A COMPARISON OF THE INVERSE NYQUIST ARRAY AND POLE ASSIGNMENT TECHNIQUES

by **Rainer Horst Erich Venzke**

Submitted to the University of Cape Town in fulfillment of the requirements for the degree of Master of Science in Engineering.

November 1988

ABSTRACT

This dissertation compares the Inverse Nyquist Array (INA) and the Pole Assignment techniques in multivariable control system design.

The representation of multivariable systems in both the frequency domain and state space is discussed. A laboratory flotation system, for which a model by step response analysis is derived, is used as a practical example for both methods.

A detailed description of the theory of both techniques is given. Particular emphasis is given to how the theory can be applied with the use of a personal computer. Computer-aided control system design programs for the INA and Pole Assignment techniques are included.

A complete feedback control scheme for the flotation rig is designed with the INA technique. Pole Assignment by state feedback is used to improve the speed of response of the rig.

The implementation of a Kalman filter, which is required for the Pole Assignment technique, is also described.

ACKNOWLEDGEMENTS

I wish to thank my supervisor Assoc. Prof. M. Braae for his numerous suggestions and prevailing enthusiasm for my project.

TABLE OF CONTENTS

Abstract i

Acknowledgements ii

Table of Contents iii

List of Figures viii

List of Tables ix

Nomenclature x

1. Introduction 1

1.1 Introduction 1

1.2 Dissertation Organisation 2

1.3 Plant Description 2

1.4 Control System Models 3
1.4.1 Linear and Time-invariant Models 3
1.4.2 Non-linear Plant Behaviour 4
1.4.3 Feedback Systems 5

1.5 Transfer function models 5
1.5.1 Restriction on gij(s) 7

1.6 State space description 8

1.7 Moving between models 10
1.7.1 State Space to Transfer Function 10
1.7.1.1 Faddeev Method 11
1.7.2 Transfer Function to State Space 11

1.8 Controllability and Observability 14
1.8.1 Controllability 14
1.8.2 Observability 14

1.9 Simulation of System Response 15

1.10 Conclusion 16

2. The Flotation Rig 18

2.1 Introduction 18

2.2 Industrial Flotation Systems 18

2.3 The Laboratory Flotation Rig 19
2.3.1 Operating Region 21

2.4 Three-stage Control Scheme 23

2.5 Black-box Identification 23

2.6 Model determination 24
2.6.1 Single-loop PI controller 25
2.6.2 Step Test Procedure 26
2.6.2.1 Step Test Results 29
2.6.2.2 Selecting Responses 40
2.6.2.3 Fitting Transfer Functions 40
2.6.3 Flotation Rig Transfer Function 41
2.6.4 Two-Tank System Model 46

2.7 Conclusion 46

3. The Inverse Nyquist Array Technique 49

3.1 Introduction 49

3.2 Models For The Inverse Nyquist Array Method 49
3.2.1 System Model 49
3.2.2 Model Restrictions 51
3.2.3 System Equations 52
3.2.4 Controllability 53

3.3 Diagonal Systems 53

3.4 The Concept of Diagonal Dominance 54
3.4.1 Graphical Interpretation 55
3.4.2 Gershgorin Bands 56
3.4.3 Extended Nyquist Stability Criterion 57
3.4.3.1 Direct Nyquist 57
3.4.3.2 Inverse Nyquist 57
3.4.4 Gain Margins 58
3.4.5 Ostrowski Bands 59

3.5 Achieving Diagonal Dominance 60
3.5.1 Direct And Pseudo-Diagonalisation 60
3.5.2 Elementary operations 60
3.5.2.1 Renumbering the inputs 61
3.5.2.2 Row operations 62

3.6 Single-Loop Controller Design 63

3.7 Conclusion 64

4. Application Of The Inverse Nyquist Array Technique **66**

4.1 Introduction 66

4.2 Flotation Rig Model 66

4.3 Closed-loop Model 67

4.4 Achieving Diagonal Dominance 68
4.4.1 Uncompensated System 68
4.4.2 Row Operations 69

4.5 Design Of Single-loop PI Controllers 74

4.6 Interpretation Of Initial Conditions 76

4.7 Digital Implementation of K 77
4.7.1 The Basic Digital Controller 78
4.7.2 The Pre-compensator 80
4.7.3 The PI Controllers 82

4.8 Closed-loop Step Response 84

4.9 Conclusion 86

5. The Pole Assignment Design Technique **88**

5.1 Introduction 88

5.2 Pole Assignment In System Design 88

5.3 Dyadic Pole Assignment By State Feedback 89
5.3.1 System Model For State Feedback 89
5.3.2 Equivalent Single-Input System 91
5.3.2.1 Open-Loop ESI System 91
5.3.2.2 Closed-Loop ESI System 93
5.3.3 Closed-Loop Multi-Input System 95
5.3.4 Numerical Techniques In Dyadic Pole Assignment 97
5.3.4.1 Frequency Domain Technique 97
5.3.4.2 Controllable Companion Form Technique 99
5.3.5 Summary Of Dyadic Feedback Design 103
5.3.6 Analysis Of Zero Movement 103
5.3.6.1 Single-Input System Zeroes 104
5.3.6.2 Multi-Input System Zeroes 106
5.3.7 Difficulties Of Method 109
5.3.7.1 Choice of vector q 109
5.3.7.2 Closed-Loop Zeroes 110
5.3.7.3 The Choice Of Pole Positions 111
5.3.7.4 Limitations In Pole Selection 111
5.3.7.5 Resulting Noise Sensitivity 111

5.3.7.6 System Interaction 112
5.3.7.7 Closed-Loop Transient Response 112

5.4 Observer Design 113
5.4.1 The Kalman Filter 113
5.4.2 Sampled Data Stochastic Systems 113
5.4.3 Kalman Filter Equations 115
5.4.4 Estimation Of Variances 117

5.5 Conclusion 119

6. Application Of Pole Assignment 121

6.1 Introduction 121

6.2 Flotation Rig Model 121

6.3 A Kalman Filter For The Flotation Rig 122
6.3.1 Calculation Of Sample Mean And Variance 123
6.3.2 Estimation Of Measurement Noise Matrix 123
6.3.3 Estimation Of System Input Noise Matrix 125
6.3.3.1 Determination of Plant Output Variance 126
6.3.3.2 System Simulation With White Noise Inputs 130
6.3.3.3 Simulation Of The Normal Distribution 132
6.3.4 Kalman Gain Matrix 136
6.3.5 Dynamic Response of Filter 136

6.4 Application Of Pole Assignment 138
6.4.1 Open-Loop Pole Considerations 138
6.4.2 Controllability Criterion 141
6.4.3 Closed-Loop Pole Positions 141
6.4.4 Open-loop Gain Considerations 142
6.4.5 Stability Analysis 144
6.4.5.1 Gain Margin 145
6.4.6 Closed-loop Step Response 146
6.4.6.1 Step Test Procedure 146
6.4.6.2 Step Test 1 146
6.4.6.3 Kalman Filter Response 149
6.4.6.4 Step Test 2 151
6.4.6.5 Step Test 3 153
6.4.6.6 Step Test 4 155
6.4.6.7 General Analysis 158

6.5 Conclusion 158

7. Conclusion 161

7.1 Introduction 161

7.2 Model Determination 161

7.3 Control System Design criteria 162
7.3.1 Stability 162
7.3.2 Integrity 164
7.3.3 Interaction 165
7.3.4 Noise Immunity 165
7.3.5 Tracking Behaviour 166
7.3.6 Controller Structure 166
7.3.7 Design Trade-offs 167

7.4 Future Work 168
7.4.1 Control of Full Flotation Rig 168
7.4.2 A Graphical Technique For Pole Assignment 168

References **171**

Appendices **175**

A) Open-loop Step Responses 176

B) Closed-loop Step Response (INA Technique) 177

C) Closed-loop Step Response (Pole Assignment) 179

D) Software Reference 196
1. INACAD Program Source Listing 196
1.1 Subroutine Overview 196
1.2 Subroutine Index 198
1.3 Include file listing 200
1.4 Source Listing 201
1.5 Menu Tree Structure 303

2. Pole Assignment Routines 310
2.1 Subroutine Overview 310
2.1.1 Implementation of BORRIE method 310
2.2 Subroutine Index 310
2.4 Source Listing 311

LIST OF FIGURES

FIG. 1.1	Multivariable process	3
FIG. 1.2	Water level control system	4
FIG. 1.3	Time domain description	6
FIG. 1.4	Frequency domain description	6
FIG. 1.5	Multivariable unity feedback control system	8
FIG. 1.6	Block diagram of state space model	10
FIG. 2.1	Laboratory flotation rig	19
FIG. 2.2	Operating region	22
FIG. 2.3	Simulated Rougher step response	44
FIG. 2.4	Simulated Scavenger step response	44
FIG. 2.5	Simulated Cleaner step response	45
FIG. 2.6	Simulated Recleaner step response	45
Fig. 3.1	Multivariable control system representation	50
FIG. 4.1	Closed-loop Control Scheme	67
FIG. 4.2	INA Diagram of $G^{-1}(s)$ with Gershgorin circles	69
FIG. 4.3	Multiplication function to eliminate (2,1)	70
FIG. 4.4	INA Diagram of $G^{-1}(s)K_1$ & Gershgorin circles	71
FIG. 4.5	Multiplication function to eliminate (1,2)	72
FIG. 4.6	INA Diagram of $G^{-1}K_1K_2$ & Gershgorin circles	74
FIG. 4.7	Direct Nyquist Diagram of final control	76
FIG. 4.8	Implementation of initial conditions	77
FIG. 4.9	Closed-loop system step response	85
FIG. 5.1	Multivariable System Block Diagram	90
FIG. 5.2	Equivalent Single-input System	92
FIG. 5.3	Closed-loop ESI System	94
FIG. 5.4	Closed-loop Multi-input System	95
FIG. 5.5	State Feedback System with Kalman Filter	116
FIG. 6.1	Level measurements under stationary cond.	124
FIG. 6.2	Steady-state measurements at operating level	127
FIG. 6.3	Calculation of variances	129
FIG. 6.4	Feedback gain configuration	143
FIG. 6.5	Gain moved into forward path	144
FIG. 6.6	Closed-loop response (10% increase in speed)	147
FIG. 6.7	Closed-loop response (20% increase in speed)	152
FIG. 6.8	Closed-loop response (50% increase in speed)	154
FIG. 6.9	Closed-loop response(100% increase in speed)	156
FIG. 7.1	Pole Assignment by State Feedback	163

LIST OF TABLES

TABLE 2.1 Selected list of experiments 28

TABLE 2.2 Step tests used to fit transfer functions 40

TABLE 6.1 Rougher tank simulation results 135

TABLE 6.2 Scavenger tank simulation results 135

TABLE 6.3 Selected closed-loop pole positions 142

NOMENCLATURE

u	Plant input vector
y	Plant output vector
D	Differential operator
G,g	Transfer function
G	Multivariable transfer function matrix
g	Transfer function element of **G**
K	Pre-compensator matrix in frequency domain
L	Post-compensator matrix in frequency domain
F	Feedback matrix in frequency domain
$\mathbf{K}_{pi}$	PI controller matrix
Q	Open-loop transfer function
H	Closed-loop transfer function
I	Identity matrix
τ	Time delay
a	Real coefficients
b	" "
x	State vector
A,B,C,D	State space model matrices
P	Controller matrix in frequency domain

1. INTRODUCTION

1.1 INTRODUCTION

The application of multivariable control theory in system design has in the past been extremely tedious, due to the computational complexity. Although most industrial applications are of multivariable nature, only singlevariable techniques were applied as it was difficult and time consuming to use computers such as mainframes, if available, in the design process. With the advent of the personal computer (PC) this has however drastically changed.

The design engineer can now readily apply the multivariable techniques by making use of computer-aided design (CAD) software. The computer will enable him to design, experiment with and simulate the control scheme before implementation. Such computer programs are now becoming increasingly available, easing the process of control system design and allowing the designer to apply techniques which may have been too complicated in the past.

In the course of this research a complete interactive design program for PC's applying the Inverse Nyquist Array (INA) design technique was developed. Routines for the Pole Assignment technique were also written. A paper outlining the approach was presented by the author.(1)

In this research two multivariable design techniques, namely the INA and Pole Assignment methods were investigated. The methods were applied to a laboratory flotation rig and the

necessary software to aid in the design process as well as implementing the control scheme were developed for a PC.

1.2 DISSERTATION ORGANISATION

The aim of this dissertation is to discuss the relative merits and disadvantages of the INA and the Pole Assignment design techniques.

In this chapter, the subject of the research and the aim of the dissertation are formulated. A basic introduction to multivariable systems is given. In chapter two a transfer function model for the flotation rig is developed. The theory of the INA technique is discussed in chapter three and then implemented in chapter four. Chapters five and six repeat the process for the pole assignment method.

The two techniques are compared in chapter seven, where suggestions for future research is also given.

The main software that was developed in the process of this research and the INA design package in particular are combined in Appendix D entitled *Software Reference*.

1.3 PLANT DESCRIPTION

A plant can be described as being some process which is driven by inputs and has a number of outputs. The outputs of the process are functions of its inputs. No restrictions are made to the types of inputs or outputs. They may be mechanical movement, electrical quantities or chemical substances and depend solely on the process being investigated. It is however necessary for these quantities to be measurable in some way.

Singlevariable processes contain plants with one input and one output only. Multivariable processes, which are investigated in this discussion, have a number of inputs and a number of output quantities. In addition, the inputs of such systems are usually interacting, a single input affects more than one output. Fig. 1.1 shows a block diagram of plant with 2 inputs and 2 outputs.

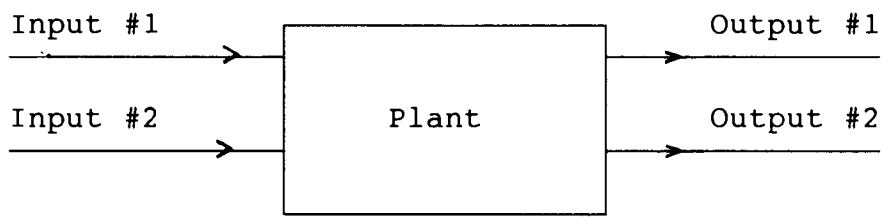


FIG. 1.1 Multivariable process

1.4 CONTROL SYSTEM MODELS

In process control the design engineer is concerned with measuring the outputs of a plant and manipulating its inputs to achieve a desired response of the outputs. To analyse and design a control system, the engineer needs a quantitative model of the plant, which relates the inputs to the outputs.

1.4.1 LINEAR AND TIME-INVARIANT MODELS

It is very intricate to describe in complete detail the behaviour of plants. To avoid complicated models, the analysis of control systems is simplified by modelling the plants with linear and time-invariant differential equations.

1.4.2 NON-LINEAR PLANT BEHAVIOUR

In most plants the requirements of linearity and time-invariance do not hold and to demonstrate the differences, a water level control system (Fig. 1.2) is analysed. In this case the first valve is controlled to keep the level of water in the tank at a fixed setpoint. The second valve acts as a disturbance into the system, which will change the level of the water. The disturbances that affect the plant are termed noise inputs. These are usually numerous and are not contained in the model.

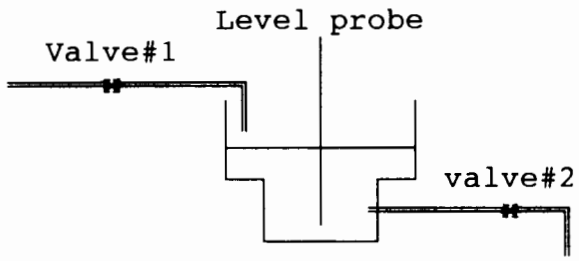


FIG. 1.2 Water level control system

A valve that is opened by pneumatic pressure to regulate the flow into a container has inherent non-linearities. The function relating the pressure to the resulting flow will not be perfectly linear and once the valve is fully open further increases in pressure will not result in bigger flow. The output variables are usually also nonlinear and can also be driven into saturation. The tank collecting the water may have a shape as depicted in Fig. 1.2. For a constant flow into the tank the level will increase more rapidly in the narrower region. Once the tank overflows the measured level will remain at its maximum.

In control theory, methods do exist to analyse non-linear systems, but it is a common practice to rely on the fact that if the input and output variables stay close enough to a given operating point they do behave in a linear fashion.

The simplification of time-invariance also does not hold for most real systems. In the depicted tank system the sensitivity of level probe may depend on the conductivity of the water contained in the tank and this may change over time.

1.4.3 FEEDBACK SYSTEMS

To a certain extent, the inaccuracies in the model can be eliminated by feedback. In feedback systems the output variables are continuously compared to a desired group of setpoints and changes are made to the inputs according to the difference between them.

Using approximate models of plant behaviour greatly simplifies the design process and will lead to acceptable controllers(2) for feedback control systems.

1.5 TRANSFER FUNCTION MODELS

A multivariable plant can be modelled in the time domain by a set of linear differential equations:

$$\mathbf{a}_n D^n \mathbf{y} + \mathbf{a}_{n-1} D^{n-1} \mathbf{y} + \ldots \mathbf{a}_1 \mathbf{y} = \mathbf{u}(t) \qquad (1.1)$$

where D represents the differential operator, vector $\mathbf{y}(t)$

the responses to the driving function vector $\mathbf{u}(t)$ and the corresponding block diagram is shown below.

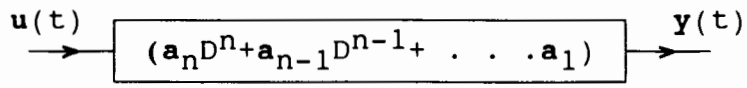


FIG. 1.3 Time domain description

By taking the Laplace transform of the above equation, the frequency domain description with zero initial conditions is then

$$\mathbf{y}(s) = \mathbf{G}(s)\mathbf{u}(s) \tag{1.2}$$

Fig. 1.4 depicts this in block diagram form.

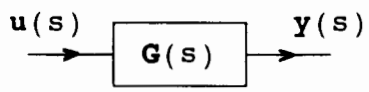


FIG. 1.4 Frequency domain description

A multivariable system given by Eqn. (1.2) having k inputs and m outputs can be written as follows:

$$\begin{bmatrix} y_1(s) \\ y_2(s) \\ \vdots \\ y_m(s) \end{bmatrix} = \begin{bmatrix} g_{11}(s) & g_{12}(s) & \cdots & g_{1k}(s) \\ g_{21}(s) & g_{22}(s) & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ g_{m1}(s) & g_{k2}(s) & \cdots & g_{mk}(s) \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \\ \vdots \\ u_k(s) \end{bmatrix}$$

where

$$\mathbf{g}_{ij}(s) = \frac{a_{c-1}s^{c-1} + a_{c-2}s^{c-2} \ldots + a_0}{s^{d-1} + b_{d-2}s^{d-2} \ldots + b_0} e^{-\tau s} \qquad (1.3)$$

with a_c and b_d being real coefficients and τ is a pure time delay.

1.5.1 RESTRICTION ON $\mathbf{g}_{ij}(s)$

In Eqn. (1.3) $c \leq d$ for all physical processes. A transfer function with numerator order higher than that of the denominator is not realisable.

For most control applications it is required that the high frequency gain of $\mathbf{G}(s)$ is zero, i.e $\mathbf{G}(s)$ is strictly proper(3):

$$\lim_{s \to \infty} \mathbf{g}_{ij}(s) = 0 \qquad (1.4)$$

In most cases $\mathbf{G}(s)$ is chosen to be a square matrix with n=k=m. This is important when multivariable feedback system are designed, because then n individual feedback loops can be constructed such that output $\mathbf{y}_i$ is related to input $\mathbf{u}_i$. This setup is detailed in Fig. 1.5.

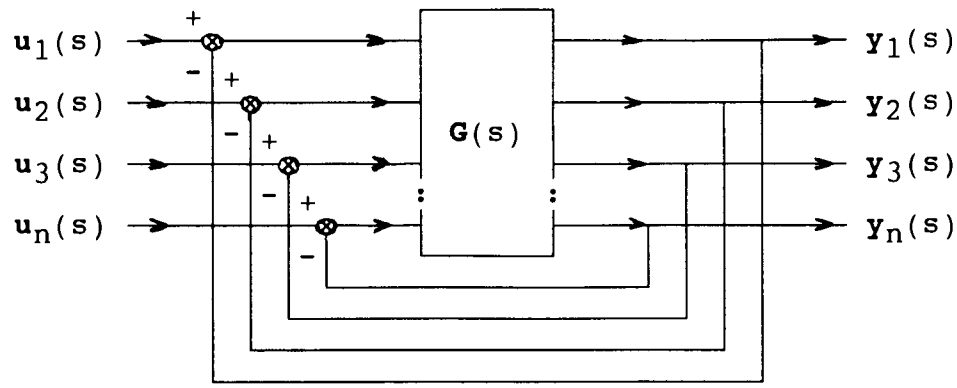


FIG. 1.5 Multivariable unity feedback control system

1.6 STATE SPACE DESCRIPTION

An alternative approach is to model the process by a set of linear first order differential equations:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\,\mathbf{x}(t) + \mathbf{B}\,\mathbf{u}(t) \qquad (1.5)$$

$$\mathbf{y}(t) = \mathbf{C}\,\mathbf{x}(t) + \mathbf{D}\,\mathbf{u}(t) \qquad (1.6)$$

The above equations relate the system variables or states $\mathbf{x}_i(t)$ and their derivatives to the system outputs $\mathbf{y}_i(t)$ and inputs $\mathbf{u}_i(t)$.

If the plant has k inputs and m outputs and the process has n states then

$$\begin{bmatrix} \mathbf{x}'_1(t) \\ \mathbf{x}'_2(t) \\ \vdots \\ \mathbf{x}'_n(t) \end{bmatrix} = \begin{bmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} & \cdots & \mathbf{a}_{1n} \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \cdots & \vdots \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \cdots & \cdot \\ \vdots & & & \\ \mathbf{a}_{n1} & \cdot & \cdots & \mathbf{a}_{nn} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \\ \vdots \\ \mathbf{x}_n(t) \end{bmatrix} + \begin{bmatrix} \mathbf{b}_{11} & \mathbf{b}_{12} & \cdots & \mathbf{b}_{1k} \\ \mathbf{b}_{21} & \mathbf{b}_{22} & \cdots & \cdot \\ \vdots & & & \\ \mathbf{b}_{n1} & \cdot & \cdots & \mathbf{b}_{nk} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1(t) \\ \mathbf{u}_2(t) \\ \vdots \\ \mathbf{u}_k(t) \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{y}_1(t) \\ \mathbf{y}_2(t) \\ \vdots \\ \mathbf{y}_m(t) \end{bmatrix} = \begin{bmatrix} \mathbf{c}_{11} & \mathbf{c}_{12} & \cdots & \mathbf{c}_{1n} \\ \mathbf{c}_{21} & \mathbf{c}_{22} & \cdots & \vdots \\ \mathbf{c}_{21} & \mathbf{c}_{22} & \cdots & \cdot \\ \vdots & & & \\ \mathbf{c}_{m1} & \cdot & \cdots & \mathbf{c}_{mn} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \\ \vdots \\ \mathbf{x}_n(t) \end{bmatrix} + \begin{bmatrix} \mathbf{d}_{11} & \mathbf{b}_{12} & \cdots & \mathbf{d}_{1k} \\ \mathbf{d}_{21} & \mathbf{b}_{22} & \cdots & \cdot \\ \vdots & & & \\ \mathbf{d}_{m1} & \cdot & \cdots & \mathbf{d}_{mk} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1(t) \\ \mathbf{u}_2(t) \\ \vdots \\ \mathbf{u}_k(t) \end{bmatrix}$$

(1.7)

where $\mathbf{x}_i' = \frac{d\mathbf{x}_i}{dt}$

The matrixes **A,B,C,D** have constant coefficients and are of dimensions as depicted above.

The block diagram of the state space model is shown in fig. 1.6. The implementation of state or output feedback is also depicted.

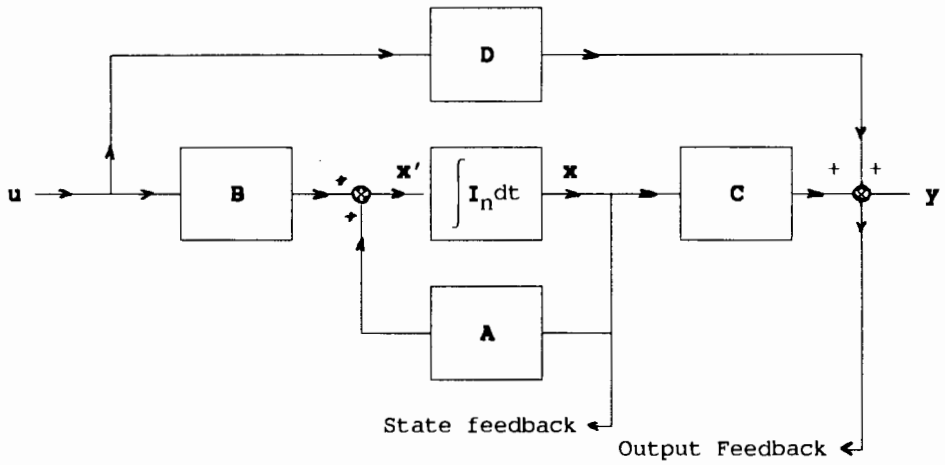


FIG. 1.6 Block diagram of state space model

In most processes **D**=0, which is equivalent to **G**(s) being strictly proper.

1.7 MOVING BETWEEN MODELS

The equation relating the state space representation to the transfer function model is give by

$$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}+\mathbf{D} \qquad (1.8)$$

1.7.1 STATE SPACE TO TRANSFER FUNCTION

The difficulty in moving from state space to the transfer function model lies in evaluating $(s\mathbf{I}-\mathbf{A})^{-1}$.

1.7.1.1 Faddeev Method

The Faddeev algorithm(**4**) computes the adjoint of (s**I**-**A**) and simultaneously calculates the characteristic polynomial $|s\mathbf{I}-\mathbf{A}|$.

$$(s\mathbf{I}-\mathbf{A})^{-1} = \frac{\text{Adj}(s\mathbf{I}-\mathbf{A})}{|s\mathbf{I}-\mathbf{A}|} \tag{1.9}$$

$$= \frac{R_0 s^{n-1}+R_1 s^{n-2}+\ldots+R_{n-2}s+R_{n-1}}{e_0 s^n+e_1 s^{n-1}+\ldots+e_{n-1}s+e_n} \tag{1.10}$$

The matrix coefficients R_i and coefficients e_i are calculated recursively from:

$$e_0 = 1 \qquad R_0 = \mathbf{I}$$

$$e_i = \frac{-1}{i}\,\text{trace}(\mathbf{A}R_{i-1})$$

and

$$R_i = \mathbf{A}R_{i-1} + e_i\mathbf{I}, \quad i=1,n$$

$$R_n = 0. \tag{1.11}$$

Although R_n need not be calculated, it can be used to check the computational accuracy of the method. Because of the recursive nature of the algorithm it is considered to be numerically unsatisfactory.(**4**) The Faddeev algorithm does however posses the feature of a computational check, which other techniques lack.

1.7.2 TRANSFER FUNCTION TO STATE SPACE

Numerous techniques exist to transform a multivariable model **G**(s) to the state space domain.(**3**)

To move from the transfer function representation to that of state space for singlevariable systems is best illustrated by an example.

Let

$$g(s)=\frac{y(s)}{u(s)}=\frac{a_1s+a_0}{s^2+b_1s+b_0} \tag{1.12}$$

By multiplying numerator and denominator of eqn. (1.12) by w(s) and equating them, then

$$s^2w(s)= -b_1sw(s) - b_0w(s) + u(s)$$

and

$$y(s) = a_1sw(s) + a_0w(s) \tag{1.13}$$

Let

$$\mathbf{x}(s) =\begin{bmatrix} sw(s) \\ w(s) \end{bmatrix} \tag{1.14}$$

then

$$sx(s) = \begin{bmatrix} -b_1 & -b_0 \\ 1 & 0 \end{bmatrix}\mathbf{x}(s) + \begin{bmatrix} 1 \\ 0 \end{bmatrix}u(s)$$

$$y(s) = \begin{bmatrix} a_1 & a_0 \end{bmatrix}\mathbf{x}(s) \tag{1.15}$$

For the general case assuming that

$$g(s) = \frac{a_{n-1}s^{n-1} + a_{n-2}s^{n-2} \ . \ . \ . + a_0}{s^{d-1} + b_{d-2}s^{d-2} \ . \ . \ . + b_0} \tag{1.16}$$

then

$$\mathbf{A} = \begin{bmatrix} b_{d-2} & b_{d-3} & \cdot & \cdot & \cdot & b_0 \\ 1 & 0 & \cdot & \cdot & \cdot & \cdot \\ 0 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & & & & & \\ \cdot & & & & & \\ 0 & \cdot & & \cdot & 1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} a_{d-2} & a_{d-3} & \cdot & \cdot & \cdot & a_0 \end{bmatrix}$$

$$D = a_{d-1} \tag{1.17}$$

This technique can also be used to transform a multivariable system $\mathbf{G}(s)$ into state space form in the following manner.

If $\mathbf{A}_{ij}$, $\mathbf{B}_{ij}$, $\mathbf{C}_{ij}$, D_{ij} were found for each $\mathbf{g}_{ij}(s)$ then the state space model can be constructed by setting

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \mathbf{A}_{12} & \cdot & \cdot & \cdot & \cdot \\ \cdot & & & & & \\ \cdot & & & & & \\ 0 & 0 & \cdot & \cdot & \cdot & \mathbf{A}_{nn} \end{bmatrix} \tag{1.18}$$

and forming matrices **B**,**C** and **D** using this structure.

1.8 CONTROLLABILITY AND OBSERVABILITY

In state space models of multivariable systems the criteria of controllability and observability play an important role. They give an indication of whether a plant can be fully controlled, and if not only a subsystem of the plant can be used to design a control scheme. It is thus important, that the model of the plant does not give rise to uncontrollable modes unnecessarily.

1.8.1 CONTROLLABILITY

A system is controllable(**5**) if all the modes can be excited or controlled by the input.

In terms of the state space representation, the system is controllable if and only if

$$\mathbf{P} = (\mathbf{B}\ \mathbf{AB}\ \mathbf{A}^2\mathbf{B}\ \ldots\ \mathbf{A}^{n-1}\mathbf{B}) \qquad (1.19)$$

has rank n.

1.8.2 OBSERVABILITY

A system is observable(**5**) if all the modes affect the output. In terms of the state equations a system is observable if and only if

$$\mathbf{Q} = (\mathbf{C}^T\ (\mathbf{CA})^T\ (\mathbf{CA}^2)^T\ \ldots\ (\mathbf{CA}^{n-1})^T) \qquad (1.20)$$

has rank n.

1.9 SIMULATION OF SYSTEM RESPONSE

The ability to simulate the system is the most important tool in control system design. It is used to develop a model of the plant by comparing the simulated step responses to those of the plant. It can further be used to analyse the closed-loop system behaviour to step inputs, its integrity and its sensitivity to dynamic changes of the plant.

The fourth-order Runge Kutta technique(**6**) can be used for this purpose. This method solves the linear first order differential equation in Eqn. (1.5). If a transfer function model is to be simulated, it must be transformed into its state space description first.

The algorithm can be summarised as follows:

Simulation of

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{1.21}$$

with initial conditions $\mathbf{x}'_0$, $\mathbf{x}_0$, $\mathbf{u}_0$ and choosing a sufficiently small time step size h is achieved by iterating:

$$\mathbf{v}_{1,t} = h(\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t)$$

$$\mathbf{v}_{2,t} = h(\mathbf{A}(\mathbf{x}_t + \tfrac{1}{2}\mathbf{v}_{1,t}) + \mathbf{B}\mathbf{u}_t)$$

$$\mathbf{v}_{3,t} = h(\mathbf{A}(\mathbf{x}_t + \tfrac{1}{2}\mathbf{v}_{2,t}) + \mathbf{B}\mathbf{u}_t)$$

$$\mathbf{v}_{4,t} = h(\mathbf{A}(\mathbf{x}_t + \mathbf{v}_{3,t}) + \mathbf{B}\mathbf{u}_t)$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{1}{6}(\mathbf{v}_{1,t} + 2\mathbf{v}_{2,t} + 2\mathbf{v}_{3,t} + \mathbf{v}_{4,t}) \tag{1.22}$$

where $\mathbf{x}_{t+1}$ is an approximation to $\mathbf{x}(t+h)$.

The accuracy of the above method relies heavily on the choice of h. The time step must be small enough for the

iterations to approximate the derivative $\mathbf{x}'(t)$ accurately, but if h is too small, then the floating point accuracy of the computer will introduce significant rounding errors. This algorithm is relatively easy to implement on a digital computer and an online adjustment of h by trial and error usually gives acceptable results. More accurate methods exist(7), but they are often unnecessarily complicated.

1.10 CONCLUSION

Although numerous representations exist, the transfer function and state space models for linear, time-invariant multivariable systems were presented in particular. A detailed description of changing from one to the other was given.

The transfer function model is usually used to establish a model from step response analysis, whereas the state space model is derived scientifically. The former will be found for the flotation rig in the next chapter.

2. THE FLOTATION RIG

2.1 INTRODUCTION

In this chapter industrial flotation systems with particular reference to the one at Black Mountain are described. A detailed description of the laboratory flotation rig is given and the desired operating region is specified.

A linear transfer function modelling the rig is found from step tests and a sub-system is selected from it, which will be used to apply the Inverse Nyquist and Pole Assignment techniques in later chapters.

2.2 INDUSTRIAL FLOTATION SYSTEMS

Flotation systems are mainly used in South Africa in the mineral extraction industry. A detailed description of such an application is given by Twidle(**8**). In that system copper, lead and zinc are recovered from ground complex sulphide ore by sequential flotation. By-products such as silver are also produced. The crushed pulp is sequentially treated with chemical reagents in different tanks of the flotation circuit to produce the desired concentrate.

The levels of slurry in the various tanks and the chemical properties such as pH make up the controlled entities.

2.3 THE LABORATORY FLOTATION RIG

The laboratory flotation rig has been built to model the flow characteristics of industrial flotation systems. The plant structure is depicted in Fig. 2.1. The model consists of one circuit, which is made up of four individual flotation extraction tanks, simulating the Rougher, Scavenger, Cleaner and Recleaner.

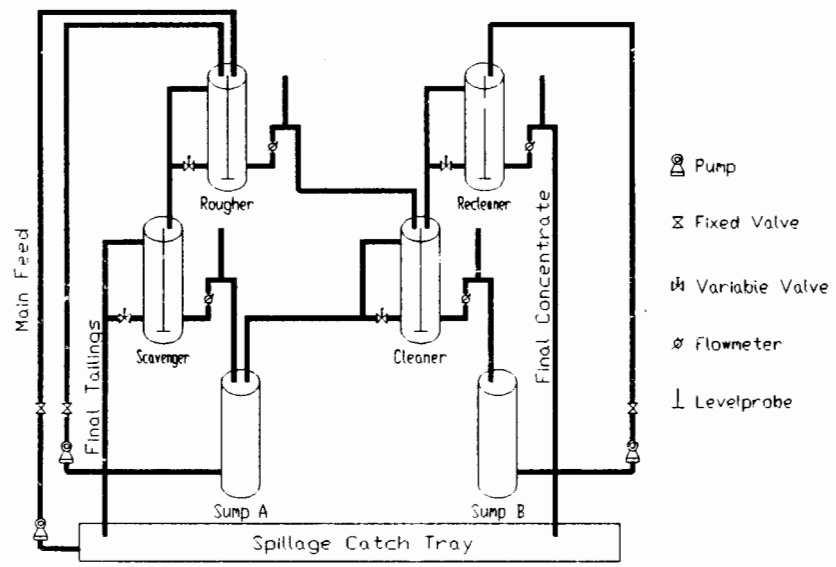


FIG. 2.1 Laboratory flotation rig

In a industrial plant the pulp is mixed with chemical substances such that the desired product or concentrate accumulates in the froth which then passes to the next extraction process, where additional chemicals may be added. Once most of the concentrate is removed the remaining pulp, termed the final tailings, may pass to another circuit where the process is repeated to extract a different product.

In the laboratory setup tap water instead of pulp is used. The water is pumped from a spillage catch tray at a fixed rate into the Rougher tank. The concentrate in the 'froth' passes onto the Cleaner tank. The pipes passing the concentrate are mounted at the bottom to simplify construction, which is acceptable as no froth is actually produced in the tanks. The tailings of the Rougher tank enter the Scavenger tank, from which the concentrate is pumped back into the Rougher to repeat the first extraction process. The tailings of the Scavenger are discarded into the spillage catch tray.

The concentrate collected from the Rougher and Scavenger tanks enter the Cleaner from which the concentrate moves on to the Recleaner and the tailings are fed back to the Rougher via a sump. The concentrate obtained from the Recleaner tank is the end-product of this flotation circuit and would normally contain a high concentration of the extracted mineral. The tailings of the Recleaner are passed back into the Cleaner to re-enter the second filtering process.

Overflow protection pipes are mounted on all the tanks.

In a real plant the individual tanks may consist of a number of tanks with their own filtering processes.

The levels of the pulp in the four tanks and the flows of the concentrate are the measured variables in the model. Other controlled entities such as the addition of chemicals to and measurements such as pH of the pulp are not included in the model.

The signals from the level probes and flowmeters are read by an analog to digital (A/D) converter into the microcomputer. A D/A card is used to control the pneumatic outlet valves of the individual tanks from the computer.

The range of values for the level measurements, flowmeters and the output to the valves are from 0 to 4095, which will be referred to as units. A maximum level and flow reading corresponds to 4095 units and a complete closure of a valve is achieved by writing a value of 4095 units.

2.3.1 OPERATING REGION

An operating region for the valves and water level needs to be defined in which a mathematical model, as discussed in chapter one, describes the behaviour of the flotation rig in a linear fashion.

The operating regions of the level of water in all tanks are defined in Fig. 2.2.

