THE DESIGN OF DECENTRALISED CONTROLLERS FOR LARGE SCALE SYSTEMS

A thesis submitted to the Department of Electrical Engineering, University of Cape Town, in fulfilment of the requirements for an MSc. Degree in Electrical Engineering

by

A.B.J. GEAR

February 1988

ACKNOWLEDGEMENTS

I would like to thank the following:

The Council for Mineral Technology for providing the funding for this research.

My supervisor Dr. M. Braae for his guidance and assistance throughout the project.

Mr. J. G. Jack for his advice on word processing and Unix.

SYNOPSIS

Decentralised control schemes are becoming more common in industry as the advantages of decentralised control become more apparent. These advantages include fewer tuning parameters than centralised controllers, the simplification and cost reduction of hardware requirements and greater reliability. In addition the application of decentralised controller design to large scale systems allows established CAD methods to be implemented easily and efficiently.

When the control engineer designs a distributed controller the system is divided up into a number of subsystems and a controller designed for each subsystem. The controllers are designed independently for each subsystem ignoring any interaction that may occur between the different subsystems. In terms of the input-output representation of the system this means that the matrix representing the controller will be in a block diagonal form.

In general the interactions between the different subsystems will not be negligible. In some cases the interactions will be such that stabilising the individual subsystems will not be sufficient to stabilise the system as a whole. Stability theorems are required to enable the designer to check if the decentralised controller that he has designed will in fact stabilise the system as a whole. Such stability theorems have been devised although at present they are too conservative. However even with such theorems available the designer must still select the subsystems to be controlled in such a way as to satisfy the conditions laid down for stability.

The stability theories usually are based on a particular matrix structure. If the matrix representing the system possesses a structure detailed by the stability theorem in

question then, subject to various conditions, the system as a whole will be stable under decentralised control. In this thesis a number of different matrix structures are considered that give information as to the stability of the closed loop system. Methods are developed that allow the designer to rearrange the matrix in such a way as to obtain a particular structure, if this is possible.

A review of the current theory in the field of decentralised controller design is given. A number of different matrix structures are discussed including block diagonal and block triangular matrices, diagonally dominant matrices, block diagonally dominant matrices and quasi-block diagonally dominant matrices. The use of interaction measures to determine the suitability of a particular matrix structure for decentralised controller design is also discussed.

The block triangular and block diagonal structures are now considered in detail. These structures are useful for a number of reasons. They depend only on the distribution of zero and non-zero elements in the matrix representing the system. Hence an analysis based on these structures can be done before the system has been completely modelled. Another advantage is that the stability theorems associated with these structures are simple and easy to test. Two algorithms are developed to analyse the system matrix. The first rearranges the matrix to be block triangular, if this is possible. The second rearranges the matrix to be block diagonal if such a structure is obtainable. A number of examples are given to illustrate the use of these algorithms.

Next the diagonal dominance structure is considered. A method is developed whereby the designer can can easily see the relative dominances of the different elements in the matrix. Any elements that are dominant on a row or column

are easily detected by means of a simple graphical test. A program has been written to implement this method and this is used in a number of examples. The problem of scaling a matrix to be diagonally dominant is also considered and a test to see if a particular matrix can be scaled to be diagonally dominant is developed.

A means of checking the decentralised stability of systems that do not have any of the structures so far considered is now discussed in detail. The method involves generating an interaction measure, the Perron-Frobenius eigenvalue of a comparison matrix. The interaction measure gives the designer an indication as to the suitability of the present matrix structure for distributed controller design. A method is then introduced that aids the designer in selecting matrix structures that are likely to have low values of the interaction measure.

The software developed to evaluate the theory is now briefly discussed. The techniques developed to analyse the system for block triangular or block diagonal structures provide the designer of large scale decentralised control systems with a powerful tool. They are simple to use and give valuable information as to the stability of the system under decentralised control. The method developed to detect dominant elements in the system matrix is also useful and gives the designer a graphical display of the relative dominances of the various matrix elements over a frequency range selected by the user. The Perron-Frobenius eigenvalue is shown to be an effective interaction measure. The method developed to select matrix structures likely to give low values of this interaction measure is not easy to use and may give results that are difficult to interpret. However there is currently no other method of rearranging the system matrix to minimise the interaction measure.

TABLE OF CONTENTS

CHAPTER 3 SIMPLE STRUCTURE ANALYSIS

CHAPTER 4 THE DIAGONAL DOMINANCE STRUCTURE

LIST OF ILLUSTRATIONS

NOMENCLATURE

A list of symbols used and their meanings is given in this section. Note that matrices are represented by uppercase characters while vectors and scalars are represented by lower case characters.

A matrix element is represented by the lowercase character corresponding to the uppercase character of the matrix with a pair of subscripted numbers that give the row and column numbers of the element. An exception to this is when the elements are themselves matrices in which case they are represented by uppercase letters. Generally the subscript i is used to denote the row number and the subscript j used for the column number. On occasion other subscripts may be used to represent the rows and columns but these are obvious in context. If the matrix is diagonal then the elements on the diagonal may each be labeled with a single subscript since in this case the row and column numbers will be identical.

The vectors are all column vectors unless otherwise stated.

SYMBOL	MEANING
BDD	Block Diagonal Dominance
BIM	Binary Interaction Matrix: a matrix whose elements are one or zero depending on whether the corresponding element in G(s) is non-zero or zero
C(s)	A comparison matrix associated with the concept of Quasi-Block Diagonal Dominance
CS	Column Subsystem
D(s)	A diagonal matrix associated with QBDD and r(C(s))
djmax(s)	Maximum element of D(s) matrix

$e(s)$	A vector of the differences between the desired outputs and the actual outputs
D	The Nyquist contour D
$diag(a_i)$	A diagonal matrix with the elements a_1, $a_2,\ldots,a_n$ on the diagonal.
E	Matrix used to determine if G(s) can be rearranged to be block diagonal
F	Matrix of feedback gains, usually diagonal
$G(s)$	Input-output matrix model of a multivariable system
$g_{ij}(s)$	The element of the matrix G(s) in the ith row and jth column
GBDD	Generalised Block Diagonal Dominance
GDD	Generalised Diagonal Dominance
$H(s)$	Matrix of transfer functions representing the closed loop system
I	Identity matrix
INA	Inverse Nyquist Array
$K(s)$	Matrix representing a multivariable controller
LTI	Linear time invariant
$M(G)$	A comparison matrix associated with GDD, this should not be confused with M(s).
$M_b(G)$	Block comparison matrix associated with GBDD
$M(s)$	A matrix of the elements of G(s) normalised by their row sums
$N(s)$	A matrix of the elements of G(s) normalised by their column sums
p_0	Number of poles in the right half s plane
$Q(s)$	Open loop transfer function of the controlled system

QBDD	Quasi-Block Diagonal Dominance
r(s)	A vector of setpoints (desired values) for the outputs of the system
r(C(s))	Perron-Frobenius eigenvalue of C(s) an interaction measure
r(s)	The Perron-Frobenius eigenvalue of a matrix. This is the same symbol as for the setpoints of G(s) but the context makes it clear as to which definition the symbol refers.
S	A matrix of scaling elements (usually diagonal)
s	Laplace variable
u(s)	vector of inputs to a system
w	angular frequency in radians
y(s)	vector of outputs from a system
Z	A matrix whose elements are either 1, -1 or 0 depending on the values of the elements in the corresponding M(s) or N(s) matrix
$\|a\|$	The absolute value of a scalar or vector a
$\|\|A\|\|$	The norm of a matrix A
β	A lower bound used to determine the values of the Z matrix
Γ_i	The contour in the s plane onto which an element q_{ii} is mapped by D
$\Omega(G)$	A diagonal matrix associated with GDD
$\Omega_b(G)$	A diagonal matrix associated with GBDD
μ	An interaction measure
ϕ_o	Characteristic function of the open loop system
ϕ_c	Characteristic equation of the closed loop system

$\delta^{i}{}_{r}$	Sum of the elements in row i of G(s)
δ^{jc}	Sum of the elements in column j of G(s)

CHAPTER 1

INTRODUCTION

1.1 DECENTRALISED VS CENTRALISED CONTROL

The application of control theory to large industrial plants may be approached in one of two fundamental ways. The designer may treat the entire plant as a single system and apply control theory to design a controller for this system. Alternatively the designer may try to divide the plant up into a number of subsystems that will then be controlled independently.

The first approach leads to a centralised control scheme. Control is implemented by a central computer into which are fed measurements of the plant outputs. A control algorithm calculates the control signals that are then transmitted to actuators situated in the plant.

The second approach in which controllers are designed for different subsections of the plant is known as decentralised control. The different subsections of the plant are controlled independently from one another. In this case individual controllers would be used for each subsystem, with the possibility of each controller being located in the section of the plant that it controls.

A number of different reasons for preferring distributed controllers to centralised ones are given in the literature. In [10] the authors suggest that 'decentralised' controllers are desirable because they result in controllers with fewer tuning parameters than centralised, non-distributed, schemes. Another advantage that is cited in [8], [9] and

[11] is the simplification and cost reduction of hardware requirements. In many cases processes can be separated by large distances and the cost of cabling can be significantly reduced by implementing a distributed control system. Kramer, Magee and Scoman in [9] also cite improved response time by locating computers close to the processes (or groups of processes) that they are controlling, ease of extension and modification by adding/removing stations and communications links and increased performance by exploiting parallelism. As an example of the distances associated with some processes, and hence the associated cabling costs, the processes in [8] were a maximum of 1.8 km apart and in [9], a coal mining application, they were up to 20 km apart. It should be noted however that as Grosdidier and Morari point out in [11], hardware issues are irrelevant in many process control situations since from safety requirements the plant is monitored and controlled from a central control room and this would apply whether the control was central or distributed. In general it would seem that the advantages of reduced hardware requirements in distributed control are limited to those applications in which the processes are separated by considerable distances.

Another reason for preferring distributed control that is often given, is that such systems are often more robust than centrally controlled systems. In [9] the authors suggest that such systems will, in general, have increased availability, since the effect of physical faults such as processor failure are confined to one control station. Reference [16] also cites improved safety and fault tolerance as advantages that accrue from distributed control. When control is distributed among many different, almost isolated, loops should one loop fail the rest of the plant need not necessarily stop functioning while the fault is corrected.

Another, important, benefit of using a distributed control scheme is that the control problem can be broken down into a number of subproblems that can then be easily solved using existing controller design techniques such as Rosenbrock's INA method [12]. This avoids the necessity of applying such methods to the full plant. In [27] for example the authors say that as yet attempts to implement existing control theory to large systems has not been successful. The control theory underlying the design techniques that have been applied to small scale systems , with only a few inputs and outputs, can be applied directly to large scale systems. Unfortunately these techniques are limited by computational and representational problems. For interactive design methods such as INA it becomes difficult to display information in a meaningful way. Another problem is that the memory storage requirements placed on the computer being used in the design exercise may be excessive. Finally the computation time increases with the size of the system and may become unreasonably long.

1.2 REPRESENTATION OF THE SYSTEM

In order to design distributed controllers the designer must be able to represent the system by a mathematical model. There are two widely used methods of modelling dynamic systems. The first is to use differential equations in the time domain, the state space representation. The second representation is the input-output representation that uses Laplacian transfer functions to model the processes of the system in the s domain. The latter representation is used throughout this work.

The plant is represented by a system matrix G(s). This matrix relates a vector of inputs, u(s), to a vector of outputs, y(s), according to the equation

$$y(s) = G(s)u(s) \tag{1.1}$$

The controller is represented by a matrix, K(s). A typical feedback system is shown in Fig. 1.1. Here r(s) represents the setpoints, e(s) the error signals, u(s) represents the inputs to the system and y(s) represents the outputs.

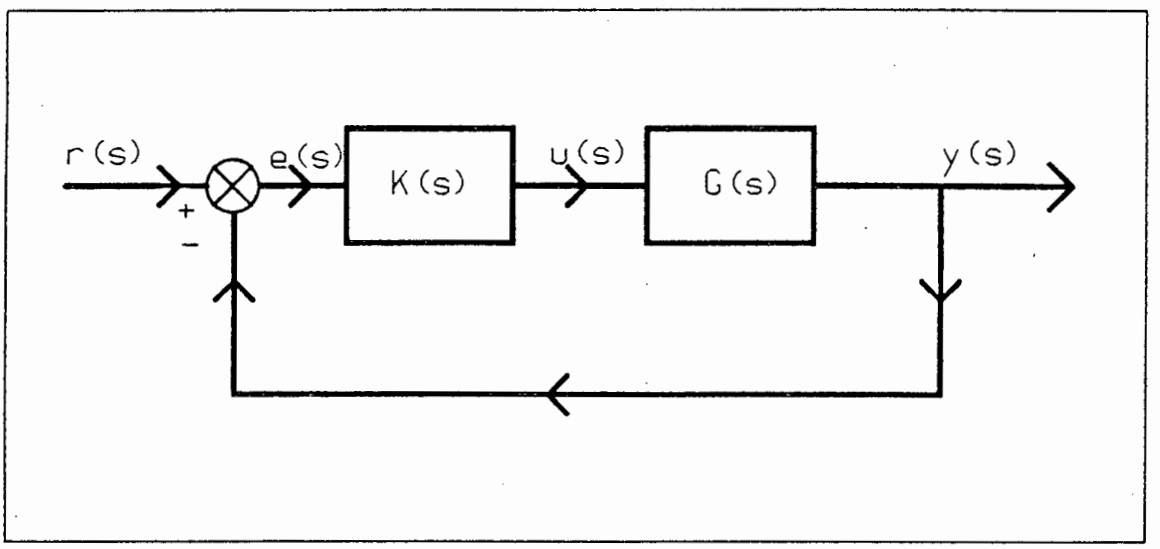

Figure 1.1. Block diagram of a feedback control system

When a distributed control system is designed the control engineer selects certain subsystems and controls these independently from one another. In terms of the matrix G(s) this means partitioning G(s) so that the subsystems chosen to be controlled are represented by the submatrices that fall on the diagonal of the partitioned system. The designer then uses each submatrix as a system model and designs a controller for the system.

For example consider a process modelled by the following 3x3 system matrix.

$$G = \left[\begin{array}{c|cc} g_{11} & g_{12} & g_{13} \\ \hline g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{array}\right] \qquad (1.2)$$

The designer has partitioned this system as shown. The matrix G can be represented as a 2x2 matrix where each element of this matrix is itself a submatrix. This new matrix is known as a composite matrix. The designer will now design controllers for the two systems G_1 and G_2 defined in (1.3) and (1.4).

$$G_1 = g_{11} \qquad (1.3)$$

$$G_2 = \begin{bmatrix} g_{22} & g_{23} \\ g_{32} & g_{33} \end{bmatrix} \qquad (1.4)$$

The final controller will have the form shown in (1.5)

$$K = \left[\begin{array}{c|cc} k_{11} & 0 & 0 \\ \hline 0 & k_{22} & k_{23} \\ 0 & k_{32} & k_{33} \end{array}\right] \qquad (1.5)$$

Note that some of the k_{ij} elements in the 2x2 submatrix may be zero.

This particular matrix structure is known as a block diagonal structure and will be encountered throughout this work.

1.3 THE STABILITY OF DISTRIBUTED CONTROL SYSTEMS

So far the assumption has been that adequate control can be obtained by designing controllers independently for different subsections of the plant. In fact distributed control cannot always be implemented. It may be impossible to partition the plant in such a way that the the plant as a whole will be stable when the on-diagonal subsystems are stabilised. This comes about because the subsystems that the designer has chosen to control will usually be coupled to one another. The control of one subsystem will effect the others and visa versa. Hence there is a need for stability theorems that will tell the designer whether the distributed control system will be stable or not. A number of theorems have been developed by various workers in the distributed control field and these are examined and utilised in this thesis.

The stability problem then, is to define the conditions necessary such that stabilising the on-diagonal subsystems independently will ensure that the system as a whole is stable. This is usually done by defining a matrix structure that has special properties. If the composite matrix modelling the controlled system has such a structure then, subject to various assumptions, the system as a whole will be stable.

1.4 MATRIX STRUCTURE

The structure of a matrix is dependent on the way in which the rows and columns are ordered and the partitioning used. An example of a matrix structure is the block diagonal structure already encountered. There are several matrix structures that give the designer information about the stability of the closed loop system. These structures can be broken into two classes. The first class of matrix

structures has been called 'simple structures' in this thesis. These structures depend only on the distribution of the zero and nonzero elements in the matrix. The second class of structures are those that have a more complicated basis. They usually rely heavily on the theory of matrix norms.

From the above discussion it is clear that the selection of the subsystems to be independently controlled is a critical part of distributed controller design. The structure of the matrix and hence its stability will depend on which elements are included in the subsystems to be independently controlled.

While stability theorems exist for decentralised systems there is as yet no systematic way of selecting the row and column arrangements and the partitionings that are likely to give stable decentralised control, [11]. To try every possible combination of row-column arrangement and partitioning is impossible for even a moderately sized plant. This can be demonstrated very simply. For a 100x100 plant there are 100! ways of arranging the rows and 100! ways of arranging the columns. Hence the total number of row-column arrangements is $(100!)^2$. This must then be multiplied by 2^{99} which is the total number of partitionings possible, while ensuring that the on-diagonal submatrices are square. The final number of combinations is $(100!)^2 x 2^{99}$, which is a very large number.

In this thesis methods are developed to select appropriate subsystems for distributed control. This involves the rearrangement of rows and columns and the use of different partitioning in a systematic manner to obtain, if possible, one of the structures that will ensure the stability of the controlled system. Both simple structures and more complex structures are considered. These methods are closely coupled

to a number of stability theorems taken from the literature.

1.5 LAYOUT

The thesis is structured as follows. In chapter two there is a review of the literature concerned with the design of decentralised controllers. This review is also intended to present the mathematical background necessary for this thesis. In chapter three the simple structures are defined and their use in analysing the system is shown. Algorithms are developed that enable the designer to quickly discover whether or not the matrix with which he is working can be rearranged to have a useful simple structure. In chapter four the diagonal dominance structure is considered. A method of rearranging a matrix to be either row or column diagonally dominant is presented. The scaling of a matrix to achieve diagonal dominance is discussed and a method of checking a matrix to see if it can be scaled to be diagonally dominant is also presented. In chapter five the theory behind an interaction measure taken from the literature is discussed. This interaction measure is then used as a measure of the suitability of a matrix structure for distributed control. A method of selecting row-column arrangements and partitionings is then presented. Chapter six gives a description of the software developed to investigate the theory including a multivariable simulator and a program to calculate the interaction measure of chapter five. Finally chapter seven is the conclusion in which the work performed in thesis is reviewed and commented on.

CHAPTER 2

REVIEW OF THEORY AND MATHEMATICAL THEOREMS

2.1 SCOPE AND LIMITATIONS

This survey of the literature is the result of a practical need to determine what research has been done in the field of decentralised controller design for large scale systems. Because the survey is motivated by a particular project, it covers work relevant to that project and is not intended to be an exhaustive review of all the available literature in the field.

This thesis has concentrated on the problem of partitioning a system into subsystems, investigating the criteria on which such partitioning should be made and the subsequent performance of the system under decentralised control. The stability of such systems is of particular importance. In fact very few papers deal with practical methods of choosing the partitioning for the system , the emphasis being on checking the stability of an already partitioned system where each controlled sub-system has been separately stabilised. As a result of this most of the papers reviewed here are concerned with stability.

Another restriction on this survey imposed by the bias of the thesis is that of system representation. In control theory papers two main representations are used, the state space and the input/output forms. The representation used in this thesis has been the input/output form, this is reflected in the survey with all of the papers considered using this form of representation.

Only a limited number of journals were searched for available literature. The journals considered were those readily available at U.C.T. although in some cases particularly relevant papers were obtained through the interlibrary loan service. Some relevant journals were not available in South Africa.

For those journals available the literature search was further limited to post 1979 issues, although some pre 1980 papers, referenced in those searched were also located. This was done because it was felt that pre 1980 work would be outdated or included in material written after that date.

2.2 PARTITIONING THE SYSTEM ON THE BASIS OF STRUCTURE

Once the desirability of decentralised control is recognised the problem of designing such systems arises. The control engineer will design controllers for subsections of the plant independently and ignore any interaction that may exist between these different sections. Mathematically this means that, for an input-output representation, the controller matrix will be block diagonal. The individual controllers for each subsection of the plant are required to stabilise the relevant subsection and to produce desired performance characteristics. The designer also requires that stabilisation of the individual subsystems will ensures that the plant as a whole is closed loop stable. In order to ensure this the subsystems that are to be controlled individually must be carefully chosen.

The subsystems to be controlled are represented by the on-diagonal blocks of the partitioned system matrix of transfer functions, if an input/output representation is used. The relationship between the different on-diagonal blocks of the partitioned matrix is known as the structure of the matrix.

Clearly the structure of a particular matrix depends on the row/column ordering used and on the way the matrix is partitioned. Further a number of different relationships between the blocks of the matrix can be defined for the same matrix as can a number of different structures. The use of the structure of a matrix provides the control engineer with a powerful tool in designing decentralised systems. This is because some matrix structures give information about the closed loop stability of the composite matrix under decentralised control. In other words if the system matrix, or the controlled system matrix, has a particular structure then the designer can tell if the system as a whole will be stable under composite control.

2.2.1 Simple Structures

The simplest structures of a matrix relate to the distribution of zero and non-zero elements throughout the system matrix. The sizes of the individual non-zero elements are not considered when the matrix structure is analysed on this basis. One way of representing the matrix so as to make this form of structure apparent is to use its corresponding adjacency matrix, [7], or binary interaction matrix, [17]. Structures based on the above will be referred to as 'simple structures' in this thesis.

There are essentially three simple structures of interest to the control engineer. The first is a block diagonal structure. This is shown in Fig. 2.1 for a matrix partitioned into a 2x2 composite matrix (a composite matrix is a matrix whose elements are submatrices). The two forms given are equivalent since one can easily be transformed into the other by row-column rearrangement. Note that the blocks can be of any order including 1x1 provided that the on-diagonal blocks are square. The off-diagonal blocks do not have to be square. In the the block diagonal case the

on-diagonal blocks are not coupled and hence controllers can be designed for each block independently with no interaction occurring between the separately controlled subsystems.

$$\begin{bmatrix} G_{11}(s) & 0 \\ 0 & G_{22}(s) \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 0 & G_{21}(s) \\ G_{21}(s) & 0 \end{bmatrix}$$

Figure 2.1. Examples of 2x2 block diagonal matrices.

The second structure of interest is the one shown in Fig. 2.2. This is a full structure. In this structure there are no zero blocks, and all the on-diagonal subsystems are coupled. Other more complicated structures are required to analyse the composite stability of such systems.

$$\begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix}$$

Figure 2.2. A 2x2 ,'full',block matrix.

The final useful structure to be considered is the one shown in Fig. 2.3. This is a block triangular structure and it is the most interesting of the three simple structures. As with the block diagonal structure there are a number of eqivalent forms, these can be made by locating the zero block in each of the four corners of the matrix, in turn. The reason for this is that such structures can often be found in real applications by suitable input/output variable pairing and matrix partitioning.

$$\begin{bmatrix} G_{11}(s) & 0 \\ G_{21}(s) & G_{22}(s) \end{bmatrix}$$

Figure 2.3. A 2x2 block triangular matrix.

Of considerable interest is the stability of composite systems with triangular structure. The stability problem is considered in detail in a paper by Callier and co-workers in [7]. In this paper they develop theorems that state the necessary conditions for closed loop stability of a system with a block triangular structure.

In [7] the authors consider both non-linear time varying operator dynamics and linear time invariant (LTI) transfer function dynamics, the latter being of interest in this thesis. The stability of a LTI lumped system is characterised by the authors as follows.

A LTI lumped system described by its closed loop transfer function H(s) is said to be exponentially stable if and only if H(s) is bounded at infinity and H(s) has all its poles in the open left half plane. When stability is mentioned below it refers to this definition.

The system considered is shown in Fig. 2.4.

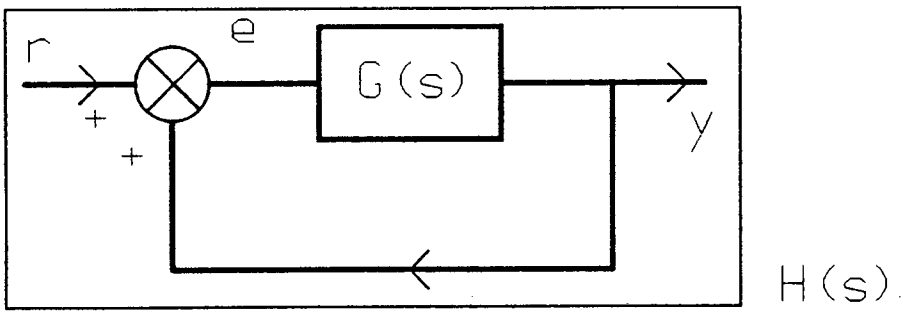

Figure 2.4. Block diagram of a simple feedback system.

Note that the authors derive their results for a positive feedback system where as it is more usual to work with negative feedback.

The relationship between the error, e, the setpoint, r, and the output, y, is described in (2.1)

$$e = r + y \tag{2.1}$$

i.e.

$$e = r + G.e \tag{2.2}$$

or

$$r = (I-G).e \tag{2.3}$$

in matrix notation

$$\begin{bmatrix} I-G_{11} & -G_{12} \cdots\cdots & -G_{1m} \\ -G_{21} & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ -G_{m1} \cdots\cdots\cdots\cdots & & I-G_{mm} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \cdot \\ \cdot \\ e_m \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \cdot \\ \cdot \\ r_m \end{bmatrix} \tag{2.4}$$

Theorem 2.1

(a) If G_{ij} is stable, for all $i \neq j$, and (b) for every unstable G_{ii}, $(I-G_{ii})^{-1}$ is stable and (c) $(I-G_{ii})^{-1}$ is stable then the overall system is stable.

This theorem requires that all the G_{ij}'s, $i \neq j$, to be stable. Note that the usual definition of e is

$$e = r-y \tag{2.5}$$

then $y = G(I+G)^{-1}.r$, or

$$r = (I+G)e \tag{2.6}$$

and the stability of the system depends on $(I+G)^{-1}$ having no zeros in the right half plane. In the discussion of [7] the convention adopted here is that used by the authors in the paper.

A system in triangular form is now considered.

$$\begin{bmatrix} I-G_{11} & 0 & \cdots\cdots & 0 \\ -G_{21} & I-G_{22} & & 0. \\ . & & . & . \\ . & & & 0 \\ -G_{\mu 1} & \cdots\cdots\cdots\cdots & & I-G_{\mu\mu} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ . \\ . \\ e_\mu \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ . \\ . \\ r_\mu \end{bmatrix} \qquad (2.7)$$

In (2.6) the G_{ij} terms are submatrices and if i=j the submatrix must be square..

The system is now divided into a hierarchy of column subsystems, CS.

For an $n_i x n_j$ subsystem G_{ij} let $D(G_{ij})$ represent the $(n_i.n_j)xn_j$ diagonal matrix made up of the columns of G_{ij}.

e.g. for a 1x2 submatrix

$$G_{ij} = \begin{bmatrix} g_{12} & g_{13} \end{bmatrix} \quad D(G_{ij}) = \begin{bmatrix} g_{12} & 0 \\ 0 & g_{13} \end{bmatrix}$$

for a 2x2 submatrix

$$G_{ij} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} \quad D(G_{ij}) = \begin{bmatrix} g_{11} & 0 \\ g_{21} & 0 \\ 0 & g_{12} \\ 0 & g_{22} \end{bmatrix}$$

The CS is then described by the elements,

$$D(G_{ij})(I-G_{jj})^{-1} \quad i=j,\ j+1,\ \ldots\ldots\mu \qquad (2.8)$$

Theorem 2.2

A CS is said to be stable if and only if $D(G_{ij}).(I-G_{jj})^{-1}$ for each G_{jj} in the CS is stable, where i=j, j+1,..., μ.

Theorem 2.3

Given the above definitions the overall system is stable iff all the CS are stable.

To illustrate this consider a 3x3 matrix partitioned as shown.

$$\left[\begin{array}{c|cc} g_{11} & 0 & 0 \\ \hline g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{array}\right] = \left[\begin{array}{c|c} G_{11} & 0 \\ \hline G_{21} & G_{22} \end{array}\right]$$

CS_1 is composed of

$$D(G_{11})(I-G_{11})^{-1}$$
$$D(G_{21})(I-G_{11})^{-1}$$

CS_2 is composed of

$$D(G_{22})(I-G_{22})^{-1}$$

where $D(G_{11})(I-G_{11})^{-1} = g_{11}(1-g_{11})^{-1}$

and $D(G_{21})(I-G_{11})^{-1} = \begin{bmatrix} g_{21} \\ g_{31} \end{bmatrix}.(1-g_{11})^{-1}$

$$D(G_{22})(I-G_{22})^{-1} = \begin{bmatrix} g_{22} & 0 \\ g_{32} & 0 \\ 0 & g_{23} \\ 0 & g_{33} \end{bmatrix} \begin{bmatrix} 1-g_{22} & -g_{23} \\ -g_{32} & 1-g_{33} \end{bmatrix}^{-1}$$

If all of these systems are stable then the system as a whole will be closed loop stable. This enables the designer to check the stablity of the triangular system relatively easily and quickly.

In [7] it is noted that if any of the elements of the off diagonal block are unstable and the system as a whole is closed loop stable then pole zero cancellation has taken place and, as the authors point out, this will probably lead to a non-robust controller. Hence for robust control they suggest that each $D(G_{ij})$ $i \neq j$ be stable i.e. all the elements in the off diagonal blocks are stable.

The detailed treatment of the stability results above may suggest that checking a triangular system for stability is a complex task. In practice however since pole zero cancellation is undesirable the designer must ensure that all the unstable elements in the matrix occur in the on-diagonal blocks. If this is done then the stability problem is vastly simplified. Under these conditions the matrix will be closed loop stable provided the on-diagonal blocks are closed loop stable and the complex stability results given above do not have to be employed.

2.2.2 Diagonal Dominance

The first 'non-simple structure' considered is diagonal dominance. This structure is probably the best known since it forms the basis for the well known INA design technique developed by Rosenbrock. The discussion that follows is based on Rosenbrock's book, [12].

An nxn matrix, G, is said to be row diagonally dominant if

$$|g_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |g_{ij}| \qquad \text{for all } i=1,2,\ldots,n \tag{2.9}$$

It is said to be column diagonally dominant if

$$|g_{jj}| > \sum_{\substack{i=1 \\ i\neq j}}^{n} |g_{ij}| \qquad \text{for all } j=1,2,\ldots,n \tag{2.10}$$

The matrix is said to be diagonally dominant if it is either row or column diagonally dominant. If G is a matrix of transfer functions, i.e. G=G(s), then the diagonal dominance condition must hold for all s on the Nyquist contour D.

The usefulness of diagonal dominance is that it allows a simple graphical stability test to be applied to a multivariable system, the test is a multivariable extension of the Nyquist stability criterion for single variable systems. In [12] Rosenbrock proves the following stability theorem for the system shown in Fig. 2.5.

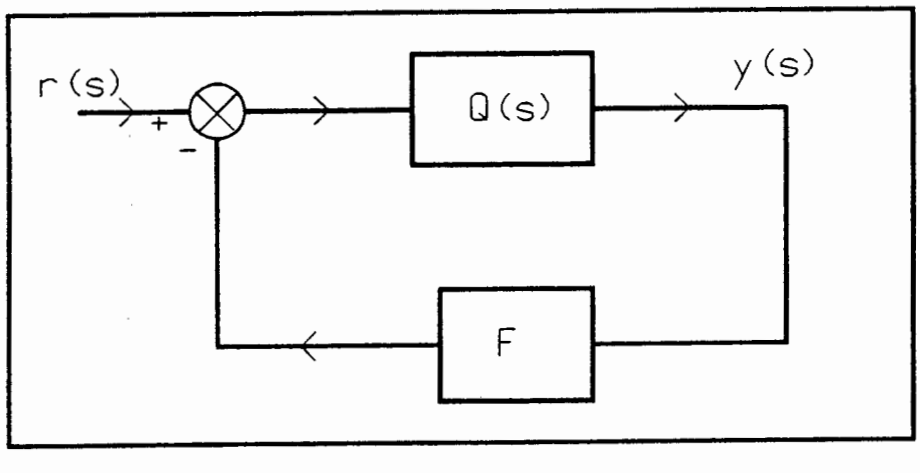

Figure 2.5. A general multivariable control system.

where Q(s)=G(s)K(s) and F is a diagonal matrix of frequency independent loop gains.

Theorem 2.4
The closed loop system is stable if and only if the map of D (the Nyquist contour) by det(I+QF) encircles the origin $-p_o$ times clockwise, where p_o is the number of poles in the right half s plane and D is taken large enough to enclose all finite poles and zeros of det(I+QF) lying in the RHP.

i.e. if N′ = number of encirclements by det(I+QF) then $N' = -p_o$.

Problems with this formulation are that :

(i) determinants are difficult to calculate accurately with existing numerical techniques.

(ii) it is not clear how changing the transfer functions in one loop, using a controller, will affect the stability of the overall system. i.e. it is not possible to design compensators separately for each loop.

These problems can be overcome if the matrix is diagonally dominant. Rosenbrock proves the following theorem.

Theorem 2.5
Let $F = \mathrm{diag}(f_i)$, where the f_i are real and nonzero, and let $[F^{-1} + Q]$ be dominant on D. Let q_{ii} map D onto Γ_i which

encircles the point $(-f_i,0)$, N_i times, $i = 1,2,\ldots,n$. Then the closed-loop system is asymptotically stable if and only if

$$\sum_{i=1}^{n} N_i = -p_o$$

where p_o is the same as in the previous theorem.

Rosenbrock also derives a similar stability criterion for the inverse of Q, Q^{-1}. This is done because the use of the inverse offers a number of advantages over the original matrix notably in the behavior of the so called Ostrowski bands that are used to locate the inverse closed loop transfer functions h_i^{-1} and to determine the stability margins of the loops. Here however only the use of the non-inverted Q will be considered.

Note that $h_i(s)$ is the transfer function relating the ith input to the ith output with the ith feedback loop open and all the other feedback loops closed.

The stability test given above can only be used if the matrix is diagonally dominant. There exists a simple graphical technique to test for diagonal dominance that can be displayed on the same Nyquist diagrams being used to check for encirclements. The method proceeds as follows for row dominance. For a particular value of s on D and with the corresponding point $g_{ii}(s)$ on Γ_i as the center draw a circle of radius

$$d_i(s) = \sum_{\substack{j=1 \\ j\neq i}}^{n} |g_{ij}(s)| \qquad (2.11)$$

This process is repeated as s goes around D to produce a band of circles. These circles are called Gershgorin circles because their use depends on applying Gershgorin's theorem. If the band swept out by these circles does not include the

origin for all s for i=1,2,...,n then the matrix is row diagonally dominant. The corresponding circles to test for column dominance have the same centre with radius

$$d_i(s) = \sum_{\substack{j=1 \\ j \neq i}}^{n} |g_{ji}(s)| \qquad (2.12)$$

Once the Gershgorin circles have been plotted the following graphical interpretation of the previous theorem can be made.

Theorem 2.6
Let each of the Gershgorin bands swept out by the circles based on q_{ii} exclude the the point $(-f_i^{-1},0)$ i=1,2,...,n. Let these bands encircle the point $(-f_i^{-1},0)$, N_i times, i=1,2,...,n. Then the closed loop system is asymptotically stable if and only if

$$\sum_{i=1}^{n} N_i = -p_o$$

The above theorem allows the designer to design controllers individually for each loop using the Gershgorin bands as though they were 'fuzzy' Nyquist plots.

A number of points are worth making about the use of the diagonal dominance structure in the design of decentralised control systems.

(i) This structure forms the basis for one of the most successful frequency domain design techniques yet developed. However very few large systems can be made diagonally dominant by appropriate ordering of input-output variables. In practice compensators must be introduced to make the system diagonally dominant. If

the compensator matrix is diagonal then it can be incorporated as part of the diagonal decentralised controller matrix. If the compensator matrix is not diagonal however the final controller incorporating this compensator will not be diagonal and control will not be truly decentralised.

(ii) It is not always possible to make the system matrix diagonally dominant by using a diagonal compensator. In this case the diagonal dominance structure cannot be used as a basis for decentralised control.

(iii) If the designer is trying to decentralise control then the use of diagonal dominance limits the controller matrix to being a diagonal matrix, i.e. to having non-zero elements only on the diagonals. However if the controller matrix is block diagonal or block triangular then this represents a distributed controller where the subsystems controlled are larger than 1x1. The diagonal dominance structure is of no help in designing such controllers.

Diagonal dominance is a very useful structure in the design of decentralised controllers but is seldom found in large systems. Other structures are necessary if decentralised controllers with blocks larger than 1x1 are to be designed.

2.2.3 Generalised Diagonal Dominance

An extension of the diagonal dominance structure is generalised diagonal dominance. This structure, like diagonal dominance limits the designer to a diagonal controller, as opposed to block diagonal, but is less conservative than diagonal dominance. Further the idea of generalised dominance can be readily extended to blocked systems.

The concept of generalised diagonal dominance is presented by Limebeer in [1]. Generalised diagonal dominance, GDD, is defined as follows. A matrix G with complex elements, g_{ij}, is said to be row GDD if there exists a vector of real numbers $x > 0$ such that

$$|g_{ii}|x_i > \sum_{\substack{j=1 \\ j\neq i}}^{n} |g_{ij}|x_j \qquad \text{for all } i=1,2,\ldots,n \qquad (2.13)$$

A matrix is column GDD if a similar x exists and

$$|g_{jj}|x_j > \sum_{\substack{i=1 \\ i\neq j}}^{n} |g_{ij}|x_i \qquad \text{for all } j=1,2,\ldots,n \qquad (2.14)$$

This implies for row GDD, for example, that

$$|g_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |g_{ji}|\frac{x_j}{x_i} \qquad \text{for all } i=1,2,\ldots,n \qquad (2.15)$$

This is equivalent to scaling G with two diagonal matrices, S and S^{-1} where $S = \text{diag}(x_1 \ldots . x_n)$ and $S^{-1} = (1/x_1 \ldots . 1/x_2)$ and the scaled matrix is S^{-1} G S.

Another way of expressing this is to say that if a matrix G can be scaled as shown above to be diagonally dominant then it is said to be generalised diagonally dominant. The eigenvalues of G are not affected by this scaling, [1], the scaled matrix has the same eigenvalues as the original matrix, hence if the scaled matrix is stable the unscaled matrix will also be stable. Notice that from the point of view of stability the actual values of the elements of S are not important only the existence of such matrices for each value of s on the Nyquist contour D matters.

Limebeer in [1] presents a stability theorem that is an extension of one developed by Rosenbrock in [12].

Theorem 2.6

For the system in Fig.2.6, let $F = \mathrm{diag}\{f_i\}$, where the f_i terms are real and nonzero. Let $F^{-1} + G(s)$ be generalised diagonally dominant for all s on the standard Nyquist contour D. Let $g_{ii}(s)$ map D onto Γ_i which encircles the point $(-f_i^{-1},0)$, N_i times, i=1,2,..,n. Then the closed loop system is asymptotically stable if and only if

$$\sum_{i=1}^{n} N_i = -p_o$$

where p_o is the number of poles of G(s) in the right half of the s plane.

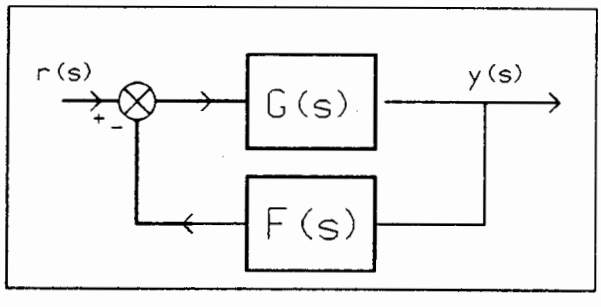

Figure 2.6 A simple feedback system

In [1] Limebeer derives a test for GDD. For a matrix G he associates a comparison matrix $M(G) = m_{ij}$ which is defined as

$$m_{ij} = |g_{ij}| \quad \text{for } i,j = 1,2,\ldots,n \tag{2.16}$$

and a normalising matrix

$$\Omega(G) = \mathrm{diag}\{m_{ii}\} \tag{2.17}$$

Limebeer then proves the following theorem

Theorem 2.8
For an irreducible matrix G the following are equivalent:

(a) $\Omega^{-1}(G)M(G)$ **has Perron-Frobenius eigenvalue** $r < 2$.
(b) G is row GDD
(c) There exists a diagonal $S > 0$ **such that** $S^{-1}GS$ **is row dominant.**
(d) G is column GDD.
(e) There exists a diagonal $S' > 0$ **such that** $S'^{-1}GS'$ **is** *column* **dominant. (Where S′ denotes a different S to that found in (c).**

The Perron-Frobenius eigenvalue is defined in the theorem by Perron and Frobenius on the spectral properties of non-negative matrices, [14]. This theorem is used to define other structures of interest to decentralised control and is stated here.

Theorem 2.8
Let G be non-negative and irreducible (see below for definitions of these properties). Then there exists an eigenvalue r (called the Perron-Frobenius eigenvalue) such that :

(a) r is real and $r > 0$;
(b) with r can be associated positive left and right eigenvectors;
(c) $r \geq$ **the absolute value of every other eigenvalue of G;**
(d) the eigenvectors associated with r are unique to constant multipliers;
(e) if $0 \leq B \leq G$ **and** β **is an eigenvalue of B, then** $|\beta| \leq r$; **moreover,** $|\beta| = r$ **implies** $B = G$; **and,**
(f) r is a simple root of the characteristic equation of G.

In order to apply the above theorems G must be irreducible and $\Omega(G)^{-1}M(G)$ must be non-negative. The latter condition is automatically satisfied since a non-negative matrix is simply one that contains no negative elements.

An matrix is said to be reducible if there exists a permutation matrix P such that PGP^T is block triangular with square on-diagonal blocks. A matrix is irreducible if it not reducible. Limebeer presents a graph theoretic test for reducibility. If the matrix is irreducible then it is rearranged to be block triangular and the test carried out on each irreducible diagonal block separately.

Thus in order to test an irreducible matrix G for GDD the designer forms $\Omega(G)^{-1}M(G)$ and finds the largest real eigenvalue of this matrix. If the eigenvalue is greater than 2 then G is GDD. If G = G(s) then this must be true for all s on D for the system to be GDD.

In order to use the above theory Limebeer shows that generalised Gershgorin circles of radius

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} |g_{ij}(s)| \frac{x_j(s)}{x_i(s)} \qquad (2.18)$$

may be used instead of ordinary Gershgorin circles (as used by Rosenbrock in [12] for example). In [1] it is suggested that the best choice of x possible is the eigenvector that corresponds to the Perron-Frobenius eigenvalue of $\Omega(G)^{-1}M(G)$. By best choice he means that if this x is used as the diagonal of the scaling matrix S then if this S does not make $S^{-1}GS$ diagonally dominant then neither will any other S.

By assuming that x is in fact the right eigenvector of the Perron-Frobenius eigenvalue of $\Omega(G)^{-1}M(G)$ Limebeer derives a formula for the radius of the generalised Gershgorin circles

based on the Perron-Frobenius eigenvalue, r. The radius of the generalised Gershgorin circles are

$$d_i=(r(s)-1)|g_{ii}(s)| \text{ for } i=1,2,\ldots,n \qquad (2.19)$$

where r(s) is the frequency dependent Perron-Frobenius eigenvalue. Limebeer also derives a generalised version of Rosenbrock's Ostrowski circles. Design using the idea of generalised dominance then proceeds as usual using Nyquist or inverse Nyquist arrays.

The overall effect of using the generalised Gershgorin circles is to shrink the usual Gershgorin circles where these are large at the expense of expanding the circles that were small. This trade off is inevitable when scaling a matrix for dominance. If the Generalised circles all exclude the origin then the matrix is GDD.

The idea of generalised diagonal dominance does not in itself greatly improve on the usual concept of diagonal dominance as far as designing decentralised controllers. It is still unlikely that a large scale system will be dominant, even though generalised dominance is less conservative than normal dominance. However the concept of generalised dominance, i.e. the scaling of the system matrix by matrices that do not change the system eigenvalues is used in the next two structures that are to be discussed.

Finally it should be stressed that generalised diagonal dominance is a way of making the usual diagonal dominance structure less conservative. It is *not* a method for designing compensators that will improve the performance of the system, that is the matrix S is not a matrix of compensators. Clearly scaling the system by S and S^{-1} will not affect stability since the poles of the system are not changed. To scale the system in such a way as to change

stability would require a scaling such as RG(s)E where $R \neq E^{-1}$.

2.2.4 Generalised Block Diagonal Dominance

A natural extension of the idea of diagonal dominance is the concept of block diagonal dominance, BDD. In [13] Varga and Feingold introduce this concept. Essentially they are interested in exploring the structure of a partitioned matrix to establish tests for singularity and inclusion regions for the eigenvalues of a matrix.

In [13] BDD is defined as follows.

Consider a nxn complex matrix G where

$$G = \begin{bmatrix} G_{11} & \cdots & G_{1m} \\ \vdots & G_{22} & \vdots \\ \vdots & & \vdots \\ G_{m1} & & G_{mm} \end{bmatrix} \qquad (2.20)$$

here G_{ij} are block submatrices and the G_{ii} are square and of order n_i, $1 \leq n_i \leq n$.

$\|G_{ij}\|$ represents any induced matrix norm of matrix G_{ij}.

i.e.

$$\|G_{ij}\| = \sup_{x \neq 0} \frac{\|G_{ij}.x\|}{\|x\|} \qquad (2.21)$$

G is said to be row BDD iff

$$(\|G_{ii}^{-1}\|^{-1}) \geq \sum_{\substack{j=1 \\ j\neq i}}^{m} \|G_{ij}\| \quad \text{for } i=1,2,\ldots,m \qquad (2.22)$$

for a 1x1 blocking this definition reduces to the usual definition for row diagonal dominance. A similar definition exists for column BDD.

In [13] the authors show that if a matrix is BDD and block irreducible then it will also be nonsingular. They go on to use the BDD structure to define bounds for the eigenvalues of G. It is important to realise that a matrix may be BDD and *not* diagonally dominant, which is why this structure is useful for decentralised control. In [13] the following example is given to illustrate this fact.

$$G = \left[\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 2/3 & 0 & 1/3 & 0 \\ \hline 0 & 1/3 & 0 & 2/3 \\ 0 & 0 & 1 & 0 \end{array}\right] = \left[\begin{array}{c|c} G_{11} & G_{12} \\ \hline G_{21} & G_{22} \end{array}\right]$$

Using $\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$ as the matrix norm, where A is any nxn matrix, $\|G_{11}^{-1}\|_\infty^{-1} = \|G_{22}^{-1}\|_\infty^{-1} = 2/3$ and $\|G_{12}\| = \|G_{21}\| = 1/3$.

Hence G is BDD but is clearly *not* diagonally dominant in the usual sense.

In [1] Limebeer develops Varga and Feingold's work to the partitioning of systems for decentralised control. Specifically he develops a stability theory for partitioned systems.

Limebeer modifies the definition of BDD given above to a less conservative formulation. This is done by introducing the concept of generalised block diagonal dominance, GBDD, using the ideas discussed in the previous section for non-partitioned matrices. The new definition for row dominance is,

$$\|G_{ii}^{-1}\|^{-1} x_i > \sum_{\substack{j=1 \\ j\neq i}}^{m} \|G_{ij}\| x_j \quad i=1,2,\ldots,m \tag{2.23}$$

generalised column dominance is similarly defined.

As with the GDD case Limebeer gives a test for GBDD. Firstly a block comparison matrix, $M_b(G)$ is introduced where

$$M_b(G)=\{m_{ij}\}=\begin{cases} \|G_{ij}\| & \text{for } i,j=1,2,\ldots,m \quad i\neq j \\ \|G_{ii}^{-1}\|^{-1} & \text{for } i=1,2,\ldots,m \end{cases} \tag{2.24}$$

and a block normalising matrix

$$\Omega_b(G)=\text{diag}\{m_{ii}\} \text{ for } i=1,2,\ldots,m \tag{2.25}$$

Limebeer then proves the following theorem

Theorem 2.9
For the block irreducible matrix of 2.20 the following are equivalent:

(a) $\Omega_b(G)^{-1}M_b(G)$ has Perron-Frobenius eigenvalue $r_b < 2$;

(b) G is generalised row block diagonally dominant;

(c) There exists a positive matrix S = diag{$x_1 I_{n1}$, $x_2 I_{n2}$, ..., $x_m I_{nm}$} such that $S^{-1}GS$ is row block diagonally dominant.

(d) G is generalised column block diagonally dominant; and

(e) There exists a positive matrix S' = diag{$x_1' I_{n1}$, $x_2' I_{n2}$, ..., $x_m' I_{nm}$} such that $S'^{-1}GS'$ is column block diagonally dominant.

Further if one of the above equivalent conditions is met then G is non-singular.

Hence if $\Omega_b(G)^{-1}M_b(G)$ has a Perron-Frobenius eigenvalue of less than 2 then G is GBDD.

Limebeer also gives a stability test for the partitioned system.

Theorem 2.10

Let F=diag{$f_1 I_{n1}$, $f_2 I_{n2}$, ..., $f_n I_{nm}$} where the f_i's are real numbers and the I_{ni}'s are conformable with the partitioning of G(s) in Fig. 2.6. Let F^{-1} + G(s) be generalised block diagonally dominant for all s on the Nyquist D contour and let Γ_i be the Nyquist diagram of the ith diagonal block $G_{ii}(s)$ for i=1,2,...,m. Then the closed loop system of Fig. 2.6 will be stable if and only if

$$\sum_{i=1}^{m} N\,(\Gamma_i,\ -f_i^{-1}) = -p_o$$

where p_o is the number of poles of G(s) in the right half s plane, and N (Γ_i, $-f_i^{-1}$) is the number of encirclements of the point $-f_i^{-1}$ by Γ_i.

Limebeer in [1] then develops a graphical test for stability by defining the radii of circles to be plotted with center

Γ_i and to be used analogously with the usual Gershgorin circles. The circles he defines in [1] suffer from a number of drawbacks which Limebeer himself points out in [2]. In [2] he presents an alternative result using the maximum and minimum singular values of the matrix blocks. The radii thus derived are

$$d_i = \beta_i \cdot g\{(r-1)\sigma_{min}(G_{ii})/\beta_i\}, \qquad \text{if } \beta_i \neq 0$$

or (2.26)

$$d_i = (r-1)\sigma_{min}(G_{ii}) \qquad \text{if } \beta_i = 0$$

for $1 \leq i \leq m$.

(i) r = Perron-Frobenius eigenvalue of $\Omega_b(G)^{-1}M_b(G)$.

(ii) $\sigma_{min}(G_{ii})$ = minimum singular value of G_{ii}

(iii) $\beta_i = \inf\{\sigma_{min}(\text{delta}_i\}$ where delta_i is derived from the Schur decomposition of G_{ii}, i.e. $G_{ii}=U_i(D_i+\text{delta}_i)U^*_i$. delta_i is upper triangular.

(iv) $g(t)$ is the unique positive solution of the following equation

$$f(x)=\frac{x^n}{1+x+\ldots+x^{n-1}} = t$$

As can be seen this formulation lacks the computational simplicity of the Gershgorin theorem for diagonal dominance. It requires the computation of singular values and the Shur decomposition of a matrix, as well as the Perron-Frobenius eigenvalue of the partitioned G matrix. Further this procedure has to be repeated for a large number of s values to generate the block equivalent of the usual Gershgorin bands.

Limebeer and Hung in [3] and [4] go on to examine the robustness of systems shown to be stable using the stability results examined above, dealing with both additive and multiplicative perturbations.

GBDD is an improvement on generalised diagonal dominance when designing decentralised control systems. The designer is no longer required to restrict the controlled subsystems to 1x1 blocks on the diagonal of the system matrix. A system matrix may be GBDD without being diagonally dominant. Further the test for GBDD, $r(s) < 2$, is relatively simple to apply although the computational effort required is considerably more than for the case of diagonal dominance. The graphical method suggested by Limebeer is involved and computationally difficult to implement for large matrices. A final and perhaps more serious problem is that the test for GBDD, $r(s) < 2$, is dependent on the controller. This means that even if $G(s)$ is GBDD there is no guarantee that $Q(s) = G(s)K(s)$ will be GBDD. Limebeer's block equivalent of Gershgorin circles test for this but it would be useful to have a measure that indicated the suitability of a particular partitioning to decentralised control as compared to another partitioning. Such a measure is described in the next section.

2.2.5 Quasi-Block Diagonal Dominance

Quasi-Block diagonal dominance, QBDD, is defined in a manner similar to the GBDD structure described in the previous section. The fullest treatment of the use of this structure in the design of decentralised controllers for systems represented by transfer function matrices is given by Ohta, Siljak, and Matsumoto in [6]. Nwokah in [5] obtains similar results while studying the robustness of the design method

under perturbations. Nwokah does not mention QBDD explicitly but he uses the same principles as Ohta and his co-workers in deriving his results.

In [6] QBDD is defined as follows. Consider an mxm complex matrix Q decomposed to give

$$Q = Q_D + Q_C \tag{2.27}$$

where

$$Q_D = \text{diag}\{Q_1, Q_2, \ldots, Q_n\} \tag{2.28}$$

and

$$Q_C = \begin{bmatrix} 0 & Q_{12} & \cdots & Q_{1n} \\ Q_{21} & 0 & \cdots & Q_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ Q_{n1} & Q_{n2} & \cdots & 0 \end{bmatrix} \tag{2.29}$$

Note: In [6] the diagonal elements of Q_C are allowed to be nonzero to permit the representation of any uncertainties in the on-diagonal blocks of Q, i.e. in $Q_1, Q_2, \ldots, Q_n$. Here the diagonal elements of Q_C are set to zero for simplicity.

The elements Q_i are taken to be square $m_i x m_i$ submatrices of Q_D while the elements Q_{ij} are taken to be $m_i x m_j$ submatrices of Q_C, where $m = m_1 + m_2 + \ldots + m_n$. Notice that the on-diagonal submatrices of Q_D are square.

A matrix Q is said to be QBDD if the nxn matrix $W = (w_{ij})$ with elements

$$w_{ij} = \begin{cases} 1 & i=j \\ -\|Q_{ij}Q_j^{-1}\| & i\neq j \end{cases} \qquad (2.30)$$

is an M matrix.

An M matrix is a real square matrix with nonpositive off diagonal elements, i.e. zero or negative, and positive principle minors . Such matrices have a number of special properties. For example in [14] there are listed fifty different properties, such as the possession of positive principle minors. Reference [14] contains a very detailed section on M matrices and most of the results obtained by Nwokah are based on proofs from [14].

The above definition seems at first to be very different from the definition given previously for GBDD. However in [6] is shown that if W is an M matrix then the following holds.

(i) There exist positive numbers x_j, j=1, 2,...,n such that

$$x_j^{-1}\sum_{i=1}^{n} x_i\|Q_{ij}Q_j^{-1}\| < 1 \qquad (2.31)$$

(i) There exist positive numbers x_i, i=1, 2,...,n such that

$$x_i^{-1}\sum_{j=1}^{n} x_j\|Q_{ij}Q_i^{-1}\| < 1 \qquad (2.32)$$

recall that for row GBDD the inequality to be satisfied is

$$x_i\|Q_i^{-1}\|^{-1} > \sum_{\substack{j=1 \\ j\neq i}}^{n} x_j\|Q_{ij}\| \qquad (2.23)$$

this can be rewritten to give

$$x_i^{-1}\sum_{j=1}^{n} x_j \|Q_{ij}\| \|Q_i^{-1}\| < 1 \qquad (2.33)$$

which is very similar to the form given above for QBDD, a similar expression can be obtained for column GBDD. It can be seen immediately that QBDD will be less conservative than GBDD since from [14]

$$\|Q_{ij}\| \|Q_j^{-1}\| \geq \|Q_{ij}Q_j^{-1}\| \qquad (2.34)$$

Using this structure as a base [6] goes on to develop stability theorems for decentralised systems. As with GBDD the Perron-Frobenius eigenvalue plays an important part. The matrix Q is assumed to be frequency dependent and satisfies the relationship $Q(s) = G(s)K(s)$.

The following assumptions are made

(i) $K(s) = K_D = \text{diag}\{K_1, K_2, \ldots, K_n\}$ i.e. K(s) is block diagonal and is partitioned conformally with G(s).

(ii) Each on-diagonal block of G(s), $G_{jj}(s)$, is nonsingular for all j and all s on the Nyquist contour D.

(iii) All the unstable poles of Q(s) occur in the on-diagonal blocks of Q(s).

(iv) The on-diagonal submatrices of Q(s), $G_{jj}(s)K_j(s)$, are closed loop stable.

A comparison matrix $B(s)=(b_{ij}(s))$ is now defined where

$$b_{ij}(s) = \begin{cases} 0 & i=j \\ \|G_{ij}(s)K_j(s)[I_{mj}+G_{jj}(s)K_j(s)]^{-1}\| & i\neq j \end{cases} \quad (2.35)$$

Theorem 2.11
If the above conditions hold then, if the Perron-Frobenius eigenvalue of B(s), r(B(s)), is less than one for all s on D, the system as a whole will be stable.

Unfortunately the above theorem does not indicate how the controllers of the individual subsystems affect the stability of the overall system. As mentioned in the previous section on GBDD it would be helpful to have some measure of the current partitioning's suitability for decentralised control that is independent of the choice of controller. To achieve this two further matrices are defined.

$C(s) = [c_{ij}(s)]$ where

$$c_{ij}(s) = \begin{cases} 0 & i=j \\ \|G_{ij}(s)G_{jj}(s)^{-1}\| & i\neq j \end{cases} \quad (2.36)$$

and $D(s) = \mathrm{diag}\{d_1(s), d_2(s),\ldots,d_n(s)\}$ (2.37)

where

$$d_j(s) = \|G_{jj}(s)K_j(s)[I_{mj}+G_{jj}(s)K_j(s)]^{-1}\| \quad (2.38)$$

Theorem 2.12
Assume that C(s) is irreducible then if each of the previous conditions hold the system will be stable if

$$d_j(s) < r(C(s))^{-1} \text{ for all } j \text{ and all } s \text{ on } D \qquad (2.39)$$

where r(C(s)) is the Perron-Frobenius eigenvalue of C(s).

Note: The above theorem gives a sufficient but *not* necessary condition for the stability of the composite system. In other words even if $d_j(s) > r(C(s))^{-1}$ the system might still be stable. The advantage of this formulation, however, is that r(C(s)) is independent of the controller matrix used and hence can be used to give an indication of which structures of G(s) are well suited to decentralised control.

To use the theory developed so far Ohta and his co-workers propose a method of designing decentralised controllers based on the Nyquist array technique, [12].

A feedback matrix is now introduced

$$F = \text{diag}\{F_1, F_2, \ldots, F_n\} \qquad (2.40)$$

with

$$F_j = \text{diag}\{f^j_1, f^j_2, \ldots, f^j_{mj}\} \qquad (2.41)$$

where the superscript ^{j} indicates jth on-diagonal block of the composite matrix.

Theorem 2.13
Given s ∈ D. Suppose that C(s) is represented by (2.36). For a given l assume the following:

(i) $$1 + q^{ll}{}_{jj}(s) f^l{}_j(s) \neq 0 \qquad (2.42)$$

for all j, where $q^{ll}{}_{jj}(s)$ is the jjth element of $Q_{ll}(s)$ which is defined by

$$Q_{ll}(s) = G_{ll}(s) K_l(s) \qquad (2.43)$$

(ii) $K_1(s)$ is nonsingular; and

(iii) $$b^1{}_j(s) < [\{(1 + r[C(s)])\|T_1(s)\|\}-1]^{-1} \quad (2.44)$$

for all j in which $\|.\|$ is given by $\|.\|_1$ or $\|.\|_\infty$, $b^1{}_j(s)$, and $T_1(s)$ are

$$b^1{}_j(s)=|q^{11}{}_{jj}(s)f^1{}_j[1+q^{11}{}_{jj}(s)f^1{}_j]^{-1}| \quad (2.45)$$

and

$$t^1{}_j(s) = |q^{11}{}_{ij}(s)|/|q^{11}{}_{jj}(s)| \quad (2.46)$$

Then

$$d_r(s) < r(C(s))^{-1} \quad (2.47)$$

and the system is stable.

The designer once having partitioned the system can then proceed to design controllers to stabilise the individual subsystems while ensuring that condition (iii) above is met for each controlled system. If the designer achieves this then the system as a whole will also be stable. The advantage of the above formulation is that it allows the designer to check the closed loop stability of the composite system while designing each block controller independently without reference to the rest of the system. This is possible because r(C(s)) is independent of K(s) since $\|Q_{ij}(s)Q_{jj}(s)^{-1}\| = \|G_{ij}(s)K_j(s)K_j(s)^{-1}G_{jj}(s)^{-1}\| = \|G_{ij}(s)G_{jj}(s)^{-1}\|$.

The QBDD structure is the most useful of the structures so far considered for decentralised controller design. Its usefulness is that it forms the basis for an interaction measure r(C(s)) that is independent of the controller used to control the system. This allows the designer to select

subsystems that are likely to yield successful results when controlled independently. Further the method allows the designer to easily check the stability of the system as a whole provided the individual subsystems are stable. Finally the QBDD structure yields results that are less conservative than Limebeer's GBDD structure.

There are however problems with the use of the interaction measure defined above. The method is still very conservative, [10], largely because it provides a check on composite stability that is sufficient but not necessary. More will be said about this in a later section on interaction measures. Another serious problem is that the method does not suggest any means of pairing the input/output variables and selecting the partitioning to yield low values of the interaction measure, r(C(s)). Currently the only way of selecting acceptable arrangements of the system matrix rows and columns and partitionings is to try all of the possible arrangements until a suitable one is found. This is a daunting task for a large scale industrial system with several hundreds of inputs and outputs, the number of possible combinations being literally hundreds of millions.

2.3 INTERACTION MEASURES

In section 2.3.5 the QBDD structure was used to derive an interaction measure for the system matrix to determine the suitability of the matrix for decentralised control. This interaction measure, r(C(s)) gave an indication as to how tightly the on-diagonal blocks of the system matrix are coupled. A weak coupling indicates that the current partitioning is probably suited to decentralised control. The concept of an interaction matrix is an extremely useful

one since it allows the designer to represent matrix structure by a single number, or curve in the case of a matrix with frequency dependent elements.

In [10] Grosdidier and Morari give an excellent review of the interaction measures currently available and introduce a new interaction measure, the μ interaction measure, which is less conservative than any of those considered so far. Unfortunately μ turns out to be very difficult to calculate.

In order to treat the different interaction measures in a unified manner, Grosdidier and Morari introduce the concept of relative error. Consider a system as shown in Fig. 2.7

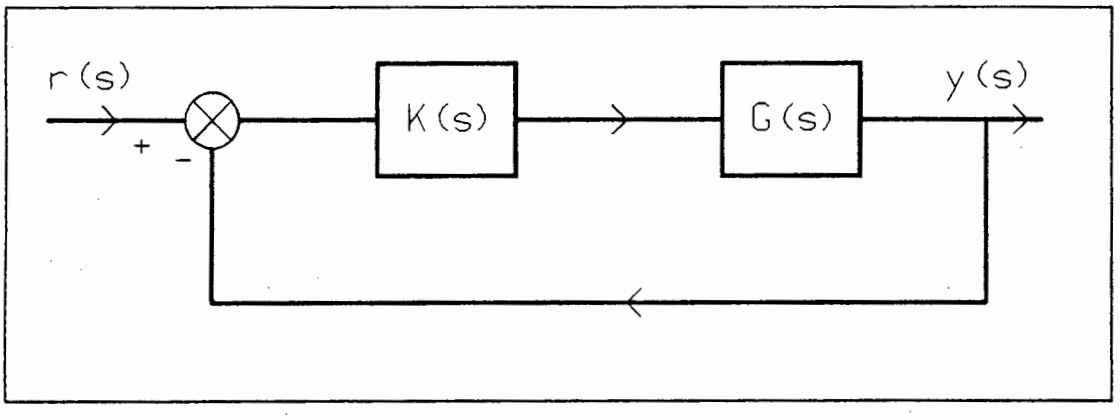

Figure 2.7. A multivariable feedback control system

The controller K(s), is to be block diagonal i.e. $K(s)=\text{diag}\{K_{11}(s), K_{22}(s),\ldots, K_{mm}(s)\}$ and is to be designed for the system

$$G_D(s) = \text{diag}\{G_{11}(s), G_{22}(s),\ldots,G_{mm}(s)\} \qquad (2.48)$$

where G(s) is partitioned into an mxm composite matrix. G(s) can be represented by

$$G(s) = G_D(s) + [G(s) - G_D(s)] \qquad (2.49)$$

$K(s)$ is designed on the assumption that $G(s) - G_D(s) = 0$.

The closed loop transfer function matrix for the diagonal system is

$$H_D(s) = G_D(s)K(s)[I + G_D(s)K(s)]^{-1} \qquad (2.50)$$

$K(s)$ is designed so that $H_D(s)$ is stable. However even if $K(s)$ does stablise $H_D(s)$ the designer must still ensure that $H(s)$ where

$$H(s) = G(s)K(s)[I + G(s)K(s)]^{-1} \qquad (2.51)$$

is also stable. An effective interaction measure must show the restrictions imposed on $H_D(s)$ such that $H(s)$ will be stable.

The relative error is defined as

$$E(s) = [G(s) - G_D(s)]G_D(s)^{-1} \qquad (2.52)$$

Initially Grosdidier and Morari consider diagonal dominance. They show that if $G_D(s)$ is diagonal then

$$\|E(s)\|_1 < 1 \qquad (2.53)$$

is equivalent to

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} |g_{ij}(s)| < |g_{jj}(s)| \text{ for all } j \text{ and all } s \in D$$

which is the definition of column diagonal dominance.

Generalised dominance is then considered. The generalised diagonal dominance condition in terms of E(s) is

$$r(|E(s)|) < 1 \tag{2.54}$$

where r is the Perron-Frobenius eigenvalue, and $|E(s)|$ represents the matrix formed by replacing all of the elements of E(s) by their norms. If the inequality is true then the matrix is generalised diagonally dominant. Grosdidier and Morari comment that both diagonal dominance and generalised diagonal dominance can be extended to block systems, as in Limebeer's work, but that the bounds on $\|H_D\|$ obtained in this way are excessively conservative and hence not very useful.

