p, Q_p, R_p and S_p which satisfy

$$A_p P_p + B_p Q_p = g_p \tag{A.2}$$

$$A_p R_p + B_p S_p = 0 \tag{A.3}$$

g_p is the greatest common divisor (GCD) of A_p and B_p. P_p, Q_p and R_p, S_p are pairs of coprime polynomials.

Algorithm:

1) Set $V = I_2$, $F = [A_p B_p]$

2) If only one non-zero polynomial in F goto 6) else say

X_p = the lower degree polynomial in F
Y_p = the other polynomial in F

Noting which columns in F they correspond to

3) $$\lambda = \frac{\text{leading coefficient of } Y_p}{\text{leading coefficient of } X_p}$$

$$n = \deg(Y_p) - \deg(X_p)$$

4) $Y_p = Y_p - \lambda(z^{-1})^n X_p$. Perform the same operations on the corresponding column of V

5) $F = [X_p Y_p]$ or $F = [Y_p X_p]$ depending on which columns X_p and Y_p correspond to. Goto 2

6) If the non-zero polynomial appears in the second column of F, interchange the columns of both F and V. Stop

Then

$$F = [g_p 0] \quad \text{and} \quad V = \begin{bmatrix} P_p & R_p \\ Q_p & S_p \end{bmatrix}$$

REFERENCES

Kucera, V. (1979)
'Discrete Linear Control: The Polynomial Equation Approach'
Wiley, Chichester, U.K.

APPENDIX B

PROGRAMS FOR THE ROBUST POLYNOMIAL CONTROLLER DESIGN

B.1 Pre-Optimisation Programs - PRO-MATLAB

All the programs presented here are for transforming the polynomial system, specified in the function MODEL, to state space form and calculating all preliminary data as outlined in chapter 5. The data is saved in a file which is accessed by the FORTRAN 77 optimisation programs.

An example model definition is

```
function [a,aorig,int,b,c,n,m,r,q,eigen,cgate] = model(p)

% define system polynomials
int = [1 -1];
aorig = [1 -0.6];
a = multpoly(int,aorig);
b = [0 1 1.5];
c = [1 -0.4];

% hence define system parameters
n = length(a) - 1;
m = 1;
r = 1;

% specify desired pole positions
j = sqrt(-1);
eigen = [0.75+0.2*j 0.75-0.2*j];

% cgate indicates which poles are complex conjugates (1)
% and which are not (0)
cgate = [0 1];

% increase parameters of system due to p
n = n+p;
m = m+p;
r = r+p;
q = n-r;

% increase the number of eigenvalues
for i = 1:p
 eigen = [eigen -0.01*i];
 cgate = [cgate 0];
end
```

The main pre-optimisation program is

```
% preopt.m

% This m-file converts polynomial systems to state space form
% and saves the data required by the fortran optimisation programs

clear

% set up parameters
sm = 1e-10;

% set up model
cf = input('Enter number of cost function to be used : ','s');
p = input('Enter value of p : ');
[a,aorig,int,b,c,n,m,r,q,eigen,cgate] = model(p);

% transform to state space form
[A,B,C] = trans_ss(a,b);

% add p states
[A,B,C] = add_state(A,B,C,p,n,m,r);

% switch states so that C = [I 0]
[A,B,C] = switch(A,B,C,p,n);

% perform some simple checks and determine the
% number of complex poles
nc = precheck(C,eigen,n,r,sm);

% Set up the alphas in zeta
for k = 1:q
 temp1 = 1;
 temp2 = temp1;
 for i = 2:r
  if cgate(i) == 0
   temp2 = 1;
  end
  temp1 = [temp1 temp2];
 end
 if k == 1
  alpha = temp1;
 else
  alpha = [alpha; temp1];
 end
end

% Calculate the vector space in which gamma must lie
S = calcspce(A,B,alpha,eigen,cgate,nc,n,m,r,q);

% Calculate LA and LAB
LA = inv(eigen(1)*eye(n)-A);
LAB = inv(eigen(1)*eye(n)-A)*B;
for i = 2:n
 LA = [LA; inv(eigen(i)*eye(n)-A)];
 LAB = [LAB; inv(eigen(i)*eye(n)-A)*B];
end

% Save data
path = ['/user/eepg/eepgkdw/ws/'];
file = ['matdata' num2str(p) cf];
eval(['save ' path file ' nc n m r q cgate S LA LAB'])
```

The associated functions are

```
function [A,B,C] = trans_ss(a,b);

% function to transform the system polynomials to state space
% observable canonical form - Ogata, discrete time control, p491

la = length(a);
lb = length(b);
% if b is lower order than a, then pad b with zeros
for i = 1:lb
  tempb(i) = b(i);
end
if lb < la
 for i = lb+1:la
   tempb(i) = 0;
 end
 lb = la;
end

% THE A MATRIX
% remove leading 1 in a polynomial
tempa = a(2:la);
la = la - 1;

% Create A without last column of a poly coefficients
A = eye(la-1);
z = zeros(1,la-1);
A = [z; A];

% switch order and negate a poly coefficients
for i = 1:la
  temp(i) = -tempa(la+1-i);
end

% add a poly coeffs to A matrix
A = [A temp'];

% THE B MATRIX
% calc B elements - don't use b0 so lb = lb-1
lb = lb - 1;
B = zeros(lb,1);
for i = 1:lb
  B(i) = tempb(lb+2-i) - a(lb+2-i)*tempb(1);
end

% THE C MATRIX
C = zeros(1,la-1);
C = [C 1];
```

```
function [A1,B1,C1] = add_state(A,B,C,p,n,m,r)

% function to augment the system with p extra states
% note n,m,r have all had p added for extra states

epsilon = 0;

% THE A MATRIX
A1 = [A zeros(n-p,p)];
A1 = [A1; zeros(p,n-p) epsilon*eye(p)];

% THE B MATRIX
B1 = [B zeros(n-p,p)];
B1 = [B1; zeros(p,m-p) eye(p)-epsilon*eye(p)];

% THE C MATRIX
C1 = [C zeros(r-p,p)];
C1 = [C1; zeros(p,n-p) eye(p)];
```

```
function [A1,B1,C1] = switch(A,B,C,p,n);

% function to switch the states such that C is of the form [I 0]
% assumes that C is in the observable canonical form, i.e [0 I]

% THE A MATRIX
% switch rows and then columns
% rows
A1 = [A(n-p:n,:); A(1:n-p-1,:)];
% columns
A1 = [A1(:,n-p:n) A1(:,1:n-p-1)];

% THE B MATRIX
% switch rows only
B1 = [B(n-p:n,:); B(1:n-p-1,:)];

% THE C MATRIX
% switch columns only
C1 = [C(:,n-p:n) C(:,1:n-p-1)];
```

```
function numcom = precheck(C,eigen,n,r,small)

% function to perform some simple checks and to
% determine the number of complex poles

% check that the correct number of eigenvalues have been specified
if n ~= length(eigen)
 error('Incorrect number of specified eigenvalues')
end

% check C is of the correct form
chk1 = [eye(r) zeros(r,n-r)];
chk2 = C - chk1;
if sum(sum(chk2 > small)) > 0
 error('C is not of the correct form');
end

% determine number of complex eigenvalues
numcom = sum(imag(eigen) ~= 0);

% check if all eigenvalues are complex
if numcom == n
 error('All poles are complex')
end
```

```
function [S] = calcspce(A,B,alpha,eigen,cgate,numcom,n,m,r,q);

% function to calculate zeta for real and complex poles
% will not work for all complex case

% CALCULATE ZETA
count = 1;
for i = 1:q
 for k = 1:r
  t1 = (inv(eigen(i+r)*eye(n)-A)-inv(eigen(k)*eye(n)-A))*B;
  t1 = alpha(i,k)*t1;
  if k == 1
   t2 = t1(1:r,:);
  else
   t2 = [t2 t1(1:r,:)];
  end
  clear t1
 end
 if i == 1
  zeta = t2;
 else
  zeta = [zeta; t2];
 end
 clear t2
end

if numcom > 0

 % SOME POLES ARE COMPLEX

 % calculate zetabr by removing complex conjugate rows and cols
 % determine which rows are complex conjugates - dependant on
 % the eigenvalues associated with P2 (i.e r+1 - n)
 zetaa = zeta(1:r,:);
 for i = 2:q
  if cgate(i+r) == 0
   % eigenvalue is not a complex conjugate so use row block
   zetaa = [zetaa; zeta(1+(i-1)*r:1+(i-1)*r+(r-1),:)];
  end
 end

 % determine which cols are complex conjugates - dependant on
 % the eigenvalues associated with P1 (i.e 1 - r)
 zetab = zetaa(:,1:m);
 for i = 2:r
  if cgate(i) == 0
   % eigenvalue is not a complex conjugate so use col block
   zetab = [zetab zetaa(:,1+(i-1)*m:1+(i-1)*m+(m-1))];
  end
 end
```

```
% calculate zetabr, the real part of zetab
zetabr = real(zetab);

% ZETABI AND ZETAGI HAVE THE SAME NUMBER OF ROWS AS ZETAA
% zetabi
for i = 1:q
 if (cgate(i+r) == 0)
  temp1 = inv(eigen(1)*eye(n) - A)*B;
  temp1 = alpha(i,1)*temp1(1:r,:);
  for k = 2:r
   if (cgate(k) == 0) & (imag(eigen(k)) > 1e-12)
    temp2 = inv(eigen(k)*eye(n) - A)*B;
    temp1 = [temp1 alpha(i,k)*temp2(1:r,:)];
   end
  end
  if sum(size(zetabi)) == 0
   zetabi = imag(temp1);
  else
   zetabi = [zetabi; imag(temp1)];
  end
 end
end

% zetagi - first column
temp1 = inv(eigen(1+r)*eye(n) - A)*B;
temp1 = temp1(1:r,:);
for i = 2:q
 if (cgate(i+r) == 0)
  temp2 = inv(eigen(i+r)*eye(n) - A)*B;
  temp1 = [temp1; temp2(1:r,:)];
 end
end
temp1 = imag(temp1);
zetagi = temp1;
for i = 2:r
 if (cgate(i) == 0)
  zetagi = [zetagi temp1];
 end
end

% hence zeta
[t1,t2] = size(zetagi);
[t3,t4] = size(zetabr);
[t5,t6] = size(zetabi);
zeta = [zetabi; zeros(t3-t5,t6)];
zeta = [zetabr zeta];
zeta = [zeta; zetagi zeros(t1,t6+t4-t2)];

% delete zero rows and columns from zeta

% rows first
firstzerorow = 0;
i = 0;
while (firstzerorow ==0) & (i < t1+t3)
 i = i+1;
 if any(zeta(i:i,:)) == 0
  % zero row
  firstzerorow = i;
 end
end
if firstzerorow > 0
 % check remaining rows are all zero
 for i = firstzerorow+1:t1+t3
  if any(zeta(i:i,:)) == 1
   % non zero row
   error('mixed zero rows in zeta')
  end
 end
 % delete rows from zeta
 zeta = zeta(1:firstzerorow-1,:);
end



% now columns
firstzerocol = 0;
i = 0;
while (firstzerocol ==0) & (i < t4+t6)
 i = i+1;
 if any(zeta(:,i:i)) == 0
  % zero col
  firstzerocol = i;
 end
end
if firstzerocol > 0
 % check remaining cols are all zero
 for i = firstzerocol+1:t4+t6
  if any(zeta(:,i:i)) == 1
   % non zero col
   error('mixed zero cols in zeta')
  end
 end
 % delete cols from zeta
 zeta = zeta(:,1:firstzerocol-1);
end



end



% CALCULATE THE NULL SPACE OF ZETA
S = null(zeta);
[t1,t2] = size(S);
if t2 < r*(m-q)
keyboard
 error('The null space of zeta is the wrong dimension')
end
```

B.2 Optimisation Programs - FORTRAN 77

All the routines associated with the numerical optimisation itself are written in FORTRAN 77 to facilitate the use of the Numerical Algorithms Group (NAG) library routines. Because the parametric state space design has to be performed during the calculation of the cost function a number of additional programs have had to be written, also outlined in chapter 5.