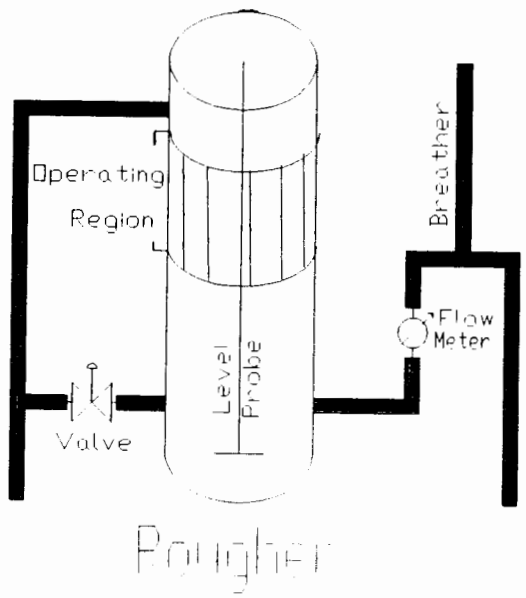


FIG. 2.2 Operating region

It is necessary to operate the plant in these regions as severe changes in the plant characteristics take place when the height of water move beyond these limits. If for example the Rougher level falls below the operating region then no connection exists between it and the Cleaner tank, and whereas before a change in the Rougher valve opening would affect the level of the Cleaner this would not occur anymore. Similarly, the tanks will overflow if the maximum levels are exceeded. Additional changes occur when the valve

inputs are saturated. The valve settings must thus be kept well inside their operating regions.

Less severe non-linearities exist too. The level probes and the valve characteristics are not perfectly linear. The flow from the tanks are functions of the height of water in them and the pumps are not perfect. Nonetheless, for small departures from a selected operating point, the behaviour can be approximated by a linear model.

If the system is kept near to this operating point a simplified linear model can describe the plant with sufficient accuracy. Obviously the plant will not always stay at this point and the success of a control scheme can be measured by its insensitivity to small deviations between the plant and its model.

2.4 THREE-STAGE CONTROL SCHEME

Flotation systems are usually controlled in three stages. During the first stage a start-up controller is applied to move the system into its operating region. A setpoint tracking controller is then inserted to control the levels and chemical balances. The third controller is used to initiate the shut down sequence as otherwise failures such as overflowing may occur.

2.5 BLACK-BOX IDENTIFICATION

The so-called Black-box identification procedure(**9**) is applied to determine the transfer function of the flotation rig by step test analysis. It is well known that feedback control does not require a very accurate model of the plant and a detailed examination of the maximum allowable

inaccuracies has been given by Donati(**10**). Further, it is widely accepted that a simple and reduced-order model is usually sufficient to design an appropriate control scheme.(**2**)

2.6 MODEL DETERMINATION

The flotation rig has four valve inputs. It is thus only possible to control four of the eight output quantities directly and so obtain a *n*-input, *n*-output model of the plant. The flow measurements, which are usually difficult to obtain in an industrial plant, are disregarded and the level measurements are taken as the plant outputs.

The transfer function relating the valve inputs $\mathbf{u}(s)$ to the level outputs $\mathbf{y}(s)$ is given by

$$\mathbf{y}(s) = \mathbf{G}(s)\,\mathbf{u}(s), \tag{2.1}$$

with

$\mathbf{y}_1(s)$ = Rougher level,

$\mathbf{y}_2(s)$ = Scavenger level,

$\mathbf{y}_3(s)$ = Cleaner level,

$\mathbf{y}_4(s)$ = Recleaner level

and

$\mathbf{u}_1(s)$ = Rougher tailings valve,

$\mathbf{u}_2(s)$ = Scavenger tailings valve,

$\mathbf{u}_3(s)$ = Cleaner tailings valve,

$\mathbf{u}_4(s)$ = Recleaner tailings valve.

2.6.1 SINGLE-LOOP PI CONTROLLER

Singlevariable PI controllers were used to move the waterlevels $\mathbf{y}_i$ of the plant into their operating regions by adjusting the corresponding inputs $\mathbf{u}_i$. These were necessary as it was exceedingly difficult to adjust the valves manually to obtain a stable operating point. This problem arose due to the high interaction between the various tanks making it difficult to predict the affect of changing a valve setting on the levels of the different tanks. As the PI controllers were ignorant of these interactions, their individual gains had to be chosen carefully to allow all levels to reach their operating points.

Although the plant can be considered to be open-loop stable, at least in the operating region, it was found that if the valve settings were not perfect the levels would slowly move out of the operating regions and some tanks would eventually run dry.

A single-loop PI controller $\mathbf{P}(s)$ was designed by trial and error.

$$\mathbf{P}(s) = \frac{1}{10s}\begin{bmatrix} 2s+2 & 0 & 0 & 0 \\ 0 & 5s+5 & 0 & 0 \\ 0 & 0 & 6s+6 & 0 \\ 0 & 0 & 0 & 3s+3 \end{bmatrix} \qquad (2.2)$$

This controller was used to move the levels to their operating levels, at which point the valve settings were fixed and the plant was given time to attain steady state. The step tests could then be performed by changing the valve settings by a specific amount.

Even with the use of the controller, it was found that depending on the moment when the inputs were fixed, the levels would sometimes still drift out of the operating regions. The reason for this was that the valve inputs were not stationary at the time when they were clamped as the PI controller's gain was large enough such that the measurement noise was transmitted to the inputs. Reducing the gain however was not an acceptable decision as it took approximately 10 minutes for the plant to reach the desired levels and decreasing the gain would have made the time even longer. Once the valve settings had been fixed, a further 10 minutes was required to allow the tank to attain steady-state.

The above difficulties could be partially elliminated by using a non-linear PI controller as applied in a previous project(**11**), where the controller gain is reduced as the levels near their setpoints. This approach was unncessary in the case of the laboratory rig, where the time involved to obtain step tests was not critical.

The manual valve setting for the main feed was also critical. If the flow into the flotation circuit was too little then the valves would be virtually closed and the levels would drop below their operating region. If alternatively the flow was too high then the valves would be completely open and the tanks would eventually overflow. The tolerance of the valve setting for the main feed was found to be extremely small.

2.6.2 STEP TEST PROCEDURE

During the step tests, which took about 10 minutes, the plant had to remain in its operating region to prevent large

changes in the model, for reasons discussed previously. It was also important that the plant had attained steady-state before a step is applied. Otherwise the step responses would be distorted, if the levels were drifting prior to the time of the step change.

The fixed valves for the pumps of Sump A and Sump B were opened sufficiently such that no water would accumulate in them, otherwise they would act as storage devices and change the model.

A large number of step-tests for each valve had to be performed such that an optimal step size could be found. If the step size was too small, no response could be detected due to the valve characteristics and because the level measurements are contaminated by noise. If however, the step size was too large, then the levels would move out of their operating regions and invalidate the assumption of linearity.

In Table 2.1 selected step tests with their corresponding step size in percent are listed. The percentage closing was calculated from:

$$\text{Percent closing} = \frac{u_t - u_o}{4096} \times 100\% \qquad (2.3)$$

where u_o is the initial valve setting and u_t is the final setting.

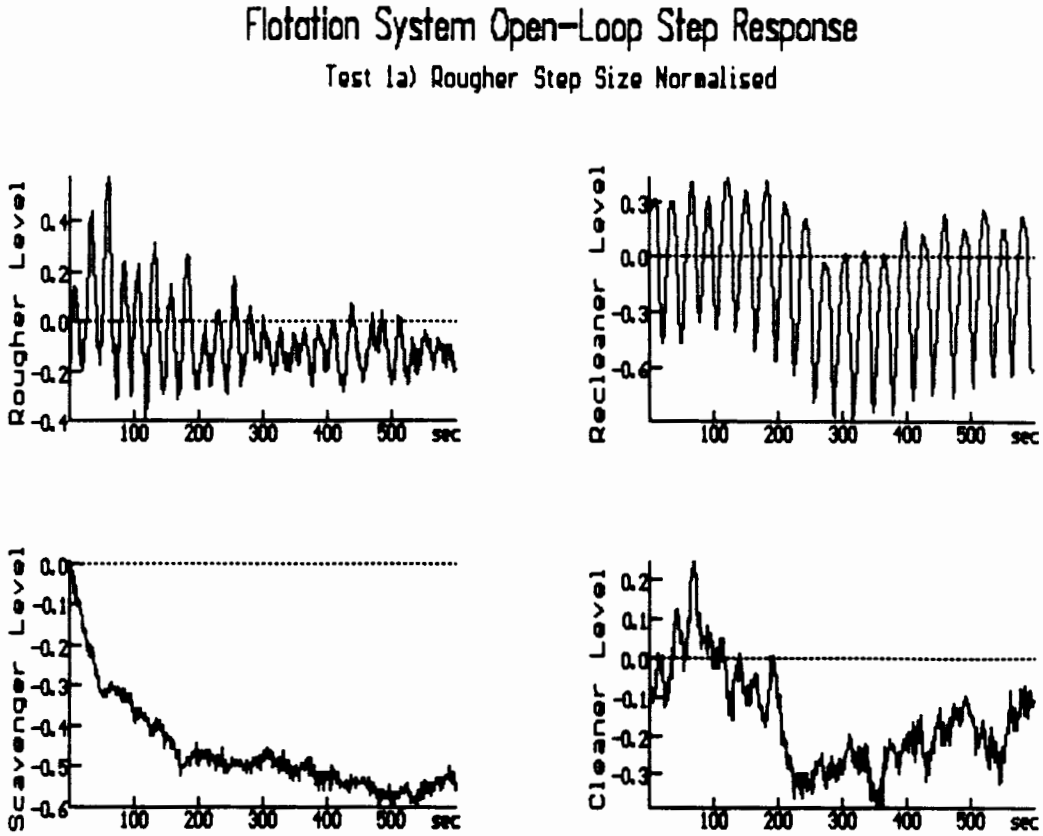
Step test No.	Valve stepped	Percentage closing
1.a)	Rougher	-5%
b)	"	10%
c)	"	-20%
d)	"	25%
e)	"	20%
f)	"	-20%
2.a)	Scavenger	-5%
b)	"	10%
c)	"	10%
d)	"	-20%
3.a)	Cleaner	5%
b)	"	-9%
c)	"	15%
d)	"	15%
e)	"	-10%
4.a)	Recleaner	10%
b)	"	-20%
c)	"	30%
d)	"	20%
e)	"	-30%

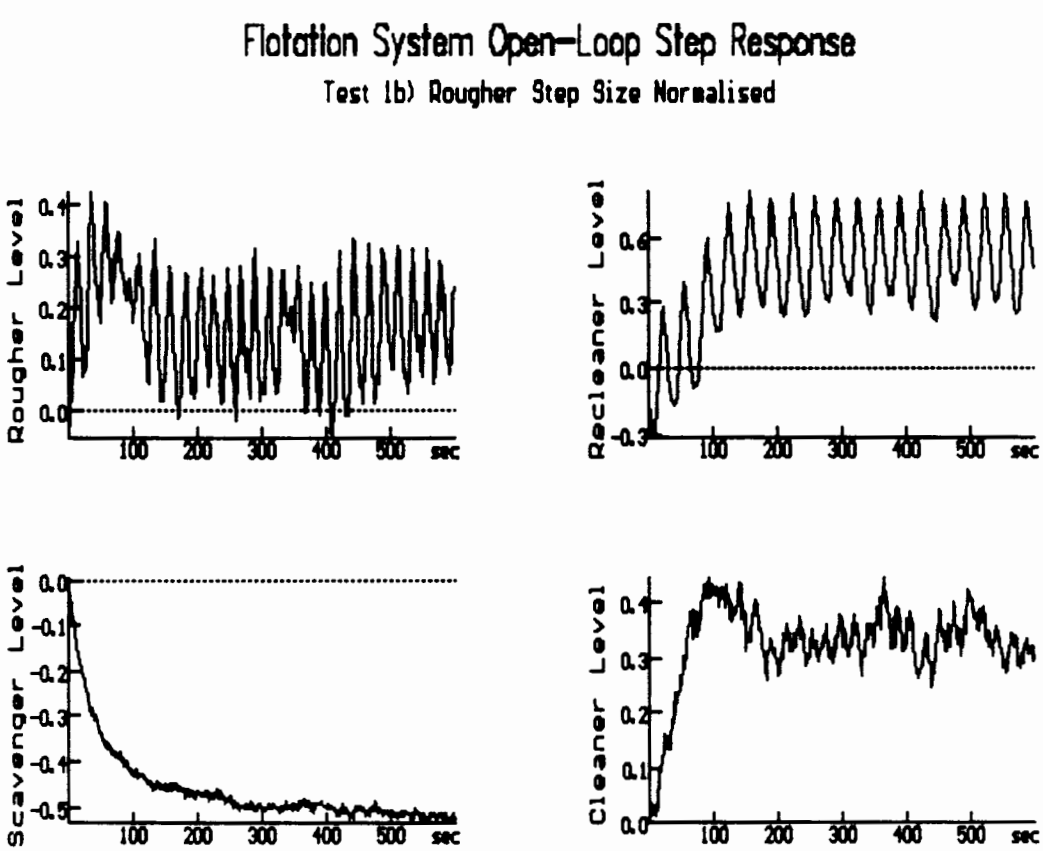
Table 2.1 Selected list of experiments

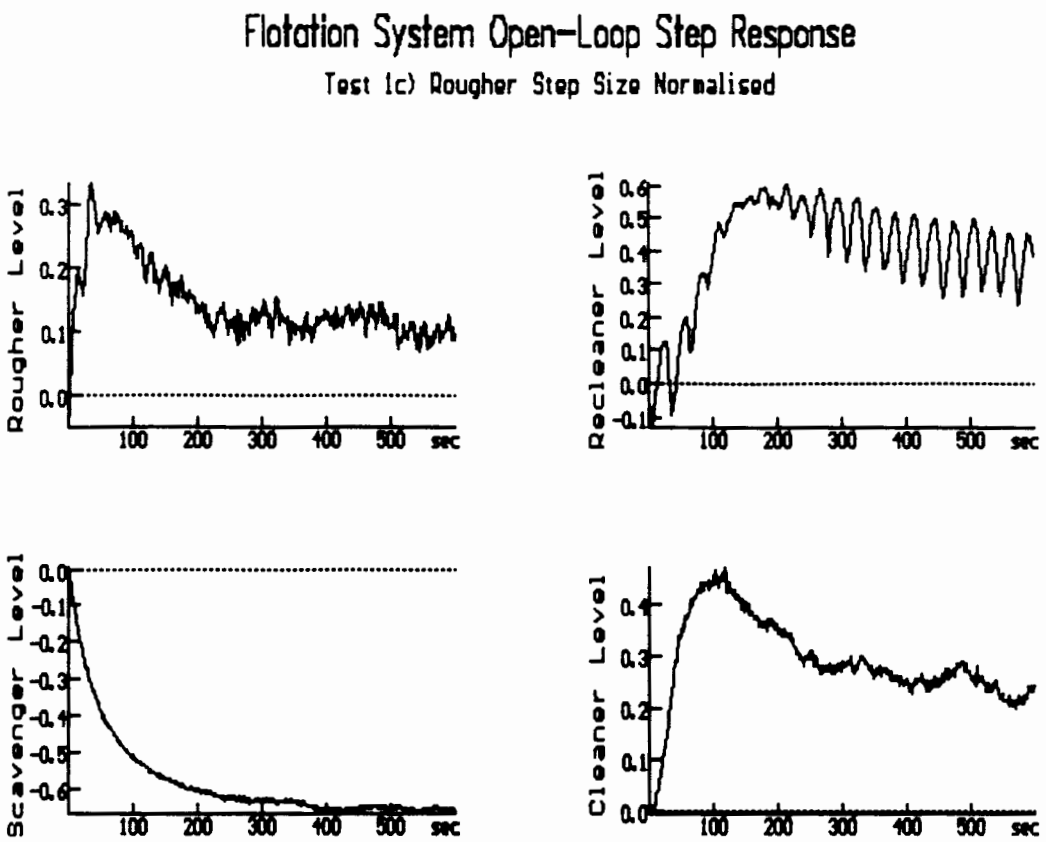
The tests were performed and the results were normalised such that they could be compared with each other and a transfer function could be fitted to them. Normalisation is achieved by subtracting the initial level from the readings and dividing the result by the step size (u_t-u_o). The resulting plots then correspond to a system with zero initial conditions and a step size of +1 units (i.e. opening the valve by 1 unit).

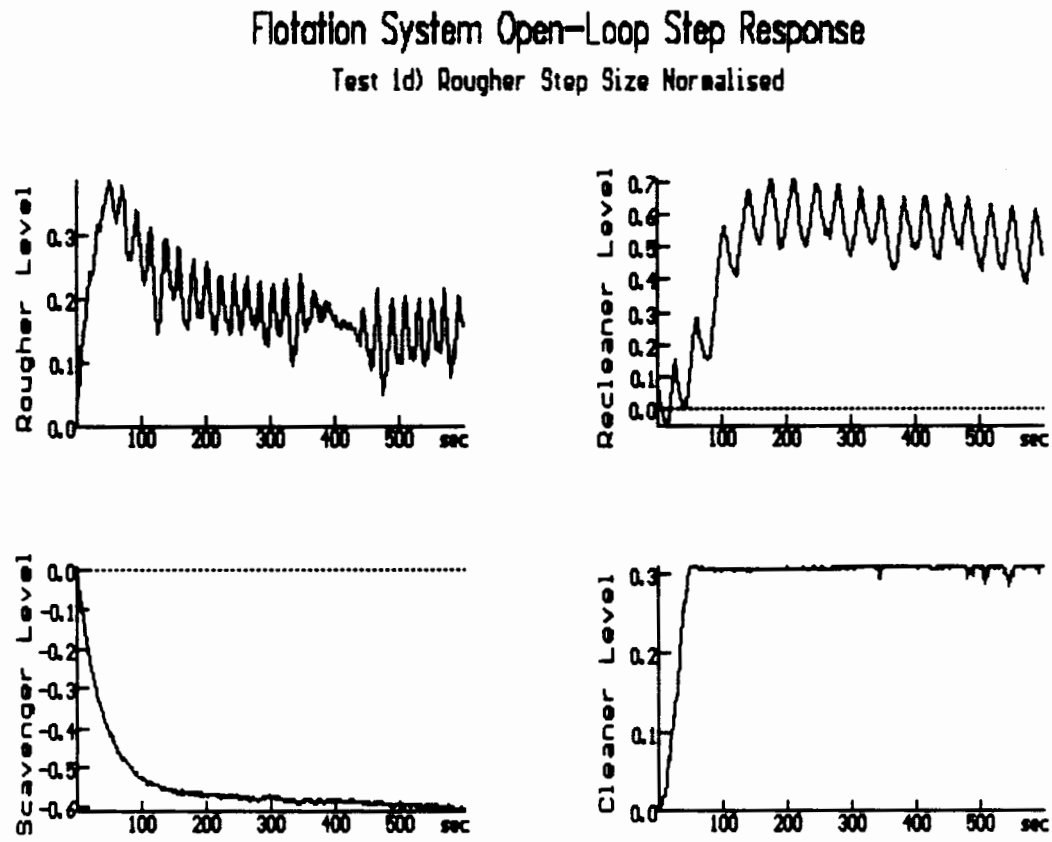
2.6.2.1 Step Test Results

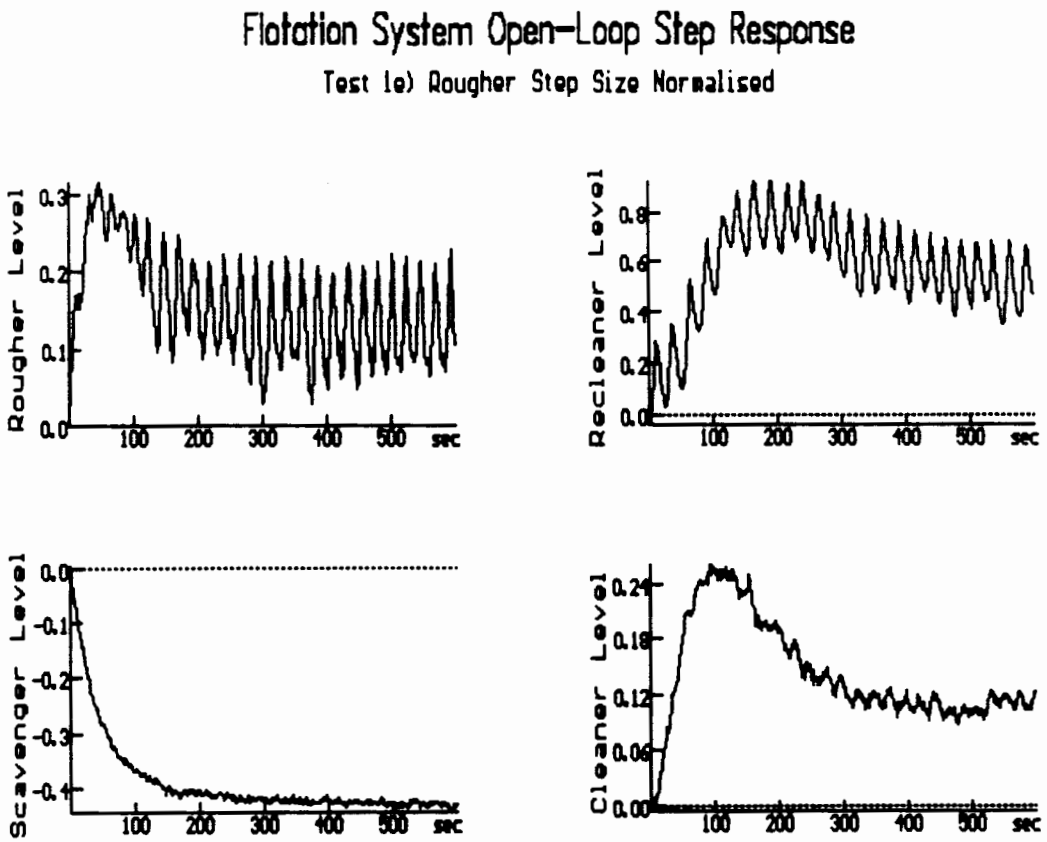
The normalised responses are shown in the following pages. The original data is contained in Appendix A.

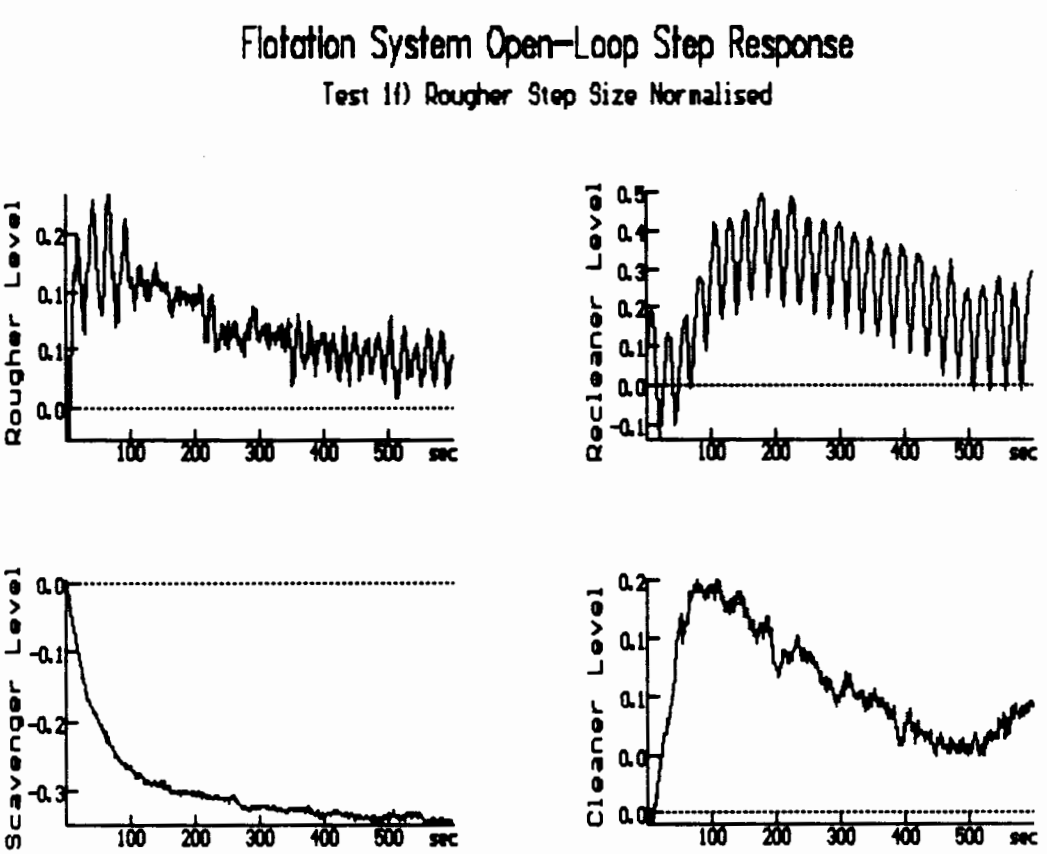

Flotation System Open-Loop Step Response
Test 1a) Rougher Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

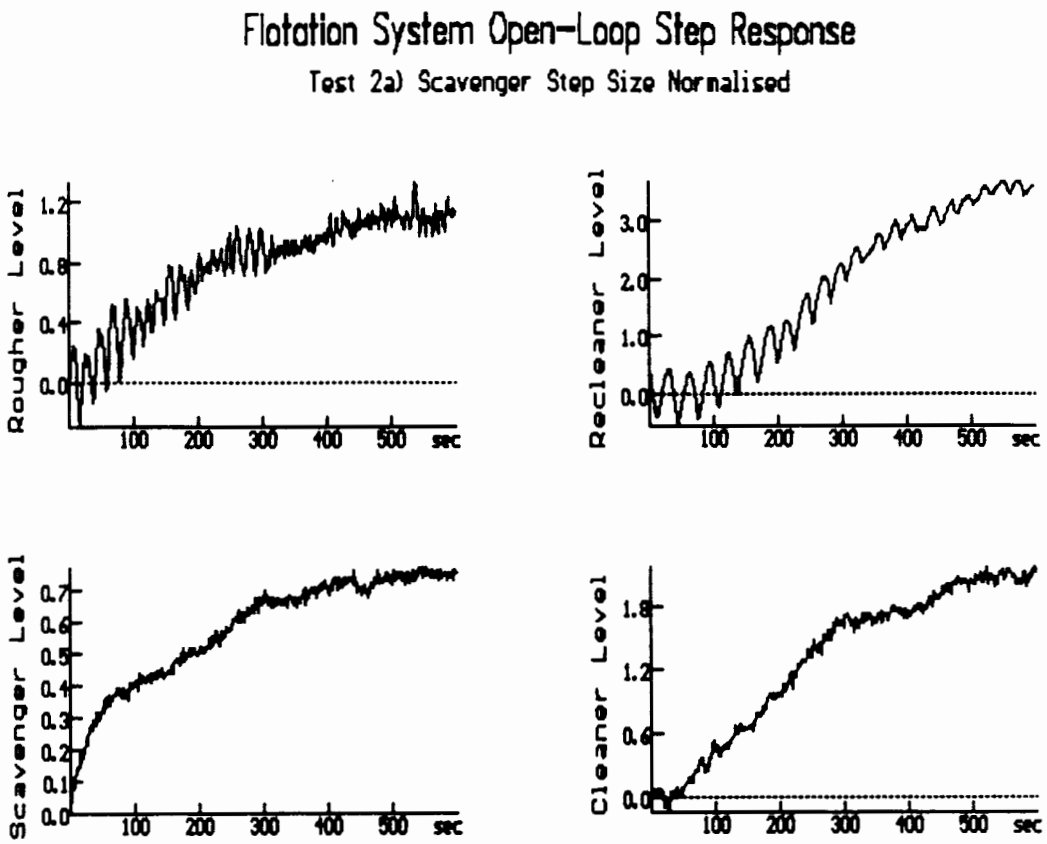

Flotation System Open-Loop Step Response
Test 1b) Rougher Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

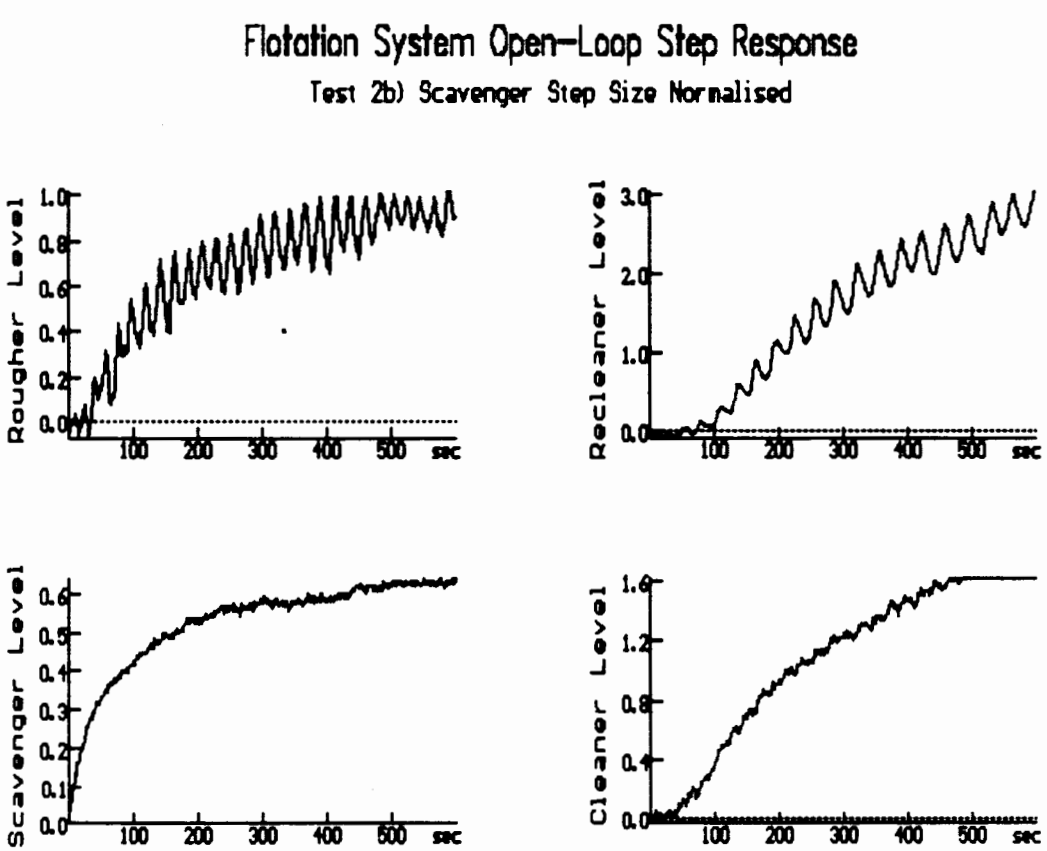

Flotation System Open-Loop Step Response
Test 1c) Rougher Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 1d) Rougher Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 1e) Rougher Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

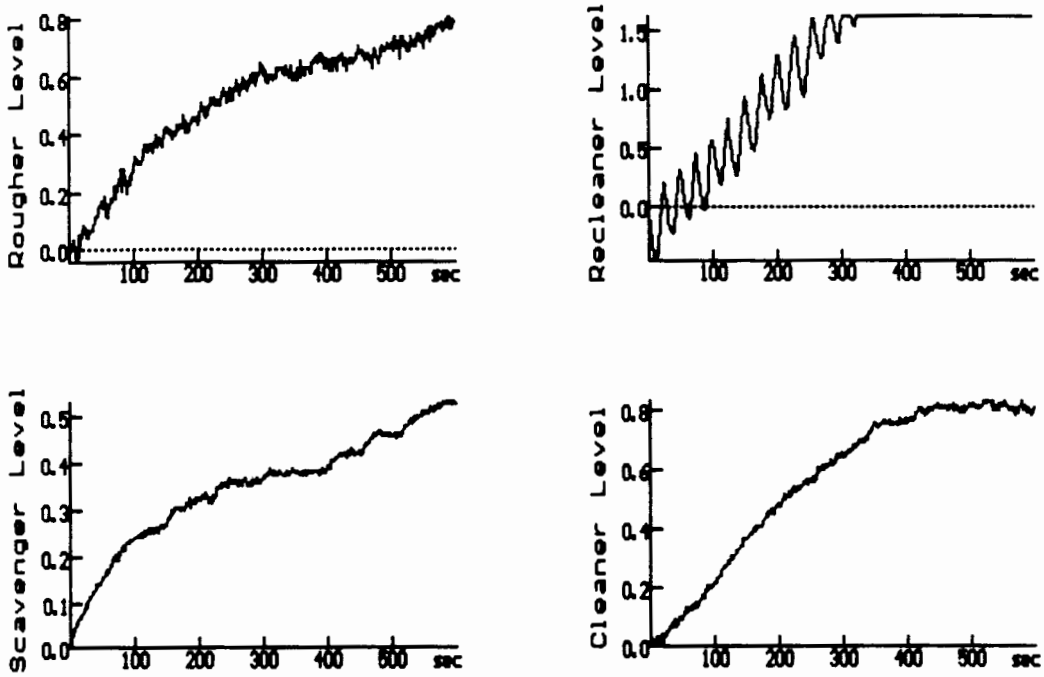

Flotation System Open-Loop Step Response
Test 1f) Rougher Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 2a) Scavenger Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 2b) Scavenger Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

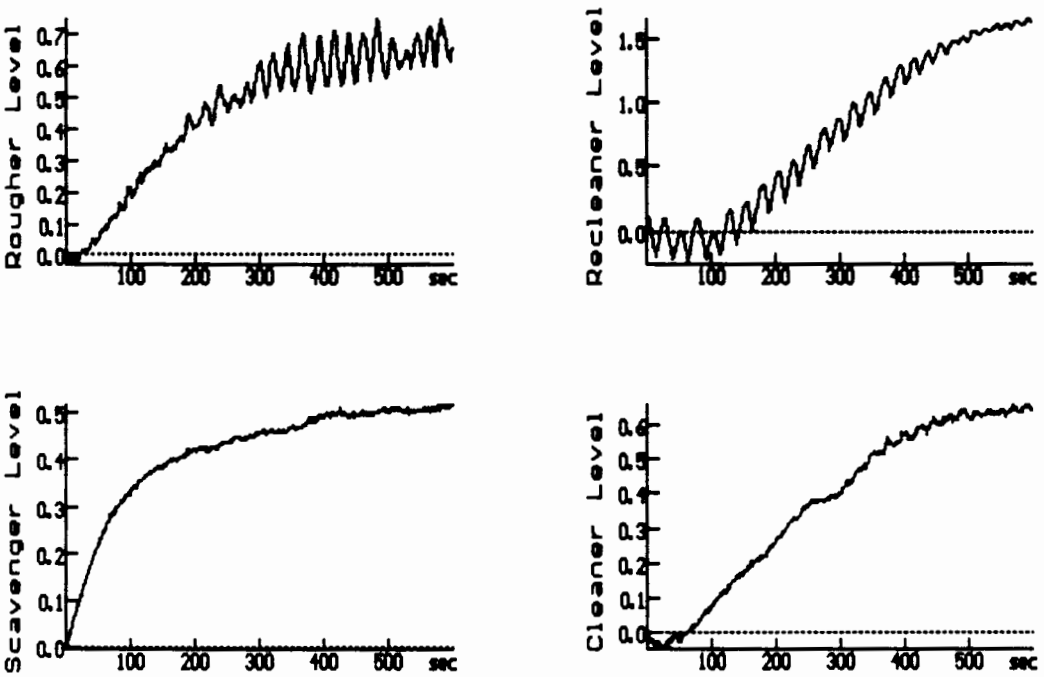
Flotation System Open-Loop Step Response