Although the r(C(s)) interaction measure derived from QBDD is not explicitly considered in [10] the following is worth noting.

The comparison matrix C(s) is defined by

$$c_{ij}(s) = \begin{cases} 0 & j=i \\ \|G_{ij}(s)G_{jj}(s)^{-1}\| & j\neq i \end{cases} \tag{2.55}$$

The elements of E(s) are defined by

$$e_{ij}(s) = \begin{cases} 0 & j=i \\ G_{ij}(s)G_{jj}(s)^{-1} & j\neq i \end{cases} \tag{2.56}$$

Hence $|E(s)| = C(s)$ this means that the interaction measure $r(C(s)) = r(|E(s)|)$. This relationship is useful when comparing r(C(s)) with the new interaction measure $\mu(E(s))$.

The interaction measure $\mu(E(s))$ is based on the idea of structured singular values originally suggested by Doyle in

[15]. The relationship between $\mu(E(s))$ and stability is as follows

Theorem 2.14
Assume that G(s) and $G_D(s)$ have the same poles in RHP and that $H_D(s)$ is stable. Then H(s) is stable if

$$\sigma_{max}(H_D(s)) < \mu(E(s))^{-1} \text{ for all } s \in D \qquad (2.57)$$

where $\sigma_{max}(H_D(s))$ is the maximum singular value of $H_D(s)$.

Note: The singular values of a matrix, A, are the non-negative square roots of the eigenvalues of the product A^TA, where A^T is the transpose of the matrix A.

The designer must therefore try to ensure that $\mu(E(s))$ is as small as possible. In particular if the controller contains integral action as is usually the case then

$$\mu(E(0)) < 1 \qquad (2.58)$$

This comes about because if the controller contains integral action

$$\lim_{s \to 0} H_D(s) = I \qquad (2.59)$$

but $\sigma_{max}(I) = 1$ hence for (2.57) to be satisfied $\mu(E(0)) < 1$.

It can be shown that

$$r(C(s)) \geq \mu(E(s)) \qquad (2.60)$$

and that $r(C(s))$ is generally a conservative upper bound for $\mu(E(s))$, [10], [11]. Hence this new interaction measure would seem to be ideal for decentralised controller design. The problem with $\mu(E(s))$ is that at present there is no way

of calculating its value for a matrix of order greater than 3. Grosdidier and Morari state that the computation of μ is an active area of research at present.

While it would seem that μ will ultimately provide a useful tool in the design of large scale decentralised control systems it is currently unfeasible due to difficulties in computation. Of the remaining interaction measures the most useful is $r(C(s))$. As seen this is more conservative than μ but it is relatively easy to calculate and at least provides an upper bound to μ.

2.4 SUMMARY

The work done in the field of decentralised controller design centers mainly on the stability of systems under this form of control. The structure of the system matrix is important in determining whether a system of independently stabilised, coupled subsystems will itself be stable.

The diagonal dominance and generalised diagonal dominance structures were shown to be useful for decentralised controller design, however they are seldom encountered in real large scale systems and any compensators used to produce dominance often remove the advantages of decentralised control.

Three block diagonal dominance structures were then considered of these Quasi-Block Diagonal Dominance, QBDD, was the least conservative and hence the most useful. The QBDD structure is useful because it gives rise to an interaction measure that can be used to determine if the current structure of the matrix is suited for decentralised control. Another interaction measure, the μ interaction measure, was also discussed. This measure was less

conservative than that used in the QBDD case but proved to be very difficult to calculate.

From the literature then the Perron-Frobenius eigenvalue of a comparison matrix C(s), r(C(s)), is the most useful interaction measure yet developed and currently offers the best indication as to the suitability of a system to decentralised control.

CHAPTER 3

SIMPLE STRUCTURE ANALYSIS

3.1 INTRODUCTION

The structure of a large scale system, i.e. the relationship between the different elements composing the system, can often be used to determine the type of control that should be applied. The control can be centralised, decentralised or some combination of the two. The plant is usually represented by a mathematical model and the structure of this model naturally reflects the structure of the plant. In the case of an input-output model the plant is represented as a matrix of transfer functions relating each plant input to a corresponding set of plant outputs.

The relationship between the different plant elements is of course fixed once the plant has been built and usually considerable effort must be made to change these relationships if so desired. However the plant structure is not always immediately discernible in the system matrix that represents the plant. This is because the ordering of the inputs and outputs as represented, by the rows and columns of the system matrix, is usually arbitrary so that rearrangement of the rows and columns may change the structure of the matrix. Further the designer may partition the matrix and this will also alter the matrix structure.

If the designer can rearrange the system matrix to get one or more defined matrix structures then he gains valuable information about the structure of the plant. In other words, the structure of the plant is, in a sense, 'hidden'

in the system matrix and by a suitable rearrangement of columns and rows and partitioning this structure can be revealed. In practice a plant may have as many structures as the designer cares to define ; however only a few structures are useful to the control engineer and it is for these structures that the engineer looks.

The particular plant structures to be considered in this chapter are a family of structures that will be referred to as 'simple structures'. These structures depend only on the relationship between the zero and non-zero elements in the system matrix and are unrelated to the sizes of the individual non-zero elements. In terms of the actual system being modelled the simple structure relates to the existence of coupling between different elements in the system, irrespective of the size of that coupling.

The simple structures are so called because they can be easily defined and do not require complicated theory to analyse. Further they can be easily represented in a matrix consisting of only ones and zeros as will be seen later.

The analysis of a system matrix to find any simple structures by rearrangement of rows and columns, is useful because of the properties of some of these structures. If a matrix can be rearranged into one of two simple structures then the designer can immediately determine the corresponding structure of his controller and can apply decentralised control to the system. If the matrix cannot be rearranged into one of these two forms then the designer gains no new information but at least now knows that decentralised control may be difficult and will certainly require further analysis to determine if decentralised control is possible.

There are a number of advantages offered by simple structure

analysis over an analysis based on more complex structures. Firstly the analysis can be performed before the plant itself has been fully modelled. This is possible because the method is not concerned with the sizes of the individual non-zero elements, or their frequency responses. Hence provided the designer can define the presence or absence of of coupling between different elements in the system the system can be analysed for simple structures. This is important because for large plants the modelling effort is large and could take a considerable amount of time. For example, a modest industrial plant containing 100 inputs and 100 outputs would require between 100 and 10000 transfer function models for a complete input-output description of the process. Thus a 100 by 100 matrix of transfer functions would define the plant dynamics. Assuming conservatively that each model would take half an hour to develop, the entire modelling exercise would occupy between one man-week and two-and-a-half man-years.

Although usually this modelling will have to be done to design the controller and to ensure stability, analysis using simple structures allows the design process to begin before modelling is completed. The exception to this might be the case where some of the plant subsections are known to be stable. In this case it may be possible to design a controller to stabilise the plant without modelling these subsections.

Another, additional, advantage stemming from the fact that simple system analysis can be performed before the plant has been built is the following. It may happen that the analysis will suggest structural changes to the proposed plant that will make it more suitable to decentralised control.

Yet another advantage to applying simple structure analysis to a matrix before applying more complex techniques is that

the method can be applied to non-square matrices. The system matrix must usually be square, i.e. it must have the same number of inputs and outputs, for controller design using existing frequency domain design techniques. Simple structure analysis can be applied to non-square matrices and the resulting structures may give the designer an indication as to which inputs and outputs he can discard to obtain the square matrix that best represents the system.

The final advantage is related to more complex methods of analysis. Some of these methods require that the system matrix be irreducible before they can be applied. i.e. that there does not exist any permutation matrix P such that PGP^T is block triangular, where G is the system matrix. Block triangularity is one of the simple structures to be considered in this chapter, hence if the matrix is analysed for this structure and cannot be rearranged to be block triangular then it will be irreducible and the more complex methods can be applied.

3.2 CLASSIFICATION OF SIMPLE STRUCTURES

There are three simple structures: A **block diagonal structure**, a **block triangular structure** and a **full structure**. Of these the first two indicate that decentralisation of control is possible while the third requires that more complex methods of analysis be used to determine the structure of the system matrix G(s).

3.2.1 The Block Diagonal Structure

In this structural class the plant model, an nxn matrix of transfer functions G(s), can be partitioned into an mxm composite matrix, i.e. an mxm matrix where the individual elements are themselves submatrices and $m < n$. Further all of the on-diagonal submatrices are square and contain at

least one non-zero element on each row and column. Finally all of the off-diagonal submatrices must only contain zero elements. For example if G(s) is partitioned into a 2x2 block matrix then if the matrix is block diagonal it will have the form shown in (3.1).

$$G(s) = \begin{bmatrix} G_{11}(s) & 0 \\ 0 & G_{22}(s) \end{bmatrix} \qquad (3.1)$$

Note that if the on-diagonal sub-matrices are all 1x1 then the system matrix is a diagonal matrix.

The on-diagonal sub-matrices in any partitioned matrix G(s) can be thought of as those systems that the control engineer would like to control, each one independently for decentralised control. The off-diagonal sub-matrices then represent coupling between the different on-diagonal subsystems. Whether or not control may be decentralised will depend on the behavior of these coupling submatrices.

In the block diagonal case the absence of coupling clearly indicates that on-diagonal subsystems will be independent of each other, i.e. in (3.1), for example, $G_{11}(s)$ and $G_{22}(s)$ can be treated as independent subsystems and controllers can be designed independently for each subsystem. In fact for a block diagonal system there is no coupling whatsoever and hence if the controller matrix is not block diagonal coupling may be introduced and the controlled system may have a worse response than that obtained for a fully decentralised controller, i.e. a block diagonal controller.

Hence the controller matrix will almost always be chosen as having a block diagonal structure as is shown in (3.2) for the controller of the system in (3.1).

$$K(s) = \begin{bmatrix} K_{11}(s) & 0 \\ 0 & K_{22}(s) \end{bmatrix} \tag{3.2}$$

If the controller is block diagonal then the open loop system, $Q(s) = G(s)K(s)$, will be

$$Q(s) = \begin{bmatrix} G_{11}(s)K_{11}(s) & 0 \\ 0 & G_{22}(s)K_{22}(s) \end{bmatrix} = \begin{bmatrix} Q_{11}(s) & 0 \\ 0 & Q_{22}(s) \end{bmatrix} \tag{3.3}$$

The closed loop system $H(s)$ is given by

$$H(s) = Q(s)(I+Q(s))^{-1} \tag{3.4}$$

i.e.

$$H(s) = \begin{bmatrix} (I+Q_{11}(s))^{-1}Q_{11}(s) & \\ 0 & (I+Q_{22}(s))^{-1}Q_{22} \end{bmatrix}$$

$$H(s) = \begin{bmatrix} H_{11}(s) & 0 \\ 0 & H_{22}(s) \end{bmatrix} \tag{3.5}$$

Hence the closed loop system is also block diagonal and the diagonal blocks $H_{ii}(s)$ correspond to the closed loop responses of each $Q_{ii}(s)$ block of the open loop system. This implies that if each $H_{ii}(s)$ block is stable then the system as a whole will also be stable. Hence each $G_{ii}(s)$ subsystem can be separately stabilised to ensure the stability of the entire system and control can be fully decentralised.

As a trivial example of the independence of the on-diagonal subsystems consider (3.6).

$$G(s) = \begin{bmatrix} \frac{2.0}{s+1} & 0 \\ 0 & \frac{4.0}{s+4} \end{bmatrix} \qquad (3.6)$$

The graphs in Fig. 3.1 show the closed loop response of G(s) to unit steps in the setpoints R1 and R2 at t=0 and t=5 seconds respectively. The second step was applied after the first output, Y1, had reached a steady state value. The input signals for each loop, marked U1 and U2, are also shown.

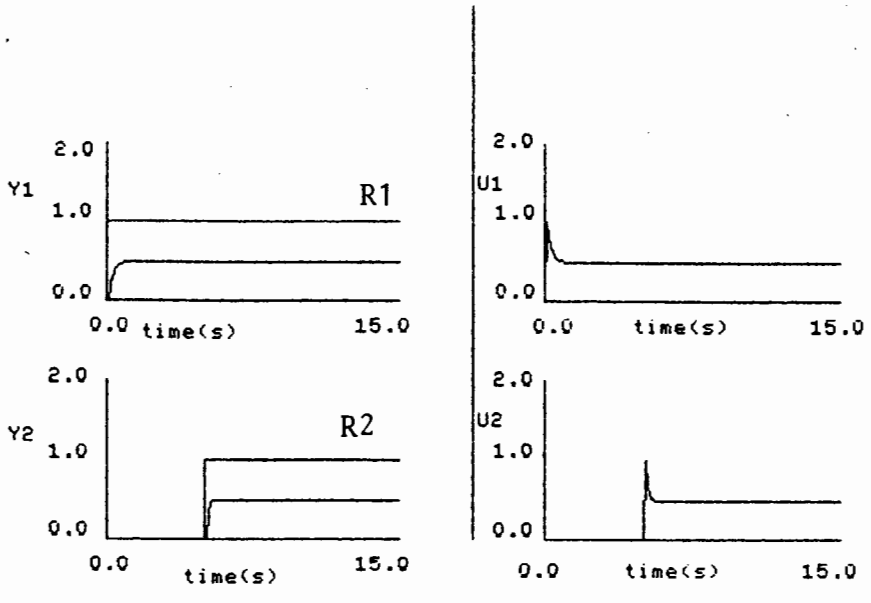

Figure 3.1. The closed loop response of the Inputs and outputs of the system represented in (3.6) to unit steps in the setpoints.

As can be seen neither Y1 nor Y2 suffer any disturbance as a result of a step applied to the input of the other loop. As expected each system is stable with a steady state offset

from the setpoint, 1.0.

3.2.2 The Block Triangular Structure

This structure of G(s) is shown in (3.7) for a 2x2 blocking.

$$G(s) = \begin{bmatrix} G_{11}(s) & 0 \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \qquad (3.7)$$

If G(s) is an nxn matrix partitioned into an mxm composite matrix with square on-diagonal submatrices, then the matrix is lower block triangular if, for each on-diagonal block $G_{ii}(s)$ all those off-diagonal blocks $G_{ij}(s)$ with $j > i$ contain only zero elements.

Obviously a matrix could also have a upper triangular form such as in (3.8)

$$G(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ 0 & G_{22}(s) \end{bmatrix} \qquad (3.8)$$

but a little thought shows that (3.8) can easily be rearranged to give the form sown in (3.7) so that the two structures are equivalent.

The block triangular structure is much more common in large systems than the block diagonal structure and hence is of more interest. Further when the block diagonal structure does occur it is often obvious through plant features, e.g. a plant consisting of two independent processing circuits, and hence is seldom difficult to detect.

In the block triangular case there is interaction between the on-diagonal subsystems, $G_{11}(s)$ and $G_{22}(s)$ in (3.7). However the important point to notice here is that the direction of interaction is not symmetrical. That is, the on-diagonal subsystems are coupled to other subsystems that occur 'higher up' in the system matrix but *not* the other way round. Hence in (3.7) $G_{22}(s)$ is coupled to $G_{11}(s)$ but $G_{11}(s)$ is *not* coupled to $G_{22}(s)$.

To consider what the structure of the controller must be the stability of a triangular system must be considered. In chapter two the stability of such systems was discussed based on [7]. However the complex analysis presented in [7] is usually unnecessary and in this chapter the following theorem is used as a basis for the stability of triangular systems.

Theorem 3.1
Let Q(s) be an nxn matrix of transfer functions which has been partitioned into an mxm composite matrix such that Q(s) is block triangular. Further let the on-diagonal blocks of the composite matrix be square.

Let F be a diagonal matrix of scalar feedback gains partitioned in conformity with Q(s) such that

$$F = \text{diag}\{F_1, F_2, \ldots, F_m\} \tag{3.9}$$

where

$$F_i = \text{diag}\{f_1, f_2, \ldots, f_{mi}\} \tag{3.10}$$

and mi is the order of the $Q_{ii}(s)$ block.

Assume the following:

(i) All the unstable poles of the open loop

system Q(s) lie in the on-diagonal submatrices of the partitioned Q(s).

(ii) The on-diagonal subsystems are all closed loop stable, i.e. $H_{11}(s)$, $H_{22}(s)$,..., $H_{mm}(s)$ are all stable and $H_{ii}(s)=Q_{ii}(s)(F^{-1}+Q_{ii}(s))^{-1}$.

If the above assumptions hold then the closed loop system as a whole, H(s), will be stable.

Proof

The stability of the closed loop system is determined by the roots of its characteristic equation

$$\phi_c(s) = \phi_o(s).\det[F^{-1}+Q(s)] = 0 \qquad (3.11)$$

where $\phi_o(s)$ is the characteristic function of the open loop system.

Clearly the effect of feedback is defined by the roots of the equation

$$\det[F^{-1}+Q(s)] = 0 \qquad (3.12)$$

Since Q(s) is block triangular and F is diagonal the matrix $[F^{-1}+Q(s)]$ will also be block triangular. To illustrate this consider the 2x2 case where

$$Q(s) = \begin{bmatrix} Q_{11}(s) & 0 \\ Q_{21}(s) & Q_{22}(s) \end{bmatrix} \qquad (3.13)$$

is block triangular and

$$F = \begin{bmatrix} F_{11} & 0 \\ 0 & F_{22} \end{bmatrix} \tag{3.14}$$

is block diagonal then

$$F^{-1}+Q(s) = \begin{bmatrix} F_{11}^{-1}+Q_{11}(s) & 0 \\ Q_{21}(s) & F_{22}^{-1}+Q_{22}(s) \end{bmatrix} \tag{3.15}$$

is also block triangular.

From [12] the determinant of (3.12) will be equal to the product of the determinants of the on-diagonal submatrices of $F^{-1}+Q(s)$ i.e.

$$\det[F^{-1}+Q(s)] = \prod_{i=1}^{m} \det[F_i^{-1}+Q_{ii}(s)] \tag{3.16}$$

If the off-diagonal submatrices of Q(s) contain unstable poles then these will appear in $\phi_o(s)$ however by assumption (i) there are no unstable poles in these blocks and hence they do not affect the stability of the closed loop system. Since in feedback only the on-diagonal blocks affect stability and since these are closed loop stable, assumption (ii), the system as a whole must be closed loop stable.

As an example consider the following system

$$G(s) = \begin{bmatrix} \frac{2.0}{s} & 0 \\ \frac{2.0}{s+1} & \frac{4.0}{s+2} \end{bmatrix} \qquad (3.17)$$

Figure 3.2 shows the response of the open loop system to a unit step in the setpoint for the first loop, R1. The two outputs of the system are shown, Y1 and Y2, and the two inputs, U1 and U2.

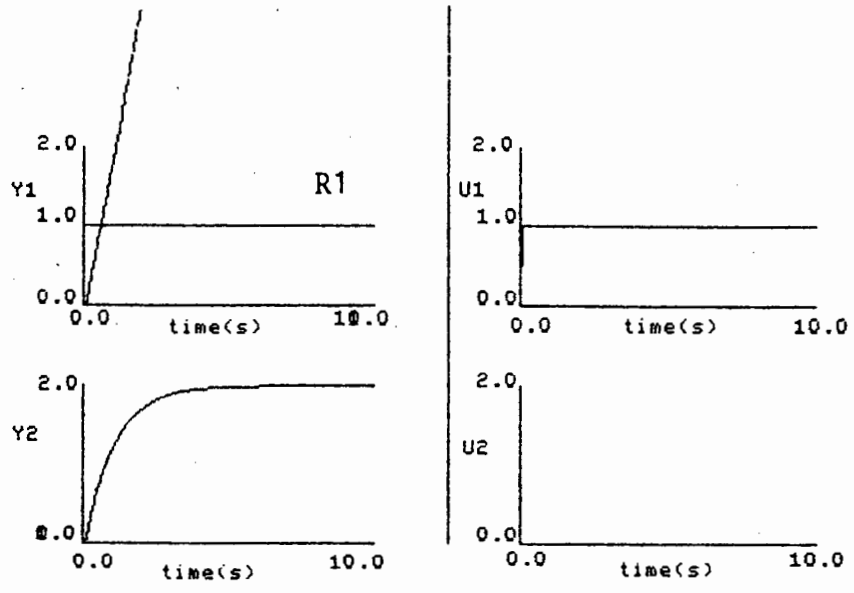

Figure 3.2. The open loop response of the inputs and outputs of the system represented by (3.17)to a step in the setpoint of the first loop.

Since $G_{11}(s)$ has an unstable pole at s=0 the open loop system is expected to be unstable. The output Y1 is seen to be unbounded and hence open loop instability is confirmed. Notice also that although the output Y2 reaches a steady state there is a large offset from the desired value of 0.0. Since $G_{21}(s)$ has no unstable poles, assumption (i) is

fulfilled and theorem 3.1 can be applied. The closed loop transfer functions for $H_{11}(s)$ and $H_{22}(s)$ are

$$H_{11}(s) = 2.0/(s+2)$$

and $$H_{22}(s) = 4.0/(s+6)$$

and so the on-diagonal elements of the closed loop system have no unstable poles. Thus from theorem 3.1 the closed loop system must be stable.

Fig.3.3 shows the response of the closed loop system to a unit step in the setpoint of the first loop. The closed system is stable with both Y1 and Y2 reaching their respective setpoints. Hence theorem 3.1 has correctly predicted the stability of the closed loop system.

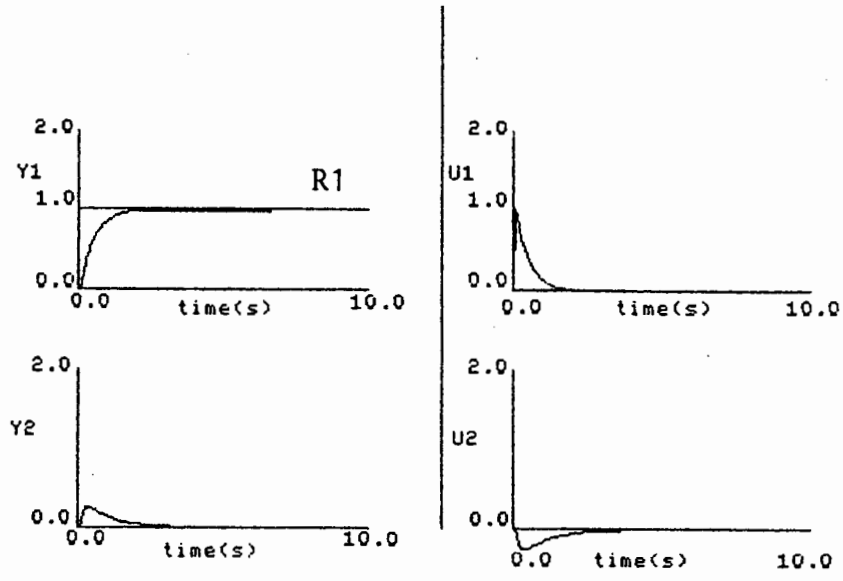

Figure 3.3. Closed loop response of the system represented by (3.17) to a step in the setpoint of the first loop.

Theorem 3.1 is only a sufficient condition for stability. If the off-diagonal elements of Q(s) do contain unstable poles the system may still be stable. As an example consider the following system.

$$G(s) = \begin{bmatrix} \frac{2}{s+1} & 0 \\ \frac{2}{s} & \frac{4}{s} \end{bmatrix} \qquad (3.18)$$

Fig. 3.4 shows the closed loop response of this system to a unit step in the setpoint of the first and second loops at t=0 and t=5 respectively. The outputs, Y1 and Y2, and the inputs, U1 and U2, are shown.

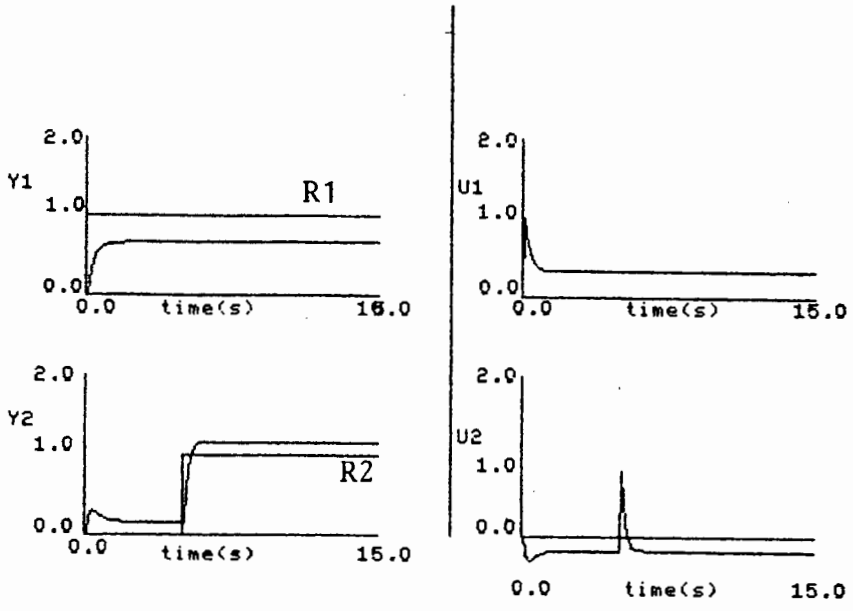

Figure 3.4. Closed loop response of the inputs and outputs of the system represented by (3.18) to steps in the values of the setpoints of the first and second loops.

The system is stable in spite of the pole at s=0 that occurs in the off-diagonal element G_{21}. As would be expected the step in input to the first loop causes a disturbance to the second loop's output, Y2, and both outputs are offset from their setpoints.

It is instructive to look at the closed loop system matrix, H(s).

$$H(s) = \begin{bmatrix} \dfrac{2}{s+3} & 0 \\ \dfrac{2(s+1)}{(s+4)(s+3)} & \dfrac{4}{s+4} \end{bmatrix} \qquad (3.19)$$

As can be seen the $H_{21}(s)$ element has no unstable poles and hence the closed loop system is stable. There has been a cancellation of the s=0 pole in $G_{21}(s)$ by the s=0 pole of the $G_{22}(s)$ element.

If a similar system is considered which does not have a $G_{22}(s)$ element with a pole at s=0.

$$G(s) = \begin{bmatrix} \dfrac{2}{s+1} & 0 \\ \dfrac{2}{s} & \dfrac{4}{s+1} \end{bmatrix} \qquad (3.20)$$

The closed loop response of this system to a unit step in the setpoint of the first loop is shown in Fig. 3.5.

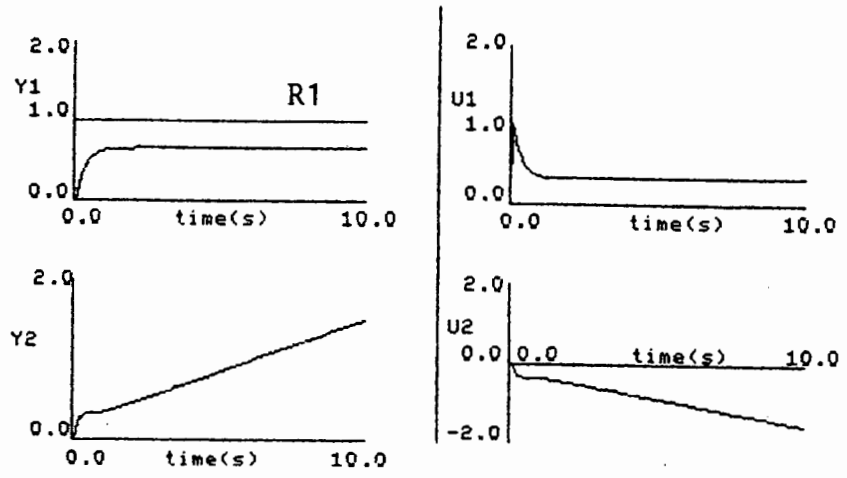

Figure 3.5. The closed loop response of the inputs and outputs of system in (3.20) to a step in the value of the setpoint of the first loop.

As can be seen the output from the first loop, Y1, is stable however the output of the second loop is unbounded and hence the system is unstable. In this case

$$H_{21}(s) = 2(s+1)^2 / (s(s+5)(s+3)) \qquad (3.21)$$

and the pole at s=0 is still present.

In general one will try to avoid pole-zero cancellations where possible since a small shift in the position of the pole or zero will often result in an unstable system, i.e. such systems are not robust. This being the case the designer will usually seek to avoid having unstable poles in the off-diagonal blocks of a block triangular system even if such a system is nominally closed loop stable.

Theorem 3.1 ensures that, provided the off-diagonal submatrices are open loop stable, a block triangular system can be controlled by any controller which (a) does not

introduce any unstable poles into the off-diagonal submatrices of $Q(s)=G(s)K(s)$, (b) does not destroy the block triangular structure of the system i.e. $Q(s)$ is block triangular, and (c) $K(s)$ ensures that each on-diagonal subsystem of $Q(s)$ is closed loop stable.

There are two controller structures that fulfill the requirements of (b) in the previous paragraph. They are the block triangular and the block diagonal structures.

If the controller is fully decentralised it will have a block diagonal structure as shown below for the 2x2 case.

$$K(s) = \begin{bmatrix} K_{11}(s) & 0 \\ 0 & K_{22}(s) \end{bmatrix} \qquad (3.21)$$

Hence the controller matrix will be block diagonal and will be partitioned in conformity with $G(s)$. If $K(s)$ is block diagonal and $G(s)$ block triangular then the open loop system matrix, $Q(s)=G(s)K(s)$, will also be block triangular. This allows theorem 3.1 to be applied. For a 2x2 $G(s)$ as in (3.7) and a $K(s)$ such as that in (3.21) the closed loop system is

$$H(s)=\begin{bmatrix} (I+Q_{11})^{-1}.Q_{11} & 0 \\ (I+Q_{22})^{-1}.Q_{21}.\{I-[I+Q_{11}]^{-1}.Q_{11}\} & (I+Q_{22})^{-1}.Q_{22} \end{bmatrix} \qquad (3.22)$$

Here the Q_{ij} elements are all functions of s. The (s) has been omitted for convenience.

As can be seen from (3.22) the $H(s)$ matrix will also be block triangular. In addition from (3.22) it is seen that the size of the interaction term $H_{21}(s)$ is effectively reduced since both $K_{11}(s)$ and $K_{22}(s)$ tend to reduce its

size. The first by setpoint tracking and the second by disturbance rejection.

Even though stability can be ensured when a block diagonal controller structure is used, the cross coupling term, $H_{21}(s)$, in the closed loop system may still produce unacceptably large disturbances in the system. If this is the case the designer may wish to implement a feedforward controller to eliminate, or reduce these disturbances.

With the addition of feedforward control the controller becomes block triangular. Hence for a block triangular, 2x2 system, like G(s) in (3.7), the controller has the structure shown in (3.23).

$$K(s) = \begin{bmatrix} K_{11}(s) & 0 \\ K_{21}(s)K_{11}(s) & K_{22}(s) \end{bmatrix} \qquad (3.23)$$

Note that the off-diagonal subsystem, $K_{21}(s)$, in the controller is designed to be independent of the on-diagonal term $K_{11}(s)$, both for design purposes and for implementation. The open loop system has the transfer function matrix

$$Q(s) = \begin{bmatrix} G_{11}(s)K_{11}(s) & 0 \\ (G_{21}(s)+G_{22}(s)K_{21}(s))K_{11}(s) & G_{22}(s)K_{22}(s) \end{bmatrix} \qquad (3.24)$$

Once again Q(s) is block diagonal. The on-diagonal subsystems, $K_{11}(s)$ and $K_{22}(s)$, in the control system are feedback controllers, while the off-diagonal term $K_{21}(s)$ is a feedforward controller designed to reduce or to eliminate the disturbance caused by $G_{21}(s)$.

All three controller subsections can be designed separately and independently. Subsection $K_{11}(s)$ of the controller is designed for subsection $G_{11}(s)$ of the plant while $K_{22}(s)$ is designed to stabilise $G_{22}(s)$.

The triangular control structure can be implemented in a distributed configuration of hardware using one control node for each element or block of the controller. This would enhance reliability of the overall system since failure in the feedforward subsection would only downgrade the performance of the system, while failure in either of the feedback control subsections would only lead to partial failure of the entire control scheme for the plant.

The closed loop system, assuming an ideal feedforward design has been made for $K_{21}(s)$, becomes

$$H(s) = \begin{bmatrix} (I+Q_{11})^{-1}.Q_{11}(s) & 0 \\ 0 & (I+Q_{22})^{-1}.Q_{22}(s) \end{bmatrix} \qquad (3.25)$$

For example consider the system in (3.26)

$$G(s) = \begin{bmatrix} \dfrac{2}{s+2} & 0 \\ \dfrac{100}{s+1} & \dfrac{2}{s+2} \end{bmatrix} \qquad (3.26)$$

Fig 3.6 shows the closed loop response of this system to a unit step in the setpoint of the first loop. The outputs, Y1 and Y2, and the system inputs U1 and U2 are shown.

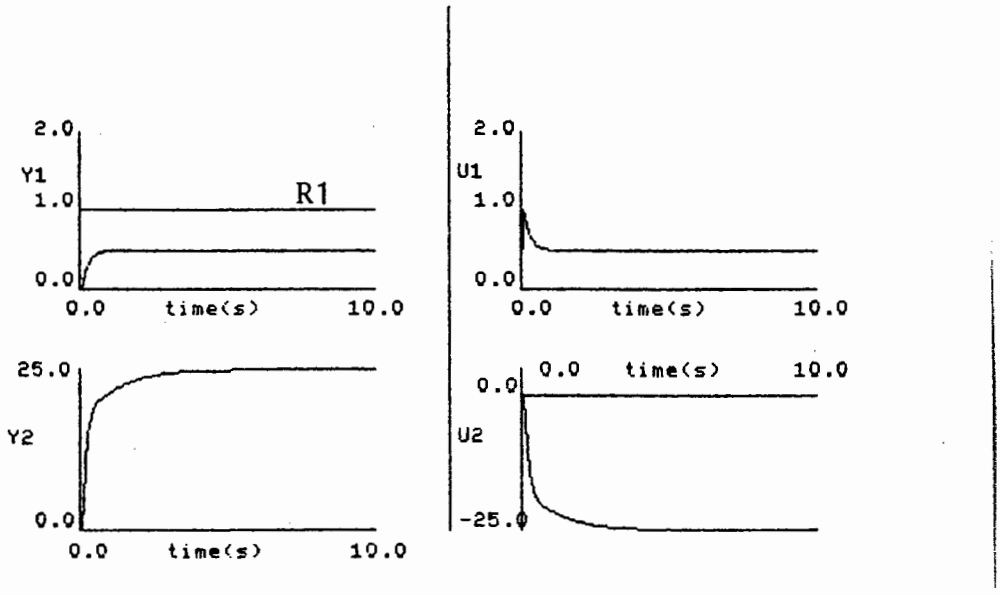

Figure 3.6. The closed loop response of the system represented by (3.26) to a step in the setpoint of the first loop.

The system is stable, however the plot of Y2 shows the large offset introduced by the coupling term $G_{21}(s)$, remembering that ideally Y2 should remain at zero. The following feedforward controller is then introduced.

$$K_{21}(s) = 50(s+2)/(s+1) \tag{3.27}$$

Hence

$$K(s) = \begin{bmatrix} 1 & 0 \\ \dfrac{50(s+2)}{s+1} & 1 \end{bmatrix} \tag{3.28}$$

The same step as before is now applied to the system, $Q(s)=G(s)K(s)$. The resulting responses for the system inputs and outputs are shown in Fig.3.7.

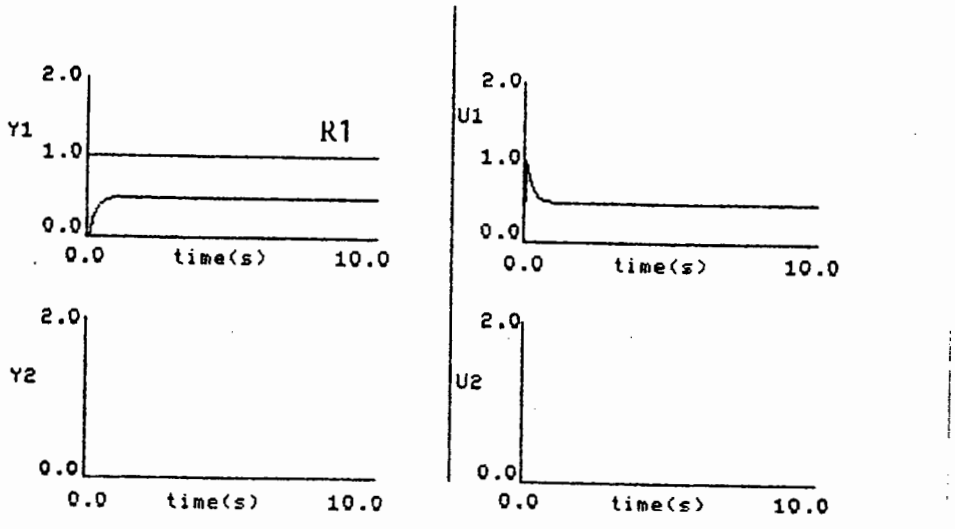

Figure 3.7. Closed loop response of the system represented by (3.26) with feedforward control, to a step change in the setpoint of the first loop.

The response of the Y1 term is unchanged from the previous response, the feedforward element does not affect the closed loop response of the first loop as expected. However the disturbance to the second loop has been eliminated by the action of the feedforward element in the controller, thus effectively decoupling the first and second loops.

3.2.3 Full Structure

A full plant structure is one in which the system matrix, G(s), cannot be rearranged to be either block diagonal or block triangular, i.e. for a 2x2 blocking.

$$G(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \qquad (3.29)$$

In a full structure all of the subsections interact fully and hence controllers designed independently for these

subsections will not necessarily ensure that the plant as a whole is stable even if all the individual elements are open loop stable. Clearly the matrix may still prove to have a useful structure, such as diagonal dominance. However any further analysis requires a complete plant model and many of the advantages of simple structure analysis will be unavoidably lost.

The controller for a full structure plant may still be block triangular but this will depend on the complex structure of the plant system matrix. It may be impossible to decentralise control in which case the controller matrix will have a full structure.

3.2.4 Summary of Simple System Structures

The results presented above are conveniently summarised by table 3.1 showing all useful relationships between plant, control and system structures. The table also indicates whether a particular configuration is 'single-variable' or 'multivariable' (in block form), and distributed or centralised.

Plant	Control	Q(s)	H(s)	Design	Hardware
D	D	D	D	SV	Distributed
T	D	T	T	SV	Distributed
T	T	D	D	SV&FF	Distributed
F	D	F	F	MV	Distributed
F	T	T	T	MV	Distributed
F	F	D	D	MV	Centralised

TABLE 3.1
In the table, the symbols are interpreted as follows:
D=Diagonal, T=Triangular, F=Full, SV=Single-Variable,
FF=Feedforward and MV=Multivariable.

3.3 ANALYSIS OF SIMPLE STRUCTURES

The analysis of a system matrix using simple structures is the process by which the designer rearranges the rows and columns of the system matrix to give either a block diagonal or block triangular system. If such a rearrangement is not possible then the designer should be able to tell this from the method.

3.3.1 The Binary Interaction Matrix

The simple structures are not dependent on the sizes or the frequency dependent behavior of any of the elements in the system matrix, only whether such elements are zero or non-zero. Hence it is convenient to represent the system matrix in terms of another matrix whose elements are either ones or zeros. This matrix is known as an 'adjacency' matrix, [14], or a 'binary interaction matrix', [17]. The latter term will be used in this section and will usually be abbreviated to BIM.

The BIM of a matrix G(s) is defined as follows. Associate with the nxn system matrix G(s) an nxn matrix B(G(s)) = $\{b_{ij}\}$ such that

$$b_{ij} = \begin{cases} 1 & g_{ij}(s) \neq 0 \\ 0 & g_{ij}(s) = 0 \end{cases} \qquad (3.30)$$

B is called the Binary interaction matrix of G(s). Note that the definition of B does not require the value of any $g_{ij}(s)$ term to be known, only whether such a term is zero or non-zero.

3.3.2 Block Triangular Analysis

The block triangular structure is developed first because it is somewhat easier to deal with than the block diagonal case. The designer requires a systematic method of rearranging the rows and columns of the system matrix so as to make the matrix block triangular, if this is possible. The following algorithm, Algorithm 3.1, is used for this purpose. The method is very quick and easy to use, requiring only addition and the manipulation of rows and columns.

Algorithm 3.1

(i) Set up the $N_i x N_j$ BIM, B for the system matrix using the definition given in (3.30). Where N_i is the number of rows of B and N_j is the number of columns.

(ii) For each row, i, assign a number, r_i, such that

$$r_i = N_i - \sum_{j=1}^{Nj} b_{ij} \qquad (3.31)$$

this is the number of zero elements in row i.

(iii) For each column, j, assign a number, c_j, such that

$$c_j = N_j - \sum_{i=1}^{Ni} b_{ij} \qquad (3.32)$$

this is the number of zero elements in column j.

(iv) Rearrange the order of the rows such that for the rearranged B matrix, $r_1 \geq r_2 \geq \ldots \geq r_{Ni}$. In other words, arrange the rows in descending values of r from top to bottom.

(v) Rearrange the order of the columns such that for the rearranged B matrix, $c_1 \leq c_2 \leq \ldots \leq c_{Nj}$. In other words, arrange the columns in descending values of c from right to left.

(vi) Now assign a number, r_i', to each row,i, where r_i' is the number of consecutive zeros as counted on row i starting from column N_j.

(vii) Rearrange the order of the rows such that for the rearranged B matrix, $r_1' \geq r_2' \geq \ldots \geq r_{Ni}'$. In other words, arrange the rows in descending values of r' from top to bottom.

(viii) Now assign a number, c_j', to each column,j, where c_j' is the number of consecutive zeros as counted on column j starting from row 1.

(ix) Rearrange the order of the columns such that for the rearranged B matrix, $c_1' \leq c_2' \leq \ldots \leq c_{Nj}'$. In other words, arrange the columns in descending values of c' from right to left.

(x) Repeat steps (vi) to (ix) until either no further rearrangement is possible or an ordering of rows and columns that has already occurred is repeated.

At this point the matrix should be in block triangular form if it is possible to rearrange the matrix into this form. If the matrix is not in block triangular form then it is impossible to get this structure by row-column

rearrangement.

Steps (i) to (v) will often result in a triangular matrix. The algorithm effectively concentrates the zeros of the matrix, B, in the upper right hand corner of the matrix. However the final four steps are necessary to order those rows and columns that have equal numbers of zero elements.

To see why this might be necessary consider the following 5x5 BIM after steps (i) to (v) have been completed.

$$
B = \begin{array}{c|ccccc|cc}
 & U_1 & U_2 & U_3 & U_4 & U_5 & & \\
\hline
Y_1 & 1 & 1 & 0 & 0 & 0 & (3) & [3] \\
Y_2 & 1 & 0 & 0 & 1 & 0 & (3) & [1] \\
Y_3 & 1 & 1 & 0 & 0 & 0 & (3) & [3] \\
Y_4 & 1 & 1 & 1 & 0 & 1 & (1) & [0] \\
Y_5 & 1 & 1 & 1 & 1 & 0 & (1) & [1]
\end{array}
\qquad (3.32)
$$

The numbers that appear in round brackets at the end of each row are the number of zero elements in each row and these have been used to order the rows in steps (ii) and (iii) of the algorithm. As can be seen no further ordering is possible on this basis. The designer can of course interchange rows with equal numbers of zeros but in a large system such a trial an error rearrangement might prove time consuming. The system in (3.32) is not block triangular as it stands. The numbers in square brackets are the number of zeros that occur consecutively in each row starting from column 5. If ordering is done according to these numbers then the matrix becomes

$$
B = \begin{array}{c|ccc|cc|cc}
 & U_1 & U_2 & U_3 & U_4 & U_5 & & \\
\hline
Y_1 & 1 & 1 & 0 & 0 & 0 & (3) & [3] \\
Y_3 & 1 & 1 & 0 & 0 & 0 & (3) & [3] \\
\hline
Y_2 & 1 & 0 & 0 & 1 & 0 & (3) & [1] \\
Y_5 & 1 & 1 & 1 & 1 & 0 & (1) & [1] \\
Y_4 & 1 & 1 & 1 & 0 & 1 & (1) & [0] \\
 & (0) & (1) & (3) & (3) & (4) & & \\
 & [0] & [0] & [3] & [2] & [4] & &
\end{array}
\qquad (3.33)
$$

The matrix is now block triangular with one 2x2 block and one 3x3 block. However this can be improved upon. The circular brackets at the foot of each column contain the number of non-zero elements that occur in that column. As with the rows no further sorting is possible with these elements. However the square brackets contain the number of zeros that occur consecutively in each column starting in row 1. Sorting on the basis of these gives

$$
B = \begin{array}{c|cc|c|c|c|cc}
 & U_1 & U_2 & U_4 & U_3 & U_5 & & \\
\hline
Y_1 & 1 & 1 & 0 & 0 & 0 & (3) & [3] \\
Y_3 & 1 & 1 & 0 & 0 & 0 & (3) & [3] \\
\hline
Y_2 & 1 & 0 & 1 & 0 & 0 & (3) & [2] \\
\hline
Y_5 & 1 & 1 & 1 & 1 & 0 & (1) & [1] \\
\hline
Y_4 & 1 & 1 & 0 & 1 & 1 & (1) & [0] \\
 & (0) & (1) & (3) & (3) & (4) & & \\
 & [0] & [0] & [2] & [3] & [4] & &
\end{array}
\qquad (3.34)
$$

The matrix is once again block triangular but now control has been distributed between one 2x2, and three 1x1 subsystems. Since no further ordering of the row or columns is possible on the basis of either the round or square brackets the control cannot be decentralised any further. Notice that the number of consecutive row zeros is affected

by the column interchanges which is why step (x) is necessary. In this case no further rearrangement on the basis of the consecutively occurring row zeros is possible and hence the method ends.

Example

As an example of the application of algorithm 3.1 consider a small industrial plant with 7 inputs and 7 outputs. Assume that it has the following BIM

$$B = \begin{array}{c|ccccccc|c} & U_1 & U_2 & U_3 & U_4 & U_5 & U_6 & U_7 & \\ \hline Y_1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & (1) \\ Y_2 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & (4) \\ Y_3 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & (1) \\ Y_4 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & (3) \\ Y_5 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & (4) \\ Y_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & (6) \\ Y_7 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & (3) \\ \hline & (5) & (2) & (4) & (2) & (3) & (2) & (4) & \end{array} \qquad (3.35)$$

The inherent structure of the plant is not obvious from the initial BIM. In fact it appears as if the plant could become a difficult, 7x7, multivariable control problem.

To identify the exact structure of the plant Algorithm 3.1 is applied. The steps mentioned below refer to steps from the algorithm.

The number of zeros in each row from step (ii) are shown in round brackets at the end of each row. The rows are now rearranged so that these numbers are in descending order as specified in step (iii). The same procedure (as specified in steps (iv) and (v)) can then be applied to the columns. The resulting BIM is shown in (3.36).

$$
B = \begin{array}{c|ccc||c||cc|c}
 & U_6 & U_2 & U_4 & U_5 & U_3 & U_7 & U_1 & \\
\hline
Y_6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & (6) \\
\hline
Y_2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & (4) \\
Y_5 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & (4) \\
Y_4 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & (3) \\
\hline
Y_7 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & (3) \\
\hline
Y_1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & (1) \\
Y_3 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & (1) \\
 & (1) & (2) & (2) & (3) & (4) & (4) & (5) &
\end{array} \qquad (3.36)
$$

The matrix is now in block triangular form. Applying steps (vi) to (x) of Algorithm 3.1 does not result in any change to this structure. The 7x7 matrix can now be represented by a 4x4 block, or composite, matrix. Thus in this representation:

$$
G(s) = \begin{array}{|c|c||c|c|}
\hline
G_{11}(s) & 0 & 0 & 0 \\
\hline
G_{21}(s) & G_{22}(s) & 0 & 0 \\
\hline
G_{31}(s) & G_{32}(s) & G_{33}(s) & 0 \\
\hline
G_{41}(s) & G_{42}(s) & G_{43}(s) & G_{44}(s) \\
\hline
\end{array} \qquad (3.37)
$$

This final partitioning is complex and could not easily have been produced by inspection of the original plant structure. An important factor in the analysis is that the partitioning has been carried out on a sound theoretical basis. The designer now knows that, provided there are no unstable elements in the off-diagonal elements, each on-diagonal subsystem can be individually stabilised to ensure the closed loop stability of the system as a whole.

The controller required for the plant will have an identical block structure, i.e it will have the same partitioning as G(s), assuming of course that feedforward elements are

required. The non-zero submatrices of the controller will not necessarily have the same distribution of zero and non-zero elements as the corresponding submatrices in G(s), the actual distribution depending on the design method used.

$$K(s) = \left[\begin{array}{c|c||c|c} K_{11}(s) & 0 & 0 & 0 \\ \hline K_{21}(s) & K_{22}(s) & 0 & 0 \\ \hline\hline K_{31}(s) & K_{32}(s) & K_{33}(s) & 0 \\ \hline K_{41}(s) & K_{42}(s) & K_{43}(s) & K_{44}(s) \end{array}\right] \qquad (3.38)$$

Thus the final conclusion obtained from the analysis of the matrix structure shows clearly that the given plant requires to be controlled by seven independently designed systems, comprising

2x SV feedback controllers (1x1), (1x1)
2x MV feedback controllers (3x3), (2x2)
3x Feedforward controllers (3x4), (3x1), (2x1)

Notice that the original G(s) matrix has been partitioned into a 4x4 composite matrix. This is shown in (3.37) where each $G_{ij}(s)$ element is a submatrix. The partitioning shown with double lines indicates the way in which control will be distributed.

The original problem has been very much simplified by the analysis of the plant structure. Notice that the controller might be simplified yet further by omitting the feedforward controller at the price of increased interaction.

Once the controllers have been designed the independent subsections of K(s) can be distributed in control hardware throughout the plant in order to realise all the advantages of distributed control that have been discussed previously.

A schematic for the robust distribution of control functions on this plant is given in Fig. 3.8. The structural analysis of the the original plant has ensured that the designer has a clear indication of the role of each block in the final distributed control scheme and, more significantly, of the consequences of failures in the control blocks. If any one of the three feedforward controllers fails then at most only two of the separately controlled subsystems will be disturbed. Should a feedback controller controller fail then the extent of the disturbance to the system will depend on which controller fails. If $K_{11}(s)$ then all of the controlled subsytems will be affected. If however $K_{44}(s)$ fails then only the $H_{44}(s)$ closed loop system will go down.

3.3.2.1 Unstable Elements

Algorithm 3.1 and the example given did not take into account the possibility of there being unstable elements in the system matrix. In practice the designer must ensure that such elements occur in the on-diagonal blocks of the composite matrix. This requires some knowledge of the behavior of at least some of the non-zero elements in the matrix but a full system model is still not required.

By marking those elements in the BIM that are unstable with a 'U', for example, the designer can quickly see whether his system is acceptable in terms of the positions of these elements.

For example consider the 7x7 plant given previously. Assume that the BIM for this plant has been analysed for a triangular structure, as before, but that the matrix contains an unstable element marked with a U.

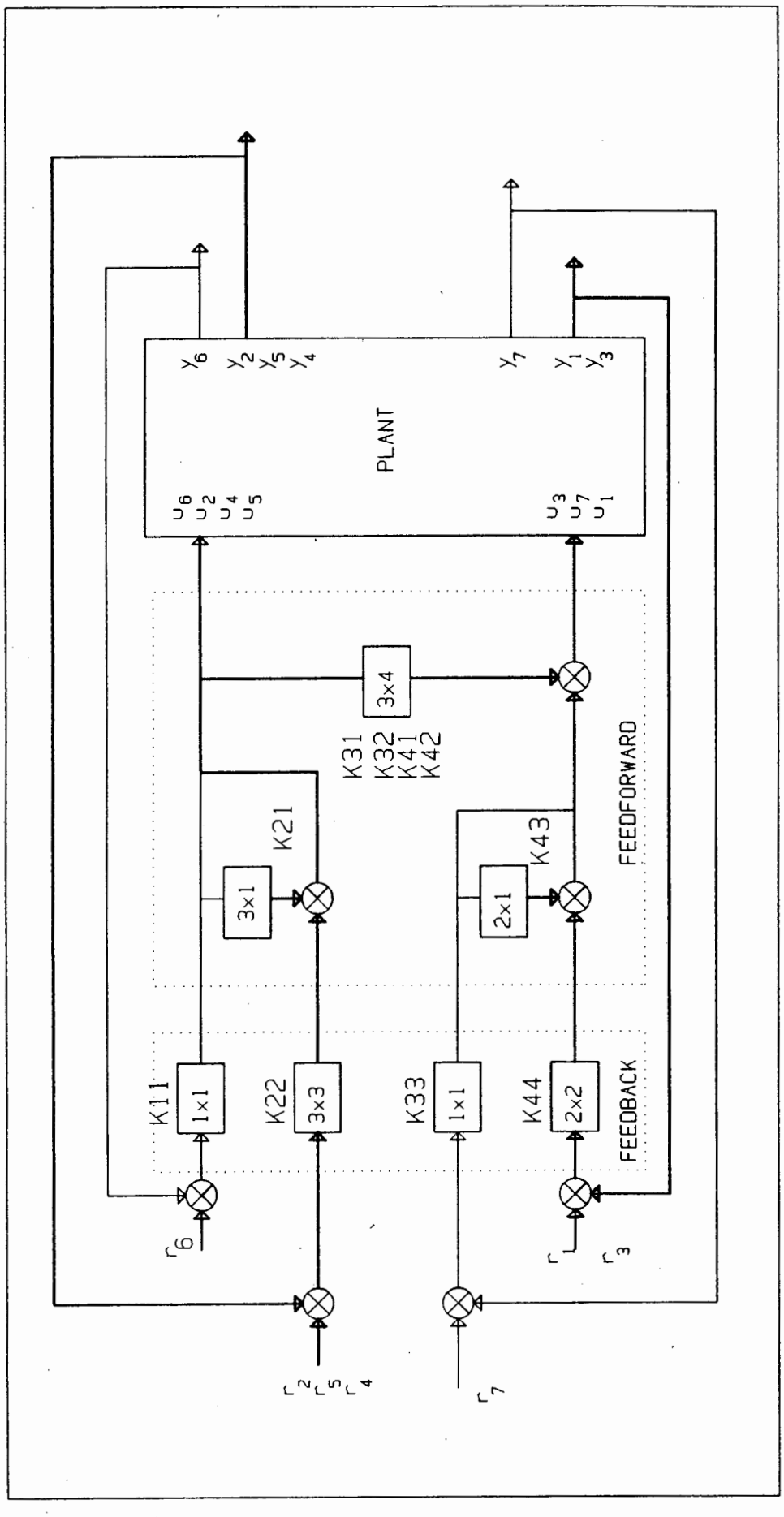

Figure 3.8. Robust, decentralised distribution of control functions.

$$B = \begin{array}{c|c|ccc|c|cc} & U_6 & U_2 & U_4 & U_5 & U_3 & U_7 & U_1 \\ \hline Y_6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline Y_2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ Y_5 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ Y_4 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline Y_7 & 1 & 1 & U & 0 & 1 & 0 & 0 \\ \hline Y_1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ Y_3 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{array} \qquad (3.39)$$

This partitioning is unacceptable because there is an unstable element in an off-diagonal block. However control can still be decentralised by changing the blocking as shown in (3.40).

$$B = \begin{array}{c|c|ccc|c|cc} & U_6 & U_2 & U_4 & U_5 & U_3 & U_7 & U_1 \\ \hline Y_6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline Y_2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ Y_5 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ Y_4 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline Y_7 & 1 & 1 & U & 0 & 1 & 0 & 0 \\ \hline Y_1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ Y_3 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{array} \qquad (3.40)$$

The unstable element has now been included in one of the on-diagonal subsystems and hence theorem 3.1 can now be applied. The disadvantage incurred is a decrease in the number of subsystems that can be independently controlled.

3.3.3 Block Diagonal Analysis

The analysis of a matrix to find a block diagonal structure is slightly more complex than for the block triangular case even though the structure itself is easier to analyse. The method used to detect the block triangular structure will not work when applied to the block diagonal structure

because the blocks of zeros are symmetrically arranged. The method described in algorithm 3.1 depends on there being an unsymmetrical distribution of zeros in order to work.

To describe algorithm 3.2 it is necessary to introduce some basic graph theoretic concepts. These definitions are based on [1].

Consider an nxn BIM, B. Associate with B a directed graph, D(B), which consists of n vertices $P_1,P_2,.....,P_n$ where an edge leads from P_i to P_j if and only if $b_{ij}\neq 0$.

For the indices $1 \leq i,j \leq n$ i is said to have *access* to j if in D(B) there is a sequence of edges which leads from P_i to P_j. i is said to *communicate* with j if i has *access* to j and j has *access* to i.

For a block diagonal B the vertices of D(B) are divided up into isolated sets of vertices that neither *access*, or are themselves *accessed*, by other groups of vertices. These isolated groups correspond to the isolated on-diagonal blocks of B. In addition within each group of vertices each vertex is either *accessed* or is *accessed* by another vertex of the same group. In terms of B this means that for any non-zero element in a particular on-diagonal submatrix there must be at least one other element in the same row and/or column, unless the submatrix has order one. The algorithm labels all of the elements that share common elements in the same rows and columns and then sorts these into the on-diagonal blocks.

Algorithm 3.2

(i) Set up the nxn BIM, $B=\{b_{ij}\}$ for the system to be analysed.

(ii) Set up a corresponding matrix E with elements e_{ij} such that

$$e_{ij} = \begin{cases} j & b_{ij}=1 \\ 0 & b_{ij}=0 \end{cases} \qquad (3.50)$$

i.e. label each non-zero element with its column number.

(iii) Proceed row by row setting each non-zero e_{ij} element in the row to the lowest valued non-zero element in the row.

(iv) Now proceed column by column setting each non-zero e_{ij} element in the column to the lowest valued non-zero element in the column.

(v) Repeat steps (iii) and (iv) until there are no further changes in the values of the e_{ij} elements.

(vi) If all of the non-zero elements of E have the same value, 1, then the matrix cannot be made block diagonal by row and/or column exchanges, Algorithm 3.1 should now be applied to see if the matrix can be made block triangular.

(vii) If there is more than one value represented among the non-zero elements of E then the matrix can be rearranged to be block diagonal. The largest value represented will be the number of on-diagonal blocks. In this case the following procedure yields the block diagonal form. Starting with 1 mark all those

rows and columns that contain 1's with ones. Next number all the rows and columns that contain 2's with twos and so on until all the rows and columns are marked.

(viii) Now reorder the matrix so that the all the rows are in order of the numbers assigned to them with the lowest numbers being assigned to those rows with the lowest row indices. The columns are then rearranged on the same basis as the rows. The matrix will now be in block diagonal form.

Once the matrix is in block diagonal form Algorithm 3.1 should be applied to each on-diagonal submatrix in turn to discover if these can be rearranged to be block triangular.

Note that a very similar algorithm has recently appeared in the literature [19]. Both algorithms are based on the same principle and are equally efficient. The algorithm presented here is in a form that lends itself more easily to implementation on a computer than the form given in [19].

Example

As an example consider the 6x6 BIM shown below.

$$B = \begin{array}{c|cccccc} & U_1 & U_2 & U_3 & U_4 & U_5 & U_6 \\ Y_1 & 0 & 1 & 0 & 1 & 0 & 1 \\ Y_2 & 0 & 0 & 1 & 0 & 0 & 0 \\ Y_3 & 0 & 0 & 0 & 1 & 0 & 0 \\ Y_4 & 1 & 0 & 0 & 0 & 1 & 0 \\ Y_5 & 0 & 0 & 0 & 0 & 0 & 1 \\ Y_6 & 1 & 0 & 0 & 0 & 0 & 0 \end{array} \qquad (3.51)$$

The step numbers below refer to the steps in Algorithm 3.2.

Apply step (ii) to form the E matrix in (3.52)

$$E = \begin{array}{c|cccccc} & U_1 & U_2 & U_3 & U_4 & U_5 & U_6 \\ Y_1 & 0 & 2 & 0 & 4 & 0 & 6 \\ Y_2 & 0 & 0 & 3 & 0 & 0 & 0 \\ Y_3 & 0 & 0 & 0 & 4 & 0 & 0 \\ Y_4 & 1 & 0 & 0 & 0 & 5 & 0 \\ Y_5 & 0 & 0 & 0 & 0 & 0 & 6 \\ Y_6 & 1 & 0 & 0 & 0 & 0 & 0 \end{array} \qquad (3.52)$$

Now apply step (iii) to (3.52) to get (3.53)

$$E = \begin{array}{c|cccccc} & U_1 & U_2 & U_3 & U_4 & U_5 & U_6 \\ Y_1 & 0 & 2 & 0 & 2 & 0 & 2 \\ Y_2 & 0 & 0 & 3 & 0 & 0 & 0 \\ Y_3 & 0 & 0 & 0 & 4 & 0 & 0 \\ Y_4 & 1 & 0 & 0 & 0 & 1 & 0 \\ Y_5 & 0 & 0 & 0 & 0 & 0 & 6 \\ Y_6 & 1 & 0 & 0 & 0 & 0 & 0 \end{array} \qquad (3.53)$$

Apply step (iv)

$$E = \begin{array}{c|cccccc|c} & U_1 & U_2 & U_3 & U_4 & U_5 & U_6 & \\ Y_1 & 0 & 2 & 0 & 2 & 0 & 2 & (2) \\ Y_2 & 0 & 0 & 3 & 0 & 0 & 0 & (3) \\ Y_3 & 0 & 0 & 0 & 2 & 0 & 0 & (2) \\ Y_4 & 1 & 0 & 0 & 0 & 1 & 0 & (1) \\ Y_5 & 0 & 0 & 0 & 0 & 0 & 2 & (2) \\ Y_6 & 1 & 0 & 0 & 0 & 0 & 0 & (1) \\ & (1) & (2) & (3) & (2) & (1) & (2) & \end{array} \qquad (3.54)$$

Further application of steps (iii) and (iv) result in no further changes to E. The highest non-zero element in E is 3. Hence the block diagonal matrix will have three on-diagonal blocks. Applying step (vi) assigns the numbers shown in round brackets in (3.54)

The matrix is now reordered as per step (viii)

$$
E = \begin{array}{c|cc|ccc|c|l}
 & U_1 & U_5 & U_2 & U_4 & U_6 & U_3 & \\
\hline
Y_4 & 1 & 1 & 0 & 0 & 0 & 0 & (1) \\
Y_6 & 0 & 1 & 0 & 0 & 0 & 0 & (1) \\
\hline
Y_1 & 0 & 0 & 2 & 2 & 2 & 0 & (2) \\
Y_3 & 0 & 0 & 0 & 2 & 0 & 0 & (2) \\
Y_5 & 0 & 0 & 0 & 0 & 2 & 0 & (2) \\
\hline
Y_2 & 0 & 0 & 0 & 0 & 0 & 3 & (3) \\
\hline
 & (1) & (1) & (2) & (2) & (2) & (3) &
\end{array} \qquad (3.55)
$$

The matrix is now a block diagonal matrix of the form

$$
G(s) = \begin{bmatrix} G_{11}(s) & 0 & 0 \\ 0 & G_{22}(s) & 0 \\ 0 & 0 & G_{33}(s) \end{bmatrix} \qquad (3.56)
$$

Where $G_{11}(s)$ is a 2x2 submatrix, $G_{22}(s)$ is a 3x3 submatrix and $G_{33}(s)$ is a 1x1 submatrix. If the BIM of the matrix in (3.55) is formed and Algorithm 3.1 is applied to the on-diagonal blocks individually, the final rearranged structure becomes

$$
B = \begin{array}{c|cc|ccc|c}
 & U_1 & U_5 & U_4 & U_6 & U_2 & U_3 \\
\hline
Y_6 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
Y_4 & 1 & 1 & 0 & 0 & 0 & 0 \\
\hline
Y_3 & 0 & 0 & 1 & 0 & 0 & 0 \\
Y_5 & 0 & 0 & 0 & 1 & 0 & 0 \\
Y_1 & 0 & 0 & 1 & 1 & 1 & 0 \\
\hline
Y_2 & 0 & 0 & 0 & 0 & 0 & 1
\end{array} \qquad (3.56)
$$

The designer can now predict the likely structure of the controller. In fact the controller matrix will have a partitioning identical to that in (3.56) assuming that feedforward controllers are required. Notice firstly that

there are three totally independent subsystems in this plant which can as a result be separately controlled. Further two of these subsystems have a block triangular structure and hence control can be distributed within these subsystems.

The first subsystem is a 2x2 subsystem. Two single variable feedback controllers are required and one 1x1 feedforward element. These will all be realised as independent controllers thus fully distributing control.

The second subsystem is a 3x3 subsystem. Three single variable feedback controllers are required. Two feedforward controllers are required to be implemented. They will reduce disturbance to the third on-diagonal element which is coupled to the first two and hence may require feedforward controllers.

The final subsystem is a 1x1 subsystem and hence requires a single single variable feedback controller. The controller structure is summarised below.

6x SV feedback controllers
3x Feedforward controllers

The analysis of the plant has revealed that a very simple controller is required and that the control of this plant can be fully decentralised. The decentralised control of this plant will be robust. If one of the feedforward controllers should fail only one of the six independently controlled processes will be disturbed. Similarly if one of the feedback controllers should fail then at most only two of the processes will be affected.

3.4 AN APPLICATION FROM INDUSTRY

This example is based on a real plant to decide on the choice of independent subsystems that are most likely to lead to an effective decentralised control scheme. The system considered is a milling plant on a gold mine. The example is based on the work reported in [22].

Figure 3.9 shows a schematic diagram of one of the milling circuits. The abbreviations in the diagram are defined in table 3.2 below. The plant itself consisted of two such circuits that operated independently of each other.

In the milling process the solid ore is mixed with water in the rod mill and broken down to form a slurry. This slurry is discharged into the primary sump where further water is added. From the primary sump the slurry is pumped into a pair of primary cyclones. The underflows of the primary cyclones containing solid material that is insufficiently small are fed into each of two pebble mills. From the pebble mills the slurry is passed back into the primary sump.