The main optimisation program and its associated routines are in the file JAF.F

```
C  JAF.F
C  Define parameters
C  {NN - no of variables = r*(m-q)+q}
   integer PNN
   character*1 P
   parameter (PNN = 13, P = '3')
C  Define parameters
   integer maxn,maxm,maxr,maxq,maxsr,maxsc,num
C  {maxn,maxm,maxr,maxq - maximum values of parameters of system}
C  {maxsr,maxsc - maximum dimension of S, r(m-q), r(r-q)}
C  {num - number of errors being considered}
   parameter (maxn = 10, maxm = 9, maxr = 9, maxq = 3, num = 1)
   parameter (maxsr = 81, maxsc = 81)
C  Common scalars
   integer nc,n,m,r,q,sr,sc,count,store
   double precision aFC
   character*1 N1
   character*30 FNAME4
C  Common arrays
   integer CGATE(maxn)
   double precision S(maxsr,maxsc),WEIGHT(3*maxn)
   double complex P(maxm,maxn),V(maxn,maxn),W(maxn,maxn),
  *          LA(maxn,maxn,maxn),LAB(maxn,maxn,maxm),
  *          KC(maxm,maxn),dA(num,maxn,maxn),
  *          dB(num,maxn,maxm)
C  Common blocks
   common /CONST1/ count,store,nc,n,m,r,q,sr,sc,N1,FNAME4
   common /CONST2/ CGATE,dA,dB,WEIGHT,LA,LAB,S,KC
   common /RESULT/ F,V,W,aFC
C  Local scalars
   integer I,J,NN,LIW,LRW,IBOUND,IFAIL
   double precision FC,FCORIG
   logical OK
   character*1 ANS,FNUM
   character*30 FNAME1,FNAME2,FNAME3
C  Local arrays, {IW(>NN+2),X(NN),RW(>12*NN+NN(NN-1)/2)}
   integer IW(100)
   double precision X(PNN),XORIG(PNN),RW(5000),BL(PNN),BU(PNN)
   double complex VORIG(maxn,maxn),KCORIG(maxm,maxn),
  *          FORIG(maxm,maxn)
C  Declare external subprograms
   external FUNCT1
C  Select cost function
   print*, ' '
   print*, 'Enter cost function number (in quotes)'
   read*, N1
C  Read the initial parameters from file (supplied by Matlab)
   FNAME1 = 'matdata11.mat'
   FNAME1(8:8) = P
   FNAME1(9:9) = N1
   call RDATA(FNAME1,S,sr,sc,LA,LAB,CGATE,nc,n,m,r,q)
C  Check NN is set to the correct value
   NN = PNN
   if (NN.ne.(r*(m-q)+q)) then
    print*, 'PNN should be set to ',(r*(m-q)+q)
    stop
   end if
C  Load start data
   FNAME1 = 'cost1/startdata11.mat'
   FNAME1(5:5) = N1
   FNAME1(16:16) = P
   FNAME1(17:17) = N1
   call RSTART(FNAME1,X,dA,dB,WEIGHT,NN)
C  Set up parameters - scalars
   OK = .TRUE.
   IBOUND = 1
   count = 0
   store = 99
   FNAME1 = 'cost1/JAF1orig11.mat'
   FNAME2 = 'cost1/JAF1opt_11.mat'
   FNAME3 = 'JAFtmp_orig.mat'
   FNAME4 = 'JAFtmp_opt.mat'
   FNAME1(5:5) = N1
   FNAME2(5:5) = N1
   FNAME1(15:15) = P
   FNAME2(15:15) = P
   FNAME1(16:16) = N1
   FNAME2(16:16) = N1
C  Set up parameters - array dimensions
   LIW = 100
   LRW = 5000
C  Store free parameters
   do I = 1,NN
    XORIG(I) = X(I)
   end do
C  Calculate original value of costfunction
   print*, ' '
   print*, 'Original values'
   call FUNCT1(NN,XORIG,FCORIG)
C  Store original values of V and F
   do I = 1,n
    do J = 1,n
     VORIG(I,J) = V(I,J)
    end do
   end do
   do I = 1,m
    do J = 1,n
     FORIG(I,J) = F(I,J)
    end do
   end do
   call CALCKC(n,m,F,W,KCORIG)
C  Store original data in temporary file in case program aborted
   call WORIG(NN,m,n,FCORIG,XORIG,FORIG,VORIG,KCORIG,FNAME3)
   print*, ' '
   print*, ' '
   print*, 'Finished set up'
   print*, ' '
```

```
      do while(OK)
       IFAIL = 1
       call E04JAF(NN,IBOUND,BL,BU,X,FC,IW,LIW,RW,LRW,IFAIL)
       print*, ' '
       print*, 'Original Value       : ',FCORIG
       print*, 'Weighted Optimal Value : ',FC
       print*, 'Actual Optimal Value  : ',aFC
       print*, ' '
       count = 0
       store = 99
       print*, 'Final values'
       call FUNCT1(NN,X,FC)
       OK = .FALSE.
       if (IFAIL.eq.0) then
        print*, 'E04JAF has found a minimum point.'
       else if (IFAIL.eq.1) then
        print*, 'Parameter out of range.'
       else if (IFAIL.eq.2) then
        print*, '400*NN function evaluations'
        print*, ' '
        print*, 'Restart with old X (y/n) ?'
        read*, ANS
        if ((ANS.eq.'y').or.(ANS.eq.'Y')) then
         OK = .TRUE.
        end if
       else if (IFAIL.eq.3) then
        print*, 'The conditions for a minimum have not all been'
        print*, 'satisfied, but a lower point could not be found.'
       else if (IFAIL.eq.4) then
        print*, 'Overflow has occured'
        print*, ' '
        print*, 'Restart with old X (y/n) ?'
        read*, ANS
        if ((ANS.eq.'y').or.(ANS.eq.'Y')) then
         OK = .TRUE.
        end if
       end if
       print*, ' '
       print*, 'Save current parameter values (y/n) ?'
       read*, ANS
       if ((ANS.eq.'y').or.(ANS.eq.'Y')) then
        print*, 'Enter file number (in quotes)'
        read*, FNUM
        FNAME1(10:10) = FNUM
        FNAME2(10:10) = FNUM
        call WORIG(NN,m,n,FCORIG,XORIG,FORIG,VORIG,KCORIG,FNAME1)
        call WOPT(NN,m,n,FC,X,dA,dB,WEIGHT,F,V,KC,FNAME2)
       end if
       print*, ' '
      end do

      stop
      end

      ------------------------

      subroutine FUNCT1(NN,XC,FC)

C     Subroutine to calculate the value of the objective function
C     residuals

C     Define parameters
      integer maxn,maxm,maxr,maxq,maxsr,maxsc,maxcount,num
C     (maxn,maxm,maxr,maxq - maximum values of parameters of system)
C     (num - number of errors being considered)
C     (maxsr,maxsc - maximum dimension of S, r(m-q), r(r-q))
      parameter (maxn = 10, maxm = 9, maxr = 9, maxq = 3, num = 1)
      parameter (maxsr = 81, maxsc = 81)
      parameter (maxcount = 20)

C     Common scalars
      integer nc,n,m,r,q,sr,sc,count,store
      double precision aFC
      character*1 N1
      character*30 FNAME4

C     Common arrays
      integer CGATE(maxn)
      double precision S(maxsr,maxsc),WEIGHT(3*maxn)
      double complex F(maxm,maxn),V(maxn,maxn),W(maxn,maxn),
     *          LA(maxn,maxn,maxn),LAB(maxn,maxn,maxm),
     *          KC(maxm,maxn),dA(num,maxn,maxn),
     *          dB(num,maxn,maxm)

C     Common blocks
      common /CONST1/ count,store,nc,n,m,r,q,sr,sc,N1,FNAME4
      common /CONST2/ CGATE,dA,dB,WEIGHT,LA,LAB,S,KC
      common /RESULT/ F,V,W,aFC

C     Scalar arguments
      integer NN
      double precision FC

C     Array arguments
      double precision XC(NN)

C     Local scalars
      integer I,T

C     Local arrays
      double precision VAL(maxn),VEC(maxn),VECSEN(num,maxn)
      double complex VALSEN(num,maxn)

C     Declare external subprograms
      external DALEY,COST1,COST2,COST3,COST4

C     Calculate output feedback using appropriate method
      call DALEY(NN,XC,S,sr,sc,LAB,CGATE,nc,n,m,r,q,F,V,W)

C     Calculate K
      call CALCKC(n,m,F,W,KC)

C     Calculate cost function
      if (N1.eq.'1') then
C      Eigenvalue differential cost function
       call COST1(n,m,V,W,KC,dA,dB,WEIGHT,FC,aFC,VALSEN)
      else if (N1.eq.'2') then
C      Eigenstructure differential cost function
       call COST2(n,m,F,V,W,KC,LA,LAB,dA,dB,WEIGHT,FC,aFC,
     *          VALSEN,VECSEN)
      else if (N1.eq.'3') then
C      Transient response differential cost function
       call COST3(n,m,F,V,W,KC,LA,LAB,dA,dB,WEIGHT,FC,aFC,
     *          VALSEN,VECSEN)
      else if (N1.eq.'4') then
C      Conditioning cost function
       call COST4(n,V,W,WEIGHT,FC,aFC)
      else
       print*, 'No other cost functions yet'
       stop
      end if

C     Print FC after set number of iterations
      count = count + 1
      if (count.ge.maxcount) then
       count = 0
       print*, FC, '  ', aFC
      end if

C     Store files after 100 iterations
      store = store + 1
      if (store.ge.100) then
       store = 0
       call WOPT(NN,m,n,FC,XC,dA,dB,WEIGHT,F,V,KC,FNAME4)
       print*, ' '
       print*, 'Saved data in temp file'
       print*, ' '
       print*, FC, '  ', aFC
       print*, ' '
       if (N1.ne.'4') then
        print*, 'Eigenvalue Sensitivities'
        do I = 1,n
         VAL(I) = 0.0
         do T = 1,num
          VAL(I) = VAL(I) +
     *          sqrt(real(VALSEN(1,I)*conjg(VALSEN(1,I))))
         end do
        end do
        print 10,(VAL(I),I = 1,n)
        print 10,(WEIGHT(I)*VAL(I),I = 1,n)
```

```
      if ((N1.eq.'2').or.(N1.eq.'3')) then
       print*, 'Eigenvector Sensitivities'
       do I = 1,n
        VEC(I) = 0.0
        do T = 1,num
         VEC(I) = VEC(I) + VECSEN(T,I)
        end do
       end do
       print 10, (VEC(I),I = 1,n)
       print 10, (WEIGHT(1+2*n)*WEIGHT(I+n)*VEC(I),I = 1,n)
      end if
10     format(10G12.4)
      print*,' '
     end if
    end if
    return
    end
```

```
    subroutine CALCKC(n,m,F,W,KC)
C   Subroutine to calculate the o/p feedback matrix
C   Define parameters
    integer maxn,maxm
    parameter (maxn = 10, maxm = 9)
C   Scalar arguments
    integer n,m
C   Array arguments
    double complex F(maxm,maxn),W(maxn,maxn),KC(maxm,maxn)
C   Local scalars
    integer I,J,K
C   Calculate KC, mult F and W
    do I = 1,m
     do J = 1,n
      KC(I,J) = dcmplx(0.0,0.0)
      do K = 1,n
       KC(I,J) = KC(I,J) + F(I,K)*W(K,J)
      end do
     end do
    end do
    return
    end
```