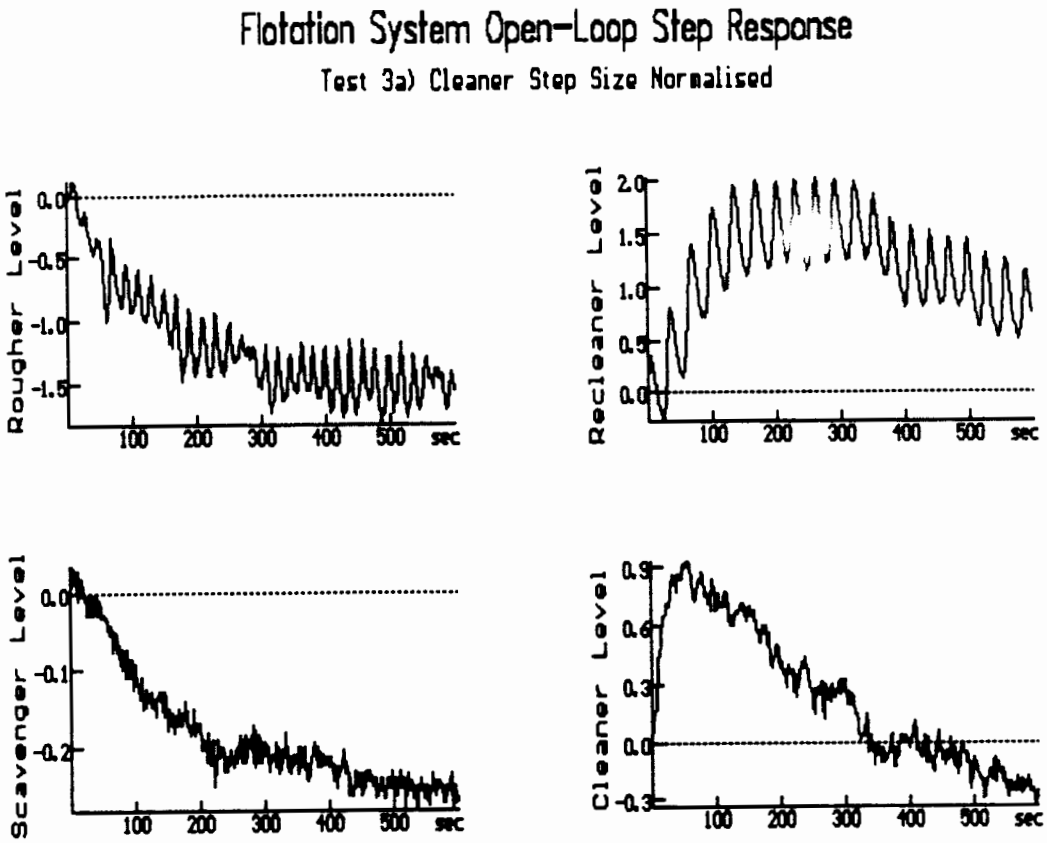
Test 2c) Scavenger Step Size Normalised

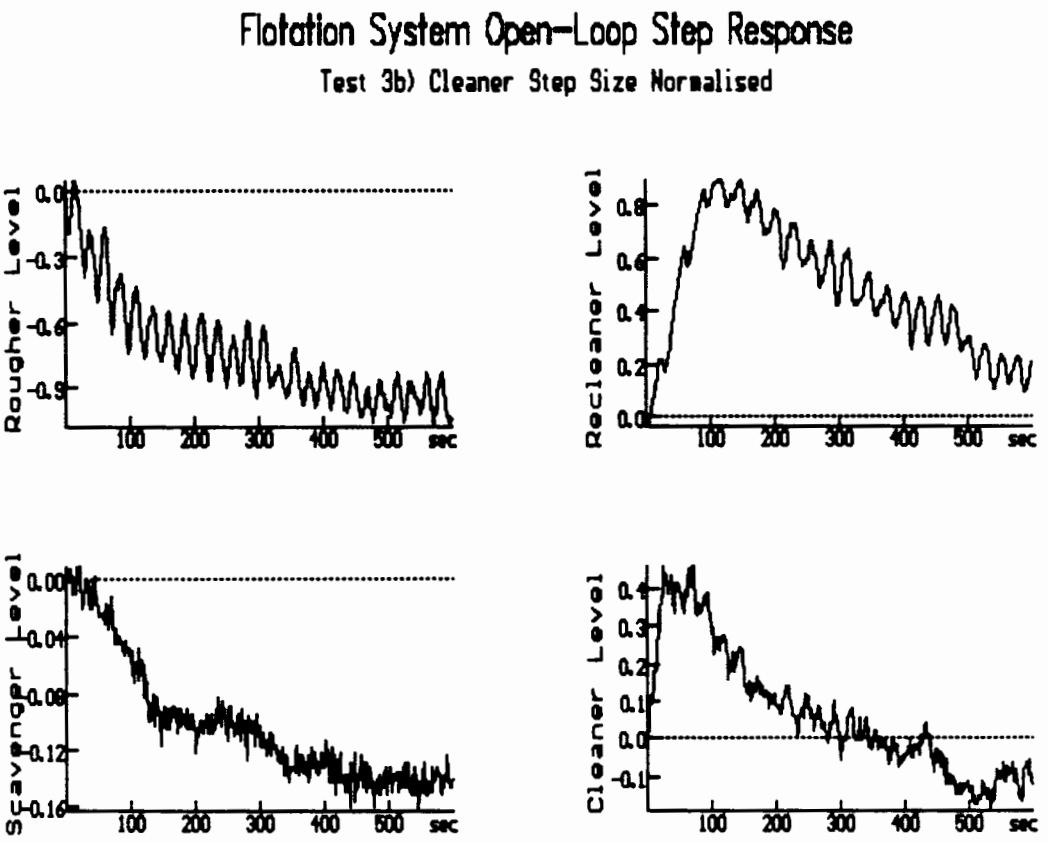


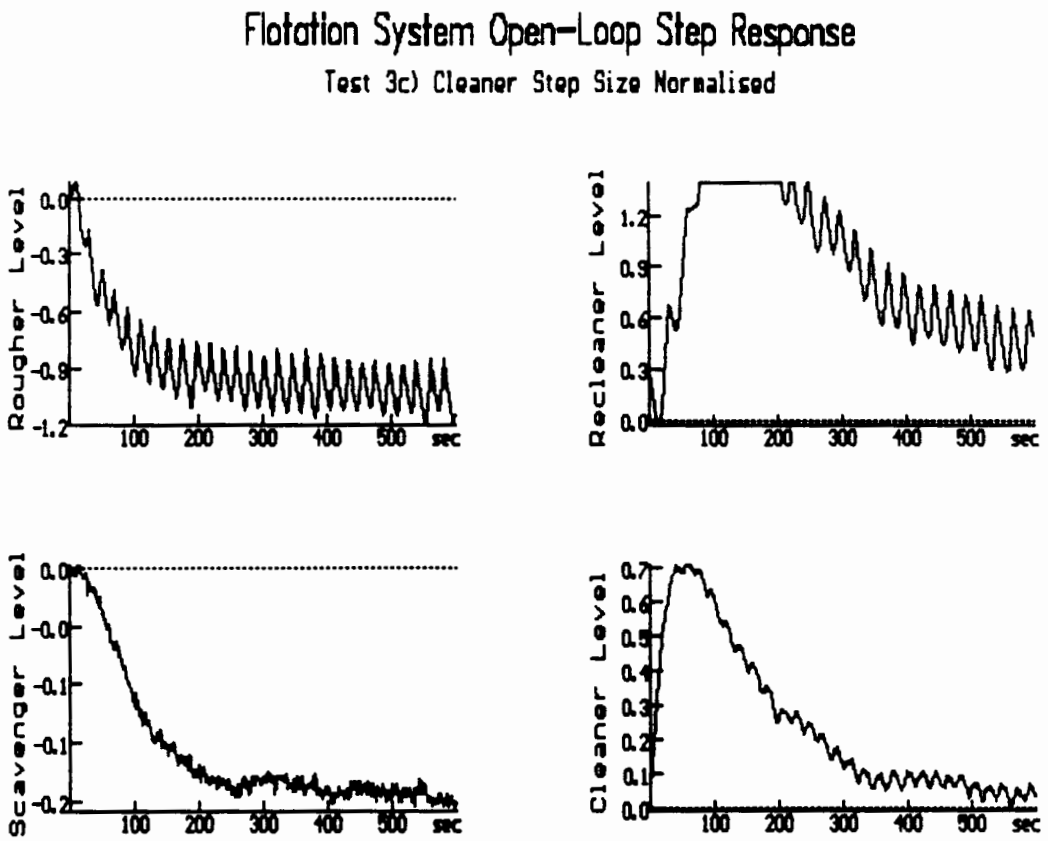
Flotation System Open-Loop Step Response

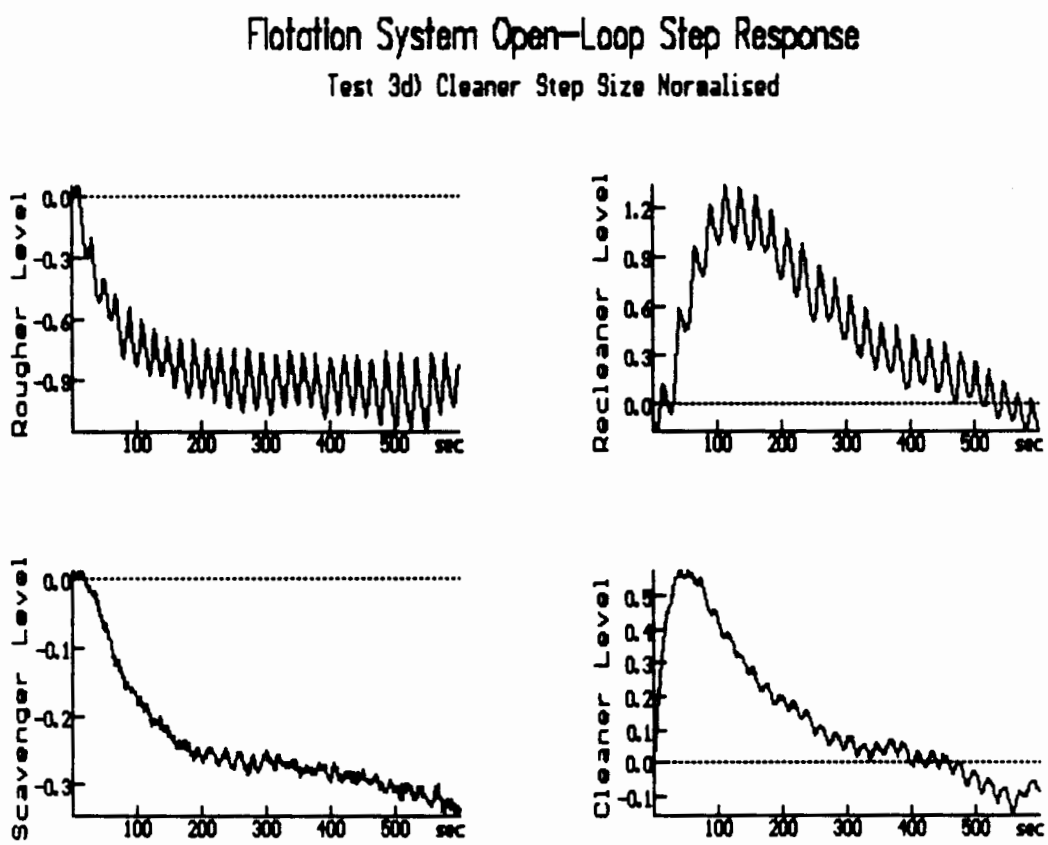
Test 2d) Scavenger Step Size Normalised

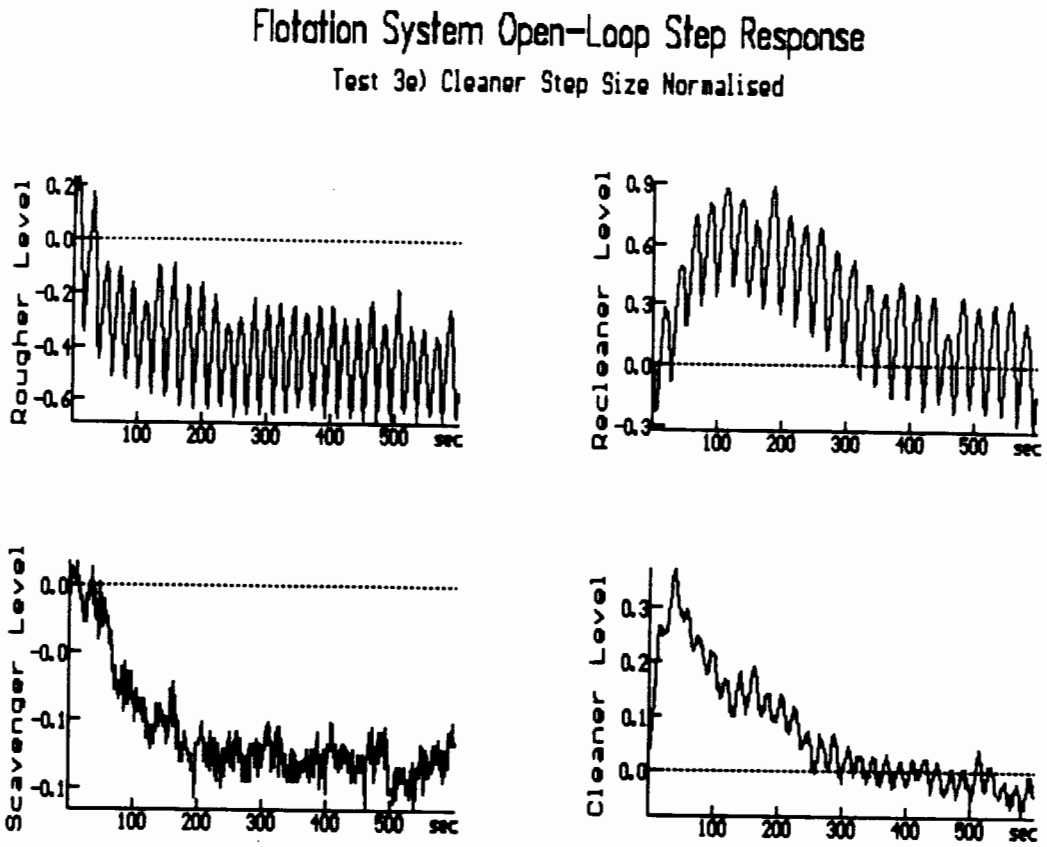


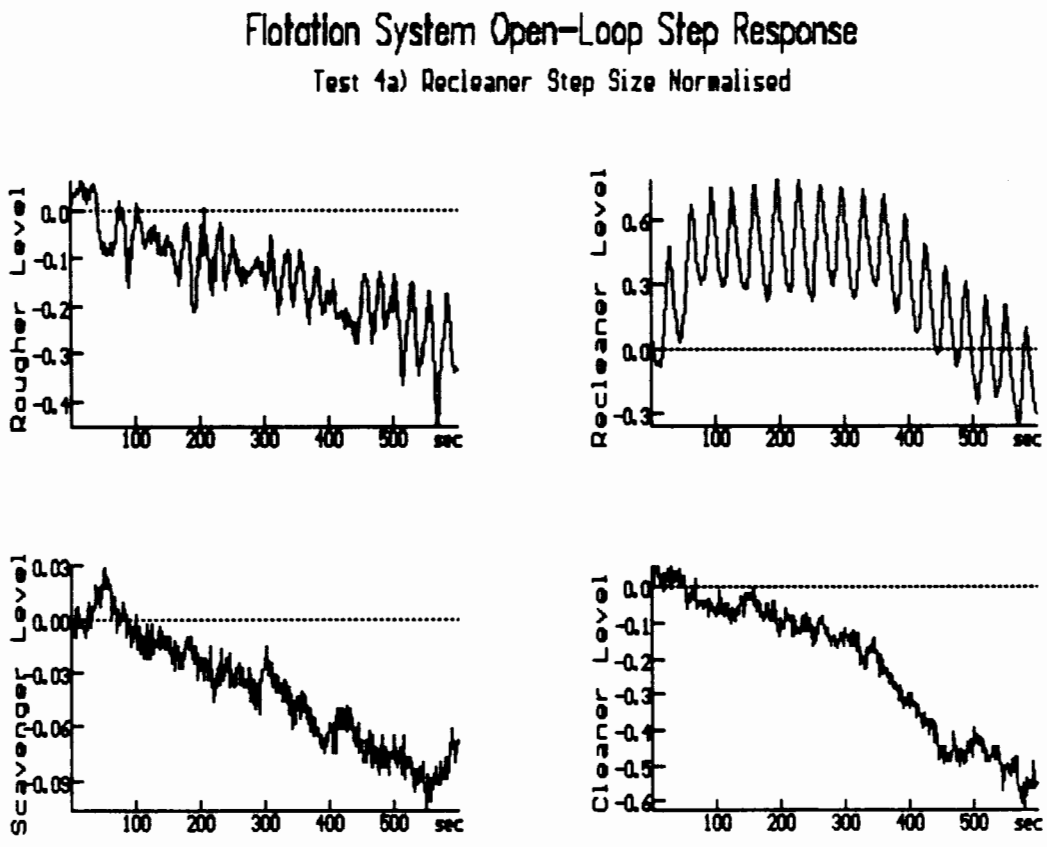

Flotation System Open-Loop Step Response
Test 3a) Cleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

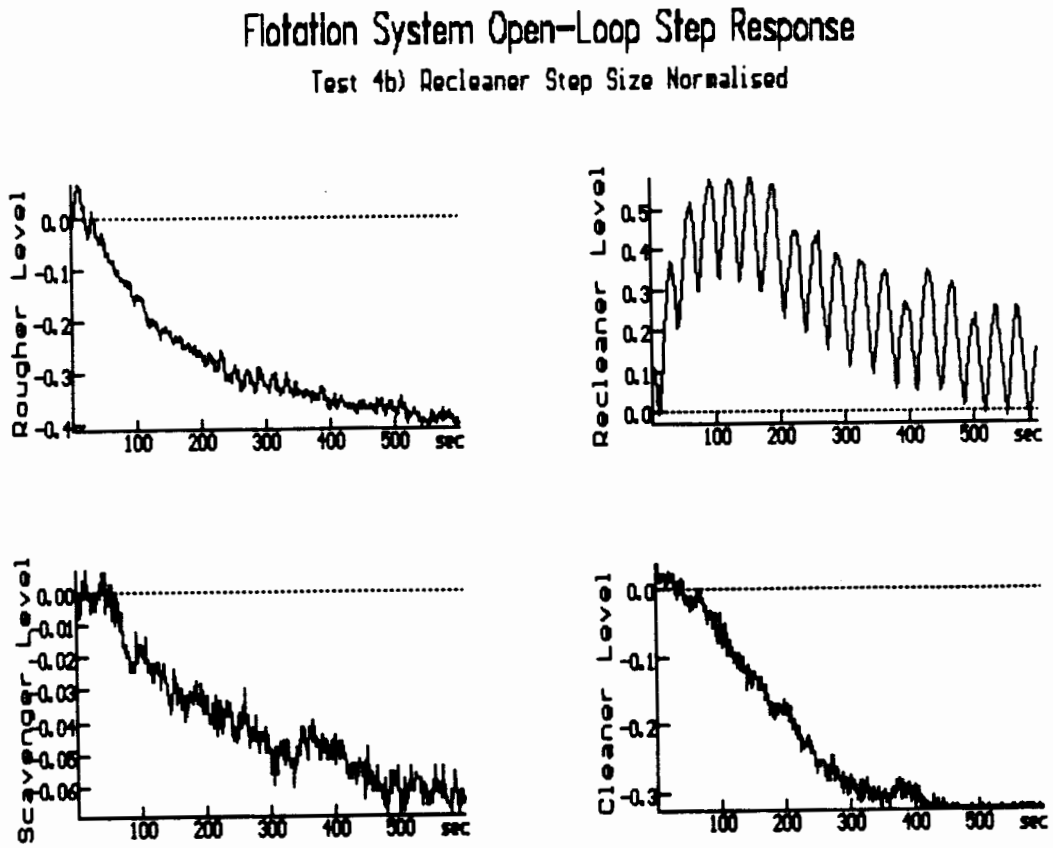

Flotation System Open-Loop Step Response
Test 3b) Cleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

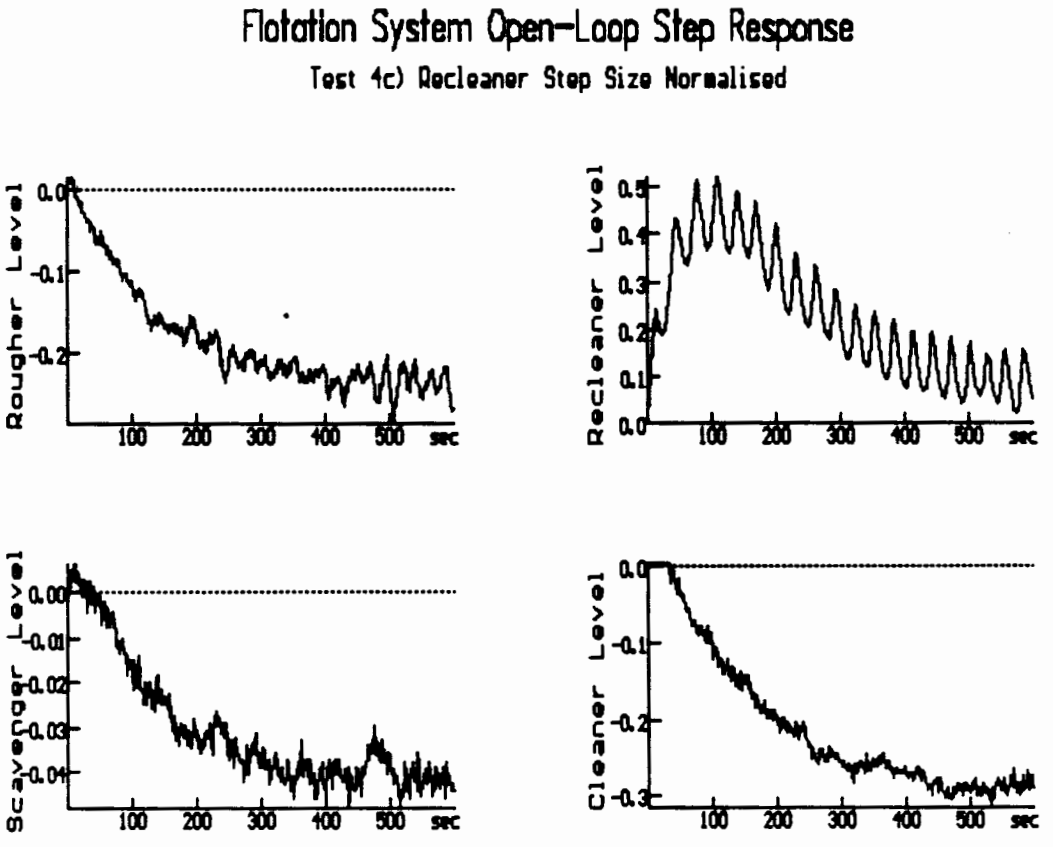

Flotation System Open-Loop Step Response
Test 3c) Cleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

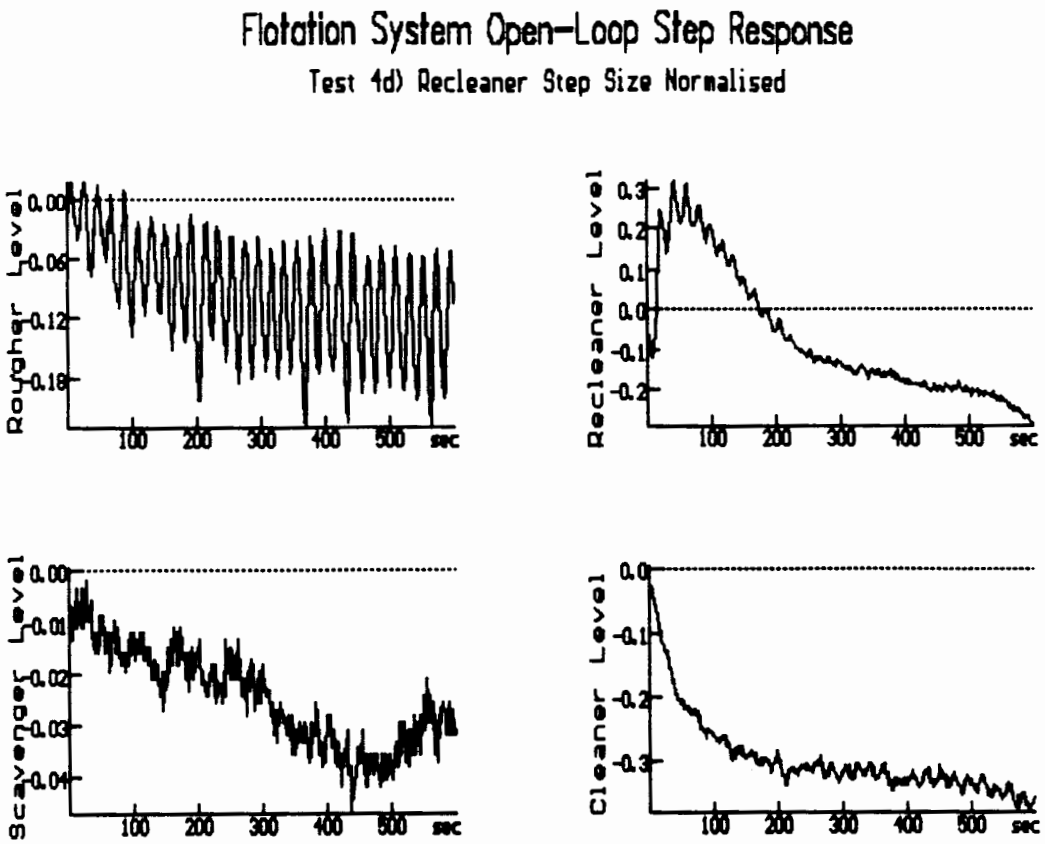

Flotation System Open-Loop Step Response
Test 3d) Cleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

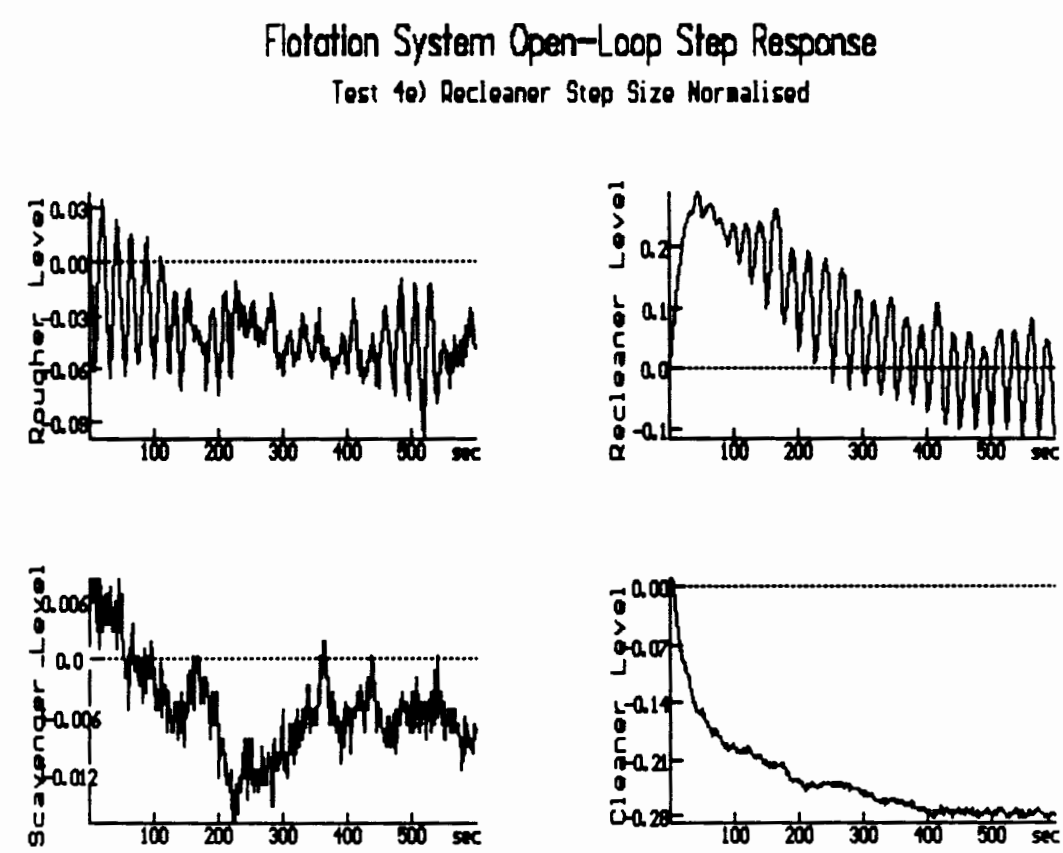

Flotation System Open-Loop Step Response
Test 3e) Cleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 4a) Recleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 4b) Recleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 4c) Recleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 4d) Recleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec


Flotation System Open-Loop Step Response
Test 4e) Recleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

2.6.2.2 Selecting Responses

Table 2.2 lists the step responses, which best characterised the behaviour of the plant to the different step inputs. These responses are then used to fit transfer functions to them.

Valve stepped	Response used
Rougher	Test 1.c)
Scavenger	Test 2.d)
Cleaner	Test 3.d)
Recleaner	Test 4.b)'

'As the Cleaner level response of Test 4.b) was saturated, it was replaced with that of Test 4.d).

Table 2.2 Step tests used to fit transfer functions

2.6.2.3 Fitting Transfer Functions

The sinusoidal variations of Rougher and Recleaner levels were caused by the pumps cavitating as the Sump tanks were emptied. This behaviour is not included in the mathematical model.

The step responses of a single output y(s) is given by

$$y(s) = g(s)\, u(s) \qquad (2.4)$$

with $u(s) = s^{-1}$ corresponding to a step input.

The transfer functions g(s) could be divided into three basic types, summarised as follows:

$$g(s) = \frac{a}{s+b} e^{-\tau s} \qquad \text{(First order)} \qquad (2.5)$$

$$g(s) = \frac{a}{s+bs+c} e^{-\tau s} \qquad \text{(Second order, no zero)} \qquad (2.6)$$

$$g(s) = \frac{as+b}{s+cs+d} e^{-\tau s} \qquad \text{(Second order with zero),} \qquad (2.7)$$

where τ corresponds to a pure time delay in seconds and s is in rad/sec.

To fit a particular function to a response, the initial coefficients are selected by trial and error. Once a reasonable correspondence is achieved, the coefficients are optimised with a nonlinear parameter adjustment technique [see NELM in the *Software Reference*].

2.6.3 FLOTATION RIG TRANSFER FUNCTION

The above method is applied and the resulting transfer function modelling the flotation rig are found to be:

(units of s are in rad/sec)

$$g_{1,1}(s) = \frac{1.40E\text{-}2\ s + 6.83E\text{-}5}{s^2 + 3.95E\text{-}2\ s + 5.91E\text{-}4} \qquad (2.6a)$$

$$g_{1,2}(s) = \frac{1.53E\text{-}4}{s^2 + 4.07E\text{-}2\ s + 2.20E\text{-}4} \qquad (2.6b)$$

$$g_{1,3}(s) = \frac{-2.00E\text{-}2}{s + 2.08E\text{-}2} e^{-10s} \qquad (2.6c)$$

$$g_{1,4}(s) = \frac{-3.02E-3}{s + 7.58E-3} e^{-20s} \qquad (2.6d)$$

$$g_{2,1}(s) = \frac{-1.22E-2}{s + 1.87E-2} \qquad (2.6e)$$

$$g_{2,2}(s) = \frac{5.97E-3}{s + 1.19E-2} \qquad (2.6f)$$

$$g_{2,3}(s) = \frac{-4.05E-3}{s + 1.37E-2} e^{-20s} \qquad (2.6g)$$

$$g_{2,4}(s) = \frac{-4.95E-4}{s + 8.13E-3} e^{-40s} \qquad (2.6h)$$

$$g_{3,1}(s) = \frac{1.66E-2\ s + 9.69E-5}{s^2 + 3.40E-2\ s + 3.81E-4} e^{-10s} \qquad (2.6i)$$

$$g_{3,2}(s) = \frac{1.27E-4}{s^2 + 4.33E-2\ s + 1.61E-4} e^{-30s} \qquad (2.6j)$$

$$g_{3,3}(s) = \frac{6.45E-2\ s - 1.31E-5}{s^2 + 9.66E-2\ s + 6.47E-4} \qquad (2.6k)$$

$$g_{3,4}(s) = \frac{-7.10E-3}{s + 2.16E-2} \qquad (2.6l)$$

$$g_{4,1}(s) = \frac{1.44E-2\ s + 7.80E-5}{s^2 + 2.50E-2\ s + 1.94E-4} e^{-40s} \qquad (2.6m)$$

$$g_{4,2}(s) = \frac{2.95E-4}{s^2 + 3.79E-2\ s + 1.57E-4} e^{-100s} \qquad (2.6n)$$

$$g_{4,3}(s) = \frac{3.15E-2\ s + 2.24E-5}{s^2 + 1.92E-2\ s + 1.74E-4} e^{-20s} \qquad (2.6o)$$

$$g_{4,4}(s) = \frac{1.84E\text{-}2\ s + 1.14E\text{-}5}{s^2 + 3.08E\text{-}2\ s + 1.41E\text{-}4} \qquad (2.6p)$$

where $g_{ij}(s)$ are in units of [%Level / %Valve Setting].

The similarity of the simulated and the actual step responses can be seen in Figs. 2.3 - 2.6.

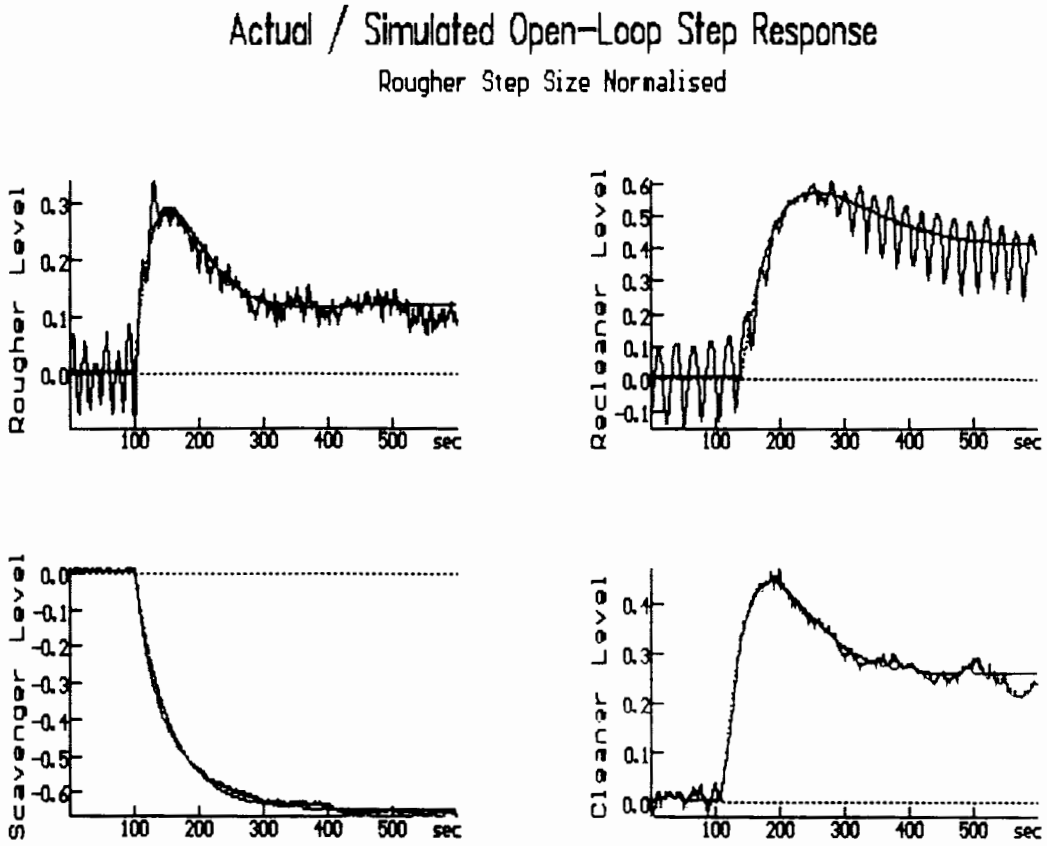

Actual / Simulated Open-Loop Step Response
Rougher Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

FIG. 2.3

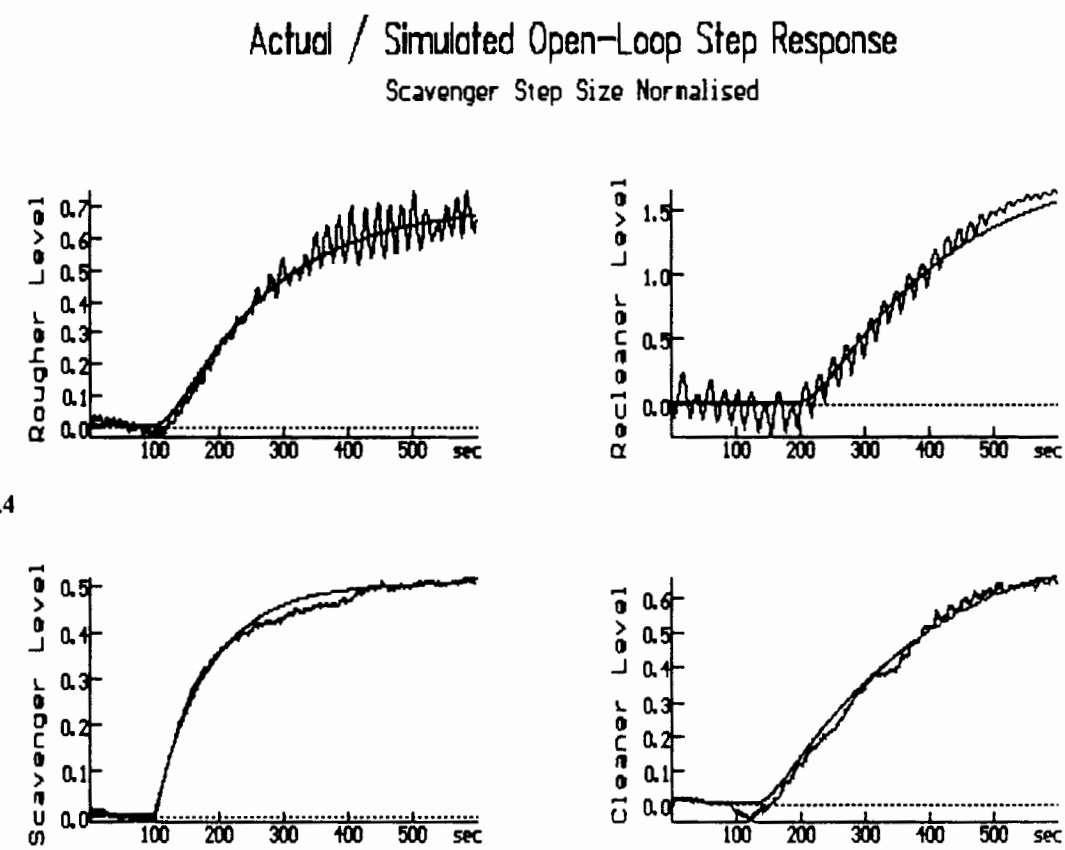

Actual / Simulated Open-Loop Step Response
Scavenger Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

FIG. 2.4

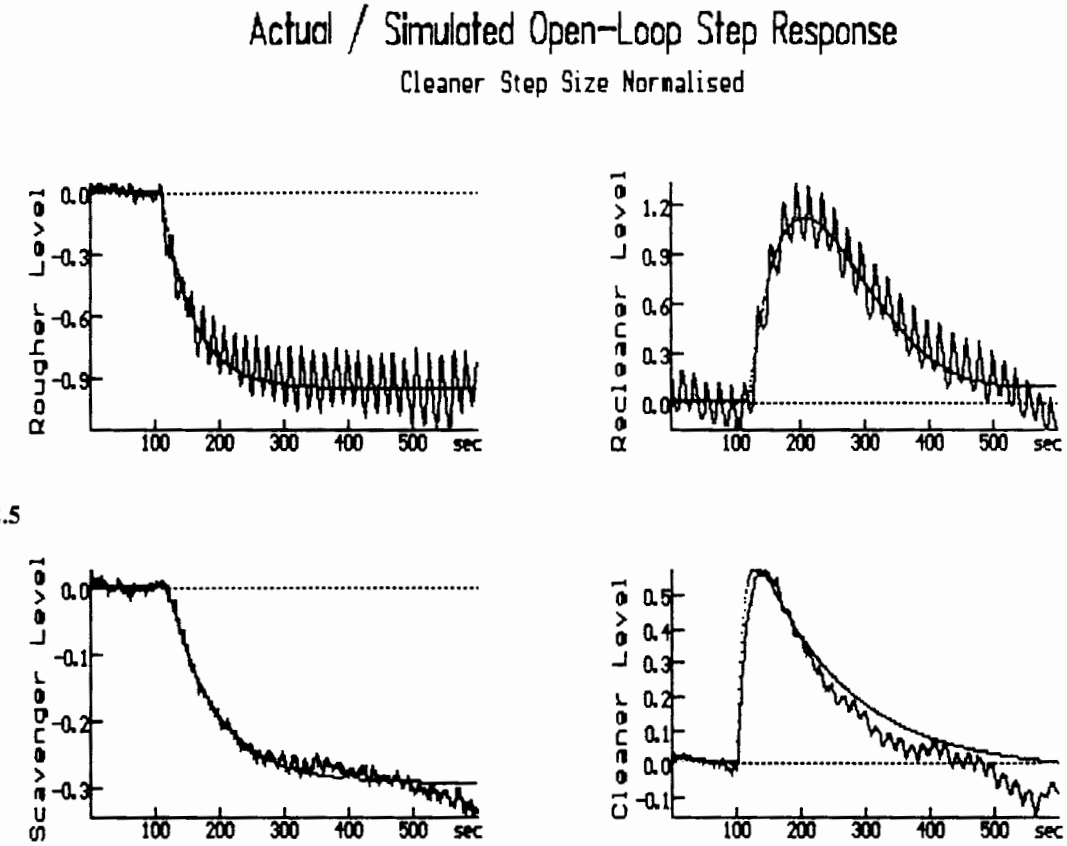

Actual / Simulated Open-Loop Step Response
Cleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

FIG. 2.5

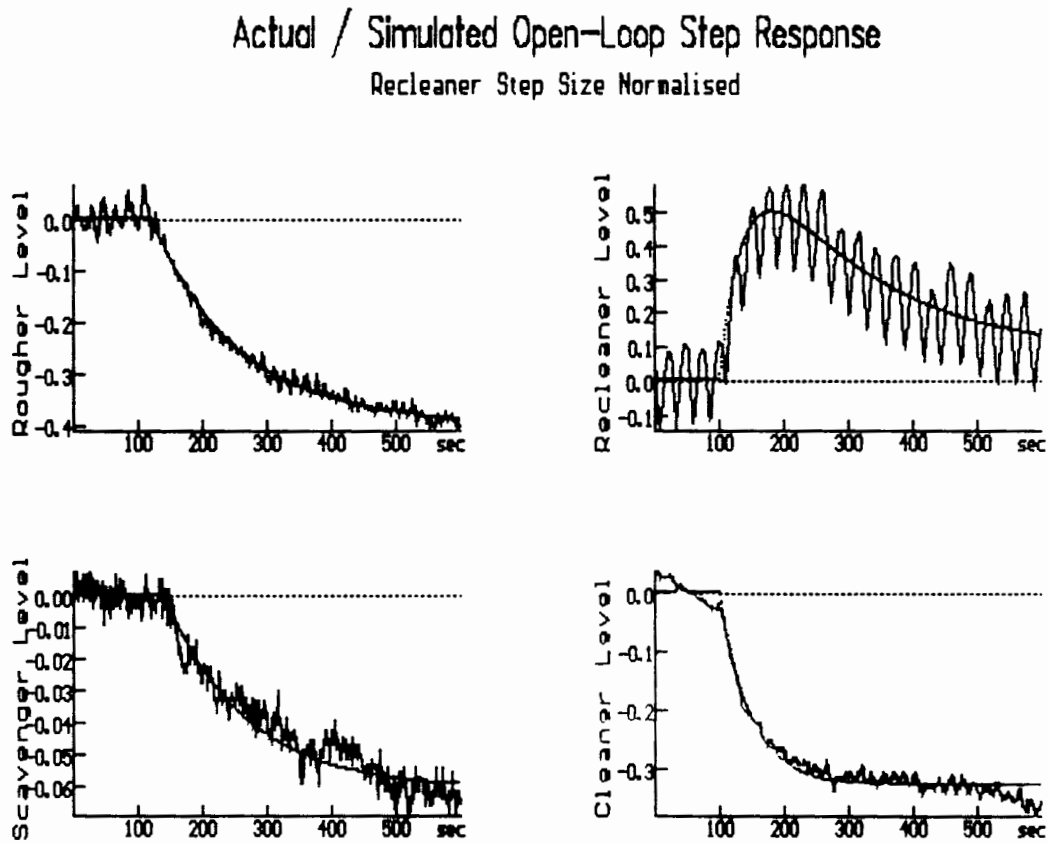

Actual / Simulated Open-Loop Step Response
Recleaner Step Size Normalised
Rougher Level
Recleaner Level
Scavenger Level
Cleaner Level
sec

FIG. 2.6

2.6.4 TWO-TANK SYSTEM MODEL

Instead of using the full flotation circuit as a practical example for the succeeding discussions on the Inverse Nyquist and the Pole Assignment techniques, a subsystem was used instead to simplify the analysis. The flotation system is of order 4, which will pose problems in the application of both methods, as the system can be considered to be of high order. Since any design procedure will become very cumbersome as the process order becomes large, it is sufficient to use a subsystem of the flotation rig to apply and compare the two techniques and only briefly mention the difficulties with large order systems.