The overflows of the primary cyclones are passed into a secondary sump where further dilution takes place. From this sump the slurry is pumped into a secondary cyclone. The overflow of this cyclone is the product of the plant, while the underflow is returned to the primary sump.

For each of the two circuits ten outputs and seven inputs were identified. The flow of water to the rod mill, RMFD, and the flow of solids to the rod mill, RMF, are both measured outputs. The RMF is controlled by the speed of the conveyor belt, the RMF belt speed (RMFBS), transporting material to the rod mill. The RMFD is controlled by a valve, the RMFD valve, (RMFV). Both the RMFD valve and the RMF belt speed are inputs to the system. Initially these two outputs,

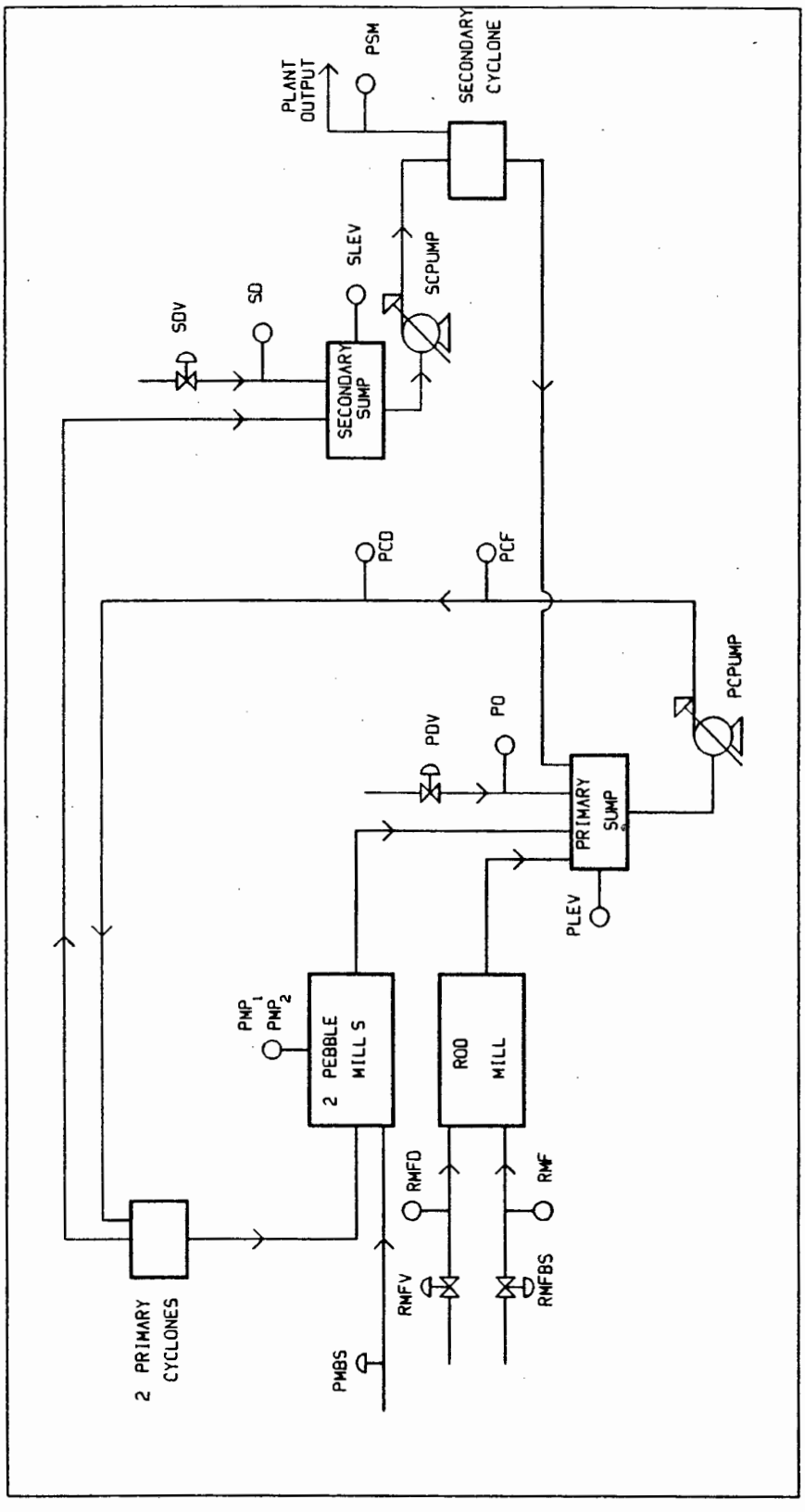

Figure 3.9. Schematic diagram of a milling circuit.

RMF and RMFD, are treated as being separate. However the ratio between the water and solids entering the rod mill is required to be fixed so that the two inputs and outputs will later be combined.

The level of the slurry in the primary sump, PLEV, was another output from the system. The input variables that effect PLEV are as follows. The RMF belt speed and the RMFD valve are both important influences on PLEV as is the dilution of the primary sump, PD, which is controlled by the primary dilution valve, PDV. Another factor that effects PLEV is the primary cyclone pump, PCPUMP. The final factors that change PLEV are the pebble mill belt speeds. There are two pebble mills and consequently two belts that feed the pebble mills. The two pebble mill belt speeds are designated $PMBS_1$ and $PMBS_2$.

The level of slurry in the secondary sump, SLEV, is affected by the secondary cyclone pump, SCPUMP, and the secondary dilution valve, SDV. The dilution of the primary sump, PD, is affected only by the PDV valve. The dilution of the secondary cyclone, SD, is only dependent on the SDV valve. The power to each of the the pebble mills, PMP_1 and PMP_2, is dependent only on the pebble mill belt speed corresponding to the particular pebble mill i.e. $PMPBS_1$ affects PMP_1 and $PMBS_2$ affects PMP_2.

The flow of material into the primary cyclone, PCF and the particle size measurement, PSM, which is measured at the overflow of the secondary cyclone, are both dependent on all seven input variables i.e. RMFV, RMFBS, PDV, SDV, PCPUMP, SCPUMP and $PMPBS_1$ and $PMPBS_2$. The final output variable is the density of the feed to the primary cyclone, PCD. This variable was dependent on RMFV, RMFBS, PDV, SDV, and $PMPBS_1$ and $PMPBS_2$.

A simple structure analysis can be carried out using the techniques developed in chapter three. This does not require a detailed plant model only knowledge as to the presence or absence of coupling between the various plant elements. In order to show the power of the simple structure analysis the inputs and outputs of the plant have not been ordered in any way. Initially the entire plant is considered, i.e. both milling plants. Since both milling circuits are identical the variables in the first circuit are identified by a one in brackets after the variable name. Those in two are identified by a two in brackets. Hence RMF(2) would be the feed rate of solids to the rod mill. Table 3.2 gives the nomenclature used in this example.

VARIABLE	DESCRIPTION
RMF	Feed rate of solids to rod mill (kgs^{-1})
RMFD	Feed rate of water to the rod mill (kgs^{-1})
PD	Dilution of the primary sump (ls^{-1})
SD	Dilution of the secondary sump (ls-1)
PLEV	Level of primary sump (cm)
SLEV	Level of the secondary sump (cm)
PMP_1	Power to 1st pebble mill (kW)
PMP_2	Power to 2nd pebble mill (kW)
PCF	Flow rate of feed to primary cyclones (ls^{-1})
PCD	Density of feed to primary cyclones ($kg.m^{-3}$)
PSM	Particle size measurement (% < 75μm)
RMFV	RMFD valve position (%)
RMFBS	RMF belt speed (% full speed)
PDV	Primary sump dilution valve position (%)
SDV	Secondary sump dilution valve position (%)
PCPUMP	Primary cyclone pump speed (% full speed)
SCPUMP	Secondary cyclone pump speed (% full speed)
$PMBS_1$	First pebble mill belt speed (% full speed)
$PMBS_2$	Second pebble mill belt speed (% full speed)

TABLE 3.2

The BIM for the 22x16 plant is shown in (3.57). As can be seen there is no discernible structure to the matrix that might make the designer's task any simpler. At this stage it

seems possible that a very complex multivariable controller is required. The matrix is however sparse, seventy nine percent of the elements are zeros, which suggests that a simple structure analysis might prove useful. Initially algorithm 3.2 will be applied to see if the matrix can be rearranged to be block diagonal.

OUTPUTS

PCD(2)	0	1	1	0	1	1	1	0	1	0	0	0	1	0	0	1
RMF(2)	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
SLEV(1)	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
PSM(2)	0	1	1	0	1	1	1	0	1	0	0	0	1	0	0	1
PLEV(2)	0	0	1	0	0	1	1	0	1	0	0	0	1	0	0	1
PD(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
PCD(2)	0	1	1	0	0	1	1	0	0	0	0	0	1	0	0	1
RMFD(2)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SD(2)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PD(2)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
PCF(1)	1	0	0	1	0	0	0	1	0	1	1	1	0	1	1	0
RMF(1)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
$PMP_2(2)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
PSM(1)	1	0	0	1	0	0	0	1	0	1	1	1	0	1	1	0
PCD(1)	1	0	0	1	0	0	0	0	0	0	1	1	0	1	1	0
PLEV(1)	0	0	0	1	0	0	0	0	0	1	1	1	0	1	1	0
$PMP_1(2)$	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$PMP_2(1)$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
SLEV(2)	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
RMFD(1)	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
$PMP_1(1)$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
SD(1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I	**S**	**S**	**P**	**R**	**S**	**R**	**P**	**S**	**P**	**P**	**P**	**R**	**R**	**P**	**P**	**P**
N	**D**	**D**	**M**	**M**	**C**	**M**	**D**	**C**	**C**	**C**	**M**	**M**	**M**	**M**	**D**	**M**
P	**V**	**V**	**B**	**F**	**P**	**F**	**V**	**P**	**P**	**P**	**B**	**F**	**F**	**B**	**V**	**B**
U	**(1)**	**(2)**	**S_1**	**B**	**U**	**D**	**(2)**	**U**	**U**	**U**	**S_2**	**D**	**B**	**S_1**	**(1)**	**S_2**
T			**(2)**	**S**	**M**	**V**		**M**	**M**	**M**	**(1)**	**V**	**S**	**(1)**		**(2)**
S				**(1)**	**P**	**(2)**		**P**	**P**	**P**		**(1)**	**(2)**			
					(2)			**(1)**	**(2)**	**(1)**						

(3.57)

When steps (i) to (v) of the algorithm have been completed the E matrix is as is shown in (3.58). Notice that there are only two non-zero values represented in the matrix. This tells the designer that the system is composed of two independent subsystems and that consequently the matrix can

be rearranged to be block diagonal with two on-diagonal blocks. The numbers in brackets above each column and at the end of each row indicate the block to which the row or column belongs.

OUTPUTS	(1)	(2)	(2)	(1)	(2)	(2)	(2)	(1)	(2)	(1)	(1)	(1)	(2)	(1)	(1)	(2)	
PCD(2)	0	2	2	0	2	2	2	0	2	0	0	0	2	0	0	2	(2)
RMF(2)	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	(2)
SLEV(1)	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	(1)
PSM(2)	0	2	2	0	2	2	2	0	2	0	0	0	2	0	0	2	(2)
PLEV(2)	0	0	2	0	0	2	2	0	2	0	0	0	2	0	0	2	(2)
PD(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	(1)
PCD(2)	0	2	2	0	0	2	2	0	0	0	0	0	2	0	0	2	(2)
RMFD(2)	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	(2)
SD(2)	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(2)
PD(2)	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	(2)
PCF(1)	1	0	0	1	0	0	0	1	0	1	1	1	0	1	1	0	(1)
RMF(1)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	(1)
PMP_2(2)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	(2)
PSM(1)	1	0	0	1	0	0	0	1	0	1	1	1	0	1	1	0	(1)
PCD(1)	1	0	0	1	0	0	0	0	0	0	1	1	0	1	1	0	(1)
PLEV(1)	0	0	0	1	0	0	0	0	0	1	1	1	0	1	1	0	(1)
PMP_1(2)	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	(2)
PMP_2(1)	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	(1)
SLEV(2)	0	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	(2)
RMFD(1)	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	(1)
PMP_1(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	(1)
SD(1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(1)

I	**S**	**S**	**P**	**R**	**S**	**R**	**P**	**S**	**P**	**P**	**P**	**R**	**R**	**P**	**P**	**P**
N	**D**	**D**	**M**	**M**	**C**	**M**	**D**	**C**	**C**	**C**	**M**	**M**	**M**	**M**	**D**	**M**
P	**V**	**V**	**B**	**F**	**P**	**F**	**V**	**P**	**P**	**P**	**B**	**F**	**F**	**B**	**V**	**B**
U	(1)	(2)	**S_1**	**B**	**U**	**D**	(2)	**U**	**U**	**U**	**S_2**	**D**	**B**	**S_1**	(1)	**S_2**
T			(2)	**S**	**M**	**V**		**M**	**M**	**M**	(1)	**V**	**S**	(1)		(2)
S				(1)	**P**	(2)		**P**	**P**	**P**		(1)	(2)			
					(2)			(1)	(2)	(1)						

(3.58)

In (3.59) the rows and columns have been rearranged in order of the bracketed numbers and as can be seen the resulting matrix is indeed block diagonal. This result was not unexpected in this example since the plant consists of two identical but separate milling circuits

OUTPUTS	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)	
SLEV(1)	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	(1)
PD(1)	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	(1)
PCF(1)	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	(1)
RMF(1)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(1)
PSM(1)	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	(1)
PCD(1)	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	(1)
PLEV(1)	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	(1)
$PMP_2(1)$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	(1)
RMFD(1)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	(1)
$PMP_1(1)$	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	(1)
SD(1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(1)
PCD(2)	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	(2)
RMF(2)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	(2)
PSM(2)	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	(2)
PLEV(2)	0	0	0	0	0	0	0	0	0	2	0	2	2	2	2	2	(2)
PCD(2)	0	0	0	0	0	0	0	0	2	2	0	2	2	0	2	2	(2)
RMFD(2)	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	(2)
SD(2)	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	(2)
PD(2)	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	(2)
$PMP_2(2)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	(2)
$PMP_1(2)$	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	(2)
SLEV(2)	0	0	0	0	0	0	0	0	2	0	2	0	0	0	0	0	(2)

INPUTS																
	SDV (1)	RMFBS (1)	SCPUMP (1)	PCPUMP (1)	PMBS$_2$ (1)	RDVF (1)	PMBS$_1$ (1)	PDV (1)	SDV (2)	PMBS$_1$ (2)	SCPUMP (2)	RMFDV (2)	PDV (2)	PCMPS (2)	RMFBS (2)	PMBS$_2$ (2)

(3.59)

Because the two circuits are identical further analysis need be performed on only one of the on-diagonal blocks. The binary interaction for one of the blocks is shown in (3.60). The BIM shows that while the control problem has already been greatly simplified, the design of a controller will be difficult unless the system can be rearranged to be block triangular.

OUTPUTS	(6)	(6)	(8)	(8)	(6)	(6)	(6)	(6)	
SLEV(1)	1	0	1	0	0	0	0	0	(6)
PD(1)	0	0	0	0	0	0	0	1	(7)
PCF(1)	1	1	1	1	1	1	1	1	(0)
RMF(1)	0	1	0	0	0	0	0	0	(7)
PSM(1)	1	1	1	1	1	1	1	1	(0)
PCD(1)	1	1	0	0	1	1	1	1	(2)
PLEV(1)	0	1	0	1	1	1	1	1	(2)
PMP_2(1)	0	0	0	0	1	0	0	0	(7)
RMFD(1)	0	0	0	0	0	1	0	0	(7)
PMP_1(1)	0	0	0	0	0	0	1	0	(7)
SD(1)	1	0	0	0	0	0	0	0	(7)
I	**S**	**R**	**S**	**P**	**P**	**R**	**P**	**P**	
N	**D**	**M**	**C**	**C**	**M**	**M**	**M**	**D**	
P	**V**	**F**	**P**	**P**	**B**	**F**	**B**	**V**	
U	**(1)**	**B**	**U**	**U**	**S_2**	**D**	**S_1**	**(1)**	
T		**S**	**M**	**M**	**(1)**	**V**	**(1)**		
S		**(1)**	**P**	**P**		**(1)**			
			(1)	**(1)**					

(3.60)

This submatrix must now be analysed to see if it can be rearranged to have a block diagonal structure this will be done by applying algorithm 3.1. The numbers in brackets at the beginning of each row and at the end of each column indicate the number of zeros present in the row or column. In (3.61) the matrix has been rearranged to order the round bracketed numbers.

OUTPUTS	(6)	(6)	(6)	(6)	(6)	(6)	(8)	(8)		
PD(1)	0	0	0	0	0	1	0	0	[2]	(7)
RMF(1)	0	1	0	0	0	0	0	0	[6]	(7)
PMP_2(1)	0	0	1	0	0	0	0	0	[5]	(7)
RMFD(1)	0	0	0	1	0	0	0	0	[4]	(7)
PMP_1(1)	0	0	0	0	1	0	0	0	[3]	(7)
SD(1)	1	0	0	0	0	0	0	0	[7]	(7)
SLEV(1)	1	0	0	0	0	0	1	0	[1]	(6)
PCD(1)	1	1	1	1	1	1	0	0	[2]	(2)
PLEV(1)	0	1	1	1	1	1	0	1	[0]	(2)
PCF(1)	1	1	1	1	1	1	1	1	[0]	(0)
PSM(1)	1	1	1	1	1	1	1	1	[0]	(0)
I	**S**	**R**	**P**	**R**	**P**	**P**	**S**	**P**		
N	**D**	**M**	**M**	**M**	**M**	**D**	**C**	**C**		
P	**V**	**F**	**B**	**F**	**B**	**V**	**P**	**P**		
U	**(1)**	**B**	**S_2**	**D**	**S_1**	**(1)**	**U**	**U**		
T		**S**	**(1)**	**V**	**(1)**		**M**	**M**		
S		**(1)**		**(1)**			**P**	**P**		
							(1)	**(1)**		

(3.61)

The numbers in square brackets at the end of each row in (3.61) are the number of consecutive zeros in the row counting left and starting in column eight . The rows are now rearranged to order these numbers and the resulting matrix is shown in (3.62). The partitioning in (3.62) indicates the structure that has emerged from the analysis at this point.

OUTPUTS	(6) [0]	(6) [1]	(6) [2]	(6) [3]	(6) [4]	(6) [5]	(8) [6]	(8) [7]		
SD(1)	1	0	0	0	0	0	0	0	[7]	(7)
RMF(1)	0	1	0	0	0	0	0	0	[6]	(7)
PMP_2(1)	0	0	1	0	0	0	0	0	[5]	(7)
RMFD(1)	0	0	0	1	0	0	0	0	[4]	(7)
PMP_1(1)	0	0	0	0	1	0	0	0	[3]	(7)
PD(1)	0	0	0	0	0	1	0	0	[2]	(7)
PCD(1)	1	1	1	1	1	1	0	0	[2]	(2)
SLEV(1)	1	0	0	0	0	0	1	0	[1]	(6)
PLEV(1)	0	1	1	1	1	1	0	1	[0]	(2)
PCF(1)	1	1	1	1	1	1	1	1	[0]	(0)
PSM(1)	1	1	1	1	1	1	1	1	[0]	(0)
INPUTS	SDV (1)	RMFBS (1)	PMB_2 (1)	RMFDV (1)	PMB_1 (1)	PDV (1)	SCPUMP (1)	PCPUMP (1)		

(3.62)

The matrix can immediately be partitioned into a 2x2 composite matrix. The first on-diagonal block is a 6x6 submatrix and can be partitioned into a block diagonal matrix where each on-diagonal block is 1x1. The second on-diagonal block is a 5x2 matrix. The designer must now choose which rows and columns of this second block to include in a second 4x4 matrix formed from the 7x4 matrix. Moving the row labeled PCD results in the structure shown in (3.63).

OUTPUTS	(6) [0]	(6) [1]	(6) [2]	(6) [3]	(6) [4]	(6) [5]	(8) [6]	(8) [7]		
SD(1)	1	0	0	0	0	0	0	0	[7]	(7)
RMF(1)	0	1	0	0	0	0	0	0	[6]	(7)
PMP_2(1)	0	0	1	0	0	0	0	0	[5]	(7)
RMFD(1)	0	0	0	1	0	0	0	0	[4]	(7)
PMP_1(1)	0	0	0	0	1	0	0	0	[3]	(7)
PD(1)	0	0	0	0	0	1	0	0	[2]	(7)
SLEV(1)	1	0	0	0	0	0	1	0	[1]	(6)
PLEV(1)	0	1	1	1	1	1	0	1	[0]	(2)
PCD(1)	1	1	1	1	1	1	0	0	[2]	(2)
PCF(1)	1	1	1	1	1	1	1	1	[0]	(0)
PSM(1)	1	1	1	1	1	1	1	1	[0]	(0)
INPUTS	SDV (1)	RMFBS (1)	PMB_2 (1)	RMFDV (1)	PMB_1 (1)	PDV (1)	SCPUMP (1)	PCPUMP (1)		

(3.63)

The whole 11x8 BIM has effectively been partitioned into two submatrices, one an 8x8 submatrix and the other a 3x8 submatrix. This corresponds to a similar division of G(s) i.e.

$$G(s) = \left[\begin{array}{c} G_1(s) \\ \hline G_2(s) \end{array}\right]$$

$G_1(s)$ is an 8x8 matrix. This matrix can be partitioned into a block triangular matrix as shown in (3.63). If a controller were to be designed for this subsystem then, if feedforward controllers were required, there would be eight single variable controllers and one feedforward controllers. All of these could be implemented using distributed

hardware. The final controller would be composed of

8 1x1 single variable feedback controllers
1 2x6 feedforward controller

This will yield a robust control scheme for $G_1(s)$. If the feedforward controller fails only two control loops will be degraded. If any one of the single variable controllers fails then at most two other processes will be affected. Since only one feedforward controller is required this indicates that the interaction between the on-diagonal elements is small. This in turn suggests that a diagonal controller matrix with no feedforward elements might well give good control.

The second submatrix $G_2(s)$ has no discernible simple structure nor does it have any input variables that have not already been paired with output variables in $G_1(s)$, a consequence of G(s) having more measured output variables than controllable input variables. However the fact that the outputs of $G_1(s)$ can be controlled by means of single variable controllers suggests that the output variables of $G_2(s)$ be controlled by making the outputs of $G_1(s)$ into the inputs of $G_2(s)$. This is in fact how control was implemented at the actual plant. This example shows the power of simple structural analysis when dealing with large systems.

3.5 SUMMARY

The analysis of a system matrix, to try and rearrange its rows and columns to reveal either of the two useful simple structures that may be inherent in the plant, has been shown to be a powerful design tool. If such structures can be detected they allow the designer to distribute the control of the plant to several separate subsystems. A major advantage associated with this type of analysis is that it

can be applied before a full plant model has been constructed. This arises from the fact that no knowledge sizes of the elements in the model or their frequency response is required. Only the presence or absence of an element in the matrix representing the system is needed.

CHAPTER 4

THE DIAGONAL DOMINANCE STRUCTURE

4.1 INTRODUCTION

If the simple structure analysis described in the previous chapter fails to rearrange the system matrix into a block diagonal or block triangular form then the designer must turn to more complex methods of analysis. Further even if the system is rearranged to be block triangular or diagonal the designer might still want to analyse the full on-diagonal blocks to try and distribute control to an even greater degree.

One matrix structure that has proved very useful to control engineers is that of diagonal dominance. This structure forms the basis of Rosenbrock's INA design technique [12], and was described in chapter two. As mentioned in chapter two very few large systems can be rearranged to be diagonally dominant without scaling the system to ensure dominance. If the scaling matrices are diagonal, so that the final controller matrix (incorporating the scaling matrices) is still diagonal then control can still be distributed. If compensation makes the final controller matrix non-diagonal then control will no longer be fully decentralised.

The scaling of the system matrix to obtain a diagonally dominant system is not an easy problem to solve. There exist linear programming approaches to the problem of making SGP diagonally dominant where S and P are scaling matrices and G is a system matrix. A diagram showing this scaling arrangement is shown in Fig. 4.1.

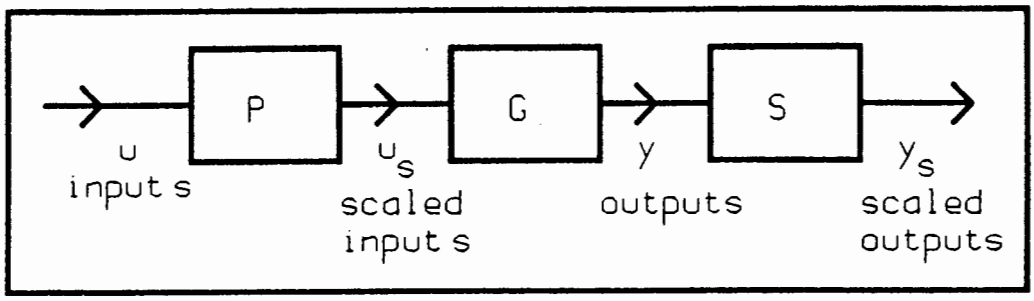

Figure 4.1. Block diagram showing input-output scaling.

Such a scheme will usually alter the poles of the closed loop system and hence affect stability. This means that if the PGS system is diagonally dominant conclusions about the stability of this system do not refer to the original system G. If the scaling arrangement SGS^{-1} is used ,as in [1], then the poles of the system are unchanged and if the SGS^{-1} system is stable then the G system will also be stable. The latter scaling arrangement is the basis for the idea of generalised diagonal dominance as described in chapter two.

In this chapter a means of analysing the system matrix, to see if it is possible to rearrange rows and columns to make the system diagonally dominant will be derived. The diagonal dominance described is the one used in [12], not the somewhat less conservative generalised diagonal dominance of [1]. Scaling the matrix using the arrangement, GD will also be briefly considered, where D is a diagonal matrix of scaling elements.

4.2 DIAGONAL DOMINANCE TESTING

A review of the definition of diagonal dominance is given here. For more details on the use of this structure in the design of controllers see chapter two and [12].

An nxn matrix, $G(s)=\{g_{ij}(s)\}$, is said to be diagonally row dominant iff

$$|g_{ii}(s)| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |g_{ij}(s)| \qquad \begin{matrix} i=1, 2,\ldots,n \\ \text{for all } s \in D \end{matrix} \qquad (4.1)$$

and column diagonally dominant if

$$|g_{jj}(s)| > \sum_{\substack{i=1 \\ i\neq j}}^{n} |g_{ij}(s)| \qquad \begin{matrix} j=1, 2,\ldots,n \\ \text{for all } s \in D \end{matrix} \qquad (4.2)$$

The matrix is said to be diagonally dominant if either (4.1) or (4.2) are satisfied for all s on the Nyquist contour D.

Note that the dominance problem is complicated by the need for (4.1) and/or (4.2) to hold over a wide frequency range.

Initially consider the single frequency case. The problem now is to find the new ordering of the rows and columns of a matrix G, such that either (4.1) or (4.2), or both, are satisfied.

One method of approaching this problem is to calculate the left and right hand sides of (4.1) and (4.2) and to check whether or not the inequalities hold. If not the matrix is rearranged and the test repeated. The main problem with this direct approach is that the values of the right hand sides of (4.1) and (4.2) change whenever the row-column ordering is changed. Ideally what is required is to assign a number to each element that is invariant with row and column interchanges and that the designer can use to establish if that element is dominant on its row or column.

4.2.1 A New Method of Checking for Diagonal Dominance

Consider row dominance. (4.1) can be rewritten as follows (the s variable does not appear as this is for the single frequency case).

$$|g_{ii}| > [\sum_{j=1}^{n} |g_{ij}|] - |g_{ii}| \tag{4.3}$$

then

$$2|g_{ii}| > \sum_{j=1}^{n} |g_{ij}| \tag{4.4}$$

i.e

$$\frac{|g_{ii}|}{\sum_{j=1}^{n} |g_{ij}|} > 0.5 \tag{4.5}$$

Note that the denominator of (4.5) is independent of the row-column arrangement.

Now define the constants δ^i_r and δ^j_c so that,

$$\delta^i_r = \sum_{j=1}^{n} |g_{ij}| \tag{4.6}$$

and

$$\delta^j_c = \sum_{i=1}^{n} |g_{ij}| \tag{4.7}$$

i.e. δ^i_r = the sum of the absolute values of the elements in row i and δ^j_c = the sum of the absolute values of the elements in column j.

If each element element, g_{ij}, in G is associated with the

values $|g_{ij}|/\delta^i_r$ and $|g_{ij}|/\delta^j_c$ then even if the row-column orders change these two values will still be the same for each of the relocated elements. This comes about because the values of all the δ^i_rs and δ^i_rs remain unchanged by row or column interchanges.

Let $|G|$ be the real matrix where the elements of $|G|$ are $|g_{ij}|$ i=1,.2,...,n, j=1, 2,...,n and $|g_{ij}|$ is the absolute value of the element g_{ij} in the matrix G.

Let

$$R = \text{diag}\{\sum_{j=1}^{n}|g_{1j}|, \sum_{j=1}^{n}|g_{2j}|, \ldots, \sum_{j=1}^{n}|g_{nj}|\} \tag{4.8}$$

$$C = \text{diag}\{\sum_{i=1}^{n}|g_{i1}|, \sum_{i=1}^{n}|g_{i2}|, \ldots, \sum_{i=1}^{n}|g_{in}|\} \tag{4.9}$$

$$M = R^{-1}|G| \tag{4.10}$$

$$N = |G|C^{-1} \tag{4.11}$$

M and N define two comparison matrices, M for row dominance and N for column dominance. The elements of M correspond to

$$|g_{ij}|/\delta^i_r \tag{4.12}$$

and the elements of N correspond to

$$|g_{ij}|/\delta^j_c \tag{4.13}$$

As stated before the values of the elements in M and N are independent of the row-column ordering of G. Selecting the dominant elements of the matrix is now a simple task. If any element, m_{ij}, has value such that $m_{ij} > 0.5$, then the g_{ij} element is dominant on row i of G. Similarly if $n_{ij} > 0.5$ then g_{ij} is dominant on column j of G.

By definition there can at most be one dominant element on each row, for row dominance, or one dominant element on each column, for column dominance. Hence once the dominant elements are selected it is easy to choose the row column ordering that will give a diagonally dominant G, if this is possible. Consider row dominance for example. If G can be rearranged to be row dominant then there will be one and only one dominant element in each row and column. If there is more than one dominant element in each column or if any rows do not contain dominant elements then the matrix cannot be made row diagonally dominant. Note that dominant elements are immediately identifiable from M and N as they correspond to those m_{ij} and n_{ij} elements that are greater than 0.5.

It is interesting to compare the above method with one given by Bryant and Yeung in [18]. Here the authors define a comparison matrix D where

$$D_{ij} = \sum_{\substack{p=1 \\ p \neq i}}^{n} |g_{ip}| / |g_{ij}| \qquad (4.14)$$

to check for row dominance.

The designer must then interchange elements so that the smallest element on each row is on the diagonal. In [18] an algorithm is given which sorts the matrix on this basis taking into account conflicts that occur if two rows have their smallest elements in the same column.

This method will yield a diagonally dominant matrix if rearrangement can create such a matrix but for large matrices the algorithm used becomes complex and difficult to implement [18]. In the method presented in this chapter the dominant elements are immediately apparent. Further the elements are all limited to the ranges

$$1 \geq n_{ij} \geq 0$$

$$1 \geq m_{ij} \geq 0$$

which simplifies the comparison of elements in the same row or column. The measure in (4.14) clearly has no upper bound.

As an example of the use of M and N in detecting dominance consider a 3x3 matrix

$$G = \begin{bmatrix} 2.0 & 3.0 & 0.5 \\ 1.0 & 4.0 & 0.1 \\ 10.0 & 2.0 & 0.4 \end{bmatrix} \qquad (4.15)$$

$$M = \begin{bmatrix} 0.36 & 0.55 & 0.09 \\ 0.20 & 0.78 & 0.02 \\ 0.81 & 0.16 & 0.03 \end{bmatrix} \qquad (4.16)$$

$$N = \begin{bmatrix} 0.15 & 0.33 & 0.5 \\ 0.08 & 0.45 & 0.1 \\ 0.77 & 0.22 & 0.4 \end{bmatrix} \qquad (4.17)$$

First consider row dominance. If a particular element in the matrix is row dominant then the corresponding element in the M matrix will have have a value greater than 0.5. Inspection of the M matrix shows that the elements m_{12}, m_{22} and m_{31} are all greater than 0.5. The elements g_{12}, g_{22} and g_{31} must be placed on the diagonal of G, if G is to be row diagonally dominant. In this particular case it is not possible to rearrange the rows and columns to get all of the dominant elements onto the diagonal. This is immediately apparent since two of the dominant elements, g_{12} and g_{22} are in the same column.

For column dominance consider the N matrix. This matrix contains only one element greater than 0.5, m_{31}. The element m_{13} is equal to 0.5 but this does not fulfill the requirement that $m_{ij} > 0.5$ for dominance. Hence the matrix G cannot be rearranged to be column dominant. This analysis shows that the matrix G cannot be made diagonally dominant by row and/or column rearrangement.

It is important to note that scaling has not been taken into account. It may still be possible to make the system diagonally dominant by scaling the inputs and outputs.

The next step is to extend this method to frequency dependent matrices, i.e. G = G(s). For a matrix of frequency dependent elements dominant elements must have corresponding m_{ij} or n_{ij} values greater than 0.5 for all s on the Nyquist contour D.

An element $g_{ik}(s)$ can be represented as $|g_{ik}(s)|e^{j[\Phi ik(s)]}$ where $\Phi_{ik}(s)$ is a frequency dependent angle in radians. (Note that the subscript k is used for the column index instead of the usual, j, to avoid confusion with the complex number j). The frequency response of this element can now be plotted on a polar diagram with an angle $\Phi_{ik}(s)$, and $|g_{ik}|$ being the distance from the origin to the the point currently being plotted. Since values of s are usually only taken along the imaginary axis, s = jw where w is the angular frequency in radians/second.

The definitions of the M and N matrices can now be modified to give a frequency dependent representation. Define M(s) as the matrix with elements $m_{ik}(s)$ such that

$$m_{ik}(s) = (|g_{ik}(s)|e^{j[\Phi ik(s)]}) \; / \; \delta^{i}_{r}(s) \qquad (4.18)$$

Similarly define the elements of N(s) as

$$n_{ik}(s) = (|g_{ik}(s)|e^{j[\Phi ik(s)]}) \; / \; \delta^{k}_{c}(s) \qquad (4.19)$$

Where δ^{i}_{r} and δ^{k}_{c} were defined in (4.6) and (4.7) respectively.

Polar plots of the $m_{ik}(s)$ and $n_{ik}(s)$ elements can now be made. Since the values of these elements are never greater than one the resulting plots will be contained within the unit circle on the polar diagram. If a circle of radius 0.5 is drawn on the polar diagram then the dominance of a particular element can be determined. If an $m_{ik}(s)$ element, for example, has a polar plot that is completely outside the 0.5 radius circle then $g_{ik}(s)$ is dominant on its row for all frequencies considered. If the plot is inside the 0.5 circle for all frequencies then the element is never dominant on its row and if the plot crosses the 0.5 circle at any point then the element is dominant over only part of the frequency range considered. This all follows from the necessity that $m_{ik}(s)$ lie in the range $1 \geq m_{ik}(s) > 0.5$ for row dominance.

In order to decide on the row-column ordering the designer considers both the M(s) and N(s) matrices separately. These matrices are represented as arrays of polar plots each plot representing an element in the corresponding M(s) or N(s) matrix. This representation provides a simple visual display of the way in which the dominance of the elements in the system matrix, G(s), varies with frequency.

Note that even if it is not possible to rearrange G(s) to be diagonally dominant this representation still gives the designer a means of selecting those elements that are closest to being dominant on their row or column. The designer can then try to place these elements on the diagonal in order to reduce coupling between the various

diagonal elements. Further note that the stability theorems in [12] , for the direct Nyquist array method, require the matrix $[F^{-1}+Q(s)]$ to be diagonally dominant where F is a matrix of feedback gains and $Q(s)=G(s)K(s)$. In the simplest case with $F = K(s) = I$ the matrix $[I+G]$ is reqired to be diagonally dominant. If G(s) is diagonally dominant then the matrix $[I+G]$ is also diagonally dominant and the stability theorems can be applied immediately. If G(s) is not diagonally dominant then $[I+G(s)]$ may still be diagonally dominant. Placing the elements with the largest $m_{ij}(s)$ or $n_{ij}(s)$ values on the diagonal of G(s), even if they are not actually dominant on their row or column, will provide the best chance of making $[I+G(s)]$ diagonally dominant.

4.2.2 Examples

A program has been written to enable the designer to plot the polar diagram arrays corresponding to the matrices M(s) and N(s). The program is described in detail in chapter six. In this chapter all of the examples have been generated using this program.

As an example of the use of the above representation consider a 2x2 matrix of first order transfer functions

$$G(s) = \begin{bmatrix} \frac{4}{s+1} & \frac{4}{s+2} \\ \frac{1}{s+1} & \frac{10}{s+2} \end{bmatrix} \qquad (4.20)$$

Figures 4.2 and 4.3 show the arrays of polar plots for M(s) and N(s) respectively. The position of each plot in the array corresponds to the position of the matrix element

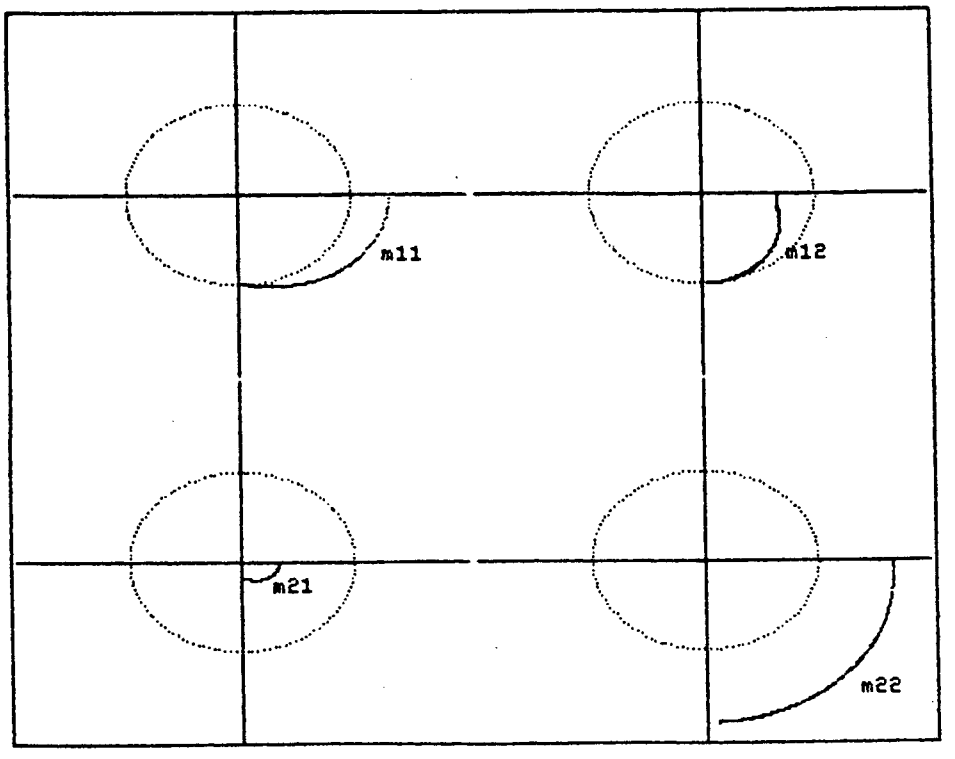

Figure 4.2. Array of polar plots of the elements of the M(s) matrix of the system represented by (4.20).

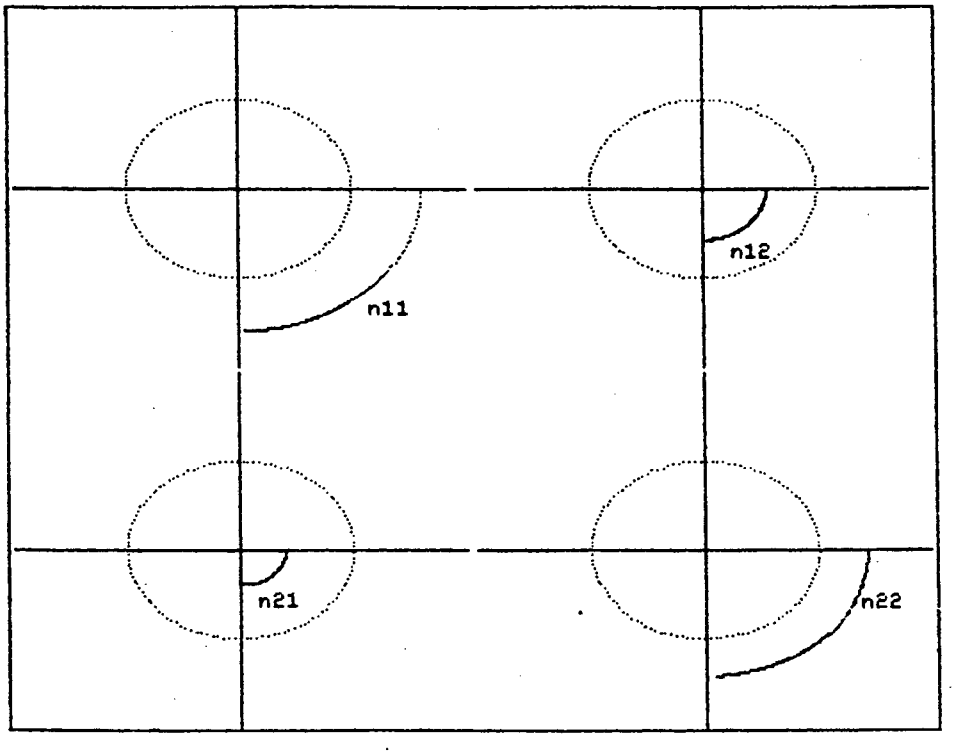

Figure 4.3. Array of polar plots of the elements of the N(s) matrix of the system represented by (4.20).

giving rise to the plot. In both cases the frequency range is 0 to 5 hz.

Since Fig. 4.2 is the array of polar plots for the elements of M(s) it is used to check for row dominance. Note that the 0.5 radius circle has been drawn in on each set of axes, the circle appears to be elliptical because of different scales on the x and y axes. Consider the polar diagram corresponding to $m_{11}(s)$, i.e. the diagram in the top left hand corner of the array. At low frequencies the plot of $m_{11}(s)$ is outside the 0.5 circle. This implies that $g_{11}(s)$ is dominant on its row at these frequencies. At high frequencies the plot of $m_{11}(s)$ touches the 0.5 circle indicating that the $g_{11}(s)$ element is not row dominant at high frequencies. Examination of the other element in this row, $m_{12}(s)$ shows that at low frequencies the plot for this element lies inside the 0.5 circle and touches the 0.5 circle at high frequencies. This indicates that the $g_{12}(s)$ element is not row dominant over at any frequency in the range considered.

Looking at the second row of Fig. 4.2 it is clear that element $g_{22}(s)$ is dominant over the entire frequency range. The plot of $m_{22}(s)$ always lies outside the 0.5 circle indicating dominance. As would be expected the $m_{21}(s)$ plot always lies within the 0.5 circle since two dominant elements cannot occur in the same row.

The above analysis shows that G(s) cannot be rearranged to be row dominant over the entire frequency range since the first row contains no element that is dominant at all frequencies.

Now consider Fig.4.3. This shows an array of polar plots for the N(s) matrix. It is easily seen from this array that G(s) is already column dominant without any need for row-column

rearrangement. This is apparent from the polar plots of the $n_{11}(s)$ and $n_{22}(s)$. In both cases the polar plots lie well outside the 0.5 circles indicating the column dominance of $g_{11}(s)$ and $g_{22}(s)$.

As a second example consider a hypothetical plant represented by the 3x3 matrix model in (4.21)

$$G(s) = \begin{bmatrix} \frac{10}{s+1} & \frac{2}{s+1} & \frac{0.1}{s+1} \\ \frac{4}{s+2} & \frac{0.5}{s+1} & \frac{6}{s+2} \\ \frac{1}{s+1} & \frac{3}{s+1} & \frac{2}{s+2} \end{bmatrix} \quad (4.21)$$

Figure 4.4 shows the array of polar plots for the M(s) matrix corresponding to this system, the frequency range is from 0 to 1 hz. The row dominant elements are immediately apparent as those elements whose polar plots lie outside the 0.5 circle. In this case the polar plots of $m_{11}(s)$, $m_{23}(s)$ and $m_{32}(s)$ all lie outside the 0.5 circle. This implies that the elements $g_{11}(s)$, $g_{23}(s)$ and $g_{32}(s)$ are dominant on their rows. Since there is a row dominant element on each row and only one such element in each column the matrix can be rearranged to be row diagonally dominant. In this example exchanging columns two and three will make the matrix row diagonally dominant. Notice however that at high frequencies the polar plot of $m_{32}(s)$ approaches the 0.5 circle which indicates that at these frequencies the dominance of the $g_{32}(s)$ element may be lost.

Figure 4.5 shows the array of polar plots for the matrix N(s) corresponding to (4.21). Now column dominance will be indicated for an element $g_{ij}(s)$ if the corresponding $n_{ij}(s)$

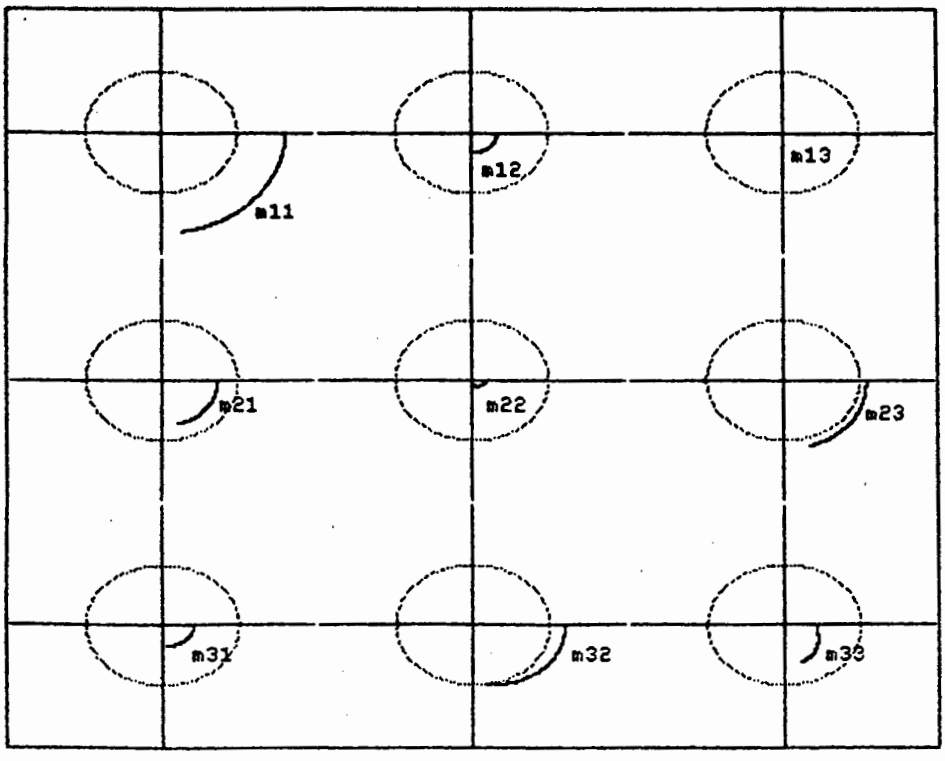

Figure 4.4. Array of polar plots of the elements of the M(s) matrix of the system represented by (4.21).

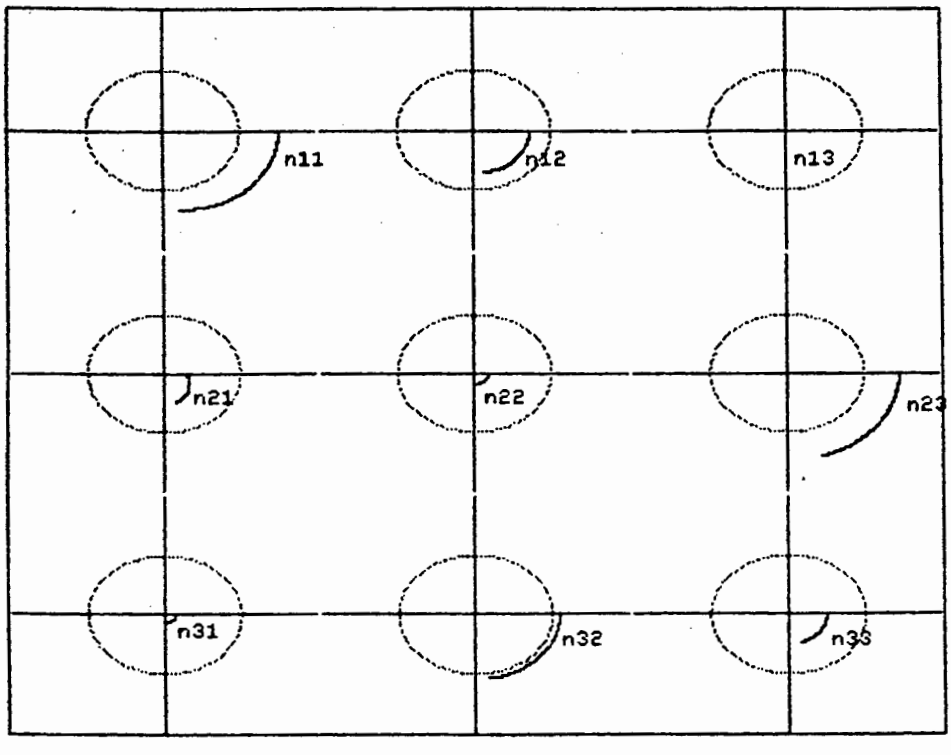

Figure 4.5. Array of polar plots of the elements of the N(s) matrix of the system represented by (4.21).

element has a polar plot lying outside the 0.5 circle. The plots of $n_{11}(s)$, $n_{23}(s)$ and $n_{32}(s)$ all lie outside the 0.5 circle which implies that elements $g_{11}(s)$, $g_{23}(s)$ and $g_{32}(s)$ are dominant on their columns. In this case the plot for $n_{32}(s)$ lies outside the 0.5 circle for all frequencies considered. Once again there is only one dominant element in each row and column and hence the matrix can be rearranged to be diagonally dominant. If columns two and three are exchanged then the matrix will be diagonally dominant.

This analysis shows that the matrix in (4.21) can be rearranged to be both row and column diagonally dominant by exchanging columns two and three.

It is often the case that an element may be dominant on its row or column for only part of the frequency range considered. The following example shows this clearly.

$$G(s) = \begin{bmatrix} \dfrac{2}{s+2} & \dfrac{2.5}{s+3} \\ \dfrac{1}{s+4} & \dfrac{2}{s+3} \end{bmatrix} \qquad (4.22)$$

Figure 4.6 shows the array of polar plots for M(s). The element $g_{22}(s)$ is dominant on its row over the entire frequency range considered, 0 to 5 hz. This is evident from the plot of $m_{22}(s)$ which always lies outside the 0.5 circle. However the elements $g_{11}(s)$ and $g_{12}(s)$ are both dominant but over different parts of the frequency range considered. At low frequencies the plot of $m_{11}(s)$ lies outside the 0.5 circle which indicates that at these frequencies $g_{11}(s)$ is dominant on the row. At higher frequencies however the plot of $m_{11}(s)$ moves inside the 0.5 circle and hence $g_{11}(s)$ is no

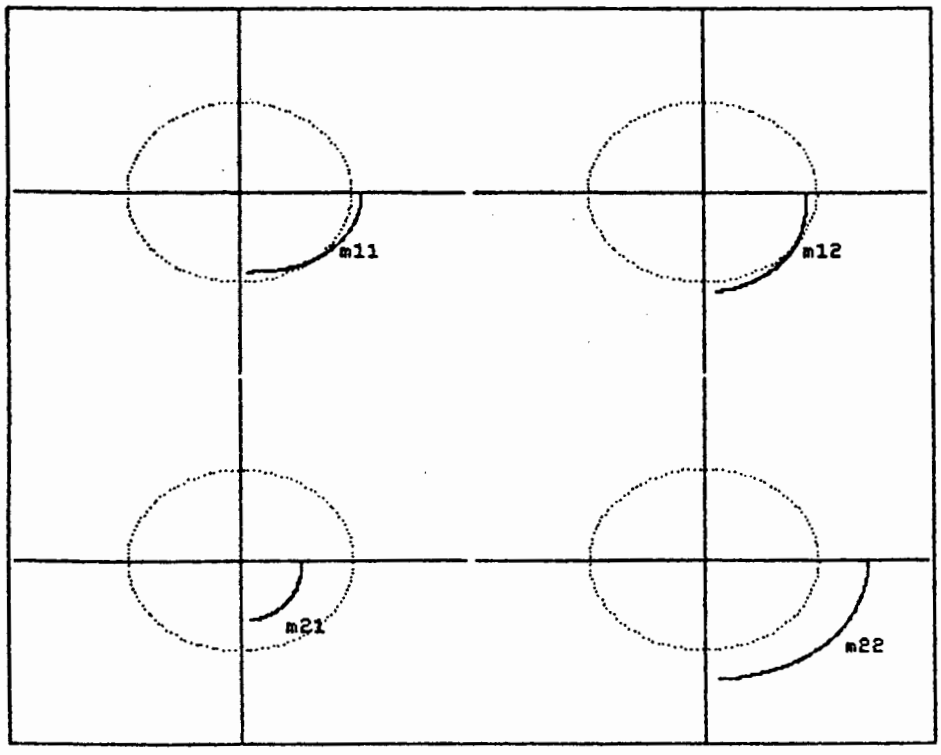

Figure 4.6. Array of polar plots of the elements of the M(s) matrix of the system represented by (4.22).

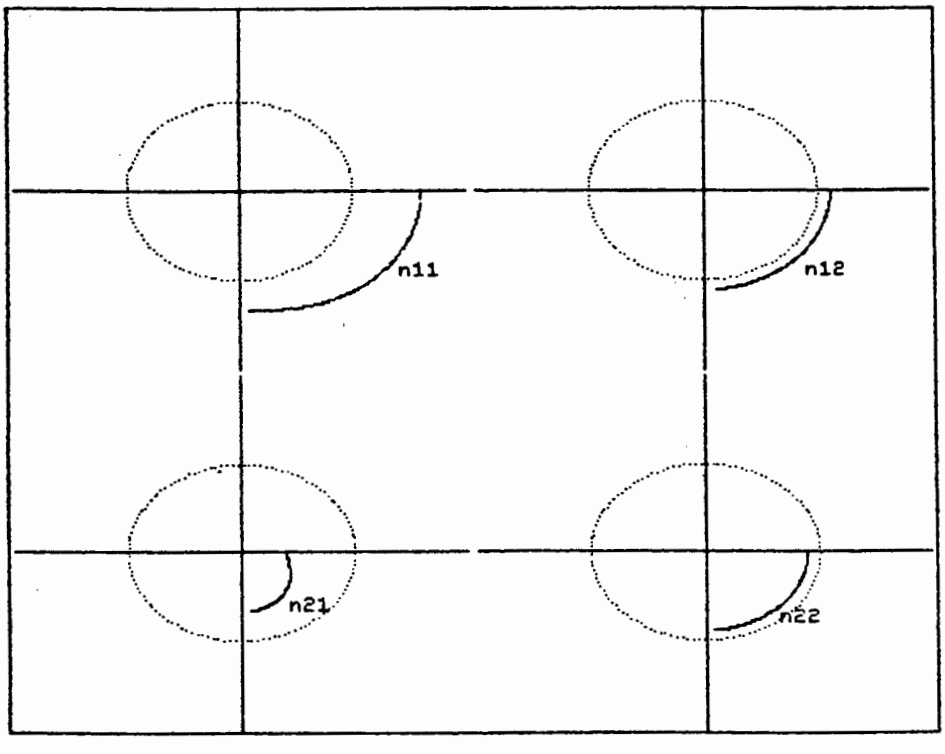

Figure 4.7. Array of polar plots of the elements of the N(s) matrix of the system represented by (4.22).

longer dominant. The element $g_{12}(s)$ has the opposite behaviour, it is only dominant at high frequencies as is shown by the plot of $m_{12}(s)$ which is inside the 0.5 circle at low frequencies and goes outside the circle at high frequencies. Clearly this matrix is row diagonally dominant only at low frequencies.

Figure 4.7 shows the polar plots for the N(s) matrix. This diagram shows that, without scaling, this matrix cannot be rearranged to be column dominant. This is because the plots of both $n_{11}(s)$ and $n_{12}(s)$ lie outside the 0.5 circle, hence $g_{11}(s)$ and $g_{12}(s)$ are both dominant on their columns and hence the matrix cannot be rearranged so that both elements lie on the diagonal.

Figures 4.8 and 4.9 show the effect of adding a dead time. The system considered is shown in (4.23).

$$G(s) = \begin{bmatrix} \dfrac{1\ e^{-0.8s}}{s+1} & \dfrac{4}{s+2} \\ \dfrac{4}{s+1} & \dfrac{10}{s+2} \end{bmatrix} \qquad (4.23)$$

The dead time does not affect the magnitude of the $m_{11}(s)$ or $n_{11}(s)$ terms but it does change the angle. Hence addition of a time delay causes the polar plot to bend around the origin but does not alter the dominance of the element.

As a more complex example consider the following 3x3 plant model which contains both time delays and second order transfer functions. The frequency range considered was 0 to 1 hz.

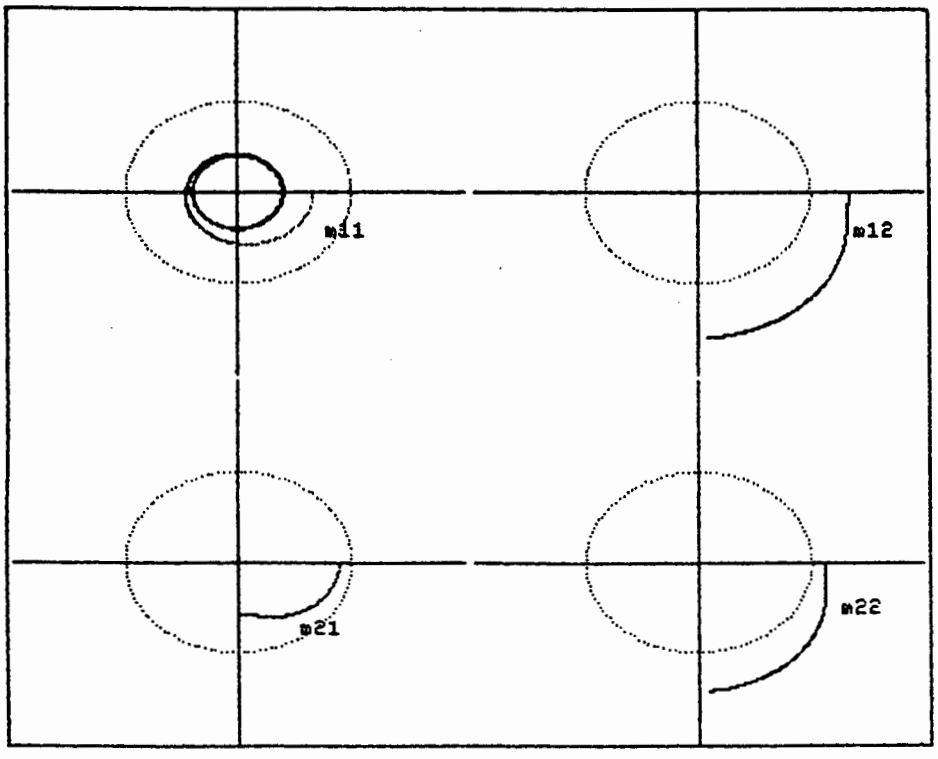

Figure 4.8. Array of polar plots of the elements of the M(s) matrix of the system represented by (4.23).

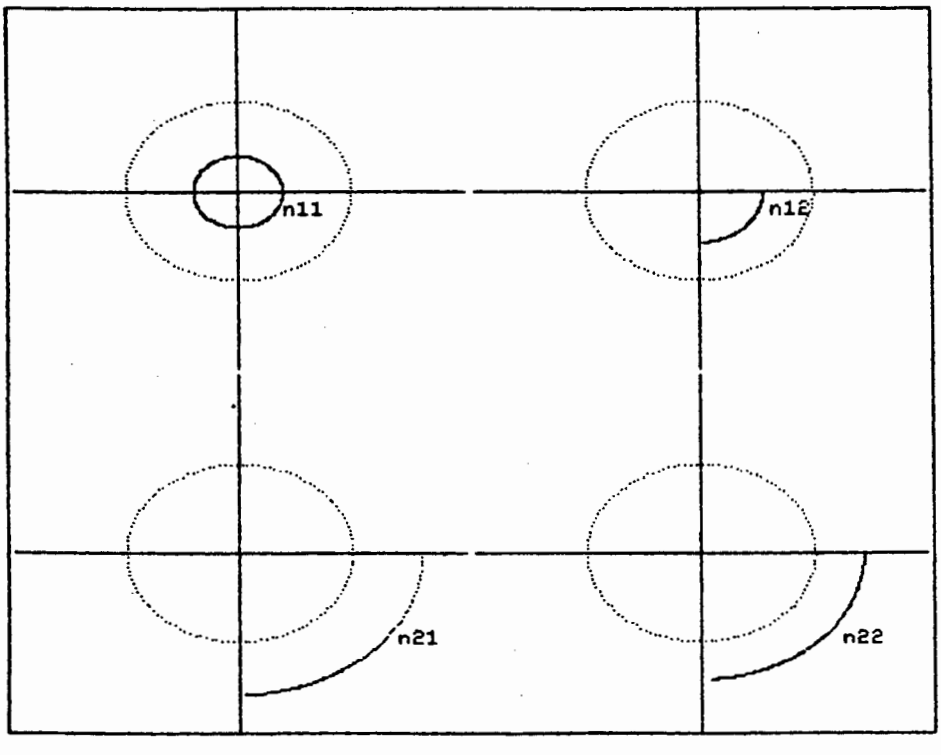

Figure 4.9. Array of polar plots of the elements of the N(s) matrix of the system represented by (4.23).

$$G(s) = \begin{bmatrix} \dfrac{0.01}{36s^2+12s+1} & \dfrac{2.0\ e^{-2s}}{800s^2+60s+1} & \dfrac{0.001\ e^{-s}}{7s+1} \\ \dfrac{4.0\ e^{-4s}}{300s^2+56s+1} & \dfrac{500.0\ e^{-0.5s}}{600s^2+50s+1} & \dfrac{6.0\ e^{-60s}}{400s+1} \\ \dfrac{10.0\ e^{-2s}}{280s^2+47s+1} & \dfrac{30000.0\ e^{-2s}}{240s^2+46s+1} & \dfrac{20.0\ e^{-3s}}{14s+1} \end{bmatrix} \qquad (4.24)$$

From Fig. 4.10 it is immediately evident that without scaling this matrix cannot be rearranged to be row diagonally dominant. The plots of the $m_{ij}(s)$ elements show that all of the dominant elements lie in the second column and hence cannot be placed on the diagonal by row-column rearrangement. Examination of Fig. 4.11, the polar plots for N(s), shows similarly that the matrix cannot be rearranged to be column dominant.

As a final example consider the matrix in (4.25). This matrix is the system model of a furnace heated by four sets of heating coils and is taken from [12].

$$G(s) = \begin{bmatrix} \dfrac{1.0}{1+4s} & \dfrac{0.7}{1+5s} & \dfrac{0.3}{1+5s} & \dfrac{0.2}{1+5s} \\ \dfrac{0.6}{1+5s} & \dfrac{1.0}{1+4s} & \dfrac{0.4}{1+5s} & \dfrac{0.35}{1+5s} \\ \dfrac{0.35}{1+5s} & \dfrac{0.4}{1+5s} & \dfrac{1.0}{1+4s} & \dfrac{0.6}{1+5s} \\ \dfrac{0.2}{1+5s} & \dfrac{0.3}{1+5s} & \dfrac{0.7}{1+5s} & \dfrac{1.0}{1+5s} \end{bmatrix} \qquad (4.25)$$

In [12] the Nyquist array was plotted and Gershgorin circles superimposed on the plots of the diagonal elements. From this analysis it was concluded that the the matrix was not diagonally dominant. Figures 4.12 and 4.13 respectively, show the polar plots of the M(s) and N(s) matrices for (4.25). The frequency range is 1 to 0 hz. These plots are seen to be identical because of the symmetrical nature of the matrix. It can be seen that there are no dominant elements on any of the rows or columns since all the plots of the $m_{ij}(s)$ and $n_{ij}(s)$ elements lie inside the 0.5 circles. Hence the conclusion obtained in [12] has been confirmed using this method. Further the current arrangement of the matrix is seen to have the most nearly dominant elements on its diagonal since the diagonal elements of M(s) and N(s) have polar plots that are very close to the 0.5 circle.

4.3 SCALING A MATRIX FOR DIAGONAL DOMINANCE

A serious problem associated with the use of the diagonal dominance structure is that it is not invariant under input-output scaling. In other words by scaling either the inputs or the outputs, or both, of G(s) the dominance of the diagonal elements of the matrix may be changed. Scaling the inputs is equivalent to postmultiplying G(s) by a matrix with the same order as G(s) while scaling the outputs is equivalent to premultiplying G(s) by another matrix, also with the same order as G(s). Usually the scaling matrices are taken to be diagonal and their elements are assumed to be independent of frequency. This is the same as multiplying each input or output signal by a constant.