The parametric state space design is performed by the routines in the file DALEY.F. This includes the routines for the calculation of an accurate inverse and the null space of a matrix via the singular value decomposition (SVD)

```
    subroutine DALEY(NN,XC,S,sr,sc,LAB,CGATE,nc,n,m,r,q,F,V,W)
C   Daley's method of o/p feedback design
C   Define parameters
    integer maxn,maxm,maxsr,maxsc
    double precision zero
C   (maxn,maxm - maximum values of parameters of system)
C   (maxsr,maxsc - maximum dimension of S, r(m-q), r(r-q))
    parameter (maxn = 10, maxm = 9)
    parameter (maxsr = 81, maxsc = 81)
    parameter (zero = 0.0)
C   Scalar arguments
    integer NN,sr,sc,n,m,r,q,nc
C   Array arguments
    integer CGATE(maxn)
    double precision XC(NN),S(maxsr,maxsc)
    double complex LAB(maxn,maxn,maxm),
   *          F(maxm,maxn),V(maxn,maxn),W(maxn,maxn)
C   Local scalars
    integer I,J,K,DIM,RANK
C   Local arrays
    double precision GAMMA(maxsr),SPCE(maxm,maxm),
   *          RZ(maxm,maxm)
C   Calculate gamma from the null space of zeta
    do I = 1,sr
     GAMMA(I) = 0.0
     do J = 1,sc
      GAMMA(I) = GAMMA(I) + XC(J)*S(I,J)
     end do
    end do
C   Extract F1 from gamma
    call CALCF1(GAMMA,CGATE,nc,n,m,r,q,F)
C   Calculate the first r vectors of V
    do I = 1,r
     do J = 1,n
      V(J,I) = LAB(I,J,1)*F(1,I)
      do K = 2,m
       V(J,I) = V(J,I) + LAB(I,J,K)*F(K,I)
      end do
     end do
    end do
C   Calculate the remaining F vectors
C   Assumes that all poles associated with F2 are real
    do I = 1,q
C    Calculate Zi as a real matrix
     call CALCRZ(LAB,F,V,I+r,m,r,RZ)
C    Find the null space of RT
     call CALCSVD(RZ,maxm,maxm,m,m,DIM,RANK,SPCE)
     if (DIM.lt.1) then
      print*, 'Error in DALEY'
      print*, 'No null space exists for z'
      stop
     end if
C    update F - assumes all poles associated with F2 are real
     do J = 1,m
      F(J,r+I) = dcmplx(XC(r*(m-q)+I)*SPCE(J,1),zero)
     end do
    end do
C   Calculate the remaining V vectors
    do I = r+1,n
     do J = 1,n
      V(J,I) = LAB(I,J,1)*F(1,I)
      do K = 2,m
       V(J,I) = V(J,I) + LAB(I,J,K)*F(K,I)
      end do
     end do
    end do
C   Calculate W = inv(V)
    call CALCINV(V,W,maxn,n)
    return
    end
```

```
    subroutine CALCF1(GAMMA,CGATE,nc,n,m,r,q,F)
C   Subroutine to extract the F1 vectors from gamma for real or
C   complex poles
C   Scalar arguments
    integer n,m,r,q,nc
C   Array arguments
    integer CGATE(10)
    double complex F(9,10)
    double precision GAMMA(81)
C   Local scalars
    integer I,J,POSREAL,POSIMAG
    double precision ZERO
C   Local arrays
```

```
C   Define starting points
    POSREAL = 0
    POSIMAG = 0
    do I = 1,r
     if (CGATE(I).eq.0) then
      POSIMAG = POSIMAG + m
     end if
    end do

C   Due to ordering of pole set (i.e [complex real]), we know
C   that the first numcomplex poles are complex
C   complex poles first
    do I = 1,nc,2
     do J = 1,m
      P(J,I) = CMPLX(GAMMA(J+POSREAL),GAMMA(J+POSIMAG))
      P(J,I+1) = CONJG(P(J,I))
     end do
     POSREAL = POSREAL + m
     POSIMAG = POSIMAG + m
    end do
C   Now real poles
    ZERO = 0.0
    do I = nc+1,r
     do J = 1,m
      P(J,I) = DCMPLX(2.0*GAMMA(J+POSREAL),ZERO)
     end do
     POSREAL = POSREAL + m
    end do

    return
    end
```

```
    subroutine CALCINV(A,B,MAXN,N)

C   subroutine to calculate the inverse of a complex matrix A
C   and store the result in B

C   Scalar arguments
    integer N,MAXN

C   Array arguments
    double complex A(MAXN,MAXN),B(MAXN,MAXN)

C   Local scalars
    integer I,J,ITS,IFAIL,ID
    double precision D1,EPS

C   Local arrays
    double precision RA(20,20),AA(20,20),RB(20,10),BB(20,10),
   *          RHS(20,10),P(20)

C   Declare external functions
    double precision X02AAF

C   Determine EPS
    EPS = X02AAF(0.0)

C   Set up real matrix RA consisting of the real and imaginary
C   parts of A
    do I = 1,N
     do J = 1,N
      RA(I,J) = real(A(I,J))
      AA(I,J) = RA(I,J)
      RA(I,J+N) = -dimag(A(I,J))
      AA(I,J+N) = RA(I,J+N)
      RA(I+N,J) = dimag(A(I,J))
      AA(I+N,J) = RA(I+N,J)
      RA(I+N,J+N) = real(A(I,J))
      AA(I+N,J+N) = RA(I+N,J+N)
     end do
    end do

C   Set up the right hand side consisting of the NxN identity
C   matrix and a zero block
    do I = 1,N
     do J = 1,N
      if (I.eq.J) then
       RHS(I,J) = 1.0
      else
       RHS(I,J) = 0.0
      end if
      RHS(I+N,J) = 0.0
     end do
    end do

C   SOLVE RA.RB = RHS
C   Factorise RA into upper and lower trainglular matrices
    IFAIL = 1
    call F03AFF(2*N,EPS,AA,20,D1,ID,P,IFAIL)
    if (IFAIL.ne.0) then
     print*, 'Error in CALCINV'
     print*, 'F03AFF failed to find a soln, IFAIL = ',IFAIL
     stop
    end if

C   Solve equation to find inverse
    IFAIL = 1
    call F04AHF(2*N,N,RA,20,AA,20,P,RHS,20,EPS,RB,20,BB,20,ITS,
   *          IFAIL)
    if (IFAIL.ne.0) then
     print*, 'Error in F04AHF'
     stop
    end if

C   Extract real and imaginary parts from RB
    do I = 1,N
     do J = 1,N
      B(I,J) = cmplx(RB(I,J),RB(I+N,J))
     end do
    end do

    return
    end
```

```
    subroutine CALCRZ(LAB,F,V,ll,m,r,RZ)

C   Subroutine to calculate RZ
C   Assumes that the poles associated with F2 are real

C   Define parameters
    integer maxn,maxm
    double precision zero
C   (maxn,maxm - maximum values of parameters of system)
C   (zero - maximum value for a number to be considered zero)
    parameter (maxn = 10, maxm = 9)
    parameter (zero = 1e-10)

C   Scalar arguments
    integer ll,m,r

C   Array arguments
    double precision RZ(maxm,maxm)
    double complex LAB(maxn,maxn,maxm),
   *          F(maxm,maxn),V(maxn,maxn)

C   Local scalars
    integer I,J,K

C   Local arrays
    double complex iV11(maxn,maxn),Z(maxm,maxm),T1(maxn,maxn)

C   Calc iV11
    call CALCINV(V,iV11,maxn,r)

C   Calculate F1*iV11, result in T1
    do I = 1,m
     do J = 1,r
      T1(I,J) = 0.0
      do K = 1,r
       T1(I,J) = T1(I,J) + F(I,K)*iV11(K,J)
      end do
     end do
    end do

C   Mult result by LAB, result in Z
    do I = 1,m
     do J = 1,m
      Z(I,J) = 0.0
      do K = 1,r
       Z(I,J) = Z(I,J) + T1(I,K)*LAB(ll,K,J)
      end do
     end do
    end do
```

```
C   Subtract result from identity matrix to give Z
    do I = 1,m
     do J = 1,m
      if (I.eq.J) then
       Z(I,J) = dcmplx(1.0,0.0) - Z(I,J)
      else
       Z(I,J) = -Z(I,J)
      end if
C      Check imaginary part is zero and assign RZ
      if (dimag(Z(I,J)).gt.zero) then
       print*, 'Imaginary part of Z is not zero'
       print*, dimag(Z(I,J))
       stop
      else
       RZ(I,J) = real(Z(I,J))
      end if
     end do
    end do

    return
    end
```

```
    subroutine CALCSVD(A,AM,AN,M,N,DIM,RANK,SPCE)

C   Subroutine to calculate the null or range space of a matrix
C   using the svd, which is calculated by finding the eigenvectors
C   of A*At and At*A

C   Define parameters - (maximum dimensions of A)
    integer maxm,maxn
    parameter (maxm = 50, maxn = 50)

C   Scalar arguments
    integer M,N,AM,AN,DIM,RANK

C   Array arguments
    double precision A(AM,AN),SPCE(AN,AN)

C   Local scalars
    integer I,J,K,IFAIL
    double precision EPS,TOL

C   Local arrays
    double precision AT(maxn,maxm),AA(maxn,maxn),
   *          V(maxn,maxn),EIG(maxn),WKSPCE(maxn)

C   Set epsilon the smallest number
    EPS = 1E-7

C   Find transpose of A
    do I = 1,M
     do J = 1,N
      AT(J,I) = A(I,J)
     end do
    end do

C   Calc V, find A'*A
    do I = 1,N
     do J = 1,N
      AA(I,J) = 0.0
      do K = 1,M
       AA(I,J) = AA(I,J) + AT(I,K)*A(K,J)
      end do
     end do
    end do

C   Find matrix of right eigenvectors = V
    IFAIL = 1
    call F02ABF(AA,maxn,N,EIG,V,maxn,WKSPCE,IFAIL)
    if (IFAIL.ne.0) then
     print*, 'Error in CALCSVD'
     print*, 'Failed to find the eigenstructure of AtA'
     stop
    end if

C   Calculate the rank - based on eigenvalues which are the
C   squares of the singular values. This seems to give better
C   numerical results
    TOL = max(M,N)*sqrt(EIG(N))*EPS
    RANK = 0
    do I = 1,N
     if (EIG(I).gt.EPS) then
      if (sqrt(EIG(I)).gt.TOL) then
       RANK = RANK + 1
      end if
     end if
    end do

C   Return the null space of A
    do I = 1,N-RANK
     do J = 1,N
      SPCE(J,I) = V(J,I)
     end do
    end do
    DIM = N - RANK

    return
    end
```