The transfer function of the two-tank sub-system with two inputs and two outputs contains the Rougher and Scavenger tank model only and the valves of the other two tanks are fixed, i.e.

$$\mathbf{u}_3(s) = \mathbf{u}_4(s) = 0. \tag{2.7}$$

The new model will be given by

$$\mathbf{y}(s) = \mathbf{G}(s)\ \mathbf{u}(s) \tag{2.8}$$

where $\mathbf{y}$ and $\mathbf{u}$ are 1x2 vectors and $\mathbf{g}_{ij}(s)$ (for i=1,2 and j=1,2) are given by Eqs. (2.6).

2.7 CONCLUSION

This chapter has demonstrated how the transfer function of the flotation rig was established by step response analysis,

with special care taken to limit the inherent non-linearities.

This type of identification can be performed on most industrial systems and usually results in a satisfactory mathematical model, which can then be used to develop a control scheme.

3. THE INVERSE NYQUIST ARRAY TECHNIQUE

3.1 INTRODUCTION

The Inverse Nyquist Array (INA) design technique is discussed in this chapter. The general model of a multivariable plant is developed in the frequency domain. A short discussion on diagonal systems is given to demonstrate the fundamentals of the INA technique.

The concept of diagonal dominance is derived to apply the Nyquist stability criterion on multivariable systems. Details of achieving dominance are presented. Finally it is shown how singlevariable techniques are applied to design single-loop controllers to complete the control scheme.

3.2 MODELS FOR THE INVERSE NYQUIST ARRAY METHOD

3.2.1 SYSTEM MODEL

In the discussion on the INA technique a system description, similar to that of Rosenbrock(**3,12**), is used here. The model is described by the transfer function notation, because the INA method is applied in the frequency domain. If the system is given by state space equations, it has to be converted into the corresponding transfer function first [see Section 1.7.2].

In industrial applications, where the plant dynamics cannot easily be described by differentail equations, a black-box identification of the plant by step response analysis is necessary. This approach was used on the flotation rig [see Section 2.6] and led to a transfer function model, which could then directly be used for the INA technique.(2)

A block diagram of a linear, time-invariant feedback control system is shown in fig. 3.1, where the plant has *n* inputs and *n* outputs and is modelled by a *nxn* transfer function matrix **G**(s). The number of inputs must equal the number of outputs such that *n* individual feedback loops can be constructed as noted in section 1.5.1. The *n* individual feedback paths are joined to simplify the block diagram in Fig. 3.1.

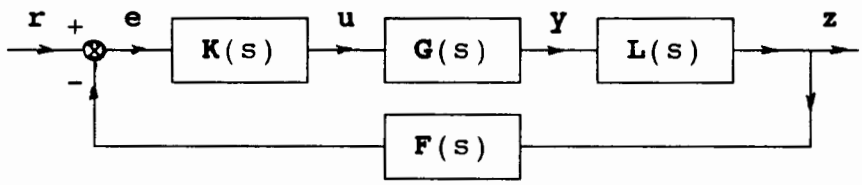


Fig. 3.1 Multivariable control system representation

The inputs and outputs of the plant model **G**(s) are represented by vectors **u** and **y** respectively. The vector **r** contains the required setpoint values, **e** the comparator output and vector **z** is the system output. All the above vectors have *n* elements.

The matrices **K**(s) and **L**(s) are termed the pre- and post-controllers respectively. The feedback matrix is denoted by **F**(s). All the matrices used have dimension *nxn*.

3.2.2 MODEL RESTRICTIONS

Various restrictions are imposed on the system matrices thus simplifying analysis of the control scheme.

The plant model **G**(s) should be strictly proper, i.e.

$$\lim_{s \to \infty} g_{ij}(s) = 0 \qquad (3.1)$$

and must be of least order, then

$$|\mathbf{G}(s)| \neq 0 \qquad (3.2)$$

The post-controller matrix **L**(s) should be diagonal, i.e with off-diagonal elements = 0, because for most applications the system outputs are to be controlled independently from each other. In addition, the matrix **L**(s) is in most cases kept independent of s, such that the system output remains the controlled variable.

Similarly, the feedback matrix **F**(s) is kept diagonal and independent of s, such that a direct relationship exists between the setpoint values and output variables. In the analysis of the closed-loop system, the open-loop gains of the individual loops are inserted into the diagonal elements f_i.

The system integrity can also be simulated by setting some or all the f_i=0. It is then relatively easy to predict the system behaviour under fault conditions.

By restricting both matrices **L**(s) and **F**(s) to be diagonal and constant it can be shown that the constants l_i of **L**(s) can be moved into the matrix **K**(s) by multiplying the ith column of **K**(s) by l_i. This would eliminate the need for a matrix **L**(s) with its restrictions in the control system

scheme. But to keep the design of the control system more general it is nevertheless included.

The restrictions governing the matrix $\mathbf{K}(s)$ are the following. Because of the eventual inclusion of $\mathbf{K}(s)$ in the plant by electronic (analog or digital) means the elements $k_{ij}(s)$ must be realizable, i.e. proper and stable. If the elements are not strictly proper, care must be taken that they do not impede the system performance by amplifying high frequency noise.

3.2.3 SYSTEM EQUATIONS

The open-loop transfer function of the above multivariable system, relating the output vector $\mathbf{z}$ to the input setpoint vector $\mathbf{r}$, is denoted by $\mathbf{Q}$, then

$$\mathbf{Q} = \mathbf{LGK} \tag{3.3}$$

Using $\mathbf{H}$ to denote the corresponding closed loop system then

$$\mathbf{H} = (\mathbf{I}_n + \mathbf{QF})^{-1}\,\mathbf{Q} \tag{3.4}$$

and the inverse relationship follows with

$$\mathbf{H}^{-1} = \mathbf{Q}^{-1} + \mathbf{F} \tag{3.5}$$

if $\mathbf{Q}^{-1}$ exists.

From the above relationship it follows that if the system analysis is directed toward using $\mathbf{Q}^{-1}$ instead of $\mathbf{Q}$ then the transformation to the closed loop system is a simple one, namely an addition of $\mathbf{F}$.

3.2.4 CONTROLLABILITY

The restriction that $\mathbf{Q}^{-1}$ must exist implies that the plant model transfer function matrix $\mathbf{G}(s)$ must be controllable and the choice of both the matrices $\mathbf{K}(s)$ and $\mathbf{L}(s)$ are not allowed to impede the controllability of $\mathbf{Q}(s)$, i.e.

$$|\mathbf{G}(s)| \neq 0$$

$$|\mathbf{K}(s)| \neq 0$$

and

$$|\mathbf{L}(s)| \neq 0 \qquad (3.6)$$

Although this constraint manifests itself from the mathematical derivation, it is important for practical reasons. If for example a specific choice of $\mathbf{K}(s)$ with $|\mathbf{K}(s)| = 0$ is used then the individual loops of the multivariable system are no longer linearly independent and the individual system outputs cannot be controlled independently any more. Therefore the system order must be reduced before a control scheme can be developed. It is therefore obvious to keep both $|\mathbf{K}(s)|$ and $|\mathbf{L}(s)| \neq 0$.

The plant model $\mathbf{G}(s)$ should have least order from outset and controllable [see Eqn.3.2], thus eliminating the problem of $|\mathbf{G}(s)| = 0$.

3.3 DIAGONAL SYSTEMS

The basic aim of the INA method is to design the matrix $\mathbf{K}$ and $\mathbf{L}$ such that the matrix $\mathbf{Q}$ becomes diagonal. This process is referred to as decoupling of the plant $\mathbf{G}$. Although matrix $\mathbf{L}$ could be used to decouple the plant by column operations on $\mathbf{G}(s)$ in general, this approach will not be applied in

this discussion, because of the restrictions imposed on **L** previously.

If the open-loop transfer matrix **Q** is diagonal then the multivariable system can be split into *n* independent singlevariable feedback systems. Controllers can then be developed by singlevariable techniques and inserted into the individual loops to complete the control scheme.

In addition, well established singlevariable methods can be applied to diagonal systems to determine stability of the various loops of **H**. If the individual loops are stable, it can be stated without restriction that the multivariable system will be stable.

This discussion has shown that multivariable system design can be greatly simplified if the system is made diagonal, because then only singlevariable techniques need to be applied. This observation is the foundation on which the INA method is based. The multivariable system is transformed into a number of singlevariable systems.

3.4 THE CONCEPT OF DIAGONAL DOMINANCE

From the discussion above it is evident that a matrix **K** such that

$$\mathbf{K}(s) = \mathbf{G}^{-1}(s) \tag{3.7}$$

would make the system diagonal. But due to the restrictions imposed on **K** in section 3.2.2, the matrix of Eqn.(3.7) will seldom be realizable. To decouple a plant completely is not possible for most applications and a more generalised approach has to be developed.

The concept of diagonal dominance was introduced by Rosenbrock(3) to relax the requirement that the system must be completely decoupled, before singlevariable techniques can be applied.

The mathematical definition of diagonal dominance is as follows:

A matrix $\mathbf{Q}(s)$ is diagonally row dominant if and only if

$$|q_{ii}(s)| - \sum_{\substack{j=1 \\ j \neq i}}^{n} |q_{ij}(s)| > 0 \text{ for } i=1,2..n \qquad (3.8)$$

and column dominant if and only if

$$|q_{jj}(s)| - \sum_{\substack{i=1 \\ i \neq j}}^{n} |q_{ij}(s)| > 0 \text{ for } j=1,2..n. \qquad (3.9)$$

A matrix $\mathbf{Q}$ is diagonally dominant if it is either row or column dominant for all relevant s values.

The concept of dominance can be regarded as a measure of coupling between different inputs and outputs of a system. A diagonally dominant multivariable system has the property, that the interaction from the off-diagonal elements will not impair the stability of the diagonal elements and the stability of the multivariable system can be determined by the application of the extended Nyquist stability criterion on the diagonal elements.

3.4.1 GRAPHICAL INTERPRETATION

Nyquist diagrams are drawn for all *nxn* elements $q_{ij}(s)$ of matrix $\mathbf{Q}(s)$ by setting $s=jw$ and $0 \leq w \leq \infty$. The real parts of s can be excluded to determine stability because $\mathbf{Q}(s)$ is

assumed to be strictly proper.(3) In addition, it is sufficient for w to range from zero to w_{max}, where w_{max} is about 10 times the fastest pole of a plant. The plot so obtained is referred to as a Nyquist Array diagram.

3.4.2 GERSHGORIN BANDS

The mathematical test for diagonal dominance of a matrix **Q** (Eqs. (3.8) & (3.9)) can easily be performed by the following graphical construction on the Nyquist Array diagram.

To check for row diagonal dominance, a circle of radius

$$d_i(s) = \sum_{\substack{j=1 \\ j \neq i}}^{n} |q_{ij}(s)| \qquad (3.10)$$

with centre at $q_{ii}(s)$ for each frequency w is drawn for i=1,2..n.

If w moves from 0 to w_{max} in sufficiently small steps, an envelope, tangential to the circles can be drawn for each diagonal element. These regions are then referred to as the Gershgorin bands.

If the bands do not include the origins (0,j0), then **Q**(s) is row dominant.

A similar construction can be made to determine column dominance. In that case the radii of the circles are calculated from:

$$d'_j(s) = \sum_{\substack{i=1 \\ i \neq j}}^{n} |q_{ij}(s)| \qquad (3.11)$$

At any operating point of $\mathbf{Q}(s)$ the actual Nyquist diagram will lie within that band. If $\mathbf{Q}(s)$ is dominant then stability, gain and phase margins can be determined from these 'fuzzy' Nyquist plots by similar methods as applied to singlevariable systems.

The bands can be constructed for $\mathbf{Q}^{-1}(s)$, by substituting ith,jth element of $\mathbf{Q}^{-1}(s)$, i.e. $\hat{q}_{ij}(s)$ for $q_{ij}(s)$ in Eqs. (3.10) & (3.11).

3.4.3 EXTENDED NYQUIST STABILITY CRITERION

3.4.3.1 Direct Nyquist

If the matrix $\mathbf{Q}(s)$ is diagonally dominant, the closed-loop system with feedback matrix $\mathbf{F}$ is stable if and only if the bands encircle and do not enclose the critical point $(-f_i^{-1}, j0)$, N_i times with

$$N_i = -p_o \tag{3.12}$$

where p_o is the number of unstable open-loop poles.

If the plant is open-loop stable then p_o is zero. The closed-loop system is then stable if the band passes the critical point on the right.

3.4.3.2 Inverse Nyquist

If the matrix $\mathbf{Q}^{-1}(s)$ is diagonally dominant, then the closed-loop system is stable if and only if the bands encircle and do not enclose the critical points $(-f_i, j0)$, N_i times and the origin N_j times such that

$$N_i + N_j = -p_o \qquad (3.13)$$

where po is the number of unstable open-loop poles.

If p_o is zero, then the multivariable system will be stable if the bands encircle the critical point and the origin the same number of times. In most situations it is sufficient if the bands do not intersect the real axis to the left of the origin and to the right of the critical points.

The critical points $(\mathbf{f}_i^{-1}, j0)$ or $(\mathbf{f}_i, j0)$ must lie to the left of the origin, because the theory is applied to negative feedback system only.

3.4.4 GAIN MARGINS

The margins of stability of the loop gains $\mathbf{f}_i$ can be easily determined from either the Nyquist or Inverse Nyquist Array diagram by the application of the above stability criteria.

The regions of $\mathbf{f}_i$ in which the closed-loop systems will remain stable is referred to as the gain space, which can be drawn for systems of order 2, i.e. two inputs and two outputs.(3) This area can then be used to check system integrity and it gives an indication of the maximum allowable variation in open-loop gain.

If the loop gains remain in these regions, the closed-loop system is stable. If however, the gains move out of these regions, it does not mean that the system will definitely be unstable. A tighter estimation of the gain margins is possible with the application of Ostrowski bands, which are discussed in the next section.

3.4.5 OSTROWSKI BANDS

The circles for the Ostrowski bands of $\mathbf{Q}^{-1}(s)$ are constructed in a similar fashion as those for the Gershgorin bands, but the radii of the circles for row dominance are now:

$$r_i = \varnothing_i(s) d_i(s)$$

where

$$\varnothing_i(s) = \max_j \frac{d_j(s)}{|\mathbf{f}_j + \hat{\mathbf{q}}_{jj}(s)|} \qquad (3.14)$$

and d(s) refer to the radii of the Gershgorin circles for $\mathbf{Q}^{-1}(s)$ given by eqn. (3.10).

In the case of column dominance d(s) is replaced by d'(s) and found from eqn. (3.11).

If $\mathbf{Q}^{-1}(s)$ is dominant then the Ostrowski bands are found to be narrower than the Gershgorin counterparts and allow tighter design of the gains $\mathbf{f}_i$. It is important to note, that a change in the gain of one loop will affect the radii of the other loops and it is therefore difficult to establish the exact gain space for which the closed-loop system $\mathbf{H}^{-1}(s)$ will be stable. The region so determined will usually be larger than that for Gershgorin bands.

In practice however, the design of a dominant and stable system is done with the aid of Gershgorin bands mainly, although this will result in a conservative choice of feedback gains. Having obtained a suitable feedback matrix **F**, the Ostrowski circles can be plotted as a final check for stability and dominance.

3.5 ACHIEVING DIAGONAL DOMINANCE

The previous sections have shown that the Nyquist stability criterion can be applied to diagonally dominant systems. The first step of the INA technique is thus to design **K** such that $\mathbf{Q}^{-1}$ is diagonally dominant, before stability is considered.

The next Sections will demonstrate different methods to achieve this.

3.5.1 DIRECT AND PSEUDO-DIAGONALISATION

The following two methods will result in a real constant **K** matrix. The one approach directly diagonalises **G**(s) at zero frequency by setting $\mathbf{K} = \mathbf{G}^{-1}(0)$ and the other is referred to as pseudo-diagonalisation(**3,13**). These two methods are not discussed in detail here although they have been applied in the past for simple systems.

Attention is focused instead on dynamic pre-compensators as they can easily be implemented with todays microcomputer technology and they are generally more successful at making $\mathbf{Q}^{-1}(s)$ diagonally dominant over the whole frequency range.

3.5.2 ELEMENTARY OPERATIONS

By operating on the rows or columns of $\mathbf{Q}^{-1}(s)$ by elementary operations, similar to Gauss elimination, it is possible to achieve dominance in most cases, if **K**(s) is allowed to be sufficiently complicated. At this point it is important to recall that a few restrictions are made to the form that K(s) may take [see Section 3.2.2], because the resulting

pre-compensator will have to be implemented in the final control scheme.

Other considerations such as non-minumum phase and controllability are discussed by Porter(**14**).

As the design involves a number of consecutive steps the pre-compensator can be written as

$$\mathbf{K}(s)=\mathbf{K}_n(s)\ldots\mathbf{K}_2(s)\mathbf{K}_1(s) \qquad (3.15)$$

where $\mathbf{K}_i(s)$ correspond to the ith elementary operation.

Each elementary operation $\mathbf{K}_i(s)$ can then be implemented in the control scheme individually, which is less complicated than implementing $\mathbf{K}(s)$ as a whole.

3.5.2.1 Renumbering the inputs

The first step involves renumbering of the inputs of $\mathbf{G}(s)$ with the aid of a permutation matrix $\mathbf{K}(s)$. This allows the 'dominant' elements of each column of $\mathbf{Q}^{-1}(s)$ to be moved into diagonal positions by swopping its rows.

The ith row can be swopped with jth row of $\mathbf{Q}^{-1}(s)$ by setting $\mathbf{K}(s)=\mathbf{I}$ and swopping the ith column with jth column of $\mathbf{K}$, then

$$\begin{array}{c} \text{Column} \\ \\ \\ \\ \\ \\ \\ \\ \mathbf{K}(s) = \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & & i & & j & & n & \text{Row} \end{array} \\ \left[\begin{array}{ccccccccc} 1 & 0 & \ldots & 0 & \ldots & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & \ldots & 0 & \ldots & 0 & 2 \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & \\ 0 & 0 & \ldots & 0 & \ldots & 1 & \ldots & 0 & i \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & \\ 0 & 0 & \ldots & 1 & \ldots & 0 & \ldots & 0 & j \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & \\ 0 & 0 & \ldots & 0 & \ldots & 0 & \ldots & 1 & n \end{array}\right] \end{array} \qquad (3.16)$$

3.5.2.2 Row operations

The second type of operation involves adding a multiple of one row to another row of $\mathbf{Q}^{-1}(s)$ to reduce the off-diagonal elements, which is related to Gauss elimination.

The operation

$$Row_i = Row_i + \alpha(s)Row_j \tag{3.17}$$

where $\alpha(s)$ is the multiplication function, is performed by setting $\mathbf{K}(s)=\mathbf{I}$ and then subtracting $\alpha(s)Col_i$ from Col_j, then

$$\begin{array}{l} \text{Column} \quad 1 \quad 2 \quad\quad i \quad\quad j \quad\quad n \quad \text{Row} \\ \mathbf{K}(s) = \left[\begin{array}{ccccccccc} 1 & 0 & \ldots & 0 & \ldots & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & \ldots & 0 & \ldots & 0 & 2 \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & \\ 0 & 0 & \ldots & 1 & \ldots & -\alpha(s) & \ldots & 0 & i \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & \\ 0 & 0 & \ldots & 0 & \ldots & 1 & \ldots & 0 & j \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & \\ 0 & 0 & \ldots & 0 & \ldots & 0 & \ldots & 1 & n \end{array}\right]. \end{array} \tag{3.18}$$

The negative sign arises because the operation are performed on $\mathbf{Q}^{-1}(s)$ and not $\mathbf{Q}(s)$.

The rational function $\alpha(s)$ must have all its poles in the open left half plane and must be proper. If the multiplier function is not strictly proper then care must be taken that the gain at high frequency will not amplify the noise of the system excessively. In practice it is sufficient that

$$\lim_{s \to \infty} |\alpha(s)| \leq 10 \ . \tag{3.19}$$

If element $\hat{q}_{ik}(s)$ is to be eliminate by using Row_j then the multiplier becomes

$$\alpha(s) = \frac{\hat{q}_{ik}(s)}{\hat{q}_{kj}(s)} \qquad (3.20)$$

After performing the row operation $\hat{q}_{ik}(s) = 0$.

In practice it is very difficult to obtain $\alpha(s)$ because the elements $\hat{q}_{ik}(s)$ and $\hat{q}_{ik}(s)$ are not known in terms of s. A symbolic derivation of $\mathbf{Q}^{-1}(s)$ from $\mathbf{G}(s)$ in terms of s is extremely burdensome and also unnecessary because the resulting $\alpha(s)$ will most probably not conform to the restrictions that are imposed on $\mathbf{K}(s)$.

In most practical applications an approximation to $\alpha(s)$ is used, which will eliminate the reduced element sufficiently.

The row operations are thus a matter of trial and error. The INA design involves choosing an appropriate row for the elimination and finding an adequate approximation of the multiplier function. It is further a matter of skill to determine in which order the off-diagonal elements should be reduced. It is thus necessary to have a computer-aided control system design (CACSD) package to apply the INA technique.(**5**) The structure of such a programme is discussed in the *Software Reference*.

3.6 SINGLE-LOOP CONTROLLER DESIGN

After having achieved diagonal dominance, a diagonal matrix $\mathbf{K}_p(s)$ can be developed to complete the feedback control scheme. The elements of $\mathbf{K}_p(s)$ control the individual loops of the decoupled system and they can be designed by singlevariable techniques.

The Inverse Nyquist diagram of the diagonal elements can be used to design individual PID controllers, phase-lag or phase-lead circuits to improve the speed, gain and phase margins of the feedback loops as for singlevariable systems(**14**).

3.7 CONCLUSION

The INA technique is a powerful multivariable method in that it includes stability analysis, which is related to singlevariable systems. The design of a pre-compensator that results in a diagonal dominant system is intricate, but can be simplified sufficiently with the aid of an interactive CACSD package.

The technique can also be applied to the design of feedback controllers for the closed-loop system by making use of the Inverse Nyquist diagrams of the diagonal elements.

4. APPLICATION OF THE INVERSE NYQUIST ARRAY TECHNIQUE

4.1 INTRODUCTION

The Inverse Nyquist Array (INA) design technique was presented in chapter three. The method is now applied to the flotation rig with specific reference to controlling the Rougher and Scavenger tanks.

This chapter demonstrates the individual steps involved in the design of a pre-compensator to achieve dominance, single-loop controllers to control the levels and finally how this control scheme can be implemented on a digital computer.

4.2 FLOTATION RIG MODEL

The transfer function representation of the two-tank model is constructed from chapter two.

$$\begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \mathbf{G}(s) \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix} \qquad (4.1)$$

where

$$\mathbf{G}(s)=\begin{bmatrix} \dfrac{1.40E\text{-}2\ s + 6.83E\text{-}5}{s2 + 3.95E\text{-}2\ s + 5.91E\text{-}4} & \dfrac{1.53E\text{-}4}{s2 + 4.07E\text{-}2\ s + 2.20E\text{-}4} \\ \dfrac{-1.22E\text{-}2}{s + 1.87E\text{-}2} & \dfrac{5.97E\text{-}3}{s + 1.19E\text{-}2} \end{bmatrix} \tag{4.2}$$

In eqn.(4.1) $\mathbf{y}_1$ and $\mathbf{y}_2$ refer to the Rougher and Scavenger tank level respectively and $\mathbf{u}_1$ and $\mathbf{u}_2$ correspond to the Rougher and Scavenger valve inputs respectively.

4.3 CLOSED-LOOP MODEL

The closed-loop control scheme, that will be designed to control the levels of the two tanks, is depicted in fig. 4.1.

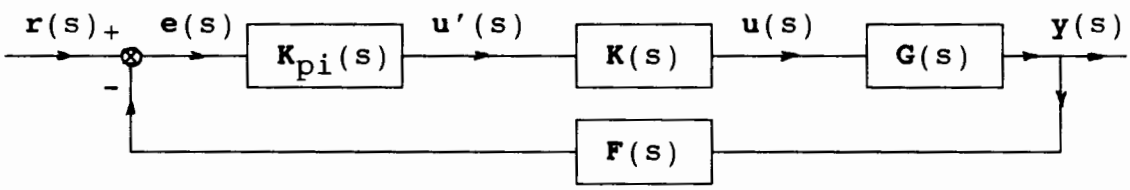


FIG. 4.1 Closed-loop Control Scheme

Vector **r** represents the setpoint levels for the two tanks, vector **e** is the comparator output, vector **u'** contains the inputs to the diagonal dominant system, vector **u** denotes the valve setting of the plant and vector **y** symbolises the tank output levels.

The diagonal matrix $\mathbf{K}_{pi}(s)$ contains the single-loop proportional plus integral (PI) controllers, matrix $\mathbf{K}(s)$ is the pre-compensator, that is designed to make the subsystem $\mathbf{Q}(s)=\mathbf{G}(s)\mathbf{K}(s)$ diagonally dominant, matrix $\mathbf{G}(s)$ represents the plant and the identity matrix $\mathbf{F}(s)$ signifies unity feedback.

4.4 ACHIEVING DIAGONAL DOMINANCE

The INACAD program [see *Software Reference*] was used throughout the design of the pre-compensator to achieve row-dominance and the PI controllers for setpoint tracking. All the Nyquist diagrams were generated by INACAD. As the software does not cater for a separate single-loop controller, the $\mathbf{L}(s)$ matrix was used for that purpose.

4.4.1 UNCOMPENSATED SYSTEM

The INA diagram of $\mathbf{G}^{-1}(s)$ with row-dominance Gershgorin circles is depicted in fig. 4.2. (The frequency range was selected from w=1E-4 to w=10). It can be seen that the interaction of the two tanks is severe, as neither the Rougher nor the Scavenger tanks are dominant.

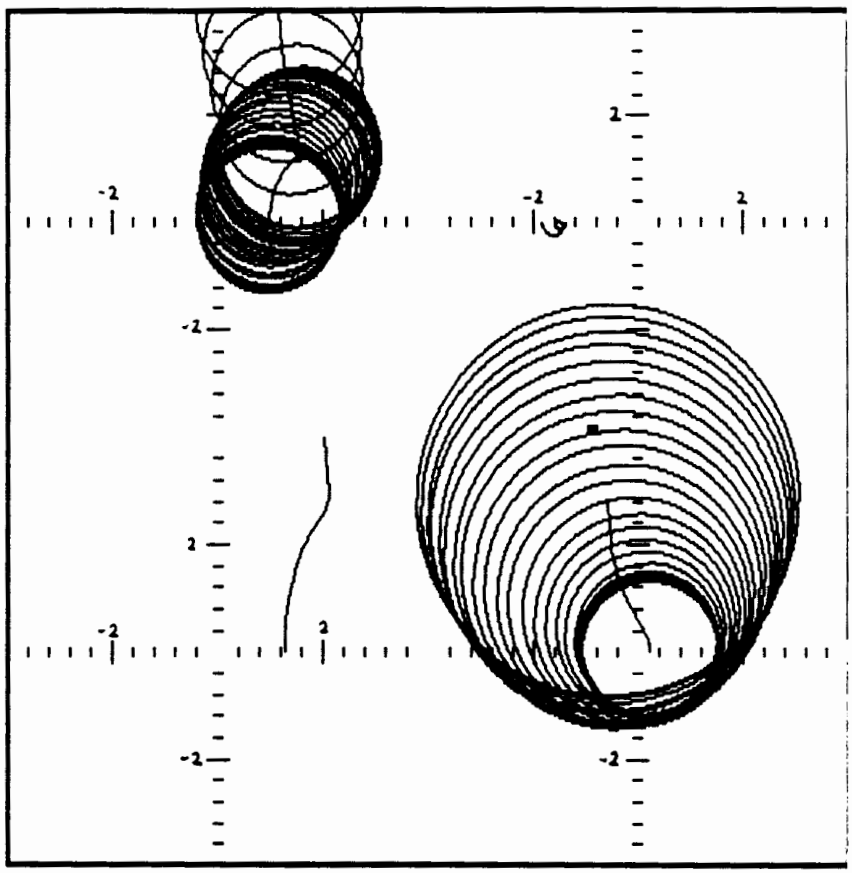


FIG. 4.2 INA Diagram of $G^{-1}(s)$ with Gershgorin circles

4.4.2 ROW OPERATIONS

The element (2,1) of $G^{-1}(s)$ is eliminated by adding a multiple of row 1 to row 2.

$$Row_2 = Row_2 + \alpha(s)\ Row_1 \qquad (4.3)$$

The multiplication function and its approximation is shown in fig. 4.3.

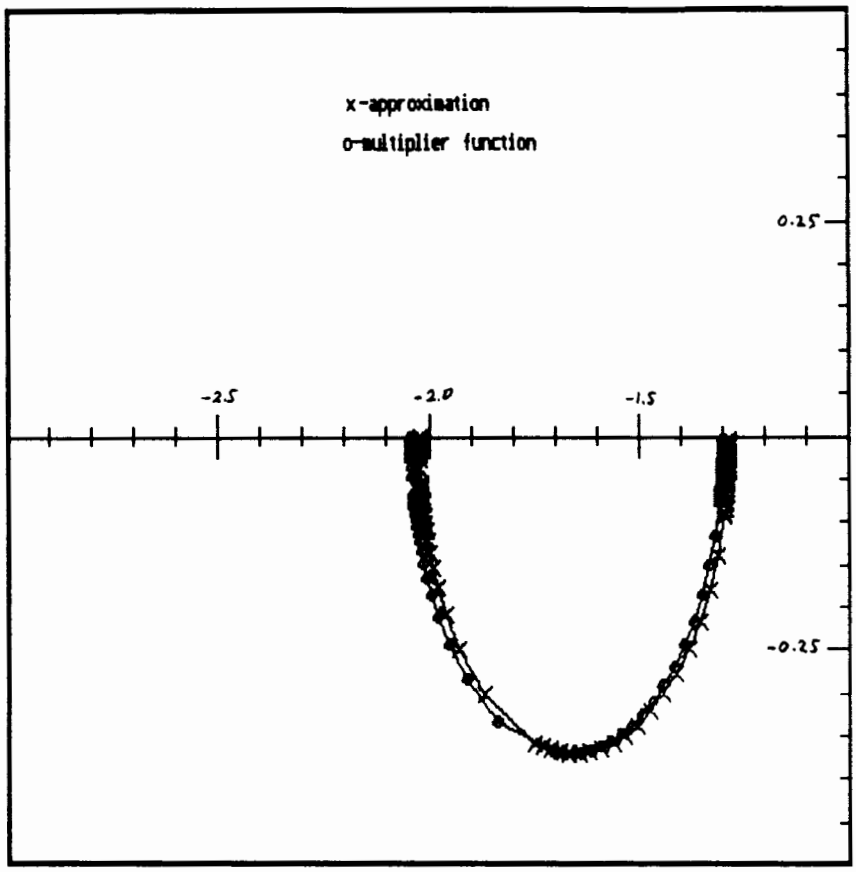


FIG. 4.3 Multiplication function to eliminate element (2,1)

The coefficients of the multiplication function were found by a non-linear parameter adjustment technique [see NELM in Chapter 7] then

$$\alpha(s) = - \frac{178.8\ s + 1.748}{88.38\ s + 1.363} \qquad (4.4)$$

and the corresponding pre-compensator matrix is

$$K_1(s) = \begin{bmatrix} 1 & 0 \\ \dfrac{178.8\ s + 1.748}{88.38\ s + 1.363} & 1 \end{bmatrix} \qquad (4.5)$$

The result of this row operation is depicted in Fig. 4.4. It can be seen that element (2,2) is now row-dominant.

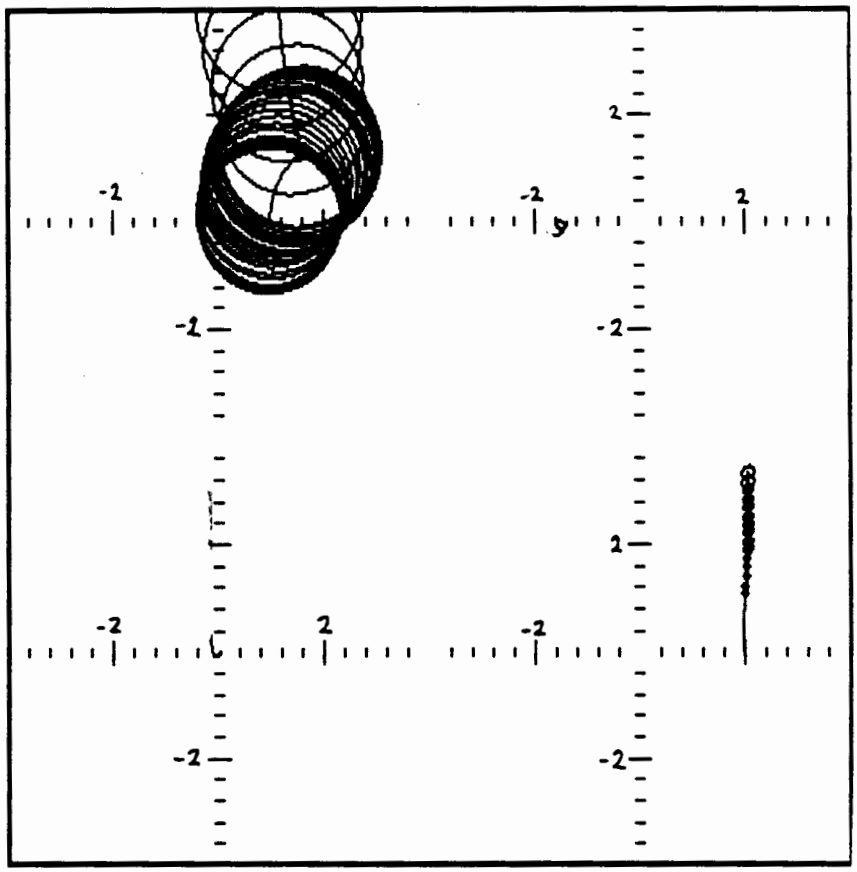


FIG. 4.4 INA Diagram of $G^{-1}(s)K_1$ with Gershgorin circles

In a similar fashion the element (1,2) of $\mathbf{G}^{-1}(s)\mathbf{K}_1$ is now eliminated by adding a multiple of row 2 to row 1.

$$\text{Row}_1 = \text{Row}_1 + \alpha(s)\ \text{Row}_2 \qquad (4.6)$$

The multiplication function and its approximation is shown in fig. 4.5.