The frequency independence of the scaling matrices is introduced to simplify the problem of selecting appropriate scaling elements, however it may prove impossible to select scaling elements that will make G(s) diagonally dominant for

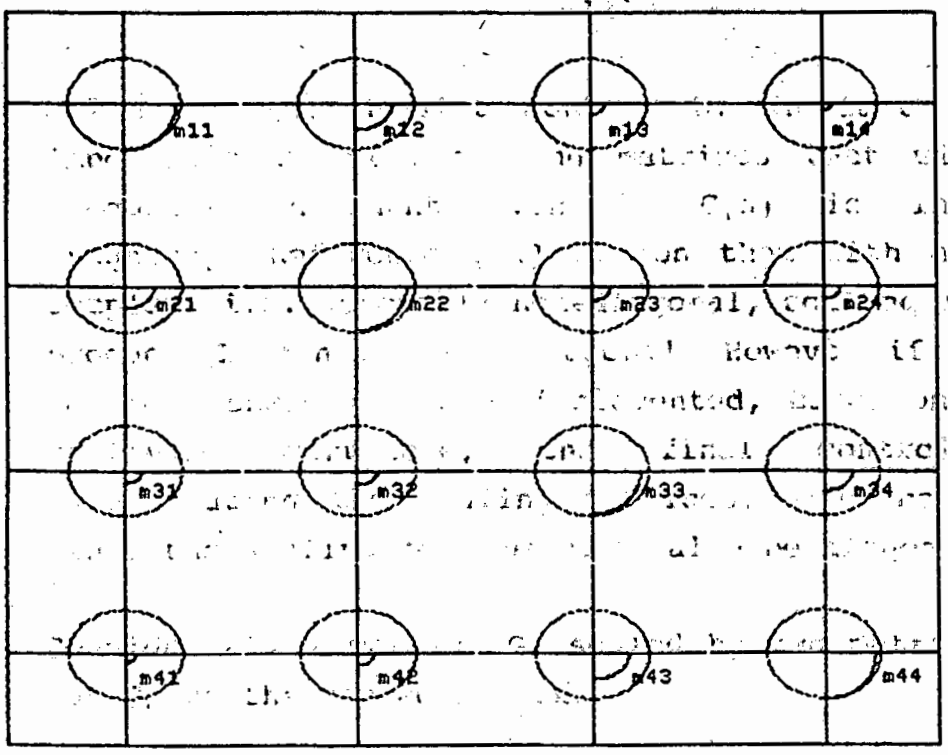

Figure 4.12. Array of polar plots of the elements of the M(s) matrix of the system represented by (4.25).

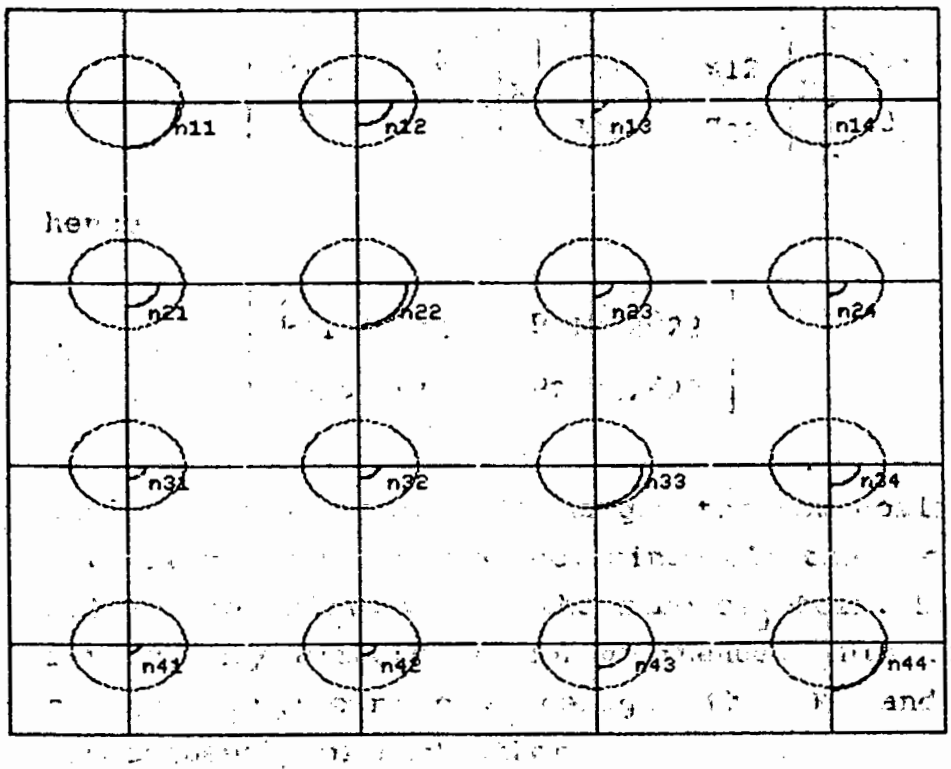

Figure 4.13. Array of polar plots of the elements of the N(s) matrix of the system represented by (4.25).

of $G_s(s)$ will not, in general, be the same as the eigenvalues of G(s). Thus if $G_s(s)$ is diagonally dominant and the closed loop system can be shown to be stable, with theorems described in [12], this does not imply the closed loop stability of the unscaled system. This means that the designer must implement the scaling in the actual plant. In chapter two and in [1], it was shown that if P is chosen to be S^{-1} then the eigenvalues of $G_s(s)$ are the same as those of G(s) and hence stability of the scaled system implies stability of G(s). This concept is referred to as generalised diagonal dominance. Now the designer does not have to implement the scaling in the actual plant in order to ensure stability.

As noted above P and S can be found independently of each other. But suppose GS is diagonally row dominant, since P does not change row dominance define P as S^{-1}. Now $S^{-1}GS$ will be row diagonally dominant and if the closed loop system containing the scaled matrix is stable then so too will be the unscaled closed loop system. Hence if P or S can be chosen such that either PG or GS are diagonally dominant then the matrix can be made generalised diagonally dominant and vise versa.

The problem remains as to how to select the scaling matrices to make the scaled matrix diagonally dominant. In [1] Limebeer uses the frequency dependent terms of the Perron-Frobenius eigenvector of a comparison matrix as the terms of S, see chapter two section 2.3.3. Work has also been done by Hulbert, [20] on finding appropriate scaling matrices using linear programming techniques.

4.3.1 A Test for the Feasibility of Scaling

The theory developed in this section is aimed at determining whether or not a matrix G can be scaled by another diagonal

matrix to be diagonally dominant. The object then is not so much as to find the scaling matrix that will make G diagonally dominant but to discover the conditions under which such a matrix exists. Only the single frequency case, i.e. G not frequency dependent, is initially considered.
In the derivations below let |G| be the matrix where each element is equal to the absolute value of the corresponding element in G.

Consider a 2x2 matrix, G, multiplied by a positive, diagonal, input scaling matrix S. For row dominance the following must hold

$$|g_{11}|s_{11} > |g_{12}|s_{22} \tag{4.28}$$

$$|g_{22}|s_{22} > |g_{21}|s_{11} \tag{4.29}$$

hence

$$|g_{11}|/|g_{12}| > s_{22}/s_{11}$$

and

$$|g_{21}|/|g_{22}| < s_{22}/s_{11}$$

which implies that

$$|g_{11}g_{22}| > |g_{12}g_{21}| \tag{4.30}$$

If the case where G is a 3x3 matrix is now considered three inequalities must hold

$$|g_{11}|s_{11} > |g_{12}|s_{22} + |g_{13}|s_{33} \tag{4.31}$$

$$|g_{22}|s_{22} > |g_{21}|s_{11} + |g_{23}|s_{33} \tag{4.32}$$

$$|g_{33}|s_{33} > |g_{31}|s_{11} + |g_{32}|s_{22} \tag{4.33}$$

these can be manipulated to give

$$s_{11} > (|g_{12}|s_{22} + |g_{13}|s_{33})/|g_{11}| \qquad (4.34)$$

$$s_{11} < (|g_{22}|s_{22} - |g_{23}|s_{33})/|g_{21}| \qquad (4.35)$$

$$s_{11} < (|g_{33}|s_{33} - |g_{32}|s_{22})/|g_{31}| \qquad (4.36)$$

from (4.34) and (4.35) we have

$$(|g_{22}|s_{22} - |g_{23}|s_{33})/|g_{21}| > (|g_{12}|s_{22} + |g_{13}|s_{33})/|g_{11}|$$

provided the principle minor $|g_{11}g_{22}|-|g_{12}g_{21}|$ of $|G|$ is positive then

$$(|g_{11}g_{22}|-|g_{12}g_{21}|)/(|g_{11}g_{23}|+|g_{13}g_{21}|) > s_{33}/s_{22} \qquad (4.37)$$

Similarly from (4.34) and (4.36)

$$(|g_{33}|s_{33} - |g_{32}|s_{22})/|g_{31}| > (|g_{12}|s_{22} + |g_{13}|s_{33})/|g_{11}|$$

provided the principle minor $|g_{11}g_{33}|-|g_{13}g_{31}|$ of $|G|$ is positive then

$$s_{33}/s_{22} > (|g_{11}g_{32}|-|g_{12}g_{31}|)/(|g_{11}g_{33}|-|g_{13}g_{31}|) \qquad (4.38)$$

combining (4.37) and (4.38) and cross multiplying gives

$$|g_{11}g_{22}g_{33}| > |g_{11}g_{32}g_{23}|+|g_{13}g_{21}g_{32}|+|g_{12}g_{21}g_{33}|+ |g_{12}g_{31}g_{23}|+|g_{13}g_{22}g_{31}| \qquad (4.39)$$

The terms in the inequalities (4.30) and (4.39) are the absolute values of the terms that appear in the expansion of the determinants of a 2x2 $|G|$ and a 3x3 $|G|$ respectively. If the terms in the expansion of a determinant are all added instead of being alternately added and subtracted, as in the

case of the determinate, then the resulting value is known as the permanent, or per, of the matrix. The product of the diagonal elements is known as the trace of the matrix. Hence both (4.30) and (4.39) can be rewritten as

$$\text{trace}(|G|) > \text{per}(|G|) - \text{trace}(|G|)$$

or

$$\text{trace}(|G|) > 0.5\text{per}(|G|) \qquad (4.40)$$

Thus for a 2x2 or 3x3 matrix the following holds

If all the principle minors of the matrix $|G|$ are positive and the inequality of (4.40) holds then there exists a matrix S such that GS is diagonally dominant.

4.3.2 Example

Consider the following matrix

$$G = \begin{bmatrix} 10^{-2} & 10^{-3} & 2 \\ 4 & 6 & 500 \\ 10 & 20 & 30000 \end{bmatrix} \qquad (4.41)$$

The principle minors of $|G|$ are all positive so the inequality of (4.40) must be tested.

$$0.5\text{per}(|G|) = 1152.5$$

$$\text{trace}(|G|) = 1800$$

Hence $\text{trace}(|G|) > 0.5\text{per}(|G|)$ and the inequality is satisfied. This implies that there exists at least one matrix S such that GS is diagonally dominant.

To find such a matrix the inequalities used to derive (4.40) can be used. First a value for s_{33} is selected, say 0.1. From (4.37) and (4.38) bounds for s_{22} can be found

$$133.3 > s_{22} > 23.3.$$

Now if s_{22} is taken to be 100 the range of possible s_{11} values can be found from (4.34), (4.35) and (4.36).

$$100 > s_{11} > 30$$

Select s_{11} to be 50. The matrix S then is

$$S = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \qquad (4.42)$$

and GS

$$GS = \begin{bmatrix} 0.5 & 0.1 & 0.2 \\ 200 & 600 & 50 \\ 500 & 2000 & 3000 \end{bmatrix} \qquad (4.43)$$

which is row diagonally dominant.

To expand this method to frequency varying systems would require the principle minors, the trace of the matrix and the per of the matrix to be calculated at a number of different frequencies in the range being considered. If the requirements for dominance were satisfied at each frequency then at each frequency a matrix S would exist such that GS is diagonally dominant. This ensures that the matrix is generalised diagonally dominant but the different scalar matrices will in general be necessarily to make the matrix diagonally dominant at different frequencies.

For matrices of higher order than three this test is difficult to apply because of problems in calculating per(|G|). The number of terms in per(|G|) increases

factorially with the order of G and so some numerical method of calculating the permanent would be necessary if this test were to be of practical value.

4.4 SUMMARY

Modification of the well known Gershgorin circle theorem has led to a fast, and conceptually simple method of rearranging a frequency dependent matrix to be diagonally dominant if this is possible. The representation used allows the designer to see quickly which elements are dominant on their rows or columns and to see if the matrix can be rearranged to be diagonally dominant. The method suffers from the fact that it does not take the possibility of matrix scaling into account.

Matrix scaling was also discussed in this chapter. A method has been developed to check if a matrix G of scalars can be made row diagonally dominant using the scaling GD where D is a diagonal matrix of scaling elements. The method is limited to matrices with order less than or equal to three by numerical restraints and was not extended to the frequency domain.

CHAPTER 5

DECENTRALISED CONTROLLER DESIGN USING AN INTERACTION MEASURE

5.1 INTRODUCTION

In the previous chapter a technique was developed to allow the control engineer to determine if the system he is dealing with can be rearranged to be diagonally dominant. In practice it often not possible to rearrange the plant system matrix to be diagonally dominant. This is particularly the case for large plants where designing a compensator to make the plant diagonally dominant is difficult and time consuming. Notice that it may still be possible to distribute control so that all the system controllers are independent single variable controllers and the system is stable. However if the system is not diagonally dominant then alternative stability criteria must be developed.

A further problem with diagonal dominance is that the subsystems to be controlled are restricted to single-input single-output systems. In terms of the plant model this means that the system is partitioned so that the on - diagonal submatrices have order one. In chapter three it was seen that if the system is block diagonal or block triangular then the on-diagonal subsystems may be independently controlled, subject to some restrictions. In the case of such systems the order of the on-diagonal blocks was not limited to one. In general then, for distributed control, the designer should be able to partition his system without restrictions on the sizes of the subsystems to be independently controlled.

If the designer is not restricted to 1x1 controlled subsystems then two further problems occur. Firstly the designer must select one or more row-column and partitioning arrangements that are likely to lead to satisfactory distributed control. Having selected a particular row-column arrangement and matrix partitioning the designer will then design controllers to stabilise each on-diagonal subsystem independently. Once this is done the designer must have some means of checking to see if the system as a whole is stable, remembering that if the on-diagonal subsystems are stable this does not necessarily imply that the combined system is stable.

In both cases there is a need for an interaction measure that will indicate how tightly the on-diagonal subsystems are coupled. The interaction measure must indicate if a particular subdivision of the system is suitable for distributed control, i.e. is likely to yield a stable system when the on-diagonal subsystems are independently stabilised. In addition the interaction measure should be linked to stability theorems that allow the designer to check if the controllers he has designed to stabilise the independently controlled subsystems will in fact stabilise the system as a whole.

The most suitable interaction measure currently available is the Perron-Frobenius eigenvalue of a comparison matrix. This interaction measure was introduced in chapter two. The measure gives a useful indication of the suitability of a particular partitioning for distributed control. Further it is linked to stability theorems that allow the stability of a system under distributed control to be checked. The work in this chapter is based on this interaction measure.

Before the interaction measure can be utilised however the designer must select row-column arrangements and

partitionings that are likely to yield low values of the measure. The other alternative is to try every possible row-column arrangement and partitioning to see which has the lowest value of the interaction measure. This is time consuming for even small systems and completely impractical for large systems. In this chapter an attempt is made to develop a systematic method of input-output pairing and partitioning to yield low values of the interaction measure.

5.2 PERRON-FROBENIUS EIGENVALUE INTERACTION MEASURE

In this section a derivation will be given for the Perron-Frobenius eigenvalue interaction measure, given symbol r. The proof given is largely based on a formulation developed by Nwokah in [5]. Nwokah uses singular values in his paper while general induced matrix norms are used in this development following Siljak and coauthors in [6]. An identical set of results has been developed in [6], based on the concept of Quasi-Block Diagonal Dominance. It has been decided to use Nwokah's approach as opposed to Siljak's development because the proofs presented by Nwokah are slightly simpler and are easier to understand. Both Nwokah and Siljak use the theory of M matrices to derive their results and so a brief introduction to M matrices is given here.

An M matrix is a real square matrix with nonpositive off diagonal elements, i.e. the off diagonal elements are zero or negative, and positive principle minors . Such matrices have a number of special properties, for example in [15] there are listed fifty different properties such as the possession of positive principle minors. Reference [15] contains a very detailed section on M matrices and most of the results obtained by Nwokah are based on proofs from this reference.

5.2.1 Mathematical Preliminaries

Let Z be a complex $n_z x n_z$ matrix partitioned into submatrices Z_{ij} such that the matrix representation of Z is

$$Z = \begin{bmatrix} Z_{11} & \cdots & Z_{1n} \\ \vdots & Z_{ii} & \\ Z_{n1} & & Z_{nn} \end{bmatrix} \qquad (5.1)$$

Where each on-diagonal Z_{ii} is square and $\sum_{i=1}^{n} n_i = n_z$.

Also if $\|x\|$ is a vector norm then the corresponding matrix norm is defined as

$$\|Z_{ij}\| = \sup \frac{\|Z_{ij}.x\|}{\|x\|} \qquad (5.2)$$

Assume that the diagonal blocks Z_{ii} are all non-singular.

Define the block diagonal matrix

$$P = \text{Block Diag}(Z_{11}, Z_{22}, \ldots, Z_{nn}) \qquad (5.3)$$

Split Z into:

$$Z = P + Q \qquad (5.4)$$

where

$$Q = Z - P \qquad (5.5)$$

Then $$ZP^{-1} = I + QP^{-1} \tag{5.6}$$

Z is non-singular provided the spectral radius of QP^{-1} is less than one, [5].

Now define a real non-negative nxn matrix $\mathbf{B} = (b_{ij})$ such that

$$b_{ij} = \begin{cases} b_{ii} = 0, & i = j \\ b_{ij} = |Q_{ij}P_j^{-1}|, & i \neq j \end{cases} \tag{5.7}$$

Write r(B) for the Perron Frobenius eigenvalue of B.

The Perron-Frobenius eigenvalue of a non-negative matrix **B** is the real positive eigenvalue of B such that r(B) is greater than or equal to the absolute value of any other eigenvalue of B. The Perron-Frobenius eigenvalue theorem was stated in chapter two, theorem 2.8. Reference [15] has a detailed discussion of this theorem.

The following result from [15] constitutes a regularity condition for Z, i.e. a condition that ensures Z is non-singular.

Lemma 1 , [5]

(spectral radius of QP^{-1}) < (spectral radius of B) = r(B).

Hence any condition that ensures r(B) < 1 also ensures that Z is non-singular.

Then $R = QP^{-1}$ is given by

$$R_{ij} = \begin{cases} R_{ii} = 0 & i = j \\ R_{ij} = G_{ij}K_j(I + G_{jj}K_j)^{-1} & i \neq j \end{cases} \qquad (5.14)$$

Now define $B = (b_{ij})$ as

$$b_{ij} = \begin{cases} b_{ii} = |R_{ii}| = 0 & i = j \\ b_{ij} = |R_{ij}| & i \neq j \end{cases} \qquad (5.15)$$

By Lemma 1 Z is non-singular for all s on the Nyquist contour provided $r(B) < 1$ for every s on the Nyquist contour.

Now from (5.15)

$$|R_{ii}| = 0$$

and

$$|R_{ij}| = |G_{ij}G_{jj}^{-1}G_{jj}K_j(I_j + G_{jj}K_j)^{-1}|$$

$$\leq |G_{ij}G_{jj}^{-1}| \cdot |G_{jj}K_j(I_j + G_{jj}K_j)^{-1}| \qquad (5.16)$$

hence from (5.8) and (5.9)

$$|R_{ij}| \leq C_{ij}.d_j \qquad (5.17)$$

and thus

$$B \leq CD \text{ elementwise.} \qquad (5.18)$$

For non-negative matrices the Perron Frobenius theorem, [15], shows that if $B \leq CD$ then $r(B) \leq r(CD)$ hence $r(CD) < 1$ implies $r(B) < 1$ which in turn implies that the system is closed loop stable and concludes the proof.

It is not clear from the above result as to how the controller affects the structure of the matrix. The following results are developed to allow Nyquist array methods to be used, and to give an interaction measure that is independent of the controller matrix, K(s).

Corollary 5.1

From the theory of M matrices in [15] if $r(CD) < 1$ then the matrix $[I-CD]$ is an M matrix. Another property of such matrices is that multiplication by a diagonal matrix of positive numbers leaves the M matrix property unchanged and hence the matrix $[(I-CD)D^{-1}] = [D^{-1}-C]$ is also an M matrix.

Now for any non-negative matrix C the matrix $[(r(C)+\tau)[I-C]]$ is an M matrix for any $\tau > 0$ no matter how small, [15]. Hence $[D^{-1}-C]$ is an M matrix provided that $d_j^{-1} > r(C)$ for $j = 1, 2, \ldots, m$. If $d_j^{-1} > r(C)$ then $[I-CD]$ is an M matrix and thus $r(CD) < 1$. This in turn implies that the system is closed loop stable.

This corollary gives a sufficient but *not* necessary condition for the stability of the closed loop composite system. Hence even if $d_j^{-1} > r(C)$ the system may still be stable.

As an illustration if G(s) is partitioned into a 2x2 composite matrix, and controlled by a 2x2 diagonal matrix, then C is

$$C = \begin{bmatrix} 0 & |G_{12}G_{22}^{-1}| \\ |G_{21}G_{11}^{-1}| & 0 \end{bmatrix} \tag{5.19}$$

The terms of the matrix D will be

$$d_1 = |G_{11}K_1(I_1 + G_{11}K_1)^{-1}| \tag{5.20}$$

$$d_2 = |G_{22}K_2(I_2 + G_{22}K_2)^{-1}| \tag{5.21}$$

If the Perron-Frobenius eigenvalue of the matrix in (5.19) is r(C) then if $d_1^{-1} > r(C)$ and $d_2^{-1} > r(C)$ the system as a whole will be closed loop stable, provided of course that the initial assumptions still hold.

A restriction that applies to C is that C must be irreducible, that is no permutation matrix, P, may exist such that P^TCP is block triangular. This restriction exists because the Perron-Frobenius theorem applies only to irreducible matrices.

Graph theoretic tests exist to check for reducibility, see [1] for example. If C is reducible, i.e. not irreducible, then the test for stability must be adjusted. In [6] a method for dealing with a reducible C is given. C must first be rearranged to be block triangular with square on-diagonal blocks, this must be possible if C is reducible. Now C will be a composite matrix of matrix norms with order l and $l < m$. The on-diagonal blocks are C_{pp} $p=1, 2,\ldots,l$. The value of $r(C_{pp})$ is now calculated for each submatrix C_{pp} $p=1, 2,\ldots,l$. The new stability requirement is that $d_j^{-1} > r(C_{pp})$ for all $j \in C_{pp}$ $p=1,2,\ldots,l$. The easiest way of understanding this is by means of an example.

The matrix in (5.22) is a C matrix that has been rearranged and partitioned to be block triangular.

$$C = \left[\begin{array}{c|cc|c} |G_{12}G_{22}^{-1}| & 0 & 0 & 0 \\ \hline 0 & |G_{21}G_{11}^{-1}| & |G_{23}G_{33}^{-1}| & 0 \\ |G_{42}G_{22}^{-1}| & |G_{41}G_{11}^{-1}| & |G_{43}G_{33}^{-1}| & 0 \\ \hline |G_{32}G_{22}^{-1}| & |G_{31}G_{11}^{-1}| & 0 & |G_{34}G_{44}^{-1}| \end{array}\right] \qquad (5.22)$$

C has now been partitioned into a 3x3 composite matrix. C therefore has three on-diagonal submatrices C_{11}, a 1x1 submatrix, C_{22}, a 2x2 submatrix and C_{33} also a 1x1 submatrix. Now the stability criterion is as follows if $d_1^{-1} > r(C_{22})$, $d_2^{-1} > r(C_{11})$, $d_3^{-1} > r(C_{22})$ and $d_4^{-1} > r(C_{33})$ then the system as a whole will be closed loop stable.

In [6] the authors develop a theorem that allows Nyquist array type design methods to be applied to distributed controller designs.

Theorem 5.2

Consider the same partitioned system, G(s), as described above. Let the assumptions given previously still hold. In addition let

$$Q_{jj} = G_{jj}K_j \qquad (5.23)$$

and

$$F = \text{Diag}(F_1, F_2, \ldots, F_m) \qquad (5.24)$$

where

$$F_j = \text{diag}(f_1, f_2, \ldots .fm_j) \tag{5.25}$$

each F_j having dimensions $m_j x m_j$.

For a given Q_{jj} let the row and column indices be l and r respectively i.e. Q_{jj} will be an $m_j x m_j$ submatrix with elements q_{lr}.

Suppose C is as represented in (5.8) . Given all s $\in$ D where D is the Nyquist contour assume the following.

(i) For each Q_{jj} submatrix

$$1 + q_{rr}f_r \neq 0 \tag{5.26}$$

(ii) K_j is nonsingular for all j = 1, 2,...,m

(iii) For each Q_{jj} submatrix

$$|q_{rr}f_r[1+q_{rr}f_r]^{-1}| < [(1+r(C))|T_{jj}|-1]^{-1} \tag{5.27}$$

where T_{jj} is the matrix with elements t_{lr} such that

$$t_{lr} = \frac{|q_{lr}|}{|q_{rr}|} \tag{5.28}$$

then $d_j^{-1} > r(C)$ for all j and the system as a whole will be stable if the individual on-diagonal subsystems are closed loop stable.

The proof of this theorem is complex and will not be given here; it can be found in [6].

An alternative way of checking stability to using the inequality (5.27) is to ensure that the following inequality is satisfied for all j.

$$\|Q_{jj}F_j(I+Q_{jj}F_j)^{-1}\| < r(C) \qquad (5.29)$$

Both these expressions are potentially useful though inequality (5.27) is best suited to Nyquist array design methods. It allows the designer to ensure that overall stability is maintained while designing controllers for the individual loops of each on-diagonal subsection.

Note that the values of the c_{ij} terms, and hence r(C), are not affected by the choice of the K_j blocks because of the identity that

$$c_{ij} = \|G_{ij}.G_{jj}^{-1}\| = \|G_{ij}K_j(G_{jj}K_j)^{-1}\| = \|Q_{ij}Q_{jj}^{-1}\|$$

The design procedure would be as follows. Initially the designer selects a partitioning for the system matrix G(s). In doing so the designer is selecting those subsystems that are to be controlled independently of each other. The designer would now design controllers for each $G_{jj}(s)$ submatrix independently using the Nyquist array design technique for example. For each $G_{jj}(s)$ subsystem the designer would design a $K_{jj}(s)$ controller and a F_j feedback matrix. For each of these designs the corresponding elements of $Q_{jj}(s) = G_{jj}(s)K_{jj}(s)$ and F_j must satisfy inequality (5.27). If the inequality is satisfied for each on-diagonal subsystem then, provided each of these systems is closed loop stable, the system as a whole will be stable. Notice again that because the condition is only sufficient if the inequality (5.23) is not satisfied the system may still be stable.

5.2.3 Using r(C(s)) as a Criterion for Selecting Subsystems

The fact that r(C(s)) remains invariant with K(s), provided that K(s) is diagonal, means that r(C(s)) provides a useful interaction measure. In general the lower the value of r(C(s)) the less coupling there is between the on-diagonal subsystems and the more suited the system will be to distributed control based on those subsystems.

The eigenvalue r(C(s)) is used to give a measure of the suitability of the current partitioning and row-column ordering for distributed control. If the inequality

$$d_j^{-1} > r(C(s)) \tag{5.30}$$

is satisfied for all j then the system will be stable provided the assumptions given in the last section still hold. Hence the smaller the value of r(C(s)) the easier it is to satisfy the inequality when a controller is being designed. In particular r(C(s)) for a block diagonal or block triangular system is always zero. This confirms that systems with these structures are particularly suited to distributed control. In general then the designer would rearrange the rows and columns and partitioning of the matrix G(s) until he obtained the lowest value of the interaction measure r(C(s)).

Note that (5.30) can be rewritten as

$$d_j r(C(s)) < 1 \tag{5.31}$$

This must be satisfied for all j, this will be the case if the inequality (5.32) is satisfied.

$$d_{jmax} \cdot r(C(s)) < 1 \tag{5.32}$$

where $d_{jmax}=\max_j \left|G_{jj}(s)K_{jj}(s)(I_j + G_{jj}(s)K_{jj}(s))^{-1}\right|$ (5.33)

As an example of the use of r(C(s)) as an interaction matrix a specific system is now considered. The effects of changing the row-column ordering and the matrix partitioning on r(C(s)) and d_{jmax}.r(C(s)) are studied and the closed loop behaviour of the system seen by means of simulations.

Programs have been written to calculate r(C(s)) and d_{jmax}.r(C(s)). The results generated in this section have been produced by these programs. For more details on the programs see chapter six.

The system considered here is represented by a 4x4 matrix model of first order transfer functions with no time delays and is taken from [12]. Three different row-column arrangements will be considered and for each matrix four different partitionings will be examined.

The first row-column arrangement is the same as that used in [12].

$$G(s) = \begin{bmatrix} \frac{1.0}{1+4s} & \frac{0.7}{1+5s} & \frac{0.3}{1+5s} & \frac{0.2}{1+5s} \\ \frac{0.6}{1+5s} & \frac{1.0}{1+4s} & \frac{0.4}{1+5s} & \frac{0.35}{1+5s} \\ \frac{0.35}{1+5s} & \frac{0.4}{1+5s} & \frac{1.0}{1+4s} & \frac{0.6}{1+5s} \\ \frac{0.2}{1+5s} & \frac{0.3}{1+5s} & \frac{0.7}{1+5s} & \frac{1.0}{1+4s} \end{bmatrix} \quad (5.34)$$

The four different partitionings to be considered are defined as follows:

Let G_{ij} represent a block of transfer functions. For each partitioning the matrix of Gij terms will be given along with the dimensions of the square G_{jj} blocks.

(i)

$$G(s) = \begin{bmatrix} G_{11} & G_{12} & G_{13} & G_{14} \\ G_{21} & G_{22} & G_{23} & G_{24} \\ G_{31} & G_{32} & G_{33} & G_{34} \\ G_{41} & G_{42} & G_{43} & G_{44} \end{bmatrix} \qquad (5.35)$$

$n_{11}=n_{22}=n_{33}=n_{44}=1$ (where n_{jj} is the order of block G_{jj}).

(ii)

$$G(s) = \begin{bmatrix} G_{11} & G_{12} & G_{13} \\ G_{21} & G_{22} & G_{23} \\ G_{31} & G_{32} & G_{33} \end{bmatrix} \qquad (5.36)$$

$n_{11}=1$ $n_{22}=1$ $n_{33}=2$

(iii)

$$G(s) = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \qquad (5.37)$$

$n_{11}=1$ $n_{22}=3$

(iv)

$$G(s) = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \qquad (5.38)$$

$$n_{11}=2 \quad n_{22}=2$$

The matrix of (5.34) was partitioned in each of the four ways given above and for each partitioning r(C(s)) and d_{jmax}.r(C(s)) were calculated over a frequency range from 0 to 5 hz. Fig. 5.1 shows the values of r(C(s)) vs frequency for each partitioning. The curve for a particular partitioning can be identified by the roman numeral next to the curve which corresponds to the numbering given in (5.35), (5.36), (5.37) and (5.38). The line r(C(s))=1.0 is drawn in on the graph of r(C(s)) vs frequency. In general a value of r(C(s)) much less than one is likely to ensure that inequality (5.30) is satisfied.

Firstly notice that three out of the four partitionings of G(s) have values of r(C(s)) that are always less than one. This suggests that the particular row-column orderings used here are good ones for distributed control since all of the partitionings yield low values of the interaction index. The lowest values for r(C(s)) are obtained when the matrix is partitioned so as to have two 2x2 on-diagonal blocks. The highest values of r(C(s)) are obtained when the on-diagonal blocks are all 1x1 blocks, however even this value is relatively low being close to one. The above suggests that the controller structure should be block diagonal with two 2x2 independently controlled subsystems on the diagonal, each designed to compensate the equivalent block in G(s). However ,as Nwokah points out in [5], the 2x2 blocks on the diagonal of G(s) are diagonally dominant and hence each loop can be independently controlled. Thus the final controller

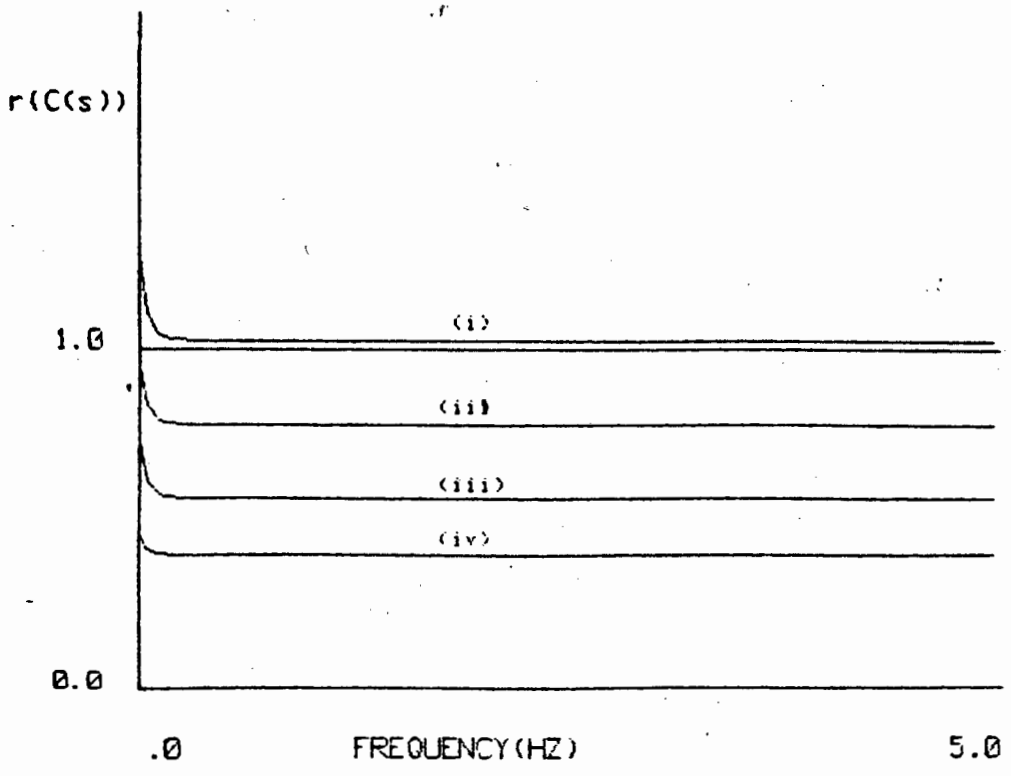

Figure 5.1. Plots of r(c(s)) vs frequency for different partitionings of the system represented by (5.34).

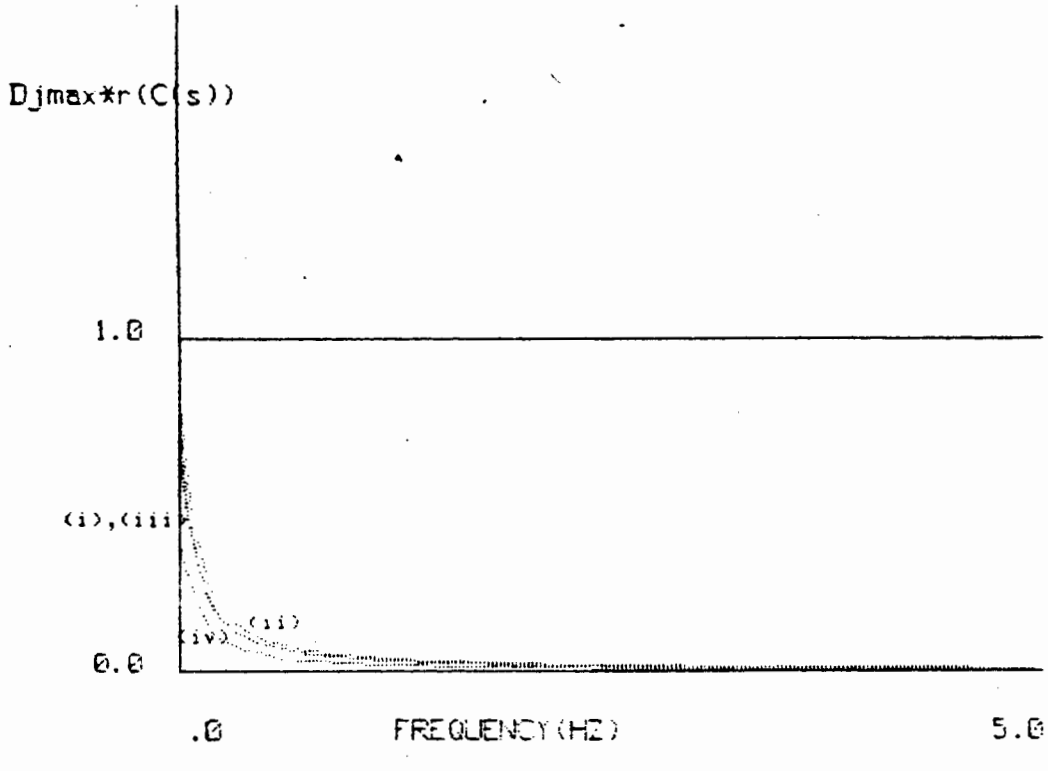

Figure 5.2. Plots of djmax*r(C(s)) vs frequency for different partitionings of the system represented by (5.34)

matrix will be a diagonal matrix with 1x1 on-diagonal blocks which is the result obtained by Rosenbrock in [12] using a different design procedure. Note that the controller must be designed so that inequality (5.30) is satisfied. This analysis suggests that it should be possible to stabilise this system with four single variable controllers.

Figure 5.2 shows plots of $d_{jmax}.r(C(s))$ vs frequency for the various partitionings. This set of curves gives information as to the closed loop stability of G(s) i.e. the stability of the system $G(s).(I + G(s))^{-1}$.

The inequality (5.32) can only be used to test for composite stability if the assumptions under which theorems 5.1 and 5.2 were derived apply to the system under study. In the case of the system in (5.34) all of these assumptions are valid.

All of the curves lie below the $d_{jmax}.r(C(s))=1.0$ line. This indicates that, for the partitionings studied here, provided the on-diagonal blocks of the partitioned G(s) are closed loop stable then G(s) will be closed loop stable. In particular if the blocking used is such that all the on-diagonal blocks are 1x1 then clearly all the on-diagonal elements are stable. Thus the theory predicts that such a system will be closed loop stable in response to setpoint changes.

The closed loop response of G(s) to step inputs was simulated and is shown in Figure 5.3. The setpoints R1, R2, R3 and R4 were each stepped in turn. The next step being made when the outputs had settled down from the previous step. In each case the setpoints were given a unit step from 0.0 to 1.0. The time response of the inputs to G(s) are shown in Figure 5.4. The closed loop system is stable as predicted ,however it is not very well behaved with large

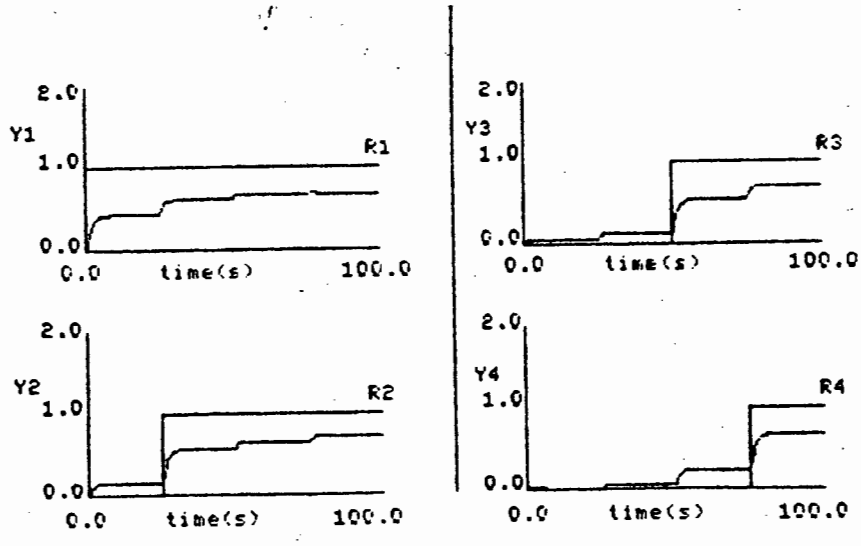

Figure 5.3. The closed loop response of the outputs of the system represented by (5.34) to steps in the setpoints of each loop at different times.

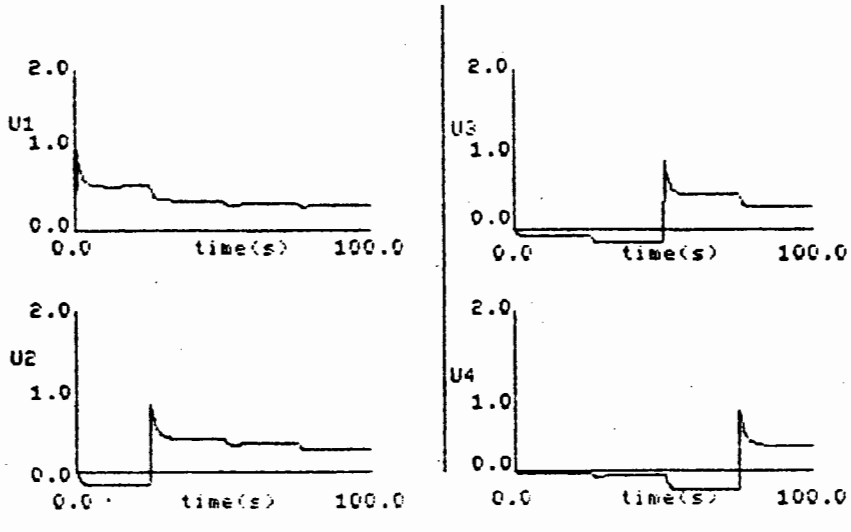

Figure 5.4. The closed loop response of the inputs of the system represented by (5.34) to steps in the setpoints of each loop at different times.

offsets and disturbances between the different loops. This is a major difficulty with interaction indices generally, they often fail to give any information on the quality of control that will be achieved. In this case a controller is clearly required to eliminate offsets and disturbances.

The system model rows and columns are now rearranged to give the structure shown in (5.39).

$$G(s) = \begin{bmatrix} \frac{0.3}{1+5s} & \frac{0.2}{1+5s} & \frac{1.0}{1+4s} & \frac{0.7}{1+5s} \\ \frac{0.4}{1+5s} & \frac{0.35}{1+5s} & \frac{0.6}{1+5s} & \frac{1.0}{1+4s} \\ \frac{1.0}{1+4s} & \frac{0.6}{1+5s} & \frac{0.35}{1+5s} & \frac{0.4}{1+5s} \\ \frac{0.7}{1+5s} & \frac{1.0}{1+4s} & \frac{0.2}{1+5s} & \frac{0.3}{1+5s} \end{bmatrix} \qquad (5.39)$$

In this matrix columns one, three, two and four of (5.34) have been exchanged.

Once again four different partitionings of this matrix are considered. These are the ones given in (5.35), (5.36), (5.37) and (5.38). Figure 5.5 shows r(C(s)) vs frequency for the different blockings.

The r(C(s)) values obtained for each of the different partitionings are all much greater than one. The lowest values obtained are around 6.8 while the highest are around 20.0. The high values of the r(C(s)) values suggest that the current row-column ordering is not suited to distributed control. The most promising partitionings are for a 1x1

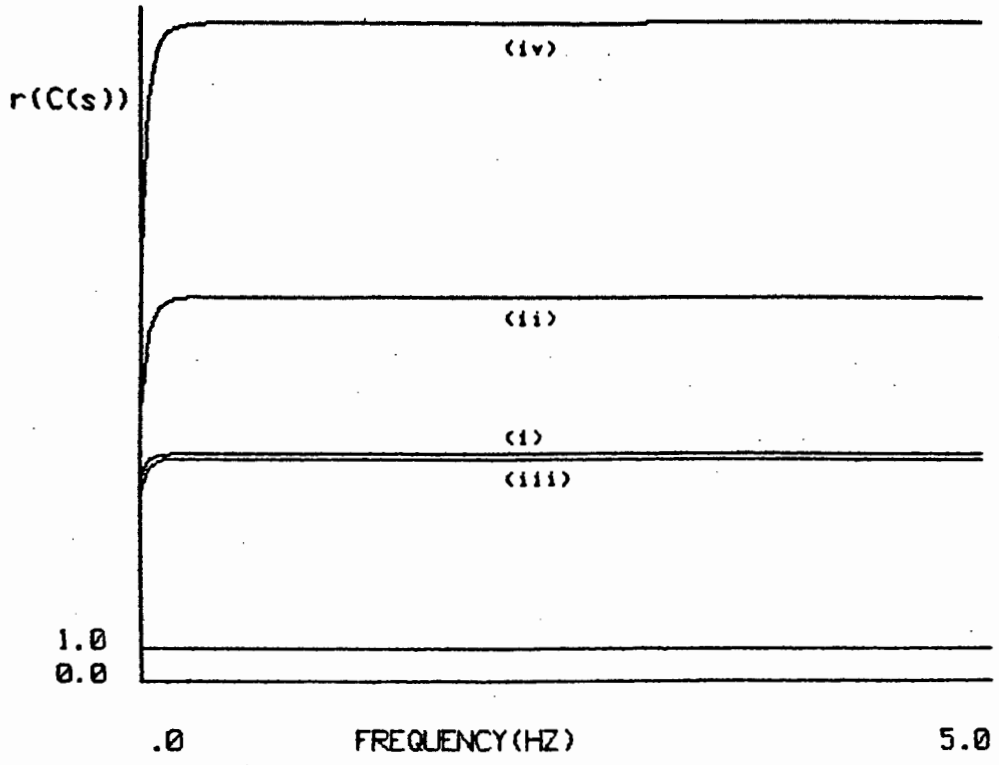

Figure 5.5. Plots of r(c(s)) vs frequency for different partitionings of the system represented by (5.39).

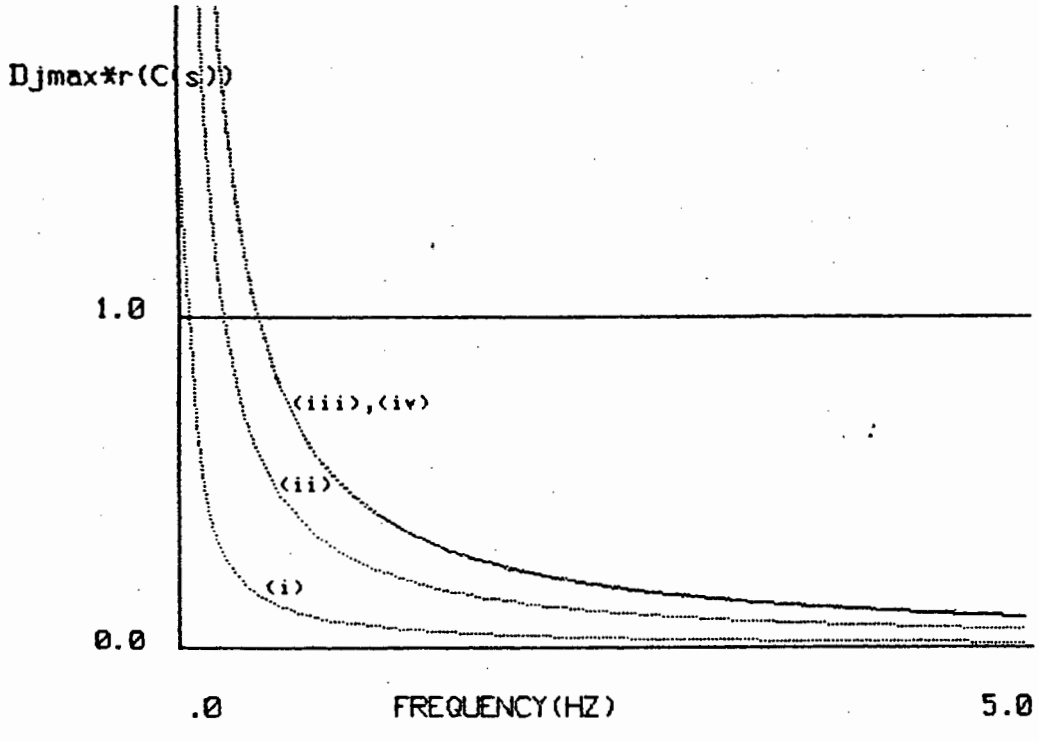

Figure 5.6. Plots of djmax*r(C(s)) vs frequency for different partitionings of the system represented by (5.39)

blocking for each of the on-diagonal elements or else for a blocking with one 1x1 and one 3x3 block on the diagonal. However with such high r(C(s)) values the designer would try different row-column orderings before attempting to design a decentralised controller for any of these partitionings.

Figure 5.6 shows $d_{jmax}.r(C(s))$ vs frequency for each of the different blockings. All of curves produced have values greater than one at low frequencies. Hence the stability of the closed loop on-diagonal blocks does not necessarily ensure the closed loop stability of the system as a whole. Notice however that because (5.30) is only a sufficient condition for stability the closed loop system might in fact be stable.

Figure 5.7 shows the response of the closed loop system outputs to a unit step in the setpoint R1. The inputs to the system are shown in Figure 5.8. The system is clearly unstable, the outputs tend to infinity as time progresses in response to a change in the setpoint of Y1. In this case the high r(C(s)) values suggested that decentralised control of the system would be difficult. It might still be possible to find a decentralised controller that would stabilise the system but the designer would first try to reduce r(C(s)) by reordering the rows and columns of the matrix.

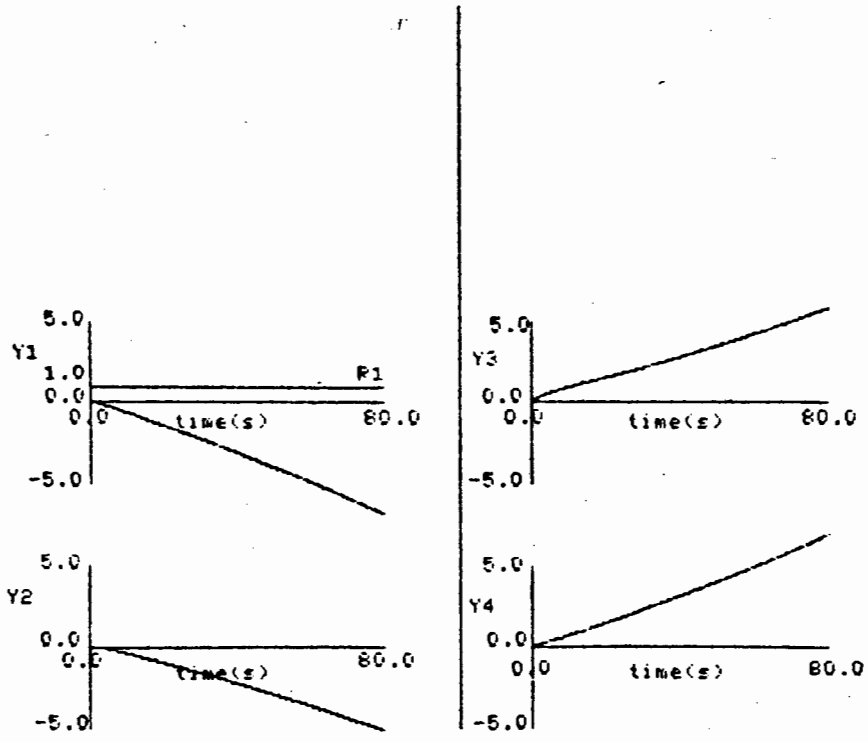

Figure 5.7. The closed loop response of the outputs of the system represented by (5.39) to steps in the setpoints of each loop at different times.

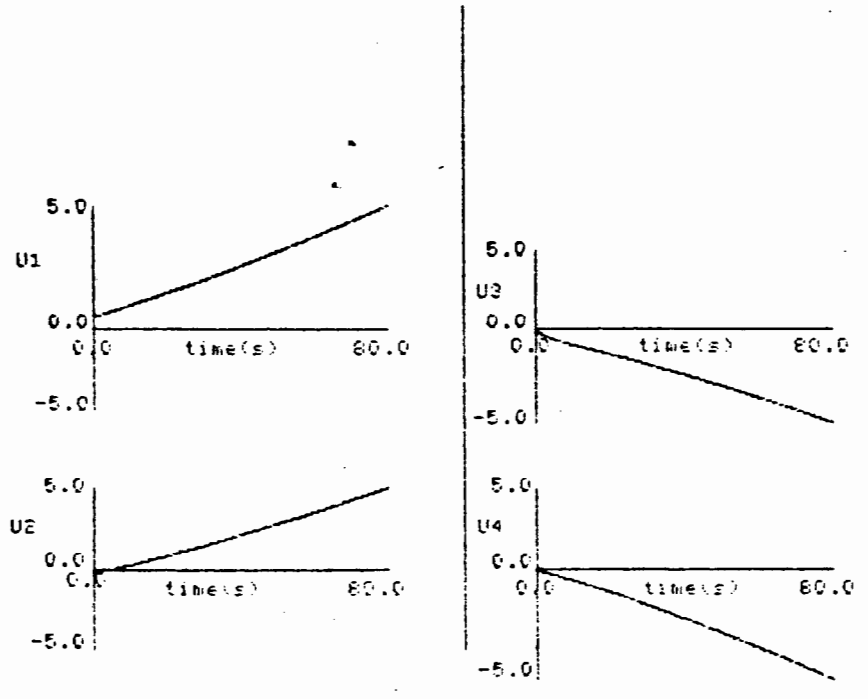

Figure 5.8. The closed loop response of the inputs of the system represented by (5.39) to steps in the setpoints of each loop at different times.

The final G(s) to be considered is shown in (5.40).

$$G(s) = \begin{bmatrix} \frac{1.0}{1+4s} & \frac{0.7}{1+5s} & \frac{0.3}{1+5s} & \frac{0.2}{1+5s} \\ \frac{0.2}{1+5s} & \frac{0.3}{1+5s} & \frac{0.7}{1+5s} & \frac{1.0}{1+4s} \\ \frac{0.6}{1+5s} & \frac{1.0}{1+4s} & \frac{0.4}{1+5s} & \frac{0.35}{1+5s} \\ \frac{0.35}{1+5s} & \frac{0.4}{1+5s} & \frac{1.0}{1+4s} & \frac{0.6}{1+5s} \end{bmatrix} \qquad (5.40)$$

In this case rows two, three, and four of (5.34) have been reordered.

The partitionings considered are the same as in the previous two cases, i.e. (5.35), (5.36), (5.37) and (5.38).

The values of r(C(s)) vs frequency are plotted for the different blockings in Figure 5.9. In this case the blocking of (5.37) produces low values of r(C(s)), less than one, for all the frequencies examined. The other partitionings all produce high values of r(C(s)) the next lowest being around 4.5 and the highest , for a partitioning with two 2x2 blocks, being around 7.0. The partitioning of (5.37), one 1x1 block and one 3x3 block on the diagonal, gives r(C(s)) of less than one and hence this suggests that this partitioning might give stable decentralised control. This in turn suggests that the controller K(s) should have a corresponding blocking.

Figure 5.10 shows $d_{jmax}.r(C(s))$ plotted vs frequency for the different partitionings. All of the curves except that

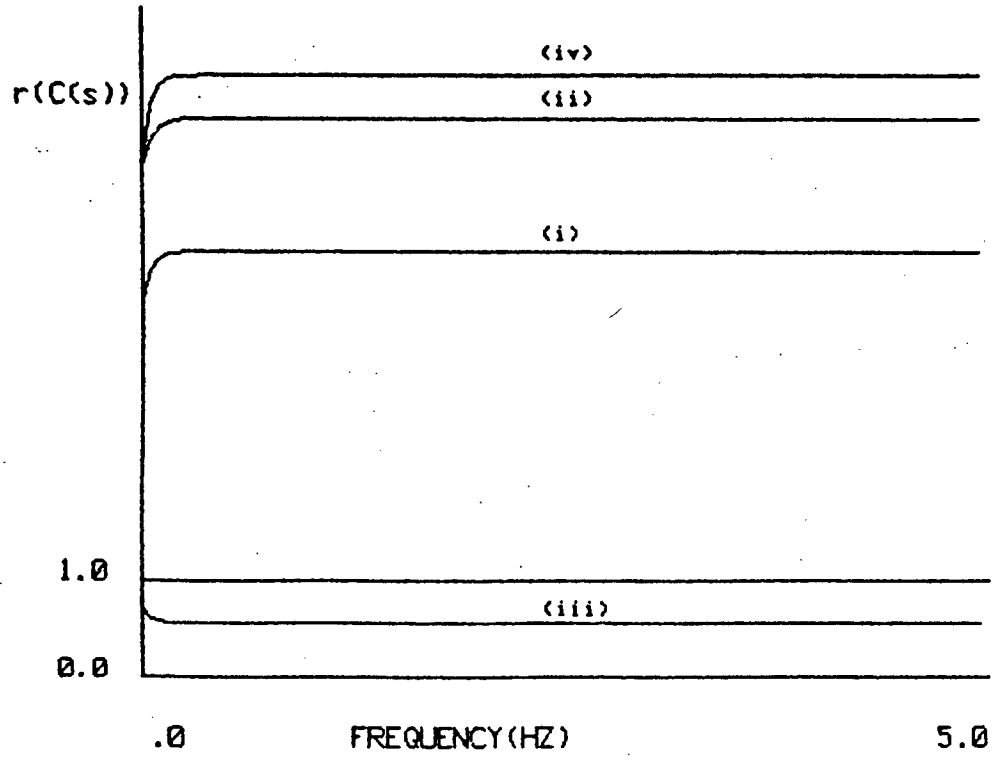

Figure 5.9. Plots of r(c(s)) vs frequency for different partitionings of the system represented by (5.40).

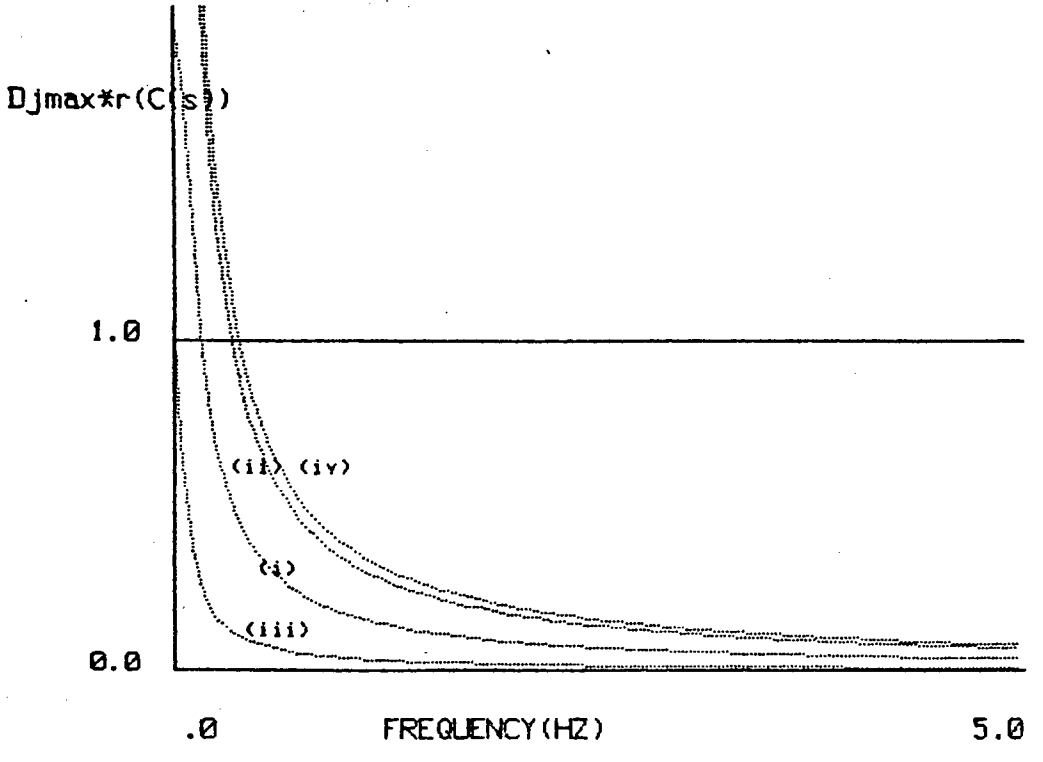

Figure 5.10. Plots of djmax*r(C(s)) vs frequency for different partitionings of the system represented by (5.40)

belonging to the blocking of (5.37) are greater than one at low frequencies. The information that this gives the designer is as follows, if the 1x1 block and the 3x3 block are closed loop stable then the closed loop system as a whole will be stable. Clearly the 1x1 block is closed loop stable hence the stability of the closed loop system depends on the closed loop stability of the 3x3 block. Simulation showed that the 3x3 block was stable when its inputs were stepped so the system as a whole is expected to be closed loop stable. Figure 5.11 shows the output of the system to steps in the setpoints of all four loops. Each setpoint was stepped in turn, each step occurring after the outputs had settled down after the previous step. The inputs to the system are shown in Figure 5.12. As expected the closed loop system is stable.

The above example illustrates the usefulness of r(C(s)) as an interaction measure. It provides the designer with a criteria for deciding on which of the different row and column orderings and matrix partitionings he should base his design. However he is still left with the problem of selecting which of the many different arrangements of the system model he should select that are likely to give low r(C(s)) values.

For a given system matrix the rows and columns may be reordered in any way that is desired and the partitioning is restricted only in that, the on diagonal blocks must be square.(Even this is not necessarily a restriction, Siljak [6] proposes a method of decomposing a matrix into overlapping diagonal blocks).

There is at present no systematic method of selecting the partitioning, and row-column arrangements, that will yield the lowest value of r(C(s)) for large systems [11]. This is a serious drawback for large scale systems that may have

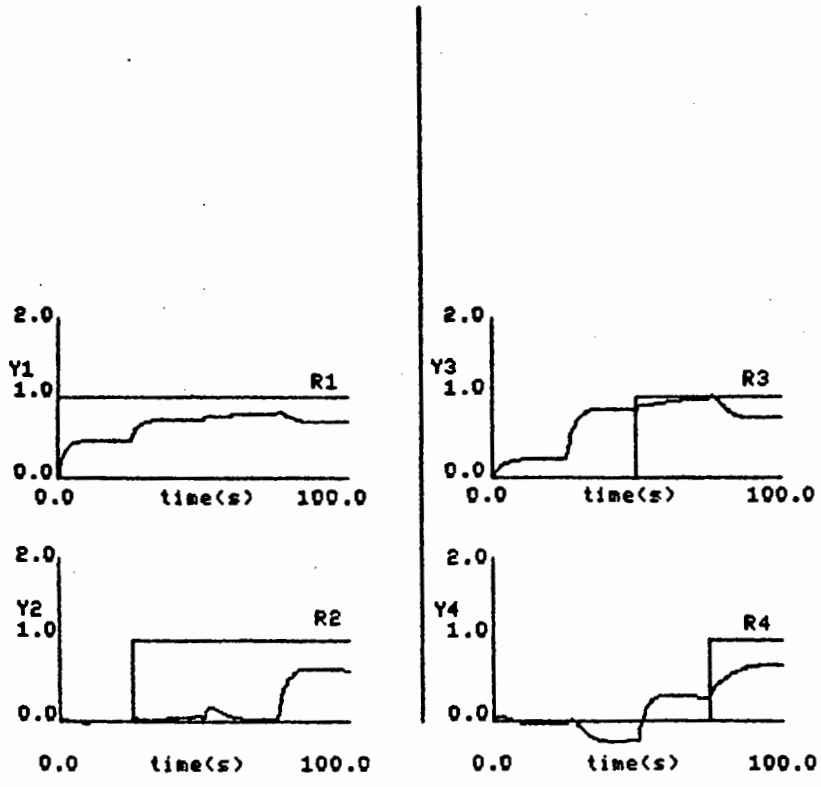

Figure 5.11. The closed loop response of the outputs of the system represented by (5.40) to steps in the setpoints of each loop at different times.

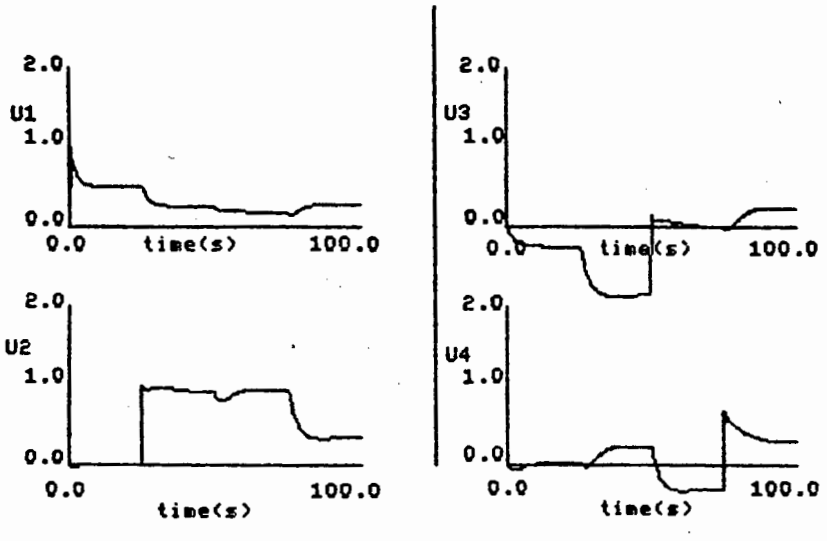

Figure 5.12. The closed loop response of the inputs of the system represented by (5.40) to steps in the setpoints of each loop at different times.

hundreds of input-output pairs, and for which there exist literally millions of possible blockings.

In the next section methods of selecting row-column orderings and partitionings that will yield low values of the interaction measure r(C(s)) are considered.

5.3 SELECTING ROW-COLUMN ORDERINGS AND PARTITIONINGS FOR DISTRIBUTED CONTROL

In this section a technique is developed to help the designer rearrange and partition the system matrix so that low values of the interaction matrix r(C(s)) are obtained.

5.3.1 Justification of Rearrangement and Partitioning method

From [6] it is known that if there are two matrices C_1 and C_2 with the same order then if $C_2 \geq C_1$, elementwise then $r(C_2) \geq r(C_1)$. Hence if the designer can rearrange the system matrix G(s) so that the elements of the corresponding comparison matrix C(s) are as small as possible then r(C(s)) will be minimised.