The cost functions and all associated routines are in the file CFUNC.F

```
    subroutine COST1(n,m,V,W,KC,dA,dB,WEIGHT,FC,aFC,VALSEN)

C   Subroutine to calculate the cost function based on eigenvalue
C   sensitivities

C   Define parameters
    integer maxn,maxm,num
C   (maxn,maxm - maximum values of parameters of system)
C   (num - number of errors being considered)
    parameter (maxn = 10, maxm = 9, num = 1)

C   Scalar arguments
    integer n,m
    double precision FC,aFC

C   Array arguments
    double precision WEIGHT(3*maxn)
    double complex V(maxn,maxn),W(maxn,maxn),KC(maxm,maxn),
   *          VALSEN(num,maxn),dA(num,maxn,maxn),
   *          dB(num,maxn,maxm)

C   Local scalars
    integer I,T
    double precision TEMP

C   Calculate eigenvalue sensitivities
    call EIGVAL(n,m,V,W,KC,dA,dB,VALSEN)

C   Calculate residuals
    FC = 0.0
    aFC = 0.0
    do I = 1,n
     do T = 1,num
      TEMP = real(VALSEN(T,I)*conjg(VALSEN(T,I)))
      FC = FC + WEIGHT(I)*TEMP
      aFC = aFC + TEMP
     end do
    end do

    return
    end
```

```
    subroutine COST2(n,m,F,V,W,KC,LA,LAB,dA,dB,WEIGHT,FC,aFC,
   *          VALSEN,VECSEN)

C   Subroutine to calculate the cost function based on eigenvalue
C   and eigenvector sensitivities

C   Define parameters
    integer maxn,maxm,num
C   (maxn,maxm - maximum values of parameters of system)
C   (num - number of errors being considered)
    parameter (maxn = 10, maxm = 9, num = 1)

C   Scalar arguments
    integer n,m
    double precision FC,aFC
```

```
C   Array arguments
    double precision WEIGHT(3*maxn),VECSEN(num,maxn)
    double complex F(maxm,maxn),V(maxn,maxn),W(maxn,maxn),
   *          LA(maxn,maxn,maxn),LAB(maxn,maxn,maxm),
   *          KC(maxm,maxn),VALSEN(num,maxn),
   *          dA(num,maxn,maxn),dB(num,maxn,maxm)

C   Local scalars
    integer I,J,T
    double precision FC1,aFC1,FC2,aFC2,TEMP,NORMV,NORMW

C   Local arrays
    double complex VSEN(num,maxn,maxn),WSEN(num,maxn,maxn)

C   Calculate eigenvalue sensitivities
    call EIGVAL(n,m,V,W,KC,dA,dB,VALSEN)

C   Calculate eigenvector sensitivities
    call EIGVEC(n,m,F,V,W,KC,LA,LAB,dA,dB,VALSEN,VSEN,WSEN)

C   Calculate the norms of each set of sensitivity vectors
    do I = 1,n
     do T = 1,num
      NORMV = 0.0
      NORMW = 0.0
      do J = 1,n
       NORMV = NORMV + (real(VSEN(T,J,I))**2.0 +
   *             dimag(VSEN(T,J,I))**2.0)
       NORMW = NORMW + (real(WSEN(T,I,J))**2.0 +
   *             dimag(WSEN(T,I,J))**2.0)
      end do
      VECSEN(T,I) = dsqrt(NORMV) + dsqrt(NORMW)
     end do
    end do

C   Calculate function - eigenvalues
    FC1 = 0.0
    aFC1 = 0.0
    do I = 1,n
     do T = 1,num
      TEMP = real(VALSEN(T,I)*conjg(VALSEN(T,I)))
      FC1 = FC1 + WEIGHT(I)*TEMP
      aFC1 = aFC1 + TEMP
     end do
    end do

C   Calculate function - eigenvectors
    FC2 = 0.0
    aFC2 = 0.0
    do I = 1,n
     do T = 1,num
      FC2 = FC2 + WEIGHT(I+n)*VECSEN(T,I)
      aFC2 = aFC2 + VECSEN(T,I)
     end do
    end do

    FC = FC1 + WEIGHT(2*n+1)*FC2
    aFC = aFC1 + aFC2

    return
    end

    ------------------------

    subroutine COST3(n,m,F,V,W,KC,LA,LAB,dA,dB,WEIGHT,FC,aFC,
   *           VALSEN,VECSEN)

C   Subroutine to calculate the cost function based on transient
C   performance using eigenvalue and eigenvector sensitivities

C   Define parameters
    integer maxn,maxm,num
C   (maxn,maxm - maximum values of parameters of system)
C   (num - number of errors being considered)
    parameter (maxn = 10, maxm = 9, num = 1)

C   Scalar arguments
    integer n,m
    double precision FC,aFC
```

```
C   Array arguments
    double precision WEIGHT(3*maxn),VECSEN(num,maxn)
    double complex F(maxm,maxn),V(maxn,maxn),W(maxn,maxn),
   *          LA(maxn,maxn,maxn),LAB(maxn,maxn,maxm),
   *          KC(maxm,maxn),VALSEN(num,maxn),
   *          dA(num,maxn,maxn),dB(num,maxn,maxm)

C   Local scalars
    integer I,T,X,Y
    double precision FC1,aFC1,FC2,aFC2,TEMP,NORM

C   Local arrays
    double complex VSEN(num,maxn,maxn),WSEN(num,maxn,maxn),
   *          T1(maxn,maxn)

C   Calculate eigenvalue sensitivities
    call EIGVAL(n,m,V,W,KC,dA,dB,VALSEN)

C   Calculate eigenvector sensitivities
    call EIGVEC(n,m,F,V,W,KC,LA,LAB,dA,dB,VALSEN,VSEN,WSEN)

C   Calculate the norm of the vector sensitivity matrix
    do I = 1,n
     do T = 1,num
C      Calculate matrix - dVi x Wit, store in T1
       do X = 1,n
        do Y = 1,n
         T1(X,Y) = VSEN(T,X,I)*W(I,Y)
        end do
       end do
C      Calculate matrix - dVi x Wit + Vi x dWit, store in T1
       do X = 1,n
        do Y = 1,n
         T1(X,Y) = T1(X,Y) + V(X,I)*WSEN(T,I,Y)
        end do
       end do
C      Calculate norm - Euclidean norm
       NORM = 0.0
       do X = 1,n
        do Y = 1,n
         NORM = NORM + (real(T1(X,Y))**2.0 +
   *             dimag(T1(X,Y))**2.0)
        end do
       end do
       VECSEN(T,I) = dsqrt(NORM)
     end do
    end do

C   Calculate function - eigenvalues
    FC1 = 0.0
    aFC1 = 0.0
    do I = 1,n
     do T = 1,num
      TEMP = real(VALSEN(T,I)*conjg(VALSEN(T,I)))
      FC1 = FC1 + WEIGHT(I)*TEMP
      aFC1 = aFC1 + TEMP
     end do
    end do

C   Calculate function - eigenvectors
    FC2 = 0.0
    aFC2 = 0.0
    do I = 1,n
     do T = 1,num
      FC2 = FC2 + WEIGHT(I+n)*VECSEN(T,I)
      aFC2 = aFC2 + VECSEN(T,I)
     end do
    end do

    FC = FC1 + WEIGHT(2*n+1)*FC2
    aFC = aFC1 + aFC2

    return
    end

    ------------------------

    subroutine COST4(n,V,W,WEIGHT,FC,aFC)

C   Subroutine to calculate the cost function based on the
C   conditioning of the matrix of eigenvectors
```

```
C   Define parameters
    integer maxn,num
C   (maxn - maximum values of parameters of system)
C   (num - number of errors being considered)
    parameter (maxn = 10, num = 1)

C   Scalar arguments
    integer n
    double precision FC,aFC

C   Array arguments
    double precision WEIGHT(3*maxn)
    double complex V(maxn,maxn),W(maxn,maxn)

C   Local scalars
    integer I,J
    double precision NORMV,NORMW

C   Local arrays
    double complex TEMPV(maxn,maxn),TEMPW(maxn,maxn)

    external CALCINV

    NORMV = 0.0
    NORMW = 0.0
    do I = 1,n
     do J = 1,n
      NORMV = NORMV + (real(V(I,J))**2.0 + imag(V(I,J))**2.0)
      NORMW = NORMW + (real(W(I,J))**2.0 + imag(W(I,J))**2.0)
     end do
    end do
    FC = log(NORMV + NORMW)
    aFC = FC

    return
    end

    ------------------------

    subroutine EIGVAL(n,m,V,W,KC,dA,dB,VALSEN)

C   Subroutine to calculate the eigenvalue sensitivities

C   Define parameters
    integer maxn,maxm,num
C   (maxn,maxm - maximum values of parameters of system)
C   (num - number of errors being considered)
    parameter (maxn = 10, maxm = 9, num = 1)

C   Scalar arguments
    integer n,m

C   Array arguments
    double complex V(maxn,maxn),W(maxn,maxn),KC(maxm,maxn),
   *        VALSEN(num,maxn),dA(num,maxn,maxn),
   *        dB(num,maxn,maxm)

C   Local scalars
    integer I,J,K,L,T
    double complex TEMP1(maxn,maxn),TEMP2(maxn)

    do T = 1,num
C     Calculate dAt + dBt*KC - store in TEMP1
      do J = 1,n
       do K = 1,n
        TEMP1(J,K) = dA(T,J,K)
        do L = 1,m
         TEMP1(J,K) = TEMP1(J,K) + dB(T,J,L)*KC(L,K)
        end do
       end do
      end do
      do I = 1,n
C      Calculate Wi*result - store in TEMP2
       do J = 1,n
        TEMP2(J) = dcmplx(0.0,0.0)
        do K = 1,n
         TEMP2(J) = TEMP2(J) + W(I,K)*TEMP1(K,J)
        end do
       end do
C      Calculate result*Vi - store in VALSEN
       VALSEN(T,I) = dcmplx(0.0,0.0)
       do J = 1,n
        VALSEN(T,I) = VALSEN(T,I) + TEMP2(J)*V(J,I)
       end do
      end do
     end do

     return
     end

     ------------------------

     subroutine EIGVEC(n,m,F,V,W,KC,LA,LAB,dA,dB,VALSEN,VSEN,WSEN)

C    Subroutine to calculate the eigenvalue sensitivities
C    reference - own derivation and Crossley and Porter

C    Define parameters
     integer maxn,maxm,num
C    (maxn,maxm - maximum values of parameters of system)
C    (num - number of errors being considered)
     parameter (maxn = 10, maxm = 9, num = 1)

C    Scalar arguments
     integer n,m

C    Array arguments
     double complex F(maxm,maxn),V(maxn,maxn),W(maxn,maxn),
    *        LA(maxn,maxn,maxn),LAB(maxn,maxn,maxm),
    *        KC(maxm,maxn),VALSEN(num,maxn),
    *        VSEN(num,maxn,maxn),WSEN(num,maxn,maxn),
    *        dA(num,maxn,maxn),dB(num,maxn,maxm)

C    Local scalars
     integer I,T,X,Y,Z

C    Local arrays
     double complex T1(maxn,maxn),T2(maxn,maxn)

C    Calculate the sensitivity of the right eigenvectors
     do T = 1,num
      do I = 1,n

C      Create diagonal matrix of eigenvalue sensitivities
       do X = 1,n
        do Y = 1,n
         if (X.eq.Y) then
          T1(X,Y) = dA(T,X,Y) - VALSEN(T,I)
         else
          T1(X,Y) = dA(T,X,Y)
         end if
        end do
       end do

C      Mult LA by T1 - result in T2
       do X = 1,n
        do Y = 1,n
         T2(X,Y) = dcmplx(0.0,0.0)
         do Z = 1,n
          T2(X,Y) = T2(X,Y) + LA(I,X,Z)*T1(Z,Y)
         end do
        end do
       end do

C      Mult answer, T2 by LAB - result in T1
       do X = 1,n
        do Y = 1,m
         T1(X,Y) = dcmplx(0.0,0.0)
         do Z = 1,n
          T1(X,Y) = T1(X,Y) + T2(X,Z)*LAB(I,Z,Y)
         end do
        end do
       end do

C      Mult LA by dBt and add to T1 - result in T1
       do X = 1,n
        do Y = 1,m
         do Z = 1,n
          T1(X,Y) = T1(X,Y) + LA(I,X,Z)*dB(T,Z,Y)
         end do
        end do
       end do
```

```
C     Mult result, T1 by Fi - result in VSEN
      do X = 1,n
       VSEN(T,X,J) = dcmplx(0.0,0.0)
       do Y = 1,m
        VSEN(T,X,J) = VSEN(T,X,J) + T1(X,Y)*F(Y,J)
       end do
      end do
     end do
    end do
C   Hence calculate the sensitivity of the left eigenrows
    call CWSEN(W,VSEN,WSEN,n)

    return
    end
```

```
    subroutine CWSEN(W,VSEN,WSEN,n)
C   subroutine to calculate WSEN
C   Define parameters
    integer maxn,num
C   {maxn - maximum values of parameters of system}
C   {num - number of errors being considered}
    parameter (maxn = 10, num = 1)
C   Scalar arguments
    integer n
C   Array arguments
    double complex W(maxn,maxn),VSEN(num,maxn,maxn),
   *          WSEN(num,maxn,maxn)
C   Local scalars
    integer I,J,K,T
C   Local arrays
    double complex T1(maxn,maxn)

    do T = 1,num
C     Mult -W x dV, result in T1
      do I = 1,n
       do J = 1,n
        T1(I,J) = dcmplx(0.0,0.0)
        do K = 1,n
         T1(I,J) = T1(I,J) + W(I,K)*VSEN(T,K,J)
        end do
        T1(I,J) = -T1(I,J)
       end do
      end do
C     Mult T1 x W, result in WSEN
      do I = 1,n
       do J = 1,n
        WSEN(T,I,J) = dcmplx(0.0,0.0)
        do K = 1,n
         WSEN(T,I,J) = WSEN(T,I,J) + T1(I,K)*W(K,J)
        end do
       end do
      end do

    end do

    return
    end
```