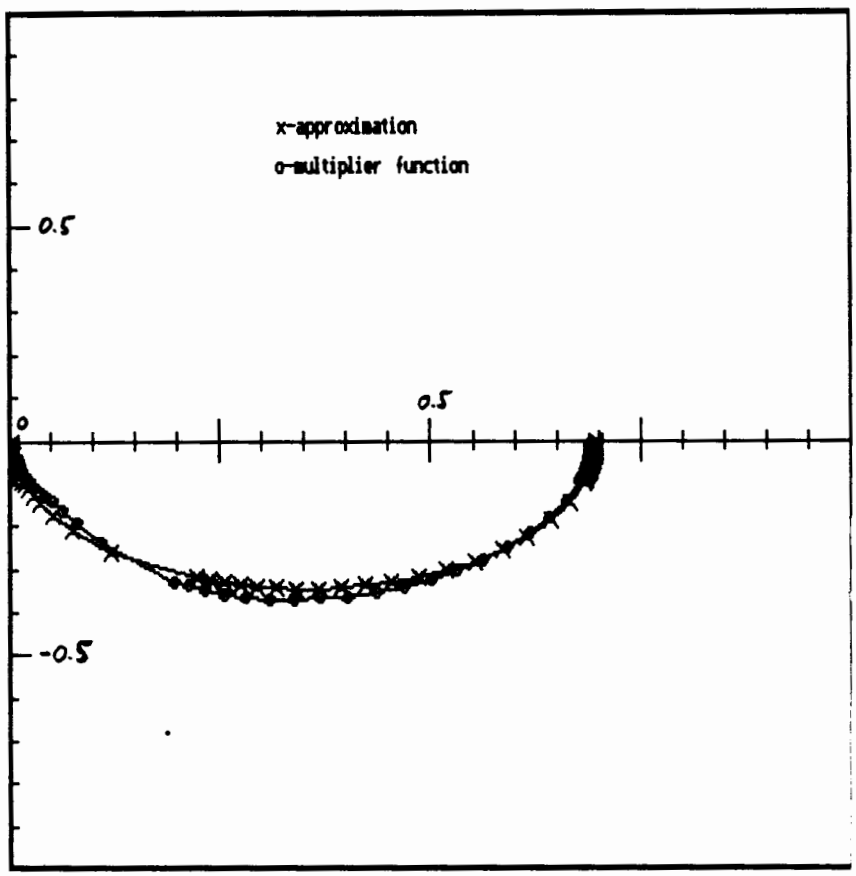


FIG. 4.5 Multiplication function to eliminate element (1,2)

The coefficients were optimised and then

$$\alpha(s) = \frac{0.2935\ s + 0.8089}{85.31\ s + 1.159} \tag{4.7}$$

and the corresponding pre-compensator matrix is

$$K_2(s) = \begin{bmatrix} 1 & -\dfrac{0.2935\ s + 0.8089}{85.31\ s + 1.159} \\ 0 & 1 \end{bmatrix} \tag{4.8}$$

The INA diagram in fig. 4.6 shows that the plant has successfully been decoupled and is now row dominant.

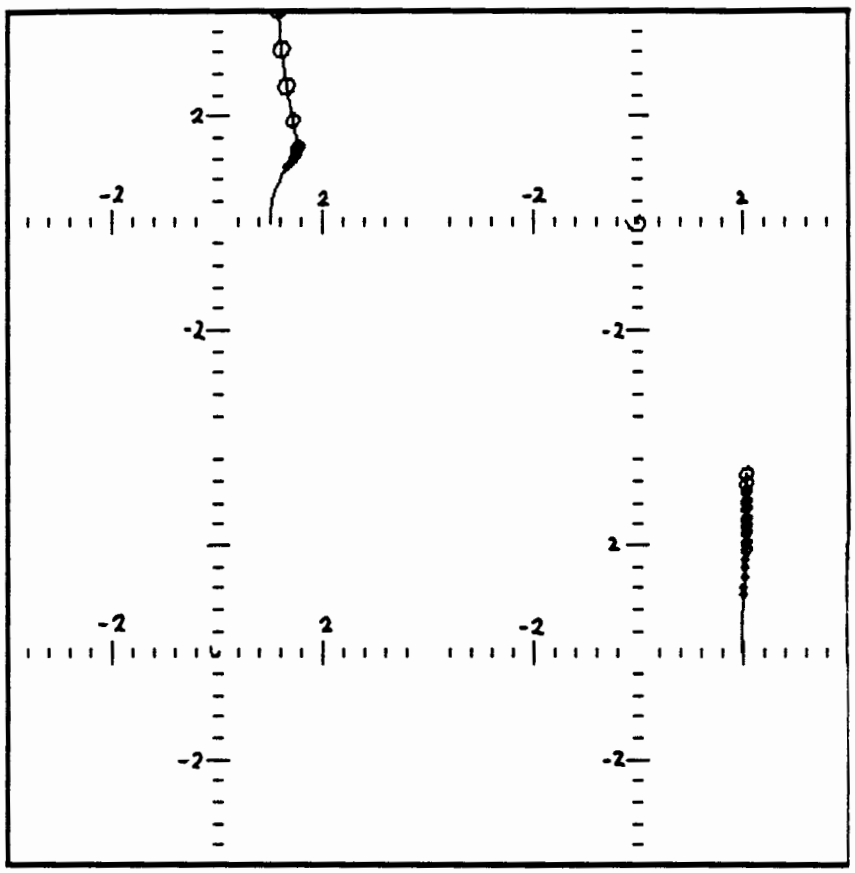


FIG. 4.6 INA Diagram of $G^{-1}(s)K_1K_2$ with Gershgorin circles

4.5 DESIGN OF SINGLE-LOOP PI CONTROLLERS

Since the control system is now diagonally dominant single-loop controllers can be designed by singlevariable techniques.

The PI controllers were designed such that the closed-loop system has step-responses with a maximum gain of 1.3, both

levels are controlled equally fast and the noise sensitivity is kept at a minimum.

The resulting PI controller was found to be:

$$\mathbf{K}_{pi}(s) = \begin{bmatrix} \dfrac{10s+1}{s} & 0 \\ 0 & \dfrac{25s+2.5}{s} \end{bmatrix} \qquad (4.9)$$

The resulting Direct Nyquist diagram is depicted in fig. 4.7.

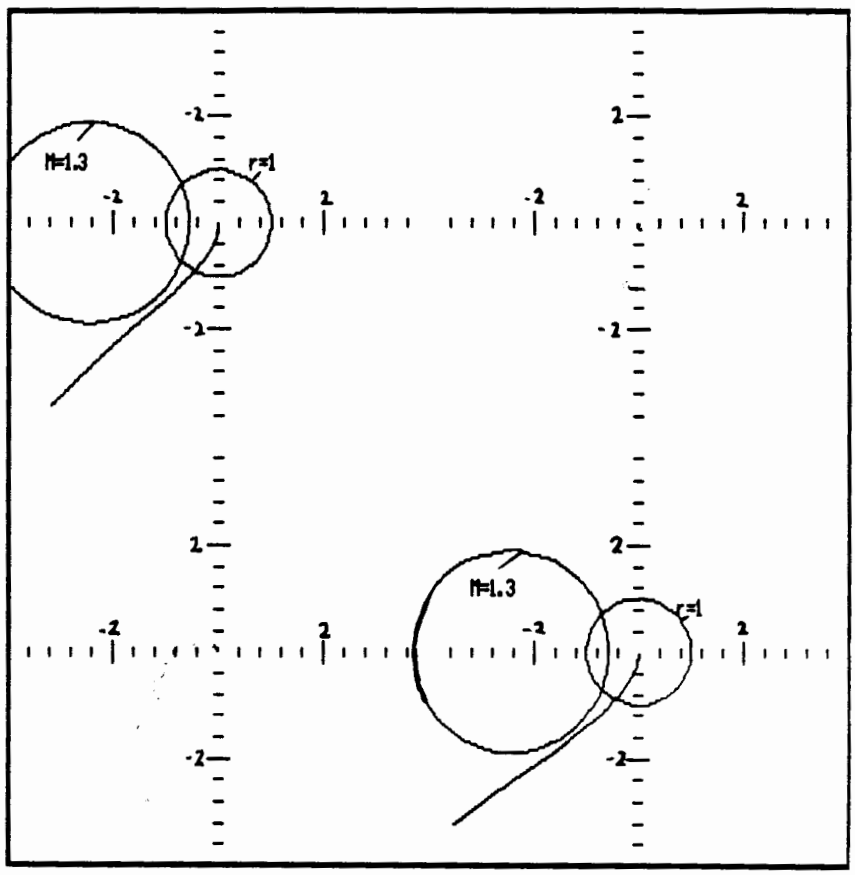


FIG. 4.7 Direct Nyquist Diagram of final control system

4.6 INTERPRETATION OF INITIAL CONDITIONS

The block diagram of Fig. 4.8 depicts the control system where $\mathbf{r}_t$ contains the setpoint levels, $\mathbf{e}_t$ the comparator output, $\mathbf{u'}_t$ the input to the dominant system, $\mathbf{u}_t$ the actual plant input and $\mathbf{y}_t$ the actual plant output, at time t and the initial valve settings are in $\mathbf{u}_o$.

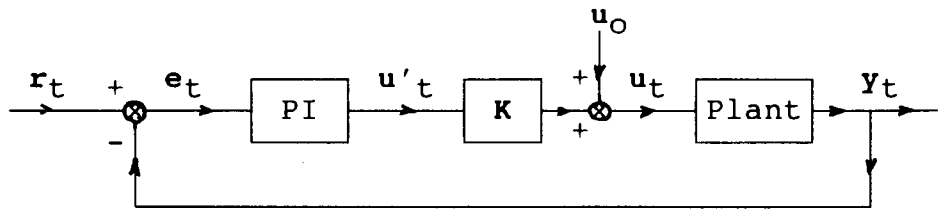


FIG. 4.8 Implementation of initial conditions

At the time t=0, when the control scheme is activated the setpoint levels are set to the initial output levels, the integrators of the PI controller and the initial conditions for **K** are set to zero, then

$$r_t = y_o ,$$

$$e_t = 0$$

$$u'_t = 0$$

and

$$u_t = u_o . \qquad (4.10)$$

The actual plant input u_t thus remains unchanged until the levels y_t of the tanks differ from the setpoint levels.

4.7 DIGITAL IMPLEMENTATION OF K

This section discusses the implementation of the above pre-compensators $k_{ij}(s)$ and the PI controllers $k_{pi}(s)$ on a digital computer.

4.7.1 THE BASIC DIGITAL CONTROLLER

The equation relating the input u'(s) to the output y'(s) of a singlevariable controller k(s) is given by:

$$y'(s) = k(s)u'(s) \tag{4.12}$$

From eqn. (4.5) & (4.8), it is found that the functions can be formed by:

$$k(s) = \frac{a_1 s + a_0}{b_1 s + b_0}$$

$$= A + \frac{B}{s + b_0/b_1} \tag{4.13}$$

where

$$A = a_1/b_1$$

$$B = \frac{a_0 b_1 - a_1 b_0}{b_1 b_1}$$

The z-transform of k(s) is

$$k(z) = v(z) + w(z) \tag{4.14a}$$

where

$$v(z) = A \tag{4.14b}$$

and

$$w(z) = \frac{Bz}{z - \sigma z^{-1}} \tag{4.14c}$$

with

$$\sigma = e^{-\tau b_0/b_1} \tag{4.14d}$$

where τ is the sampling period.

The controller function is then in discrete form:

$$k_n = v_n + w_n \tag{4.15}$$

As v_n is a simple gain function, using eqs.(4.12) & (4.14b), then

$$v_n = A\, u'_n \tag{4.16}$$

but w_n is a function of z, therefore the direct programming method, as it uses the minimum number of past values, is applied as follows:

Both the numerator and the denominator of the transfer function are multiplied by $z^{-1}q(z)$ to give:

$$\frac{w(z)}{u'(z)} = \frac{Bz}{z-\sigma}\,\frac{z^{-1}\,q(z)}{z^{-1}\,q(z)} \tag{4.17}$$

Equating the denominator of Eqn. (4.17) gives

$$\begin{aligned} u'(z) &= (1-\sigma z^{-1})q(z) \\ &= q(z) - \sigma z^{-1}q(z) \end{aligned} \tag{4.18}$$

then

$$q(z) = u'(z) + \sigma z^{-1}q(z) \tag{4.19}$$

In discrete form the above equation is written as

$$q_n = u'_n + \sigma q_{n-1} \tag{4.20}$$

By equating the numerator of eqn. (4.17), it is found that

$$w(z) = Bq(z) \tag{4.21}$$

and in discrete from this is written as

$$w_n = Bq_n \tag{4.22}$$

From eqs. (4.15b),(4.16) and (4.22) the discrete controller becomes

$$k_n = Au'_n + Bq_n \tag{4.23}$$

where q_n is given by Eqn.(4.20).

4.7.2 THE PRE-COMPENSATOR

Having established how a digital controller is implemented, the next step is to realise the row operation matrices.

The designed pre-compensator K(s) can be written as

$$\mathbf{K}(s) = \mathbf{K}_2(s)\mathbf{K}_1(s) \tag{4.24}$$

From the block diagram in fig. 4.1, the corresponding transfer function is

$$\mathbf{u}(s) = \mathbf{K}(s)\ \mathbf{u}'(s) \tag{4.25}$$

This can be split into two equations such that

$$\mathbf{u}^*(s) = \mathbf{K}_1(s)\ \mathbf{u}'(s) \tag{4.26a}$$

$$\mathbf{u}(s) = \mathbf{K}_2(s)\ \mathbf{u}^*(s) \tag{4.26b}$$

The pre-compensator matrix $\mathbf{K}_1$ from eqn.(4.5) can be written as

$$\mathbf{K}_1(s) = \begin{bmatrix} 1 & 0 \\ k^1(s) & 1 \end{bmatrix} \tag{4.27a}$$

where

$$k^1(s) = \frac{178.8\ s + 1.748}{88.38\ s + 1.363} \tag{4.27b}$$

The pre-compensator can then be implemented in discrete form, where $u_{i,n}$ corresponds to the ith input at time n, as

$$u^*_{1,n} = u'_{1,n}$$
$$u^*_{2,n} = k^1_n + u'_{2,n} \qquad (4.28)$$

where k^1_n is calculated by methods outlined in the previous Section with a sampling period $\tau=1$, then

$$k^1_n = A\, u'_{1,n} + Bq_{1,n} \qquad (4.29a)$$

where

$$q_{1,n} = u'_{1,n} + \sigma q_{1,n-1} \qquad (4.29b)$$

with initial condition $q_{1,n=0} = 0$.

In this case

$$A = 2.023$$

$$B = -0.01142$$

and

$$\sigma = 0.9847$$

In a similar fashion as above, the pre-compensator matrix $\mathbf{K}_2$ from Eqn.(4.8) can be written as

$$\mathbf{K}_2(s) = \begin{bmatrix} 1 & k^2(s) \\ 0 & 1 \end{bmatrix} \qquad (4.27a)$$

where

$$k^2(s) = \frac{-0.2935\ s\ -0.8089}{85.31\ s\ +\ 1.159} \qquad (4.7)$$

The pre-compensator can then be implemented in discrete form, where $u_{i,n}$ corresponds to the ith input at time n, as

$$u_{1,n} = u^*_{1,n} + k^2_n$$
$$u_{2,n} = u^*_{2,n} \qquad (4.28)$$

k^2_n is calculated by methods outlined in the previous Section, again with a sampling period τ=1 and found to be

$$k^2_n = A\, u'_{2,n} + Bq_{2,n} \qquad (4.29a)$$

where

$$q_{2,n} = u'_{2,n} + \sigma q_{2,n-1} \qquad (4.29b)$$

with initial condition $q_{2,n=0} = 0$.

In this case

$$A = 3.440E\text{-}3$$

$$B = -9.435E\text{-}3$$

and

$$\sigma = 0.9865$$

4.7.3 THE PI CONTROLLERS

The PI controllers can be implemented in the following fashion.

From fig. 4.1, the transfer equation for the PI controller is

$$\mathbf{u}'(s) = \mathbf{K}_{pi}(s)\mathbf{e}(s) \qquad (4.30)$$

The PI controller matrix of Eqn.(4.9) can be written as

$$\mathbf{K}_{pi}(s) = \begin{bmatrix} k^1_{pi}(s) & 0 \\ 0 & k^2_{pi}(s) \end{bmatrix} \tag{4.31a}$$

where

$$k^1_{pi}(s) = \begin{bmatrix} \frac{10s+1}{s} \end{bmatrix} \tag{4.31b}$$

and

$$k^2_{pi}(s) = \begin{bmatrix} \frac{25s+2.5}{s} \end{bmatrix} \tag{4.31c}$$

In discrete form the individual PI controllers are realised as follows:

$$u'_{1,n} = k_1(e_{1,n}+i_{1,n}) \tag{4.32a}$$

$$i_{1,n} = i_{1,n-1} + \frac{e_{1,n}}{\tau_1} \tag{4.32b}$$

$$u'_{2,n} = k_2(e_{2,n}+i_{2,n}) \tag{4.33a}$$

$$i_{2,n} = i_{2,n-1} + \frac{e_{2,n}}{\tau_2} \tag{4.33b}$$

where

$k_1 = 10$

$k_2 = 25$

$\tau_1 = \tau_2 = 10$

and the initial conditions are

$$i_{1,n=0} = i_{2,n=0} = 0$$

4.8 CLOSED-LOOP STEP RESPONSE

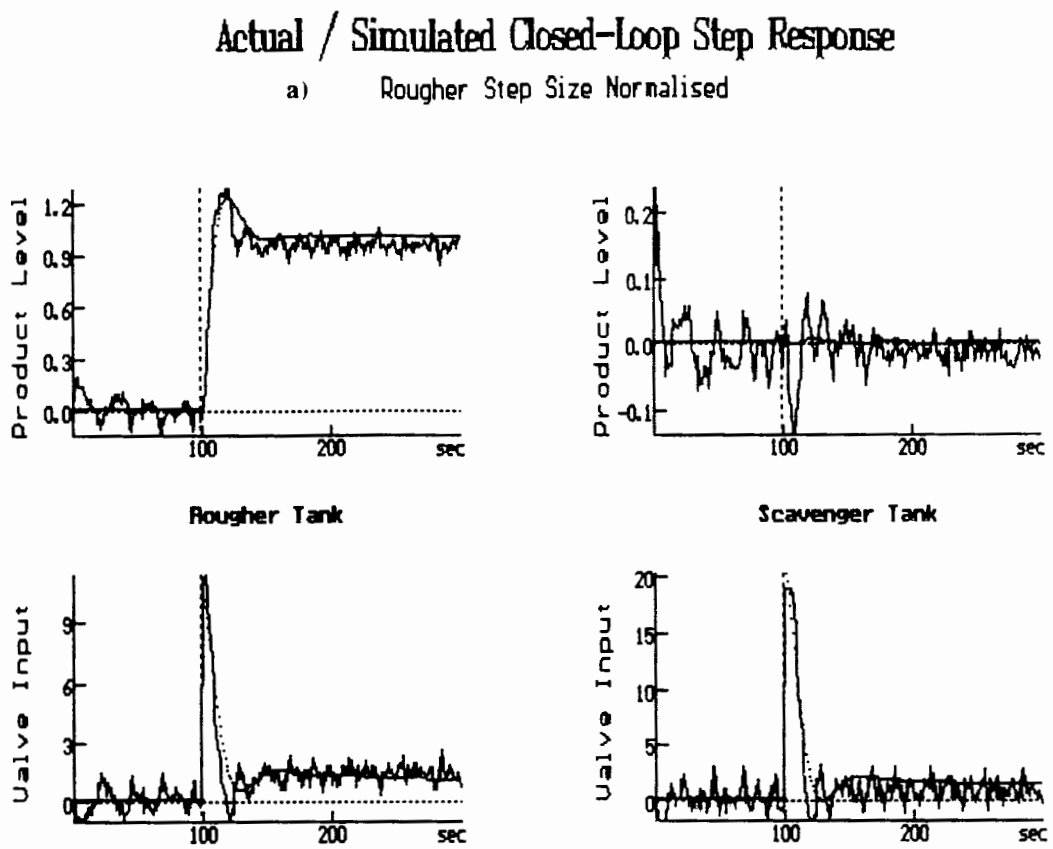
The level of all four tanks of the flotation rig were moved into their operating regions and allowed to attain steady-state before the designed control scheme was introduced, as discussed in chapter two. As the control system is designed to control only the two specified tanks, the Cleaner and Recleaner valves remained fixed throughout the tests. A step in setpoint level was applied after 100 seconds had elapsed.

The original step responses can be found in Appendix B. To compare the responses to those of a simulation, the responses were normalised by subtracting the initial level and dividing the result by the step size. The initial level is found by averaging the first 100 samples. The normalised responses of both the tanks and simulations are shown in fig. 4.9. The simulated values are shown by solid lines, except during extreme changes when they become dotted.

The responses correspond well to the simulated one if the noise of the plant is ignored.

The difference between the actual and the simulated level of the Rougher tank in Fig. 4.9a) can be attributed to the fact, that the initial level used to normalise the response was erroneous because the level had not been at steady-state at t=0. It should in fact have been lower.

The reason for the Scavenger valve inputs not reaching their simulated peaks is that the valve was driven into saturation. This also explains why the overshoot of the Scavenger level in Fig. 4.9b) is actually higher than predicted.



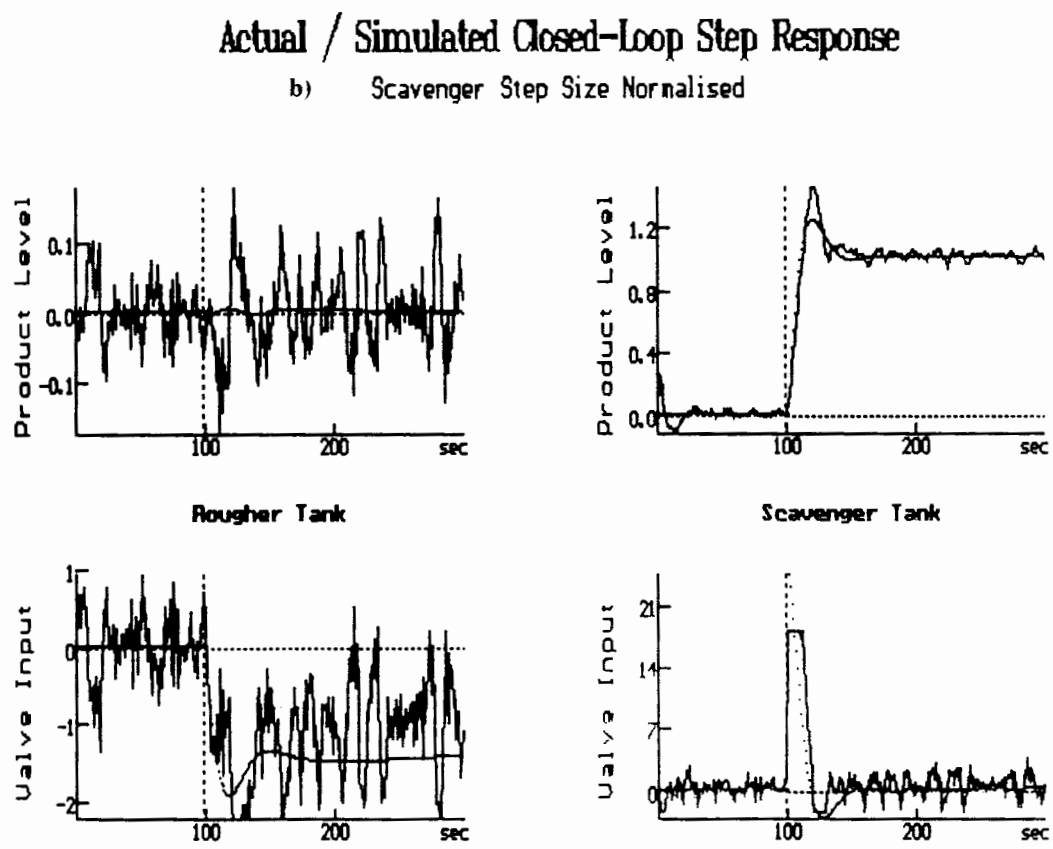


FIG. 4.9 Closed-loop system step response

It can be seen that the control system behaves in a manner as it was designed to do. The overshoot was limited to 0.3 times the step size. This can readily be confirmed by the previous step responses where the step sizes are normalised to 1. The noise sensitivity of the closed-loop system is also acceptable, as it does not impede the level control. The speed of response after a setpoint change are similar for both the Rougher and Scavenger tanks as they were designed to be.

4.9 CONCLUSION

The INA technique could successfully be applied to the flotation rig. Dominance could be achieved and single-loop PI controllers were designed by singlevariable methods for the two loops. The interaction was reduced considerably and the designed control scheme was adequate.

The similarity of the predicted and actual responses indicate that the model of the plant had been accurate enough for this application.

5. THE POLE ASSIGNMENT DESIGN TECHNIQUE

5.1 INTRODUCTION

This chapter will discuss a particular state space design method for multivariable systems, namely the Pole Assignment technique. It will show how Pole Assignment by state feedback can be implemented in multivariable systems and how it is related to single-input systems. It will focus on the dyadic approach and discuss the difficulties associated with the technique.

Observer design will also be discussed with particular reference to the Kalman filter for state estimation in a noisy environment. The latter must usually be included in any state space design method because multivariable systems tend to have a large number of states and only a few measurable outputs. The number of outputs are limited due to the high cost of instrumentation or unavailability of further measurements.

5.2 POLE ASSIGNMENT IN SYSTEM DESIGN

The pole assignment technique is well known for its application in single-input, single-output (SISO) plants to improve the closed-loop dynamic response of the system. The method is straightforward for SISO systems because the system zeroes are invariant to state feedback and the choice of closed-loop poles gives rise to a unique feedback gain

vector operating on the system states. Unfortunately other aspects of performance such as gain and integrity cannot be included into this state feedback design, because of the uniqueness of the feedback for a specific set of closed-loop pole positions.

An optimal design strategy(**14**) is possible by using dynamic feedback in conjunction with a cost function related to both states and inputs to increase the flexibility of the design, but considerable trial and error is involved until the resulting system has a satisfactory transient response.

In multivariable systems however a multiple of feedback configurations result in a specific set of pole locations. The associated degrees of freedom in this application complicate the design. A possible solution to achieve a more stringent design method is given by the dyadic approach to pole assignment.

5.3 DYADIC POLE ASSIGNMENT BY STATE FEEDBACK

Although pole assignment is possible by either output or state feedback the following discussion will be aimed toward the latter, because output feedback is in effect a design of proportional controllers, which is seldom adequate and in addition most multivariable system have less number of outputs than states, which may limit the number of poles that can be assigned.

5.3.1 SYSTEM MODEL FOR STATE FEEDBACK

To derive dyadic pole assignment the state equations of a linear, time-invariant multivariable system are considered,

because the basis for the method stems from the state space representation.

The state equation for a plant with n states, m inputs and 1 outputs is described by:

$$\dot{\mathbf{x}}(s) = \mathbf{A}\mathbf{x}(s) + \mathbf{B}\mathbf{u}(s)$$

$$\mathbf{y}(s) = \mathbf{C}\mathbf{x}(s) \qquad (5.1)$$

where **A**,**B** are real matrices with dimension nxn, nxm, lxn, respectively. The vector **x** represents the system states and vector **u** represents the inputs. Fig. 5.1 shows the system model in block diagram form.

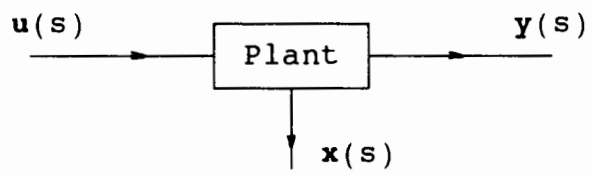


FIG. 5.1 Multivariable System Block Diagram

By applying pole assignment in state feedback the derivation is concerned with the state equation and the output equation may be omitted from the discussion.

The corresponding transfer-function matrix relating the inputs to the system states is given by matrix **G**(s) as follows:

$$\mathbf{x}(s) = \mathbf{G}(s)\,\mathbf{u}(s)$$

$$\text{where } \mathbf{G}(s) = (s\mathbf{I}-\mathbf{A})^{-1}\,\mathbf{B}$$

$$= \frac{\text{Adj}(s\mathbf{I}-\mathbf{A})\,\mathbf{B}}{|s\mathbf{I}-\mathbf{A}|} \qquad (5.2)$$

Matrix **I** is the nxn identity matrix and Adj() represents the Adjoint of a matrix.

The dynamic response of the system is then determined by its characteristic equation given by:

$$\phi(s) = |s\mathbf{I}-\mathbf{A}| = 0. \qquad (5.3)$$

5.3.2 EQUIVALENT SINGLE-INPUT SYSTEM

The dyadic approach translates the multivariable system into an equivalent single input (ESI) system. The pole assignment strategy is then performed on this SI system by similar methods as used for single-variable systems. Only one particular method(**12**), related to dyadic or unity rank feedback controllers will be applied in this discussion.

As in SI systems, the resulting feedback will only affect the pole positions and not the zeroes of the initial ESI system. The final step is to translate the control law into its multivariable representation.

5.3.2.1 Open-Loop ESI System

To transform the multivariable system into an ESI system a vector **q** is chosen such that the initial state equation:

$$\dot{\mathbf{x}}(s) = \mathbf{A}\mathbf{x}(s)+\mathbf{B}\mathbf{u}(s) \qquad (5.4)$$

is transformed to

$$\dot{\mathbf{x}}(s) = \mathbf{A}\mathbf{x}(s)+\mathbf{b}\mathbf{w}(s) \qquad (5.5)$$

where

$$\mathbf{b} = \mathbf{B}\mathbf{q}$$

Vector **q** is a nx1 vector and w(s) is the single input.

The vector **q** must be chosen such that the ESI system (**A**,**b**) is state-controllable. Such a vector can always be found if the original system (**A**,**B**) is controllable and only certain directions of the **q**-vector must be avoided. Fig. 5.2 shows the block diagram of the ESI system.

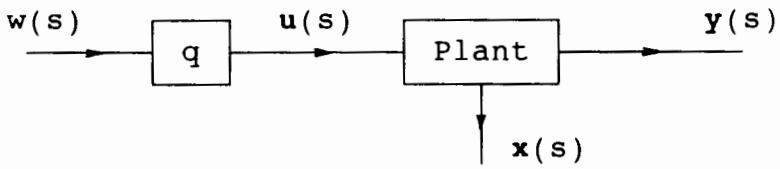


FIG. 5.2 Equivalent Single-input System

The open-loop ESI transfer function representation is then

$$\mathbf{x}(s) = \mathbf{G}_o(s)\, w(s),$$

where

$$\mathbf{G}_o(s) = (s\mathbf{I}-\mathbf{A})^{-1}\,\mathbf{b} = \frac{\mathrm{Adj}(s\mathbf{I}-\mathbf{A})^{-1}\,\mathbf{b}}{|s\mathbf{I}-\mathbf{A}|} \qquad (5.6)$$

which can be written as

$$\mathbf{G}_o(s) = \frac{\mathbf{g}_o(s)}{d_o(s)} \qquad (5.7a)$$

$$= \frac{1}{s^n+d_n s^{n-1}+\ldots+d_1}\begin{bmatrix} k_{11}+\ldots+k_{1n}s^{n-1} \\ k_{n1}+\ldots+k_{nn}s^{n-1} \end{bmatrix} \qquad (5.7b)$$

where $\mathbf{g}_O(s)$ is a n-vector of polynomials in s describing the zero locations and $d_O(s)$ is the characteristic polynomial whose roots are the open-loop poles for the ESI system.

5.3.2.2 Closed-Loop ESI System

The closed-loop system description can be found by using a state feedback controller as follows:

$$w(s) = v(s) - \mathbf{f}^T\mathbf{x}(s), \tag{5.8}$$

where v(s) is the new external single input and **f** is the scalar feedback nx1 vector.

Then the closed loop ESI system becomes

$$\dot{\mathbf{x}}(s) = (\mathbf{A}-\mathbf{b}\mathbf{f}^T)\mathbf{x}(s) + \mathbf{b}v(s) \tag{5.9}$$

and the closed-loop ESI transfer function relating the input to the states is:

$$\mathbf{x}(s) = \mathbf{G}_C(s)\, v(s) \tag{5.10}$$

$$\mathbf{G}_C(s) = \{s\mathbf{I}-(\mathbf{A}-\mathbf{b}\mathbf{f}^T)\}^{-1}\,\mathbf{b}$$

$$= \frac{\mathrm{Adj}\{s\mathbf{I}-(\mathbf{A}-\mathbf{b}\mathbf{f}^T)\}\,\mathbf{b}}{|s\mathbf{I}-(\mathbf{A}-\mathbf{b}\mathbf{f}^T)|}$$

$$= \frac{\mathbf{g}_C(s)}{d_C(s)} \tag{5.11}$$

where $\mathbf{g}_C(s)$ and $d_C(s)$ are the corresponding representation as before, but for the closed-loop ESI system.

Alternatively the closed-loop system may be found by using Eqn. (5.6) and Eqn. (5.8) then:

$$\mathbf{x}(s) = \frac{\mathbf{g}_o(s)\ w(s)}{d_o(s)}$$

$$= \frac{\mathbf{g}_o(s)\ \{v(s) - \mathbf{f}^T\mathbf{x}(s)\}}{d_o(s)}$$

$$= \frac{\mathbf{g}_o(s)\ v(s)}{d_o(s)+\mathbf{f}^T\mathbf{g}_o(s)} \qquad (5.12)$$

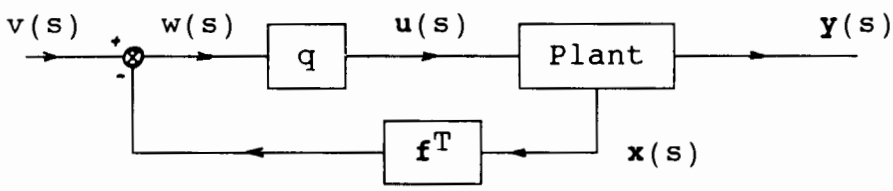


FIG. 5.3 Closed-loop ESI System

Fig. 5.3. shows the block diagram of the resulting closed-loop ESI system.

From equating Eqn. (5.7a) and Eqn. (5.12) it follows that:

$$\mathbf{g}_c(s) = \mathbf{g}_o(s) \qquad (5.13)$$

and

$$d_c(s) = d_o(s) + \mathbf{f}^T\mathbf{g}_o(s) \qquad (5.14)$$

From Eqn. (5.13) the invariability of the open- and closed-loop zeroes are confirmed.

Eqn. (5.14) is known as the *pole assignment equation* and determines the scalar state feedback vector **f** for a given set of closed-loop pole positions. An examination of the relationship shows that if no poles are to be moved by the

feedback then $\mathbf{f} = 0$, which is an obvious but important result, when analysing system integrity.

From Eqn. (5.14) also follows that in general arbitrary pole assignment is possible for controllable systems(**16**). The constraints are that the desired poles, if complex must occur in conjugate pairs and all states must be available for feedback. If all states are not measurable in a observable system, they can be made available by using a Luenberger or reduced observer(**7**), eliminating the problems associated with incomplete state feedback, where only some poles can be assigned to specific positions.

5.3.3 CLOSED-LOOP MULTI-INPUT SYSTEM

To transform the resulting feedback system back into the multivariable system the final control law is given by:

$$\mathbf{K} = \mathbf{q}\mathbf{f}^T \tag{5.15}$$

such that

$$\mathbf{u}(s) = -\mathbf{K}\mathbf{x}(s) + \mathbf{v}(s) \tag{5.16}$$

where vector $\mathbf{u}(s)$ represents the plant inputs and $\mathbf{v}(s)$ is the new system input vector.

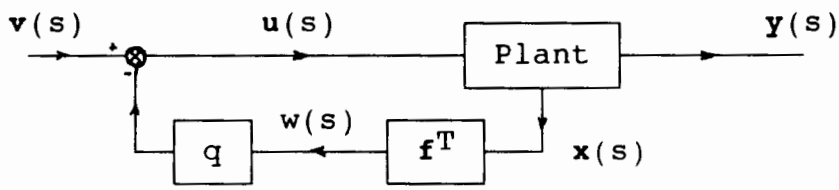


FIG. 5.4 Closed-loop Multi-input System

A block diagram of the closed-loop MI system is given in Fig. 5.4.

Matrix **K** is termed a unity rank feedback controller because it is constructed from the product of two vectors or dyadic pair. The vector **q** can be regarded as distributing the scalar state feedback value $\mathbf{f}^T\mathbf{x}$ across all the inputs to the plant.

The general closed-loop multivariable system becomes:

$$\dot{\mathbf{x}}(s) = \mathbf{A}_C\mathbf{x}(s) + \mathbf{B}\mathbf{v}(s)$$

$$\mathbf{y}(s) = \mathbf{C}\mathbf{x}(s) + \mathbf{D}\{-\mathbf{K}\mathbf{x}(s) + \mathbf{v}(s)\} \qquad (5.17)$$

where

$$\mathbf{A}_C = (\mathbf{A}-\mathbf{B}\mathbf{K})$$

$$= (\mathbf{A}-\mathbf{b}\mathbf{f}^T)$$

The complete closed-loop transfer function for a strictly proper system (**i.e.** **D**=0) is then:

$$\mathbf{G}(s) = \mathbf{C}\,(s\mathbf{I}-\mathbf{A}_C)^{-1}\,\mathbf{B}$$

$$= \frac{\mathbf{C}\ \mathrm{Adj}(s\mathbf{I}-\mathbf{A}_C)\ \mathbf{B}}{|s\mathbf{I}-\mathbf{A}_C|} \qquad (5.18)$$

The new characteristic equation being:

$$\phi_C = |s\mathbf{I}-\mathbf{A}_C| = 0 \qquad (5.19)$$

containing the desired pole positions.

The transient response of the final multivariable system is then governed by the selected pole positions specified with $d_C(s)$ and zero locations given by $\mathbf{C}\ \mathrm{Adj}(s\mathbf{I}-\mathbf{A}_C)\ \mathbf{B}$.

5.3.4 NUMERICAL TECHNIQUES IN DYADIC POLE ASSIGNMENT

The pole assignment technique contains considerable computational effort and would be extremely tedious to use if calculations were to be made by hand, but by programming a digital computer with a suitable algorithm its use can be simplified greatly. This section will show how the feedback law is computed and suggest appropriate numerical techniques. A complete FORTRAN listing of the program is available in the *Software Reference*.

Determination of Controllability

The first step of the numerical technique involves choosing an appropriate vector **q**.