If the nxn matrix G(s) is partitioned into an mxm composite matrix with elements $G_{ij}(s)$ then the matrix C(s) has the elements

$$c_{ij}(s) = \begin{cases} 0 & i=j \\ \|G_{ij}(s)G_{jj}^{-1}\| & i \neq j \end{cases} \qquad (5.41)$$

Hence the designer wants the size of all the $\|G_{ij}(s)G_{jj}^{-1}\|$ terms to be a minimum. It is not immediately clear as to how these terms can be reduced by row column manipulation and matrix partitioning however the left hand side of the

inequality in (5.42) gives an upper bound for the value of $\|G_{ij}(s)G_{jj}^{-1}\|$.

$$\|G_{ij}(s)\| \, \|G_{jj}(s)^{-1}\| \geq \|G_{ij}(s)G_{jj}(s)^{-1}\| \qquad (5.42)$$

In particular if $1 \geq \|G_{ij}(s)\| \, \|G_{jj}(s)^{-1}\|$ then $1 \geq \|G_{ij}(s)G_{jj}(s)^{-1}\|$. Hence the problem now is to try and reduce the value of each of the $\|G_{ij}(s)\| \, \|G_{jj}(s)^{-1}\|$ elements.

The first possibility is to reduce the value of $\|G_{ij}\|$. The norm of a matrix is bounded below by the largest element in the matrix, [15], so clearly if the elements of $G_{ij}(s)$ are small then $\|G_{ij}(s)\|$ will be small. In the discussion that follows the norm used is defined as follows. For a nxn matrix A the norm of A is defined as the maximum row sum of a nxn matrix containing the normalised elements of A i.e.

$$\|A\| = \max_i \sum_{j=1}^{n} |a_{ij}| \qquad (5.43)$$

This matrix norm is usually given the symbol $\|A\|_\infty$. All the matrix norms from now on are assumed to be defined as in (5.43).

Using the above definition the designer requires the maximum row sum of the normed elements of each $G_{ij}(s)$ to be as small as possible.

The second way of reducing the upper bound of $\|G_{ij}(s)G_{jj}^{-1}\|$ is to reduce the size of $\|G_{jj}(s)^{-1}\|$. It is once again not immediately apparent how changing the elements of $G_{jj}(s)$ will effect the norm of $G_{jj}(s)^{-1}$. In (5.45) a lower bound for $\|G_{jj}(s)^{-1}\|$ is derived.

$$\|G_{jj}(s)\| \, \|G_{jj}(s)^{-1}\| \geq \|G_{jj}(s)G_{jj}(s)^{-1}\| \geq 1 \qquad (5.44)$$

rearrangement gives

$$\|G_{jj}(s)^{-1}\| \geq \|G_{jj}(s)\|^{-1} \qquad (5.45)$$

Reducing the lower bound does not of course limit the maximum size of $\|G_{jj}(s)^{-1}\|$ but it does at least allow $\|G_{jj}(s)^{-1}\|$ to have a low value. In particular if

$$\|G_{jj}(s)\|^{-1} \geq 1$$

then

$$\|G_{jj}(s)^{-1}\| \geq 1$$

In order to decrease the value of $\|G_{jj}(s)\|^{-1}$ the size of $\|G_{jj}(s)\|$ must be increased. This means that the largest elements should be incorporated into the on-diagonal blocks of the matrix G(s) and in particular that the row sums of the on-diagonal blocks be maximised.

As examples of the theory discussed above consider the following 2x2 matrices.

The first example looks at the effect on the interaction measure r(C) when the value of the off-diagonal elements is reduced. The matrix in (5.45) has a r(C) value of 1.2. The matrix in (5.46) has the same diagonal elements but the off diagonal elements have been reduced. The new value for r(C) is 0.67. This shows that if the designer can reduce the value of the norms of the off-diagonal submatrices of the matrix G then the interaction measure will be reduced.

$$G = \begin{bmatrix} 1.0 & 2.0 \\ 0.6 & 1.0 \end{bmatrix} \tag{5.45}$$

$$G = \begin{bmatrix} 1.0 & 1.5 \\ 0.3 & 1.0 \end{bmatrix} \tag{5.46}$$

In the second example the first matrix in (5.47) has an r(C) value of 1.1 in the second matrix, shown in (5.48), the off-diagonal elements have the same values as in (5.47) but the on-diagonal elements have been increased in value. The interaction measure now has the value 0.7. Thus increasing the value of the on-diagonal elements will serve to decrease the interaction measure as expected.

$$G = \begin{bmatrix} 0.7 & 0.6 \\ 0.4 & 0.3 \end{bmatrix} \tag{5.47}$$

$$G = \begin{bmatrix} 1.0 & 0.6 \\ 0.4 & 0.5 \end{bmatrix} \tag{5.48}$$

A special case occurs if an element $g_{jj}(s)$ of the nxn matrix G(s) is dominant on its column i.e.

$$|g_{jj}(s)| > \sum_{\substack{i=1 \\ j \neq i}}^{n} |g_{ij}| \tag{5.49}$$

In this case if the matrix is partitioned so that the $g_{jj}(s)$ element of G(s) is the sole member of a 1x1 on-diagonal submatrix $G_{pp}(s)$, then the off-diagonal elements in the pth

column of the comparison matrix, C(s), will all be less than one. This is because for any off-diagonal submatrix $G_{rp}(s)$ which contains the rows 1 to m inclusive of G(s)

$$|G_{rp}(s)G_{pp}(s)^{-1}| = \max_i \left(|g_{ij}(s)|\ |g_{jj}(s)^{-1}|\right)\ i=1,..,m \tag{5.50}$$

however from (5.49) it is evident that the right hand side of (5.50) must always be less than one and hence all the off-diagonal elements in column p of C(s) will be less than one. Notice that the element does not have to be dominant to ensure that the right hand side of (5.50) is less than one. Provided $|g_{jj}|$ is greater than each of the other elements in the column then the right hand side of (5.50) will be less than one, this is automatically true if $g_{jj}(s)$ is dominant on its column.

Taking all of the above into account guidelines for choosing row-column orderings and matrix partitionings that are likely to yield low values of r(C(s) can now be given.

(i) Any column dominant elements should be placed on the diagonal if this is possible. If such an element is placed on the diagonal and no other column dominant elements occur in the same row then the partitioning chosen for the matrix should be such that the element is contained in a 1x1 on-diagonal submatrix. If more than one column dominant element occurs in the same row the most dominant should be placed on the diagonal and the others placed in adjacent columns. In this case the designer should try partitionings that (a) put the on-diagonal element in a 1x1 submatrix and (b) put all the column dominant elements in the same row in the same on-diagonal block.

(ii) For columns that do not contain dominant elements then the largest element in the column should be placed on the diagonal if this is possible. Such rearrangement should not displace any column dominant elements from the diagonal.

(iii) In general the designer should try to ensure that the largest i.e. most dominant elements in each column are included in the on-diagonal blocks and the smallest elements are included in the off-diagonal blocks.

5.3.2 Description of Reordering and Partitioning Method

The method used here is based on the method utilised in chapter four to rearrange a matrix to be diagonally dominant if such rearrangement is possible. In chapter four two matrices M(s) and N(s) were defined. The N(s) matrix was used to compare the dominance of elements in the same column and the M(s) matrix was used to compare the dominance of elements in the same row. If any element in the N(s) matrix was greater than 0.5 then that element was dominant on its column while if any element in M(s) was greater than 0.5 the element was dominant on its row. Finally all the elements of M(s) and N(s) lay between one and zero.

In this chapter column dominance is of more interest than row dominance because of the way in which r(C(s)) is calculated. However row dominance must also be checked for initially since if the matrix can be rearranged to be row diagonally dominant this suggests that controller can be diagonal i.e. each on-diagonal 1x1 subsystem is controlled independently.

A new representation for the N(s) and M(s) matrices has been

its row will be represented by a one in the Z matrix. Otherwise all the elements in Z will be zero. The value of β is reduced in increments until β equals zero. Each time the Z matrix is calculated and displayed. The designer rearranges the matrix according to the guidelines given previously, trying to concentrate the most dominant elements in the on-diagonal blocks. During the course of the analysis the designer will probably select several different row-column arrangements and partitionings of G(s) that he feels are likely to yield low values of r(C(s)). When the analysis has been completed the designer would then test these rearrangements and partitionings to see which one gives the lowest value for r(C(s)). Note that the designer will usually treat elements that partially fall into a particular range, $1 \geq n_{ij}(s) \geq \beta$, as if they fell completely into the range for purposes of rearrangement. However these should be checked with the polar plot representation of the previous chapter to see over what frequency range these elements fall into the range $1 \geq n_{ij}(s) \geq \beta$ for a particular β.

5.3.3 Examples

A program has been written that calculates the N(s), M(s) and Z matrices. The program then displays the Z matrix for different values of β which have been chosen by the designer. Instead of displaying a matrix of ones, zeros and minus ones however a grid of lines is drawn, the rectangles formed by this grid represent the elements of the Z matrix. If a particular z_{ij} element is zero then the corresponding rectangle is left unfilled. If the element is a one then the rectangle is filled and if the element is a minus one the element is filled and crosshatched. For more information on the program see chapter seven. All the results presented below have been generated using this program.

As an example of the use of the method given above consider

the boiler plant that was used in a previous section to demonstrate the use of r(C(s)). The plant is represented by a 4x4 matrix of transfer functions and is taken from [12]. In order to demonstrate how the choice of rows and columns might be affected the matrix in [12] has been rearranged by exchanging columns one and three and two and four as well this is the matrix that was given in (5.39) and is reproduced below in (5.52).

$$G(s) = \begin{bmatrix} \frac{0.3}{1+5s} & \frac{0.2}{1+5s} & \frac{1.0}{1+4s} & \frac{0.7}{1+5s} \\ \frac{0.4}{1+5s} & \frac{0.35}{1+5s} & \frac{0.6}{1+5s} & \frac{1.0}{1+4s} \\ \frac{1.0}{1+4s} & \frac{0.6}{1+5s} & \frac{0.35}{1+5s} & \frac{0.4}{1+5s} \\ \frac{0.7}{1+5s} & \frac{1.0}{1+4s} & \frac{0.2}{1+5s} & \frac{0.3}{1+5s} \end{bmatrix} \qquad (5.52)$$

Initially the matrix is checked for row dominance by setting β to 0.5. The frequency range considered is from 0 to 1 hz. The resultant Z matrix is represented in Fig. 5.13.

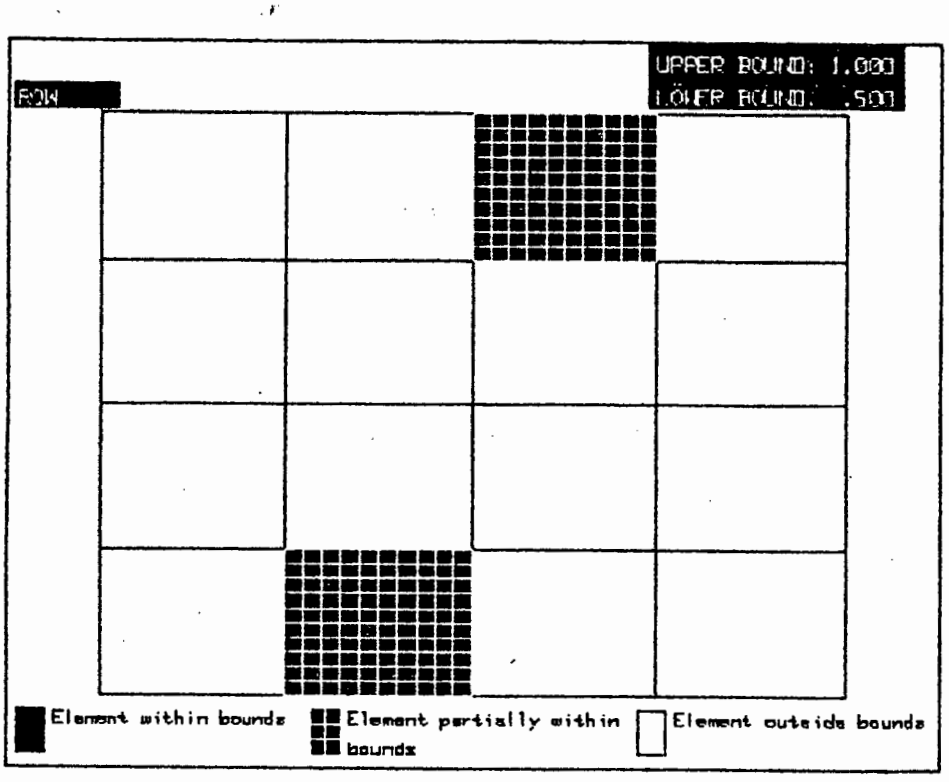

Figure 5.13. Representation of Z matrix for (5.52) showing row dominance.

In the representation used here the value of β is the value of the lower bound shown in the figure. The lable 'row' indicates that the Z matrix is formed from the M(s) matrix.

From the figure it is immediately apparent that the matrix cannot be rearranged to be row dominant since there are only two row dominant elements and these are not dominant over the entire frequency range. The rest of the analysis uses column dominance, i.e. the Z matrix is derived from N(s). The frequency range throughout is from 0 to 1 hz.

Figure 5.14 shows the result of setting β to 0.9. No elements are shaded and hence there are no $n_{ij}(s)$ elements in this range. The value of β was decreased in steps of 0.1. Only those values that resulted in a change in the Z matrix were recorded. Figure 5.15 shows the Z matrix for β set to 0.5. Two elements fall partly into the range $1 \geq n_{ij}(s) \geq 0.5$. These elements should be place on the diagonal as the

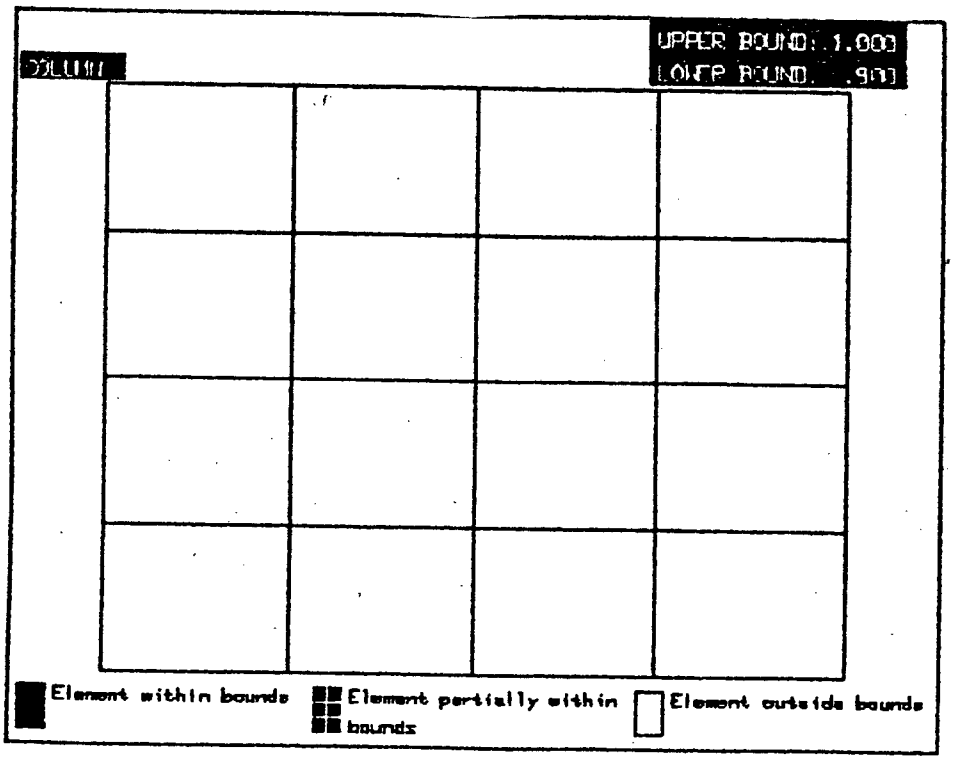

Figure 5.14. Diagram representing the Z matrix of the system represented by (5.52) for β=0.9.

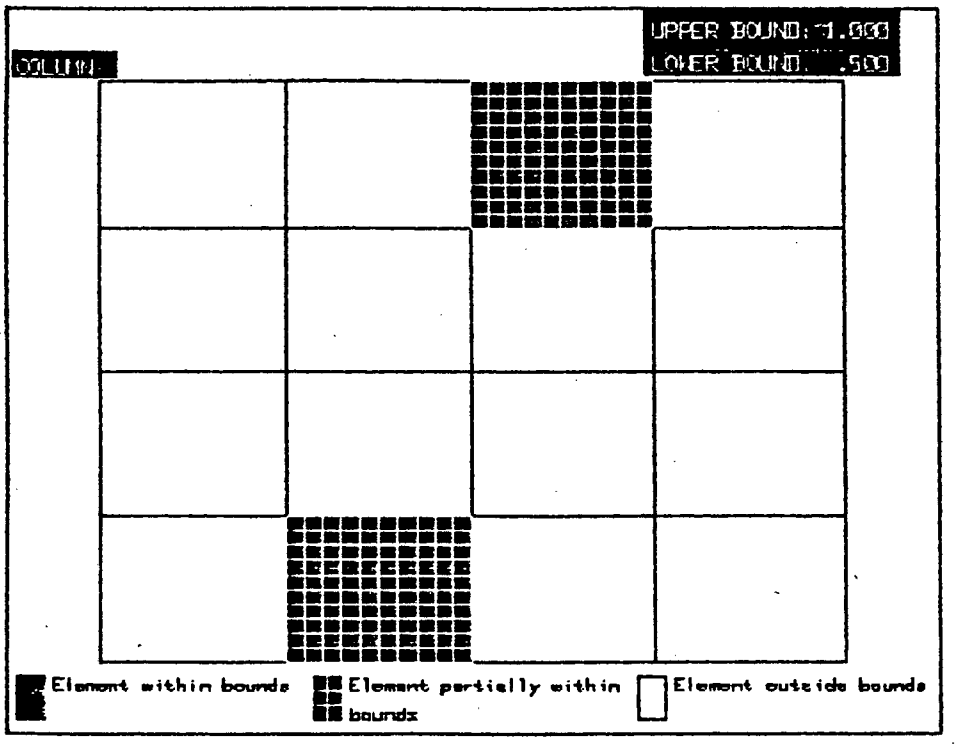

Figure 5.15 Diagram representing the Z matrix of the system represented by (5.52) for β=0.5.

next figure, Fig. 5.16, clearly shows since the two elements fall wholly within the range $1 \geq n_{ij}(s) \geq \beta$. In Fig. 5.16 β is 0.4. There are four elements that lie in this range. The matrix is now rearranged to place these elements on the diagonal. This is accomplished by exchanging columns one and three and columns two and four. The Z matrix representation for the new matrix is shown in Fig. 5.17. This diagonal structure suggests that the new matrix be partitioned so that each on-diagonal block is 1x1.

The next value of β for which any change is seen is when β is equal to 0.2. The resultant Z matrix is represented in Fig. 5.18. Here the most dominant elements in the range are clustered into two 2x2 on-diagonal blocks. This immediately suggests that partitioning should be such that there are two 2x2 on-diagonal blocks. Further reduction of β to 0.1 gives the result shown in Fig. 5.19. No blockings are readily apparent here although the designer might wish to try partitioning the matrix into two blocks, one 1x1 and another 3x3 block. This would take advantage of the small $n_{41}(s)$ and $n_{14}(s)$ elements.

This analysis has suggested one row-column arrangement likely to give low interaction measure values. For this arrangement the analysis gives two partitionings that are very likely to give low values of the interaction measure r(C(s)) and two others that may give low values. The plots for r(C(s)) vs frequency were calculated previously and appear in Fig. 5.1. From Fig. 5.1 it is apparent that the blocking that gives the lowest value of r(C(s)) is the one in which there are two 2x2 submatrices on the diagonal. However all the blockings tried for the row-column arrangement found by this method (i.e. columns one and three and columns two and four of the original matrix exchanged) give values of r(C(s)) that are close to one.

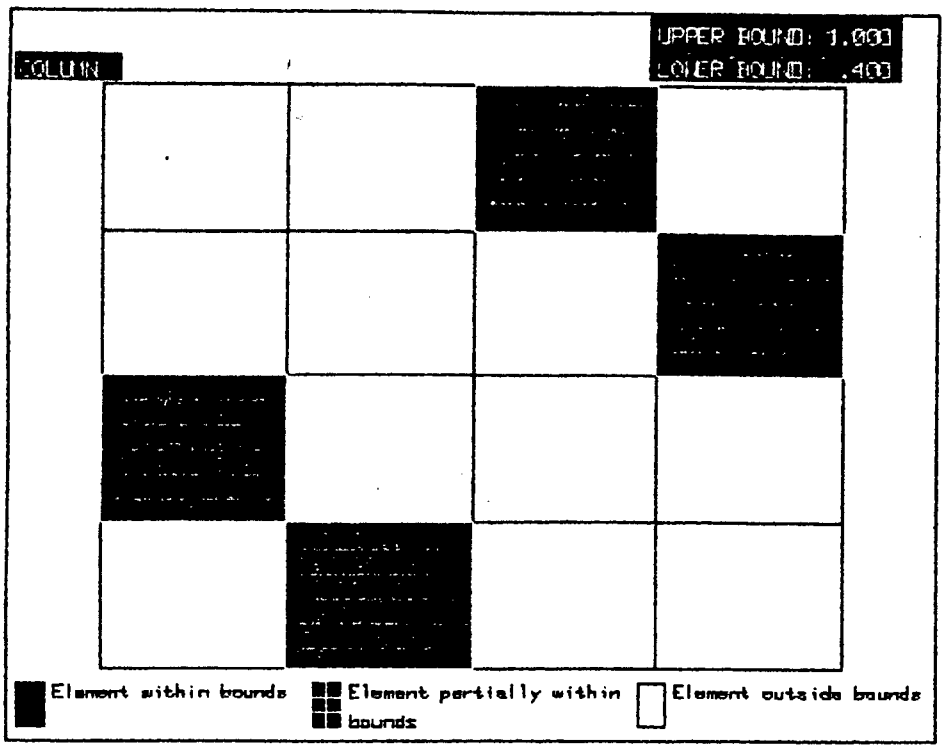

Figure 5.16. Diagram representing the Z matrix of the system represented by (5.52) for β=0.4.

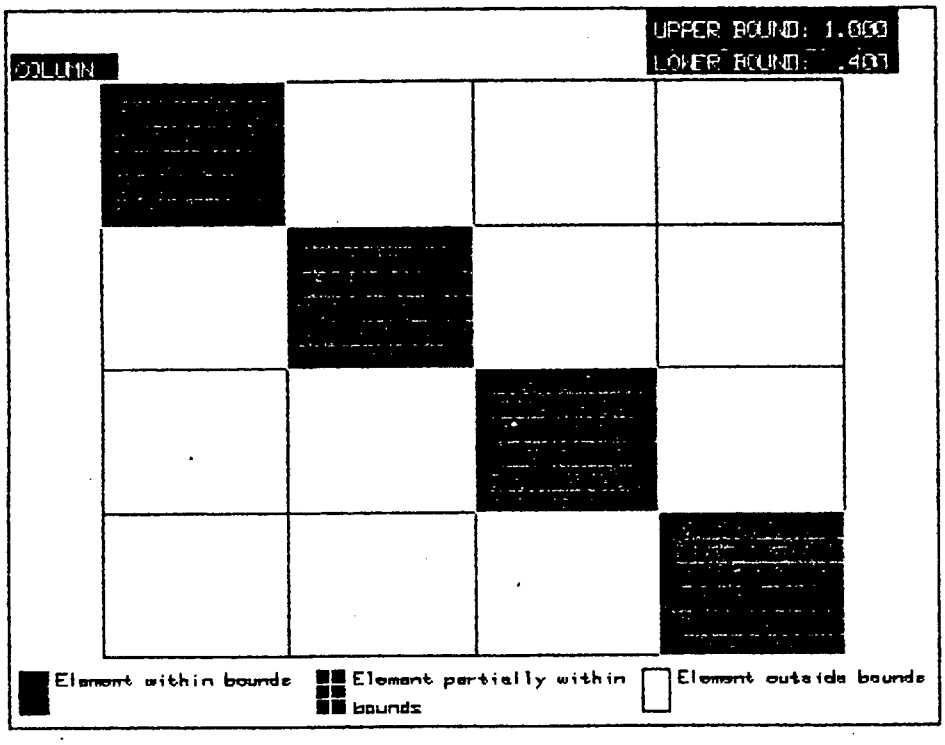

Figure 5.17. Diagram representing the Z matrix of the system represented by (5.52) for β=0.4 after rearrangement.

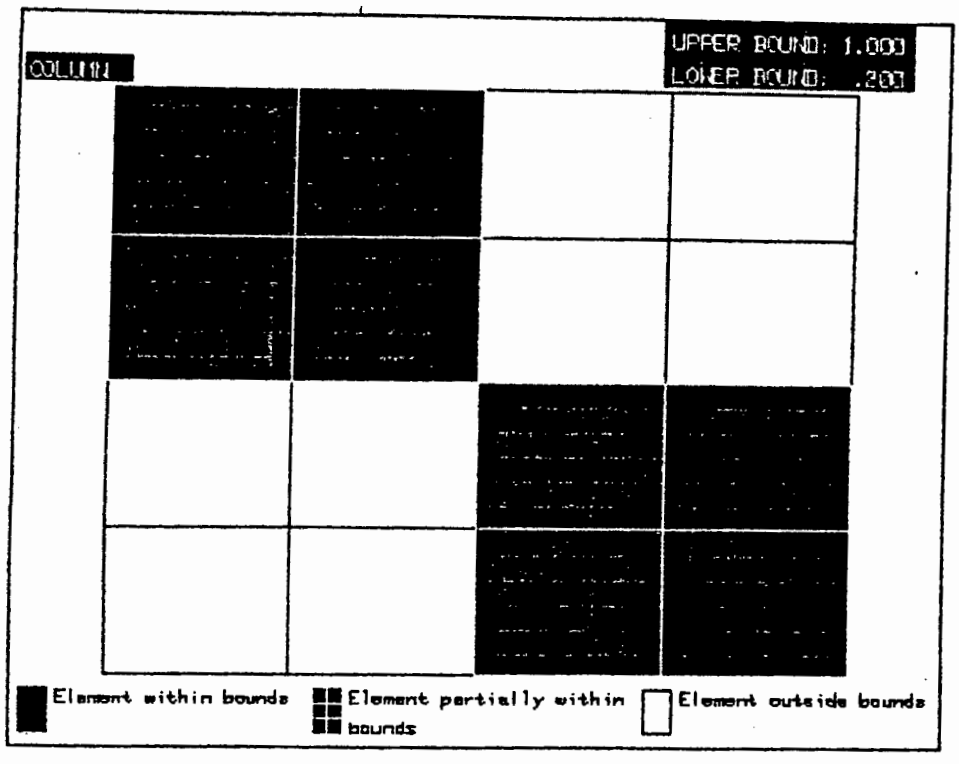

Figure 5.18. Diagram representing the Z matrix of the system represented by (5.52) after rearrangement for β=0.2.

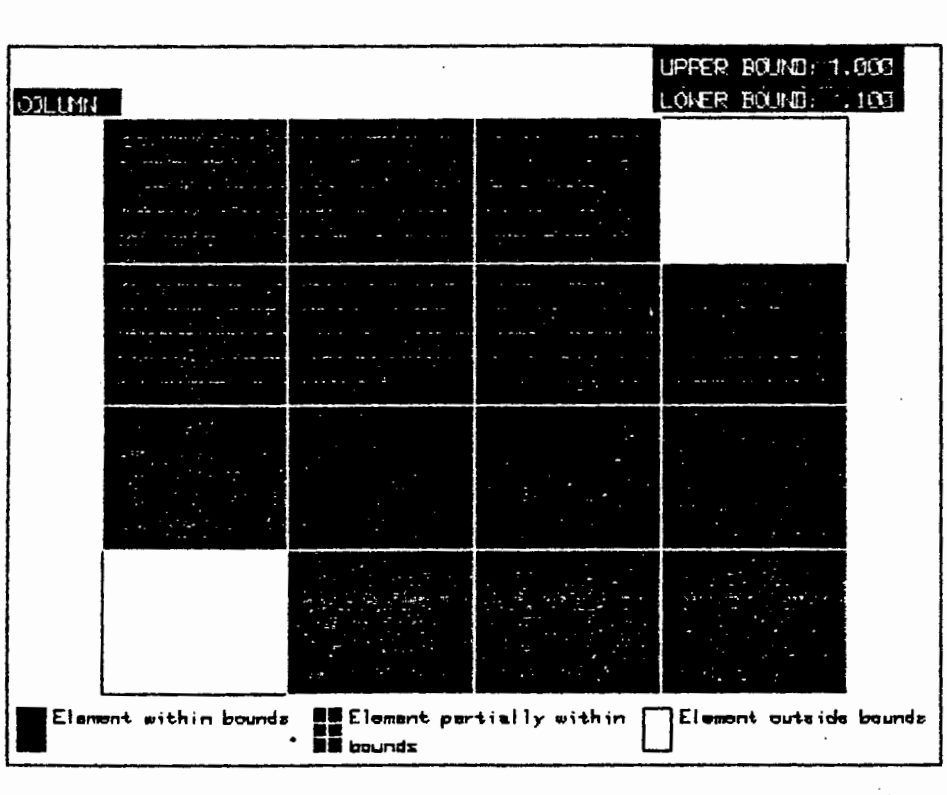

Figure 5.19. Diagram representing the Z matrix of the system represented by (5.52) after rearrangement for β=0.1.

In the next example the plant analysed is taken from [21]. The plant is a distillation system and is modelled by a 4x4 system model the transfer functions for which are shown in table 5.1. In the discussion that follows this is the G(s) matrix. The frequency range considered is 0 to 1 hz throughout.

Initially the M(s) and N(s) matrices are calculated over the chosen frequency range. The first step in the analysis is to check the matrix to ensure that it is not row dominant. This is done by calculating the Z matrix for M(s) with β, the lower bound, set to 0.5. The resultant representation for Z is shown in Fig. 5.20. Since there is only one shaded block, representing z_{22}, the matrix cannot be rearranged to be row diagonally dominant over the entire frequency range.

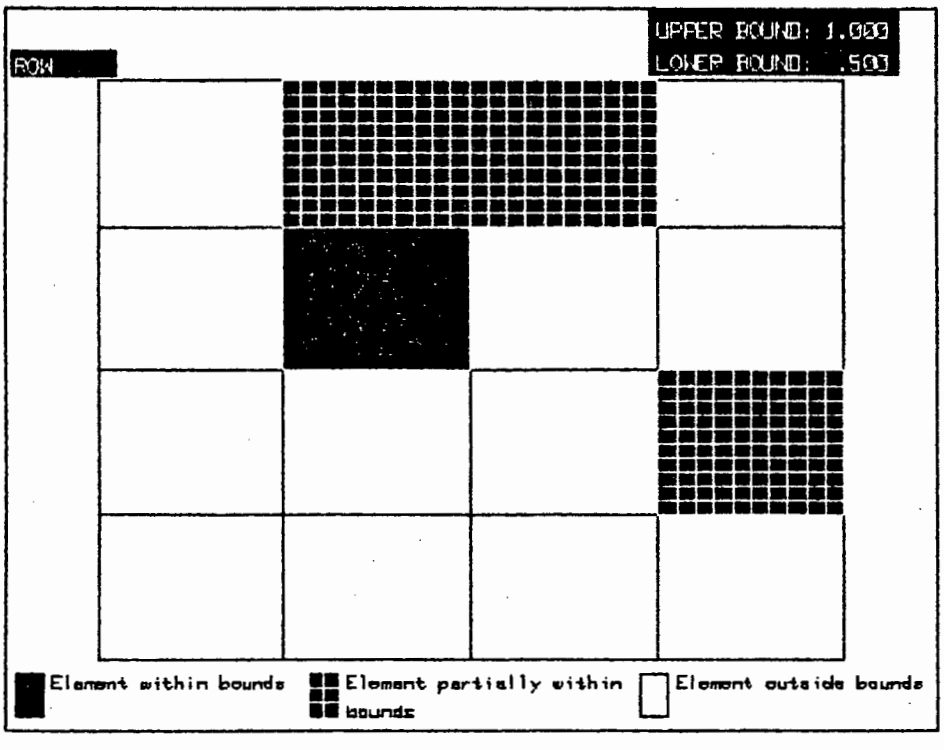

Figure 5.20. Diagram showing the row dominant elements of the matrix in table 5.1.

The rest of the analysis uses the N(s) matrix to derive the

g_{ij}	Transfer Function
g_{11}	$\frac{4.09\ e^{-1.3s}}{(33s+1)(8.3s+1)}$
g_{12}	$\frac{-6.36\ e^{-0.2s}}{(31.6s+1)(20s+1)}$
g_{13}	$\frac{-0.25\ e^{-0.4s}}{21s+1}$
g_{14}	$\frac{-0.49\ e^{-5s}}{(22s+1)^2}$
g_{21}	$\frac{-4.17\ e^{-4s}}{45s+1}$
g_{22}	$\frac{6.93\ e^{-1.01s}}{44.6s+1}$
g_{23}	$\frac{-0.05\ e^{-5s}}{(34.5s+1)^2}$
g_{24}	$\frac{1.53\ e^{-2.8s}}{48s+1}$

g_{ij}	Transfer Function
g_{31}	$\frac{-1.73\ e^{-17s}}{(13s+1)^2}$
g_{32}	$\frac{5.11\ e^{-11s}}{(13.3s+1)^2}$
g_{33}	$\frac{4.61\ e^{-1.02s}}{18.5s+1}$
g_{34}	$\frac{-5.48\ e^{-0.5s}}{15s+1}$
g_{41}	$\frac{-11.18\ e^{-2.6s}}{(43s+1)(6.5s+1)}$
g_{42}	$\frac{14.04\ e^{-0.02s}}{(45s+1)(10s+1)}$
g_{43}	$\frac{-0.1\ e^{-0.05s}}{(31.6s+1)(5s+1)}$
g_{44}	$\frac{4.49\ e^{-0.6s}}{(48s+1)(6.3s+1)}$

TABLE 5.1

This table is taken from [21] and contains the transfer functions for the mathematical model of a distillation system.

Z matrices that are examined. The next diagram, Fig. 5.21, represents the Z matrix for $\beta=0.5$. In this matrix there are five highlighted elements, one shaded and four crosshatched. From the shaded element the designer infers that the element $g_{33}(s)$ of the plant model is dominant on its column over the entire frequency range. This element is already on the diagonal, if it were not the matrix would be rearranged to place it on the diagonal. The crosshatched elements are dominant on their columns for part of the frequency range considered. As a general rule the designer will want to place these elements in on-diagonal submatrices when he partitions the system and so will arrange the rows and columns of the matrix so as to cluster these elements about the diagonal. However this must be done with care because of the frequency dependent nature of the elements. Specifically a particular element may be dominant over a very narrow range of frequencies and then rapidly decrease in value until it is smaller than many of the other elements in the column. In such a case if the element is placed on the diagonal then the interaction measure might be very large over a large frequency range. It is difficult to judge how an element, that has a wide spread of values over the frequency range, will effect the interaction measure. The only real aid to the designer in such cases is the array of polar diagrams of the elements of the matrix N(s) which will show the designer over how large a frequency range the element is dominant. The polar diagrams of the $n_{ij}(s)$ elements are shown in Fig. 5.22. Of the five elements that are highlighted when $\beta=0.5$ four are dominant over a large part of the frequency range. They correspond to the elements $g_{21}(s)$, $g_{22}(s)$, $g_{34}(s)$ and $g_{33}(s)$, the last being dominant over the entire frequency range. The only other crosshatched element, corresponds to $g_{41}(s)$. This element is seen to be dominant for only a very small part of its frequency range and should therefore be treated with care.

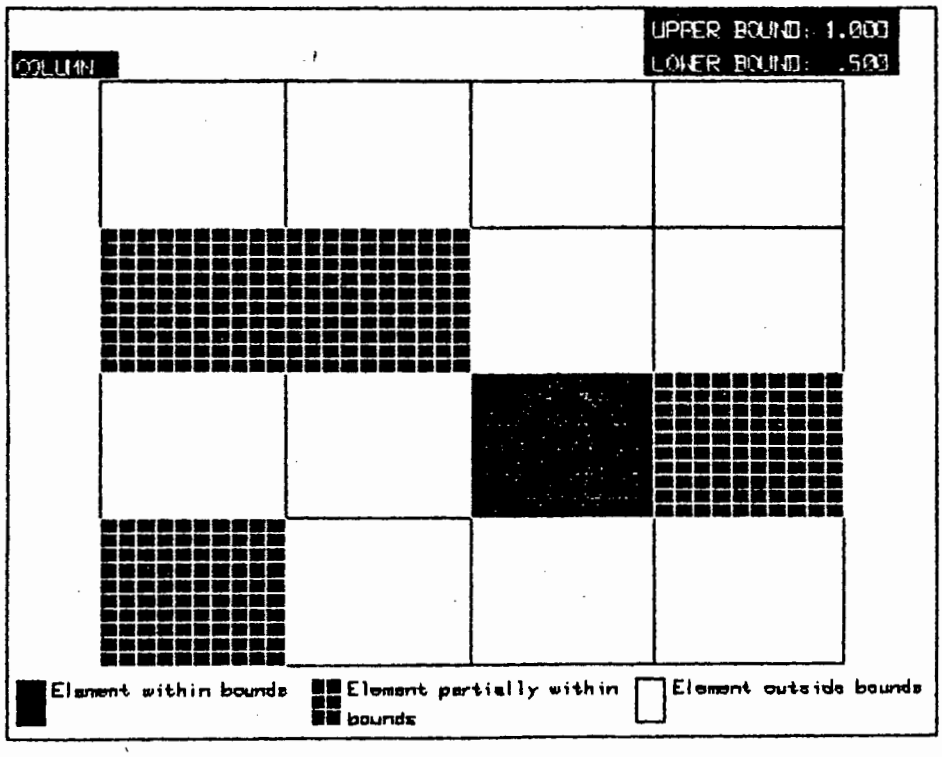

Figure 5.21. Diagram representing the Z matrix of the system represented by table 5.1 for β=0.5.

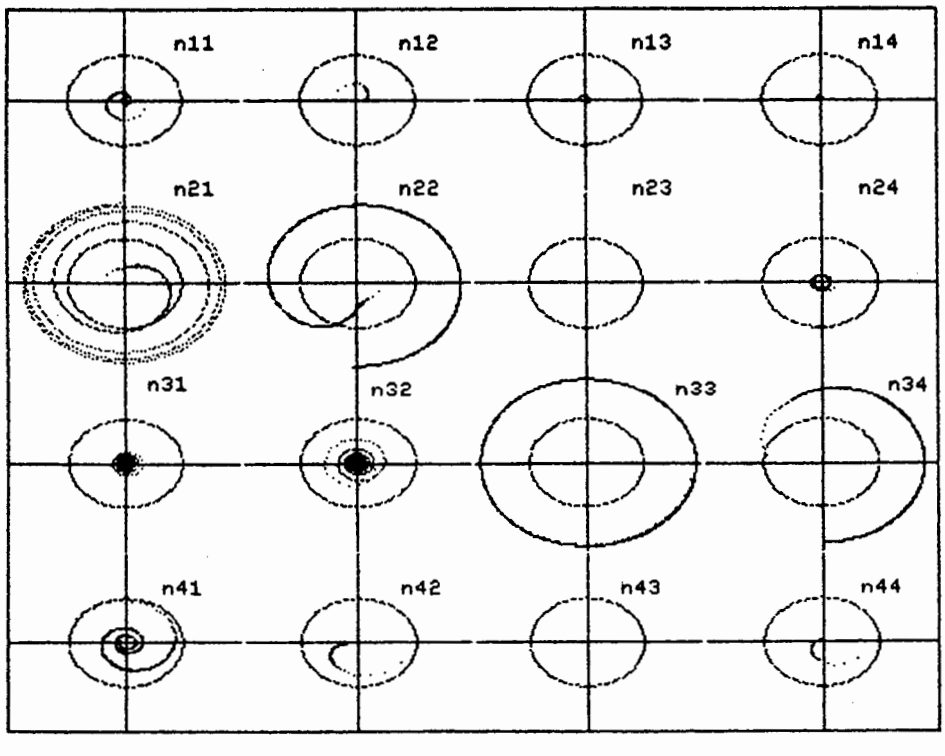

Figure 5.22. Array of polar plots of the elements of the N(s) matrix of the system in table 5.1.

At this point two possible partitionings are readily suggested. The first is a partitioning that results in four 1x1 on-diagonal submatrices. The two dominant elements on the diagonal, $g_{22}(s)$ and $g_{33}(s)$, will give rise to small elements in columns two and three of the comparison matrix C(s). However the dominances of the $g_{11}(s)$ and $g_{44}(s)$ elements have yet to be determined. If these elements are very small then the resulting large elements in the C(s) matrix in columns one and four will offset the low elements in columns two and three. The second partitioning suggested is one that results in there being two 2x2 submatrices on the diagonal. In this case four of the highlighted elements would be included in the 2x2 blocks. By exchanging rows one and four of the current matrix the crosshatched element corresponding to $g_{41}(s)$ can be placed on the diagonal and included in one of the 2x2 blocks. Whether this will result in a lower or higher value of r(C(s)) for either of the two partitionings mentioned above will depend on the values of the $g_{13}(s)$ and $g_{14}(s)$ elements.

The next value of β considered is β=0.3. The corresponding Z matrix is represented in Fig. 5.23. The same partitionings as before are suggested. Once again exchanging rows one and four is a possibility note however that in doing so a large element is moved to $g_{14}(s)$. This may increase the interaction measure especially if the matrix diagonal is partitioned into four 1x1 submatrices.

The final β value considered is β=0.2. The corresponding Z matrix is represented in Fig. 5.24. Once again the same partitionings as before are suggested. An additional partitioning with $g_{11}(s)$ in a 1x1 block and a second 3x3 block incorporating elements $g_{22}(s)$, $g_{33}(s)$ and $g_{44}(s)$. As before the exchange of rows one and four is a possibility.

The foregoing analysis then has isolated two possible sets

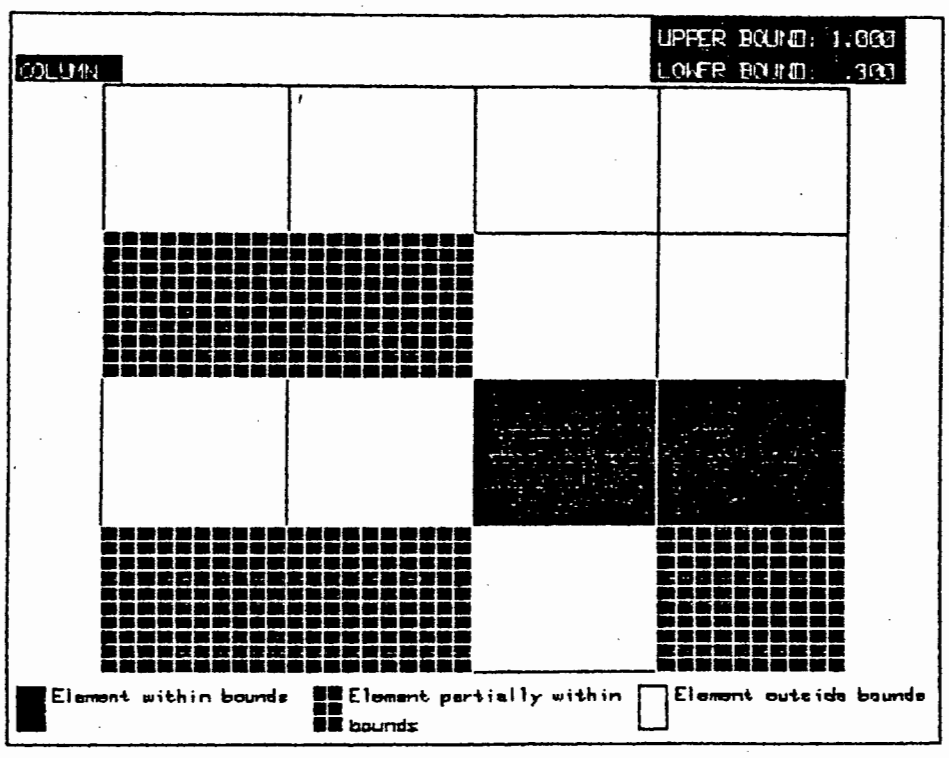

Figure 5.23. Diagram representing the Z matrix of the system represented by table 5.1 for β=0.3.

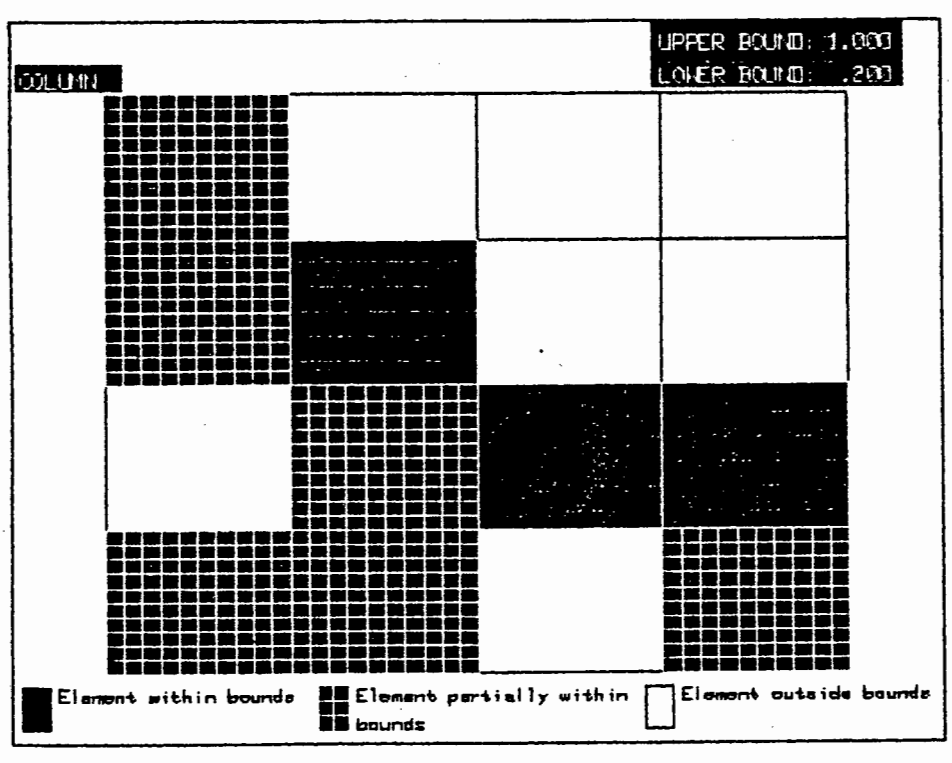

Figure 5.24. Diagram representing the Z matrix of the system represented by table 5.1 for β=0.2.

of row-column orderings and has suggested three possible partitionings for each set. Let (5.53) represent the G(s) matrix in table 5.1.

$$G(s) = \begin{bmatrix} g_{11}(s) & g_{12}(s) & g_{13}(s) & g_{14}(s) \\ g_{21}(s) & g_{22}(s) & g_{23}(s) & g_{24}(s) \\ g_{31}(s) & g_{32}(s) & g_{33}(s) & g_{34}(s) \\ g_{41}(s) & g_{42}(s) & g_{43}(s) & g_{44}(s) \end{bmatrix} \qquad (5.53)$$

The second possible row column arrangement would then give rise to the matrix shown in (5.54).

$$G(s) = \begin{bmatrix} g_{41}(s) & g_{42}(s) & g_{43}(s) & g_{44}(s) \\ g_{21}(s) & g_{22}(s) & g_{23}(s) & g_{24}(s) \\ g_{31}(s) & g_{32}(s) & g_{33}(s) & g_{34}(s) \\ g_{11}(s) & g_{12}(s) & g_{13}(s) & g_{14}(s) \end{bmatrix} \qquad (5.54)$$

The partitionings to be considered in each case would be

(i) A partitioning into a 4x4 composite matrix with each on-diagonal submatrix, $G_{jj}(s)$, having order one, i.e. $G_{jj}(s)=g_{jj}(s)$.

(ii) A partitioning into a 2x2 composite matrix with the two on-diagonal submatrices $G_{11}(s)$ and $G_{22}(s)$ both having order two.

(iii) A partitioning into a 2x2 composite matrix with the on-diagonal submatrix $G_{11}(s)$ being 1x1 and $G_{22}(s)$ being a 3x3 submatrix.

Figure 5.25 shows the plots of the interaction measure r(C(s)) vs frequency for the row-column arrangement shown in

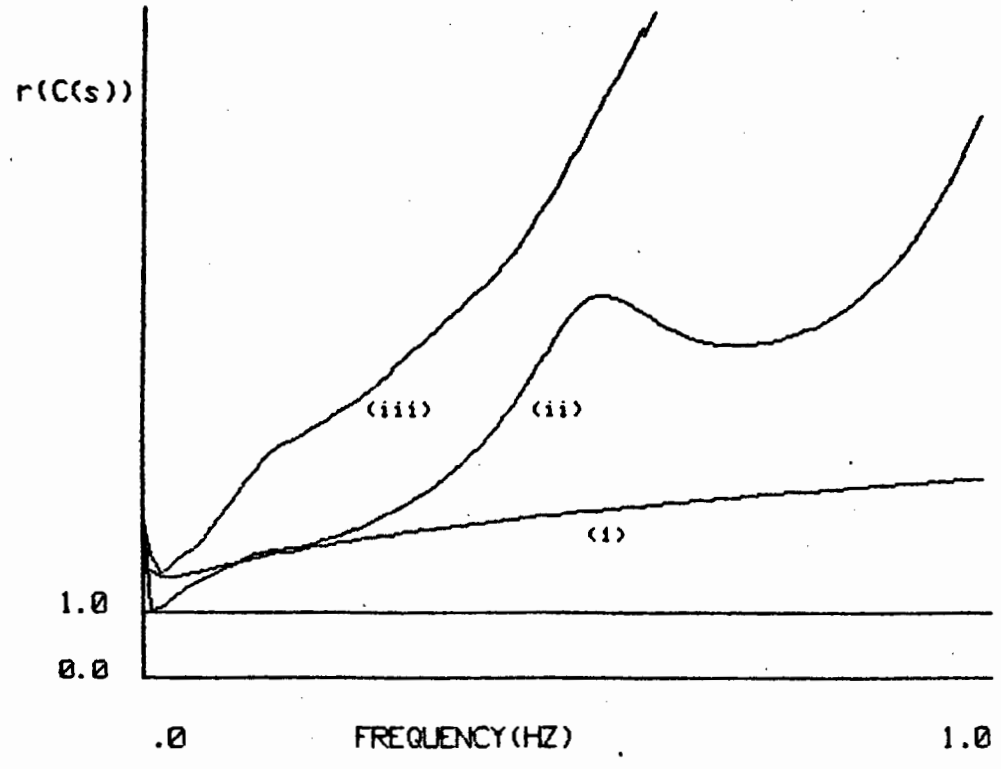

Figure 5.25. Plots of r(C(s)) vs frequency for different partitionings of the system represented by table 5.1.

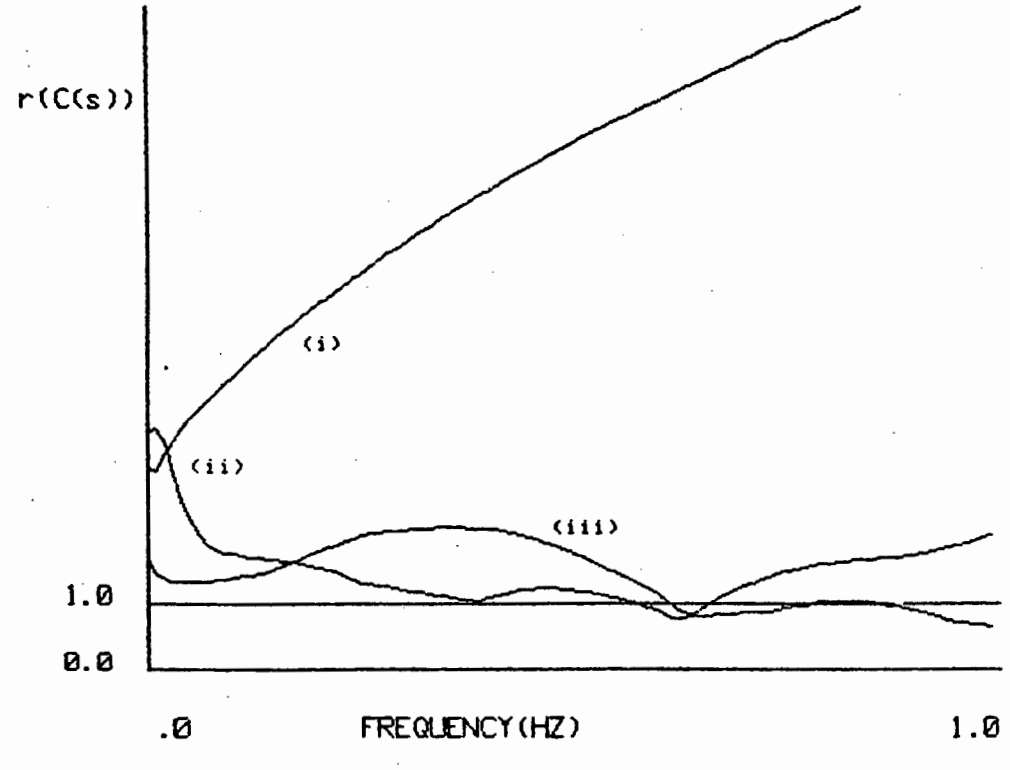

Figure 5.26. Plots of r(C(s)) vs frequency for different partitionings of the system represented by table 5.1 with rows one and four exchanged.

(5.53) for the different partitionings listed above, (i), (ii) and (iii). Figure 5.26 shows the plots of r(C(s)) vs frequency for the row-column arrangement of (5.54).

The plots in Fig. 5.25 show that at low frequencies all three partitionings yield low values of r(C(s)) for the row-column arrangement of (5.53). However for the two partitionings in (ii) and (iii) the interaction measure increases rapidly with frequency. The partitioning of (i), four 1x1 on-diagonal submatrices, also increases but more slowly staying close to r(C(s))=1.0 throughout the chosen frequency range. The designer would probably select the partitioning of (i) for this row-column arrangement.

Figure 5.26 shows the r(C(s)) vs frequency plots for different partitionings of the matrix with the row-column ordering of (5.54). The partitioning of (i) has a large corresponding value of r(C(s)) which increases rapidly with frequency. The blocking of (ii) is high at low frequencies but decreases to about r(C(s))=1.0 at high frequencies. The partitioning of (iii) gives rise to a low r(C(s)) at low frequencies and remains fairly low although the r(C(s)) value at high frequencies is higher than for the partitioning in (ii). Since the designer in the case of a distillation plant is probably more concerned with low frequencies the blocking of (iii) might well be selected for this row column arrangement.

The most probable choice for the designer then is the row-column ordering of the original matrix in table (5.1) and the partitioning that yields four 1x1 on-diagonal subsystems that are to be separately controlled. This is chosen in preference to the second row-column ordering and the partitioning of (iii) because control can be distributed further with the partitioning in (i). The designer would now design four single variable controllers for the subsystems,

$g_{11}(s)$, $g_{22}(s)$, $g_{33}(s)$ and $g_{44}(s)$. Using the stability theorems previously given to ensure that the system as a whole is stable.

Finally it is worth looking at the arrangement of rows and columns of the matrix in table 5.1 that is predicted to have very high values of r(C(s)). This will show that the method gives a good indication of which row-column arrangements are most unfavorable. Figure 5.27 shows the representation of the Z matrix for β=0.5 for a matrix formed by exchanging columns one and three, and four and two of the matrix in table 5.1. From this diagram it is seen that the most dominant elements in this matrix are positioned well off the diagonal instead of being clustered about the diagonal. Hence one would expect high values of the interaction measure for the partitionings already considered. Figure 5.28 shows the plots of r(C(s)) vs frequency for the partitionings previously defined. As can be seen all the partitionings give high values of r(C(s)) with very large values occurring at high frequencies which confirms the prediction made on the basis of Fig. 5.28.

5.4 SUMMARY

In this chapter an interaction measure, r(C(s)), was introduced as a criterion for choosing row-column orderings and matrix partitionings for distributed control. The interaction measure was linked to stability theorems that enable the stability of a system under distributed control to be tested. A drawback of these theorems is that they are only sufficient for stability and not necessary. Hence a design based on these theorems might well be very conservative. Yet another problem is that there exists no systematic way of selecting row-column orderings and matrix partitionings that are likely to give low values of the interaction measure.

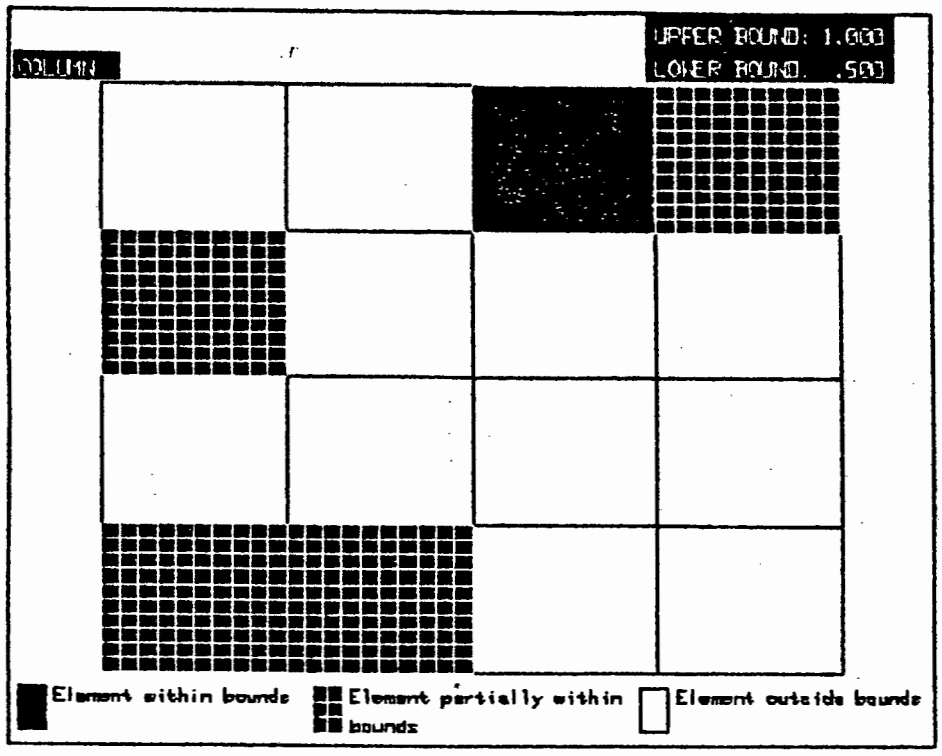

Figure 5.27. Diagram representing the Z matrix of the system represented by table 5.1 for β=0.5 after the matrix was rearranged to give high values of r(C(s)).

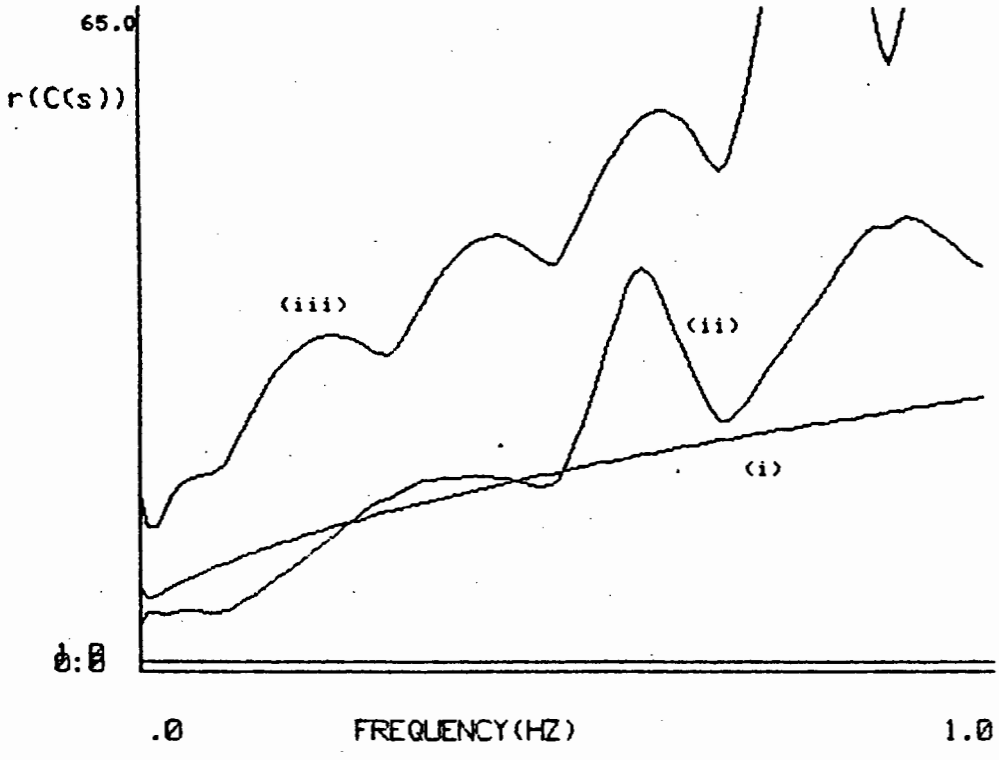

Figure 5.28. Plots of r(C(s)) vs frequency for different partitionings of the system represented by table 5.1 after being rearranged in an attempt to maximise r(C(s)).

A method of selecting row-column arrangements and partitionings was introduced. The method provided a graphical representation of the most dominant elements in the plant model, leaving the designer to rearrange the matrix. The method indeed gives some indication of which elements should be included in the on-diagonal blocks of the partitioned matrix and also indicates when the current row-column arrangement is unfavorable to distributed controller design.

CHAPTER 6

DESCRIPTION OF SOFTWARE

6.1 INTRODUCTION

In this chapter the software developed to test the theories and techniques of previous chapters will be examined. The programs discussed here were written to demonstrate the applicability of the theory and to perform calculations that would be very time consuming if performed by hand. An example of the later being eigenvalue calculations. The programs are not intended to constitute a CAD package for distributed controller design. However they have been made user friendly and designed to show the interactive nature of the techniques developed.

The software was developed on an HP 9000 mini-computer. The terminal used was a HP 150 series touchscreen terminal with graphics capability. The software was written entirely in a superset of FORTRAN 77 for which a compiler was available on the HP 9000. The graphics were all performed using a graphics package called STARBASE that was resident in the system.

There are three different programs. Each of them uses the same set of routines to enter the system matrix, and the matrix I/O routines are described separately. The matrix I/O program is menu driven allowing the user to enter a new matrix, save the current matrix or retrieve a previously saved matrix.

The first program is a multivariable simulator. The user

having entered the system model may now simulate its open or closed loop behaviour in response to step changes in the setpoint values for the different loops. The user may also specify any controllers that are to be applied by giving the name of the file in which the relevant controller matrix is stored. The program can display up to six graphs at a time. The user controls the scaling used and chooses which input or output variables are to be displayed.

The second program deals with the rearrangement of the system matrix to yield a matrix structure suitable for distributed control. The program calculates the M(s) and N(s) matrices over a user specified frequency range, M(s) and N(s) were defined in chapter four. The user can then display polar plots of M(s) or N(s). The program also calculates the Z matrix (chapter 5) for incremental values of the lower bound β, the increment being specified by the user. The Z matrix is displayed as a grid of rectangles. These are shaded to indicate the relative dominance of each element. The user can specify whether the relative row or column dominance is to be displayed. Further, after each increment the user can rearrange the rows and columns of the matrix and display the resulting Z matrix on the screen.

The final program calculates the Perron-Frobenius eigenvalue of the comparison matrix C(s), r(C(s)). The user specifies the partitioning of the system matrix and the frequency range over which the eigenvalue is to be calculated. The program displays a plot of the eigenvalue vs frequency. The user can also get the program to generate a plot of the product of r(C(s)) and $d_j$max (see chapter five).

Each of the three programs and the matrix I/O routines will be examined in more detail below. The listings for the subroutines used can be found in Appendix II. A list of subroutines for the matrix I/O module and each of the three

programs can be found in Appendices Ia, Ib, Ic and Id.

6.2 ENTERING AND EDITING MATRICES

The subroutines discussed in this section are used in each of the three programs. They enable the user to enter a matrix either from a file or from the keyboard. Once the matrix has been entered the user can either edit the current matrix or save it in a file. The subroutines allow the user to enter both a system matrix and a controller matrix.

The matrix is stored using five, one dimensional arrays and three constants. The three constants define the dimensions of the matrix and the maximum order of any element. The number of rows is stored in NI, the number of columns in NJ and the maximum order of any denominator or numerator polynomial is stored in NK. The order of the numerator of each element in the matrix is stored in a one dimensional array, M, which will contain (NIxNJ) elements. The denominator orders are similarly stored in D. The deadtimes for each element are also stored in a one dimensional array of order (NIxNJ). If the row and column indices of any element in the system matrix G(s) are i and j respectively then the corresponding element, n, in D or M is defined as

$$n = (i*j)+(j-1)*(NI-i)$$

The coefficients of the numerator polynomials are stored in the one dimensional matrix, GN, while those of the denominator are stored in GD. For an element $g_{ij}(s)$ the coefficient of s^m in the denominator or numerator will be stored in the Nnth element of GN and GD where Nn is

$$Nn = (n*m)+(m-1)*(NI*NJ-n)$$

When the matrix is stored it is placed in a sequential file. The order of the storage is, NI,NJ,NK,M,D,GD,GN,GD.

The user is initially presented with the matrix I/O menu.

(1) Retrieve an existing system stored in a file.
(2) Enter a new system matrix
(3) Save the current matrix
(4) Edit the current matrix
(5) Exit back to the main menu

If the first option is selected the user is prompted to enter the name of the file containing the matrix to be entered. If the file does not exist an error message is generated and the user prompted once again to enter the file name.

The third option, that of saving the current matrix performs in a manner similar to the first option. The user, on selecting this option, is asked to provide a file name of up to six characters. The program automatically prefixes the name entered with gs. hence if the user enters the name xxx the file name will be gs.xxx. This is done to enable the user to identify which files are data files for the program. If the user enters the name of an existing file an error message is generated and he is asked to enter a new file name.

If the user chooses to enter a new system he is asked to enter the number of rows, the number of columns and the maximum order of any element in the matrix. At this point he is asked if he wishes to change any of the variables that he has just entered. This is his only opportunity to edit these three variables. If the user does not wish to edit these variables he is passed to the matrix element edit menu. The

menu is headed with the row and column numbers of the current matrix element. The menu presents the user with the following options.