The routines required to read PRO-MATLAB format files are in the file INPUT.F

```
    subroutine RDATA(fname,S,sr,sc,LA,LAB,CGATE,nc,n,m,r,q)
C   Subroutine to read the data supplied by MATLAB
C   Define parameters
    integer maxn,maxm,maxsr,maxsc
C   {maxn,maxm - maximum values of parameters of system}
C   {maxsr,maxsc - maximum dimension of S}
    parameter (maxn = 10, maxm = 9)
    parameter (maxsr = 81, maxsc = 81)
C   Scalar arguments
    integer sr,sc,nc,n,m,r,q
C   Array arguments
    integer CGATE(maxn)
    double precision S(maxsr,maxsc)
    double complex LA(maxn,maxn,maxn),LAB(maxn,maxn,maxm)
    character*30 fname
C   Local scalars
    integer lunit,irec,row,col,I,J,K
C   Local arrays
    double precision TEMPR(1,maxn)
    double complex TEMPC(maxn*maxn,maxn)
C   Open file
    lunit = 1
    irec = 1
    open(UNIT = lunit, FILE = fname, STATUS = 'old',
   *   FORM = 'unformatted', ACCESS = 'direct', RECL = 1)
C   Load scalars
    call getint(nc,lunit,irec)
    call getint(n,lunit,irec)
    call getint(m,lunit,irec)
    call getint(r,lunit,irec)
    call getint(q,lunit,irec)
C   Load arrays
C   CGATE
    call getmatr(TEMPR,1,maxn,row,col,lunit,irec)
    do I = 1,col
     CGATE(I) = int(TEMPR(1,I))
    end do
C   S
    call getmatr(S,maxsr,maxsc,sr,sc,lunit,irec)
C   LA
    call getmatc(TEMPC,maxn*maxn,maxn,row,col,lunit,irec)
    do I = 1,n
     do J = 1,n
      do K = 1,n
       LA(I,J,K) = TEMPC(J+(I-1)*n,K)
      end do
     end do
    end do
C   LAB
    call getmatc(TEMPC,maxn*maxn,maxn,row,col,lunit,irec)
    do I = 1,n
     do J = 1,n
      do K = 1,m
       LAB(I,J,K) = TEMPC(J+(I-1)*n,K)
      end do
     end do
    end do
C   Close file
    close(lunit)

    return
    end
```

```
    subroutine RSTART(fname,X,dA,dB,WEIGHT,NN)
C   Subroutine to read the data supplied by MATLAB
C   Define parameters
    integer max,maxn,maxm,num
C   {max - maximum number of variables (> 6*maxn)}
    parameter (max = 100, maxn = 10, maxm = 9, num = 1)
C   Scalar arguments
    integer NN
C   Array arguments
    double precision X(NN),WEIGHT(3*maxn)
    double complex dA(num,maxn,maxn),dB(num,maxn,maxm)
    character*30 fname
C   Local scalars
    integer lunit,irec,row,col,I,J,K
C   Local arrays
    double precision TEMPR(1,max)
    double complex TEMPC(num*maxn,maxn)
```

```
C   Open file
    lunit = 1
    irec = 1
    open(UNIT = lunit, FILE = fname, STATUS = 'old',
   *    FORM = 'unformatted', ACCESS = 'direct', RECL = 1)

C   Load arrays

C   X
    call getmatr(TEMPR,1,max,row,col,lunit,irec)
    do I = 1,col
     X(I) = TEMPR(1,I)
    end do

C   dA
    call getmatc(TEMPC,num*maxn,maxn,row,col,lunit,irec)
    do I= 1,num
     do J = 1,row/num
      do K = 1,col
       dA(I,J,K) = TEMPC(J+(I-1)*row/num,K)
      end do
     end do
    end do

C   dB
    call getmatc(TEMPC,num*maxn,maxn,row,col,lunit,irec)
    do I = 1,num
     do J = 1,row/num
      do K = 1,col
       dB(I,J,K) = TEMPC(J+(I-1)*row/num,K)
      end do
     end do
    end do

C   WEIGHT
    call getmatr(TEMPR,1,max,row,col,lunit,irec)
    do I = 1,col
     WEIGHT(I) = TEMPR(1,I)
    end do

C   Close file
    close(lunit)

    return
    end
```

```
    subroutine getmatc(A,ar,ac,row,col,lunit,irec)

C   Subroutine to load a complex matrix from a matlab file

C   Declare parameters
    integer rcmax
    double precision zero
C   (rcmax - maximum value of rows*cols of A)
    parameter (rcmax = 1000, zero = 0.0)

C   Scalar arguments
    integer ar,ac,row,col,lunit,irec

C   Array arguments
    double complex A(ar,ac)

C   Local scalars
    integer I,J,type,imagf,m,n,len,rdflag

C   Local arrays
    double precision RA(rcmax),IA(rcmax)
    character*20 name

C   Load matrix from file
    call LOADMAT(type,m,n,imagf,len,name,RA,IA,lunit,irec,rdflag)
    if (rdflag.ne.0) then
     print*, 'Failed to load complex matrix from file'
     stop
    end if

C   Check if matrix is complex
    if (imagf.eq.0) then
     print*, 'Warning : ',name,' is not complex'
    endif

C   Construct actual complex matrix
    do J = 1,n
     do I = 1,m
      if (imagf.eq.0) then
       A(I,J) = dcmplx(RA(I+(J-1)*m),zero)
      else
       A(I,J) = dcmplx(RA(I+(J-1)*m),IA(I+(J-1)*m))
      end if
     end do
    end do

    row = m
    col = n

    return
    end
```

```
    subroutine getmatr(A,ar,ac,row,col,lunit,irec)

C   Subroutine to load a real matrix from a matlab file

C   Declare parameters
    integer rcmax
C   (rcmax - maximum value of rows*cols of A)
    parameter (rcmax = 6600)

C   Scalar arguments
    integer ar,ac,row,col,lunit,irec

C   Array arguments
    double precision A(ar,ac)

C   Local scalars
    integer I,J,type,imagf,m,n,len,rdflag

C   Local arrays
    double precision RA(rcmax),IA(rcmax)
    character*20 name

C   Load matrix from file
    call LOADMAT(type,m,n,imagf,len,name,RA,IA,lunit,irec,rdflag)
    if (rdflag.ne.0) then
     print*, 'Failed to load real matrix from file'
     stop
    end if

C   Construct actual real matrix
    if (imagf.ne.0) then
     print*, 'Failed to extract matrix as not real'
     stop
    else
     do J = 1,n
      do I = 1,m
       A(I,J) = RA(I+(J-1)*m)
      end do
     end do
    end if

    row = m
    col = n

    return
    end
```

```
    subroutine getint(a,lunit,irec)

C   Subroutine to load a real matrix from a matlab file

C   Declare parameters
    integer rcmax
C   (rcmax - maximum value of rows*cols = 1 for a scalar)
    parameter (rcmax = 1)

C   Scalar arguments
    integer a,lunit,irec

C   Local scalars
    integer type,imagf,m,n,len,rdflag

C   Local arrays
    double precision ar(rcmax),ai(rcmax)
    character*20 name
```

```
C   Load matrix from file
    call LOADMAT(type,m,n,imagf,len,name,ar,ai,lunit,irec,rdflag)
    if (rdflag.ne.0) then
     print*, 'Failed to load real scalar from file'
     stop
    end if
C   Convert to integer
    if (imagf.ne.0) then
     print*, 'Failed to extract scalar as not real'
     stop
    else
     a = int(ar(1))
    end if
    return
    end
```

```
    subroutine LOADMAT(TYPE,M,N,IMAGF,NAMLEN,NAME,
   *                   RPART,IPART,LUNIT,IREC,RDFLG)
C   Subroutine to read matlab files
C   20 byte header
    integer  TYPE,M,N,IMAGF,NAMLEN
C   Character string for name (length of name plus one)
    character  NAME(*)*1
C   Double precision data arrays for example
    double precision  RPART(*),IPART(*)
C   Output file logical unit number
    integer LUNIT
C   Read flag
    integer  RDFLG
C   Direct access record counter
    integer IREC
C   Local scalars
    integer MN
C   Define functions
    integer READC
C   Read header
    if (READC(LUNIT,IREC,4,TYPE)) 998,10,999
10  if (READC(LUNIT,IREC,4,M)) 998,20,999
20  if (READC(LUNIT,IREC,4,N)) 998,30,999
30  if (READC(LUNIT,IREC,4,IMAGF)) 998,40,999
40  if (READC(LUNIT,IREC,4,NAMLEN)) 998,50,999
50  if (READC(LUNIT,IREC,NAMLEN,NAME)) 998,60,999
60  MN = M*N
    if (READC(LUNIT,IREC,8*MN,RPART)) 998,70,999
70  if (IMAGF.eq.1) then
    if (READC(LUNIT,IREC,8*MN,IPART)) 998,80,999
    end if
C   Set read flag to ok and return
80  RDFLG =0
    return
C   Error during read
998 RDFLG = -1
    return
C   End of file
999 RDFLG = 1
    return
    end
```

```
    integer function READC(LUNIT,IREC,NC,CARRAY)
C   Array arguments
    integer LUNIT,NC,IREC
    character CARRAY(*)*1
C   Local scalars
    integer I
    do I=1,NC
    read(LUNIT,rec=IREC,err=998,end=999) CARRAY(I)
    irec = irec + 1
    end do
    READC = 0
    return
998 READC = -1
    return
999 READC = 1
    return
    end
```

The routines to write files in PRO-MATLAB format are in the file OUTPUT.F

```
    subroutine WORIG(NN,m,n,FC,X,F,V,KC,filename)
C   Subroutine to write the original values to file
C   Note dimension of KK is m by n i.e state feedback
C   Define parameters
    integer maxn,maxm
C   (maxn,maxm - maximum values of parameters of system)
    parameter (maxn = 10, maxm = 9)
C   Scalar arguments
    integer NN,m,n
    double precision FC
C   Array arguments
    double precision X(*)
    double complex F(maxm,maxn),V(maxn,maxn),KC(maxm,maxn)
    character*30 filename
C   Local scalars
    integer len,lunit,irec
C   Local arrays
    character*20 name
C   Open file
    lunit = 1
    irec = 1
    open(UNIT = lunit, FILE = filename,
   *   FORM = 'unformatted', ACCESS = 'direct', RECL = 1)
C   Write scalars - FC
    name = 'FCORIG'
    len = 7
    call putr(FC,name,len,lunit,irec)
C   Write arrays
    name = 'XORIG'
    len = 6
    call putvecr(X,NN,NN,name,len,lunit,irec)
    name = 'FORIG'
    len = 6
    call putmatc(F,maxm,maxn,m,n,name,len,lunit,irec)
    name = 'VORIG'
    len = 6
    call putmatc(V,maxn,maxn,n,n,name,len,lunit,irec)
    name = 'KCORIG'
    len = 7
    call putmatc(KC,maxm,maxn,m,n,name,len,lunit,irec)
C   Close file
    close(lunit)
    return
    end
```

```
    subroutine WOPT(NN,m,n,FC,X,dA,dB,WEIGHT,F,V,KC,filename)
C   Subroutine to write the optimal values to file
C   Note dimension of KC is m by n i.e state feedback
```

```
C   Define parameters
    integer maxn,maxm,num
C   (maxn,maxm - maximum values of parameters of system)
    parameter (maxn = 10, maxm = 9, num = 1)
C   Scalar arguments
    integer NN,m,n
    double precision PC
C   Array arguments
    double precision X(*),WEIGHT(3*maxn)
    double complex F(maxm,maxn),V(maxn,maxn),KC(maxm,maxn),
   *          dA(num,maxn,maxn),dB(num,maxn,maxm)
    character*30 filename
C   Local scalars
    integer len,lunit,irec,I,J,K
C   Local arrays
    double complex TEMPC(num*maxn,maxn)
    character*20 name
C   Open file
    lunit = 1
    irec = 1
    open(UNIT = lunit, FILE = filename,
   *   FORM = 'unformatted', ACCESS = 'direct', RECL = 1)
C   Write scalars - PC
    name = 'PC'
    len = 3
    call putr(PC,name,len,lunit,irec)
C   Write arrays
    name = 'X'
    len = 2
    call putvecr(X,NN,NN,name,len,lunit,irec)
    name = 'dA'
    len = 3
    do I = 1,num
     do J = 1,n
      do K = 1,n
       TEMPC(J+(I-1)*n,K) = dA(I,J,K)
      end do
     end do
    end do
    call putrnatc(TEMPC,num*maxn,maxn,num*n,n,name,len,lunit,irec)
    name = 'dB'
    len = 3
    do I = 1,num
     do J = 1,n
      do K = 1,m
       TEMPC(J+(I-1)*n,K) = dB(I,J,K)
      end do
     end do
    end do
    call putrnatc(TEMPC,num*maxn,maxn,num*n,n,name,len,lunit,irec)
    name = 'WEIGHT'
    len = 7
    call putvecr(WEIGHT,3*maxn,3*n,name,len,lunit,irec)
    name = 'F'
    len = 2
    call putrnatc(F,maxm,maxn,m,n,name,len,lunit,irec)
    name = 'V'
    len = 2
    call putrnatc(V,maxn,maxn,n,n,name,len,lunit,irec)
    name = 'KC'
    len = 3
    call putrnatc(KC,maxm,maxn,m,n,name,len,lunit,irec)
C   Close file
    close(lunit)
    return
    end
```

```
    subroutine putrnatc(A,ar,ac,row,col,name,len,lunit,irec)
C   Subroutine to write a complex matrix to file
C   Declare parameters
    integer rcmax
C   (rcmax - maximum value of rows*cols of A)
    parameter (rcmax = 100)
C   Scalar arguments
    integer ar,ac,row,col,len,lunit,irec
C   Array arguments
    double complex A(ar,ac)
    character*20 name
C   Local scalars
    integer I,J,type,imagf,m,n,wtflag
C   Local arrays
    double precision RA(rcmax),IA(rcmax)
C   Check that row*col does not exceed rcmax
    if (row*col.gt.rcmax) then
     print*, 'RCMAX exceeded in putrnatc'
     stop
    end if
C   Set up parameters
    type = 1000
    m = row
    n = col
    imagf = 1
C   Copy data to RA and IA
    do J = 1,col
     do I = 1,row
      RA(I+(J-1)*row) = real(A(I,J))
      IA(I+(J-1)*row) = dimag(A(I,J))
     end do
    end do
C   Write matrix from file
    call SAVEMAT(type,m,n,imagf,len,name,RA,IA,lunit,irec,wtflag)
    if (wtflag.ne.0) then
     print*, 'Failed to write complex matrix to file'
     stop
    end if
    return
    end
```

```
    subroutine putvecr(A,ac,col,name,len,lunit,irec)
C   Subroutine to write a real vector to file
C   Declare parameters
    integer cmax
C   (cmax - maximum value of cols of vector A)
    parameter (cmax = 100)
C   Scalar arguments
    integer ac,col,len,lunit,irec
C   Array arguments
    double precision A(ac)
    character*20 name
C   Local scalars
    integer I,type,imagf,m,n,wtflag
C   Local arrays
    double precision RA(cmax),IA(cmax)
C   Check that row*col does not exceed rcmax
    if (col.gt.cmax) then
     print*, 'CMAX exceeded in putvecr'
     stop
    end if
C   Set up parameters
    type = 1000
    m = 1
    n = col
    imagf = 0
C   Copy data to RA
    do I = 1,col
     RA(I) = A(I)
    end do
```

```
C   Write matrix to file
    call SAVEMAT(type,m,n,imagf,len,name,RA,IA,lunit,irec,wtflag)
    if (wtflag.ne.0) then
     print*, 'Failed to write real vector to file'
     stop
    end if

    return
    end

------------------------------------------------------------

    subroutine putr(a,name,len,lunit,irec)

C   Subroutine to write a real scalar to file

C   Declare parameters
    integer rcmax
C   {rcmax - maximum value of rows*cols = 1 for a scalar}
    parameter (rcmax = 1)

C   Scalar arguments
    integer len,lunit,irec
    double precision a
    character*20 name

C   Local scalars
    integer type,imagf,m,n,wtflag

C   Local arrays
    double precision ar(rcmax),ai(rcmax)

C   Set up parameters
    type = 1000
    m = 1
    n = 1
    imagf = 0

C   Copy data to ar
    ar(1) = a

C   Load matrix from file
    call SAVEMAT(type,m,n,imagf,len,name,ar,ai,lunit,irec,wtflag)
    if (wtflag.ne.0) then
     print*, 'Failed to write real scalar to file'
     stop
    end if

    return
    end

------------------------------------------------------------

    subroutine SAVEMAT(TYPE,M,N,IMAGF,NAMLEN,NAME,
   *                   RPART,IPART,LUNIT,IREC,WTFLG)

C   Subroutine to save files in .mat format

C   20 byte header
    integer  TYPE,M,N,IMAGF,NAMLEN

C   Character string for name (length of name plus one)
    character  NAME(*)*1

C   Double precision data arrays for example
    double precision  RPART(*), IPART(*)

C   Output file logical unit number
    integer LUNIT

C   Write flag
    integer WTFLG

C   Direct access file record counter
    integer IREC

C   Local scalars
    integer MN

    MN = M*N

C   Write header
    call WRITEC(LUNIT,IREC,4,TYPE)
    call WRITEC(LUNIT,IREC,4,M)
    call WRITEC(LUNIT,IREC,4,N)
    call WRITEC(LUNIT,IREC,4,IMAGF)
    call WRITEC(LUNIT,IREC,4,NAMLEN)
    call WRITEC(LUNIT,IREC,NAMLEN-1,NAME)
    call WRITEC(LUNIT,IREC,1,0)
    call WRITEC(LUNIT,IREC,8*MN,RPART)

    if (IMAGF.eq.1)  call WRITEC(LUNIT,IREC,8*MN,IPART)

C   Good write
    WTFLG = 0
    return

C   Error during write
999 WTFLG = -1
    return
    end

------------------------------------------------------------

    subroutine WRITEC(LUNIT,IREC,NC,STRING)

C   Array arguments
    integer LUNIT,IREC,NC
    character STRING(*)*1

C   Local scalars
    integer I

    do I = 1,NC
     write(LUNIT,rec=IREC) STRING(I)
     IREC = IREC + 1
    end do

    return
    end
```