The ESI system will be controllable if and only if

$$\mathbf{P} = (\mathbf{Bq}\ \mathbf{ABq}\ \mathbf{A}^2\mathbf{Bq}\ \ldots\ \mathbf{A}^{n-1}\mathbf{Bq}) \qquad (5.20)$$

has rank n.(**13**)

To test for controllability a good numerical technique is to calculate the determinant of **P** by manipulating it into echolon form and calculating the trace of the resulting matrix.(**5**) If any diagonal element of the resulting matrix is zero then the resulting determinant will be zero and the system will be rank deficient.

5.3.4.1 Frequency Domain Technique

A possible numerical technique to calculate the required state feedback matrix is related to the frequency domain representation of control systems.

For this method, $\mathbf{g}_o(s)$ and $d_o(s)$ must be calculated from:

$$\frac{\mathbf{g}_o(s)}{d_o(s)} = \frac{\text{Adj}(s\mathbf{I}-\mathbf{A})\ \mathbf{b}}{|s\mathbf{I}-\mathbf{A}|} \tag{5.21}$$

A recursive procedure by D.K. Faddeev to compute this has been shown previously in section 1.7.1.1.

The closed-loop characteristic polynomial $d_c(s)$ can simply be calculated from:

$$\begin{aligned} d_c(s) &= (s-p_1)(s-p_2)..(s-p_n) \\ &= s^n+d_{n-1}s^{n-1}+...+d_0 \end{aligned} \tag{5.22}$$

where the desired pole positions are given by p_i, i=1,n.

An approach to calculate the unity rank feedback controller for pole assignment by state feedback has been given by Fallside and Seraji(**17**).

The technique transforms the matrix $\mathbf{g}_o(s)$, with polynomials of s as its elements to a matrix **M** of constant coefficients.

Specifically:

$$\mathbf{g}_o(s) = \mathbf{M}^T\mathbf{s} \tag{5.23}$$

where **s** is a vector of powers of s:

$$\mathbf{s} = \begin{bmatrix} 1 \\ s \\ s^2 \\ \vdots \\ s^{n-1} \end{bmatrix} \tag{5.24}$$

The characteristic function $d_o(s)$ is also transformed from a polynomial of s to a constant coefficient vector $\mathbf{a}_o$.

Then the closed-loop characteristic function becomes:

$$d_C(s) = s^n + \mathbf{a}_C^T \mathbf{s} \tag{5.25}$$

and that for the open-loop

$$d_O(s) = s^n + \mathbf{a}_O^T \mathbf{s} \tag{5.26}$$

The pole assignment equation given by:

$$d_C(s) = d_O(s) + \mathbf{f}^T \mathbf{g}_O(s) \tag{5.27}$$

becomes

$$\mathbf{a}_C^T \mathbf{s} = \mathbf{a}_O^T \mathbf{s} + \mathbf{f}^T \mathbf{M}^T \mathbf{s} \tag{5.28}$$

From Eqn. (5.28) it follows that:

$$\mathbf{a}_C = \mathbf{a}_O + \mathbf{f}\mathbf{M} \tag{5.29}$$

The feedback vector **f** is then found by solving the simultaneous equations given by:

$$\mathbf{M}\mathbf{f} = \mathbf{a}_C - \mathbf{a}_O \tag{5.30a}$$

where matrix **M** is related to $\mathbf{g}_O(s)$, while $\mathbf{a}_O$ and $\mathbf{a}_C$ are related to $d_O(s)$ and $d_C(s)$ respectively.

Any numerical method to solve the simultaneous equations can be applied here. A less accurate method would be to use a matrix inversion routine to find $\mathbf{M}^{-1}$ and then:

$$\mathbf{f} = \mathbf{M}^{-1}(\mathbf{a}_C - \mathbf{a}_O) \tag{5.30b}$$

5.3.4.2 Controllable Companion Form Technique

A different method which may be used as a check or as an alternative to the algorithm presented above is used by Borrie(**5**).

The matrices **A** and **b** of the ESI system are converted to controllable companion form using the transformation **x=Tz** to give the following open-loop state equation:

$$\dot{\mathbf{z}} = \mathbf{A}_o\mathbf{z}+\mathbf{b}_o w \tag{5.31}$$

where

$$\mathbf{A}_o = \mathbf{T}^{-1}\mathbf{A}\mathbf{T} \tag{5.32}$$

and

$$\mathbf{b}_o = \mathbf{T}^{-1}\mathbf{b} = (0\ 0\ \ldots\ 1)^T \tag{5.33}$$

The nxn matrix **T** can be found from the controllability matrix **P** given by Eqn.(5.20) as follows:

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{p}_n{}^T \\ \mathbf{p}_n{}^T\mathbf{A} \\ \vdots \\ \mathbf{p}_n{}^T\mathbf{A}^{n-1} \end{bmatrix} \tag{5.34}$$

where $\mathbf{p}_n{}^T$ is the nth (last) row of $\mathbf{P}^{-1}$. Care must be taken that $\mathbf{A}^i$ is computed first before it is multiplied by $\mathbf{p}_n{}^T$.

As before it is necessary for matrix **P** to have rank n, i.e. the system (**A**,**b**) must be fully controllable.

It can be shown that the matrix **T** is related to the matrix **M** used in the method by Fallside(**17**) as follows:

$$\mathbf{T} = \mathbf{M}^T \tag{5.35}$$

In the controllable form the closed-loop state equations become:

$$\dot{\mathbf{z}} = (\mathbf{A}_O - \mathbf{b}_O \mathbf{f}_C^T)\mathbf{z} + \mathbf{b}_O v \tag{5.36}$$

where $\mathbf{f}_C$ is the state feedback vector in controllable form.

The closed-loop characteristic equation is found from the desired pole positions $p_1, p_2 \ldots p_n$ as before:

$$(s-p_1)(s-p_2)..(s-p_n) = s^n + a_{c(n-1)}s^{n-1} + \ldots + a_{c0} \tag{5.37}$$

The desired closed-loop state equations are then also given by:

$$\dot{\mathbf{z}} = \mathbf{A}_C \mathbf{z} + \mathbf{b}_C v \tag{5.38}$$

where the nth (last) row of $\mathbf{A}_C$ is:

$$-a_{c0} \; -a_{c1} \; \cdots \; -a_{c(n-1)} \tag{5.49}$$

By equating Eqn. (5.36) and Eqn. (5.38) and using the fact that:

$$\mathbf{b}_O = \mathbf{b}_C \tag{5.50}$$

for both the open- and closed-loop system it is found that:

$$(\mathbf{A}_O - \mathbf{b}_O \mathbf{f}_C^T) = \mathbf{A}_C \tag{5.51}$$

and

$$\mathbf{b}_O \mathbf{f}_C^T = \mathbf{A}_O - \mathbf{A}_C \tag{5.52}$$

It can be shown that the nxn matrix $\mathbf{b}_O \mathbf{f}_C^T$ has as its nth (last) row the vector $\mathbf{f}_C^T$ and all other elements equal to zero.

Then

$$\mathbf{f}_C = \mathbf{a}_C - \mathbf{a}_O \tag{5.53}$$

where $-\mathbf{a}_i^T$ are nth (last) rows of A_i respectively.

The feedback vector **f** for the ESI system can then be determined from:

$$\begin{aligned} \mathbf{f}^T\mathbf{x} &= \mathbf{f}^T\mathbf{T}\mathbf{z} \\ &= \mathbf{f}_C{}^T\mathbf{z} \\ &= \mathbf{f}_C{}^T\mathbf{T}^{-1}\mathbf{x} \end{aligned} \tag{5.53}$$

and then

$$\begin{aligned} \mathbf{f}^T &= \mathbf{f}_C{}^T\mathbf{T}^{-1} \\ &= (\mathbf{a}_C - \mathbf{a}_O)^T\mathbf{T}^{-1} \end{aligned} \tag{5.54a}$$

and

$$\mathbf{f} = (\mathbf{T}^{-1})^T(\mathbf{a}_C - \mathbf{a}_O) \tag{5.54b}$$

Comparing Eqn. (5.30b) and Eqn. (5.54b) shows the similarity of the two methods. In the one technique matrix $(\mathbf{T}^{-1})^T$ is found from the controllability matrix **P**, and in the other $\mathbf{M}^{-1}$ is calculated by the Faddeev algorithm.

From this it follows that the two matrices are equivalent (Eqn. 5.35) and this fact can be used to compare the results of the alternative method to the algorithm presented before. As the computation involved differs for the two methods, it is found that **T** will in fact be different to $\mathbf{M}^T$ and as the system order increases they will diverge. This will be shown in detail in chapter six.

Both methods have been found not to be numerically adequate for ill-conditioned and high-order systems. For the plant used in this study however, the computational accuracy was

acceptable, but alternative methods have been found lately(**18,19,20**).

5.3.5 SUMMARY OF DYADIC FEEDBACK DESIGN

1) Choose the vector **q** to transform the multi-input system to an ESI system and check for controllability.

2) Select the closed-loop system pole positions and calculate vector $\mathbf{f}^T$.

3) Insert the state feedback $\mathbf{K} = \mathbf{q}\mathbf{f}^T$ into the multivariable system.

4) Simulate the transient response of the closed-loop system (**A-BK**) for step inputs and if unsatisfactory change the selection of **q** and/or pole positions and repeat the procedure.

5.3.6 ANALYSIS OF ZERO MOVEMENT

This example will show the effects of dyadic pole assignment on a multivariable system. It will illustrate first the outcome of choosing a specific **q**-vector and varying the pole positions and secondly fixing the pole positions and varying the **q**-vector.

This illustration will concentrate on the affects of pole assignment on the system zeroes. The definition of multivariable system zeroes used here varies from those given by other authors(**3,13**).

Consider a system with transfer function:

$$\mathbf{y}(s) = \mathbf{G}(s)\ \mathbf{u}(s)$$

where

$$\mathbf{G}(s) = \begin{bmatrix} 1/(s+2) & 0 \\ 1/(s+1) & 1/(s+3) \end{bmatrix}$$

Then a possible state space representation of form:

$$\dot{\mathbf{x}}(s) = \mathbf{A}\mathbf{x}(s) + \mathbf{B}\mathbf{u}(s)$$

$$\mathbf{y}(s) = \mathbf{C}\mathbf{x}(s)$$

has

$$\mathbf{A} = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

As the first step of the procedure choose

$$\mathbf{q}^T = (1,1)$$

which is found to make the system $(\mathbf{A},\mathbf{b})$ controllable i.e.

$$|\mathbf{b}\ \mathbf{A}\mathbf{b}\ \mathbf{A}^2\mathbf{b}| = 2$$

then from $\mathbf{b}=\mathbf{B}\mathbf{q}$

$$\mathbf{b}^T = (1,1,1)$$

5.3.6.1 Single-Input System Zeroes

The open-loop ESI system transfer function has:

$$\mathbf{g}_O(s) = \begin{bmatrix} s^2+4s+3 \\ s^2+5s+6 \\ s^2+3s+2 \end{bmatrix}$$

The ESI system zeroes are therefore at:

$s=-1,-3,-2,-3,-1$ and -2

It is important to note, that the definition of the system zeroes given above is not strictly correct, because the zeroes above would in fact vanish on evaluating the system transfer function $\mathbf{g}_o(s)/d_o(s)$. It will become apparent later, that for the purpose of this example, it is however necessary to observe all the zeroes before cancellation.

To assign the closed-loop pole positions to

$s=-2,-2$ and -3

the characteristic function for the open-loop system is:

$$d_o(s) = s^3+6s^2+11s+6$$

and that for the closed-loop system is:

$$d_c(s) = s^3+7s^2+16s+12$$

The resulting feedback law has

$$\mathbf{f}^T = (0,1,0)$$

The matrix A_c is found to be

$$\mathbf{A}_c = \begin{bmatrix} -2 & -1 & 0 \\ 0 & -2 & 0 \\ 0 & -1 & -3 \end{bmatrix}$$

with poles at:

$s=-2,-2,-3$

and zero positions given by:

$$\mathbf{g}_C(s) = \begin{bmatrix} s^2+4s+3 \\ s^2+5s+6 \\ s^2+3s+2 \end{bmatrix}$$

$$= \mathbf{g}_O(s)$$

The result above shows that the zero positions of the open- and closed-loop ESI system are invariant to pole shifting, as predicted by the theory.

The element zeroes of the transfer function $\mathbf{g}_C(s)/d_C(s)$ are in this case:

$$s=-1,-1$$

From this follows, that there is no relationship between the open-loop zeroes of $\mathbf{g}_O(s)/d_O(s)$ which had in fact none, and the closed-loop zeroes of $\mathbf{g}_C(s)/d_C(s)$. This again emphasises the need to analyse the zeroes before cancellation can take place as mentioned before.

5.3.6.2 Multi-Input System Zeroes

To analyse the effect of dyadic pole positioning on the zeroes of the multivariable system the following functions need to be calculated:

For the open-loop case:

$$\mathbf{T}_O(s) = \mathbf{C}\ \mathrm{adj}(s\mathbf{I}-\mathbf{A})\ \mathbf{B}$$

and for the closed-loop multi-input (MI) system:

$$\mathbf{T}_C(s) = \mathbf{C}\ \mathrm{adj}(s\mathbf{I}-\mathbf{A}_C)\ \mathbf{B}$$

As before it is noteworthy, that the zeroes found from the equations above could cancel when forming the transfer function.

Choosing $\mathbf{q}^T=(1,1)$ as above, the open-loop zero positions are given by:

$$\mathbf{T}_O(s) = \begin{bmatrix} s^2+4s+3 & 0 \\ s^2+5s+6 & s^2+3s+2 \end{bmatrix}$$

i.e zeroes at s=-1,-3,-2,-3,-1,and -2

and the open-loop transfer function is:

$$\mathbf{G}_O(s) = \begin{bmatrix} 1/(s+2) & 0 \\ 1/(s+1) & 1/(s+3) \end{bmatrix}$$

Assigning the poles to

s= -2,-2,-3 as before

then the closed-loop zero positions will be given by:

$$\mathbf{T}_C{}^a(s) = \begin{bmatrix} s^2+4s+3 & 0 \\ s^2+4s+4 & s^2+4s+4 \end{bmatrix}$$

$$\dotplus\ \mathbf{T}_O(s)$$

The new zero positions are at:

s= -1,-3,-2,-2,-2 and -2

The resulting transfer function will be:

$$\mathbf{G}_C{}^a(s) = \begin{bmatrix} (s+1)/(s+2)^2 & 0 \\ 1/(s+3) & 1/(s+3) \end{bmatrix}$$

The pole assignment method has not only shifted the poles, but also transformed element $\mathbf{G}_{O(1,1)}(s)$ from a first order system to a second order system.

Using $\mathbf{q}^T=(1,1)$ again and assigning the poles to:

s= -1,-3,-3

then

$$\mathbf{T}_C{}^b(s) = \begin{bmatrix} s^2+4s+3 & 0 \\ s^2+4s+5 & s^2+4s+3 \end{bmatrix}$$

$$\neq \mathbf{T}_C{}^a(s)$$

This design will move the zeroes to:

s= -1,-3,-2±j,-1 and -3

The closed-loop transfer function is now:

$$\mathbf{G}_C{}^b(s) = \begin{bmatrix} 1/(s+3) & 0 \\ (s+2-j)(s+2+j)/\{(s+1)(s+3)^2\} & 1/(s+3) \end{bmatrix}$$

Assigning poles to:

s= -1,-3,-3 as above

and using $\mathbf{q}^T=(-1,1)$ instead, which also assures controllability results in closed-loop zero positions given by:

$$\mathbf{T}_C{}^C(s) = \begin{bmatrix} s^2+4s+3 & 0 \\ s^2+6s+7 & s^2+4s+3 \end{bmatrix}$$

The zeroes are now moved to:

s= -1,-3,-4.4,-1.6,-1 and -3

The closed-loop transfer function becomes:

$$\mathbf{G}_C{}^C(s) = \begin{bmatrix} 1/(s+3) & 0 \\ (s+4.4)(s+1.6)/\{(s+1)(s+3)^2\} & 1/(s+3) \end{bmatrix}$$

This example has shown that the zeroes of the ESI system are set by the vector **q** only, but the zeroes of the MI system change as **q** and/or pole positions are changed. In addition, it was shown that the closed-loop transfer function change considerably to accommodate the new pole positions.

5.3.7 DIFFICULTIES OF METHOD

This section will elaborate on the implications of choosing a specific **q** and reasons for moving the poles. Additionally, the limitations associated with the selected pole positions and with the pole assignment technique in general are discussed.

5.3.7.1 Choice of vector q

The **q** vector is required to transform the MI system to an ESI system. The theory applied above implies that for a

specific vector **q** and different closed-loop pole positions the open- and closed-loop zeroes remain the same, i.e.:

$$\mathrm{Adj}(s\mathbf{I}-\mathbf{A})\,\mathbf{Bq} = \mathrm{Adj}\{s\mathbf{I}-(\mathbf{A}-\mathbf{bf}^{T})\}\,\mathbf{Bq} \tag{5.55}$$

but this does not apply for the multivariable case since:

$$\mathrm{Adj}(s\mathbf{I}-\mathbf{A})\,\mathbf{B} \neq \mathrm{Adj}\{s\mathbf{I}-(\mathbf{A}-\mathbf{bf}^{T})\}\,\mathbf{B} \tag{5.56}$$

and

$$\mathbf{C}\,\mathrm{Adj}(s\mathbf{I}-\mathbf{A})\,\mathbf{B} \neq \mathbf{C}\,\mathrm{Adj}\{s\mathbf{I}-(\mathbf{A}-\mathbf{bf}^{T})\}\,\mathbf{B} \tag{5.57}$$

This shows that the zeros of the MI system will change in a complicated way.

It follows that choosing **q** will fix the closed-loop zeroes of the ESI system irrespective of pole position, i.e only the selection of **q** determines the closed-loop zeroes. But this is not the case for the MI system.

From Eqn. (5.15) it follows that **q** distributes the scalar state feedback quantity $\mathbf{f}^{T}\mathbf{x}$ among the plant inputs. If the plant inputs have different ranges, **q** can be used to scale the feedback quantity between the various inputs. Each input is then affected by the feedback in the same proportion.

5.3.7.2 Closed-Loop Zeroes

The zeroes of the closed-loop MI system are not invariant and cannot be fixed by a specific **q**, instead the closed-loop zeroes are determined by both the choice of **q** and the assigned pole positions. The actual movements of the zeroes from the open- to the closed MI system are complicated and a general formulation of the movement is laborious.

The closed-loop system may as a result have zeroes in the open right half plane resulting in a system with non-minimum

phase. The system will then have a negative response initially to a positive step input in a sense opposite to the response expected. This behaviour is similar to that of positive feedback and it will increase the difficulty of designing an appropriate control scheme for that system.

5.3.7.3 The Choice Of Pole Positions

The decisions that are involved in specifying the closed-loop pole positions cannot be designated by a set of fixed rules. In general **K** is chosen such that the dominant poles of the closed-loop system have good damping factors and lie sufficiently far to the left of the origin, resulting in a well-behaved, quickly decaying dynamic response.

5.3.7.4 Limitations In Pole Selection

Although **K** can be chosen to move the poles of the closed-loop system to any position, this is in practice not possible. If the poles are moved by too large an amount the elements of matrix **K** are found to be large thus saturating the inputs to the plant. In addition small parameter changes of the plant result in large changes in the closed-loop pole positions. Furthermore, the steady-state gains of the closed-loop system, which are related to the zeroes produced by the matrix **K**, may be too large and cause saturation of the system outputs.

5.3.7.5 Resulting Noise Sensitivity

An important factor governed by the gain of the feedback matrix **K** is the system sensitivity to noise. If the state measurements are contaminated with noise and the feedback

gains are too large, the system inputs will be driven by noise thus impairing the control system considerably, or even resulting in a so-called Bang-Bang controller. These problem can avoided by keeping the elements of the **K** matrix sufficiently small and by proper tuning of an observer, which is discussed in section 5.4.

5.3.7.6 System Interaction

The resulting interaction between elements of the closed-loop system from a choice of **K** cannot be formally predicted by the theory and may also increase the problem of finding a suitable control scheme. Decoupling is possible in pole assignment(**21,22**) by designing an additional input compensator.

5.3.7.7 Closed-Loop Transient Response

The example given in section 5.3.6 has shown that the resulting transfer function will from a dynamic point of view be governed by the assigned pole positions. It is very difficult however, to deduce the transient response to step inputs from the choice of **q** and pole positions. The transient response may in fact be unsatisfactory.

The design of a suitable feedback configuration is thus a matter of trial and error and must be combined with a simulation of the closed-loop system to analyse the transient response to step inputs and from it deduce the effects of the feedback matrix **K** on the system.(**23**)

5.4 OBSERVER DESIGN

The pole assignment technique discussed above assumes that all the system states are available for measurement. In most multivariable plants this is not the case and the pole assignment design must include a state estimator. An observer can always be designed to estimate all the system states if the system is fully observable. If the system is not completely observable, then only some states can be estimated limiting the scope of pole selection.

For most practical applications, where the available measurements are contaminated with noise it is found that the Luenberger observer is unsatisfactory and the discussion will focus instead on the Kalman Bucy filter.(**13**)

5.4.1 THE KALMAN FILTER

The Kalman filter is a state estimator, that is usually used for sampled stochastic systems utilising digital computer control(**24**). Many different types of Kalman filters(**25**) are available and they have been successfully applied for linear time-invariant, linear time-variant and also nonlinear system(**26**).

5.4.2 SAMPLED DATA STOCHASTIC SYSTEMS

Sampled data stochastic systems appear in most digital control schemes where the measurements are made at discrete time intervals and the systems are contaminated by noise.

A state space description of a linear time invariant stochastic system is:

$$\dot{\mathbf{x}} = \mathbf{Ax}+\mathbf{Bu}+\mathbf{w}$$

$$\mathbf{y} = \mathbf{Cx} + \mathbf{v} \qquad (5.58)$$

where the covariance of the system input white noise with zero mean is:

$$\text{cov}(\mathbf{w}) = E[\mathbf{w}\ \mathbf{w}^T] = \mathbf{Q}\ \delta(\tau) \qquad (5.59)$$

and the covariance of the measurement white noise with zero means is:

$$\text{cov}(\mathbf{v}) = E[\mathbf{v}\ \mathbf{v}^T] = \mathbf{R}\ \delta(\tau) \qquad (5.60)$$

The Kronecker Delta function is denoted by $\delta(\tau)$ and **Q** and **R** are the system noise and the observation noise covariance matrices respectively.

The Kalman filter requires that the measurement and the system input noise is uncorrelated,i.e.:

$$E[\mathbf{w}\ \mathbf{v}^T] = 0$$

The corresponding state equations for the sampled data system, sampled at period T, are as follows:

$$\mathbf{x}(n+1) = \boldsymbol{\phi}\mathbf{x}(n)+\boldsymbol{\Phi}\mathbf{u}(n)+\mathbf{w}_d(n)$$

$$\mathbf{y}(n) = \mathbf{Cx}(n)+\mathbf{Du}(n)+\mathbf{v}_d(n) \qquad (5.61)$$

where

$$\text{cov}(\mathbf{w}_d(n)) = E[\mathbf{w}_d(n)\mathbf{w}_d^T(n)] = \mathbf{Q}_d\ \delta(\tau) \qquad (5.62)$$

$$\text{cov}(\mathbf{v}_d(n)) = E[\mathbf{v}_d(n)\mathbf{v}_d^T(n)] = \mathbf{R}_d\ \delta(\tau) \qquad (5.63)$$

The relationship between the discrete and continuous system(**5**) is:

$$\boldsymbol{\phi} = e^{\mathbf{A}T} \approx \mathbf{I} + \mathbf{A}T \qquad (5.64)$$

$$\Phi = \int_0^T e^{\mathbf{A}\tau}\,\mathbf{B}\,d\tau \approx \mathbf{B}T \tag{5.65}$$

$$\mathbf{Q}_d = \int_0^T e^{\mathbf{A}\tau}\,\mathbf{Q}\,(e^{\mathbf{A}\tau})^T\,d\tau \approx \mathbf{Q}T \tag{5.66}$$

$$\mathbf{R}_d = \mathbf{R} \tag{5.67}$$

The approximations shown above only hold for short sample periods T compared to the time constants associated with the matrix A.

5.4.3 KALMAN FILTER EQUATIONS

There are many different ways to implement the Kalman filter(**27**). The following asymptotic state estimator equations were used for linear time invariant (LTI) systems to implement the Kalman filter.

The system must be observable, allowing all states to be estimated, i.e.:

$$O = (\mathbf{C}^T\ (\mathbf{CA})^T\ (\mathbf{CA}^2)^T\ \ldots\ (\mathbf{CA}^{n-1})^T)$$

has rank n.

then the discrete-time model is:

$$\hat{\mathbf{x}}(n+1) = \mathbf{\Phi}\hat{\mathbf{x}}(n) + \mathbf{M}(n)\,\{\mathbf{y}(n)-\mathbf{C}\hat{\mathbf{x}}(n)-\mathbf{Du}(n)\} + \mathbf{\Phi u}(n) \tag{5.68}$$

$$\mathbf{P}(n) = \mathbf{R}_d + \mathbf{C}\,\mathbf{G}(n)\,\mathbf{C}^T \tag{5.69}$$

$$\mathbf{M}(n) = \mathbf{\Phi}\,\mathbf{G}(n)\,\mathbf{C}^T\,\mathbf{P}(n)^{-1} \tag{5.70}$$

$$\mathbf{G}(n+1) = \{\mathbf{\Phi}-\mathbf{M}(n)\mathbf{C}\}\,\mathbf{G}(n)\,\mathbf{\Phi}^T + \mathbf{Q}_d \tag{5.71}$$

where

$\hat{\mathbf{x}}(n+1)$ is the estimate of the actual states $\mathbf{x}(n+1)$.

The matrix **M** is termed the *Kalman filter gain* and **G** is referred to as the *Error covariance matrix*. The initial value of **G**(0) can be assumed to be zero. If the value of **G**(0) is known more accurately, it can be used to reduce the duration of the initial transient.

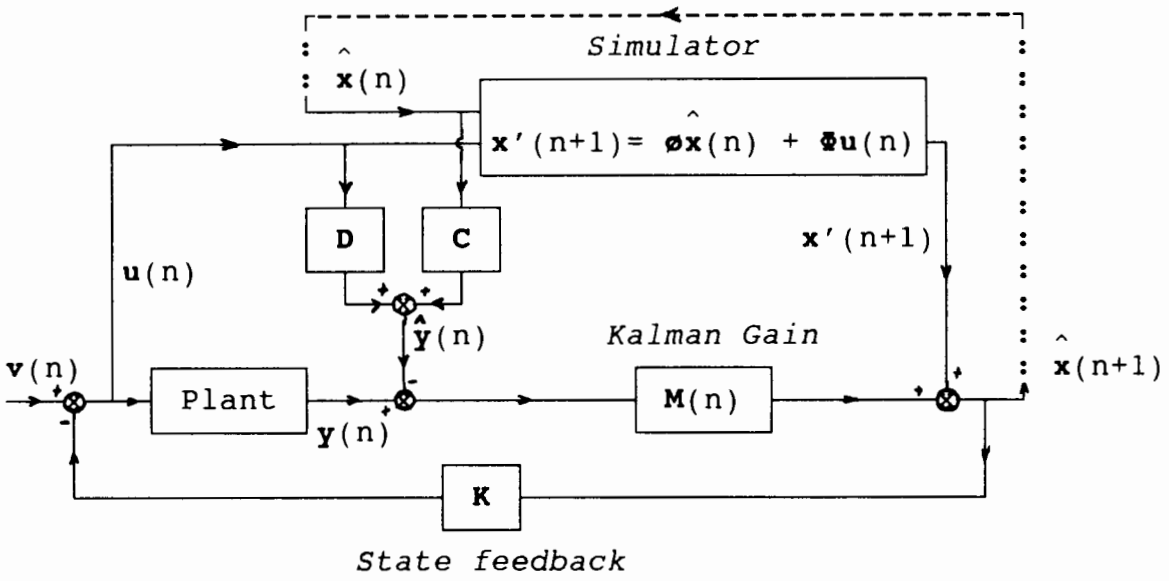


FIG. 5.5 State Feedback System with Kalman Filter

Fig. 5.5 shows the system configuration in block diagram form to implement pole assignment with the use of a Kalman filter. The diagram includes a simulator that calculates the new system states $\mathbf{x}'(n+1)$ using the previous states $\hat{\mathbf{x}}(n)$ and the current plant input $\mathbf{u}(n)$. The evaluated states are then corrected through the Kalman gain matrix to produce the new state estimates $\hat{\mathbf{x}}(n+1)$, which will replace $\hat{\mathbf{x}}(n)$ on the next cycle. The amount by which the states are corrected depends

on the error between the estimated $\hat{\mathbf{y}}(n)$ and actual plant outputs $\mathbf{y}(n)$.

From Eqs. (5.69), (5.70) and (5.71) it can be seen that the matrix $\mathbf{M}(n)$ develops purely from the covariance matrices $\mathbf{Q}_d$ and $\mathbf{R}_d$, and will approach limiting values independent of $\mathbf{G}(0)$ if the LTI system is stable. It follows that matrix $\mathbf{M}(n)$ can be calculated off-line and only Eqn. (5.68) needs to be computed at each iteration $n = 0,1,2...\infty$. This is only true for LTI systems and in the case of time variant system matrices $\mathbf{P},\mathbf{M},\mathbf{G}$ must be computed for each step.

It is important to note, that both $\mathbf{R}_d$ and $\mathbf{Q}_d$ may not be zero, because then the algorithm fails to find an optimal matrix $\mathbf{M}(n)$. The matrix $\mathbf{Q}_d$ may be zero, but then $\mathbf{G}(0)$ must contain some non-zero elements.

5.4.4 ESTIMATION OF VARIANCES

The difficulty of designing a Kalman filter lies in the estimation of the covariance matrices $\mathbf{Q}$d and $\mathbf{R}$d.(**28,29**) Both these matrices are usually assumed to be diagonal, as the individual noise signals are generally uncorrelated. The coefficients then represent the variance or amplitude of the noise signals affecting the states and outputs of the system.

In most cases it is simple to find $\mathbf{R}$d because the variance or uncertainty of the measurements can be deduced from the inaccuracies of the instruments used. Alternatively, $\mathbf{R}_d$ can be calculated from a series of measurements while the external noise input is removed or sufficiently reduced.

Finding the matrix $\mathbf{Q}$d however, poses a problem because it is difficult to deduce which states are contaminated by noise

and to estimate the magnitude of the noise signals. A detailed discussion on the calculation of the variance matrices, if they are known for the continuous case, has been given by Brown.(**30**)

An alternative approach is related to the design of Luenberger observers(**5**). After having chosen some matrices **Rd** and **Qd** iterate Eqs. (5.69), (5.70) and (5.71) until matrix **M**(n) reaches its limiting values **M** and then the response of the estimator is governed by

$$\dot{\mathbf{x}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{M}(\mathbf{y}-\mathbf{C}\hat{\mathbf{x}})$$
$$= (\mathbf{A}-\mathbf{MC})\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{M}\mathbf{y} \qquad (5.72a)$$

Where the 'error' of the estimator behaves according to:

$$(\dot{\mathbf{x}}-\dot{\hat{\mathbf{x}}}) = (\mathbf{A}-\mathbf{MC})(\mathbf{x}-\hat{\mathbf{x}}) \qquad (5.72b)$$

The matrices **Rd** and **Qd** should be selected to ensure that the eigenvalues of (**A-MC**) result in transient responses of the estimator errors that are stable and well-behaved.

If in addition, the time constants of the estimator error equation are kept sufficiently fast to allow the estimator to track the system states, but yet slow enough to be ignorant to system noise, the resulting filter should be adequate.

An intuitive approach to estimate the system input covariance matrix for the flotation rig in particular, will be discussed in detail in chapter six. This method uses a simulation of the plant to estimate the variances of the system noise input vector **w**. A white noise signal, with specific variance, is applied to a plant simulator and the variance of the output is computed. The variance of the

input signal is then adjusted until the simulated output variance corresponds to that found at the plant. The input variance so found is then used to develop Q_d.

5.5 CONCLUSION

Although greater degrees of freedom are available in the design of pole assignment in multivariable than in single-variable system it is very difficult to take advantage of them in a formal way, because of the problems associated with the zero locations, steady state gain, and interaction of the resulting closed-loop system.

It was also shown that the amount by which the poles can be shifted in practice is severely limited.

A further disadvantage of the pole assignment technique is the need to design an observer to estimate the system states, which is usually necessary for multivariable systems. The choice of the system model is thus critical not only for the pole assignment design, but also for the estimation of the states.(**31**) If the model used by the observer is unsatisfactory then the pole assignment method will not operate properly.

6. APPLICATION OF POLE ASSIGNMENT

6.1 INTRODUCTION

Application of the theory on pole assignment and Kalman filter design, which has been discussed previously in chapter five is discussed. The application is specific to the flotation rig and aimed at controlling the Rougher and Scavenger tanks in particular.

A Kalman filter is designed to estimate the states of the two-tank system, because they could not be measured directly from the plant. An intuitive method to estimate the covariance matrices by simulation techniques is developed.

The pole-assignment technique is applied to the system by state feedback to improve its speed of response.

6.2 FLOTATION RIG MODEL

The state-space description of the two-tank model is constructed from chapter two, where

$$\dot{\mathbf{x}}(s) = \mathbf{A}\mathbf{x}(s) + \mathbf{B}\mathbf{u}(s)$$

$$\mathbf{y}(s) = \mathbf{C}\mathbf{x}(s) \qquad (6.1)$$

with

$$\mathbf{A} = \begin{bmatrix} -3.95E\text{-}2 & -5.9E\text{-}4 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -4.07E & -2.20E\text{-}4 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.87E\text{-}2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.19E\text{-}2 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and

$$\mathbf{C} = \begin{bmatrix} 1.40E\text{-}2 & 6.83E\text{-}5 & 0 & 1.53E\text{-}4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.22E\text{-}2 & 5.97E\text{-}3 \end{bmatrix}$$

In this model $\mathbf{y}_1$ and $\mathbf{y}_2$ refer to the Rougher and Scavenger tank levels respectively, $\mathbf{u}_1$ and $\mathbf{u}_2$ corresponds to the Rougher and Scavenger valves respectively.

6.3 A KALMAN FILTER FOR THE FLOTATION RIG

It was necessary to design a Kalman filter for the flotation rig before applying the pole assignment technique, because the states of the plant could not be measured from it directly.

The model must include noise inputs for the design of the Kalman filter.

The model describing the flotation rig with system noise vector **w** and measurement noise vector **v** is then

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} + \mathbf{w}$$

$$\mathbf{y} = \mathbf{Cx} + \mathbf{v} \qquad (6.2)$$

6.3.1 CALCULATION OF SAMPLE MEAN AND VARIANCE

At this point it is necessary to recall how the variance of a collection of measurements can be found.(**6**)

The mean value of a sample $x_1, x_2, \ldots x_n$ is defined by the formula:

$$\bar{x} = \frac{1}{n} \sum_{j=1}^{n} x_j \qquad (6.3)$$

The variance of a sample is given by:

$$\sigma^2 = \frac{1}{n-1} \sum_{j=1}^{n} (x_j - \bar{x})^2 \qquad (6.4)$$

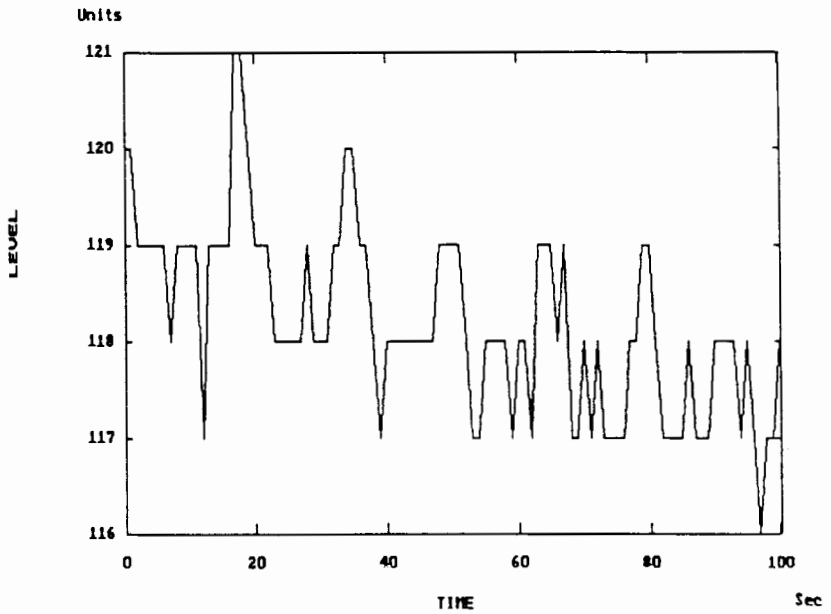
where $\sigma = \sqrt{\sigma^2}$ is the standard deviation of the sample.

6.3.2 ESTIMATION OF MEASUREMENT NOISE MATRIX

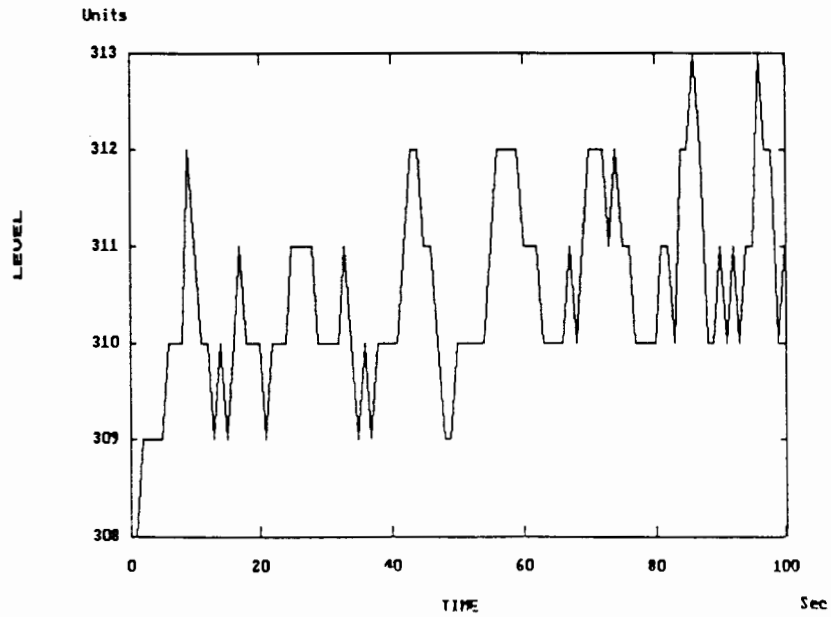
The level of each tank was observed under stationary conditions to find the measurement co-variance matrix **R**. The Rougher and Scavenger tanks were filled with water. All pumps were switched off and all valves closed. When the levels **y** had settled down, a number of measurements of the levels were taken to estimate **v**.