(1) Edit the numerator.
(2) Edit the denominator.
(3) Edit the dead time.
(4) Jump to the next element in the column.
(5) Select the row and column of the next element and jump to this element.
(6) Finish editing.

When the new matrix is set up before the user has begun to edit, the elements all have their numerators set to zero and their denominators set to one. This means that the user need only enter the non-zero elements.

When editing the numerator the user is asked to enter the order of the numerator and is then prompted to enter the coefficients of the numerator elements in ascending order. Each time the current value of the coefficient, or order, is displayed and if the user does not want to change any existing entry he pushes return and the value entered is the existing value.

Editing the denominator proceeds in similar fashion to the numerator. The deadtime is also entered in response to a prompt from the program once again the current value is displayed and this value can be retained by entering a return.

The user, when he has finished editing the current element, can move to a new matrix element in one of two ways. First he can jump to the next element down in the present column (or to the first element in the next column if he is currently editing the last element in the present column).

The other option is to enter the row and column indices of the element to which he wishes to jump.

When the user has finished editing the matrix he can then return to the matrix I/O menu.

If a matrix has been entered either from a file or by hand the user can choose to edit this matrix. The matrix edit element menu is displayed when the user chooses this option. The editing proceeds as for the entry of a new system except that the user cannot change the number of rows and columns and the maximum order permitted for any element.

Finally once the current system matrix is to the user's satisfaction the user can return to the main menu. A list of the subroutines used for entering, saving and editing matrices is given in Appendix Ia

6.3 THE SIMULATOR PROGRAM

The simulation program must perform two essential tasks. The first is to find the coordinates of the differential equations that correspond to the Laplacian transfer functions of the matrix elements, the state space equations. The second is to solve the differential equations at specified time intervals so that the behavior of the system vs time can be plotted.

The coefficients of the terms of the differential equations corresponding to a particular transfer function are determined directly using a method given in [23].Once these coefficients are established then the state space equations are solved over a user defined time interval using the well known Runge-Kutta method, see [24] for example.

The program uses the matrix I/O routines that were discussed in the previous section to enable the user to enter a system matrix and a controller if one is desired. Note that the user must store the controller matrix in a file before he can use it in the simulation.

The user starts the program by entering 'simulator' and return. The user the enters the system matrix and enters and stores any new controller matrices. The user the selects the simulator option on the main menu. The user is then presented with another menu. The main simulation menu.

(1) New simulation
(2) Redisplay the results of the last simulation performed.
(3) Return to main menu

If the user selects the first option the program asks the user if a controller is to be incorporated into the loop. If the user answers yes then the program prompts him to enter the name of the file containing the controller matrix.

The program now asks the user whether or not the current simulation and display parameters are to be changed. If the answer is no the simulation proceeds. If the user wishes to change the simulation and display parameters a menu listing the parameters is displayed. The options listed are

(1) Open/Closed loop
(2) Change time step dt
(3) Change duration of simulation
(4) Change setpoints
(5) Change display parameters
(6) Start simulation
(7) End simulation

If the user selects the first option then the current loop status is shown, 1=closed, 0=open, and the user is required to enter a new value. This is done for each loop in the system.The second option allows the user to change the value of the Runge-Kutta time step dt. The current value is displayed and the user enters the new value.

The third option allows the user to change the duration of the simulation. The present maximum time is displayed and the user can enter a new finishing time. The fourth option allows the user to change the setpoints for each loop. The current value of each setpoint is displayed and the user enters the new values from the key board.

The fifth option allows the user to change the parameters that govern which variables are to be displayed and the maximum and minimum values of the graphs to be displayed. Once the user selects this option he is presented with another menu with three available options.

(1) Change the variables to be displayed.
(2) Change the maximum and minimum values of the graph axes.
(3) Return to previous menu.

If the first option is chosen then the user is prompted to enter the number of graphs to be displayed on the screen, maximum number six. The user then enters the name of the variable to be plotted against time. Either output variables, y1, y2 etc. or input variables, u1,u2 etc.. If the user selects the second option the program prompts the user to enter the maximum and minimum points on the x and y axes of the graph of the variable selected. The final option returns the user to the previous menu.

The sixth option of the second menu starts the simulation. The axes and setpoints are drawn on the screen and the simulation begins. After each time step the points calculated are plotted on the display. The seventh option returns the user to the main simulation menu.

A list of the simulator subroutines is supplied in Appendix Ib together with a description of each subroutine. The listings for the subroutines are given in Appendix II.

6.4 MATRIX REARRANGEMENT PROGRAM

This program implements the techniques presented in chapters four and five. The program allows the user to plot polar diagrams of the elements of the M(s) and N(s) matrices over a user defined frequency range. The program will also draw a grid of rectangles which represents the Z matrix, introduced in chapter five, for a particular value of β.

The user enters the matrix to be analysed using the matrix I/O routines already discussed. The program then requires the user to enter the maximum and minimum frequencies of the frequency range to be considered and the number of points to be calculated in this frequency range. The program then presents the user with a list of options.

(1) Display G(jw), M(jw), N(jw) values
(2) Display polar plots
(3) Select a new frequency range
(3) Display Z matrix
(4) Save current matrix
(5) End

The first option displays the values of the elements in the

G(jw), M(jw) and N(jw) matrices. The user supplies the frequency and the program displays the values of the matrices calculated at the closest frequency to the one selected. The second option allows the user to plot polar diagrams of the M(s) or N(s) matrices. The program prompts the user to enter 'r' for the M(s) matrix (row normalised) and 'c' for the N(s) matrix (column normalised).

The third option allows the user to display the Z matrix corresponding to M(s) or N(s) and some value of β. The user initially selects whether he wants row or column normalisation, corresponding to the M(s) or N(s) matrices. He then enters the starting upper bound, usually one, and the amount by which β is to be incremented at each step. β is initially set to the same value as the upper bound. The program then displays the Z matrix for $\beta=0$. The rectangle corresponding to an element of M(s) or N(s) is shaded uniformly if the element falls completely in the range defined by the starting upper bound and β over the entire frequency range. If the element falls into the range over part of the frequency range then the rectangle is shaded and crosshatched. If the element lies entirely outside the range at all the frequencies considered then the rectangle is left blank.

The user is now given the option of exchanging rows and columns of the matrix being analysed. The user can also choose to return to the main menu after each step. If the user elects to continue the analysis then β is incremented and the next Z matrix is represented on the screen. The analysis will terminate when $\beta=0$.

The fourth option allows the user to store G(s) in its rearranged form if the user has exchanged any rows and columns. The user simply supplies the name of the file in which the matrix is to be stored. The final option

terminates the program . A list of the subroutines used in this program together with descriptions of them is given in Appendix Ic. The program listing is given in Appendix II.

6.5 INTERACTION MEASURE PROGRAM

This program calculates and displays the interaction measure r(C(s)) that was defined in chapter five. It also calculates and displays the product, r(C(s))*djmax vs frequency. The program uses a matrix inversion routine taken from [28]. The eigenvalues are calculated using a QR algorithm. The subroutine used to implement the QR algorithm and to reduce the original matrix to upper Hessenburg form are based on the ALGOL routines in [25].

The user enters the matrix to be analysed using the usual matrix I/O routines. The user then enters the frequency range and the number of points to calculate. The program then prompts the user to enter the number of on-diagonal blocks of the partitioned matrix. This done the user then must supply the order of each of these blocks. The program then calculates r(C(s)) and djmax at the frequencies defined by the user. A menu is then displayed with the following options.

(1) Plot r(C(s)) vs frequency.
(2) Plot djmax*r(C(s)) vs frequency.
(3) Select new partitioning

The first option allows the user to plot a graph of r(C(s)) vs frequency on the screen. The user can specify the maximum value of the y axis or else allow the program to perform the scaling. The second option plots a graph of djmax*r(C(s)) vs frequency on the screen. The third option allows the user to select a new partitioning. The program then recalculates

r(C(s)) and djmax and redisplays the previous menu.

CHAPTER 7

CONCLUSION

7.1 GENERAL PROBLEMS ASSOCIATED WITH DISTRIBUTED CONTROLLER DESIGN

In the course of the research reported in this thesis it has become evident that there are a number of limitations associated with decentralised controller design. These are either due to the general concept of distributed control or are due to current limitations in control theory dealing with decentralised systems.

One of the most serious drawbacks of decentralised controller design is that all the open loop unstable elements of the system must fall in the controlled subsystems. In a system with unstable poles this limits the choice of which subsystems are to be controlled. There would seem to be no way of stabilising unstable elements that are not included in the controlled subsystems, using decentralised control. The only exception would seem to be the case where pole zero cancellation occurs between the on-diagonal elements of the system and the off-diagonal elements. This limits the systems to which decentralised control may be applied.

In this thesis a number of theorems have been presented that allow the stability of the system under decentralised control to be checked. However as pointed out in chapter two these are all very conservative. The theorem connected with the r(C(s)) interaction measure is currently the most useful of the stability theorems but because it provides a sufficient rather than necessary condition for stability it

is very conservative. A less conservative interaction measure, the μ interaction measure, has been proposed but this is currently impossible to calculate for systems of higher order than three.

7.2 SIMPLE STRUCTURES

The analysis of a matrix to see if it could be arranged to give a block triangular or block diagonal form was carried out in this thesis. It was shown that the use of these simple structures provides the designer with a powerful design tool that can be applied before the system has been completely modelled. A theorem was presented which ensures the stability of a block triangular or diagonal system under decentralised control provided that there are no unstable poles in uncontrolled subsystems. Two algorithms were developed that allow the designer to rearrange a matrix into a block triangular or diagonal form, if this is possible. These algorithms are very simple to implement requiring only addition and sorting.

7.3 DIAGONAL DOMINANCE

A method was also developed to enable the designer to rearrange a matrix to be diagonally dominant, if such rearrangement is possible. Central to this rearrangement is a representation of the system matrix based on Gershgorin's theorem. Two matrices the M(s) and N(s) matrices are generated. These matrices contain only positive elements all of which lie between one and zero. If an element in the M(s) matrix has a value greater than 0.5 then it is dominant on its row. If an element in the N(s) matrix has a value greater than 0.5 then it is dominant on its column. The values of the individual elements are invariant with row and/or column rearrangement. A program has been written that

displays an array of polar diagrams of the elements of M(s) or N(s). A circle of radius 0.5 is plotted on each of these diagrams. If the plot of any element lies outside the 0.5 circle for all frequencies then the element is dominant. The user can use this representation to rearrange the rows and columns of the matrix.

This method of rearranging the system matrix is effective and easy to use. One problem with this method is that scaling is not taken into account. A matrix which is not diagonally dominant may often be made so by scaling the matrix by multiplying it with another, diagonal matrix. This is equivalent to scaling the inputs or outputs of the system. The problem of scaling was considered. A test was developed to enable the designer to check if it is possible to make his system diagonally dominant. The test is limited in that the principle minors of the matrix have to be positive and the permanent of the matrix has to be calculated. The later requires numerical techniques to be developed as direct determination is not feasible for large systems.

7.4 THE PERRON-FROBENIUS EIGENVALUE INTERACTION MEASURE

The use of the Perron-Frobenius eigenvalue of a comparison matrix as an indication of the suitability of a particular row-column ordering and matrix partitioning for distributed control was considered. A program was written to calculate the interaction measure and to plot the measure vs frequency. A method was then proposed to allow the designer to choose those row-column orderings and partitionings that would give a low value of the interaction measure. This method was based on the M(s) and N(s) matrices previously used to detect diagonal dominance. The method developed was successful in giving the designer some indication of the elements that should be incorporated into the on-diagonal

blocks. However the method proved to be awkward to use and the interpretation of the displayed diagrams was difficult and often ambiguous.

The problem of selecting appropriate subsystems for decentralised control when the system is not block diagonal or block triangular remains largely unsolved. The development of less conservative stability theorems will be useful in checking for stability but the problem of selecting the appropriate subsystems will remain. Future research in this area should concentrate on two aspects. First the development of stability theorems that provide necessary conditions for decentralised stability. These theorems should be developed with interactive controller design in mind and hence be associated with some form of graphical display. This display should enable the designer to see the effect that a controller designed for an individual subsystem has on overall system stability. Secondly methods of systematically rearranging the system matrix to get a matrix structure most likely to satisfy the stability conditions must be developed. One possible approach to this problem might be to define an interaction measure, such as r(C(s)), and then use linear programming techniques to try and minimise this measure.

7.5 SUMMING UP

The research reported in this thesis has resulted in the development of theory that will prove useful to control engineers designing large scale decentralised systems. The simple structure analysis presented is particularly useful, and provides a powerful tool for decentralised controller design.

The M(s) and N(s) matrices provide a new method of rearranging the system matrix to be diagonally dominant that

is quick and easy to use. The method operates over any chosen frequency range and hence is particularly suited for use with INA type design techniques. One drawback to the method is that it is not invariant under scaling. The scaling problem was considered and test to see if a matrix can be scaled to be diagonally dominant was developed.

The method of rearranging the system matrix to minimise the r(C(s)) interaction matrix is not completely satisfactory. However there is currently no other method of performing this rearrangement and the method at least gives some help to the designer when choosing the subsystems that he wishes to control.

REFERENCES

[1] **LIMEBEER, D.J.N.:** 'The application of generalised diagonal dominance to linear system stability theory', Int. J. Control, 1982, Vol. 36, No.2, pp. 185-212

[2] **LIMEBEER, D.J.N.:** 'An inclusion region for the eigenvalues of partitioned matrices', Int. J. Control, 1983, Vol. 37, No.2, pp. 429-436

[3] **LIMEBEER, D.J.N., and HUNG, Y.S.:** 'The robust stability of interconnected systems', IEEE Trans. Aut. Control, 1983, Vol. AC-28, No.6, pp. 710-716

[4] **LIMEBEER, D.J.N., and HUNG, Y.S.:** 'Robust stability of additively perturbed interconnected systems', IEEE Trans. Aut. Control, 1984, Vol. AC-29, No.12, pp. 1069-1075

[5] **NWOKAH, O.D.I.:** 'The robust decentralised stabilisation of complex feedback systems', IEE Proc., 1987, Vol. 134, Pt. D, No.1, pp. 43-47

[6] **OHTA, Y., SILJAK, D.D., and MATSUMOTO, T.:** 'Decentralized control using quasi-block diagonal dominance of transfer function matrices', IEEE Trans., 1986, Vol. AC-31, No.5, pp. 420-429

[7] **CALLIER, F.M., CHAN, W.S., and DESOER, C.A.:** 'Input-output stability of interconnected systems using decompositions: an improved formulation', IEEE Trans. Aut. Control, 1978, Vol. AC-23, No.2, pp. 150-162

[8] **WALSTROM, B., JUUSELA, A., OLLUS, M., NARVAINEN, P., LEHMUS, I., and LONNQVIST, P.:** 'A distributed control system and its application to a board mill', AUTOMATICA, 1983, Vol. 19, No.1, pp.1-14

[9] **KRAMER, J., MAGEE, J., and SLOMAN, M.:** 'A software architecture for distributed computer control systems', AUTOMATICA, 1984, Vol. 20, No.1, pp.93-102

[10] **GROSDIDIER, P., and MORARI M.:** 'Interaction measures for systems under decentralized control', AUTOMATICA, 1986, Vol. 22, No.3, pp. 309-319

[11] **GROSDIDIER, P., and MORARI M.:** 'The μ Interaction measure', Ind. Eng. Chem. Res., 1987, Vol. 26, pp. 1193-1202

[12] **ROSENBROCK, H.H.:** 'Computer aided control system design', Academic Press, New York, 1974

[13] **FEINGOLD, D.G., and VARGA, R.S.:** 'Block diagonally dominant matrices and generalizations of the Gerschgorin circle theorem', Pacific J. Appl. Math., 1962, Vol.12, pp. 1241-1250

[14] **BERMAN, A., PLEMMONS, T.J.:** 'Non-negative matrices in the mathematical sciences', Academic Press, New York, 1975

[15] **DOYLE, J.:** 'Analysis of feedback systems with structured uncertainties', IEE Proc., Vol.129, Pt. D, pp 242-250

[16] **MANOUSIOUTHAKIS, V., SAVAGE, R., and ARKUN, Y.:** 'Synthesis of decentralised process control structures using the concept of block relative gain', AICHE J., 1986, Vol. 32, No.6

[17] **SAGE, A.P.:** 'Methodology for large scale systems', McGrawHill, New York, 1977

[18] **BRYANT, G.F., and YEUNG, L.F.:** 'Methods and concepts of dominance optimisation', IEE Proc., 1983, Vol. 130, Pt. D, No.2, pp 72-82

[19] **KUSIAK, A., and CHOW, W.S. :** 'An efficient cluster identification algorithm',IEEE Trans. Syst., Man, Cybern, 1987, Vol. smc-17, No.4, pp. 696-699

[20] **HULBERT, D.J., KOUDSTAAL, J., BRAAE, M., AND GOSSMAN, G. I.:** 'Multivariable Control of an industrial grinding circuit', 3rd IFAC Symposium on Automation in Mining, Mineral and Metal Processing, Montreal (Canada), 1980.

[21] **LUYBEN, W.L.:** 'Simple method for tuning SISO controllers in multivariable systems', Ind. Eng. Chem Process Des. Dev., 1980, Vol. 25, pp 654-660

[22] **HULBERT, D.G., and BRAAE, M.:** 'Multivariable control of a milling circuit at East Driefontein gold mine', NIM Report, Measurement and Control Division, No. 2113, 14 August 1981.

[23] **BRAAE, M., and HAMES, J.:** 'Evaluation by PC of pole-zero configurations in control systems', Research Review dept. Electrical & Electronic Engineering U.C.T., April & May 1986, Vol. 11, No.3, pp 110-117.

[24] **RAINVILLE, E.D.,and BEDIENT, P.E.:** 'Elementary differential equations (sixth edition)', Macmillan Publishing Company, New York, 1981

[25] **WILKINSON, J.H., and REINSCH:** 'Linear Algebra', Springer-Verlag, Berlin, 1971

[26] HP-UX Concepts and Tutorials, Vol. 6, 1895, Hewlet-Packard Company

[27] **ÅSTRÖM, K.J., WITTENMARK, B.:** 'Computer controlled systems', Prentice-Hall, Inc, Englewood Cliffs, N.J., 1984

[28] **CONTE, S.D., and De BLOOR, C.:** 'Elementary numerical analysis', McGraw-Hill Kogakusha, Tokyo, 1972.

BIBLIOGRAPHY

(1) **ÅSTRÖM, K.J., WITTENMARK, B.:** 'Computer controlled systems', Prentice-Hall, Inc, Englewood Cliffs, N.J., 1984

(2) **BARNETT, S., and STOREY, C.:** 'Matrix methods in stability theory', Nelson and Sons Ltd., Great Britain, 1970

(3) **BELL, D.J., COOK, P.A., and MUNRO, N.:** 'Design of modern control systems', Peter Peregrinus Ltd., Exeter, 1982

(4) **BERMAN, A., PLEMMONS, T.J.:** 'Non-negative matrices in the mathematical sciences', Academic Press, New York, 1975

(5) **CONTE, S.D., and De BLOOR, C.:** 'Elementary numerical analysis', McGraw-Hill Kogakusha, Tokyo, 1972.

(6) **DORF, R.C.:** 'Modern control systems (3rd edition)', Addison-Wesley, London, 1980

(7) **GEAR, A.B.J., and BRAAE, M.:** 'Design of robust distributions for large-scale control schemes', paper presented at 3rd SACAC Workshop on Multivariable Control.

(8) **GEAR, A.B.J.:** 'Design of robust decentralised controllers for large scale systems', paper accepted for SOCCO 88, 5th IFAC/IFIP Symposium on Software for Computer Control

(9) **GOURLAY, A.R., and WATSON, G.A.:** 'Computational methods for matrix eigen problems', John Wiley and Sons, New York, 1973

(10) **KREYSZIG, E.:** 'Advanced Engineering Mathematics', John Wiley and Sons Inc., New York, 1972

(11) **LANCASTER, P. :** 'Theory of Matrices', Academic Press, New York, 1969

(12) **MUNAKATA, T.:** 'Matrices and linear programming with applications', Holden-Day Inc., San Francisco, 1979

(13) **PERSE, M.C.:** 'Methods of matrix algebra',Academic Press, New York, 1965

(14) **RAINVILLE, E.D.,and BEDIENT, P.E.:** 'Elementary differential equations (sixth edition)', Macmillan Publishing Company, New York, 1981

(15) **RALSTON, A., and WILF, H.S.:** 'Mathematical methods for digital computers',John Wiley and Sons Inc., New York, 1964

(16) **ROSENBROCK, H.H.:** 'Computer aided control system design', Academic Press, New York, 1974

(17) **SENATA, E.:** 'Non-negative matrices', George Allen and Unwin Ltd., London, 1973

(18) **SINGH, M.G., and TITLI, A.:** 'Systems decomposition, optimisation and control', Pergamon Press, London, 1978

(19) **STEPHANOPOULOS, G.:** 'Chemical process control. An introduction to theory and practice', Prentice-Hall, New Jersey, 1984

(20) **STOER, J., and BULIRSCH, R.:** 'Introduction to numerical methods',Springer-Verlag, Heidelberg, 1976

(21) **WILKINSON, J.H., and REINSCH:** 'Linear Algebra', Springer-Verlag, Berlin, 1971

(22) **YOUNG, D.M., and GREGORY, R.T.:** 'A survey of numerical mathematics vol. II', Addison-Wesely, Reading, Massachusetts, 1973

APPENDICES

APPENDIX Ia
MATRIX EDITING SUBROUTINES

SUBROUTINE	DESCRIPTION
choice1	CALLS: menu, old_system, new_system, save_system, edit_matrix, This subroutine calls menu to print out the matrix I/O menu for entering the system matrix G(s). It then reads the user's option. Depending on the option chosen the subroutine then calls either old_system, new_system, save_system or edit_matrix
choice4	CALLS: menu, old_system, new_system, save_system, edit_matrix This subroutine calls menu to print out the matrix I/O menu for entering the controller matrix K(s). It then reads the user's option. Depending on the option chosen the subroutine then calls either old_system, new_system, save_system or edit_matrix
DEAD_read	CALLS: string_to_number(a function) This subroutine displays the deadtime of the current i,j element and waits for the user to enter a new value. If the user enters a return or a non-numeric value then the current deadtime is retained.
denordread	CALLS: string_to_number(a function) This subroutine displays the order of the denominator of the current i,j element and reads in the new value from the keyboard. If the user enters a non-numeric input or just presses return then the old value is retained.

SUBROUTINE	DESCRIPTION
den_read	CALLS: string_to_number(a function) This subroutine displays each coefficient of the denominator of the current i,j element, in turn, in ascending order. After each coefficient has been displayed the subroutine reads the new value of the coefficient from the keyboard. If the user enters a non-numeric input or just presses return then the old value is retained.
edit_matrix	CALLS: numordread, num_read, denordread, den_read, dead_time, errorchk This subroutine displays a menu which lists the options available for editing the current element of the matrix. The element row and column numbers are also displayed. The subroutine then reads the user's option from the keyboard and calls the appropriate subroutine.
errorchk	CALLS: This subroutine displays one of several error messages depending on the value of a parameter that is passed from the calling subroutine.
mat_param	CALLS: read_NI, read_NJ, read_NK This subroutine calls three other subroutines that read in the number of rows, NI, the number of columns, NJ and the maximum order of any denominator or numerator polynomial. The subroutine then checks to see if the user wishes to change the values entered. If the user does wish to change the values,the subroutine loops and calls the three subroutines again.
menu	CALLS: This subroutine displays a menu on the screen. The subroutine displays one of four menus depending on the value of a parameter, menu_no, passed from the calling subroutine. This subroutine is also used by the simulator program.

SUBROUTINE	DESCRIPTION
new_system	CALLS: mat_param, edit_matrix This subroutine clears the screen and then calls mat_param which allows the user to define the dimensions of the new matrix to be entered. The subroutine then sets all of the denominators of the new matrix to 1.0. The subroutine then calls edit_matrix to allow the user to enter the elements of the matrix.
numordread	CALLS: string_to_number(a function) This subroutine displays the order of the numerator of the current i,j element and reads in the new value from the keyboard. If the user enters a non-numeric input or just presses return then the old value is retained.
num_read	CALLS: string_to_number(a function) This subroutine displays each coefficient of the numerator of the current i,j element, in turn, in ascending order. After each coefficient has been displayed the subroutine reads the new value of the coefficient from the keyboard. If the user enters a non-numeric input or just presses return then the old value is retained.
old_system	CALLS: errorchk This subroutine reads the name of a file from the keyboard. It attempts to access this file. If it succeeds then the subroutine transfers the contents of the files into the variables that define the dimensions of the matrix and the arrays that store the orders and coefficients of the denominator and numerator polynomials. If the file named does not exist an error is generated and the user required to enter another file name.

SUBROUTINE	DESCRIPTION
read_NI	CALLS: errorchk This subroutine reads the number of rows from the keyboard.
read_NJ	CALLS: errorchk This subroutine reads the number of columns from the keyboard.
read_NK	CALLS: errorchk This subroutine reads the maximum order of any denominator or numerator polynomial from the keyboard.
save_system	CALLS: errorchk This subroutine reads the name of a file (maximum length six characters) and adds the prefix gs. to the name. It then opens a file of this name and saves the dimensions of the current matrix and the matrix elements in the file. If the file already exists an error is generated and the user required to enter another file name.
strtonum	CALLS: This function is passed a string of characters and a real number. It then converts the string into a real number if this is possible and returns this value. If the string does not correspond to a real number then the real number that was passed to the function is returned.

APPENDIX Ib
SIMULATOR SUBROUTINES

Subroutines listed are only those that were written for this project. More information on the supplied graphics routines, marked with (SB) can be found in [26].

SUBROUTINE	DESCRIPTION
Matrix I/O routines	See Appendix Ia
chg_maxmin	CALLS: errorchk This subroutine enables the user to enter the maximum and minimum x and y values for each of the axes to be drawn.
chgsim_deflt	CALLS: errorchk, simdisp_init This subroutine allows the user to change the simulation parameters. It displays a menu and the user selects which parameter he wishes to change. The old value of the variable is then displayed and the user enters the new value from the keyboard. If the user wishes to change the variables to be displayed or to change the x or y axes then simdisp_init is called.
chg_simvar	CALLS: errorchk This routine allows the user to select which variables are to be displayed on the screen. The subroutine reads the number of variables to be displayed and then reads the name of each variable in turn. The names are then stored in the arrays usim and ysim.
control_set-up	CALLS:errorchk This subroutine checks to see if the user wishes to use a controller. If the user indicates that a controller is to be used then the name of the file in which the controller matrix is stored is read from the keyboard.

SUBROUTINE	DESCRIPTION
display_sim	CALLS: sim_inital, draw_axes,vdc_extent(SB) , clip_rectangle(SB), mapping_mode(SB) , plot_graphs, polyline(SB) This subroutine displays the results of the last simulation performed, which have been stored in a file. The user is given the opportunity to change the plotting parameter before the graphs are plotted.
draw_axes	CALLS: vdc_extent(SB), clip_rectangle(SB), mapping-mode(SB),move2d(SB), draw2d(SB), clear_view_surfaces(SB), This subroutine calculates the position and scaling of the axes for the graphs to be displayed and draws these axes on the screen The subroutine also draws the setpoints in on the graphs.
errorchk	CALLS: This subroutine displays one of several error messages depending on the value of a parameter that is passed from the calling subroutine.
eval_control	CALLS: eval_CX, eval_CY This subroutine reads each controller from file into RAM. The subroutine then calls subroutines that calculate the outputs from the controller.
eval_CX	CALLS: This subroutine calculates the coefficients of the differential equations that correspond to a laplacian transfer function of the controller. These are then used to solve the differential equations at a time t, using the Runge-Kutta method.
eval_CY	CALLS: This subroutine uses the solutions to the differential equations found by eval_CX to determine the outputs of the controller at time t.

SUBROUTINE	DESCRIPTION
eval_X	CALLS: This subroutine calculates the coefficients of the differential equations that correspond to a laplacian transfer function of the system matrix. These are then used to solve the differential equations at a time t,using the Runge-Kutta method.
eval_Y	CALLS: This subroutine uses the solutions to the differential equations found by eval_X to determine the outputs of the system at time t.
plot_graphs	CALLS: This subroutine calculates the co-ordinates of the input and output variables that are to be plotted on the screen and that have been retrieved from a file
plot_points	CALLS: This subroutine calculates the co-ordinates of the input and output variables that are to be plotted on the screen. That have just been calculated.
simdisp_init	CALLS:errorchk, menu, chg_simvar, chg_maxmin This subroutine enables the user to choose whether to change the variables that are to be plotted on the screen or to change the maximum and minimum values represented on the axes.
sim_inital	CALLS: errorchk, chgsim_deflt This subroutine enables the designer to set up default values for the various simulator variables. If the user wishes to change any of the simulator parameters the subroutine calls chgsim_deflt.

SUBROUTINE	DESCRIPTION
sim_mv	CALLS: control_setup, sim_initial, vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB), line_type(SB), eval_X, eval_Y, eval_control, plot_points, move2d(SB), draw2d(SB) make_picture_current(SB) This subroutine controls the flow of the actual simulation. It calls subroutines that enable the user to set up the simulation parameters and to include controllers. After each time step the subroutine plots the latest calculated values on the screen.
simulator	CALLS: errorchk ,menu ,sim_mv , display_sim This subroutine calls menu to display a list of options. The subroutine then reads the option entered from the keyboard and calls the appropriate subroutine.
start	CALLS: errorchk, menu, choice1, choice4, simulator This is the main program. It displays a list of options that the user may implement and calls the appropriate subroutine in response to the entry made by the user.

APPENDIX Ic
MATRIX REARRANGEMENT SUBROUTINES

Subroutines listed are only those that were written for this project. More information on the supplied graphics routines, marked with (SB) can be found in [26].

SUBROUTINE	DESCRIPTION
Matrix I/O routines	See Appendix Ia
axes	CALLS: graphplace This subroutine determines the virtual device coordinates for the origins of the axes upon which the polar plots of M(s) and N(s) are to be plotted. It also calculates the maximum and minimum axes values in virtual device coordinates.
bound_entry	CALLS: This subroutine checks to see if the current i,j element is still in the bounds if not the i,j element in temparray is set to -1.
bound_inc	CALLS: sort_bounds, displ_bounds, change_row , change_col This subroutine reads the starting upper bound from the keyboard and the amount by which it is to be incremented each step. The subroutine calls sort_bounds to establish which elements are in the bounds for each step. The user can choose to reorder the rows and columns after each step in which case change_row and change_col are called. After each iteration displ_bounds is called to show which elements are in the bounds.
change_col	CALLS: This subroutine changes the array of pointers that keep track of the elements of G(s) and the elements of bound_array which record the relative dominance of each element. The user enters the two columns to be swapped and the subroutine reorders the pointers and the matrix.

SUBROUTINE	DESCRIPTION
change_row	CALLS: This subroutine changes the array of pointers that keep track of the elements of G(s) and the elements of bound_array which record the relative dominance of each element. The user enters the two rows to be swapped and the subroutine reorders the pointers and the matrix.
check_bounds	CALLS: This subroutine checks to see if a matrix element falls in or out of the current upper and lower bounds.
col_sum	CALLS: This subroutine calculates the sum of the elements in a column of a matrix.
displ_bounds	CALLS: vdc_extent(SB), mapping_mode(SB) clip_rectangle(SB), text_2d(SB) interior_style(SB), draw2d(SB) character_height(SB), move2d(SB), clear_view_surface(SB), rectangle(SB) , text_color_index(SB), draw_grid, , line_color_index(SB), fillrectangle This subroutine draws the borders and writes the text for the diagram that shows the relative dominances of the elements. It also calls the subroutine that draws the a grid of rectangles on the screen. It also calls the subroutine that shades the rectangles or crosshatches them, depending on the relative dominance of the corresponding element in G(s).
displ_matrix	CALLS: matrix_write This subroutine reads the value of the frequency at which the elements of G(s), N(s) and M(s) are to be displayed. It then calls a subroutine that writes the values of the elements of these matrices to the screen.

SUBROUTINE	DESCRIPTION
draw_circles	CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB), move2d(SB), draw2d(SB) This subroutine draws a circle on the screen according to x-y coordinates supplied by the user.
draw_grid	CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB), move2d(SB), draw2d(SB), line_type(SB), This subroutine draws a grid of rectangles on the screen.
fillrec- -tangle	CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB), move2d(SB), draw2d(SB), line_type(SB), perimeter_color_index(SB), interior_style(SB), rectangle(SB), line_color_index(SB) This subroutine colours in a rectangle on the screen. It either fills in the rectangle and returns or else fills in the rectangle and then crosshatches it depending on a user supplied parameter.
GIJ_calc	CALLS: tran_eval This subroutine calculates the value of each element of G(s) at points over a user specified frequency range and stores their absolute values in an array.
graphplace	CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB), move2d(SB), draw2d(SB), line_type(SB), interior_style(SB), rectangle(SB), line_color_index(SB) This subroutine draws the axes of a graph on the screen.
matrix_write	CALLS: This subroutine writes out the values of a matrix on the screen.

SUBROUTINE	DESCRIPTION
norm_cols	CALLS: This subroutine divides each each element in a matrix by the sum of the elements in the column in which the element falls.
norm_rows	CALLS: This subroutine divides each each element in a matrix by the sum of the elements in the row in which the element falls.
options	CALLS: This subroutine displays a list of options available to the user and reads the user's response from the keyboard.
plot_matrix	CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB), move2d(SB), draw2d(SB) This subroutine plots the polar diagrams of the elements of a matrix on the screen.
row_sum	CALLS: This subroutine calculates the sum of the elements in a row of a matrix.
scale_plot	CALLS: axes, draw_circles, plot_matrix This subroutine enables the user to plot polar diagrams of the M(s) and N(s) matrices . The subroutine reads a user supplied character from the key board to decide which set of polar plots to plot.
scale_prog	CALLS: GIJ_calc, row_sum, col_sum, norm_cols , norm_rows, options, displ_matrix , scale_plot, bound_inc, swap_save This the main program for the matrix sorting routines. It controls the program flow and calls subroutines that implement options chosen by the user.

SUBROUTINE	DESCRIPTION
sort_bounds	CALLS: check_bounds, bound_entry This subroutine sets up an array whose elements are either 1, 0 or -1. The elements of the array correspond to the elements of M(s) or N(s)over a frequency range. The value of an element depends on whether or the $m_{ij}(s)$ or $n_{ij}(s)$ element falls into predetermined bounds.
swap_save	CALLS: errorchk This subroutine stores the elements of the G(s) matrix in a file. The order of the rows and columns of the saved matrix is determined by the pointer array.
tran_eval	CALLS: This subroutine evaluates the value of a laplacian transfer function for a particular frequency.

APPENDIX Id
INTERACTION MEASURE PROGRAM

Subroutines listed are only those that were written for this project. More information on the supplied graphics routines, marked with (SB) can be found in [26].

SUBROUTINE	DESCRIPTION
Matrix I/O routines	See Appendix Ia
complex_ - matrix	CALLS: This subroutine reads the matrix stored in a one dimensional array into a two dimensional array.
define_part	CALLS: This subroutine reads the number of on-diagonal submatrices of the partitioned system and the order of each block.
dlam_plot	CALLS: vdc_extent(SB), clip_rectangle(SB) mapping_mode(SB), character_height(SB) line_type(SB), move2d(SB), draw2d(SB) text_path(SB), text2d(SB) This subroutine plots the product of r(C(s)) and djmax on the screen.
dnorm	CALLS: invert_matrix, matrix_mult, norm_matrix, errorchk This subroutine calculates $\|G_{jj}(I+G_{jj})^{-1}\|$
eigen_plot	CALLS: vdc_extent(SB), clip_rectangle(SB) mapping_mode(SB), character_height(SB) line_type(SB), move2d(SB), draw2d(SB) text_path(SB), text2d(SB) This subroutine plots the eigenvalue r(C(s)) vs frequency.

SUBROUTINE	DESCRIPTION
eigen_prog	CALLS: matrix_read, choice1(appendix 1a), define_part, trans_calc,dlam_plot, complex_matrix, lamdapff,eigen_plot This is the main program. It calls subroutines to read in the matrix G(s) and to determine the matrix partitioning required. The program also calls subroutines that calculate r(C(s)) and r(C(s))*djmax. Finally subroutines are called that plot these values on the screen vs frequency.
eval_tran	CALLS: This subroutine evaluates a laplacian transfer function at a particular frequency.
hess_red	CALLS: This subroutine calculates the hessenburg reduction of the matrix G(jw).
invert_mat--rix	CALLS: This subroutine inverts a square matrix if possible. If the matrix is singular then a warning is generated.
lamdapf	CALLS: submatrix, dnorm, invert_matrix, matrix_mult, norm_matrix, errorchk, qr_eigenvalues This subroutine calls subroutines to calculate the matrix C(jw) and the eigenvalue r(C(s)). It also calculates djmax
matrix_mult	CALLS: This subroutine calculates the product of two matrices
norm_matrix	CALLS: This subroutine calculates the row sum norm of a matrix.

SUBROUTINE	DESCRIPTION
qr_- -eigenvalues	CALLS: hess_red, qr This subroutine calls subroutines that calculate the hessenburg reduction of a matrix and the eigenvalues of the reduced matrix.
qr	CALLS: This subroutine calculates the eigenvalues of a complex matrix using the QR algorithm
submatrix	CALLS: This subroutine stores a submatrix from G(s) in an array so that it can be separately manipulated.
trans_calc	CALLS: eval_tran This subroutine evaluates the elements of G(s) over a user specified frequency range and stores the results in an array.

APPENDIX II

PROGRAM LISTINGS

INCLUDE FILES

```
C ***************************************************************************
C
C                    COMMON1
C
C ***************************************************************************
C ***************************************************************************
C *                                                                         *
C * An include file with common system variables.                           *
C *                                                                         *
C ***************************************************************************


      common M,D,GN,GD,GT,NI,NJ,NK
      integer NI,NJ,NK,M(900),D(900)
      real GN(2700),GD(2700),GT(900)

C ***************************************************************************
C
C                           COMMON2
C***************************************************************************
C
C***************************************************************************
C*                                                                         *
C* An include file with common simulator variables                         *
C*                                                                         *
C***************************************************************************


      common/com2/y0,x0,dt,sig,t0,t1,nlag,loop,y,sx,dx,u,display,x
     c            ,ndisplay,ydisp,udisp,ywindow,uwindow,ysim,usim
     c            ,ud,origins,setpoint,ydold,ydnew,yd,udold,udnew
     c            ,settime,first_sim

      real y0(900),dt,t0,t1,nlag(30),dx(10),sx(10),yd(30)
     c     ,u(30),ywindow(8),uwindow(8),ysim(210),ydnew(30),
     c      usim(210),origins(40),ud(3000),y,setpoint(30),ydold(30)
     c     ,udold(30),udnew(30),x0(1800),x(10),settime(30)
      integer loop(30),ndisplay,ydisp(8),udisp(8),first_sim
      character*4 sig,display
```

```
C*****************************************************************************
C
C                                  COMMON3
C
C*****************************************************************************
C*****************************************************************************
C*                                                                           *
C* An include file with matrix I/O common variables.                         *
C*                                                                           *
C*****************************************************************************

      integer norows,nocols,maxord,Nord(1),Dord(1)
      real numval(1),denval(1),dtime(1)


C  ***************************************************************************
C
C                                  COMMON 4
C
C  ***************************************************************************
C
C  ***************************************************************************
C  *                                                                         *
C  * An include file with common controller variables.                       *
C  *                                                                         *
C  ***************************************************************************

      common /com4/CNI,CNJ,CNK,CM,CD,CGN,CGD,CGT,x,x0
      integer CNI,CNJ,CNK,CM(900),CD(900),max_x
      real CGN(2700),CGD(2700),CGT(900),x(10),x0(16000)
```