B.3 Post-Optimisation Programs - PRO-MATLAB

After the optimisation a set of programs are needed to convert the state space controller to polynomial form and hence complete the robust polynomial controller design.

The main post-optimisation program is

```
% postopt.m
% this m-file calls all other routines to calculate original
% and robust polynomial controllers
clear
% set up parameters
small = 1e-10;
% set temp = 1 if want to use temposarary file data
temp = 0;
% ENTER INFO
% Enter details of cost func and method
opt = input('JAF (1), FDF (2) : ');
cf = input('Enter number of cost function used : ','s');
if temp ~= 1
 fnum = input('Enter file number : ','s');
end
p = input('Enter value of p : ');
% SET UP FILE INFO
path = ['/user/eepg/eepgkdw/ws/'];
if opt == 1
 opttype = 'JAF';
elseif opt == 2
 opttype = 'FDF';
end
% SET UP MODEL
[a,aorig,int,b,c,n,m,r,q,eigen,cgate] = model(p);
ps = num2str(p);
% LOAD DATA
% load matlab data
eval(['load ' path 'matdata' ps cf])
% load fortran data
if temp == 1
 origfile = [opttype 'tmp_orig'];
 optfile = [opttype 'tmp_opt'];
else
 origfile = ['cost' cf '/' opttype fnum 'orig' ps cf];
 optfile = ['cost' cf '/' opttype fnum 'opt_' ps cf];
end
eval(['load ' path origfile]);
eval(['load ' path optfile]);
% CHECK IMAG PART OF KC IS NEAR ZERO
if sum(sum(imag(KCORIG) > small)) > 0
 imag(KCORIG)
 error('KCORIG is not completely real')
end
KCORIG = real(KCORIG);
if sum(sum(imag(KC) > small)) > 0
 imag(KC)
 error('KC is not completely real')
end
KC = real(KC);
% convert state feedback to o/p feedback
if sum(sum(KCORIG(:,r+1:n) > small)) > 0
 KCORIG(:,r+1:n)
 error('Second part of KCORIG is not zero')
end
KCORIG = KCORIG(:,1:r);
if sum(sum(KC(:,r+1:n) > small)) > 0
 KC(:,r+1:n)
 error('Second part of KC is not zero')
end
KC = KC(:,1:r);
% ORIGINAL CONTROLLER
[F,G,H] = calcsscontrol(KCORIG,c,p);
% save controller polynomials
origfile = [origfile 'FGH'];
eval(['save ' path origfile ' F G H'])
% ROBUST CONTROLLER
[F,G,H] = calcsscontrol(KC,c,p);
% save controller polynomials
optfile = [optfile 'FGH'];
eval(['save ' path optfile ' F G H'])
% CONTROLLER DERIVED FROM DIOPHANTINE EQN - MIN ORDER
[F,G,H] = calcpolycontrol(a,b,c,n,p,eigen);
if temp == 1
 defile = [opttype 'tmp_deqnFGH'];
else
 defile = ['cost' cf '/' opttype fnum 'deqn' ps cf 'FGH'];
end
eval(['save ' path defile ' F G H'])
```

The associated functions are

```
function [F,G,H] = calcsscontrol(K,c,p)
% calculate controller polynomials from feedback matrix
[F,G] = trans_poly(K,p);
% calculate H
gsum = 0;
for i = 1:length(G)
 gsum = gsum + G(i);
end
if length(c) > 0
 csum = 0;
 for i = 1:length(c)
  csum = csum + c(i);
 end
 for i = 1:length(c)
  H(i) = c(i)*gsum/csum;
 end
else
 H(1) = gsum;
end
```

```
function [F,G] = trans_poly(Ky,p);
% function to transform the feedback matrix to polynomial form
% assuming originally in observable canonical form
% check that Ky is the correct dimension
[t1,t2] = size(Ky);
if t1 ~= t2
 error('Ky is not square');
end
if t1 ~= p+1
 error('Ky is of the wrong dimension');
end
if p > 0
```

```
% partition Ky
K11 = Ky(1:1,1:1);
K12 = Ky(1:1,2:p+1);
K21 = Ky(2:p+1,1:1);
K22 = Ky(2:p+1,2:p+1);

% calculate polynomial form
[num,den] = ss2tf(K22,K21,K12,0,1);

% F controller polynomial
F = den;

% G controller polynomial
temp1 = K11*den;
G = -1*addpoly(num,temp1);

else

K11 = Ky;
F = 1;
G = -K11;

end
```

```
function [F,G,H] = calcpolycontrol(a,b,c,n,p,eigen)

% function to calculate the solution to the diophantine equation
% using matrix methods

% SET UP ORDERS (specify number of coeffs)
na = length(a);
nb = length(b);
nf = nb - 1;
ng = na - 1;

% SET UP T
T = [1 -eigen(1)];
for i = 2:n-p
 temp = [1 -eigen(i)];
 T = multpoly(T,temp);
end

% SET UP THE SYLVESTER MATRIX

dimA = max(na+nf-1,nb+ng-1);

% stores a coefficients in A
for i = 1:nf
 temp = a;
 for j = na+1:dimA
  temp = [temp,0];
 end
 if i == 1
  A = temp';
 else
  A = [A,temp'];
 end
 na = na+1;
 a = [0,a];
end

% stores b coefficients in A
for i = 1:ng
 temp = b;
 for j = nb+1:dimA
  temp = [temp,0];
 end
 A = [A,temp'];
 nb = nb+1;
 b = [0,b];
end

% ASSIGN RHS
% calculate the RHS
if length(c) == 0
 rhs = T';
else
 rhs = multpoly(c,T)';
end

% CALC F AND G
% must ensure RHS is of the same, or smaller, dimension as A
lrhs = length(rhs);
[t1,t2] = size(A);
if t1 ~= t2
 error('A matrix not square')
end
if lrhs > t1
 error('RHS larger than A')
end
if lrhs < t1
 for i = lrhs+1:t1
  rhs = [rhs' 0]';
 end
end

% calculate the controller coefficients
x = A\rhs;
F = x(1:nf,:)';
G = x(nf+1:nf+ng,:)';

% H = sum of the G coefficients divided by sum of C coefficients
% multiplied by C
gsum = 0;
for i = 1:ng
 gsum = gsum + G(i);
end
if length(c) == 0
 H(1) = gsum;
else
 csum = 0;
 for i = 1:length(c)
  csum = csum + c(i);
 end
 for i = 1:length(c)
  H(i) = c(i)*gsum/csum;
 end
end
```

APPENDIX C

DETAILS OF THE NAG LIBRARY ROUTINES

C.1 Accurate Inverse of a Real Matrix

As previously mentioned in chapter 5, the inverse of a real matrix, A, can be accurately obtained by solving the equation

$$A X = I \qquad (C.1)$$

where I is the identity matrix and X the inverse of A.

This type of equation can be solved by a number of methods and in this case two NAG library routines are used. The first F03AFF computes an LU factorisation of the matrix with partial pivoting and the second, F04AHF, uses the result and iterative refinement to obtain the solution.

This section gives some brief details of the two routines, further information can be found in NAG (1990)

F03AFF

Specification

SUBROUTINE F03AFF(*N,EPS,A,IA,D1,ID,P,IFAIL*)

INTEGER	*N,IA,ID,IFAIL*
DOUBLE PRECISION	*EPS,A(IA,N),D1,P(N)*

Parameters

N	Input	On entry, *N* specifies the order of the matrix *A*.
EPS	Input	On entry, *EPS* must be set to the value of machine precision. This value is implementation dependant.

$A(IA,N)$	Input/Output	On entry, the N by N matrix A. On exit, A is overwritten by the lower triangular matrix L and the off-diagonal elements of the upper triangular matrix U. The unit diagonal elements of U are not stored.
IA	Input	On entry, IA specifies the first dimension of the array A as declared in the (sub) program from which F03AFF is called. $IA \geq N$
$D1$	Output	On exit, $D1$ can be used to calculate the determinant of A.
ID	Output	On exit, ID is also used to calculate the determinant of A.
$P(N)$	Output	On exit, $P(i)$ gives the row index of the i'th pivot.
$IFAIL$	Input/Output	On entry, it is recommended that $IFAIL$ be set to 0. Further information can be found in NAG (1990). On exit, $IFAIL = 0$: successful termination $IFAIL = 1$: A is singular, possibly due to rounding errors

To avoid overflow or underflow, the determinant can be calculated using

$$\det(A) = D1\,(2.0)^{ID} \qquad (C.2)$$

F04AHF

Specification

SUBROUTINE F04AHF(*N,IR,A,IA,AA,IAA,P,B,IB,EPS,X,IX,BB,IBB,K,IFAIL*)

INTEGER	*N,IR,IA,IAA,IB,IX,IBB,K,IFAIL*
DOUBLE PRECISION	*A(IA,N),AA(IAA,N),P(N),B(IB,IR),EPS,X(IX,IR),BB(IBB,IR)*

Parameters

N	Input	On entry, N specifies the order of the matrix A.
IR	Input	On entry, IR specifies the number of right hand sides. For the calculation of the inverse $IR = N$.
$A(IA,N)$	Input	On entry, the N by N matrix A.