A time-series of the observed level measurements of the Rougher and Scavenger tanks are shown in Fig. 6.1. It can be

seen, that the level transducer signals are quantized and were hardly contaminated by noise. The variation in the measurements are mainly due to quantization.



a) Rougher Tank



b) Scavenger Tank

FIG. 6.1 Level measurements under stationary conditions

The variance of the measurements are calculated using Eqn. (6.4), and found to be

$$\sigma^2_{Rougher} = 0.98 \approx 1 \text{ Units}^2$$
$$\sigma^2_{Scavenger} = 0.90 \approx 1 \text{ Units}^2 \qquad (6.5)$$

The measurement covariance matrix is then simply

$$\mathbf{R}_d = \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (6.6)$$

The off-diagonal elements of $\mathbf{R}_d$ are zero, because the quantization error of the Rougher measurement is uncorrelated with that of the Scavenger.

6.3.3 ESTIMATION OF SYSTEM INPUT NOISE MATRIX

It was relatively easy to find the measurement noise covariance matrix **R**. To estimate the system input noise covariance matrix **Q** from a theoretical basis does however pose a problem. Instead of using an explicit approach, the solution can be found from the following intuitive treatment of the problem, which was developed specifically for the flotation rig.

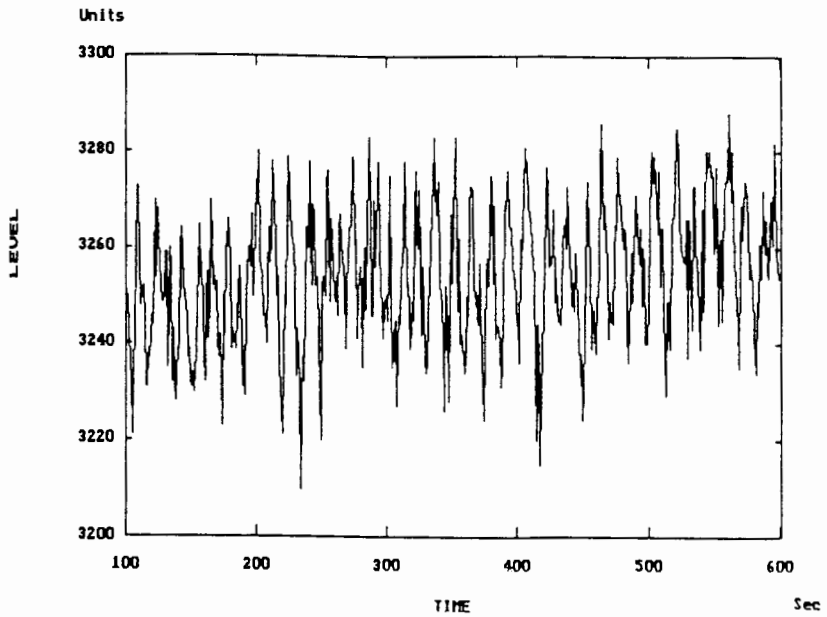
The system input noise covariance matrix **Q** is estimated in the following manner. Firstly, the plant output noise variance is determined from a set of measurements done on the plant during normal operation at steady-state. From it the measurement noise variance found previously is subtracted and an input noise variance resulting in that output variance is determined by simulation.

The input noise variance so obtained can then be used to estimate the covariance matrix **Q**.

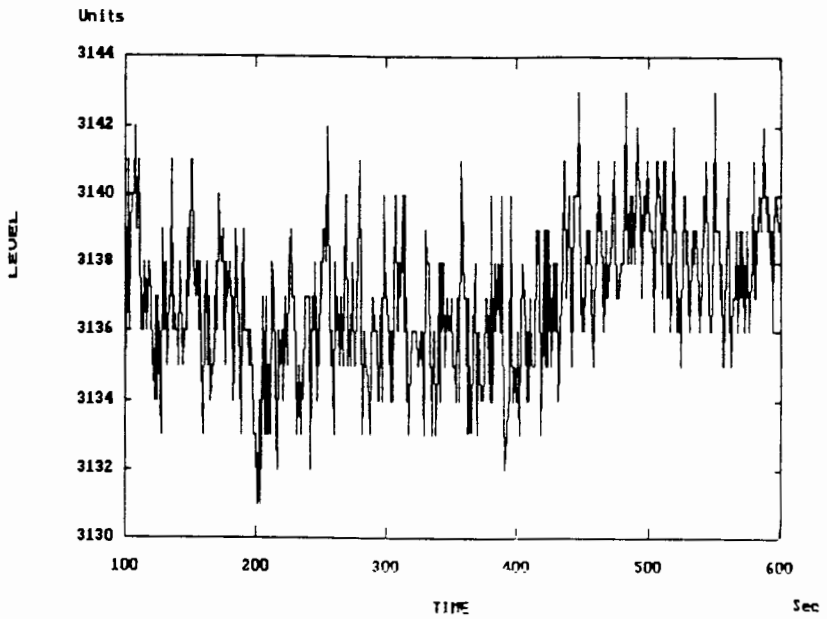
6.3.3.1 Determination of Plant Output Variance

To estimate the system output noise variance, measurements of the levels of the Rougher and Scavenger tanks at steady-state were taken. The levels of all tanks were brought to their operating levels by the use of the single-loop PI controller as discussed in chapter two. When the tanks attained the desired levels the valves were fixed at their current setting.

After a period of about 10 minutes had elapsed to allow the rig to settle down and reach steady-state, a number of measurements of the Rougher and Scavenger Tanks were taken. The time-series for these measurements, which were done at 1 second intervals, are shown in Fig. 6.2.



a) Rougher Tank



b)Scavenger Tank

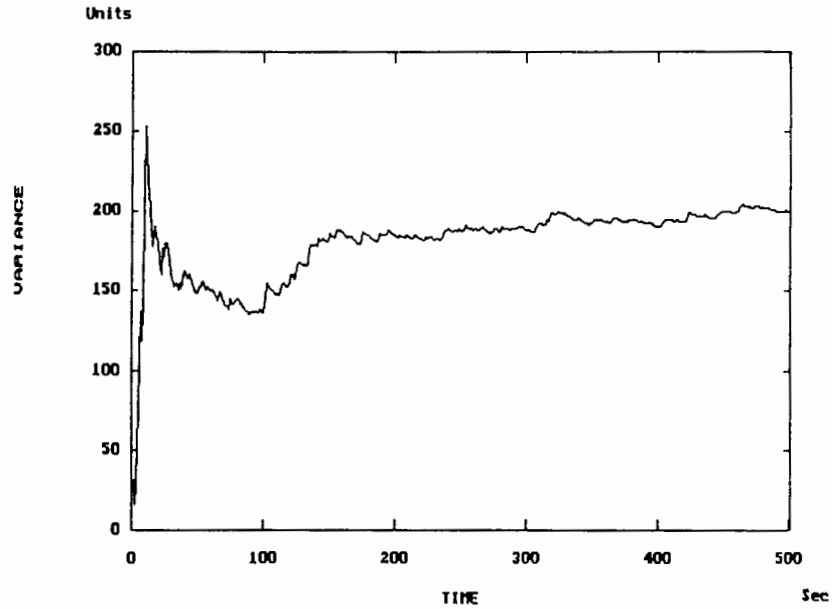
FIG. 6.2 Steady-state level measurements at operating levels

The variance of the above measurements, which now include the system input noise, is calculated and found to be:

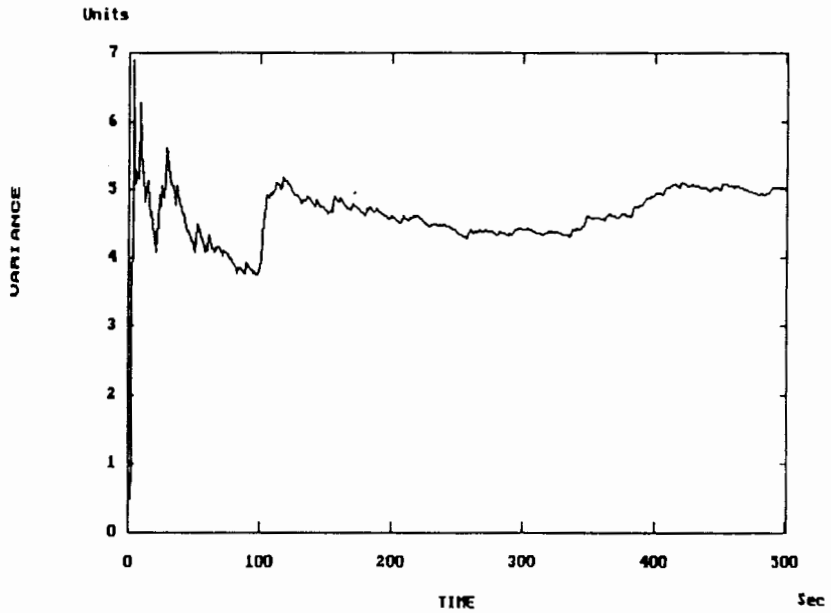
$$\sigma^2_{Rougher} = 199 \text{ Units}^2$$
$$\sigma^2_{Scavenger} = 5.20 \text{ Units}^2 \qquad (6.7)$$

It is important to determine whether the variances found above do converge. If they do not converge then this will be an indication that either too few samples were taken or that the levels drifted. If the levels did indeed drift then the plant had not been at steady state, because the plant is open-loop stable and in that case the variances found above would not be correct.

The variances of the level measurements are calculated at each sampling interval using the past samples and a time-series of the variance versus time (or number of samples) for both the Rougher and Scavenger outputs are shown in Fig. 6.3.



a) Rougher Tank



b) Scavenger Tank

FIG. 6.3 Calculation of variances

From Fig. 6.3, it can be seen that the respective variances do indeed converge to limiting values as the number of samples increases.

Assuming that the input and measurement noise signals are uncorrelated, then the respective variances, due to input noise only are

$$\sigma^2_{Rougher} = 199\text{-}1 \approx 200 \text{ Units}^2$$
$$\sigma^2_{Scavenger} = 5.2\text{-}1 \approx 4 \text{ Units}^2 \qquad (6.8)$$

To justify the above calculation, it is important to note that the variance of the sum of independent random variables equals the sum of the variances of these variables.(**6**)

6.3.3.2 System Simulation With White Noise Inputs

Having established the output variances that are due to the system white noise **w**(s) vector only, it is now necessary to estimate the variances of **w**(s) itself, that would result in the system output having the variances given in Eqn. (6.8). To derive this analytically from the system model is cumbersome as mentioned before and instead the following technique is applied.

The plant is simulated on a digital computer. A white noise signal of given variance is then applied to the individual inputs **u** of the system and the noise on the corresponding system output **y** is determined. These are then compared to that of the real plant and the variance of the input noise is adjusted until the variance of the output of the simulated plant is approximately the same as that of the real plant.

To perform the simulation the noise signals of the plant are included in the model in the following way:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \mathbf{A} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_{ro} \\ u_{sc} \end{bmatrix} + \begin{bmatrix} w_1(\sigma_{ro}{}^2) \\ 0 \\ 0 \\ 0 \\ 0 \\ w_6(\sigma_{sc}{}^2) \end{bmatrix} \qquad (6.9)$$

where $\sigma_{ro}{}^2$ and $\sigma_{sc}{}^2$ are the variances of the uncorrelated white noise inputs to the Rougher and Scavenger tank levels respectively.

For the ensuing discussion, it is necessary to recall that states x_1 and x_2 are related to the Rougher level, x_3-x_5 correspond to the interaction of the two tank, and x_6 is related to the Scavenger level.

The above model is not a true account of the real plant because the following simplifications were performed.

In the model (6.9) above, the fluctuations of the Rougher tank level are due to a single noise input (w_1) only. In a similar fashion the Scavenger level is only affected by the noise input w_6. Both noise signals do not affect the interaction between the two tanks.

For the real plant, it can be argued that the noise injected onto the Rougher would affect the level of the Scavenger, but a more precise model of the plant would have resulted in an unnecessarily complicated description. The results, that

will be obtained later, will show, that the model was in fact adequate.

6.3.3.3 Simulation Of The Normal Distribution

Having established the model that is to be used to simulate the plant with noise inputs, it is now necessary to derive a method to simulate those white noise inputs numerically.

The pseudo-random-number generators that are available for personal computers through *BASIC* or *TURBO PASCAL*, produce sequences of random numbers, that are uniformly distributed from 0 to 1. Hansen(**32**) has shown that by summing these numbers a Gaussian distribution can be attained.

The function *random* in *TURBO PASCAL* produces real numbers x_i such that

$$0 \leq x_i < 1 \tag{6.10}$$

The fact that $x_i=1$ is excluded is of little consequence and can be ignored.

The mean value of x_i is then

$$\bar{x} = \tfrac{1}{2} \tag{6.11}$$

To obtain a normal distribution, the sum S of n such random numbers is computed

$$S = \sum_{i=1}^{n} x_i \tag{6.12}$$

then the mean value of S is

$$\bar{S} = \tfrac{1}{2}n \tag{6.13}$$

with the maximum deviation from the mean for any sample

$$\text{deviation}_{max} = \tfrac{1}{2}n \qquad (6.14)$$

and the corresponding variance is

$$\sigma^2 = 1/(12n) \qquad (6.15)$$

It can be seen from Eqn. (6.14) that n determines the maximum deviation from the mean for any sample. It is therefore necessary to choose n sufficiently large to achieve a wide spread of samples. In addition, as n is increased, the distribution of the simulated sequence will tend to that of a pure Gaussian distribution by the Central Limit Theorem(**6**).

To obtain a random sample r_σ with specified variance σ^2, the following algorithm is used:

A) The sum of n random number is computed:

$$S = \sum_{i=1}^{n} x_i$$

where x_i is obtained from the internal function *random*.

B) Adjusting the sum to have a mean of zero as follows:

$$S_0 = S/n - 0.5$$

C) The sample r_1 with unity variance is computed from

$$r_1 = S_0 \sqrt{(12n)}$$

D) A random sample r_σ with variance σ^2 is obtained from

$$r_\sigma = \sigma r_1 \qquad (6.16)$$

Steps C) and D) are valid statistically and have been proved elsewhere.(**6**)

The procedure above is repeated to obtain a sequence of normally distributed numbers with zero mean and with fixed variance.

A value of n=50 is used for the simulation done below. This value results in an acceptable distribution and computational times for the simulation are tolerable.

The following test sequence was performed, initially for the Rougher tank and then for the Scavenger tank.

The input noise variance was selected arbitrarily and used to generate normally distributed samples. The actual input noise variance was found from the samples so generated. These samples were then applied to the simulated tank noise inputs. The variance of the resulting tank level was then found and tabulated.

This procedure was repeated for each selected input variance, to give an indication of the repeatability of the experiments. The input variance was then increased until the output variance was similar to that of the real plant.

The noise input variances corresponding to a system output variance as found in Eqn. (6.8) can be read from Tables (6.1),(6.2) and estimated.

Input Noise Variance	Actual Input Noise Variance	Output Variance
1	0.980	0.00177
1	1.020	0.00174
1,000	975.000	2.73000
1,000	1,067.000	1.78000
100,000	97,800.000	341.00000
100,000	95,400.000	247.00000

TABLE 6.1 Rougher tank simulation results

Input Noise Variance	Actual Input Noise Variance	Output Variance
1	0.98	0.00177
1	1.02	0.00174
1,000	975.00	2.73
1,000	1,067.00	1.78
100,000	97,800.00	341.00
100,000	95,400.00	247.00

TABLE 6.2 Scavenger tank simulation results

The variance of the system noise signal w_1 is thus

$$\sigma_{ro}^2 \approx 100{,}000$$

and that for w_6 is

$$\sigma_{sc}^2 \approx 10{,}000 \qquad (6.17)$$

The input noise signals were assumed to be uncorrelated and a sampling period τ of 1 second is used. The system noise covariance matrix is then

$$\mathbf{Q}_d \approx \mathbf{Q}\tau = E(\mathbf{w}\,\mathbf{w}^T)\tau = \begin{bmatrix} 100{,}000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10{,}000 \end{bmatrix} \qquad (6.18)$$

6.3.4 KALMAN GAIN MATRIX

The resulting Kalman gain matrix was calculated using the equations from chapter five.

After 414 iterations the gain matrix converged with 4 digit accuracy to

$$\mathbf{M} = \begin{bmatrix} 65.40 & 0 \\ 71.14 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 72.85 \end{bmatrix} \qquad (6.19)$$

6.3.5 DYNAMIC RESPONSE OF FILTER

From matrix **M** above, it can be seen that only states $\mathbf{x}_1$,$\mathbf{x}_2$ and $\mathbf{x}_6$ will be corrected by the Kalman filter. The remaining states are estimated solely from a plant simulation. This is as expected because of the block-diagonal structure of matrix **A** and the format of the chosen noise vector **w**.

The dynamic behaviour of the estimator 'error' can be predicted from the eigenvalues of (**A-MC**).(**4**)

$$\mathbf{A\text{-}MC} = \begin{bmatrix} -0.9550 & -0.00506 & 0 & -0.01000 & 0 & 0 \\ 0.00404 & -0.00486 & 0 & -0.01090 & 0 & 0 \\ 0 & 0 & -0.0407 & -0.00022 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.0187 & 0 \\ 0 & 0 & 0 & 0 & 0.8890 & -0.447 \end{bmatrix}$$

with eigenvalues in rad/sec corresponding to states $\mathbf{x}_i$ are at

$$s_1 = -9.5508E\text{-}01 + 0j$$

$$s_2 = -4.8815E\text{-}03 + 0j$$

$$s_3 = -3.4283E\text{-}02 + 0j$$

$$s_4 = -6.4172E\text{-}03 + 0j$$

$$s_5 = -1.8700E\text{-}02 + 0j$$

$$s_6 = -4.4681E\text{-}01 + 0j \qquad (6.20)$$

It can be seen that all eigenvalues have real parts only. The dynamic response of the filter 'error' will therefore be well-behaved.

Comparing the eigenvalues above to those of Eqn. (6.21) shows that the states $\mathbf{x}_3$-$\mathbf{x}_5$ have identical time-constants as those of the model, which is to be expected. The time-constants associated with states $\mathbf{x}_1$,$\mathbf{x}_2$ and $\mathbf{x}_6$ have however changed drastically. The states $\mathbf{x}_1$ and $\mathbf{x}_6$ are estimated with time-constants that are faster than those of the model by a factor of 50, which is excellent. The eigenvalue associated with state $\mathbf{x}_2$ is however very small compared to that of the model. From a point of asymptotic observer design, it can be argued that this fact may impede the performance of the Kalman filter, because the settling time for the estimation of $\mathbf{x}_2$ if the plant remains at steady-state is about

$4/4.88E\text{-}3 \approx 820$ seconds.

Because the true states of the plant are unknown, it is difficult to ascertain how well the Kalman filter operates. To judge the filter's success by comparing the estimated with the actual output levels is ineffective because the algorithm is based on correcting the states based on that information. It is thus not possible to determine from an application of the estimator alone, if the large time-constant does in fact impede its performance. All that could be determined in this manner is whether the states reach limiting values, if the tank is in steady-state.

As the pole assignment technique by state feedback relies on the correct estimation of the system states, the employment of pole assignment can be used not only to discuss its success, but also the quality of the Kalman filter.

6.4 APPLICATION OF POLE ASSIGNMENT

This section will demonstrate the application of pole assignment on the laboratory flotation rig. The main purpose of the discussion will be to show the difficulty of determining gain margins and to what extent the simulated response will correspond to that of the real plant. The results that will be obtained can then be used to judge the performance of pole assignment on a real plant.

6.4.1 OPEN-LOOP POLE CONSIDERATIONS

The eigenvalues of the matrix **A** in the plant model, give an indication of the dynamic response of the system. As pole assignment is primarily concerned with moving these eigenvalues to 'better' positions, the open-loop pole positions have to be analysed first.

The eigenvalues in rad/sec of **A** that are related to the ith state are found to be:

s_1 = -1.9750E-02 - 1.4175E-02j

s_2 = -1.9750E-02 + 1.4175E-02j

or

s_2 = -1.9750E-02 - 1.4175E-02j

s_1 = -1.9750E-02 + 1.4175E-02j

and

s_3 = -3.4283E-02 + 0j

s_4 = -6.4172E-03 + 0j

or

s_4 = -3.4283E-02 + 0j

s_3 = -6.4172E-03 + 0j

and

s_5 = -1.8700E-02 + 0j

s_6 = -1.1900E-02 + 0j (6.21)

The above results are obtained from the eigenvalue routine listed in the *Software Reference*. Due to the block diagonal structure of matrix **A**, the eigenvalues could be related to specific states. For a general **A** matrix this is not possible.

Analysis of the above eigenvalues shows that the open-loop poles all in the left-half of the s-plane which ensures stability. This was expected because the plant showed no signs of instability at the chosen operating levels. Unfortunately, the main feature of pole assignment, namely

to improve stability, can thus not be demonstrated in this case.

The poles can however be moved further away from the origin to improve the stability margin. If the inherent nonlinearities of the rig, which have been ignored in the model, should move these poles nearer to the origin, a larger stability margin will be advantageous.

In addition, moving the poles away from the origin will also improve the speed of the dynamic response of the system. As the flotation rig is a very slow process, with time-constants ranging from 1 to 3 minutes [see Eqn. 6.21], it is desirable to improve its speed of response.

A further improvement on the pole positions can be achieved by forcing the poles to be real. Although no oscillations were detected from the plant step responses, it will be better to make the imaginary parts of the complex poles as small as physically possible.

The following approach will be used to analyse the performance of pole assignment. The imaginary parts of the eigenvalues are reduced by a factor of 10 and the poles are moved progressively further away from the origin.

No considerations are given to system performance criteria such as interaction and steady-state gain during the initial design, because these are difficult to predict from the selected vector $\mathbf{q}$ and pole positions, as has been shown previously in chapter five. It was in fact found that the closed-loop step responses were generally unpredictable, which becomes apparent from the results discussed later.

6.4.2 CONTROLLABILITY CRITERION

The **q**-vector was arbitrarily selected to be

$$\mathbf{q} = (1,1)^T \tag{6.22}$$

This choice fulfilled the controllability criterion, because the determinant of the controllability matrix **P** was found to be

$$|\mathbf{P}| = 1.3E\text{-}24 \neq 0 \tag{6.23}$$

This value is very small numerically, which is to be expected because the elements of **A** are in the order of 1/100. This does however show the limitations in the numerical technique to determine controllability. If the order of the model is very large and the elements of **A** are not ≈1, then the resulting determinant will either be too large and cause an overflow, or be too small and result in a value of zero. In both cases the calculated determinant of P will give no indication of controllability or otherwise.

6.4.3 CLOSED-LOOP POLE POSITIONS

Four different sets of pole positions were selected to increase the speed of response of the flotation rig progressively as depicted in Table 6.3 (An increase of speed by 100% corresponds to doubling the speed of response). O/L indicates the approximate open-loop system pole positions.

Speed	Closed-Loop Pole Positions
O/L ≈	(-2.00E-2,-1.50E-2),(-2.00E-2,+1.50E-2),(-3.50E-2,0),(-6.50E-3,0),(-1.90E-2,0),(-1.20E-2,0)
1: **10%**	(-2.20E-2,-1.50E-3),(-2.20E-2,+1.50E-3),(-3.85E-2,0),(-7.15E-3,0),(-2.09E-2,0),(-1.32E-2,0)
2: **20%**	(-2.40E-2,-1.50E-3),(-2.40E-2,+1.50E-3),(-4.20E-2,0),(-7.80E-3,0),(-2.28E-2,0),(-1.44E-2,0)
3: **50%**	(-3.00E-2,-1.50E-3),(-3.00E-2,+1.50E-3),(-5.25E-2,0),(-9.75E-3,0),(-2.85E-2,0),(-1.80E-2,0)
4:**100%**	(-4.00E-2,-1.50E-3),(-4.00E-2,+1.50E-3),(-7.00E-2,0),(-1.30E-3,0),(-3.80E-2,0),(-2.40E-2,0)

TABLE 6.3 Selected closed-loop pole positions

The corresponding state feedback matrices $\mathbf{K}_i$ are calculated with the aid of the routines listed in Appendix D. Both the frequency domain and the controllable companion technique of chapter five result in

$$\mathbf{K}_1 = \begin{bmatrix} 1.01E\text{-}2 & 2.94E\text{-}5 & 2.03E\text{-}3 & 3.52E\text{-}5 & 1.38E\text{-}4 & 7.03E\text{-}4 \\ 1.01E\text{-}2 & 2.94E\text{-}5 & 2.03E\text{-}3 & 3.52E\text{-}5 & 1.38E\text{-}4 & 7.03E\text{-}4 \end{bmatrix}$$

$$\mathbf{K}_2 = \begin{bmatrix} 1.73E\text{-}2 & 2.68E\text{-}4 & 4.08E\text{-}3 & 1.05E\text{-}4 & 5.17E\text{-}4 & 2.28E\text{-}3 \\ 1.73E\text{-}2 & 2.68E\text{-}4 & 4.08E\text{-}3 & 1.05E\text{-}4 & 5.17E\text{-}4 & 2.28E\text{-}3 \end{bmatrix}$$

$$\mathbf{K}_3 = \begin{bmatrix} 1.25E\text{-}2 & 7.09E\text{-}4 & 3.11E\text{-}2 & 1.06E\text{-}3 & 1.03E\text{-}3 & 1.33E\text{-}2 \\ 1.25E\text{-}2 & 7.09E\text{-}4 & 3.11E\text{-}2 & 1.06E\text{-}3 & 1.03E\text{-}3 & 1.33E\text{-}2 \end{bmatrix}$$

$$\mathbf{K}_4 = \begin{bmatrix} -1.30E\text{-}1 & -2.20E\text{-}3 & 3.69E\text{-}1 & 1.27E\text{-}2 & -5.19E\text{-}2 & -7.29E\text{-}2 \\ -1.30E\text{-}1 & -2.20E\text{-}3 & 3.69E\text{-}1 & 1.27E\text{-}2 & -5.19E\text{-}2 & -7.29E\text{-}2 \end{bmatrix}$$

(6.24)

6.4.4 OPEN-LOOP GAIN CONSIDERATIONS

This section discusses the effect of variations in the open-loop gain of the plant when state feedback is employed.

From the results above (Eqn. 6.24) and in general it is noteworthy that there is no linear relationship between the elements of matrix **K** and the amount by which the poles are moved. It is thus not possible to multiply **K** by some constant *c* to move all the poles by the same proportion. In fact it is difficult to predict in general how the poles will move if **K** is replaced by $\propto$**K**. A block diagram of this situation is shown in Fig. 6.4. $\mathbf{I}_m$ is an identity matrix of dimension *mxm*, where *m* corresponds to the number of outputs and inputs.

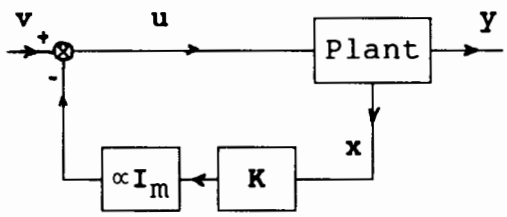


FIG. 6.4 Feedback gain configuration

The reason for this can be found from chapter five, eqn.4.30 where it was shown that the feedback vector **f** of ($\mathbf{K}=\mathbf{q}\mathbf{f}^T$) is constructed from the coefficients of the characteristic polynomials for the open- and closed-loop system. By changing some or all poles of the closed-loop system in a linear fashion, the coefficients of the closed-loop characteristic polynomial will not change in a linear way.

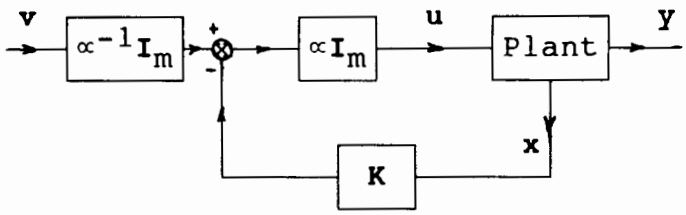


FIG. 6.5 Gain moved into forward path

The feedback gain matrix αI_m can be moved into the forward path as depicted in Fig. 6.5. In this way α can be regarded as a variation of the open-loop plant gain or as an inaccuracy in the model obtained from the plant.

6.4.5 STABILITY ANALYSIS

Although pole assignment assures stability if the selected pole positions are in the stable region, this can be misleading. The discussion in the previous section has shown that a variation in the open-loop gain of the plant can affect the pole positions of the closed-loop system considerably and could indeed make it unstable.

It is obvious that variations other than open-loop gain could also impede the stability of the system. This can be argued for any control system design method and is therefore not significant for this discussion.

It is however important for a control scheme to give an indication of gain margins. This is especially so in the design of a control system for the flotation rig, because of the inherent non-linearities of the valves, which affect the open-loop gain of the plant.

The valve characteristics could be partially linearised in the same way, that level transducer outputs were linearised in a previous project(**11**). This is not a satisfactory solution in control system design however, because in most industrial applications the inherent non-linearities can often not be determined numerically. The designed control system should thus be robust enough to tolerate non-linear behaviour of the plant.

6.4.5.1 Gain Margin

The pole assignment technique does not lend itself to an elegant method to determine gain margins of a system. The closed-loop pole positions, including a gain variations matrix **P** with diagonal element equal to α_i and off-diagonal elements zero, are given by the roots of

$$|sI-(A-PBK)| \qquad (6.25)$$

which shows that it is extremely difficult to determine stability margins for **P**.

A possible solution would be to examine the root-loci as the open-loop gain matrix **P** is varied. This would give an indication of whether the poles remain stable over that range. This method is extremely tedious and relies heavily on the accuracy of the algorithm that computes the roots of Eqn.(6.35).

The alternative, namely to analyse the step response of the system with state feedback, which can also be used to determine stability, is employed here.

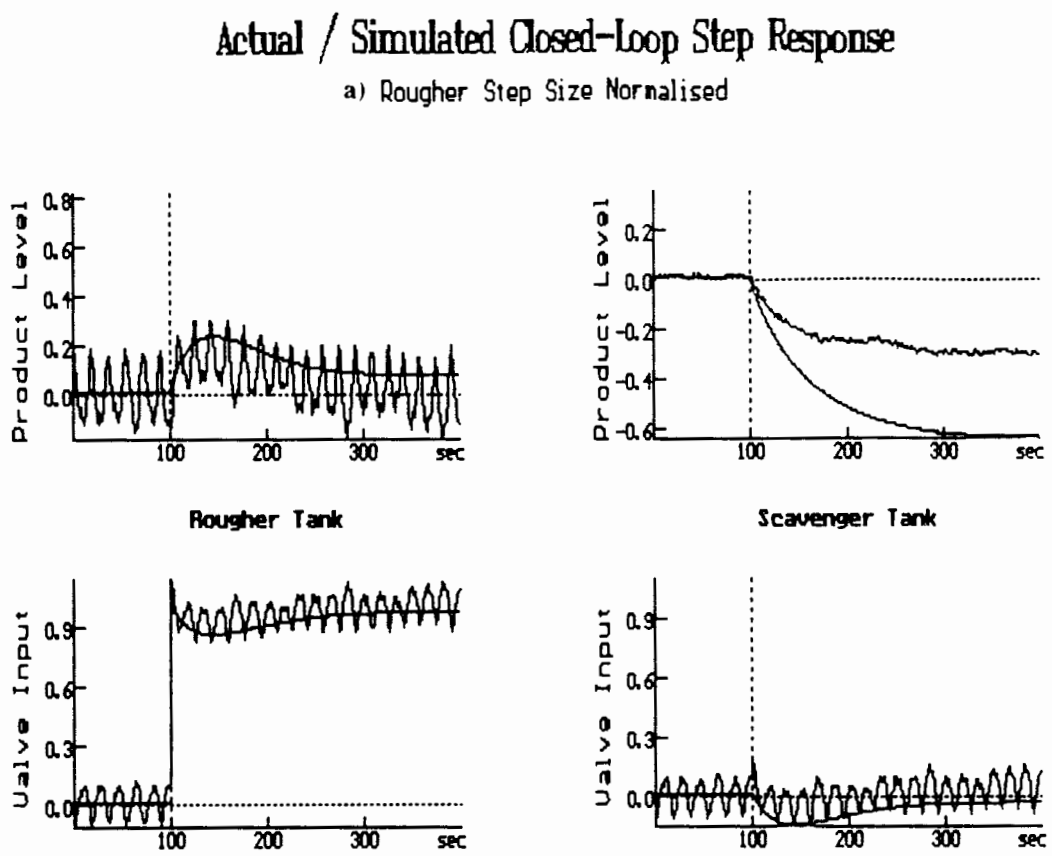
6.4.6 CLOSED-LOOP STEP RESPONSE

6.4.6.1 Step Test Procedure

Step responses were performed for the various selections of pole positions on the flotation rig. The step size for all tests was fixed at 10% = 401 units. The step was introduced after 100 seconds. The data is normalised by subtracting the offset and dividing the measurements and the valve inputs by the step size. The offset has to be calculated by averaging the first 100 samples due to the periodic variation. The normalisation is necessary such that the simulated response can be superimposed on the graphs. The original step responses can be found in Appendix C .

6.4.6.2 Step Test 1

The closed-loop step response to increase the speed of response by 10% by is shown in Fig. 6.6.



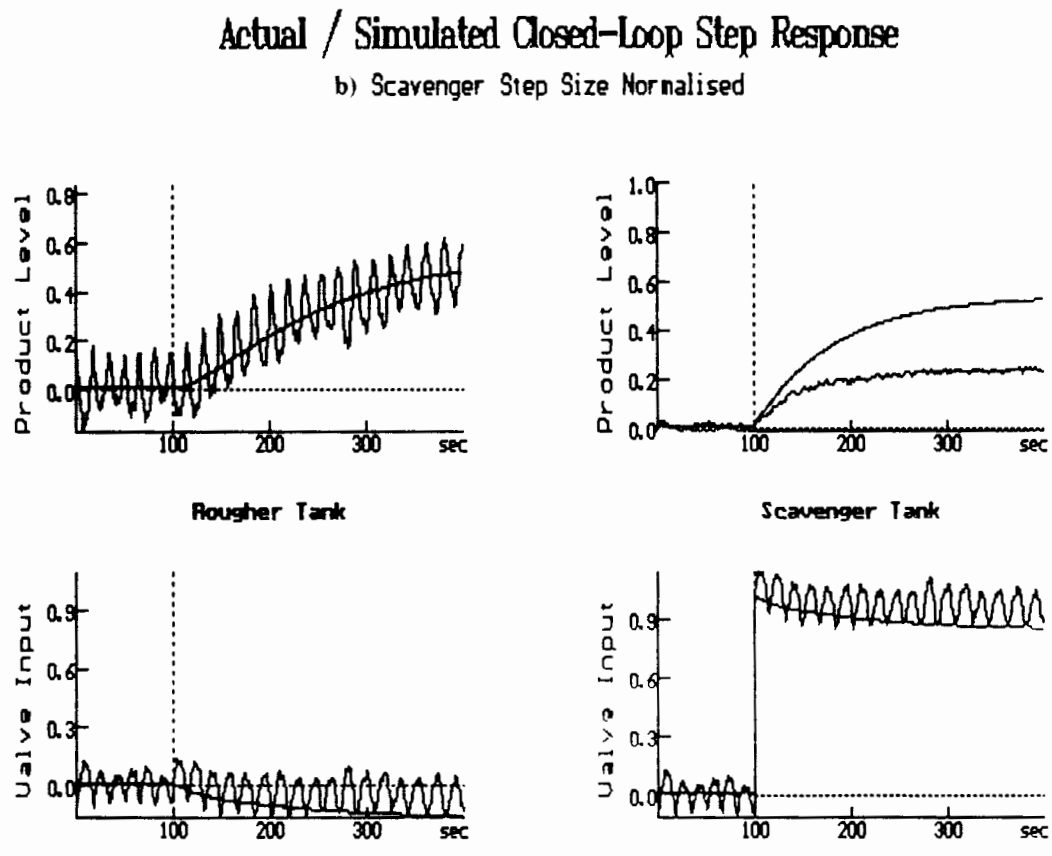


FIG. 6.6 Closed-loop response with 10% increase in speed

The step responses exhibit a sinusoidal wave that is superimposed on the Rougher level. This phenomena has been discussed in detail in chapter two and found to be due to the pumps cavitating. It is important to note here that these periodic variations are not due to the changing valve inputs, but rather that they cause the sinusoidal behaviour of the inputs.

Analysing the Rougher step response shows that the gain or slope of the cyclic variations on the Rougher level are much larger than that introduced by the step in valve opening, which indicates that the waves on the Rougher level could not have been caused by the Rougher valve.

It can be seen from both the Rougher step and Scavenger step results ignoring the superimposed sinusoid that the dynamic responses of both the levels and the plant inputs correspond very well with the simulated one. This shows that the pole assignment design has produced the expected results.

It is noticeable that the steady-state gain of the Scavenger level is less than predicted by the simulation by a factor of ½. This could have arisen from the non-linearity of the valves. The step in the valve settings may have occurred over a region of low gain, limiting the amount by which the Scavenger level will rise or fall to.

Alternatively, the levels of the four tanks may have moved out of the operation region, from which a model was obtained. If all four step tests are consistently wrong then a probable explanation is that the model is incorrect. This is possible because some valves were replaced during the course of this investigation.

Due to the small increase of 10% in the closed-loop speed of response compared to the open-loop it is difficult to determine whether the designed speed improvement has in fact been achieved in practice. However the experimental and simulated dynamics suggest that it has.

6.4.6.3 Kalman Filter Response

Fig. 6.6 c) depicts the states estimated by the Kalman filter during the above Rougher step with 10% increase in speed of response.