MATRIX I/O LISTINGS

```
C ***********************************************************************
C
C                    SUBROUTINE choice1
C
C ***********************************************************************

C ***********************************************************************
C *                                                                     *
C * CALLS: menu, old_system, new_system,save_system, edit_matrix       *
C *                                                                     *
C * DESCRIPTION: This subroutine calls menu to print out the matrix    *
C *              I/O menu for entering the controller matrix K(s). It  *
C *              then reads the user's option. Depending on the         *
C *              option chosen the subroutine then calls one of the     *
C *              subroutines listed above.                              *
C *                                                                     *
C ***********************************************************************

      subroutine choice1
      include 'common1'
      integer menu_no,error,on,off,exit,l
      character*4 choice,cls
      cls=char(27)//'H'//char(27)//'J'
      off=0
      on=1
      menu_no=2
      write(*,110)cls
      exit=off
      do while(exit.ne.on)
      error=on
      do while (error.eq.on)
        write(*,100)
        call menu(menu_no)
        read(*,200)choice
        if ((choice.eq.'old').or.(choice.eq.'1')) then
                              call old_system(NI,NJ,NK,M(1),D(1),GN(1)
     c                                       ,GD(1),GT(1))
                              error=off
        end if
        if ((choice.eq.'new').or.(choice.eq.'2')) then
                              call new_system(NI,NJ,NK,M(1),D(1),GN(1)
     c                                       ,GD(1),GT(1))
                              error=off
        end if
        if ((choice.eq.'sav').or.(choice.eq.'3')) then
                              call save_system(NI,NJ,NK,M(1),D(1),GN(1)
     c                                       ,GD(1),GT(1))
                              error=off
```

```
          end if
          if ((choice.eq.'chg').or.(choice.eq.'4')) then
                              call edit_matrix(NI,NJ,NK,M,GN,D,GD,GT)
                               error=off
          end if
          if ((choice.eq.'end').or.(choice.eq.'5')) then
                               exit=on
                               error=off
          end if
          if (error.eq.on) then
                          write(*,*)'invalid command'
          end if
        end do
        end do
100     format(//,'(SYSTEM)',//,'enter option  =>  ',NN)
110     format(4a4)
200     format(1a3)
        return
        end


C ***************************************************************************
C
C                            SUBROUTINE choice4
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: menu, old_system, new_system, save_system, edit_matrix           *
C *                                                                         *
C * DESCRIPTION: This subroutine calls menu to print out the matrix         *
C *              I/O menu for entering a controller matrix K(s).            *
C *              It then calls one of the above subroutines depending       *
C *              on the option chosen.                                      *
C *                                                                         *
C ***************************************************************************

        subroutine choice4
        include 'common4'
        integer menu_no,error,on,off,exit
        character*4 choice,cls

        cls=char(27)//'H'//char(27)//'J'
        off=0
        on=1
        menu_no=2

        write(*,110)cls
        exit=off
        do while(exit.ne.on)
        error=on
        do while (error.eq.on)
          write(*,100)
```

```
      call menu(menu_no)
      read(*,200)choice
      if ((choice.eq.'old').or.(choice.eq.'1')) then
                          call old_system(CNI,CNJ,CNK,CM,CD,CGN,CGD
     c                                          ,CGT)
                          error=off
      end if
      if ((choice.eq.'new').or.(choice.eq.'2')) then
                          call new_system(CNI,CNJ,CNK,CM,CD,CGN,CGD
     c                                          ,CGT)
                          error=off
      end if
      if ((choice.eq.'sav').or.(choice.eq.'3')) then
                          call save_system(CNI,CNJ,CNK,CM,CD,CGN
     c                                           ,CGD,CGT)
                          error=off
      end if
      if ((choice.eq.'chg').or.(choice.eq.'4')) then
                          call edit_matrix(CNI,CNJ,CNK,CM,CGN
     c                                            ,CD,CGD,CGT)
                          error=off
      end if
      if ((choice.eq.'end').or.(choice.eq.'5')) then
                          exit=on
                          error=off
      end if
      if (error.eq.on) then
                       write(*,*)'invalid command'
      end if
    end do
    end do
100 format(//,'(CONTROLLER)',//,'enter option  =>  ',NN)
110 format(4a4)
200 format(1a3)
    return
    end
```

```
C ************************************************************************
C
C                          SUBROUTINE DEAD_read
C
C ************************************************************************
C
C ************************************************************************
C *                                                                      *
C *  CALLS: string_to_number(a function)                                 *
C *                                                                      *
C *  DESCRIPTION: This subroutine displays the deadtime of the current*
C *               i,j element and waits for the user to enter a new   *
C *               value.                                                 *
C *                                                                      *
C ************************************************************************

      subroutine DEAD_read(i,j,norows,nocols,maxord,dtime)
      include 'common3'
      integer flag,error,value,repeat_,on,off,n,i,j
      character*20 ans,cls,ret,string
      on=1
      off=0
      cls=char(27)//'H'//char(27)//'J'
      ret=char(32)

      n=(i*j)+(j-1)*(norows-i)

      write(*,97)
      write(*,100)dtime(n)
      read(*,fmt=200,iostat=flag)string
      dtime(n)=string_to_number(string,dtime(n))



90    format(4a4)
97    format(////,20x,'OLD VALUE',4x,'NEW VALUE')
100   format('DEAD TIME',14x,f7.2,'        ',NN)

110   format(//, 'non numeric input, try again')
200   format(20a)

      return
      end
```

```
C ***************************************************************************
C
C                          SUBROUTINE denordread
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: string_to_number                                                 *
C *                                                                         *
C * DESCRIPTION: This subroutine displays the order of the                  *
C *              denominator of the current i,j element and reads           *
C *              in the new value from the keyboard.                        *
C *                                                                         *
C ***************************************************************************

      subroutine denordread (i,j,norows,nocols,maxord,Dord)
      include 'common3'
      integer flag,error,value,repeat_,on,off,n,i,j
      character*20 ans,cls,ret,string
      real rnum rvalue,string_to_number
      on=1
      off=0
      cls=char(27)//'H'//char(27)//'J'
      ret=char(32)

      n=(i*j)+(j-1)*(norows-i)
10    write(*,50)
      write(*,100)Dord(n)
      read(*,fmt=200,iostat=flag)string
      rnum=real(Dord(n))
      rvalue=string_to_number(string,rnum)
      value=int(rvalue)
            if (value.lt.0) then
                        error=on
                        write(*,120)
            else
                  if (value.gt.maxord) then
                                    error=on
                                    write(*,130)maxord
                  else
                        error=off
                        Dord(n)=value
                  end if
            end if

      if (error.eq.on) then
                  go to 10
      end if
```

```
50      format(/,'                    OLD VALUE          NEW VALUE')
100     format(//,'DENOMINATOR ORDER   ',i3,'                    ',NN)

110     format(//, 'non numeric input, try again')
120     format(//, 'order cannot be less than 0')
130     format(//, 'maximum order allowed = ',i3)
140     format(//, 'change entered value?(y/n)')
150     format(//, 'please enter y or n')

200     format(20a)
210     format(a1)
        return
        end

C ***************************************************************************
C
C                          SUBROUTINE den_read
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: string_to_number                                                 *
C *                                                                         *
C * DESCRIPTION: This subroutine displays each coefficient of the           *
C *              denominator of the current i,j element, in turn, in        *
C *              ascending order. After each coefficient has been           *
C *              displayed the subroutine reads the new value of the        *
C *              coefficient from the keyboard.                             *
C *                                                                         *
C ***************************************************************************


        subroutine den_read(i,j,norows,nocols,maxord,Dord,denval)
        include 'common3'


        integer flag,error,repeat_,on,off,Nmax,n,i,j,l,Nn
        real value
        character*20 ans,cls,ret,string
        on=1
        off=0
        cls=char(27)//'H'//char(27)//'J'
        ret=char(32)

        n=(i*j)+(j-1)*(norows-i)
        Nmax=norows*nocols
        write(*,90)
        continue
```

```
        do l=1,Dord(n)+1
          Nn=(n*l)+(l-l)*(Nmax-n)
          write(*,105)l-1,denval(Nn)
          read(*,fmt=200,iostat=flag)string
          denval(Nn)=string_to_number(string,denval(Nn))
        end do

90      format(///,'DENOMINATOR COEFFICIENTS   OLD VALUE     NEW VALUE')
105     format(//,'S^',i4,15x,f13.5,'              ',NN)


200     format(20a)
        return
        end

C ***********************************************************************
C
C                   SUBROUTINE edit_matrix
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: numordread, num_read, denordread, den_read, dead_time,       *
C *        errorchk                                                     *
C *                                                                     *
C * DESCRIPTION: This subroutine displays a menu which lists the        *
C *              options available for editing the current element      *
C *              of the matrix. The element row and column numbers      *
C *              are also displayed. The subroutine then reads the      *
C *              user's option from the keyboard and then calls the     *
C *              appropriate subroutine.                                *
C *                                                                     *
C ***********************************************************************

        subroutine edit_matrix(norows,nocols,maxord,Nord,numval,
     c                         Dord,denval,dtime)

        integer i,j,norows,nocols,maxord,Nord(1),Dord(1),exit,on,off,
     c          error,flgl,flg2
        real numval(1),denval(1),dtime(1)
        character*4 cls,option
        cls=char(27)//'H'//char(27)//'J'

        on=1
        off=0
        i=1
        j=1
        exit=off
        do while(exit.eq.off)
          write(*,10)cls
          write(*,20)i,j
          write(*,30)
          write(*,40)
```

```
      write(*,50)
      write(*,60)
      write(*,70)
      write(*,80)
      write(*,90)
      read(*,200)option
      write(*,10)cls
      if ((option.eq.'n').or.(option.eq.'1')) then
                              write(*,110)i,j
                              call numordread(i,j,norows,nocols,
     c                                        maxord,Nord)
                              call num_read(i,j,norows,nocols,
     c                                      maxord,Nord,numval)
      end if
      if ((option.eq.'d').or.(option.eq.'2')) then
                              call denordread(i,j,norows,nocols,
     c                                        maxord,Dord)
                              call den_read(i,j,norows,nocols,
     c                                      maxord,Dord,denval)
      end if
      if ((option.eq.'t').or.(option.eq.'3')) then
                              call dead_read(i,j,norows,nocols,
     c                                       maxord,dtime)
      end if
      if ((option.eq.'e').or.(option.eq.'4')) then
                               i=i+1
                               if(i.gt.norows)then
                                             i=1
                                             j=j+1
                               end if
                               if (j.gt.nocols)then
                                              i=1
                                              j=1
                               end if
      end if
      if ((option.eq.'s').or.(option.eq.'5')) then
                            write(*,10)cls
                            error=on
                            do while(error.eq.on)
                                error=off
                                write(*,100)
                                read(*,fmt=210,iostat=flg1)i
                                read(*,fmt=210,iostat=flg2)j
                                if((flg1.gt.0).or.(flg2.gt.0))then
                                    error=on
                                    errtype=10
                                    call errorchk(errtype)
                                end if
                                if((i.gt.norows).or.(i.le.0))then
                                    error=on
                                    errtype=30
                                    call errorchk(errtype)
                                end if
```

```
                                    if((j.gt.nocols).or.(j.le.0))then
                                        error=on
                                        errtype=30
                                        call errorchk(errtype)
                                    end if
                                end do
          end if
          if ((option.eq.'q').or.(option.eq.'6')) then
                                    exit=on
          end if
        end do
10      format(4a4)
20      format(//,'(EDIT ELEMENT)(',i3,',',i3,')')
30      format(//,'(1)Edit Numerator.........(n)')
40      format(//,'(2)Edit Denominator.......(d)')
50      format(//,'(3)Edit Dead Time.........(t)')
60      format(//,'(4)Next Element...........(e)')
70      format(//,'(5)Select Next Element....(s)')
80      format(//,'(6)Quit Edit..............(q)')
90      format(/,'option > ',NN)
100     format(//,'Enter new i and j values (i,j) > ',NN)
110     format(//,'NUMERATOR, ELEMENT(',i2,',',i2,')')
200     format(1a1)
210     format(i3,i3)
        return
        end

C   ***********************************************************************
C
C                        SUBROUTINE errorchk
C
C   ***********************************************************************
C
C   ***********************************************************************
C   *                                                                     *
C   * CALLS:                                                              *
C   *                                                                     *
C   * DESCRIPTION: This subroutine displays one of several error          *
C   *              messages depending on the value of a parameter         *
C   *              that is passed from the calling subroutine.            *
C   *                                                                     *
C   ***********************************************************************

        subroutine errorchk(errtype)
        integer errtype

        if (errtype.eq.10) then
                        write(*,10)
10                      format(/,'***ERROR*** 10 entry incorrect type')
        end if
        if (errtype.eq.30) then
                        write(*,30)
30                      format(/,'***ERROR*** 30 entry out of range')
        end if
```

```
        if (errtype.eq.40) then
                        write(*,40)
40                      format(/,'***ERROR*** 40 incorrect entry must
     c                                  answer y/n')
        end if
        if (errtype.eq.50) then
                        write(*,50)
50                      format(/,'***ERROR*** 50 file does not exist')
        end if
        if (errtype.eq.60) then
                        write(*,60)
60                      format(/,'***ERROR*** 60 invalid command')
        end if
        if (errtype.eq.70) then
                        write(*,70)
70                      format(/,'***ERROR*** 70 file already exists')
        end if
        if (errtype.eq.80)then
                         write(*,80)
80                       format(/,'***ERROR*** 80 Inversion failed')
        end if
        return
        end

C ***********************************************************************
C
C                        SUBROUTINE mat_param
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: read_NI, read_NJ, read_NK                                    *
C *                                                                     *
C * DESCRIPTION: This subroutine calls three other subroutines that     *
C *              read in the number of rows, NI, the number of          *
C *              columns, NJ, and the maximum order of any              *
C *              denominator or numerator.                              *
C *                                                                     *
C ***********************************************************************

        subroutine mat_param(norows,nocols,maxord)

        integer on,off,exit
        character ans
        on=1
        off=0

        write(*,20)
        exit=off
        do while(exit.eq.off)
          call read_NI(norows)
          call read_NJ(nocols)
          call read_NK(maxord)
```

```
          write(*,10)
          read(*,100)ans
          if (ans.ne.'y') then
                          exit=on
          end if
        end do

10      format(//,'Change values (y/n) > ',NN)
20      format(//,'(MATRIX PARAMETERS)')
100     format(1a1)

        return
        end

C  ************************************************************************
C
C                              SUBROUTINE menu
C
C  ************************************************************************
C
C  ************************************************************************
C  *                                                                      *
C  * CALLS:                                                               *
C  *                                                                      *
C  * DESCRIPTION: This subroutine displays a menu on the screen. The      *
C  *              subroutine displays one of four menus depending on      *
C  *              the value of a parameter, menu_no, passed from the      *
C  *              calling subroutine.                                     *
C  *                                                                      *
C  ************************************************************************
        subroutine menu(menu_no)
        integer*4 menu_no,choice,on,off,error,flag,value
        character*4 cls,ret

        on=1
        off=0

        cls=char(27)//'H'//char(27)//'J'
        ret=char(32)

C
C  */ First simulator menu /*
C
        if (menu_no.eq.1) then
                          write(*,120)
                          write(*,130)
                          write(*,140)
                          write(*,150)
                          write(*,500)
        end if
C
C  */ System entry,retrieval ect. /*
C
```

```
        if (menu_no.eq.2) then
                          write(*,5)cls
                          write(*,4)
                          write(*,10)
                          write(*,20)
                          write(*,30)
                          write(*,40)
                          write(*,48)
        end if
C
C  */ second simulator menu /*
C
        if (menu_no.eq.3) then
                          write(*,5)cls
                          write(*,4)
                          write(*,300)
                          write(*,310)
                          write(*,320)
        end if
C
C  */ simulator menu to alter the no. of var. ect. /*
C
        if (menu_no.eq.5) then
                          write(*,5)cls
                          write(*,4)
                          write(*,200)
                          write(*,210)
                          write(*,220)
        end if
4       format(/,'Enter number of option or name in brackets.')
5       format(4a4)
6       format(/,15x,'FUNCTION',12x,'MAIN')
10      format(//,10x,'(1) Retrieve an existing system(old)')
20      format(//,10x,'(2) Enter a new system matrix..(new)')
30      format(//,10x,'(3) Save the current matrix....(sav)')
40      format(//,10x,'(4) Edit the current matrix....(chg)')
48      format(//,10x,'(5) Exit back to main menu.....(end)')
120     format(//,10x,'(1) Loading/Editing the system..(sys)')
130     format(//,10x,'(2) Loading/Editing controller..(con)')
140     format(//,10x,'(3) Simulator.................(sim)')
150     format(//,10x,'(4) exit......................(end)')
200     format(//,10x,'(1) change display variables...(cvar)')
210     format(//,10x,'(2) change max min axes values.(max)')
220     format(//,10x,'(3) return to previous menu....(exit)')
300     format(//,10x,'(1) New simulation.............(go)')
310     format(//,10x,'(2) Redisplay last simulation..(disp)')
320     format(//,10x,'(3) Return to main menu........(end)')
500     format(//,'Enter option name or number and RETURN')
        return
        end
```

```
C ***********************************************************************
C
C                          SUBROUTINE new_system
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: mat_param, edit_matrix                                       *
C *                                                                     *
C * DESCRIPTION: This subroutine clears the screen and the calls        *
C *              mat_param which allows the user to define the          *
C *              dimensions of the new matrix to be entered. The        *
C *              subroutine then sets all of the denominators to        *
C *              one. The subroutine then calls edit_matrix.            *
C *                                                                     *
C ***********************************************************************

      subroutine new_system(norows,nocols,maxord,Nord,Dord,numval
     c                      ,denval,dtime)
      integer i,j,l,norows,nocols,maxord,Nord(1),Dord(1)
      real numval(1),denval(1),dtime(1)
      character*4 cls

      cls=char(27)//'H'//char(27)//'J'
      write(*,10)cls
      do i=1,2700
       numval(i)=0.0
       denval(i)=0.0
      end do
      do i=1,900
        Nord(i)=0
        Dord(i)=0
        dtime(i)=0.0
      end do
      call mat_param(norows,nocols,maxord)
C */ Set denominators to 1 to avoide divide by zero/*
      do i=1,norows*nocols
        denval(i)=1.0
      end do
      call edit_matrix(norows,nocols,maxord,Nord,numval,Dord,denval,
     c                 dtime)
10    format(4a4)
      return
      end
```

```
C ***********************************************************************
C
C                          SUBROUTINE numordread
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C *  CALLS: string_to_number                                            *
C *                                                                     *
C * DESCRIPTION: This subroutine displays the order of the current      *
C *              i,j element and reads in the new value from the        *
C *              keyboard.                                              *
C *                                                                     *
C ***********************************************************************


      subroutine numordread (i,j,norows,nocols,maxord,Nord)
      include 'common3'
      integer flag,error,value,repeat_,on,off,n,i,j
      character*20 ans,cls,ret,string
      real rnum rvalue,string_to_number
      on=1
      off=0
      cls=char(27)//'H'//char(27)//'J'
      ret=char(32)

      n=(i*j)+(j-1)*(norows-i)
10    write(*,50)
      write(*,100)Nord(n)
      read(*,fmt=200,iostat=flag)string
      rnum=real(Nord(n))
      rvalue=string_to_number(string,rnum)
      value=int(rvalue)
            if (value.lt.0) then
                        error=on
                        write(*,120)
            else
                  if (value.gt.maxord) then
                                    error=on
                                    write(*,130)maxord
                  else
                        error=off
                        Nord(n)=value
                  end if
            end if

      if (error.eq.on) then
                  go to 10
      end if
```

```
50      format(/,'                OLD VALUE          NEW VALUE')
100     format(//,'NUMERATOR ORDER   ',i3,'                    ',NN)

110     format(//, 'non numeric input, try again')
120     format(//, 'order cannot be less than 0')
130     format(//, 'maximum order allowed = ',i3)
140     format(//, 'change entered value?(y/n)')
150     format(//, 'please enter y or n')

200     format(20a)
210     format(a1)
        return
        end

C ***************************************************************************
C
C                       SUBROUTINE num_read
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: string_to_number                                                 *
C *                                                                         *
C * DESCRIPTION: This subroutine displays each coefficient of the           *
C *              numerator of the current i,j element, in turn, in          *
C *              acending order. After each coefficient has been            *
C *              displayed the subroutine reads the new value of the        *
C *              coefficient from the keyboard. If the user enters a        *
C *              non_numeric input or just presses return then the          *
C *              old value is retained.                                     *
C *                                                                         *
C ***************************************************************************


        subroutine num_read(i,j,norows,nocols,maxord,Nord,numval)
        include 'common3'


        integer flag,error,repeat_,on,off,Nmax,n,i,j,l,Nn
        real value
        character*20 ans,cls,ret,string
        on=1
        off=0
        cls=char(27)//'H'//char(27)//'J'
        ret=char(32)

        n=(i*j)+(j-1)*(norows-i)
        Nmax=norows*nocols
        write(*,90)
        continue
        do l=1,Nord(n)+1
          Nn=(n*l)+(l-1)*(Nmax-n)
          write(*,105)l-1,numval(Nn)
```

```
          read(*,fmt=200,iostat=flag)string
          numval(Nn)=string_to_number(string,numval(Nn))
        end do

90      format(///,'NUMERATOR COEFFICIENTS    OLD VALUE     NEW VALUE')
105     format(//,'S^',i4,15x,f13.5,'             ',NN)


200     format(20a)
        return
        end

C  ***********************************************************************
C
C                           SUBROUTINE old_system
C
C  ***********************************************************************
C
C  ***********************************************************************
C  *                                                                     *
C  * CALLS: errorchk                                                     *
C  *                                                                     *
C  * DESCRIPTION: This subroutine reads the name of a file from the      *
C  *              keyboard. It attempts to access this file. If it       *
C  *              succeeds then the subroutine transfers the contents    *
C  *              of the file into variables that define the             *
C  *              dimensions of the matrix and the arrays that store     *
C  *              the orders and values of the numerator and             *
C  *              denominator polynomials. If the file does not exist    *
C  *              an error is generated and the user is required to      *
C  *              enter a valid file name.                               *
C  *                                                                     *
C  ***********************************************************************
        subroutine old_system(norows,nocols,maxord,Nord,Dord,numval
     c                        ,denval,dtime)
        include 'common3'
        character*10 name,cls
        integer*4 flag
        integer 1,on,off
        cls=char(27)//'H'//char(27)//'J'
        on=1
        off=0

        write(*,100)cls
        error=on
        do while (error.ne.off)
          write(*,110)
          read(*,fmt=200,iostat=flag)name
          open(9,file=name,status='OLD',access='sequential',iostat=
     c           flag)
```

```
          if (flag.ne.0) then
                        errtype=50
                        call errorchk(errtype)
          else
                error=off
          end if
        end do
        read(9,*)norows,nocols,maxord
        read(9,*)(Nord(l),l=1,norows*nocols)
        read(9,*)(Dord(l),l=1,norows*nocols)
        read(9,*)(dtime(l),l=1,norows*nocols)
        read(9,*)(numval(l),l=1,norows*nocols*(maxord+1))
        read(9,*)(denval(l),l=1,norows*nocols*(maxord+1))
        close(9,status='keep')
100     format(4a4)
110     format(//,'enter file name:    ',NN)
200     format(1a10)
        end

C ***************************************************************************
C
C                    SUBROUTINE read_NI
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: errorchk                                                         *
C *                                                                         *
C * DESCRIPTION: This subroutine reads the number of rows of the            *
C *              matrix from the keyboard.                                  *
C *                                                                         *
C ***************************************************************************

        subroutine READ_NI(norows)
        include 'common3'
        integer flag,error,value,repeat_,on,off,NImax,errtype


        character*4 ans,cls,ret
        NImax=30
        on=1
        off=0
        cls=char(27)//'H'//char(27)//'J'
        ret=char(32)

C       write(*,90)cls
10      write(*,100)
        read(*,fmt=200,iostat=flag)value
```

```
      if (flag.gt.0) then
                     error=on
                     errtype=10
                     call errorchk(errtype)
      else
         .    if (value.le.0) then
                           error=on
                           write(*,120)
             else
                    if (value.gt.NImax) then
                                        error=on
                                        write(*,130)NImax
                    else
                           error=off
                    end if
             end if
      end if

      if (error.eq.on) then
                      go to 10
      end if


      if (repeat_.eq.on) then
                         go to 10
      end if

      norows=value
C90   format(4a4)
100   format(//, 'enter the number of rows > ',NN)

120   format(//, 'number of rows must be >0')
130   format(//, 'maximum no. of rows = ',i3)
140   format(//, 'change entered value?(y/n)')
150   format(//, 'please enter y or n')
200   format(i3)
210   format(a1)

      return
      end
```

```
C ***********************************************************************
C
C                       SUBROUTINE read_NJ
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: errorchk                                                     *
C *                                                                     *
C * DESCRIPTION: This subroutine reads in the number of columns of      *
C *              the matrix from the keyboard.                          *
C *                                                                     *
C ***********************************************************************

        subroutine  READ_NJ(nocols)
        include 'common3'
        integer flag,error,value,repeat_,on,off,NJmax,errtype
        character*4 ans,cls,ret
        NJmax=30
        on=1
        off=0
        cls=char(27)//'H'//char(27)//'J'
        ret=char(32)

C       write(*,90)cls
10      write(*,100)
        read(*,fmt=200,iostat=flag)value

        if (flag.gt.0) then
                       error=on
                       errtype=10
                       call errorchk(errtype)
        else
               if (value.le.0) then
                               error=on
                               write(*,120)
               else
                      if (value.gt.NJmax) then
                                          error=on
                                          write(*,130)NJmax
                      else
                             error=off
                      end if
               end if
        end if

        if (error.eq.on) then
                         go to 10
        end if
```

```
        if (repeat_.eq.on) then
                          go to 10
        end if

        nocols=value

C90     format(4a4)
100     format(//, 'enter the number of columns > ',NN)

110     format(//, 'non numeric input, try again')
120     format(//, 'number of columns must be >0')
130     format(//, 'maximum no. of columns = ',i3)
140     format(//, 'change entered value?(y/n)')
150     format(//, 'please enter y or n')

200     format(i3)
210     format(a1)

        return
        end

C ***************************************************************************
C
C                        SUBROUTINE read_NK
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: errorchk                                                         *
C *                                                                         *
C * DESCRIPTION: This subroutine reads the maximum order of any             *
C *              denominator or numerator polynomial from the               *
C *              keyboard.                                                  *
C *                                                                         *
C ***************************************************************************

        subroutine READ_NK(maxord)
        include 'common3'

        integer flag,error,value,repeat_,on,off,NKmax
        character*4 ans,cls,ret
        NKmax=3
        on=1
        off=0
        cls=char(27)//'H'//char(27)//'J'
        ret=char(32)

C       write(*,90)cls
10      write(*,100)
        read(*,fmt=200,iostat=flag)value
```

```
      if (flag.gt.0) then
                    error=on
                    call errorchk(10)
      else
            if (value.lt.0) then
                          error=on
                          write(*,120)
            else
                  if (value.gt.NKmax) then
                                      error=on
                                      write(*,130)NKmax
                  else
                        error=off
                        maxord=value
                  end if
            end if
      end if

      if (error.eq.on) then
                     go to 10
      end if

      if (repeat_.eq.on) then
                      go to 10
      end if

C90   format(4a4)
100   format(//, 'enter the maximum order of any element > ',NN)

110   format(//, 'non numeric input, try again')
120   format(//, 'order cannot be less than 0')
130   format(//, 'maximum order allowed = ',i3)
140   format(//, 'change entered value?(y/n)')
150   format(//, 'please enter y or n')

200   format(i3)
210   format(a1)
      return
      end
```

```
C ***********************************************************************
C
C                          SUBROUTINE save_system
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: errorchk                                                     *
C *                                                                     *
C * DESCRIPTION: This subroutine reads the name of a file and adds      *
C *              the prefix gs. to the file name. It then opens a       *
C *              file of this name and saves the dimensions of the      *
C *              current matrix and the matrix elements in the file.    *
C *              If the file already exists an error is generated.      *
C *                                                                     *
C *                                                                     *
C ***********************************************************************


      subroutine save_system(norows,nocols,maxord,Nord,Dord,numval
     c                       ,denval,dtime)
      include 'common3'
      character*10 name,cls
      integer flag,l,on,off,errtype,error
      cls=char(27)//'H'//char(27)//'J'
      on=1
      off=0
      error=on
      do while (error.ne.off)
      write(*,110)
      read(*,fmt=200,iostat=flag)name
      name='gs.'//name
      open(9,file=name,status='new',access='sequential',iostat=flag)
      if (flag.ne.0) then
                  errtype=70
                  call errorchk(errtype)
      else
          error=off
      end if
      end do
      write(9,*)norows,nocols,maxord
      write(9,*)(Nord(l),l=1,norows*nocols)
      write(9,*)(Dord(l),l=1,norows*nocols)
      write(9,*)(dtime(l),l=1,norows*nocols)
      write(9,*)(numval(l),l=1,norows*nocols*(maxord+1))
      write(9,*)(denval(l),l=1,norows*nocols*(maxord+1))
      close(9,status='keep')
100   format(4a4)
110   format(//,'enter file name:   ',4X,NN)
200   format(1a5)
      return
      end
```

```
C ************************************************************************
C
C                        FUNCTION strtonum
C
C ************************************************************************
C
C ************************************************************************
C *                                                                      *
C * CALLS:                                                               *
C *                                                                      *
C * DESCRIPTION: This function is passed a string of characters and      *
C *              a real number. It then converts the string into a       *
C *              real number if this is possible and returns this        *
C *              value. If the string does not correspond to a real      *
C *              number then the real number that was passed to the      *
C *              function is returned.                                   *
C *                                                                      *
C ************************************************************************


      function string_to_number(string,oldnum)
      real string_to_number,oldnum,value
      integer flag
      character*20 string,blank
      blank='                    '
      if (string.eq.blank) then
                       string_to_number=oldnum
      else
         read(string,'(f13.0)',iostat=flag)value
         if (flag.le.0) then
                       string_to_number=value
         else
             call errorchk(10)
             string_to_number=oldnum
         end if
      end if
      return
      end
```

```
C   ***********************************************************************
C
C                          SUBROUTINE  chg_maxmin
C-
C   ***********************************************************************
C
C   ***********************************************************************
C   *                                                                     *
C   * CALLS: errorchk                                                     *
C   *                                                                     *
C   * DESCRIPTION: This subroutine enables the user to enter the max      *
C   *              and min values for the variables to be plotted i.e.*
C   *              ymax,ymin,umax,umin,tmax,tmin for each of the          *
C   *              variables to be plotted. These values are stored in*
C   *              ysim() and usim().                                     *
C   *                                                                     *
C   ***********************************************************************

      subroutine chg_maxmin
      include 'common1'
      include 'common2'
      integer on,off,exit,error,flag,ie,n1,n2,n3,number,errtype
      character*4 cls,charc(3),ans,var
      on=1
      off=0
      cls=char(27)//'H'//char(27)//'J'

      write(*,10)cls
      exit=off
      do while (exit.eq.off)
        error=on
        do while (error.eq.on)
          write(*,20)
          read(*,fmt=100,iostat=flag)var
          if(flag.ne.0) then
                        errtype=10
                        call errorchk(errtype)
          else
              do ie=1,3
                charc(ie)=var(ie:ie)
              end do
              if (charc(3).eq.' ') then
                                   number=ichar(charc(2))-48
              else
                  number=ichar(charc(2))*10+ichar(charc(3))-528
              end if
              if((number.lt.1).or.(number.gt.NI)) then
                                              errtype=30
                                            call errorchk(errtype)
              else
```

```
                error=off
            end if
          end if
        end do
        n1=(number*1)
        n2=(number*2)+(NI-number)
        n3=(number*3)+2*(NI-number)
        n4=(number*4)+3*(NI-number)
        if(charc(1).eq.'y') then
                            write(*,30)'ymax',ysim(n1)
                            read(*,110)ysim(n1)
                            write(*,30)'ymin',ysim(n2)
                            read(*,110)ysim(n2)
                            write(*,30)'tmax',ysim(n3)
                            read(*,110)ysim(n3)
                            write(*,30)'tmin',ysim(n4)
                            read(*,110)ysim(n4)
        end if
        if(charc(1).eq.'u') then
                            write(*,30)'umax',usim(n1)
                            read(*,110)usim(n1)
                            write(*,30)'umin',usim(n2)
                            read(*,110)usim(n2)
                            write(*,30)'tmax',usim(n3)
                            read(*,110)usim(n3)
                            write(*,30)'tmin',usim(n4)
                            read(*,110)usim(n4)
        end if
        write(*,70)
        read(*,130)ans
        if(ans.eq.'y') then
                       exit=on
        end if
      end do
10    format(4a4)
20    format(//,'enter variable name, eg y1 or u1, => ',NN)
30    format(//,'present value',a4,'=',f4.0,/,'new value=',NN)
70    format(//,'do you wish to end these changes?(y/n) => ',NN)
100   format(a3)
110   format(f4.0)
130   format(a1)
      return
      end
```

```
C   ************************************************************************
C
C                          SUBROUTINE chgsim_deflt
C
C   ************************************************************************

C   ************************************************************************
C   *                                                                      *
C   * CALLS: errorchk, simdisp_init                                        *
C   *                                                                      *
C   * DESCRIPTION: This subroutine allows the user to change the          *
C   *              simulation parameters. It displays a menu and the       *
C   *              user selectswhich parameter he wishes to change.        *
C   *              The old value of the variable is then displayed         *
C   *              and the user enters the new value from the keyboard*
C   *                                                                      *
C   ************************************************************************

        subroutine chgsim_deflt
        include 'common2'
        include 'common1'
        integer exit,end,flag,errtype,on,off,1,finish
        real temp
        character*4 parameter
        on=1
        off=0
        end=1
        exit=0
       do while (exit.ne.end)
        error=on
        do while(error.eq.on)
C
C /***** Write out parameter list ***************************************/
C
          write(*,10)
          write(*,20)
          write(*,30)
          write(*,40)
          write(*,50)
          write(*,60)
          write(*,70)
C       /* select parameter to change */
          write(*,110)
          read(*,fmt=200,iostat=flag)parameter
          if (flag.ne.0) then
                        errtype=10
                        call errorchk(errtype)
          end if
```

```
C
C /**** Read new display values ***********************************************
C
         if ((parameter.eq.'disp').or.(parameter.eq.'5')) then
                              call simdisp_init
                              error=off
         end if
C
C /**** Read new setpoint values **************************************
C
         if ((parameter.eq.'setp').or.(parameter.eq.'4')) then
                           do l=1,NI
                              write(*,145)l,setpoint(l)
                             read(*,fmt=210,iostat=flag)setpoint(l)
                             write(*,146)
                             read(*,fmt=210,iostat=flag)temp
                             settime(l)=temp
                              if (flag.ne.0) then
                                           errtype=10
                                           call errorchk(errtype)
                              else
                                  error=off
                              end if
                            end do
         end if
C
C
C /**** Read new loop values **************************************************
C
         if ((parameter.eq.'loop').or.(parameter.eq.'1')) then
                            write(*,100)
                           do l=1,NI
                              write(*,150)l,loop(l)
                              read(*,fmt=220,iostat=flag)loop(l)
                              if (flag.ne.0) then
                                           errtype=10
                                           call errorchk(errtype)
                              else
                                  error=off
                              end if
                            end do
         end if
C
C /**** Read new dt value *****************************************************
C
         if ((parameter.eq.'dt  ').or.(parameter.eq.'2')) then
                           write(*,160)dt
                           read(*,fmt=210,iostat=flag)dt
```

```
                              if (flag.ne.0) then
                                             errtype=10
                                             call errorchk(errtype)
                              else
                                   error=off
                              end if
          end if
C
C /**** Read new t1 value ***********************************************
C
          if ((parameter.eq.'t1  ').or.(parameter.eq.'3')) then
                              write(*,180)t1
                              read(*,fmt=210,iostat=flag)t1
                              if (flag.ne.0) then
                                             errtype=10
                                             call errorchk(errtype)
                              else
                                   error=off
                                   do 1=1,NI
                                     n=(1*3)+2*(NI-1)
                                     ysim(n)=t1
                                     usim(n)=t1
                                    end do
                              end if
          end if
C
C /**** start simulation*************************************************
C
          if ((parameter.eq.'star').or.(parameter.eq.'6')) then
                              exit=end
                              error=off
          end if
C
C /**** return to menu***************************************************
C
          if ((parameter.eq.'end').or.(parameter.eq.'7')) then
                              exit=end
                              error=off
                              finish=on
          end if
          if (error.eq.on) then
                            write(*,195)
          end if
        end do
       end do
```

```
10      format(/,'(1) Open/Closed loop.............(loop)')
20      format(/,'(2) Change time step, dt..... ..(dt)')
30      format(/,'(3) Change duration of sim. ....(t1)')
40      format(/,'(4) Change setpoints.............(setp)')
50      format(/,'(5) Change display parameters...(displ)')
60      format(/,'(6) Start simulation.............(start)')
70      format(/,'(7) End simulation...............(end)')
100     format(/,'enter 1 for closed loop, and 0 for oprn loop')
110     format(//,'enter parameter that you wish to change => ',NN)
120     format(//,'present value y(',i4,')=',f7.0,/,'new value=> ',NN)
130     format(//,'present value x(',i4,')=',f7.0,/,'new value=> ',NN)
140     format(//,'present value nlag(',i4,')=',f7.0,/,'new value=> ',NN
     c        )
145     format(//,'present value step change(',i4,')=',f7.0,/,'new value
     c        => ',NN)
146     format(//,'enter time at which step is to occur=> ',NN)
150     format(//,'present value loop(',i4,')=',f7.0,/,'new value=> ',NN
     c        )
160     format(//,'present value of dt=',f3.4,/,'new value=> ',NN)
170     format(//,'present value of t0=',f7.0,/,'new value=> ',NN)
180     format(//,'present value of t1=',f7.0,/,'new value=> ',NN)
190     format(//,'present display=',a4,/,'new display=',NN)
195     format(//,'***ERROR***non existant option try again')
200     format(a4)
210     format(f7.0)
220     format(i4)

        return
        end
```

```
C  ************************************************************************
C
C                          SUBROUTINE chg_simvar
C
C  ************************************************************************

C  ************************************************************************
C  *                                                                      *
C  *  CALLS: errorchk                                                     *
C  *                                                                      *
C  *  DESCRIPTION: This routine enables the user to set up the            *
C  *               variables to be displayed on the screen.The            *
C  *               user supplies the number of variables and              *
C  *               inputs the names of the variables e.g. 'y1'.           *
C  *               The number of the y or u variable is then              *
C  *               stored in one of two arrays whose elements             *
C  *               are subscripted according to the position              *
C  *               that the plot of the variable will occupy              *
C  *               on the screen.                                         *
C  *                                                                      *
C  *                                                                      *
C  ************************************************************************


        subroutine chg_simvar
        include'common1'
        include'common2'
        integer on,off,errtype,i,flag,number,n
        character*4 var,charc(3)
        on=1
        off=0
C
C /*****Read in the number of variables to be displayed, 'ndisplay'****
C
        do i=1,8
           udisp(i)=0
           ydisp(i)=0
        end do
        error=on
        do while (error.ne.off)
          write(*,100)
          read(*,fmt=200,iostat=flag)ndisplay
          if (flag.ne.0) then
                        errtype=10
                        call errorchk(errtype)
          else
              if ((ndisplay.lt.1).or.(ndisplay.gt.8)) then
                                                  errtype=30
                                                  call errorchk(errtype)
              else
                   error=off
              end if
          end if
        end do
```

```
C
C /****Read in the names of the displayed variables ********************
C
        do i=1,ndisplay
          error=on
          do while(error.eq.on)
            write(*,110)i
            read(*,fmt=210,iostat=flag)var
            if (flag.ne.0) then
                          errtype=10
                          call errorchk(errtype)
            end if
C
C /****Check that the variable names entered are of the form yn or un
C      where n is an integer between 0 and NI. If the entry is valid
C      then the number n is obtained and placed in either ydisp or udisp
C      . These arrays are ordered according to the windows that will
C      appear on the screen. ******************************************
C
            do ie=1,3
              charc(ie)=var(ie:ie)
            end do
            write(*,700)charc(1),charc(2),charc(3)
700         format(3a2)
            if (charc(3).eq.' ') then
                           number=ichar(charc(2))-48
            else
                number=ichar(charc(2))*10+ichar(charc(3))-528
            end if
            write(*,800)number
800         format(i4)
            n=(number*5)+4*(NI-number)
            if ((number.ge.1).and.(number.le.NI)) then
                       if (charc(1).eq.'y')then
                                      ydisp(i)=number
                                      ysim(n)=i
                                      error=off
                       else if (charc(1).eq.'u') then
                                           udisp(i)=number
                                           usim(n)=i
                                           error=off
                       end if
             else
                 error=on
                 errtype=40
                 call errorchk(errtype)
             end if
             if (error.eq.on) then
                           errtype=20
                            call errorchk(errtype)
             end if
           end do
        end do
```

```
100     format(//,'enter the number of variables to be displayed =>',NN)
110     format(//,'enter variable to be displayed in window no.',
     c  i4,'(e.g. y1/ul)=>',NN)
200     format(i4)
210     format(a3)
        return
        end

C  ***********************************************************************
C
C                       SUBROUTINE control_setup
C
C  ***********************************************************************

C  ***********************************************************************
C  *                                                                     *
C  * CALLS: errorchk                                                     *
C  *                                                                     *
C  * DESCRIPTION: This subroutine checks to see if the user  wishes to *
C  *              use a controller. If the user indicates that a       *
C  *              controller is to be used then the name of the file   *
C  *              in which the controller matrix is stored is read from*
C  *              the keyboard.                                        *
C  *                                                                     *
C  ***********************************************************************

        subroutine control_setup(num_control,contname)
        include 'common4'
        integer i,on,off,error,errtype,num_control
        integer*4 flag
        character*4 cls
        character*10 blank,name,contname(10),ans

        cls=char(27)//'H'//char(27)//'J'
        blank='          '
        on=1
        off=0
C  ***/clear the controller name array/***
        do i=1,10
          contname(i)=blank
        end do
C  ***/clear the x variables/***
        do i=1,CNI*CNJ*CNK
          x0(i)=0.0
        end do
C  ***/check if a controller is needed/***
        error=on
        do while (error.ne.off)
          write(*,100) cls
          write(*,110)
          read(*,fmt=300,iostat=flag)ans
```

```
          if (flag.ne.0) then
                        errtype=10
                        call errorchk(errtype)
                        error=on
          else
              if ((ans.eq.'y').or.(ans.eq.'n')) then
                                                error=off
              else
                  errtype=40
                  call errorchk(errtype)
              end if
          end if
        end do
C  ***/ If a controller is needed then enter the controller file names
C       /***
        if (ans.eq.'y') then
          error=on
          do while (error.eq.on)
            write(*,120)
            read(*,fmt=310,iostat=flag)num_control
            if (flag.ne.0) then
                        errtype=30
                        call errorchk(errtype)
                        error=on
            else
                error=off
            end if
          end do
          do i=1,num_control
             error=on
             do while (error.eq.on)
               write(*,130)
               read(*,320)name
C  ***/ Check that the file exists /***
               open(9,file=name,access='sequential',status='OLD',
     c              iostat=flag)
               close(9)
               if (flag.ne.0) then
                           errtype=50
                           call errorchk(errtype)
                           error=on
               else
                    error=off
               end if
             end do
             contname(i)=name
          end do
        else
            num_control=0
        end if
```

```
100     format(a4)
110     format(//,'Do you wish to use one or more controllers?(y/n) ',NN
    c          )
120     format(//,'Enter the number of controllers to be used => ',NN)
130     format(//,' Enter the name of the file storing the controller,
     c         ',/,'note that the order entered will be the order in',/
     c         ,'which they will occur in the loop. => ',NN)
300      format(a1)
310      format(i2)
320      format(a9)

        return
        end

C ***************************************************************************
C
C                       SUBROUTINE display_sim
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: sim_inital, draw_axes, vdc_extent(SB), clip_rectangle(SB)        *
C *        mapping_mode(SB), plot_graphs, polyline(SB)                      *
C *                                                                         *
C * DESCRIPTION: This subroutine displays the results of the last           *
C *              simulation performed, which have been stored in a          *
C *              file. The user is given the opportunity to change          *
C *              the current plotting parameters.                           *
C *                                                                         *
C ***************************************************************************

        subroutine display_sim

        include 'common1'
        include 'common2'
        include'/usr/include/starbase.f1.h'
        include'/usr/include/starbase.f2.h'
        integer*4 fildes
        character NULL
        parameter(NULL=char(0))
        real inital_array(20),t,yrow,newy(3000)
        integer time,nsteps,ie,l

        call sim_inital
        do l=1,1500
          newy(l)=0.
        end do
        nsteps=int(1.5+(t1-t0)/dt)
        call draw_axes
        fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
        if (fildes.eq.-1) stop
        call vdc_extent(fildes,-5.0,-5.0,0.0,30.0,45.0,0.0)
        call clip_rectangle(fildes,-5.0,30.0,-5.0,45.0)
```

```
        call mapping_mode(fildes,TRUE)

C
C  ****Plot the appropriate values on the screen. Plot_points determine
C      the position of the points relative to the vdc.*****************
C
          do i=1,ndisplay
            call plot_graphs(i,newy,nsteps)
            call polyline2d(fildes,newy,nsteps,FALSE)
          end do
        fildes=gclose(fildes)
        return
        end

C   ***********************************************************************
C
C                              SUBROUTINE draw_axes
C
C   ***********************************************************************

C   ***********************************************************************
C   *                                                                     *
C   * CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB),       *
C   *        move2d(SB), draw2d(SB), clear_view_surfaces(SB)              *
C   *                                                                     *
C   * DESCRIPTION: This subroutine calculates the position and scaling*
C   *              of the axes for the graphs to be displayed and        *
C   *              draws these axes on the screen. The subroutine also*
C   *              draws the setpoints on the graph.                     *
C   *                                                                     *
C   ***********************************************************************

        subroutine draw_axes
        include'common1'
        include'common2'
        include '/usr/include/starbase.f1.h'
        include '/usr/include/starbase.f2.h'
        integer*4 fildes
        integer no_windows,rem,ie,inc,row,col,wind_no,n1,n2,n3,n4,n5,n6
     c                    ,n7
        real originy,origint,tl,yl,tg,yt,relorigint,reloriginy,timest
     c              ,toffset,yoffset,ymax,ymin,umax,umin,tmax,tmin,setp
        character*80 number
        character NULL
        parameter(NULL = char(0))



C
C   ****Determine parameters for the axes ******************************
C
        rem=mod(ndisplay,2)
C
```

```
C   ****The number of windows is always made to be even, the exception
C       being if ndisplay is one in which case the no. of windows is
C       also set to one. ***********************************************
C
        if (ndisplay.eq.1) then
                          no_windows=1
        else
            if (rem.gt.0) then
                         no_windows=ndisplay+1
            else
                 no_windows=ndisplay
            end if
        end if
C
C   ****The lenghts of the axes and their positions are now set up******
        if (no_windows.eq.6) then
                            tl=10.0
                            yl=10.0
                            tg=5.0
                            gy=5.0
        else
            if (no_windows.eq.4) then
                                tl=10.0
                                yl=10.0
                                tg=5.0
                                gy=5.0
            else
                if (no_windows.eq.2) then
                                    tl=17.0
                                    yl=17.0
                                    tg=5.0
                                    gy=5.0
                else
                    tl=25.0
                    yl=30.0
                    tg=5.0
                    gy=5.0
                end if
            end if
        end if
C     ****The  visual display coordinates are now set up.***************
C
        fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
        if (fildes .eq. -1) stop

        call vdc_extent(fildes,-5.0,-5.0,0.0,30.0,45.0,0.0)
        call clip_rectangle(fildes,-5.0,30.0,-5.0,45.0)
        call mapping_mode(fildes,TRUE)

        call clear_view_surface(fildes)
        if (no_windows.gt.2) then
                            call move2d(fildes,12.5,0.0)
                            call draw2d(fildes,12.5,44.5)
        end if
```

```
      do i=1,ndisplay
        n1=(ydisp(i)*1)
        n2=(ydisp(i)*2)+(NI-ydisp(i))
        n3=(ydisp(i)*3)+2*(NI-ydisp(i))
        n4=(ydisp(i)*4)+3*(NI-ydisp(i))
        n5=(ydisp(i)*5)+4*(NI-ydisp(i))
        n6=(ydisp(i)*6)+5*(NI-ydisp(i))
        n7=(ydisp(i)*7)+6*(NI-ydisp(i))
        n7=(ydisp(i)*7)+6*(NI-ydisp(i))
C****Save the parameters for the plot in ysim*************************

        if (ydisp(i).ne.0) then
                           ymax=ysim(n1)
                           ymin=ysim(n2)
                           tmax=ysim(n3)
                           tmin=ysim(n4)
                           wind_no=ysim(n5)
C**** The scaling factor for the axes is now saved*********************
                           ysim(n6)=yl*(1/(ymax-ymin))
                           ysim(n7)=tl*(1/(tmax-tmin))

        else if(udisp(i).ne.0) then
                              n1=(udisp(i)*1)
                              n2=(udisp(i)*2)+(NI-udisp(i))
                              n3=(udisp(i)*3)+2*(NI-udisp(i))
                              n4=(udisp(i)*4)+3*(NI-udisp(i))
                              n5=(udisp(i)*5)+4*(NI-udisp(i))
                              n6=(udisp(i)*6)+5*(NI-udisp(i))
                              n7=(udisp(i)*7)+6*(NI-udisp(i))
                              ymax=usim(n1)
                              ymin=usim(n2)
                              tmax=usim(n3)
                              tmin=usim(n4)
                              wind_no=usim(n5)
                              usim(n6)=yl*(1/(ymax-ymin))
                              usim(n7)=tl*(1/(tmax-tmin))
        end if
C****To prevent a divide by zero *********************************
        if (ymax.eq.0.0) then
                         ymax=.001
        end if
        if (tmax.eq.0.0) then
                         tmax=0.001
        end if

        reloriginy=yl*abs(ymin)/(ymax-ymin)
        relorigint=tl*abs(tmin)/(tmax-tmin)
        if (no_windows.gt.2) then
                            if(wind_no.gt.no_windows/2) then
                              toffset=tl+tg
                              yoffset=(no_windows-wind_no)*(gy+yl)
                            else
                                toffset=0.0
```

```
                              yoffset=(no_windows*.5-wind_no)*(gy+yl)
                              end if
          else
              yoffset=(no_windows-wind_no)*(gy+yl)
              toffset=0.0
          end if
          origint=relorigint+toffset
          originy=reloriginy+yoffset
          origins(wind_no)=originy
          n=(wind_no*2)+(10-wind_no)
          origins(n)=origint
          call move2d(fildes,origint,yoffset)
          call draw2d(fildes,origint,yoffset+yl)
          call move2d(fildes,toffset,originy)
          call draw2d(fildes,toffset+tl,originy)
C */ Draw in setpoints /*
         if(ydisp(i).ne.0)then
         setp=setpoint(ydisp(i))*ysim(n6)
         timest=settime(ydisp(i))*ysim(n7)
         call move2d(fildes,origint+timest,originy)
         call draw2d(fildes,origint+timest,originy+setp)
         call move2d(fildes,origint+timest,originy+setp)
         call draw2d(fildes,toffset+tl,originy+setp)
         end if
         end do
         fildes=gclose(fildes)
C        call lable_axes
         return
         end

C   ************************************************************************
C
C                          SUBROUTINE errorchk
C
C   ************************************************************************
C
C   ************************************************************************
C   *                                                                      *
C   * CALLS:                                                               *
C   *                                                                      *
C   * DESCRIPTION: This subroutine displays one of several error           *
C   *              messages depending on the value of a parameter          *
C   *              that is passed from the calling subroutine.             *
C   *                                                                      *
C   ************************************************************************

         subroutine errorchk(errtype)
         integer errtype

         if (errtype.eq.10) then
                         write(*,10)
```

```
10                      format(/,'***ERROR*** 10 entry incorrect type')
        end if
        if (errtype.eq.30) then
                        write(*,30)
30                      format(/,'***ERROR*** 30 entry out of range')
        end if
        if (errtype.eq.40) then
                        write(*,40)
40                      format(/,'***ERROR*** 40 incorrect entry must
     c                                  answer y/n')
        end if
        if (errtype.eq.50) then
                        write(*,50)
50                      format(/,'***ERROR*** 50 file does not exist')
        end if
        if (errtype.eq.60) then
                        write(*,60)
60                      format(/,'***ERROR*** 60 invalid command')
        end if
        if (errtype.eq.70) then
                        write(*,70)
70                      format(/,'***ERROR*** 70 file already exists')
        end if
        if (errtype.eq.80)then
                         write(*,80)
80                       format(/,'***ERROR*** 80 Inversion failed')
        end if
        return
        end

C*********************************************************************
C
C                          SUBROUTINE eval_control
C
C *******************************************************************
C
C *******************************************************************
C *                                                                 *
C * CALLS: eval_CX, eval_CY                                         *
C *                                                                 *
C * DESCRIPTION: This subroutine reads each controller from file    *
C *              into RAM. The subroutine then calls subroutines    *
C *              that calculate the outputs from the controller.    *
C *                                                                 *
C *******************************************************************

        subroutine eval_control(uold,contname,num_control,dt)
        include 'common4'
        character*10 contname(10)
        real t,yrow,uold(20),unew(20)
        integer time,nsteps,ie,l,n,Nn,num_control,con_num
```

```
      do con_num=1,num_control
        open(9,file=contname(con_num),status='old')
        read(9,*)CNI,CNJ,CNK
        read(9,*)(CM(l),l=1,CNI*CNJ)
        read(9,*)(CD(l),l=1,CNI*CNJ)
        read(9,*)(CGT(l),l=1,CNI*CNJ)
        read(9,*)(CGN(l),l=1,CNI*CNJ*(CNK+1))
        read(9,*)(CGD(l),l=1,CNI*CNJ*(CNK+1))
        close(9)
      write(*,*)'tr=',CGD(1),CNI,CNJ,CNK
        do i=1,CNI
          yrow=0.0
          do j=1,CNJ
            call eval_CX(i,j,con_num,dt,uold)
            call eval_CY(i,j,con_num,y,uold)
            yrow=yrow+y
          end do
          unew(i)=yrow
        end do
        do l=1,CNI
          uold(l)=unew(l)
        end do
      end do
      return
      end

C   ***********************************************************************
C
C                              SUBROUTINE eval_CX
C
C   ***********************************************************************
C
C   ***********************************************************************
C   *                                                                     *
C   * CALLS:                                                              *
C   *                                                                     *
C   * DESCRIPTION: This subroutine calculates the coefficients of        *
C   *              the differential equations that correspond to a        *
C   *              laplacian transfer function of the controller.         *
C   *              These are then used to solve the differential          *
C   *              equations at time t, using the Runge-Kutta method.     *
C   *                                                                     *
C   ***********************************************************************

      subroutine eval_CX(i,j,con_num,dt,u)
      include'common4'
      integer n,Nn,i,j,ie,con_num,kr
      real c1(4),c2(3),dx(10),sx(10),u(30)
      data c1/1.,2.,2.,1./,c2/.5,.5,1./
      max_x=1200

      n=(i*j)+(j-1)*(CNI-i)
C  **** note that dx/dt=0 for order *****
      do ie=1,CD(n)
```

```
  Nn=(n*ie)+(ie-1)*(CNI*CNJ-n)+(con_num-1)*max_x
  sx(ie)=0.0
  x(ie)=x0(Nn)
end do
do kr=1,4
  if (CD(n).ne.0) then
                dx(CD(n))=0
                do ie=1,CD(n)
                  Nn=(n*ie)+(ie-1)*(CNI*CNJ-n)
                  dx(CD(n))=dx(CD(n))-CGD(Nn)*x(ie)
                end do
                Nn=(n*(CD(n)+1))+((CD(n)+1)-1)*(CNI*CNJ-n)
                dx(CD(n))=(dx(CD(n))+u(j))/CGD(Nn)
  end if
  if (CD(n).gt.1) then
                do ie=1,CD(n)-1
                  dx(ie)=x(ie+1)
                end do
  end if
  do ie=1,CD(n)
    Nn=(n*ie)+(ie-1)*(CNI*CNJ-n)+(con_num-1)*max_x
    sx(ie)=sx(ie)+c1(kr)*dx(ie)
    if (kr.eq.4) then
                x(ie)=x0(Nn)+sx(ie)*dt/6
    else
         x(ie)=x0(Nn)+c2(kr)*dx(ie)*dt
    end if
  end do
end do
if (CD(n).eq.0) then
              x(1)=0
end if
do ie=1,CD(n)
  Nn=(n*ie)+(ie-1)*(CNI*CNJ-n)+(con_num-1)*max_x
  x0(Nn)=x(ie)
end do
return
end
```

```
C ***************************************************************************
C
C                        SUBROUTINE eval_CY
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine uses the solutions to the                  *
C *              differential equations found by eval_CX to determine*
C *              the outputs of the controller at time t.                   *
C *                                                                         *
C ***************************************************************************

      subroutine eval_CY(i,j,con_num,y,u)
      include 'common4'
      integer n,Nn,ie,i,j,con_num
      real y,u(20)
      n=(i*j)+(j-1)*(CNI-i)
      y=0.0
      if (CM(n).eq.CD(n)) then
                        do ie=1,CM(n)
                          Nn=(n*ie)+(ie-1)*(CNI*CNJ-n)
                          y=y-CGD(Nn)*x(ie)
                        end do
                        Nn=(n*(1+CM(n)))+((CM(n)+1)-1)*(CNI*CNJ-n)
      write(*,*)'Nn=',Nn
      write(*,*)'Dn=',CGD(Nn)
                        y=CGN(Nn)*(y+u(j))/CGD(Nn)
                        do ie=1,CM(n)
                          Nn=(n*ie)+(ie-1)*(CNI*CNJ-n)
                          y=y+CGN(Nn)*x(ie)
                        end do
      else
      do ie=1,CM(n)+1
        Nn=(n*ie)+(ie-1)*(CNI*CNJ-n)
        y=y+CGN(Nn)*x(ie)
      if (i*j.eq.2) then
      write(*,*)'gn=',CGN(Nn),'x=',x(ie),'y=',y
      end if
      end do
      end if
      return
      end
```

```
C ***************************************************************************
C
C                          SUBROUTINE eval_X
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the coefficients of the         *
C *              differential equations that correspond to a                *
C *              laplacian transfer function of the system matrix.          *
C *              These are then used to solve the differential              *
C *              equations at time t, using the Runge-Kutta method.         *
C *                                                                         *
C ***************************************************************************


        subroutine eval_X(i,j)
        include 'common1'
        include 'common2'
        integer n,Nn,i,j,ie
        real c1(4),c2(3)
        data c1/1.,2.,2.,1./,c2/.5,.5,1./

        n=(i*j)+(j-1)*(NI-i)
C  **** note that dx/dt=0 for order *****
        do ie=1,D(n)
          Nn=(n*ie)+(ie-1)*(NI*NJ-n)
          sx(ie)=0.0
          x(ie)=x0(Nn)
        end do
        do kr=1,4
          if (D(n).ne.0) then
                        dx(D(n))=0
                        do ie=1,D(n)
                          Nn=(n*ie)+(ie-1)*(NI*NJ-n)
                          dx(D(n))=dx(D(n))-GD(Nn)*x(ie)
                        end do
                        Nn=(n*(D(n)+1))+((D(n)+1)-1)*(NI*NJ-n)
                        dx(D(n))=(dx(D(n))+u(j))/GD(Nn)
          end if
          if (D(n).gt.1) then
                        do ie=1,D(n)-1
                          dx(ie)=x(ie+1)
                        end do
          end if
          do ie=1,D(n)
            Nn=(n*ie)+(ie-1)*(NI*NJ-n)
            sx(ie)=sx(ie)+c1(kr)*dx(ie)
            if (kr.eq.4) then
                        x(ie)=x0(Nn)+sx(ie)*dt/6
            else
```

```
                x(ie)=x0(Nn)+c2(kr)*dx(ie)*dt
              end if
            end do
          end do
          if (D(n).eq.0) then
                        x(1)=0
          end if
          do ie=1,D(n)
            Nn=(n*ie)+(ie-1)*(NI*NJ-n)
            x0(Nn)=x(ie)
          if ((i.eq.1).and.(j.eq.1)) then
          write(*,*)'x(',ie,')=',x(ie)
          end if
          end do
          return
          end

C ***************************************************************************
C
C                       SUBROUTINE eval_Y
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: This subroutine uses the solutions to the differential          *
C *        equations  found by eval_X to determine the outputs of the*
C *        system at time t.                                                *
C *                                                                         *
C ***************************************************************************

          subroutine eval_Y(i,j)
          include 'common1'
          include 'common2'
          integer n,Nn,ie,i,j
          n=(i*j)+(j-1)*(NI-i)
          y=0.0
          if (M(n).eq.D(n)) then
                          do ie=1,M(n)
                            Nn=(n*ie)+(ie-1)*(NI*NJ-n)
                            y=y-GD(Nn)*x(ie)
                          end do
                          Nn=(n*(1+M(n)))+((M(n)+1)-1)*(NI*NJ-n)
                          y=GN(Nn)*(y+u(j))/GD(Nn)
          if (i*j.eq.2) then
          write(*,*)'gn=',GN(Nn),'gd=',GD(Nn),'u=',u(j),'y=',y
          end if

          end if
          do ie=1,M(n)+1
            Nn=(n*ie)+(ie-1)*(NI*NJ-n)
            y=y+GN(Nn)*x(ie)
          if (i*j.eq.2) then
          write(*,*)'gn=',GN(Nn),'x=',x(ie),'y=',y
```

```
        end if
        end do

        return
        end

C   ************************************************************************
C
C
C                              SUBROUTINE plot_graphs
C
C   ************************************************************************

C   ************************************************************************
C   *                                                                      *
C   * CALLS:                                                               *
C   *                                                                      *
C   * DESCRIPTION: This subroutine calculates the co-ordinates of the      *
C   *              input and output variables that are to be plotted       *
C   *              on the screen and that have been retrieved from         *
C   *              file.                                                   *
C   *                                                                      *
C   ************************************************************************

        subroutine plot_graphs(i,newy,nsteps)
        include 'common1'
        include'common2'
        integer no_windows,ie,inc,row,col,wind_no,n1,n2,n3,n4,n5
     c                    ,n6,n7,n8,na,nb,nc,nd,i,time,count,l,nsteps
        real originy,originx,point
     c              ,toffset,yoffset
     c              ,yscale,tscale,t,newy(3000)
          if (ydisp(i).ne.0) then
           open(2,file='yrow',access='direct',form='formatted',recl=16)
                           n5=(ydisp(i)*5)+4*(NI-ydisp(i))
                           n6=(ydisp(i)*6)+5*(NI-ydisp(i))
                           n7=(ydisp(i)*7)+6*(NI-ydisp(i))
                           n=(ydisp(i)*time)+(time-1)*(NI-ydisp(i))
                           yscale=ysim(n6)
                           tscale=ysim(n7)
                           wind_no=ysim(n5)
                           na=wind_no
                           nb=(wind_no*2)+(10-wind_no)
                           originy=origins(na)
                           originx=origins(nb)
                           count=1
                           do l=1,nsteps
                            n=(ydisp(i)*l)+(l-1)*(NI-ydisp(i))
                            read(2,'(f7.3)',rec=n)point
                            newy(count+1)=originy+point*yscale
                            newy(count)=originx+(t0+l-1)*dt*tscale
                            count=count+2
                           end do
                           close(2)
```

```
      else
       open(3,file='urow',access='direct',form='formatted',recl=16)
          n5=(udisp(i)*5)+4*(NI-udisp(i))
          n6=(udisp(i)*6)+5*(NI-udisp(i))
          n7=(udisp(i)*7)+6*(NI-udisp(i))
          n=(udisp(i)*time)+(time-1)*(NI-udisp(i))
          yscale=usim(n6)
          tscale=usim(n7)
          wind_no=usim(n5)
          na=wind_no
          nb=(wind_no*2)+(10-wind_no)
          originy=origins(na)
          origint=origins(nb)
         count=1
         do l=1,nsteps
           n=(udisp(i)*l)+(l-1)*(NI-udisp(i))
           read(3,'(f7.3)',rec=n)point
           newy(count+1)=originy+point*yscale
           newy(count)=origint+(t0+l-1)*dt*tscale
           count=count+2
         end do
         close(3)
       end if
      return
      end

C   ***********************************************************************
C
C                          SUBROUTINE plot_points
C
C   ***********************************************************************

C   ***********************************************************************
C   *                                                                     *
C   * CALLS:                                                              *
C   *                                                                     *
C   * DESCRIPTION: This subroutine calculates the co-ordinates of the *
C   *              input and output variables that are to be plotted  *
C   *              on the screen.                                         *
C   *                                                                     *
C   ***********************************************************************

      subroutine plot_points(time,oldy,oldt,newy,newt,t)
      include 'common1'
      include'common2'
      include '/usr/include/starbase.f1.h'
      include '/usr/include/starbase.f2.h'
      integer*4 fildes
      integer no_windows,rem,ie,inc,row,col,wind_no,n1,n2,n3,n4,n5
     c                  ,n6,n7,n8,na,nb,nc,nd,i,time,l
      real originy,origint,originy0,origint0,t1,y1,tg,yt,relorigint,reloriginy
     c             ,toffset,yoffset,ymax,ymin,umax,umin,tmax,tmin
     c             ,yscale,tscale,t,newy(20),newt(20),oldy(20),oldt(20)
      character*80 number
```

```
      character NULL
      parameter(NULL = char(0))



      do i=1,ndisplay
        if (ydisp(i).ne.0) then
                          n5=(ydisp(i)*5)+4*(NI-ydisp(i))
                          n6=(ydisp(i)*6)+5*(NI-ydisp(i))
                          n7=(ydisp(i)*7)+6*(NI-ydisp(i))
                          n=(ydisp(i)*time)+(time-1)*(NI-ydisp(i))
                          yscale=ysim(n6)
                          tscale=ysim(n7)
                          wind_no=ysim(n5)
                          na=wind_no
                          nb=(wind_no*2)+(10-wind_no)
                          originy=origins(na)
                          origint=origins(nb)
                           n=ydisp(i)
                           oldy(i)=originy+ydold(n)*yscale
                           if (time.ne.1) then
                                 oldt(i)=origint+(t-dt)*tscale
                           else
                                oldt(i)=origint+t*tscale
                           end if
                           n=ydisp(i)
                           newy(i)=originy+ydnew(n)*yscale
                           newt(i)=origint+t*tscale
        else
            n5=(udisp(i)*5)+4*(NI-udisp(i))
            n6=(udisp(i)*6)+5*(NI-udisp(i))
            n7=(udisp(i)*7)+6*(NI-udisp(i))
            n=(udisp(i)*time)+(time-1)*(NI-udisp(i))
            yscale=usim(n6)
            tscale=usim(n7)
            wind_no=usim(n5)
            na=wind_no
            nb=(wind_no*2)+(10-wind_no)
            originy=origins(na)
            origint=origins(nb)
                          n=udisp(i)
                          oldy(i)=originy+udold(n)*yscale
                          if (time.ne.1) then
                                oldt(i)=origint+(t-dt)*tscale
                          else
                               oldt(i)=origint+t*tscale
                          end if
                          n=udisp(i)
                          newy(i)=originy+udnew(n)*yscale
                          newt(i)=origint+t*tscale
C         if (time.ne.1) then
c           n=(udisp(i)*(time-1))+((time-1)-1)*(NI-udisp(i))
c           oldy(i)=originy+ud(n)*yscale
C           oldt(i)=origint+(t-dt)*tscale
```

```
c            else
c                n=(udisp(i)*time)+(time-1)*(NI-udisp(i))
c                oldy(i)=originy+ud(n)*yscale
c                oldt(i)=origint+t*tscale
c            end if
c            n=(udisp(i)*time)+(time-1)*(NI-udisp(i))
c            newy(i)=originy+ud(n)*yscale
c            newt(i)=origint+t*tscale
          end if
        end do

        do l=1,NI
          ydold(l)=ydnew(l)
          udold(l)=udnew(l)
        end do
        return
        end

C     ***********************************************************************
C
C                        SUBROUTINE simdisp_init
C
C     ***********************************************************************
C
C     ***********************************************************************
C     *                                                                     *
C     * CALLS: errorchk, menu, chg_simvar, chgmaxmin                        *
C     *                                                                     *
C     * DESCRIPTION: This subroutine enables the user to change the         *
C     *              variables that are to be plotted on the screen         *
C     *              or to change the maximum and minimum values that       *
C     *              are represented on the graph axes.                     *
C     *                                                                     *
C     ***********************************************************************

        subroutine simdisp_init
        include'common1'
        include'common2'
        integer flag,error,exit,on,off,errtype
        character*4 cls,ans
        on=1
        off=0
        cls=char(27)//'H'//char(27)//'J'
        exit=off
        write(*,90)cls
        do while (exit.eq.off)
          error=on
          do while (error.eq.on)
C
C     ****Select the simulation parameters to be changed ****************
            write(*,100)
            menu_no=5
            call menu(menu_no)
            read(*,fmt=200,iostat=flag)ans
```

```
            if (flag.ne.0) then
                        errtype=10
                        call errorchk(errtype)
            end if
            if ((ans.eq.'cvar').or.(ans.eq.'1')) then
                         call chg_simvar
                         error=off
            end if
            if ((ans.eq.'max ').or.(ans.eq.'2')) then
                         call chg_maxmin
                         error=off
            end if
            if ((ans.eq.'exit').or.(ans.eq.'3'))then
                        exit=on
                        error=off
            end if
            if (error.eq.on) then
                        errtype=30
                        call errorchk(errtype)
            end if
          end do
        end do
90      format(a4)
100     format(//,'CHANGE DISPLAY PARAMETERS',//,'enter option => ',NN)
110     format(//,'option(enter help for menu)=> ',NN)
200     format(a4)
210     format(i4)
        return
        end

C ***************************************************************************
C
C                     SUBROUTINE sim_inital
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: errorchk, chgsim_deflt                                           *
C *                                                                         *
C * DESCRIPTION: This subroutine enables the designer to change             *
C *              various parameters for the simulation.                     *
C *                                                                         *
C ***************************************************************************


        subroutine sim_inital
        include'common1'
        include 'common2'
        integer flag,l,error,on,off,errtype,n1,n2,n3,n4,n5,n6,n7,finish
        character*4 cls,ans,ans1
        on=1
        off=0
        cls=char(27)//'H'//char(27)//'J'
```

```
      error=on

      do l=1,NI*NJ
         y0(l)=0.0
      end do
      do l=1,NI*NJ*NK
        x0(l)=0.0
      end do
      do l=1,10
         x(l)=0.0
      end do

      write(*,100)cls
      do while (error.eq.on)
        write(*,130)
        read(*,fmt=200,iostat=flag)ans
        if (flag.ne.0) then
                       errtype=10
                       call errorchk(errtype)
        elseif ((ans.ne.'y').and.(ans.ne.'n')) then
                                               errtype=10
                                               call errorchk(errtype)
        else
            error=off
        end if
      end do
      if (first_sim.ne.on) then
                           ans='y'
                           first_sim=on
      end if

C     /* set up default values */

            if (ans.eq.'y') then
                            do l=1,30
                              nlag(l)=0.0
                              loop(l)=0
                              setpoint(l)=0.0
                              settime(l)=0.0
                            end do
                            t0=0.0
                            t1=10.0
                            dt=0.1
C     /***Initalize the display parameters***/
                            ndisplay=1
                            ydisp(1)=1
                            do ie=1,NI
                              n1=(ie*1)
                              n2=(ie*2)+(NI-ie)
                              n3=(ie*3)+2*(NI-ie)
                              n4=(ie*4)+3*(NI-ie)
                              n5=(ie*5)+4*(NI-ie)
                              n6=(ie*6)+5*(NI-ie)
                              n7=(ie*7)+6*(NI-ie)
```

```
                              ysim(n1)=10.0
                              ysim(n2)=0.0
                              ysim(n3)=10.0
                              ysim(n4)=0.0
                              ysim(n5)=1
                              ysim(n6)=1.0
                              ysim(n7)=1.0
                              usim(n1)=10.0
                              usim(n2)=0.0
                              usim(n3)=10.0
                              usim(n4)=0.0
                              usim(n5)=1
                              usim(n6)=1.0
                              usim(n7)=1.0
                            end do
          end if
C       /* deceide whether to change default values */
          error=on
          do while (error.eq.on)
             write(*,110)
             read(*,fmt=200,iostat=flag)ans1
             if (flag.ne.0) then
                              errtype=10
                              call errorchk(errtype)
             else
C       /* change the value of the sim parameters */
                 if (ans1.eq.'y') then
                                  call chgsim_deflt
                                  error=off
                 else
                      if (ans1.eq.'n') then
                                       error=off
                      end if
                 end if
             end if
          end do
100       format(a4)
110       format(//,'do you wish to change present parameters?(y/n)=>',NN)
130       format(//,'set up default values?(y/n)=> ',NN)
200       format(a1)
          return
          end
```

```
C ***************************************************************************
C
C                          SUBROUTINE sim_mv
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: control_setup, sim_inital, vdc_extent(SB), clip_rectangle*
C *        , mapping_mode(SB), line_type(SB), eval_X, eval_Y,       *
C *        eval_control, plot_points, move2d(SB), draw2d(SB),       *
C *        make_picture_current(SB)                                 *
C *                                                                 *
C * DESCRIPTION: This subroutine controls the flow of the actual    *
C *              simulation. It calls up subroutines that enable the*
C *              user to set up the simulation parameters and  calls*
C *              subroutines that perform the calculations. The     *
C *              subroutine also plots the calculated variables on  *
C *              the screen.                                        *
C *                                                                 *
C ***************************************************************************

        subroutine sim_Mv

C */ common1 is a common area for matrix variables./*
C */ common2 is a common area for simulator variables./*
        include 'common1'
        include 'common2'
C */ include data needed for use of starbase graphics package./*
        include'/usr/include/starbase.f1.h'
        include'/usr/include/starbase.f2.h'
        integer*4 fildes
        character NULL
        parameter(NULL=char(0))
        character*10 contname(10)
        real t,yrow,oldy(20),oldt(20),newy(20),newt(20),ystore(90000)
     c       ,uold(20),ts
        integer time,nsteps,ie,l,n,Nn,deadput(9000),deadget(9000),
     c           num_control,finish

C */select any controllers to be used./*
        call control_setup(num_control,contname)
C */initalise simulator variables and select options/*
        finish=0
        call sim_inital
        if (finish.eq.1) then
                        go to 10
        end if

        do l=1,20
C */oldy and oldt are arrays that will contain the last y values and
C   the last t values, respectively, to be plotted. newy and newt are
C   the current values of y and t to be plotted./*
          oldy(l)=0.
```

```
          oldt(l)=0.
          newy(l)=0.
          newt(l)=0.
        end do
C */nsteps is the number of time steps to be made.t1 is the maximum
C   time and t0 is the starting time for the simulation/*
        nsteps=int(1.5+(t1-t0)/dt)
        do l=1,NI
C */ ydold contains the values of the output variables at time step 1./*
C */ ydnew is an array of the current values of the output variables./*
          ydold(l)=0.0
          ydnew(l)=0.0
          ud(ie)=0.0
        end do
        do l=1,NI
C */u is an array of inputs into the system./*
          u(l)=0.0
        end do
        do l=1,NI*NJ*100
C */ystore is an array of values of the elements of the sytem matrix
C   as they vary with time./*
           ystore(l)=0.0
        end do
        do i=1,NI
          do j=1,NJ
            n=(i*j)+(j-1)*(NI-i)
C */deadput is used to determine the position in ystore in which an
C   element y(i,j,t) should be stored taking any time delays into
C   account. Deadget is used to retrieve the y(i,j,t) element at time
C   t./*
            deadput(n)=int(1.5+GT(n)/dt)
            deadget(n)=1
          end do
        end do
C
C  ***/draw the axes and set up starbase for output ***/
C
        call draw_axes
        fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
        if (fildes.eq.-1) stop
        call vdc_extent(fildes,-5.0,-5.0,0.0,30.0,45.0,0.0)
        call clip_rectangle(fildes,-5.0,30.0,-5.0,45.0)
        call mapping_mode(fildes,TRUE)
        call line_type(fildes,SOLID)
        t=t0
C
C  ***/open files to store y and u values ***/
C
        open(2,file='yrow',access='direct',recl=16,form='formatted')
        open(3,file='urow',access='direct',recl=16,form='formatted')
C
C  ***/calculate y values using x, state, values. ***/
        do time=1,nsteps
          do i=1,NI
```

```
            yrow=0.0
            do j=1,NJ
              call eval_X(i,j)
C */eval_Y gives the value of y(i,j,t)./*
              call eval_Y(i,j)
C
C ****add in any delays caused by dead times **************************
C
              n=(i*j)+(j-1)*(NI-i)
             Nn=(n*deadput(n))+(deadput(n)-1)*(NI*NJ-n)
              ystore(Nn)=y
              Nn=(n*deadget(n))+(deadget(n)-1)*(NI*NJ-n)
              yrow=yrow+ystore(Nn)
              deadput(n)=deadput(n)+1
              deadget(n)=deadget(n)+1
              if (deadput(n).gt.9000) then
                                 deadput(n)=1
              end if
              if (deadget(n).gt.9000) then
                                 deadget(n)=1
              end if
            end do
            ydnew(i)=yrow
            if (time.eq.1) then
                         ydold(i)=yrow
            end if
C  **** Display current y values and store u and uvalues in files *****
          write(*,*)'yrow= ',yrow
          n=(time*i)+(time-1)*(NI-i)
          write(2,'(f9.3)',rec=n)yrow
          end do
C  **** Calculate new u values ****
         write(*,*)'time=',time,'t=',t
          do ie=1,NI
            if (t.gt.settime(ie)) then
                       n=(time*ie)+(time-1)*(NI-ie)
                if ((loop(ie).ne.1).and.(loop(ie).ne.-1)) then
                                       u(ie)=setpoint(ie)
                       else
                            u(ie)=setpoint(ie)-ydnew(ie)*loop(ie)
                            write(*,*)'error=',u(ie)
                       end if
            else
                if ((loop(ie).ne.1).and.(loop(ie).ne.-1)) then
                                       u(ie)=0.0
                       else
                            u(ie)=0.0-ydnew(ie)*loop(ie)
                       end if

            end if
          end do
C
C  ***/ Include any controllers ***/
C
```

```
          if (num_control.gt.0) then
                                do ie=1,NI
                                   uold(ie)=u(ie)
                                end do
                     call eval_control(uold,contname,num_control,dt)
                                do ie=1,NI
                                   u(ie)=uold(ie)
                                end do
          end if
          do ie=1,NI
            n=(time*ie)+(time-1)*(NI-ie)
            udnew(ie)=u(ie)
            write(3,'(f9.3)',rec=n)u(ie)
          end do

C
C  ****Plot the appropriate values on the screen. Plot_points determine
C      the position of the points relative to the vdc.*****************
C
          call plot_points(time,oldy,oldt,newy,newt,t)
          do l=1,ndisplay
C           if (time.ne.1) then
              call line_type(fildes,SOLID)
              call move2d(fildes,oldt(l),oldy(l))
              call draw2d(fildes,newt(l),newy(l))
              call make_picture_current(fildes)
C           end if
          end do
          t=t+dt
        end do
        close(2)
        close(3)
        fildes=gclose(fildes)
10      continue
        return
        end
```

```
C  ***********************************************************************
C
C                  SUBROUTINE simulator
C
C  ***********************************************************************
C
C  ***********************************************************************
C   CALLS             : menu,sim_mv,display_sim,errorchk
C
C   DESCRIPTION       : This subroutine controls the flow of the simul-
C                       -lation program, calling up an options menu,
C                       a calculation and display program and a display
C                       program.
C
C  ***********************************************************************
       subroutine simulator

C */ common1 is a common area for system matrix variables./*
C */ common2 is a common area for simulator variables. /*

       include 'common1'
       include 'common2'
       integer menu_no,error,on,off,exit,errtype
       character*4 choice,cls

C */ cls is an escape character string that clears the screen./*
       cls=char(27)//'H'//char(27)//'J'
       off=0
       on=1
       menu_no=3

       write(*,110)cls
       exit=off
       do while(exit.ne.on)
       error=on
C */ select option./*
       do while (error.eq.on)
         write(*,100)
         call menu(menu_no)
         read(*,200)choice
C */ start simulation./*
         if ((choice.eq.'go').or.(choice.eq.'1')) then
                              call sim_mv
                              error=off
         end if
C */ display last simulation /*
         if ((choice.eq.'dis').or.(choice.eq.'2')) then
                              call display_sim
                              error=off
         end if
C */ return to main program./*
         if ((choice.eq.'end').or.(choice .eq.'3')) then
                              exit=on
                              error=off
```

```
          end if
C */ write error message if incorrect option selected./*
          if (error.eq.on) then
                          errtype=60
                          call errorchk(errtype)
          end if
        end do
        end do
100     format(//,'(SIMULATOR)',//,'enter option  =>  ',NN)
110     format(4a4)
200     format(1a3)
        return
        end

C ***************************************************************************
C
C                              PROGRAM  start
C
C ***************************************************************************

C ***************************************************************************
C *                                                                         *
C * CALLS: errorchk,menu, sim_mv, display_sim                               *
C *                                                                         *
C * DESCRIPTION: This is the main program for the simulator. It             *
C *              displays a list of options that the user may               *
C *              implement and calls the appropriate subroutine in          *
C *              response to the entry made by the user.                    *
C *                                                                         *
C ***************************************************************************

        program  start
        include 'common1'
        integer menu_no,error,on,off,exit,errtype
        character*4 choice
        character*4 cls

        cls=char(27)//'H'//char(27)//'J'
        off=0
        on=1
        menu_no=1
        NI=1
        NJ=1

C */ Clear the screen /*
        write(*,100)cls
        exit=off
        do while(exit.ne.on)
        error=on
        do while (error.eq.on)
          write(*,110)
C */display menu /*
          call menu(menu_no)
C */ select option /*
```

```
      write(*,120)
      read(*,200)choice
      if ((choice.eq.'sys').or.(choice.eq.'1')) then
                                                call choice1
                                                error=off
                                                write(*,100)cls
      end if
      if ((choice.eq.'con').or.(choice.eq.'2')) then
                     call choice4
                     error=off
      end if
      if ((choice.eq.'sim').or.(choice.eq.'3')) then
                                                call simulator
                                                error=off
                                                write(*,100)cls
      end if
      if ((choice.eq.'end').or.(choice.eq.'4')) then
                                                exit=on
                                                error=off
      end if
      if (error.eq.on) then
                     errtype=60
                     call errorchk(errtype)
      end if
    end do
    end do
100 format(4a4)
110 format(//,'(COMMAND LEVEL)')
120 format(/,'option > ',NN)
200 format(1a3)
    end
```