IA	Input	On entry, *IA* specifies the first dimension of the array *A* as declared in the (sub) program from which F04AHF is called. $IA \geq N$
AA(*IAA*,*N*)	Input	On entry, *AA* contains details of the *LU* factorisation as returned by F03AFF.
IAA	Input	On entry, *IAA* specifies the first dimension of the array *AA* as declared in the (sub) program from which F04AHF is called. $IAA \geq N$
P(*N*)	Input	On entry, *P*(*N*) contains details of the row interchanges as returned by F03AFF.
B(*IB*,*IR*)	Input	On entry, the *N* by *IR* right hand side matrix *B*. For the calculation of the inverse this will be set to the *N* by *N* identity matrix.
IB	Input	On entry, *IB* specifies the first dimension of the array *B* as declared in the (sub) program from which F04AHF is called. $IB \geq N$
EPS	Input	On entry, *EPS* must be set to the value of machine precision. This value is implementation dependant.
X(*IX*,*IR*)	Output	On exit, the *N* by *IR* solution matrix *X*.
IX	Input	On entry, *IX* specifies the first dimension of the array *X* as declared in the (sub) program from which F04AHF is called. $IX \geq N$
BB(*IBB*,*IR*)	Output	On exit, the *N* by *IR* residual matrix $R = B - AX$.
IBB	Input	On entry, *IBB* specifies the first dimension of the array *BB* as declared in the (sub) program from which F04AHF is called. $IBB \geq N$
K	Output	On exit, *K* specifies the number of iterations needed in the refinement process.

IFAIL	Input/Output	On entry, it is recommended that *IFAIL* be set to 0. Further information can be found in NAG (1990). On exit, *IFAIL* = 0 : successful termination *IFAIL* = 1 : The matrix *A* is too ill-conditioned to produce a correctly rounded solution.

C.2 Calculation of the Null Space of a Matrix

It was shown in chapter 5 that the null space of a matrix can be found using the singular value decomposition (SVD) and that it is necessary to obtain the eigenvectors of a matrix to carry out this decomposition. The NAG library routine F02ABF can be used to obtain the eigenvalues and eigenvectors of a matrix and hence allow the null space to be found. Again further information can be found in NAG (1990).

Specification

SUBROUTINE F02ABF(*A,IA,N,R,V,IV,E,IFAIL*)

INTEGER	*IA,N,IV,IFAIL*
DOUBLE PRECISION	*A(IA,N),R(N),V(IV,N),E(N)*

Parameters

A(IA,N)	Input	On entry, the lower triangle of the *N* by *N* matrix *A*. The elements of the matrix above the diagonal need not be set.
IA	Input	On entry, *IA* specifies the first dimension of the array *A* as declared in the (sub) program from which F02ABF is called. $IA \geq N$
N	Input	On entry, *N* specifies the order of the matrix *A*.
R(N)	Output	On exit, the eigenvalues in ascending order.
V(IV,N)	Output	On exit, the normalised eigenvectors, stored by columns.
IV	Input	On entry, *IV* specifies the first dimension of the array *V* as declared in the (sub) program from which F02ABF is called. $IV \geq N$
E(N)	-	Used as workspace.

IFAIL	Input/Output	On entry, it is recommended that *IFAIL* be set to 0. Further information can be found in NAG (1990). On exit, *IFAIL* = 0 : successful termination *IFAIL* = 1 : More than $30 \times N$ iterations are required to isolate all the eigenvalues.

C.3 Non-linear Optimisation

There are a number of NAG optimisation routines for various types of problem. The most suitable routine in this case is E04JAF. It is a quasi-Newton algorithm for finding the minimum of a function without explicit first or second order derivative information. It provides the facility for specifying bounds which will not be used in this case. Further information on this routine and other optimisation algorithms can be found in NAG (1990).

Specification

SUBROUTINE E04JAF(*N,IBOUND,BL,BU,X,F,IW,LIW,W,LW,IFAIL*)
INTEGER *N,IBOUND,IW(LIW),LIW,LW,IFAIL*
DOUBLE PRECISION *BL(N),BU(N),X(N),F,W(LW)*

Parameters

N	Input	On entry, *N* specifies the number of independent variables.
IBOUND	Input	On entry, To specify no bounds *IBOUND* = 2.
BL,BU	-	Not used.
X(N)	Input/Output	On entry, *X* specifies the starting point. On successful exit, it contains the position of the minimum.
F	Output	On exit, *F* contains the value of the function at the minimum.
IW(LIW)	-	Used as workspace
LIW	Input	On entry, *LIW* specifies the first dimension of the array *IW* as declared in the (sub) program from which E04JAF is called. $LIW \geq N + 2$
W(IW)	-	Used as workspace

LW	Input	On entry, *LW* specifies the first dimension of the array *W* as declared in the (sub) program from which E04JAF is called. $LW \geq \max((N(N-1)/2)+12N, 13)$
IFAIL	Input/Output	On entry, it is recommended that *IFAIL* be set to -1. Further information can be found in NAG (1990). On exit,

IFAIL = 0 : successful termination

IFAIL = 1 : specified parameter not in required range.

IFAIL = 2 : there have been $400 \times N$ function evaluations yet the algorithm does not seem to be converging.

IFAIL = 3 : the conditions for a minimum have not all been satisfied but no lower point could be found.

IFAIL = 4 : an overflow has occurred.

IFAIL = 5 :
IFAIL = 6 :
IFAIL = 7 :
IFAIL = 8 : there is some doubt about whether the point found is a minimum. The degree of confidence decreases as *IFAIL* increases.

IFAIL = 9 : The modulus of one of the variables has become very large.

The routine needs an associated user specified routine to calculate the value of the cost function at any given point. The routine must be declared as EXTERNAL in the calling (sub) program and have the following format.

Specification

SUBROUTINE FUNCT1(*N*,*XC*,*FC*)

INTEGER	*N*
DOUBLE PRECISION	*XC*(*N*),*FC*

Parameters

N	Input	On entry, *N* specifies the number of independent variables.
XC(*N*)	Input	On entry, *XC* specifies the point at which the cost function is to be evaluated.
FC	Output	On exit, *FC* contains the value of the cost function.

REFERENCES

NAG (1990)
'Fortran Library Manual Mk 14'
Numerical Algorithms Group Ltd, Oxford, U.K.

APPENDIX D

ACSL SIMULATION PROGRAMS

D.1 Open-Loop Simulation

This program simulated the open-loop hydraulic rig with nominal parameter values. Whilst in the interactive ACSL environment these parameters can be easily altered using the SET command to allow simulations of the perturbed system. The OUTPUT command is used to effectively perform sampling as the displayed data is sent to a data file every NCIOUT*CINT iterations, where CINT is the time interval at which the DYNAMIC section is executed. An appropriate choice of NCIOUT selects the sampling rate. The simulation is based on the fourth order Runge-Kutta integration algorithm.

```
PROGRAM OLRIG
"-----OPEN LOOP SIMULATION OF THE HYDRAULIC RIG-----"
REAL TSTOP,AMP,OFFSET,TZ,PERIOD,WIDTH
REAL PS,B,KTHETA,KS,CR,ETAM,ETAP,I,D,VT,K1,KT,PP,TP,PMIC,THDIC

"-----SET DEMAND SIGNAL PARAMETERS AND STOP TIME-----"
CONSTANT TSTOP=1.5, AMP=0.16, OFFSET=0.72
CONSTANT TZ=0.1, PERIOD=0.3333, WIDTH=0.16666

"-----SET HYDRAULIC RIG PARAMETERS-----"
CONSTANT PS=68.96E5
CONSTANT PP=22.98E5
CONSTANT B=7000E5
CONSTANT KTHETA=2.4E-6
CONSTANT KS=0.0625
CONSTANT CR=9.56E-7
CONSTANT ETAM=1
CONSTANT ETAP=1
CONSTANT I=1.08E-4
CONSTANT D=5.94E-4
CONSTANT VT=3.51E-5
CONSTANT K1=2.12E-13
CONSTANT KT=8.0E-3

INITIAL

 "-----SET UP CINT AND THE NUMBER OF INTEGRATION STEPS-----"
 CINTERVAL CINT=0.0012
 NSTEPS   NSTEP=10
 "-----CALCULATE ANY DEPENDANT EQNS AND SET INITIAL CONDITIONS-----"
 TP = PP*CR*(1/ETAP)
 PMIC=0.0
 THDIC=0.0
END $"OF INITIAL"

DYNAMIC
 DERIVATIVE
  "-----SET UP DEMAND SIGNAL-----"
  VI=AMP*PULSE(TZ,PERIOD,WIDTH) + OFFSET
  "-----CALCULATE FLOW RATE THROUGH VALVE-----"
  QV=KTHETA*KS*VI*SQRT(PS-PMB)
  "-----CALCULATE THE PRESSURE DIFFERENTIAL ACROSS THE MOTOR-----"
  PM=INTEG(((QV-CR*THD-K1*PMB)*2*B/VT),PMIC)
  "-----PM CANNOT EXCEED THE SUPPLY PRESSURE-----"
  PMB=BOUND(-PS,PS,PM)
  "-----CALCULATE THE MOTOR TORQUE-----"
  TM=PM*CR*ETAM
  "-----CALCULATE THE VELOCITY-----"
  THD=INTEG((TM-D*THD-TP)/I,THDIC)
  "-----HENCE THE OUTPUT VOLTAGE-----"
  VO=KT*THD
  "-----CHECK WHETHER STOP TIME EXCEEDED-----"
  TERMT(T.GE.TSTOP)
 END $"OF DERIVATIVE"
END $"OF DYNAMIC"
END $"OF PROGRAM"
```

D.2 Closed-Loop Simulation

This section contains the program used to perform the closed-loop simulation of the hydraulic rig. The DYNAMIC section which simulates the actual rig is the same as for the open-loop simulation. The controller is contained in the DISCRETE section which simulates its implementation on a computer. The interval at which this section is executed is determined by DTSAMP.

All four controllers are defined and whilst in the ACSL environment the SET command can be used to change the value of TYPE and hence select which controller is to be used. The appropriate values are

1 - The minimum order controller
2 - The robust P_s controller
3 - The robust P_p controller
4 - The robust P_s and P_p controller