MATRIX REARRANGEMENT SUBROUTINES

```
C ***********************************************************************
C
C                        SUBROUTINE axes
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: graphplace                                                   *
C *                                                                     *
C * DESCRIPTION: This subroutine determines the virtual device          *
C *              coordinates for the origins of the axes for the        *
C *              polar plots. It also calculates the max and min        *
C *              axes values in virtual device coordinates.             *
C *                                                                     *
C ***********************************************************************


        subroutine axes(scale,originx,originy)
        include'common1'
        include '/usr/include/starbase.f1.h'
        include '/usr/include/starbase.f2.h'
        integer*4 fildes
        character NULL
        real axislen,originy(1000),originx(1000),
     +    scale,xmax,xmin,ymax,ymin,blocksize,indent,axes_len,blockno
        integer i,j,n

        parameter(NULL=char(0))

C/* The screen is set up to be 100x100 units square. This square is
C   then divided into a grid of NxN blocks where N is the order of
C   the current system matrix. 'blocksize' refers to the dimensions
C   of each block in the grid. 'indent' is the amount by which each
C   side of each block is indented to give another block internal to
C   first that defines the limits of the axes to be drawn.*/
        blocksize=100.0/real(NI)
        indent=0.01*blocksize
        axes_len=blocksize-2*indent
        scale=axes_len/2

        do i=1,NI
          blockno=real(NI+1-i)
          do j=1,NJ
            n=(i*j)+(j-1)*(NI-i)
            originy(n)=(blockno-1)*blocksize+indent+axes_len/2
          end do
        end do
        do j=1,NJ
          blockno=real(j)
```

```
          do i=1,NI
            n=(i*j)+(j-1)*(NI-i)
            originx(n)=(blockno-1)*blocksize+indent+axes_len/2
          end do
        end do
        xmax=axes_len/2
        xmin=axes_len/2
        ymax=axes_len/2
        ymin=axes_len/2
        do n=1,NI*NJ
          call graphplace(originx(n),originy(n),xmax,xmin,ymax,ymin)
        end do

        return
        end

C ***************************************************************************
C
C                            SUBROUTINE bound_entry
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine checks to see if the current i,j           *
C *              element is still in the bounds, if not the i,j             *
C *              element in temparray is set to -1.                         *
C *                                                                         *
C ***************************************************************************


        subroutine bound_entry(temp_array,n,bound,multi_bounds,f
     c                          ,setmember)

        include 'common1'
        integer temp_array(500),n,bound,multi_bounds(9000),f,Nn
     c          ,setmember,yes,no

        yes=1
        no=0
        Nn=(n*bound)+(bound-1)*(NI*NJ-n)

        if (setmember.eq.yes)then

                          if(f.eq.1)then
                                    temp_array(n)=bound
                                    multi_bounds(Nn)=1
                          else
                             if (temp_array(n).ne.bound)then
                                               temp_array(n)=-1
                                   if (multi_bounds(Nn).eq.0) then
```

```
                                                    multi_bounds(Nn)=f
                                    end if
                                end if
                            end if
        else
           if(f.eq.1)then
                    temp_array(n)=0
           else
              if (temp_array(n).ne.0)then
                                    temp_array(n)=-1
                                    if (multi_bounds(Nn).eq.0) then
                                                multi_bounds(Nn)=f
                                    end if
              end if
           end if
        end if
        return
        end

C ***********************************************************************
C
C                       SUBROUTINE bound_inc
C
C ***********************************************************************

C ***********************************************************************
C *                                                                     *
C * CALLS: sort_bounds, displ_bounds, change_row, change_col            *
C *                                                                     *
C * DESCRIPTION: This subroutine reads the starting upper bound         *
C *              from the keyboard and the amount by which it is to     *
C *              be incremented at each step. The subroutine calls      *
C *              sort_bounds to establish which elements are in         *
C *              the bounds for each step. The user can choose to       *
C *              reorder the rows and columns after each step in        *
C *              which case change_row and change_col are called.       *
C *              After each iteration displ_bounds is called to show    *
C *              which elements are in the bounds.                      *
C *                                                                     *
C ***********************************************************************


        subroutine bound_inc(matrix,no_steps,point_array,norm_type)

        include 'common1'
        integer on,off,end,error,fin,bound_no,bound_array(500),
     c          multi_bounds(9000),uarray(20),yarray(20),count,no_steps
     c          ,point_array(900)
        real start,inc,matrix(9000),Ubound(20),Lbound(20)
        character ans,UBtype(20),LBtype(20)
        character*4 cls,norm_type

        on=1
        off=0
```

```
      cls=char(27)//'H'//char(27)//'J'
      do count=1,20
        uarray(count)=count
        yarray(count)=count
      end do

      write(*,80)cls
C  */Select new parameters or default/*
      end=off
      do while(end.eq.off)
        write(*,10)
        read(*,100)ans
        if (ans.eq.'d') then
C  */Set default parameters./*
                        start=1.0
                        inc=-0.5
                        end=on
        else
C  */Set new start value and new increment value./*
           if (ans.eq.'n') then
              error=on
              do while(error.eq.on)
                write(*,80)
                write(*,20)
                read(*,110)start
                if ((start.le.1.0).and.(start.ge.0.0))then
                                                    error=off
                end if
              end do
              error=on
              do while(error.eq.on)
                write(*,80)
                write(*,30)
                read(*,110)inc
                if ((abs(inc).le.1.0).and.(abs(inc).ge.0.0))then
                                                          error=off
                end if
              end do
              end=on
           end if
        end if
      end do
      fin=off
      Ubound(1)=start
      Lbound(1)=start
      UBtype(1)='i'
      LBtype(1)='i'
      bound_no=2
      do count=1,NI*NJ
        bound_array(count)=0
      end do
      call sort_bounds(matrix,bound_array,Ubound,Lbound,UBtype,
     c                 LBtype,no_steps,bound_no,multi_bounds
     c                 ,uarray,yarray)
```

```
      write(*,80)cls
      write(*,70)Ubound(1),Lbound(1)
      call displ_bounds(bound_array,uarray,yarray,Ubound(1),Lbound(1)
     c                  ,norm_type)
      fin=off
      do while(fin.eq.off)
        end=off
        do while(end.eq.off)
          write(*,80)cls
          write(*,40)
          read(*,100)ans
          end=on
          if(ans.eq.'r')then
                    call change_row(bound_array,yarray,multi_bounds
     c                                ,bound_no,point_array)
                    call displ_bounds(bound_array,uarray,yarray
     c                             ,Ubound(1),Lbound(1),norm_type)

                     end=off
          else
              if (ans.eq.'c') then
                    call change_col(bound_array,uarray,multi_bounds
     c                                ,bound_no,point_array)
                    call displ_bounds(bound_array,uarray,yarray
     c                          ,Ubound(1),Lbound(1),norm_type)
                     end=off
              end if
          end if
        end do
C */ Enter an 'e' to exit or return to increment the bound and return
C    ./*
        error=on
        do while(error.eq.on)
          write(*,80)cls
          write(*,50)
          read(*,100)ans
          if((ans.eq.' ').or.(ans.eq.'e'))then
                                    error=off
          end if
        end do
        if (ans.eq.' ')then
                    if(inc.le.0.0) then
                                Lbound(1)=Lbound(1)+inc
                                if (Lbound(1).le.0.0)then
                                                Lbound(1)=0.0
                                                LBtype(1)='n'
                                                 fin=on
                                end if
                    end if
                    if(inc.ge.0.0) then
                                Ubound(1)=Ubound(1)+inc
                                if (Ubound(1).ge.1.0)then
                                                Ubound(1)=1.0
                                                 fin=on
```

```
                                    end if
                    end if
          do count=1,NI*NJ
            bound_array(count)=0
          end do
          call sort_bounds(matrix,bound_array,Ubound,Lbound,UBtype,
     c                     LBtype,no_steps,bound_no,multi_bounds
     c                     ,uarray,yarray)
          write(*,80)cls
          write(*,70)Ubound(1),Lbound(1)
          call displ_bounds(bound_array,uarray,yarray
     c                           ,Ubound,Lbound,norm_type)
        else
            fin=on
        end if
      end do
      error=on
      do while(error.eq.on)
         write(*,80)cls
         write(*,60)
         read(*,100)ans
         if (ans.eq.' ')then
                    error=off
         end if
      end do

10    format('use default values (d), or enter new values(n)?=> '
     c        ,NN)
20    format('enter starting value(1>=start>=0.0) => ',NN)
30    format('enter the amount by which bound is to be',/,
     c        'incremented and direction(e.g.+.5) => ',NN)
40    format('enter r or c to reorder or return to continue =>',NN)
50    format('enter e to end or return to continue => ',NN)
60    format('press return key to goto menu.')
70    format('bounds= ',F6.3,'-->',F6.3,//)
80    format(4a4,NN)
100   format(1a1)
110   format(F6.3)

      return
      end
```

```
C ***************************************************************************
C
C                        SUBROUTINE change_col
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine changes the array of pointers that         *
C *              keep track of the elements of G(s) and the elements       *
C *              of bound_array which record the relative dominance         *
C *              of each element. The user enters two columns to be         *
C *              swapped and the subroutine reorders the pointers           *
C *              and the matrix.                                            *
C *                                                                         *
C ***************************************************************************

      subroutine change_col(bound_array,uarray,multi_bounds,bound_no
     c                      ,point_array)
      include 'common1'
      integer bound_array(500),uarray(20),on,off,col1,col2,exit
     c       ,j,n1,n2,store,multi_bounds(9000),bound_no,Nn1,Nn2,
     c       point_array(900)

      on=1
      off=0
      exit=off
      do while(exit.eq.off)
        write(*,10)
        read(*,100)col1,col2
        if ((col1.ge.1).and.(col1.le.NJ)) then
           if ((col2.ge.1).and.(col2.le.NJ)) then
                        do i=1,NI
                           n1=(i*col1)+(col1-1)*(NI-i)
                           n2=(i*col2)+(col2-1)*(NI-i)
                           store=bound_array(n1)
                           bound_array(n1)=bound_array(n2)
                           bound_array(n2)=store
                           store=point_array(n1)
                           point_array(n1)=point_array(n2)
                           point_array(n2)=store
                           do bound=1,bound_no-1
                             Nn1=(n1*bound)+(bound-1)*(NI*NJ-n1)
                             Nn2=(n2*bound)+(bound-1)*(NI*NJ-n2)
                             store=multi_bounds(Nn1)
                             multi_bounds(Nn1)=multi_bounds(Nn2)
                             multi_bounds(Nn2)=store
                           end do
                        end do
                        store=uarray(col1)
                        uarray(col1)=uarray(col2)
                        uarray(col2)=store
```

```
                              exit=on
            end if
          end if
        end do

10      format('enter the two cols to be swaped(col1,col2)=> ',NN)
100     format(2i2,2i2)

        return
        end

C ***************************************************************************
C
C                          SUBROUTINE change_row
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine changes the array of pointers that  *
C *              keep track of the elements of G(s) and the elements *
C *              of bound_array which record the relative dominance  *
C *              of each element. The user enters the two rows to be *
C *              swapped and the subroutine reorders the pointers    *
C *              and the matrix.                                     *
C *                                                                         *
C ***************************************************************************

        subroutine change_row(bound_array,yarray,multi_bounds,bound_no
     c                        ,point_array)
        include 'common1'
        integer bound_array(500),yarray(20),on,off,row1,row2,exit
     c          ,j,n1,n2,store,multi_bounds(9000),bound_no,Nn1,Nn2
     c          ,point_array(900)

        on=1
        off=0
        exit=off
        do while(exit.eq.off)
          write(*,10)
          read(*,*)row1,row2
          if ((row1.ge.1).and.(row1.le.NI)) then
             if ((row2.ge.1).and.(row2.le.NI)) then
                            do j=1,NJ
                               n1=(j*row1)+(j-1)*(NI-row1)
                               n2=(j*row2)+(j-1)*(NI-row2)
C/* swap the columns of the binary matrix*/
                               store=bound_array(n1)
                               bound_array(n1)=bound_array(n2)
                               bound_array(n2)=store
C/* swap the columns of the matrix of pointers*/
                               store=point_array(n1)
```

```
                              point_array(n1)=point_array(n2)
                              point_array(n2)=store
                              do bound=1,bound_no-1
                                Nn1=(n1*bound)+(bound-1)*(NI*NJ-n1)
                                Nn2=(n2*bound)+(bound-1)*(NI*NJ-n2)
                                store=multi_bounds(Nn1)
                                multi_bounds(Nn1)=multi_bounds(Nn2)
                                multi_bounds(Nn2)=store
                              end do
                            end do
                            store=yarray(row1)
                            yarray(row1)=yarray(row2)
                            yarray(row2)=store
                            exit=on
                        end if
                      end if
                    end do

10      format('enter the two rows to be swaped(row1,row2)=> ',NN)

        return
        end

C ***************************************************************************
C
C                              SUBROUTINE check_bounds
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine checks to see if a matrix element          *
C *              falls in or out of the current upper and lower             *
C *              bounds.                                                    *
C *                                                                         *
C ***************************************************************************

        subroutine check_bounds(Nn,UBtype,LBtype,bound,f,matrix,Ubound
     c                          ,Lbound,setmember)

        include 'common1'
        integer n,bound,setmember,f,Nn,bounded,unbounded,upper,yes,no
        real Ubound(20),Lbound(20),matrix(9000)
        character UBtype(20),LBtype(20)

        bounded=1
        unbounded=0
        yes=1
        no=0
        if(UBtype(bound).eq.'i') then
                              if (Ubound(bound).ge.matrix(Nn)) then
                                          upper=bounded
```

```
                                else
                                    upper=unbounded
                                end if
        else
            if(Ubound(bound).gt.matrix(Nn)) then
                                        upper=bounded
            else
                upper=unbounded
            end if
        end if
        if (upper.eq.bounded) then
                  if (LBtype(bound).eq.'i') then
                              if (Lbound(bound).le.matrix(Nn)) then
                                          setmember=yes
                              else
                                  setmember=no
                              end if
                  else
                      if (Lbound(bound).lt.matrix(Nn)) then
                                                  setmember=yes
                      else
                           setmember=no
                      end if
                  end if
        else
            setmember=no
        end if

        return
        end

C ***************************************************************************
C
C                   SUBROUTINE col_sum
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the sum of the elements  *
C *              in a column of a matrix.                                   *
C *                                                                         *
C ***************************************************************************

        subroutine col_sum(GIJ,nosteps,colsum)
        include 'common1'

        real sum,GIJ(9000),colsum(3000)
        integer i,j,f,n,nosteps
```

```
do i=1,NJ*nosteps
  colsum(i)=0.0
end do

do j=1,NJ
  do f=1,nosteps
    sum=0.0
    do i=1,NI
      n=(f*j)+(j-1)*(nosteps-f)
      Nn=(n*i)+(n-1)*(NI-i)
      sum=sum+GIJ(Nn)
    end do
    n=(f*j)+(f-1)*(NJ-j)
    colsum(n)=sum
  end do
end do
return
end
```

```
C ************************************************************************
C
C                          SUBROUTINE displ_bounds
C
C ************************************************************************
C
C ************************************************************************
C *                                                                      *
C * CALLS: vdc_extent(SB), mapping_mode(SB), clip_rectangle(SB,          *
C *        text2d(SB), interior_style(SB), draw2d(SB),                   *
C *        character_height(SB), move2d(SB), clear_view_surface(SB),     *
C *        rectangle(SB), text_color_index(SB), draw_grid,               *
C *        line_color_index(SB), fillrectangle                           *
C *                                                                      *
C * DESCRIPTION: This subroutine draws the borders and writes the        *
C *              text for the diagram that shows the relative            *
C *              dominances of the elements. It also calls the           *
C *              subroutine that draws the grid of rectangles on the     *
C *              screen. It also calls a subroutine that shades the      *
C *              rectangles or crosshatches them depending on the        *
C *              relative dominance of the corresponding element in      *
C *              G(s).                                                   *
C *                                                                      *
C ************************************************************************


      subroutine displ_bounds(bound_array,uarray,yarray,Ubound,Lbound
     c                        ,norm_type)

      include 'common1'
      include '/usr/include/starbase.f1.h'
      include '/usr/include/starbase.f2.h'
      integer bound_array(500),i,j,n,uarray(20),yarray(20),on,off,
     c        exit,cross,full,fildes
      real Ubound,Lbound
      character*4 cls,ans,NULL,norm_type
      character*80 number
      parameter(NULL=char(0))

      cls=char(27)//'H'//char(27)//'J'
      on=1
      off=0
      full=1
      cross=0

      fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
      if (fildes.eq.-1) stop
      call vdc_extent(fildes,-5.0,-5.0,0.0,105.0,105.0,0.0)
      call mapping_mode(fildes,TRUE)
      call clip_rectangle(fildes,0.0,100.0,0.0,100.0)
      call character_height(fildes,3.5)
      call text_color_index(fildes,0.0)
      call clear_view_surface(fildes)
      call interior_style(fildes,INT_SOLID,1)
```

```
   call rectangle(fildes,0.5,90.5,12.0,94.0)
   if (norm_type.eq.'r')then
                        call text2d(fildes,1.0,91.0,'ROW',
+                        VDC_TEXT,FALSE)
   else
      call text2d(fildes,1.0,91.0,'COLUMN',VDC_TEXT,FALSE)
   end if
   call rectangle(fildes,69.0,91.0,96.0,100.0)
   call text2d(fildes,70.0,96.0,'UPPER BOUND:',VDC_TEXT,FALSE)
   call text2d(fildes,70.0,91.5,'LOWER BOUND:',VDC_TEXT,FALSE)
   write(unit=number,fmt='(f5.3)')Ubound
   call text2d(fildes,88.0,96.0,number(1:5),VDC_TEXT,FALSE)
   write(unit=number,fmt='(f5.3)')Lbound
   call text2d(fildes,88.0,91.5,number(1:5),VDC_TEXT,FALSE)
   call rectangle(fildes,1.0,2.0,4.0,8.0)
   call character_height(fildes,3.0)
   call text_color_index(fildes,1.0)
   call text2d(fildes,5.0,6.0,'Element within bounds',
+              VDC_TEXT,FALSE)
   call rectangle(fildes,33.0,2.0,36.0,8.0)
   call line_color_index(fildes,0.0)
   call move2d(fildes,33.0,6.0)
   call draw2d(fildes,36.0,6.0)
   call move2d(fildes,33.0,4.0)
   call draw2d(fildes,36.0,4.0)
   call move2d(fildes,34.5,8.0)
   call draw2d(fildes,34.5,2.0)
   call text2d(fildes,37.0,6.0,'Element partially within',VDC_TEXT
+                             ,FALSE)
   call text2d(fildes,37.0,2.0,'bounds',VDC_TEXT,FALSE)
   call interior_style(fildes,INT_HOLLOW,1)
   call rectangle(fildes,68.0,2.0,71.0,8.0)
   call text2d(fildes,72.0,6.0,'Element outside bounds',
+              VDC_TEXT,FALSE)
   fildes=gclose(fildes)

   write(*,10)cls
   call draw_grid(NI,NJ)
   do i=1,NI
     do j=1,NJ
       n=(i*j)+(j-1)*(NI-i)
       if(bound_array(n).eq.1) then
                        call fillrectangle(NI,NJ,i,j,full)
       else
          if(bound_array(n).eq.-1) then
                           call fillrectangle(NI,NJ,i,j,cross)
          end if
       end if
     end do
   end do
```

```
10      format(4a4)
        return
        end

C ***************************************************************************
C
C                    SUBROUTINE displ_matrix
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: matrix_write                                                     *
C *                                                                         *
C * DESCRIPTION: This subroutine reads the value of the frequency at        *
C *              which the elements of G(s), N(s) and M(s) are to be        *
C *              displayed. It then calls a subroutine that writes          *
C *              the values of the elements of these matrices on the        *
C *              screen.                                                    *
C *                                                                         *
C ***************************************************************************


      subroutine displ_matrix(nosteps,normGIJ,colnormGIJ,fmax,fmin
     c                        ,GIJ)
      include 'commonl'
      real rnumsteps,rnosteps,diffreq,fmax,fmin,normGIJ,colnormGIJ
     c     freq,GIJ
      integer on,off,exit,numsteps,nosteps
      character*4 cls,ans
      cls=char(27)//'H'//char(27)//'J'
      on=1
      off=0
      end=off
      rnosteps=real(nosteps)
      do while(end.ne.on)
      exit=off
       do while(exit.ne.on)
         if (fmax-fmin.ne.0) then
                             write(*,*)'enter frequency =>'
                             read(*,*)freq
                             diffreq=freq-fmin
                            rnumsteps=rnosteps*(diffreq/(fmax-fmin))+1
         else
             rnumsteps=1.0
         end if
         if ((rnumsteps.lt.1).or.(rnumsteps.gt.rnosteps)) then
                                                          exit=off
         else
            exit=on
         end if
       end do
       numsteps=int(rnumsteps)
       write(*,50)cls
```

```
      fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
      if (fildes.eq.-1) stop
      call vdc_extent(fildes,-5.0,-5.0,0.0,105.0,105.0,0.0)
      call clip_rectangle(fildes,-5.0,105.0,-5.0,105.0)
      call mapping_mode(fildes,TRUE)
      do i=1,NI
        do j=1,NJ
          radius=scale*.5
          n=(i*j)+(j-1)*(NI-i)
          do count=1,1
            call move2d(fildes,radius+originx(n),0.0+originy(n))
            do l=1,2*pie*20
              rel=real(l)
              ang=rel/20
              xpoint=radius*cos(ang)+originx(n)
              ypoint=radius*sin(ang)+originy(n)
              call move2d(fildes,xpoint,ypoint)
              call draw2d(fildes,xpoint,ypoint)
            end do
C           radius=scale*.5
          end do
        end do
      end do
      fildes=gclose(fildes)
      return
      end
```

```
C ************************************************************************
C
C                      SUBROUTINE draw_grid
C
C ************************************************************************
C
C ************************************************************************
C *                                                                      *
C * CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB),         *
C *        move2d(SB), draw2d(SB), linetype(SB)                          *
C *                                                                      *
C * DESCRIPTION: This subroutine draws a grid of rectangles on the       *
C *              screen.                                                 *
C *                                                                      *
C ************************************************************************


      subroutine draw_grid(ni,nj)

      character NULL
      integer*4 fildes,ni,nj
      real x,y,rowsize,colsize
      parameter(NULL=char(0))
      include '/usr/include/starbase.fl.h'
      include '/usr/include/starbase.f2.h'

      fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
      if (fildes.eq.-1) stop

      call vdc_extent(fildes,-5.0,-5.0,0.0,105.0,105.0,0.0)
      call mapping_mode(fildes,TRUE)
      call clip_rectangle(fildes,-5.0,105.0,-5.0,105.0)
      call line_type(fildes,SOLID)
      call move2d(fildes,0.0,0.0)
      call draw2d(fildes,0.0,100.0)
      call draw2d(fildes,100.0,100.0)
      call draw2d(fildes,100.0,0.0)
      call draw2d(fildes,0.0,0.0)

      rowsize=80.0/real(ni)
      colsize=80.0/real(nj)
      x=10.0
      do j=1,NJ+1
        call move2d(fildes,x,90.0)
        call draw2d(fildes,x,10.0)
        x=x+colsize
      end do
      y=10.0
      do j=1,NI+1
        call move2d(fildes,10.0,y)
        call draw2d(fildes,90.0,y)
        y=y+rowsize
      end do
```

```
      fildes=gclose(fildes)
      return
      end

C ***********************************************************************
C
C                  SUBROUTINE fillrectangle
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB),        *
C *        move2d(SB), draw2d(SB), line_type(SB),                       *
C *        perimeter_color_index(SB), interior_sytle(SB),               *
C *        rectangle(SB), line_color_index(SB)                          *
C *                                                                     *
C * DESCRIPTION: This subroutine colours in a rectangle on the screen*
C *              It either fills in the rectangle and returns or else*
C *              fills in the rectangle and then crosshatches it,     *
C *              depending on a user supplied parameter.              *
C *                                                                     *
C ***********************************************************************


      subroutine fillrectangle(NI,NJ,i,j,shade)

      character NULL
      integer*4 fildes,ni,nj,i,j,shade,full,cross
      real x,y,rowsize,colsize,ir,jr
      parameter(NULL=char(0))
      include '/usr/include/starbase.f1.h'
      include '/usr/include/starbase.f2.h'

      ir=real(i)
      jr=real(j)
      full=1
      cross=0

      fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
      if (fildes.eq.-1) stop

      call vdc_extent(fildes,-5.0,-5.0,0.0,105.0,105.0,0.0)
      call mapping_mode(fildes,TRUE)
      call clip_rectangle(fildes,-5.0,105.0,-5.0,105.0)
      call line_type(fildes,SOLID)
      call perimeter_color_index(fildes,0.0)
      call interior_style(fildes,INT_SOLID,1)

      rowsize=80.0/real(NI)
      colsize=80.0/real(NJ)
      y2=90-rowsize*(i-1)
      x1=10+colsize*(j-1)
```

```
        x2=x1+colsize
        y1=y2-rowsize
        if (shade.eq.full)  then
                            call rectangle(fildes,x1,y1,x2,y2)
        else
           if (shade.eq.cross)then
                              call rectangle(fildes,x1,y1,x2,y2)
                              call line_color_index(fildes,0.0)
                              row_inc=rowsize/10
                              col_inc=colsize/10
                              x=x1
                              do count=1,9
                                call move2d(fildes,x+col_inc,y2)
                                call draw2d(fildes,x+col_inc,y1)
                                x=x+col_inc
                              end do
                              y=y1
                              do count=1,9
                                call move2d(fildes,x1,y+row_inc)
                                call draw2d(fildes,x2,y+row_inc)
                                y=y+row_inc
                              end do
           end if
        end if
        fildes=gclose(fildes)
        return
        end

C ***************************************************************************
C
C                  SUBROUTINE GIJ_calc
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: tran_eval                                                        *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the value of each element*
C *              of G(s) at points over a user specified frequency   *
C *              range and stores their absolute values in an array. *
C *                                                                         *
C ***************************************************************************



        subroutine GIJ_calc(GIJ,ANG,fmax,fmin,nosteps)
        include 'common1'
        integer i,j,f,n,Nn,nosteps
        real rnosteps,step,revalue,fmax,fmin,GIJ(9000),ANG(9000)
     c       ,imvalue
        complex value
        rnosteps=real(nosteps)
        step=(fmax-fmin)/rnosteps
```

```
      do i=1,NI
        freq=fmin
        do f=1,nosteps
          do j=1,NJ
            call tran_eval(i,j,freq,value)
            n=(f*j)+(j-1)*(nosteps-f)
            Nn=(i*n)+(n-1)*(NI-i)
            GIJ(Nn)=cabs(value)
            revalue=real(value)
            if (revalue.eq.0.0) then
                                revalue=1E-4
            end if
            imvalue=aimag(value)
            ANG(Nn)=atan2(imvalue,revalue)
          end do
         freq=freq+step
        end do
      end do
      return
      end

C ***************************************************************************
C
C                        SUBROUTINE graphplace
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB),      *
C *        move2d(SB), draw2d(SB), line_type(SB), interior_style(SB) *
C *        , rectangle(SB), line_color_index(SB)                      *
C *                                                                         *
C * DESCRIPTION: This subroutine draws the axes of a graph on the     *
C *              screen.                                                *
C *                                                                         *
C ***************************************************************************


      subroutine graphplace(x0,y0,xmax,xmin,ymax,ymin)

      character NULL
      integer*4 fildes
      real x0,y0,xmax,xmin,ymax,ymin
      parameter(NULL=char(0))
      include '/usr/include/starbase.f1.h'
      include '/usr/include/starbase.f2.h'


      fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
      if (fildes.eq.-1) stop

      call vdc_extent(fildes,-5.0,-5.0,0.0,105.0,105.0,0.0)
```

```
      call mapping_mode(fildes,TRUE)
      call clip_rectangle(fildes,-5.0,105.0,-5.0,105.0)
      call line_type(fildes,SOLID)
C     call interior_style(fildes,INT_SOLID,1)
C     call rectangle(fildes,0.0,0.0,100.0,100.0)
C     call line_color_index(fildes,0.0)
      call move2d(fildes,0.0,0.0)
      call draw2d(fildes,0.0,100.0)
      call draw2d(fildes,100.0,100.0)
      call draw2d(fildes,100.0,0.0)
      call draw2d(fildes,0.0,0.0)

      call move2d(fildes,x0-xmin,y0)
      call draw2d(fildes,x0+xmax,y0)
      call move2d(fildes,x0,y0-ymin)
      call draw2d(fildes,x0,y0+ymax)

      fildes=gclose(fildes)
      return
      end

C ***************************************************************************
C
C                        SUBROUTINE matrix_write
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine writes out the values of a matrix          *
C *              on the screen.                                             *
C *                                                                         *
C ***************************************************************************

      subroutine matrix_write(f,matrix,nosteps)
      include 'common1'

      real temparray(10),matrix(9000)
      integer f,i,j,n,Nn,l,nosteps
      do i=1,NI
       do j=1,NJ
          n=(j*f)+(j-1)*(nosteps-f)
          Nn=(i*n)+(n-1)*(NI-i)
          temparray(j)=matrix(Nn)
        end do
        write(*,*)(temparray(l),l=1,NJ)
      end do
      return
      end
```

```
C ***************************************************************************
C
C                          SUBROUTINE norm_cols
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine divides each element in a matrix by        *
C *              by the sum of the elements in the column in which          *
C *              the element falls.                                         *
C *                                                                         *
C ***************************************************************************
C
      subroutine norm_cols(GIJ,nosteps,colsum,colnormGIJ)
      include 'common1'
      integer f,i,j,n1,n2,Nn1,nosteps
      real GIJ(9000),colsum(3000),colnormGIJ(9000)
      do i=1,NI*NJ*nosteps
        colnormGIJ(i)=0.0
      end do

      do f=1,nosteps
        do j=1,NJ
          n1=(j*f)+(j-1)*(nosteps-f)
          do i=1,NI
            Nn1=(n1*i)+(n1-1)*(NI-i)
            n2=(j*f)+(f-1)*(NJ-j)
            colnormGIJ(Nn1)=GIJ(Nn1)/colsum(n2)
          end do
        end do
      end do
      return
      end
```

```
C ***************************************************************************
C
C                          SUBROUTINE norm_rows
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine divides each element in a matrix by        *
C *              the sum of the elements in the row in which the            *
C *              element falls.                                             *
C *                                                                         *
C ***************************************************************************
      subroutine norm_rows(GIJ,nosteps,rowsum,normGIJ)
      include 'common1'
      integer f,i,j,n1,n2,Nn1,nosteps
      real GIJ(9000),rowsum(3000),normGIJ(9000)
      do i=1,NI*NJ*nosteps
        normGIJ(i)=0.0
      end do

      do f=1,nosteps
        do j=1,NJ
          n1=(j*f)+(j-1)*(nosteps-f)
          do i=1,NI
            Nn1=(n1*i)+(n1-1)*(NI-i)
            n2=(i*f)+(f-1)*(NI-i)
            normGIJ(Nn1)=GIJ(Nn1)/rowsum(n2)
          end do
        end do
      end do
      return
      end
```

```
C ***************************************************************************
C
C                        SUBROUTINE options
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine displays a list of options available*
C *              to the user and reads the user's response from the  *
C *              keyboard.                                                  *
C *                                                                         *
C ***************************************************************************


        subroutine options(option)
        integer exit,on,off,option

        on=1
        off=0

        write(*,10)

        write(*,20)
        write(*,30)
        write(*,40)
        write(*,50)
        write(*,60)
        write(*,70)
        exit=off
        do while(exit.ne.on)
          write(*,80)
          read(*,200)option
          if((option.gt.0).and.(option.lt.14)) then
                                              exit=on
          end if
        end do

10      format(/,'OPTIONS LIST')
20      format(//,10x,'(1) display G(jw), M(jw), N(jw)')
30      format(//,10x,'(2) display polar plots ')
40      format(//,10x,'(3) select new frequency range')
50      format(//,10x,'(4) display Z matrix')
60      format(//,10x,'(5) save current matrix')
70      format(//,10x,'(6) end')
80      format(//,'Enter option number =>  ',NN)
200     format(i2)

        return
        end
```

```
C ***********************************************************************
C
C                        SUBROUTINE plot_matrix
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB),        *
C *        move2d(SB), draw2d(SB)                                       *
C *                                                                     *
C * DESCRIPTION: This subroutine plots the polar diagrams of the        *
C *              elements of a matrix on the screen.                    *
C *                                                                     *
C ***********************************************************************


      subroutine plot_matrix(originx,originy,scale,matrix,nosteps,ANG)
      include 'common1'
      include '/usr/include/starbase.f1.h'
      include '/usr/include/starbase.f2.h'
      character NULL
      real xpoint,ypoint,scale,originx(20),matrix(9000)
     c     ,originy(20),ANG(9000),x
      integer*4 fildes
      integer i,j,f,count,n,nosteps,Nn,p
      parameter(NULL=char(0))


      fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
      if (fildes.eq.-1) stop
      call vdc_extent(fildes,-5.0,-5.0,0.0,105.0,105.0,0.0)
      call clip_rectangle(fildes,-5.0,105.0,-5.0,105.0)
      call mapping_mode(fildes,TRUE)
      do i=1,NI
        do j=1,NJ
          p=(i*j)+(j-1)*(NI-i)
          n=(1*j)+(j-1)*(nosteps-1)
          Nn=(n*i)+(n-1)*(NI-i)
          xpoint=originx(p)+(matrix(Nn)*cos(ANG(Nn)))*scale
          ypoint=originy(p)+(matrix(Nn)*sin(ANG(Nn)))*scale
          call move2d(fildes,xpoint,ypoint)
          do f=2,nosteps
            n=(f*j)+(j-1)*(nosteps-f)
            Nn=(n*i)+(n-1)*(NI-i)
            xpoint=originx(p)+(matrix(Nn)*cos(ANG(Nn)))*scale
            ypoint=originy(p)+(matrix(Nn)*sin(ANG(Nn)))*scale
            call move2d(fildes,xpoint,ypoint)
            call draw2d(fildes,xpoint,ypoint)
          end do
        end do
```

```
        end do
        fildes=gclose(fildes)
        return
        end

C ***************************************************************************
C
C                          SUBROUTINE row_sum
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the sum of the elements        *
C *              in a row of a matrix.                                      *
C *                                                                         *
C ***************************************************************************
      subroutine row_sum(GIJ,nosteps,rowsum)
        include 'common1'

        real sum,GIJ(9000),rowsum(3000)
        integer i,j,f,n,nosteps

        do i=1,NI*nosteps
          rowsum(i)=0.0
        end do

        do i=1,NI
          do f=1,nosteps
            sum=0.0
            do j=1,NJ
              n=(f*j)+(j-1)*(nosteps-f)
              Nn=(n*i)+(n-1)*(NI-i)
              sum=sum+GIJ(Nn)
            end do
            n=(f*i)+(f-1)*(NI-i)
            rowsum(n)=sum
          end do
        end do
        return
        end
```

```
C ***************************************************************************
C
C                          SUBROUTINE scale_plot
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: axes, draw_circles, plot_matrix                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine enables the user to plot polar             *
C *              diagrams of the M(s) and N(s) matrices.                    *
C *                                                                         *
C ***************************************************************************

      subroutine scale_plot(nosteps,normGIJ,colnormGIJ,ANG)
      include 'common1'
      real originx(1000),originy(1000),scale,normGIJ(9000),ANG(9000)
     c       ,colnormGIJ(9000),store(9000)
      integer on,off,exit,count,nosteps
      character*4 ans,cls
      cls=char(27)//'H'//char(27)//'J'
      on=1
      off=0
      exit=off
      do while(exit.eq.off)
        write(*,*)'row (r) or col. (c) normalised plot?'
        read(*,100)ans
        if ((ans.eq.'r').or.(ans.eq.'c')) then
                                          exit=on
        end if
      end do
      write(*,1000)cls
      call axes(scale,originx,originy)
      call draw_circles(originx,originy,scale)
      if (ans.eq.'r') then
                      do count=1,NI*NJ*nosteps
                        store(count)=normGIJ(count)
                      end do
      else
         do count=1,NI*NJ*nosteps
           store(count)=colnormGIJ(count)
         end do
      end if
      if (ans.eq.'r') then
                      call plot_matrix(originx,originy,scale,normGIJ
     c                                 ,nosteps,ANG)
      else
         call plot_matrix(originx,originy,scale,colnormGIJ
     c                                 ,nosteps,ANG)
      end if
```

```
100     format(1a1)
1000    format(4a4)
        return
        end

C ***************************************************************************
C
C                          PROGRAM SCALE_PROG
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: GIJ_calc, row_sum, col_sum, norm_cols, norm_rows, options,*
C *        displ_matrix, scale_plot, bound_inc, swap_save                   *
C *                                                                         *
C * DESCRIPTION: This is the main program for the matrix sorting            *
C *              routines. It controls the program flow and calls           *
C *              subroutines that implement options chosen by the           *
C *              user.                                                      *
C *                                                                         *
C ***************************************************************************


        program scale_prog
        include 'common1'
        real fmax,fmin,ANG(9000),GIJ(9000),rowsum(3000),colsum(3000)
     +       ,normGIJ(9000),colnormGIJ(9000),Ubound(20),Lbound(20)
     +       ,rnosteps,normmatrix(9000)
        integer exit,on,off,end,option,nosteps,term,bound_no,
     +          bound_array(500),multi_bounds(9000),uarray(20),
     +          yarray(20),count,finish,point_array(900),fin
        character UBtype(20),LBtype(20)
        character*4 cls,ans,norm_type
        character*20 string

        cls=char(27)//'H'//char(27)//'J'
        off=0
        on=1
        exit=off
        bound_no=1
        write(*,100) cls


        do while(exit.ne.on)
C         call matrix_read
          call choice1
          end=off
C/* set up an array of pointers for the elements in the system matrix*/
          do count=1,NI*NJ
            point_array(count)=count
          end do
          do while(end.eq.off)
            do count=1,20
```

```
              uarray(count)=count
              yarray(count)=count
              Lbound(count)=0.0
              Ubound(count)=0.0
            end do
            do count=1,500
              bound_array(count)=0.0
            end do
            do count=1,9000
              multi_bounds(count)=0.0
            end do
C       /enter the frequency range to be used/
            write(*,120)
            read(*,200)string
            fmax=string_to_number(string,fmax)
            write(*,130)
            read(*,200)string
            fmin=string_to_number(string,fmin)
            write(*,140)9000/(NI*NJ)
            read(*,200)string
            rnosteps=real(nosteps)
            rnosteps=string_to_number(string,rnosteps)
            nosteps=int(rnosteps)
C/* calculate the comparison matrices */
            call GIJ_calc(GIJ,ANG,fmax,fmin,nosteps)
            call row_sum(GIJ,nosteps,rowsum)
            call col_sum(GIJ,nosteps,colsum)
            call norm_cols(GIJ,nosteps,colsum,colnormGIJ)
            call norm_rows(GIJ,nosteps,rowsum,normGIJ)
C/* enter the user option*/
            term=off
            do while(term.eq.off)
              call options(option)

              if (option.eq.1) then
                              call displ_matrix(nosteps,normGIJ,
     +                              colnormGIJ,fmax,fmin,GIJ)
              end if
              if (option.eq.2) then
                              call scale_plot(nosteps,normGIJ
     +                                        ,colnormGIJ,ANG)
              end if
              if (option.eq.3) then
                             term=on
              end if
              if (option.eq.4) then
                            write(*,100)cls
                            fin=off
                            do while(fin.eq.off)
                               write(*,150)
                               read(*,210)ans
                               if(ans.eq.'r')then
                                    norm_type='r'
                                    call bound_inc(normGIJ,nosteps
```

```
     +                                        ,point_array,norm_type)
                                   fin=on
                               else
                                   if(ans.eq.'c')then
                                     norm_type='c'
                                     call bound_inc(colnormGIJ,nosteps
     +                                        ,point_array,norm_type)
                                   fin=on
                                   else
                                     write(*,*)'incorrect entry'
                                   end if
                                end if
                            end do
                end if
                if (option.eq.5) then
                                call swap_save(point_array)
                end if
                if (option.eq.6) then
                                end=on
                                term=on
                end if
              end do
            end do
            exit=on
          end do

100       format(4a4)
110       format(1a1)
120       format(//,'enter fmax=> ',NN)
130       format(//,'enter fmin=> ',NN)
140       format(//,'enter number of points to be calculated',/,
     +            '(must not exceed  ',i6,' )=> ',NN)
150       format('row or column normalised?(enter r or c)=> ',NN)
200       format(a20)
210       format(1a1)
          end
```

```
C ***********************************************************************
C
C                        SUBROUTINE sort_bounds
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: check_bounds, bound_entry                                    *
C *                                                                     *
C * DESCRIPTION: This subroutine sets up an array whose elements are *
C *              either 1, 0 or -1. The elements of the array        *
C *              correspond to the elements of M(s) or N(s) over a   *
C *              frequency range. The value of an element depends on *
C *              whether  the nij or mij element , as the case may be*
C *              , falls into the predetermined bounds.              *
C *                                                                     *
C ***********************************************************************

      subroutine sort_bounds(matrix,bound_array,Ubound,Lbound,UBtype
     c                       ,LBtype,nosteps,bound_no,multi_bounds
     c                       ,uarray,yarray)

      include 'common1'
      integer bound_array(500),nosteps,bound_no,multi_bounds(9000)
     c         ,temp_array(500),i,j,f,bound,setmember,yes,no,Nn1,n1
     c         ,newi,newj,yarray(20),uarray(20),n
      real matrix(9000),Ubound(20),Lbound(20)
      character UBtype(20),LBtype(20)

      yes=1
      no=0

      do j=1,NJ
        do i=1,NI
          newi=yarray(i)
          newj=uarray(j)
          n=(i*j)+(j-1)*(NI-i)
          if(bound_array(n).eq.0) then
                                  do f=1,nosteps
                                    n1=(f*newj)+(newj-1)*(nosteps-f)
                                    Nn1=(newi*n1)+(n1-1)*(NI-newi)
                                    do bound=1,bound_no-1
                                      call check_bounds(Nn1,UBtype
     c                                 ,LBtype,bound,f,matrix,Ubound
     c                                 ,Lbound,setmember)
C                                     if(setmember.eq.yes)then
                                      call bound_entry(temp_array
     c                               ,n,bound,multi_bounds,f,setmember)
C                                     end if
                                    end do
                                  end do
          else
              temp_array(n)=bound_array(n)
```

```
            end if
          end do
        end do

        do n=1,NI*NJ
          bound_array(n)=temp_array(n)
        end do
        return
        end

C ***************************************************************************
C
C                          SUBROUTINE swap_save
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: errorchk                                                         *
C *                                                                         *
C * DESCRIPTION: This subroutine stores the elements of the G(s)     *
C *              matrix in a file. The order of the rows and columns *
C *              is determined by the pointer array.                 *
C *                                                                         *
C ***************************************************************************
        subroutine swap_save(point_array)

        include 'common1'
        character*10 name,cls
        integer flag,l,on,off,errtype,error,point_array(900),
     +          temp_array(900),element
        real rtemp_array(2700)
        cls=char(27)//'H'//char(27)//'J'
        on=1
        off=0
        error=on
        do while (error.ne.off)
        write(*,110)
        read(*,fmt=200,iostat=flag)name
        name='gs.'//name
        open(9,file=name,status='new',access='sequential',iostat=flag)
        if (flag.ne.0) then
                    errtype=70
                    call errorchk(errtype)
        else
            error=off
        end if
        end do

        write(9,*)NI,NJ,NK
        do n=1,NI*NJ
          element=point_array(n)
          temp_array(n)=M(element)
        end do
```

```
      write(9,*)(temp_array(l),l=1,NI*NJ)
      do n=1,NI*NJ
        element=point_array(n)
        temp_array(n)=D(element)
      end do
      write(9,*)(temp_array(l),l=1,NI*NJ)
      do n=1,NI*NJ
        element=point_array(n)
        rtemp_array(n)=GT(element)
      end do
      write(9,*)(rtemp_array(l),l=1,NI*NJ)
      do n=1,NI*NJ
        element=point_array(n)
        do l=1,M(n)+1
          Nnn=(n*l)+(l-1)*(NI*NJ-n)
          Nn=(element*l)+(l-1)*(NI*NJ-element)
          rtemp_array(Nnn)=GN(Nn)
        end do
      end do
      write(9,*)(rtemp_array(l),l=1,NI*NJ*(NK+1))
      do n=1,NI*NJ
        element=point_array(n)
        do l=1,D(n)+1
          Nnn=(n*l)+(l-1)*(NI*NJ-n)
          Nn=(element*l)+(l-1)*(NI*NJ-element)
          rtemp_array(Nnn)=GD(Nn)
        end do
      end do
      write(9,*)(rtemp_array(l),l=1,NI*NJ*(NK+1))

      close(9,status='keep')
100   format(4a4)
110   format(//,'enter file name:   ',4X,NN)
200   format(1a5)
      return
      end
```

```
C ***************************************************************************
C
C                        SUBROUTINE tran_eval
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine evaluates the value of a laplacian        *
C *              transfer function at a particular frequency                *
C *                                                                         *
C ***************************************************************************
      subroutine tran_eval(i,j,freq,value)
      include 'common1'
      integer l,n,i,f,Nn,maxlth
      real mag,w
      complex sumN,sumD,sumT,jw,magc,value


      w=2*3.142*freq
      jw=cmplx(0.0,w)
      sumN=cmplx(0.0,0.0)
      sumD=cmplx(0.0,0.0)
      sumT=cmplx(0.0,0.0)
      maxlth=NI*NJ
      n=(i*j)+(j-1)*(NI-i)

      do l=1,M(n)+1
        Nn=(l*n)+(l-1)*(maxlth-n)
        sumN=(GN(Nn)*(jw**(l-1)))+sumN
      end do

      do l=1,D(n)+1
        Nn=(l*n)+(l-1)*(maxlth-n)
        sumD=(GD(Nn)*(jw)**(l-1))+sumD
      end do
      if ((real(sumD).lt.1E-6).and.(aimag(sumD).lt.1E-6)) then
                                                          sumD=1E-9
      end if
      sumT=sumN/sumD
      value=sumT*cexp(-GT(n)*jw)
      return
      end
```

PERRON-FROBINIUS EIGENVALUE ROUTINES

```
C ***********************************************************************
C
C                  SUBROUTINE complex_matrix
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS:                                                              *
C *                                                                     *
C * DESCRIPTION: This matrix reads a matrix stored in a one             *
C *              dimensional array into a two dimensional array.        *
C *                                                                     *
C ***********************************************************************


      subroutine complex_matrix(GIJ,matrix,nosteps,f)

      include 'common1'
      integer i,j,n,n1,Nn,f,nosteps
      complex*16 GIJ(9000),matrix(20,20)

      do i=1,NI
        do j=1,NJ
          n=(f*j)+(j-1)*(nosteps-f)
          Nn=(i*n)+(n-1)*(NI-i)
          matrix(i,j)=GIJ(Nn)
        end do
      end do
      return
      end
```

```
C ***********************************************************************
C
C                     SUBROUTINE define_part
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS:                                                              *
C *                                                                     *
C * DESCRIPTION: This subroutine reads the number of on_diagonal        *
C *              submatrices of the partitioned system and the order    *
C *              each block.                                            *
C *                                                                     *
C ***********************************************************************


      subroutine define_part(NInew,partions)
      include 'common1'
      integer NInew,partions(20),count,on,off,numpart

      on=1
      off=0

      error=on
      do while (error.eq.on)
        write(*,10)
        read(*,100)NInew
        if (NInew.le.NI) then
                        error=off
        else
            write(*,*)'No. of blocks to great'
        end if
      end do
      error=on
      do while (error.eq.on)
        numpart=0
        do count=1,NInew
           write(*,20)count,count
           read(*,110)partions(count)
           numpart=partions(count)+numpart
        end do
        if (numpart.eq.NI) then
                          error=off
        else
            write(*,*)'sum of rows of blocks must equal NI'
        end if
      end do
```

```
10      format(/,'Enter number of square on-diagonal blocks=> ',NN)
20      format(/,'Enter the no. of row elements in Block',i3,',',i3,
     c          ' => ',NN)
100     format(i3)
110     format(i3)

        return
        end

C ***********************************************************************
C
C                           SUBROUTINE dlam_plot
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB),        *
C *        character_height(SB), line_type(SB), move2d(SB),             *
C *        draw2d(SB), text_path(SB), text2d(SB)                        *
C *                                                                     *
C * DESCRIPTION: This subroutine plots the product of r(C(S)) and       *
C *              djmax on the screen.                                   *
C *                                                                     *
C ***********************************************************************

        subroutine dlam_plot(eigen,nosteps,fmax,fmin)

        include '/usr/include/starbase.f1.h'
        include '/usr/include/starbase.f2.h'
        complex*16 eigen(20)
        real reigen(1000),xscale,yscale,f(1000),fmin,fmax
        integer*4 fildes,i
        integer nosteps
        character*80 number
        character NULL
        parameter(NULL=char(0))

        fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
        if (fildes.eq.-1) stop
        call vdc_extent(fildes,-2.0,-2.0,0.0,10.0,10.0,0.0)
        call clip_rectangle(fildes,-5.0,17.0,-5.0,20.0)
        call mapping_mode(fildes,TRUE)
        call character_height(fildes,0.5)
        call line_type(fildes,SOLID)
        call move2d(fildes,0.0,0.0)
        call draw2d(fildes,0.0,10.0)
        call move2d(fildes,0.0,0.0)
        call draw2d(fildes,10.0,0.0)
        call move2d(fildes,0.0,5.0)
        call draw2d(fildes,10.0,5.0)
        call text_path(fildes,PATH_RIGHT)
        call text2d(fildes,-1.9,8.5,'Djmax*r(C(s))',VDC_TEXT,FALSE)
        call text_path(fildes,PATH_RIGHT)
```

```
        call text2d(fildes,-1.0,5.0,'1.0',VDC_TEXT,FALSE)
        call text2d(fildes,2.5,-1.0,'FREQUENCY(HZ)',VDC_TEXT,FALSE)
        call line_type(fildes,DOT)
        write(unit=number,fmt='(f5.1)')fmax
        call text2d(fildes,9.0,-1.0,number(1:5),VDC_TEXT,FALSE)
        write(unit=number,fmt='(f5.1)')fmin
        call text2d(fildes,-0.5,-1.0,number(1:5),VDC_TEXT,FALSE)
        call text2d(fildes,-1.0,0.0,'0.0',VDC_TEXT,FALSE)



        xscale=10.0/real(nosteps)
        yscale=0.2
        do i=1,nosteps
          reigen(i)=real(eigen(i))/0.2
        end do
        do i=1,nosteps
          f(i)=real(i-1)*xscale
        end do
        call move2d(fildes,f(1),reigen(1))
        do i=2,nosteps
        write(*,*)f(i),reigen(i)
C         call move2d(fildes,f(i),reigen(i))
          call draw2d(fildes,f(i),reigen(i))
        end do
        fildes=gclose(fildes)
        return
        end

C ***************************************************************************
C
C                   SUBROUTINE dnorm
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: invert_matrix, matrix_mult, norm_matrix, errorchk                *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the norm of Gjj(I+Gjj)-1        *
C *                                                                         *
C ***************************************************************************



        subroutine dnorm(norm,a,n)

        complex*16 d(20,20),a(20,20),invd(20,20),norm
        integer i,n
        logical invert

C */ Form I+G /*
        do i=1,n
           d(i,i)=a(i,i)+(1,0)
```

```
        end do
C */ Invert (I+G) /*
        call invert_matrix(d,n,invd,invert)
        if(invert) then
C */ Form the norm of G(I+G)^-1 /*
                call matrix_mult(n,n,n,n,a,invd,d)
                call norm_matrix(d,n,n,norm)
        else
            call errorchk(80)
        end if
        return
        end


C ***********************************************************************
C
C                       SUBROUTINE eigen_plot
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: vdc_extent(SB), clip_rectangle(SB), mapping_mode(SB),      *
C *        character_height(SB), line_type(SB), move2d(SB),           *
C *        draw2d(SB), text_path(SB), text2d(SB)                      *
C *                                                                     *
C * DESCRIPTION: This subroutine plots the eigenvalue r(C(s)) vs      *
C *              frequency.                                             *
C *                                                                     *
C ***********************************************************************

        subroutine eigen_plot(eigen,nosteps,fmax,fmin,eigmax)

        include '/usr/include/starbase.f1.h'
        include '/usr/include/starbase.f2.h'
        complex*16 eigen(20)
        real reigen(1000),xscale,yscale,f(1000),fmin,fmax,eigmax
        integer*4 fildes,i
        integer nosteps
        character*80 number
        character NULL
        parameter(NULL=char(0))

        fildes=gopen('/dev/tty'//NULL,OUTDEV,'hp2623'//NULL,SPOOLED)
        if (fildes.eq.-1) stop
        call vdc_extent(fildes,-2.0,-2.0,0.0,10.0,10.0,0.0)
        call clip_rectangle(fildes,-5.0,17.0,-5.0,20.0)
        call mapping_mode(fildes,TRUE)
        call character_height(fildes,0.5)
        call line_type(fildes,SOLID)
        call move2d(fildes,0.0,0.0)
        call draw2d(fildes,0.0,10.0)
        call move2d(fildes,0.0,0.0)
        call draw2d(fildes,10.0,0.0)
```

```
      call text_path(fildes,PATH_RIGHT)
      call text2d(fildes,-1.5,8.5,'r(C(s))',VDC_TEXT,FALSE)
      call text_path(fildes,PATH_RIGHT)
      call text2d(fildes,2.5,-1.0,'FREQUENCY(HZ)',VDC_TEXT,FALSE)
      write(unit=number,fmt='(f5.1)')fmax
      call text2d(fildes,9.0,-1.0,number(1:5),VDC_TEXT,FALSE)
      write(unit=number,fmt='(f5.1)')fmin
      call text2d(fildes,-0.5,-1.0,number(1:5),VDC_TEXT,FALSE)
      call text2d(fildes,-1.0,0.0,'0.0',VDC_TEXT,FALSE)



      xscale=10.0/real(nosteps)
      yscale=eigmax/10.0
      call move2d(fildes,0.0,1.0/yscale)
      call draw2d(fildes,10.0,1.0/yscale)
      call text2d(fildes,-1.0,1.0/yscale,'1.0',VDC_TEXT,FALSE)
      do i=1,nosteps
        reigen(i)=real(eigen(i))/yscale
      end do
      do i=1,nosteps
        f(i)=real(i-1)*xscale
      end do
      call move2d(fildes,f(1),reigen(1))
      do i=2,nosteps
C       call move2d(fildes,f(i),reigen(i))
        call draw2d(fildes,f(i),reigen(i))
      end do
      fildes=gclose(fildes)
      return
      end
```

```
C ***********************************************************************
C
C                        PROGRAM EIGEN_PROG
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS: choice1, define_part, trans_calc, dlam_plot, lamdapff        *
C *        complex_matrix, eigen_plot                                   *
C *                                                                     *
C * DESCRIPTION: This is the main program. It calls subroutines to      *
C *              read in the matrix G(s) and to determine the           *
C *              partitioning required. The program also calls          *
C *              subroutines that calculate r(C(s)) and                 *
C *              r(C(s))*djmax. Finally subroutines are called that     *
C *              plot these values on the screen.                       *
C *                                                                     *
C ***********************************************************************


      program eigen_prog
      include 'common1'
      integer exit,on,off,end,nosteps,term,numsteps,steps(1000)
     c          ,count,finish,yes,no,NInew,partions(20),f,i,j,invertable
     c          ,iord,jord,subiord
      character*4 cls,ans
      complex*16 GIJ(9000),matrix(20,20),lamdapf,eigen_array(1000)
     c  ,dnorm_max,maxd_array(1000)
      real fmax,fmin,eigmax

      cls=char(27)//'H'//char(27)//'J'
      off=0
      on=1
      yes=1
      no=0
      exit=off
      eigmax=25.0
      write(*,100) cls

C     /enter the frequency range to be used/

        call choice1
10      continue
        call define_part(NInew,partions)
        end=off
C       do while(end.eq.off)
          write(*,*)'enter fmax,fmin,nosteps'
          read(*,*)fmax,fmin,nosteps
          call trans_calc(GIJ,fmax,fmin,nosteps)
          numsteps=nosteps
          do count=1,numsteps
            steps(count)=count
          end do
```

```
            do count=1,numsteps
              f=steps(count)
C */ load values of G(s) for current freq. into matrix. /*
              call complex_matrix(GIJ,matrix,nosteps,f)
              call lamdapff(NI,NInew,partions,matrix,lamdapf,dnorm_max)
              if (lamdapf.eq.(0.0,0.0))then
                                    if(count.ne.1)then
                                         lamdapf=eigen_array(count-1)
                                    else
                                         lamdapf=(0.0,0.0)
                                    end if
              end if
              maxd_array(count)=dnorm_max*lamdapf
              eigen_array(count)=lamdapf
            end do
            write(*,*)'use autoscale?'
            read(*,200)ans
            if(ans.eq.'n')then
                         write(*,*)'enter max value'
                         read(*,*)eigmax
            else
                eigmax=2.0+zabs(eigen_array(1))
                do l=1,numsteps
                  write(*,*)eigen_array(l)
                  if (zabs(eigen_array(l)).gt.eigmax)then
                                  eigmax=2.0+zabs(eigen_array(l))
                  end if
                end do
            end if
          end=off
          do while(end.eq.off)
            write(*,110)
            write(*,120)
            write(*,130)
C           write(*,140)
            read(*,200)ans
            if (ans.eq.'1')then
                        call eigen_plot(eigen_array,numsteps,fmax,fmin
     c                                  ,eigmax)
            end if
            if (ans.eq.'2')then
                         call dlam_plot(maxd_array,numsteps,fmax,fmin)
            end if
            if (ans.eq.'3')then
                          goto 10
            end if
           end do

C
```

```
100     format(4a4)
110     format(/,'(1) ..........plot lamdapf vs freq')
120     format(/,'(2) ..........plot dj*lamdapf vs freq')
130     format(/,'(3) ..........continue               ')
200     format(1a1)
        end

C ***************************************************************************
C
C                       SUBROUTINE eval_tran
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the value of a laplacian        *
C *              transfer function at a particular frequency.               *
C *                                                                         *
C ***************************************************************************

        subroutine eval_tran(i,j,freq,value)
        include 'common1'
        integer l,n,i,f,Nn,maxlth
        real mag,w
        complex*16  sumN,sumD,sumT,jw,magc,value


        w=2*3.142*freq
        jw=dcmplx(0.0,w)
        sumN=dcmplx(0.0,0.0)
        sumD=dcmplx(0.0,0.0)
        sumT=dcmplx(0.0,0.0)
        maxlth=NI*NJ
        n=(i*j)+(j-1)*(NI-i)

        do l=1,M(n)+1
          Nn=(l*n)+(l-1)*(maxlth-n)
          sumN=(GN(Nn)*(jw**(l-1)))+sumN
        end do

        do l=1,D(n)+1
          Nn=(l*n)+(l-1)*(maxlth-n)
          sumD=(GD(Nn)*(jw)**(l-1))+sumD
        end do
        if ((real(sumD).lt.1E-6).and.(dimag(sumD).lt.1E-6)) then
                                                          sumD=1E-9
        end if
        sumT=sumN/sumD
        value=sumT*zexp(-GT(n)*jw)
        return
        end
```

```
C ***************************************************************************
C
C                    SUBROUTINE hess_red
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the hessenburg                  *
C *              reduction of the matrix G(jw).                             *
C *                                                                         *
C ***************************************************************************
C
        subroutine hess_red(a,n)
        integer n,k,l,order,i,j,la,m,int(20)
        complex*16  a(20,20),x,y,eigen(20)
        k=1
        l=n
        la=l-1
        do m=k+1,la
          i=m
          x=(0.0,0.0)
          do j=m,l
            if(zabs(a(j,m-1)).gt.zabs(x))then
                                        x=a(j,m-1)
                                        i=j
            end if
           end do
           int(m-1)=i
            if(i.ne.m)then
                      do j=m-1,n
                        y=a(i,j)
                        a(i,j)=a(m,j)
                        a(m,j)=y
                      end do
                      do j=1,l
                        y=a(j,i)
                        a(j,i)=a(j,m)
                        a(j,m)=y
                      end do
              end if
              if(x.ne.(0.0,0.0))then
                          do i=m+1,l
                            y=a(i,m-1)
                            if (y.ne.(0.0,0.0))then
                                        y=y/x
                                        do j=m,n
                                          a(i,j)=a(i,j)-y*a(m,j)
                                        end do
                                do j=1,l
                                  a(j,m)=a(j,m)+y*a(j,i)
                                end do
```

```
                                end if
                            end do
                        end if
                    end do
                    return
                    end

C ***************************************************************************
C
C                       SUBROUTINE invert_matrix
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine inverts a square matrix, if                *
C *              possible. If the matrix is singular a warning is           *
C *              generated.                                                 *
C *                                                                         *
C ***************************************************************************

      subroutine invert_matrix(matrix1,order,invmatrix,invertable)

      integer mark,n,order,i,j,nplsy,max,k,l,n1,n2,s
      real amax,maxtemp
      complex*16 matrix(40,40),matrix1(20,20),temp,const,det,sign,
     c           x(20),invmatrix(20,20)
      logical invertable
      do l=1,order
         do j=1,order
           matrix(l,j)=(0.0,0.0)
           invmatrix(l,j)=(0.0,0.0)
         end do
         x(l)=(0.0,0.0)
      end do
      sign=(1.0,0.0)
      mark=0
      do i=1,order
         do j=1,order
        matrix(i,j)=matrix1(i,j)
      end do
      end do
      do i=1,order
        do j=order+1,2*order
          if(i.eq.j-order) then
                          matrix(i,j)=(1.0,0.0)
          else
              matrix(i,j)=(0.0,0.0)
          end if
        end do
      end do
```

```
      nplsy=2*order
      do i=1,order-1
        max=i
        n=(i*i)+(i-1)*(order-i)
        amax=zabs(matrix(i,i))
        k=i
10      k=k+1
        n=(k*i)+(i-1)*(order-k)
        maxtemp=zabs(matrix(k,i))
        if (maxtemp.gt.amax) then
                             max=k
                             amax=matrix(k,i)
                             if(k.ne.order) goto 10
                             if (i.eq.max) goto 20
                             l=i-1
                             do while(l.ne.nplsy)
                               l=l+1
                               n1=(i*l)+(l-1)*(order-i)
                               temp=matrix(i,l)
                               n2=(max*l)+(l-1)*(order-max)
                               matrix(i,l)=matrix(max,l)
                               matrix(max,l)=temp
                             end do
                             sign=-sign
        end if
20      j=i
        do while(j.ne.order)
          j=j+1
          n1=(i*j)+(i-1)*(order-j)
          if (matrix(j,i).ne.0.0)then
                               n2=(i*i)+(i-1)*(order-i)
                               const=-matrix(j,i)/matrix(i,i)
                               l=i-1
                               do while(l.ne.nplsy)
                                 l=l+1
                                 n1=(j*l)+(l-1)*(order-j)
                                 n2=(i*l)+(l-1)*(order-i)
                                 matrix(j,l)=matrix(j,l)+matrix(i,l)
     c                                      *const
                                end do
          end if
        end do
      end do
      temp=(1.0,0.0)
      i=0
      do while(i.ne.order)
        i=i+1
        n=(i*i)+(i-1)*(order-i)
        if (zabs(matrix(i,i)).gt.1E-360) then
                             temp=temp*matrix(i,i)
        else
            mark=1
            i=order
```

```
      temp=0.0
  end if
end do
det=sign*temp
if (mark.ne.1) then
               i=order
               do while(i.ne.nplsy)
                 i=i+1
                 k=order
                 k=k+1
                 do while(k.ne.1)
                   k=k-1
                   n=(k*i)+(i-1)*(order-k)
                   x(k)=matrix(k,i)
                   if(k.ne.order) then
                                  j=k
                                  do while(j.ne.order)
                                    j=j+1
                                    n=(k*j)+(j-1)*(order-k)
                                    x(k)=x(k)-matrix(k,j)*x(j)
                                  end do
                   end if
                   n=(k*k)+(k-1)*(order-k)
                   x(k)=x(k)/matrix(k,k)
                 end do
                 l=0
                 do while(l.ne.order)
                   l=l+1
                   n=(l*i)+(i-1)*(order-l)
                   matrix(l,i)=x(l)
                 end do
               end do
end if
if(mark.eq.1) then
              invertable=.false.
else
   invertable=.true.
end if
s=1
do i=1,order
  do j=order+1,2*order
  invmatrix(i,j-order)=matrix(i,j)
end do
end do
return
end
```

```
C ************************************************************************
C
C                    SUBROUTINE lamdapf
C
C ************************************************************************
C
C ************************************************************************
C *                                                                      *
C * CALLS: submatrix, dnorm, invert_matrix, matrix_mult, norm_matrix,    *
C *        errorchk, qr_eigenvalues                                      *
C *                                                                      *
C * DESCRIPTION: This subroutine calls subroutines to calculate the      *
C *              matrix C(jw) and the eigenvalue r(C(s)). It also        *
C *              calculates djmax.                                       *
C *                                                                      *
C ************************************************************************

      subroutine lamdapff(n,nnew,part,G,lamdapf,dnorm_max)
      complex*16 G(20,20),subjj(20,20),invsubjj(20,20),
     c            C(20,20),subc(20,20),subij(20,20),norm,
     c            eigen(20),lamdapf,dnorm_max
      integer i,j,n,nnew,part(20),errnum,z,x
      logical inverted

      inverted=.true.
      do j=1,nnew
        if (inverted) then
          call submatrix(j,j,part,subjj,G)
          call dnorm(norm,subjj,part(j))
          if (j.gt.1) then
                    if (zabs(norm).gt.zabs(dnorm_max))then
                                  dnorm_max=norm
                    end if
          else
              dnorm_max=norm
          end if
          call invert_matrix(subjj,part(j),invsubjj,inverted)
          if (inverted) then
                      do i=1,nnew
                        if (i.eq.j)then
                                  C(i,j)=(0,0)
                        else
                            call submatrix(i,j,part,subij,G)
                            call matrix_mult(part(j),part(j),
     c                        part(i),part(j),subij,invsubjj,subc)
                            call norm_matrix(subc,part(i),part(j)
     c                           ,norm)
                            C(i,j)=norm
                        end if
                       end do
          else
             errnum=80
             call errorchk(errnum)
          end if
```

```
          end if
        end do
        if (inverted) then
                    call qr_eigenvalues(C,nnew,eigen)
                    lamdapf=eigen(1)
                    do i=2,nnew
                      if (zabs(eigen(i)).gt.zabs(lamdapf))then
                                              lamdapf=eigen(i)
                      end if
                    end do
        end if
        return
        end


C ***************************************************************************
C
C                   SUBROUTINE matrix_mult
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine multiplies two matrices together.          *
C *                                                                         *
C ***************************************************************************


        subroutine matrix_mult(NIA,NJA,NIB,NJB,B,A,C)

        integer NIA,NIB,NJA,NJB,jA,iB,na,nb,nc,l
        complex*16 A(20,20),B(20,20),C(20,20),sum
        if (NJB.ne.NIA) then
                        write(*,*)'incompatible matrix dimensions'
        else
            do jA=1,NJA
              do iB=1,NIB
                sum=(0.0,0.0)
                do l=1,NJB
                  sum=B(iB,l)*A(l,jA)+sum
                end do
                C(iB,jA)=sum
              end do
            end do
        end if
        return
        end
```

```
C ************************************************************************
C
C                     SUBROUTINE norm_matrix
C
C ************************************************************************
C
C ************************************************************************
C *                                                                      *
C * CALLS:                                                               *
C *                                                                      *
C * DESCRIPTION: This subroutine calculates the row sum norm of a        *
C *              matrix.                                                 *
C *                                                                      *
C ************************************************************************

      subroutine norm_matrix(C,NIC,NJC,norm)

      integer i,j,n,NIC,NJC
      real maxsum,sum
      complex*16 C(20,20),norm
      maxsum=0.0
      do i=1,NIC
         sum=0.0
         do j=1,NJC
           sum=sum+zabs(C(i,j))
         end do
         if (sum.gt.maxsum) then
                            maxsum=sum
         end if
      end do
      norm=dcmplx(maxsum)
      return
      end
```

```
C ***************************************************************************
C
C                  SUBROUTINE qr_eigenvalues
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: hess_red, qr                                                     *
C *                                                                         *
C * DESCRIPTION: This subroutine calls the subroutines that calculate*
C *              the hessenburg reduction of a matrix and the               *
C *              eigenvalues of the reduced matrix.                         *
C *                                                                         *
C ***************************************************************************

        subroutine qr_eigenvalues(G,n,eigen)
        complex*16 G(20,20),eigen(20),temp(20,20)
        integer n,m,i,j
        real macheps
C       open(1,file='test')
C       read(1,*)n
C       read(1,*)((G(i,j),i=1,n),j=1,n)
        macheps=2E-20
        do i=1,n
          do j=1,n
            temp(i,j)=G(i,j)
          end do
        end do
        m=n
        call hess_red(temp,m)
        call qr(temp,m,macheps,eigen)
C       write(*,*)(eigen(i),i=1,n)
        return
        end
```

```
C ***************************************************************************
C
C                        SUBROUTINE QR
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS:                                                                  *
C *                                                                         *
C * DESCRIPTION: This subroutine calculates the eigenvalues of a            *
C *              complex matrix using the QR algorithm.                     *
C *                                                                         *
C ***************************************************************************

      subroutine QR(h,n,macheps,eigen)

      integer i,j,n,k,l,m,na,its,cnt(20)
      complex*16 h(20,20),p,s,q,u,r,t,w,x,y,z,eigen(20)
      real macheps,sreal,simag
      logical notlast
      t=(0.0,0.0)
10    if (n.eq.0) goto 100
      its=0
      na=n-1
20    do l=n,2,-1
        if(zabs(h(l,l-1)).le.macheps*(zabs(h(l-1,l-1))+zabs(h(l,l))))
     c                                                    goto 30
      end do
      l=1
30    x=h(n,n)
      if(l.eq.n) goto 60
      y=h(na,na)
      w=h(n,na)*h(na,n)
      if(l.eq.na) goto 70
      if(its.eq.30) goto 80
      if((its.eq.10).or.(its.eq.20))then
                          t=t+x
                          do i=1,n
                            h(i,i)=h(i,i)-x
                          end do
                    sreal=abs(real(h(n,na)))+abs(real(h(na,n-2)))
                   simag=abs(dimag(h(n,na)))+abs(dimag(h(na,n-2)))
                          s=sreal*(1.0,0.0)+simag*(0.0,1.0)
                          y=0.75*s
                          x=y
                          w=-0.4375*s**2
      end if
      its=its+1
      do m=n-2,l,-1
        z=h(m,m)
        r=x-z
        s=y-z
        p=(r*s-w)/h(m+1,m)+h(m,m+1)
```

```
          q=h(m+1,m+1)-z-r-s
          r=h(m+2,m+1)
          s=zabs(p)+zabs(q)+zabs(r)
          p=p/s
          q=q/s
          r=r/s
          if(m.eq.l) goto 40
          if(zabs(h(m,m-1))*(zabs(q)+zabs(r)).le.macheps*zabs(p)*
     c     (zabs(h(m-1,m-1))+zabs(z)+zabs(h(m+1,m+1)))) goto 40
        end do
40      do i=m+2,n
          h(i,i-2)=(0.0,0.0)
        end do
        do i=m+3,n
          h(i,i-3)=(0.0,0.0)
        end do
        do k=m,na
          notlast=k.ne.na
          if(k.ne.m)then
                    p=h(k,k-1)
                    q=h(k+1,k-1)
                    if(notlast)then
                              r=h(k+2,k-1)
                    else
                        r=(0.0,0.0)
                    end if
                    x=zabs(p)+zabs(q)+zabs(r)
                    if(x.eq.(0.0,0.0)) goto 50
                    p=p/x
                    q=q/x
                    r=r/x
            end if
            s=zsqrt((p**2)+(q**2)+(r**2))
            if(real(p).lt.0.0)then
                              s=-s
            else
               if(dimag(p)*dimag(s).lt.0.0)then
                                        s=-s
               end if
            end if
            if(k.ne.m)then
                        h(k,k-1)=-s*x
            else
                if(l.ne.m)then
                          h(k,k-1)=-h(k,k-1)
                end if
            end if
            p=p+s
            x=p/s
            y=q/s
            z=r/s
            q=q/p
            r=r/p
```

```
            do j=k,n
              p=h(k,j)+q*h(k+1,j)
              if(notlast)then
                        p=p+r*h(k+2,j)
                        h(k+2,j)=h(k+2,j)-p*z
              end if
              h(k+1,j)=h(k+1,j)-p*y
              h(k,j)=h(k,j)-p*x
            end do
            if(k+3.lt.n)then
                       j=k+3
            else
                j=n
            end if
            do i=1,j
              p=x*h(i,k)+y*h(i,k+1)
              if(notlast)then
                        p=p+z*h(i,k+2)
                        h(i,k+2)=h(i,k+2)-p*r
              end if
              h(i,k+1)=h(i,k+1)-p*q
              h(i,k)=h(i,k)-p
            end do
50          continue
          end do
          goto 20
60        eigen(n)=x+t
          n=na
          goto 10
70        continue
          u=x*y
          p=x+y
          q=p**2
          y=zsqrt(q-4*(u-w))
          eigen(na)=t+(p+y)/2
          eigen(n)=t+(p-y)/2
          cnt(n)=-its
          cnt(na)=its
          n=n-2
          goto 10
80        write(*,*)'QR algorithm failed to converge'
100       continue
          return
          end
```

```
C ***********************************************************************
C
C                       SUBROUTINE submatrix
C
C ***********************************************************************
C
C ***********************************************************************
C *                                                                     *
C * CALLS:                                                              *
C *                                                                     *
C * DESCRIPTION: This subroutine stores a submatrix from G(s) in an     *
C *              array so that it can be separately manipulated.        *
C *                                                                     *
C ***********************************************************************

      subroutine submatrix(inew,jnew,partions,sub,matrix)

      include 'common1'
      integer sumi,sumj,istart,jstart,iend,jend,partions(20),subno
     c         ,count,n,inew,jnew,NInew,i,j,iord,jord
      complex*16 matrix(20,20),sub(20,20)

      do i=1,20
        do j=1,20
         sub(i,j)=(0.0,0.0)
        end do
      end do
      sumj=0
      if(jnew.ne.1) then
                    do count=1,jnew-1
                      sumj=partions(count)+sumj
                    end do
      end if
      sumi=0
      if(inew.ne.1) then
                    do count=1,inew-1
                      sumi=partions(count)+sumi
                    end do
      end if
      istart=sumi+1
      jstart=sumj+1
      iend=sumi+partions(inew)
      jend=sumj+partions(jnew)
      iord=iend+1-istart
      jord=jend+1-jstart
      subno=0
      do j=jstart,jend
        do i=istart,iend
          subno=subno+1
          sub(i-istart+1,j-jstart+1)=matrix(i,j)
        end do
      end do
      return
      end
```

```
C ***************************************************************************
C
C                    SUBROUTINE trans_calc
C
C ***************************************************************************
C
C ***************************************************************************
C *                                                                         *
C * CALLS: eval_tran                                                        *
C *                                                                         *
C * DESCRIPTION: This subroutine evaluates the elements of G(s) over        *
C *              a user specified frequency range and stores the            *
C *              results in an array.                                       *
C *                                                                         *
C ***************************************************************************

      subroutine trans_calc(GIJ,fmax,fmin,nosteps)
      include 'common1'
      integer i,j,f,n,Nn,nosteps
      real rnosteps,step,fmax,fmin
     c     ,imvalue
      complex*16 value,GIJ(9000)
      rnosteps=real(nosteps)
      step=(fmax-fmin)/rnosteps
      do i=1,NI
        freq=fmin
        do f=1,nosteps
          do j=1,NJ
            call eval_tran(i,j,freq,value)
            n=(f*j)+(j-1)*(nosteps-f)
            Nn=(i*n)+(n-1)*(NI-i)
            GIJ(Nn)=value
          end do
         freq=freq+step
        end do
      end do
      return
      end
```