The output of the integrator has been rate limited and bounds set on its output as in Daley (1987).

```
PROGRAM CLRIG

"-----CLOSED LOOP SIMULATION OF THE HYDRAULIC RIG-----"

REAL TSTOP,AMP,OFFSET,TZ,PERIOD,WIDTH
REAL PS,B,KTHETA,KS,CR,ETAM,ETAP,I,D,VT,K1,KT,PP,TP,PMIC,THDIC
INTEGER LH(4),LG(4),LF(4),X,TYPE,LENGTH
REAL VC,UDOT,H(1,4),G(4,4),F(4,4),Y(10),U(10),UI(10),W(10)
REAL RATEL,RATEU,MAGL,MAGU

"-----SELECT CONTROLLER-----"
CONSTANT TYPE=1

"-----SET DEMAND SIGNAL PARAMETERS AND STOP TIME-----"
CONSTANT TSTOP=3.0, AMP=0.8, OFFSET=2.6
CONSTANT TZ=0.1, PERIOD=0.66666, WIDTH=0.3333

"-----SET HYDRAULIC RIG PARAMETERS-----"
CONSTANT PS=68.96E5
CONSTANT PP=22.98E5
CONSTANT B=7000.0E5
CONSTANT KTHETA=2.4E-6
CONSTANT KS=0.0625
CONSTANT CR=9.56E-7
CONSTANT ETAM=1
CONSTANT ETAP=1
CONSTANT I=1.08E-4
CONSTANT D=5.94E-4
CONSTANT VT=3.51E-5
CONSTANT K1=2.12E-13
CONSTANT KT=8.0E-3


INITIAL

"-----SET UP CINT AND THE NUMBER OF INTEGRATION STEPS-----"
CINTERVAL CINT=0.005
NSTEPS  NSTEP=10

"-----CALCULATE ANY DEPENDANT EQNS AND SET INITIAL CONDITIONS-----"
TP = PP*CR*(1/ETAP)
PMIC=0.0
THDIC=0.0

"-----DEFINE MINIMUM ORDER CONTROLLER-----"
LH(1)=1
LG(1)=2
LF(1)=1
H(1,1)=0.0818
G(1,1)=-0.1096
G(2,1)=0.1914
G(3,1)=0.0
G(4,1)=0.0
F(1,1)=1.0
F(2,1)=0.0
F(3,1)=0.0
F(4,1)=0.0

"-----DEFINE ROBUST PS CONTROLLER-----"
LH(2)=1
LG(2)=4
LF(2)=4
H(1,2)=0.0818
G(1,2)=0.1374
G(2,2)=-0.0594
G(3,2)=0.0038
G(4,2)=8.8239E-7
F(1,2)=1.0
F(2,2)=-0.6412
F(3,2)=-1.4714E-4
F(4,2)=2.023E-10

"-----DEFINE ROBUST PP CONTROLLER-----"
LH(3)=1
LG(3)=4
LF(3)=4
H(1,3)=0.0818
G(1,3)=0.0544
```

```
G(2,3)=0.0248
G(3,3)=0.0026
G(4,3)=5.9111E-7
P(1,3)=1.0
P(2,3)=-0.4256
P(3,3)=-9.7342E-5
P(4,3)=2.0230E-10
"-----DEFINE ROBUST PS AND PP CONTROLLER-----"
LH(4)=1
LG(4)=4
LP(4)=4
H(1,4)=0.0818
G(1,4)=0.1321
G(2,4)=-0.054
G(3,4)=0.0037
G(4,4)=8.6386E-7
P(1,4)=1
P(2,4)=-0.6274
P(3,4)=-1.4397E-4
P(4,4)=2.023E-10
"----INITIALISE SIGNALS----"
DO L1 X = 1,10
  Y(X) = 0.0
  U(X) = 0.0
  UI(X) = 0.0
  W(X) = 0.0
L1..CONTINUE
VC = 0.0
"-----SET INTEGRATOR OUTPUT BOUNDS-----"
RATEL = -0.5
RATEU = 0.5
MAGL = -2.0
MAGU = 2.0
END $"OF INITIAL"

DYNAMIC
 DERIVATIVE
  VI=AMP*PULSE(TZ,PERIOD,WIDTH) + OFFSET
  QV=KTHETA*KS*VC*SQRT(PS-PMB)
  PM=INTEG(((QV-CR*THD-K1*PMB)*2*B/VT),PMIC)
  PMB=BOUND(-PS,PS,PM)
  TM=PM*CR*ETAM
  THD=INTEG((TM-D*THD-TP)/I,THDIC)
  VO=KT*THD
  TERMT(T.GE.TSTOP)
END $"OF DERIVATIVE"

DISCRETE
 INTERVAL DTSAMP=0.012
 PROCEDURAL (VC=VO,VI)
  "-----MAINTAIN ARRAYS OF LAST TEN SAMPLES-----"
  DO L2 X = 1,9
    Y(X) = Y(X+1)
    W(X) = W(X+1)
    U(X) = U(X+1)
    UI(X) = UI(X+1)
  L2..CONTINUE
  "-----SAMPLE INPUT AND OUTPUT-----"
  Y(10) = VO
  W(10) = VI
  "-----CALCULATE CONTROL SIGNAL-----"
  U(10) = 0.0
  LENGTH = LH(TYPE)
  DO L3 X = 1,LENGTH
    U(10) = U(10) + H(X,TYPE)*W(11-X)
  L3..CONTINUE
  LENGTH = LG(TYPE)
  DO L4 X = 1,LENGTH
    U(10) = U(10) - G(X,TYPE)*Y(11-X)
  L4..CONTINUE
  LENGTH = LP(TYPE)
  DO L5 X = 2,LENGTH
    U(10) = U(10) - P(X,TYPE)*U(11-X)
  L5..CONTINUE
  "-----CALCULATE INTEGRATED CONTROL SIGNAL UI-----"
  UI(10) = U(10) + UI(9)
  "-----RATE LIMIT CONTROL SIGNAL-----"
  UDOT = UI(10) - UI(9)
  IF (UDOT.LT.RATEL) UI(10) = UI(9)+RATEL
  IF (UDOT.GT.RATEU) UI(10) = UI(9)+RATEU
  "-----MAGNITUDE BOUNDS ON CONTROL SIGNAL-----"
  IF (UI(10).LT.MAGL) UI(10) = MAGL
  IF (UI(10).GT.MAGU) UI(10) = MAGU
  VC = UI(10)
 END $"OF PROCEDURAL"
END $"OF DISCRETE"
END $"OF DYNAMIC"
END $"OF PROGRAM"
```

NOMENCLATURE AND SYMBOLS

GENERAL

RHS	Right hand side.
w.r.t	With respect to.
z^{-1}	Backward shift operator.
$a(t)$	Discrete time signal.
$a(t-n)$	Value of $a(t)$ at the n'th previous sample.
$E[a^2(t)]$	Variance of $a(t)$.
$A_p(z^{-1})$	Polynomial in terms of the backward shift operator. Note that in chapter 2 polynomials are expressed as A_p to make the notation more easy to follow.
$a_0, a_1, \cdots$	Coefficients of the polynomial $A_p(z^{-1})$.
$\hat{A}_p(z^{-1})$	Estimate of $A_p(z^{-1})$.
$A_p(1)$	Steady state value of $A_p(z^{-1})$.
I_n	Identity matrix of dimension $n \times n$.
0_{ij}	Zero matrix of dimension $i \times j$.
diag[·]	Diagonal matrix.
$\underline{a}$	Column vector.
$\underline{a}^T$	Row vector.
$\|\cdot\|$	General vector or matrix norm.
$\|\cdot\|_F$	Frobenius norm.
$\|\cdot\|_p$	P or Holder norm, where $p = 1,2$ or ∞.
$\kappa(A)$	Condition number of the matrix A, defined as $\kappa(A) = \|A\| \, \|A^{-1}\|$
$[A]_x$	The first x rows of the matrix A.
A_y	The first y columns of the matrix A.
A_y	Used in context to represent the sub-block of the matrix A.
ΔA	Incremental change in A.
dA	Differential of A.
A^*	Complex conjugate of A.
R^n	Real vector space of real n-dimensional vectors.

POLYNOMIAL SYSTEMS

$w(t)$	Demand signal.
$u(t)$	Control input.
$y(t)$	System output.
$e(t)$	White noise sequence.
σ_e^2	Variance of $e(t)$
$d(t)$	Disturbance term.
$A_p(z^{-1})$, $B_p(z^{-1})$	Open-loop system polynomials.
$a_1, a_2, \cdots$	Coefficients of $A_p(z^{-1})$.
$b_0, b_1, \cdots$	Coefficients of $B_p(z^{-1})$.
n_a, n_b	Orders of $A_p(z^{-1})$ and $B_p(z^{-1})$ respectively.
$C_p(z^{-1})$	Colouring polynomial for the white noise sequence $e(t)$.
$c_1, c_2, \cdots$	Coefficients of $C_p(z^{-1})$.
n_c	Order of $C_p(z^{-1})$.
t_d	Time delay.
$F_p(z^{-1})$, $G_p(z^{-1})$	Controller polynomials.
$f_0, f_1, \cdots$	Coefficients of $F_p(z^{-1})$.
$g_0, g_1, \cdots$	Coefficients of $G_p(z^{-1})$.
n_f, n_g	Orders of $F_p(z^{-1})$ and $G_p(z^{-1})$ respectively.
$H_p(z^{-1})$	Precompensator polynomial.
$h_0, h_1, \cdots$	Coefficients of $H_p(z^{-1})$.
n_h	Order of $H_p(z^{-1})$.
$H_p'(z^{-1})$	Precompensator term selected for zero steady state error.
$n_{h'}$	Order of $H_p'(z^{-1})$.
$T_p(z^{-1})$	Polynomial used to specify the desired closed-loop pole positions.
$t_1, t_2, \cdots$	Coefficients of $T_p(z^{-1})$.
n_t	Order of $T_p(z^{-1})$.

Solution of the Diophantine Equation by Polynomial Methods

GCD	Greatest common divisor.
g_p	GCD of $A_p(z^{-1})$ and $B_p(z^{-1})$.
F_0, G_0	Solution to the diophantine equation.
X_p	Arbitrary polynomial in general solution.
C_0T_0	Result of dividing $C_p(z^{-1})T_p(z^{-1})$ by g_p and must be polynomial for a solution to exist.
P_p, Q_p	Pair of coprime polynomials which satisfy $A_pP_p + B_pQ_p = g_p$.
R_p, S_p	Pair of coprime polynomials which satisfy $A_pR_p + B_pS_p = 0$.
u_p	Quotient polynomial.
V_p	Remainder polynomial.

Solution of the Diophantine Equation by Matrix Methods

A_s	Sylvester matrix.
$\underline{x}$	Vector of the coefficients of $F_p(z^{-1})$ and $G_p(z^{-1})$.
$\underline{b}$	Vector of the coefficients of $C_p(z^{-1})T_p(z^{-1})$.
U_b	Upper bound on the relative perturbation, $\|\Delta\underline{x}\|/\|\underline{x}\|$.
N_c	Number of columns of A_s.
N_r	Number of rows of A_s.
N_u	Number of unknowns (the number of $F_p(z^{-1})$ and $G_p(z^{-1})$ coefficients).
N_e	Number of equations represented by $A_s\underline{x} = \underline{b}$.

STATE SPACE SYSTEMS

n	Number of states.
m	Number of inputs.
r	Number of outputs.
k	Discrete time sequence.
$u(k)$	Input signal equivalent to the input for the polynomial system.
$y(k)$	Output signal equivalent to the output for the polynomial system.
$\underline{u}(k)$	Multivariable input signal (with augmented dynamic compensator).
$\underline{y}(k)$	Multivariable output signal (with augmented dynamic compensator).
$\underline{x}(k)$	State vector.
A, B, C	Open-loop system matrices
A_c	Closed-loop system matrix.
p	Order of the dynamic compensator.

K_y	Output feedback controller gain matrix.
K_x	State feedback controller gain matrix.
λ_i	The i'th closed-loop eigenvalue.
Λ	Set of closed-loop eigenvalues.
$\underline{v}_i$	The i'th right eigenvector.
V	Matrix of right eigenvectors.
$\underline{f}_i$	The associated i'th right free parameter vector.
F	Matrix of right free parameter vectors.
$\underline{w}_i^T$	The i'th left eigenrow.
W	Matrix of left eigenrows.
$\underline{g}_i^T$	The associated i'th left free parameter vector.
G	Matrix of left free parameter vectors.

The Parametric Method of Fahmy and O'Reilly

s	Defines the split for the multistage design.
$\tilde{B}$	Input reduction matrix.
$\tilde{C}$	Output reduction matrix.
K	General Output feedback controller gain matrix obtained by either the first or second stage.
K_1	Controller gain matrix obtained by the first stage.
K_2	Controller gain matrix obtained by the second stage.
$\tilde{K}$	Output feedback controller gain matrix obtained after the application of either an input or output reduction matrix.

The Parametric Method of Daley

q	$= n - r$.
F_1	The first r columns of the matrix F.
F_2	The $r+1$ to n columns of the matrix F.
$V_{11}, V_{12},$ V_{21}, V_{22}	Sub-blocks of the matrix V.
$\underline{v}_i^{11}$	i'th column of V_{11}.
$\underline{\Gamma}$	Vector consisting of all the vectors in F_1.
ζ	$\underline{\Gamma}$ must lie in the null space of this matrix.
Z_i	For $i = r+1 \rightarrow n$, specifies the set of matrices whose null spaces the F_2 vectors must lie in.

COST FUNCTIONS

u	Number of error terms.
P_t	Structural information on the errors in A.
Q_t	Structural information on the errors in B.
ε_t	Unknown magnitude of the t'th error term.
J_1	Eigenvalue differential cost function.
J_2	Eigenstructure differential cost function.
J_3	Transient response differential cost function.
J_4	Conditioning cost function.
β_{it}	Weights used in the eigenvalue differential cost function.
η_{it}	Weights used in the eigenstructure differential cost function.
γ_{it}	Weights used in the transient response differential cost function.
σ	Weight used when combining terms in a cost function.

IMPLEMENTATION AND APPLICATION OF THE METHOD

SVD	Singular value decomposition.
A_r	Real matrix or real part of a complex matrix.
A_i	Imaginary part of a complex matrix.
A_c	Used in context to denote a complex matrix.
N_f	Number of free parameters.
N_{re}	Number of residuals.
$\underline{X}^T$	Vector specifying the initial values for the free parameters.
$\underline{W}^T$	Vector specifying the values of the weights
J_{orig}	Original value of the cost function.
J_{opt}	Final value of the cost function.

HYDRAULIC RIG

Q_v	Flow rate through the spool valve.
X_s	Spool valve displacement.
P_s	Supply pressure.
P_m	Pressure differential across the motor.
K_θ	Valve flow coefficient.
θ	Shaft position.
C_r	Motor displacement.

V_t	Total trapped volume.
B	Oil bulk modulus.
K_1	Leakage coefficient.
T_m	Motor torque.
η_m	Efficiency of the motor.
I	Total inertia.
D	Viscous friction coefficient.
P_p	Pressure differential across the pump.
η_p	Efficiency of the pump.
K_s	Pure gain term for the servo and torque motor.
K_t	Tachometer constant.
u	Input voltage.
y	Output voltage.
V_N	Loss function.
FPE	Akaike's final prediction error.
N	Number of data points.
P	Number of parameters in the model.
θ_p	Parameter vector.
$\varepsilon(t, \theta_p)$	Prediction error.