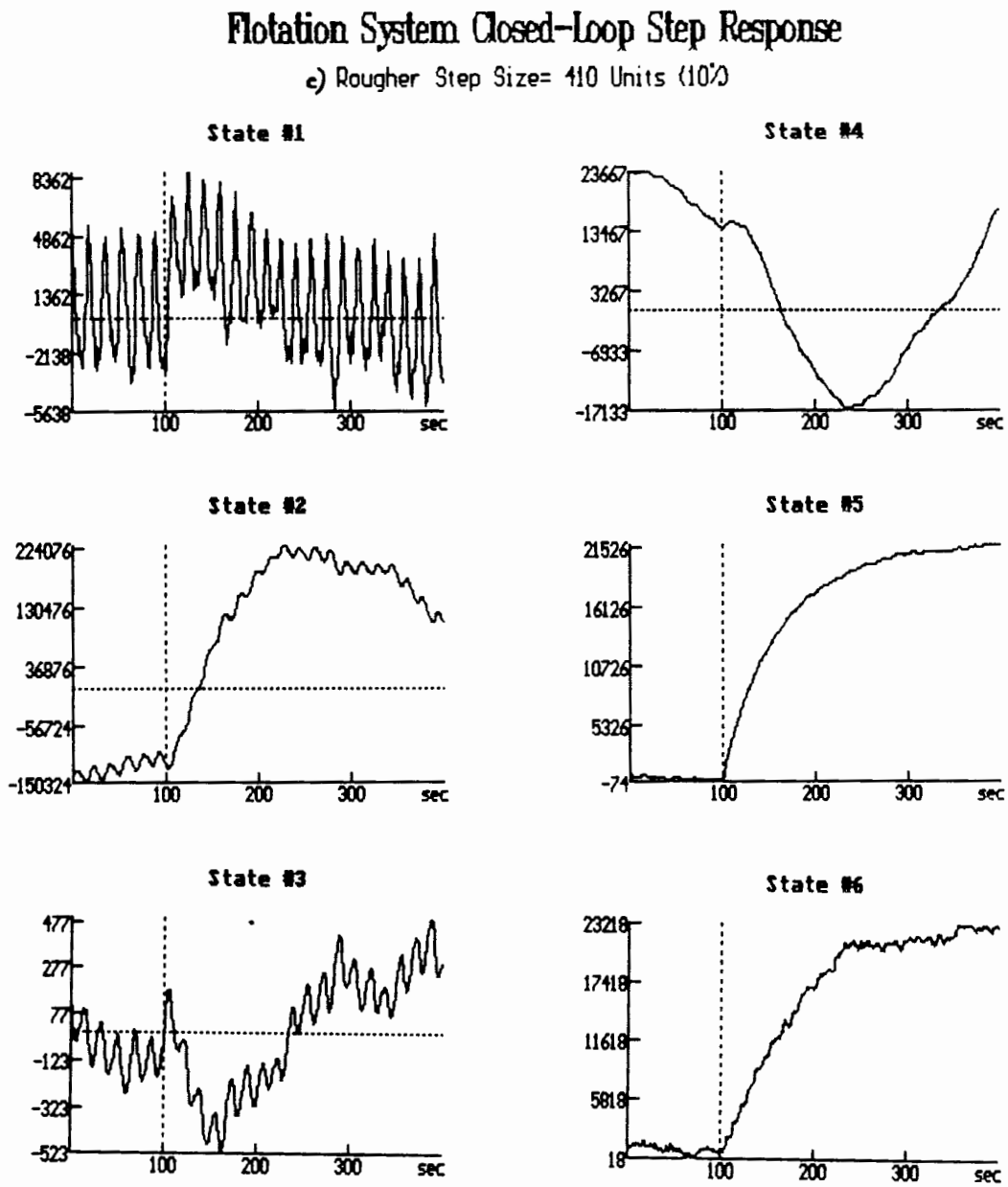


FIG. 6.6 c] Rougher step state response (10% increase in speed)

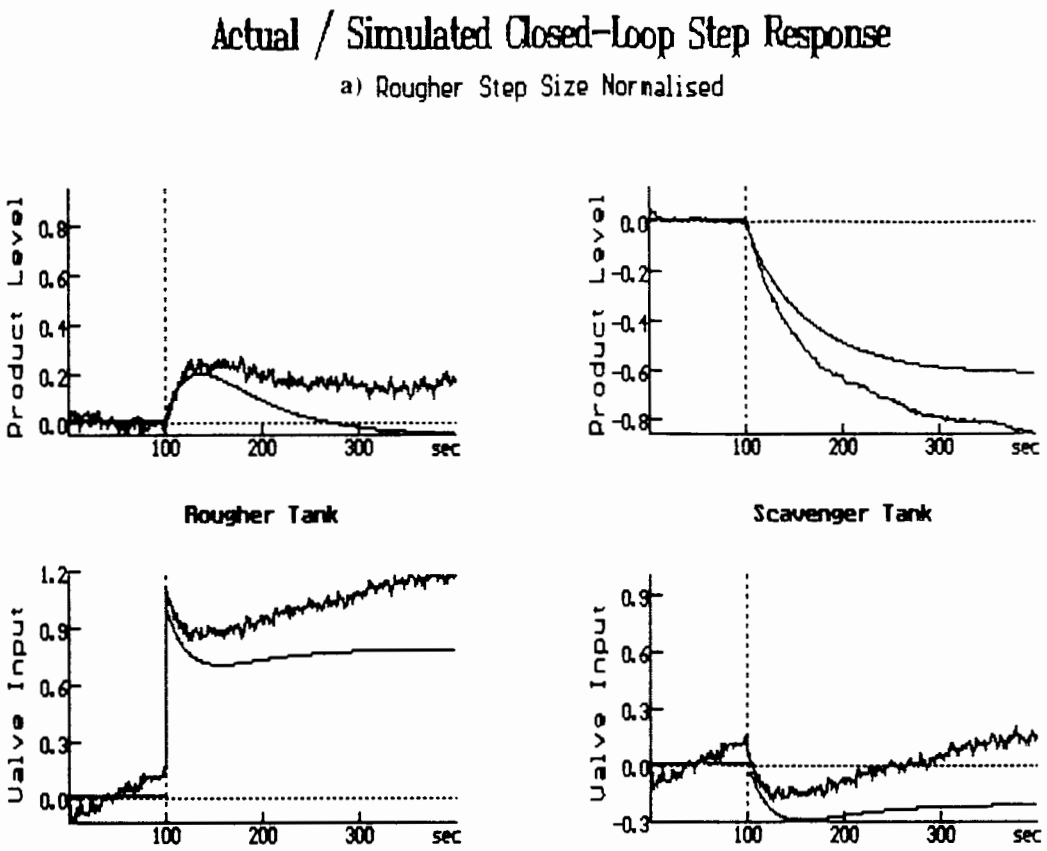
Analysing the response for state $\mathbf{x}_2$ gives an indication of the long time-constant associated with it, as it had not attained steady state after 400 seconds. This has been mentioned before in [Section 6.3.5]. State $\mathbf{x}_4$ did also not reach steady state. These two states are related to the acceleration of the Rougher level and are thus difficult to interpret from a physical point of view.

In general the states are well behaved with exception of the superimposed sinusoids, which are not introduced by the Kalman filter. It is noticeable, that the states give rise to large numbers and it is important that the digital computer used will be able to treat these without problems or that a normalised model is chosen for use in the filter. In this application, the *TURBO PASCAL* real number format was sufficient.

To conclude the discussion on the Kalman filter, the results obtained up to this point have shown convincingly that it operated suitably well. The step responses in Fig. 6.6 agree with those of the simulator and the states in Fig. 6.7 are well behaved.

6.4.6.4 Step Test 2

In Fig. 6.7 the step test results for an increase in speed of 20% are depicted.



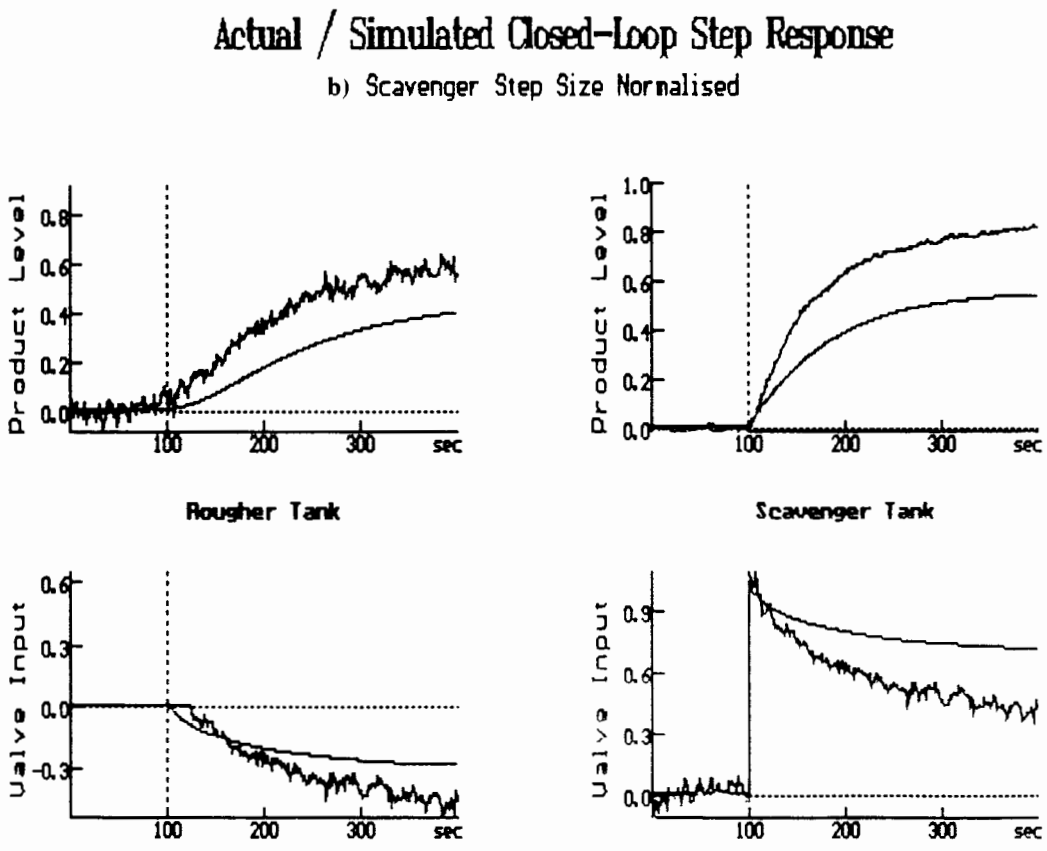


FIG. 6.7 Closed-loop response with 20% increase in speed

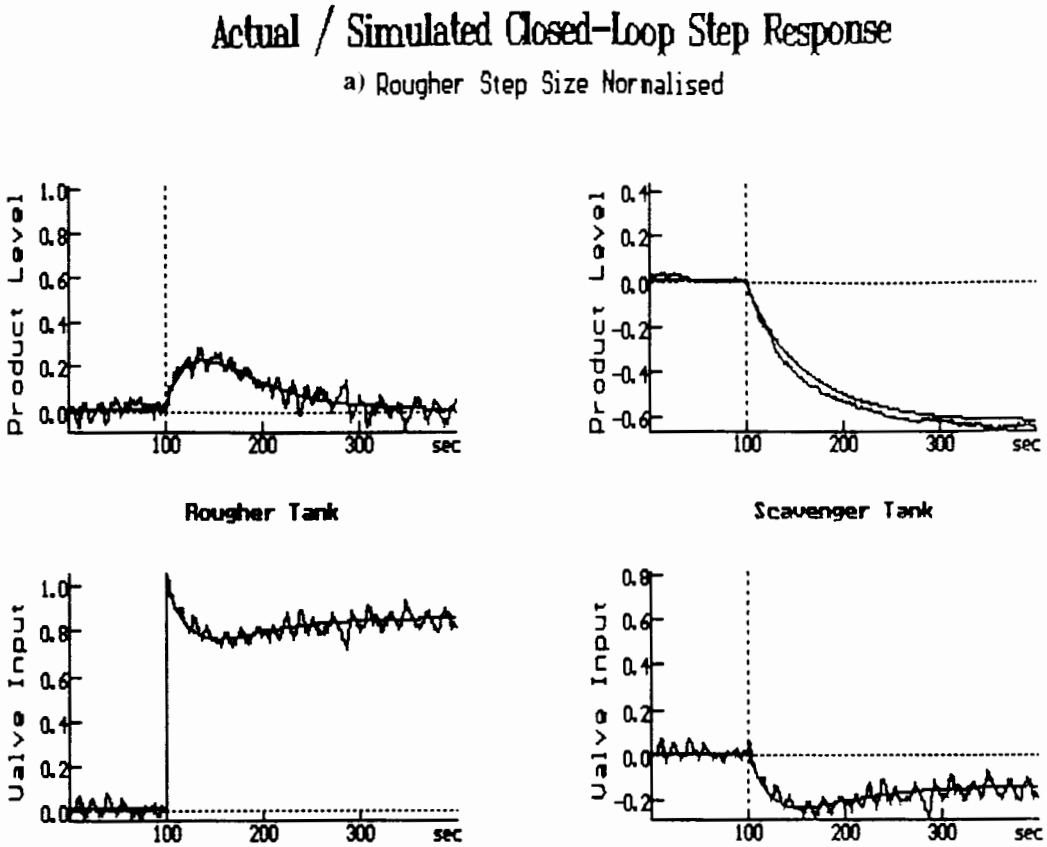
The results in Fig. 6.7 a) show a moderate deviation from the expected response. This fact can be explained by observing that the plant was not completely at steady-state. The rising valve inputs indicate that the levels were drifting. Close examination would show that the Rougher level did indeed drop steadily. This problem has been discussed in chapter two.

The plant response to a Scavenger step input [Fig. 6.7 b)] also deviated from the simulated response. This can be accounted for by inspecting the Rougher valve input, which was at its limit and could not open even more. It did take a significant amount of time (25 sec) for the input to move into the operating range and this will have definitely obscured the plant response.

This test in particular underlines the necessity for the plant to be completely at steady-state and for the valves to remain in their operating region throughout the experiment. Only then can the response be successfully compared to a simulated one.

6.4.6.5 Step Test 3

The experiment in Fig. 6.8 used state-feedback to improve the system response time by 50%.



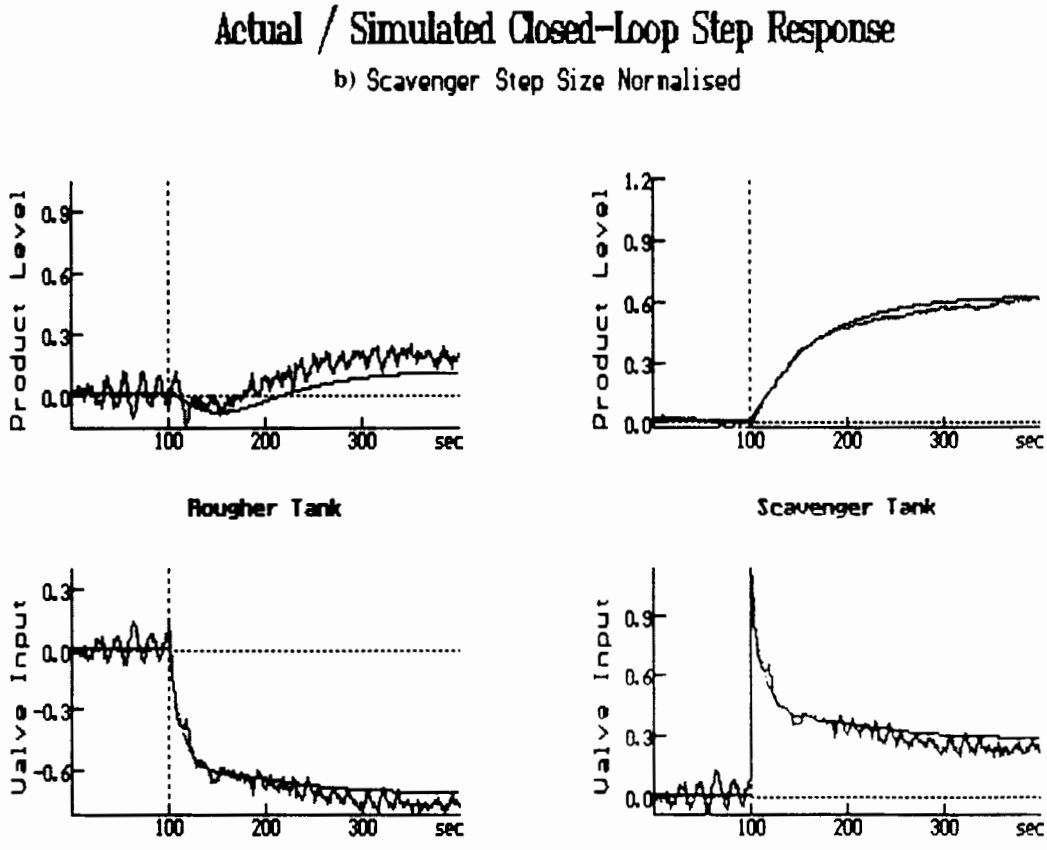


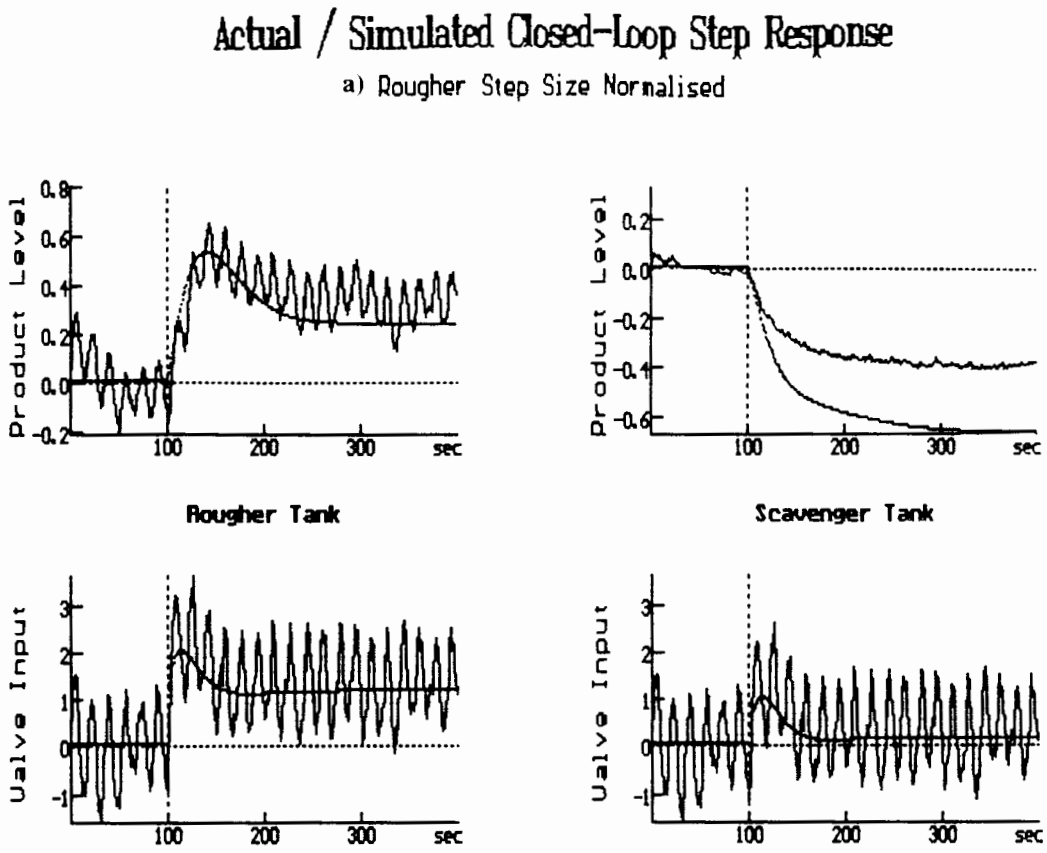
FIG. 6.8 Closed-loop response with 50% increase in speed

The correspondence obtained in this trial was exceptionally good. This result proves the success of state-feedback on a real plant and the achievement of the Kalman filter to predict the states correctly.

The behaviour of the Rougher tank level to a Scavenger input change exhibits a non-minimum phase response. This may be unsatisfactory for a setpoint tracking control scheme and different pole positions would have to be selected to remove this problem.

6.4.6.6 Step Test 4

The final step test with the intention to double the speed of response is shown in Fig. 6.9.



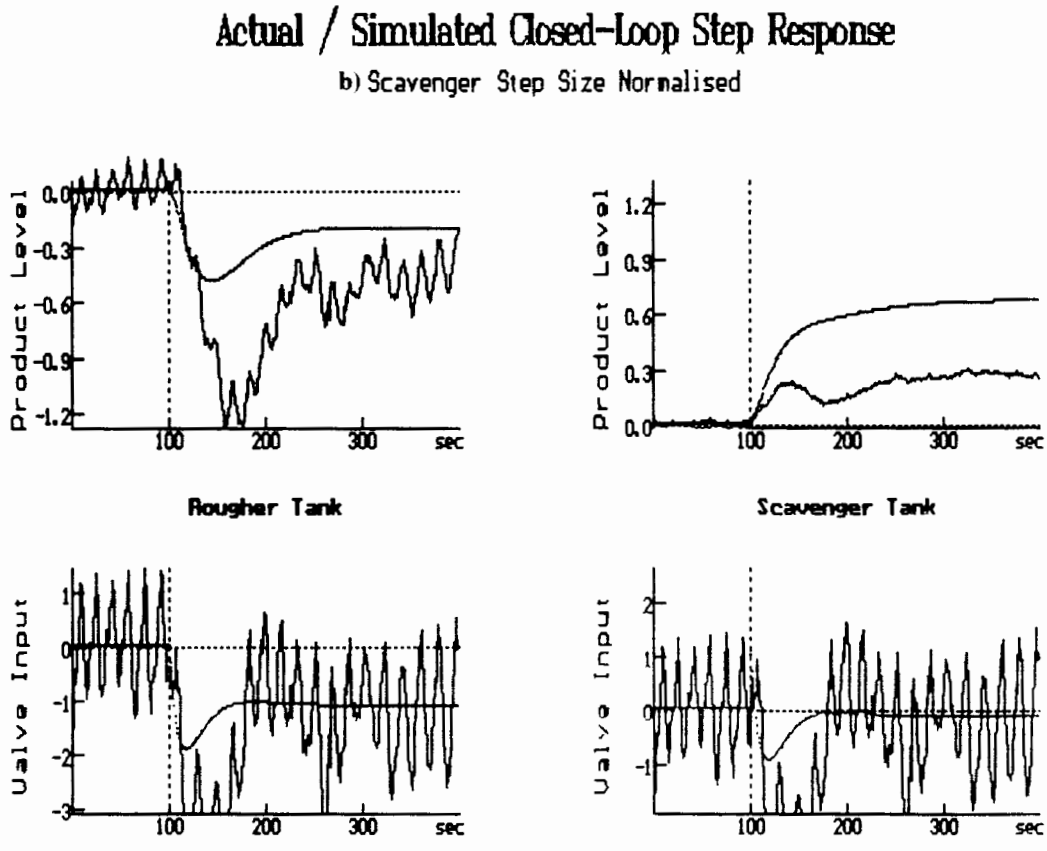


FIG. 6.9 Closed-loop response with 100% increase in speed

The experiment in Fig. 6.9a) shows that although the levels of both tanks were drifting in the first 50 seconds, this variation was successfully counteracted by the valve inputs before the step took place. The predicted transient response corresponds well to the actual, except for the Scavenger level. This can again be ascribed to the fact that the valve settings may have been in a low gain region, or that the plant had moved out of the operating region for which its model was determined.

Analysing the findings in Fig. 6.9 b) shows large deviations from the predicted results. This can be attributed to the fact that both valve inputs were saturated for about 50 seconds. The over-driven inputs affected the Scavenger level adversely. Whereas a response similar to first order was expected a complicated one of high order governed that tank. This shows that pole assignment can be unfavorably affected by extreme non-linearities.

It is interesting to observe that the responses here have reversed in sign compared to the other three step tests discussed previously. It is thus important to analyse the step response of a particular selection of poles when developing a setpoint tracking scheme for the system.

It is further noticeable that the Scavenger step results in the valve inputs behaving as if the Rougher had been stepped with a negative step. This gives rise to a situation where it will be difficult or even impossible to design a setpoint tracking controller for this system as the levels cannot be controlled independently. The system has in this case only one input effectively.

6.4.6.7 General Analysis

By analysing the last step test (Fig. 6.9), it is apparent that the system has indeed improved on its speed of response. The settling time for the transient response has halved compared to that of the first test (Fig. 6.6). Close inspection of the predicted Scavenger level response shows that it is not of first order anymore, but instead exhibits a pronounced bend justifying the increase in speed.

An examination of the amplitude of the sinusoid on the valve inputs for all four step tests, shows that the feedback gain varies from one set of poles to the other. The variation is large for a 10% and 100% improvement of speed. For the 20% and 50% improvement however the gain is small. This is again not a linear relationship, i.e. the gain does not increase as the poles are moved progressively further from the origin.

The diversity of transient responses, that were observed for various pole positions has been demonstrated by the results above and shows that the pole assignment technique must include a critical analysis of the closed-loop system transient response to step inputs before any setpoint tracking scheme can be employed.

The closed-loop step responses have also shown that the plant remained stable as no signs of instability could be found.

6.5 CONCLUSION

In this chapter an intuitive method for the design of Kalman filters has been developed and applied successfully.

It was also shown that pole assignment by state feedback could be achieved on a real plant. The sensitivity of state feedback to parameter changes or non-linearities have been demonstrated.

The various step responses of different sets of poles have again emphasised that the closed-loop system behaviour cannot be directly predicted from a choice of **q** and a set of poles.

7. CONCLUSION

7.1 INTRODUCTION

The fact that both the Inverse Nyquist Array (INA) and pole assignment techniques could be applied successfully to the flotation rig has been demonstrated in previous chapters.

The discussion is concluded in this chapter with a direct comparison of the two methods and suggestions for future work.

7.2 MODEL DETERMINATION

The INA technique is applied in the frequency domain, requiring a transfer function representation of the plant.

In addition to using an existing state space description, a transfer matrix model of a multivariable plant can be obtained with relative ease from step response analysis as demonstrated in chapter two. It is further possible and usually adequate to minimise the order of the individual transfer functions to simplify the model.

The state space model needed to perform pole assignment can be established from the transfer matrix, which usually results in a system with a large number of states. This strains the numerical accuracy of any design technique and makes physical interpretations difficult.(10)

The alternative method to obtain a model in state space is by describing the plant in terms of differential equations.

This however can only be applied to multivariable systems of which the various elements are known scientifically allowing construction of these equations. A black-box type identification is thus not possible and it is further difficult to establish to what degree of accuracy these formulations must be performed.

7.3 CONTROL SYSTEM DESIGN CRITERIA

A discussion on desirable properties of control systems has been given by Layton (**7**). These and additional features are used to compare the two techniques.

7.3.1 STABILITY

The main criterion in system design is that of stability.

In pole assignment stability is theoretically guaranteed if the desired pole positions lie in the left half plane.(**31**) However, even if these lie in stable regions, it is found in practice that instability can occur for a number of reasons.

If the inputs to the plant saturate, the system may become unstable. Referring to Fig. 7.1, input saturation is equivalent to the feedback path being broken, as the plant inputs are not affected by further changes in the states. The stability of the resulting open-loop system depends on the open-loop stability of the plant. If the plant is open-loop unstable then saturation will cause instability.

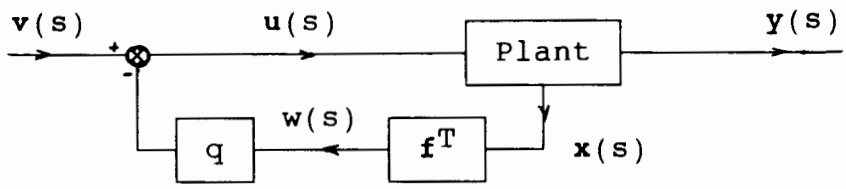


FIG. 7.1 Pole Assignment by State Feedback

In addition, if the states are not measurable directly from the plant and an estimator is used, the effects of saturation are more difficult to predict. Fig. 5.5 in chapter five shows an implementation of a Kalman filter, where the plant inputs are used to estimate the states. The Kalman filter may be unaware of saturation of the inputs and then estimate the states incorrectly, and may cause instability.

A similar situation can occur if the plant outputs saturate, as these are also used to estimate the states of the plant. If they saturate, they may distort the estimated states and result in unstable operation of the control system.

Changes in the open-loop gain caused by non-linear behaviour of the inputs or outputs can also adversely affect the control scheme employing pole assignment. To predetermine how changes in the gain will affect the closed-loop system is difficult. This was discussed in detail in chapter six, section 6.4.5.1.

The pole assignment technique gives no distinct indication of how these factors influence the stability and this omission limits its usefulness in practice.

In this respect the INA technique is much better, because not only does it determine stability by the application of

the Nyquist criterion, but it also indicates a stability or gain margin with the aid of Gershgorin and Ostrowski circles. From these a stability region or gain space can be constructed [see Chapter 3, Section 3.4.4]. The knowledge of the maximum and minimum gains of the nonlinear inputs of a plant can thus be included in the design. Similarly, saturation, which is equivalent to the gain becoming zero, can be incorporated. And if these extremes lie within the gain space, stability is assured. This is not a stringent design criterion as the system may remain stable even though the gains lie outside this region.(**2**)

The problem of changes in the dynamics of a plant is always difficult to include in any design procedure. The only way to experiment with these parameter changes is to insert a number of different models of the plant in the final control scheme and analyse their effects.

7.3.2 INTEGRITY

The concept of system integrity is also related to stability. A good feature of control systems is to remain stable even if failures such as disruption of some or all the feedback paths occur. The INA technique allows investigation of such an event, which can be emulated by setting the corresponding elements of matrix **F** equal to zero and analysing the resulting gain space. The pole assignment technique does not have this feature although it is very sensitive to failure in the state feedback path. This can result in instability in a similar way as saturation of the plant inputs [Section 7.3.1].

7.3.3 INTERACTION

The main feature of the INA technique lies in its ability to reduce the interaction of system inputs and outputs. Graphical techniques can be used to 'measure' the amount of interaction. If the method can be successfully applied then individual inputs of the closed-loop system control individual outputs. The multivariable system can then be approximated by a number of single-input, single-output processes, simplifying the controller design considerably.

The amount of interaction that results by applying pole assignment is difficult to predict and can only be established by simulation.

7.3.4 NOISE IMMUNITY

Another important feature of system design is noise immunity.

The INA method allows single-loop PI controllers to be designed for the dominant system, which can be used to limit the noise sensitivity of the system. The sensitivity can be established from the INA diagrams.

The pole assignment techniques does not have this feature. As noted in chapter six, no relationship exists between pole location and resulting noise sensitivity. It is extremely difficult to predict the system behaviour to noise even by simulation. It can only be established by applying the control scheme to the plant directly and observing the noise on the plant inputs.

7.3.5 TRACKING BEHAVIOUR

The INA technique allows single-loop PI controllers to be designed for the dominant system to limit the amount of overshoot, speed and error in setpoint tracking.

As the pole assignment method does not result in a dominant or non-interaction system, it is difficult to design single-loop PI controllers for it. It does however allow the designer to change the dynamic behaviour of the plant. If the system response is oscillatory, a damped response can be achieved by moving the poles to better positions. The speed of response can also be improved by moving the poles as demonstrated in chapter six.

7.3.6 CONTROLLER STRUCTURE

The structure of the controller resulting from the INA technique depends on the complexity of the plant model. For small system with two inputs and two outputs, a constant pre-compensator is usually sufficient to ensure dominance.(**2**) As the order of the individual elements of **G**(s) and the number of inputs and outputs increases dynamic controllers become necessary. Standard PI controllers can be inserted to complete the control scheme as shown in chapter four.

The pole assignment technique employs constant gain state feedback which is simple to implement if the states can be measured directly from the plant. If however a state estimator is required the structure of the control system becomes more intricate [Chapter 6].

In both cases thus, the controller structure is relatively complicated.

7.3.7 DESIGN TRADE-OFFS

To design a control system that contains all the above features is usually not possible and compromise is essential. By increasing the controller gain for example, the speed may be improved but the noise sensitivity will also increase. It is thus at the discretion of the designer to decide which features are more important.

A good design technique should allow the designer to compromise between different requirements. It should thus be possible to predict system behaviour with respect to the above features.

In this respect, the INA technique contains most information on the closed-loop system behaviour. The amount of interaction, noise sensitivity, stability with gain margins and integrity of the closed-loop system are contained in the INA diagrams and by application of the Gershgorin and Ostrowski circles. These factors were used to derive a control scheme for the flotation rig in chapter four.

The pole assignment method gives little insight into the problem as most of the features of the resulting closed-loop system are not fully predictable. No quantitive indications can be obtained of characteristics such as interaction, noise sensitivity, integrity, gain margins and stability during the design process. These can only be established by simulation or on the plant itself. Furthermore, no clear strategy exists on how to improve or design around individual requirements.

The failure of the pole assignment technique in this respect is related to the state space description of control system. It is easy to obtain the dynamic behaviour of a system from

its state space model, but to recover further information is difficult.

7.4 FUTURE WORK

7.4.1 CONTROL OF FULL FLOTATION RIG

Although only a subsystem of the flotation rig was used as a practical example for this discussion, the full model was developed in chapter two which can be used to design a control system for the rig. A project of this nature would generate considerable interest especially for the mineral extraction industry.

The suitability of the Inverse Nyquist Array technique for this purpose has been well demonstrated during the course of this dissertation. Industrial flotation rigs do however consist of a large number of circuits, which complicates the design process. Large systems have recently become an area of interest and the INA technique can be extended towards these with the concept of block-dominance(**34**).

7.4.2 A GRAPHICAL TECHNIQUE FOR POLE ASSIGNMENT

The success of pole assignment by state feedback has been demonstrated, but difficulties arise from the resulting closed-loop zero positions. A graphical technique that plots the closed-loop zero positions as the direction of the q vector is varied, while the closed-loop poles remain fixed, could ease the design process.

A similar technique could be used to show the movement in pole positions of the closed-loop system as the open-loop

gain is varied. This would give an indication of the system response to nonlinear changes in the behaviour of the plant.

The flotation rig did not necessitate the application of pole assignment as the pole positions of the model were, except for their slowness, acceptable. A system with oscillatory or even unstable behaviour would be more suitable. Such a system must however not be too large, i.e. contain more states than ±10, which diminishes the computational accuracy of the technique.

Further research could be done to use state feedback to decouple a plant(**21**). This technique however limits the number of assignable poles, which in the case of the flotation rig could be acceptable, if the resulting system has limited interaction.

CHAPTER 7 CONCLUSION

REFERENCES

1. VENZKE R.:"*Multivariable control software on a personal computer*", SACAC workshop, July 1986.

2. OWENS D., CHOTAI A.:"*Approximate models in multivariable process control - an inverse Nyquist array and robust tuning regulator interpretation*", IEE Proceedings, Vol.133, pp.1-12, January 1986.

3. ROSENBROCK H.H.:"*Computer-aided control system design*", Academic Press, 1974.

4. PATEL R.V., MUNRO N.:"*Multivariable system theory and design*", Pergamon Press, 1982

5. BORRIE J.A.:"*Modern control systems*", Prentice Hall International, 1986.

6. KREYSZIG E.:"*Advanced engineering mathematics*", 5th Edition, John Wiley & Sons, 1983.

7. LAYTON J.M.:"*Multivariable control theory*", Peter Peregrinus Ltd., 1976.

8. TWIDLE T., ENGELBRECHT P., KOEL J.:"*Improvements in stabilizing control at Black Mountain*", J.S.Afr.Inst.Min.Metall., Vol. 86, no.1, pp. 15-24, January 1986.

9. LJUNG L., YUAN Z.:"*Asymptotic properties of Black-box identification of transfer functions*",IEEE Transactions on Automatic Control, Vol. AC-30, June 1985.

10. DONATI F., VALLAURI M.:"*Guaranteed Control of 'almost linear' plants*", IEEE Transactions on Automatic Control, Vol. AC-29, January 1984.

11. VENZKE R.H.E.:"*Nonlinear controller design for a coupled tank system*", Undergraduate Thesis, University of Cape Town, 1985.

12. MUNRO N.:"*Modern approaches to control system design*", Peter Peregrinus Ltd., 1979.

13. MUNRO N., PATEL R.:"*Multivariable system theory and design*", Pergamon Press, 1982.

14. PORTER B., JONES A.:"*Time-domain identification of transmission zero locations of linear multivariable plants*", IEEE Transactions on Automatic Control, Vol. AC-30, No. 9, September 1985.

15. VAN DE VEGTE J.:"*Feedback control systems*", Prentice-Hall, 1986.

16. WONHAM W.M.:"*On pole assignment in multi-input controllable linear systems*",IEEE Transactions on Automatic Control, Vol. AC-12, pp. 660-665, December 1967.

17. FALLSIDE F., SERAJI H.,"*Direct design procedure for multivariable feedback systems*", IEE Proceedings on Control & Science, Vol.118, pp. 797-801, June 1971.

18. PETKOV P., CHRISTOV N., KONSTANTINOV M.,"*A computational algorithm for pole assignment of linear single-input systems*", IEEE Transactions on Automatic Control, Vol. AC-29, pp. 1045-1048, November 1984.

19. TSUI C.:"*An algorithm for computing state feedback in multi-input linear systems*", IEEE Transactions on Automatic Control, Vol. AC-31, pp.243-246, March 1986.

20. MIMINIS G., PAIGE C.:"*A direct algorithm for Pole Assignment of time-invariant multi-input linear systems using state feedback*", Automatica, Vol.24, No.3, pp.343-356, 1988.

21. FALB P.L., WOLOVICH W.A.,"*Decoupling in the design and synthesis of multivariable control systems*", IEEE Transactions on Automatic Control, Vol. AC-12, pp.651-659, December 1967.

22. KONO M.:"*Diagonal decoupling of linear multivariable systems with constant gain measurement feedback*", IEEE Transactions on Automatic Control, Vol. AC-30, pp. 1237-1238, December 1985.

23. PORTER B., JONES A.H.,"*Time-domain identification of transmission zero locations of linear multivariable plants*", IEEE Transactions on Automatic Control, Vol. AC-30, pp. 1050-1053, September 1985.

24. WILLIAMSON D.:"*Finite wordlength design of digital Kalman filters for state estimation*", IEEE Transactions on Automatic Control, Vol. AC-30, pp.930-939, October 1985.

25. BOZIC S.M.:"*Digital and Kalman filtering*", Edward Arnold, 1979.

26. SONG T.L., SPEYER J.L.:"*A stochastic analysis of a modified gain extended Kalman filter with applications to estimation with bearings only measurements*", IEEE Transactions on Automatic Control, Vol.AC-30, pp. 940-949, October 1985.

27. MENDEL J.M.:"*Computational requirements for a discrete Kalman filter*", IEEE Transactions on Automatic Control, Vol. AC-16, pp. 748-758, December 1971.

28. MEHRA R.K.," *On the identification of variances and adaptive Kalman filtering*", IEEE Transactions on Automatic Control, Vol. AC-15, pp.175-184, April 1970.

29. SORENSON H.W.,"*On the error behavior in linear minimum variance estimation problems*", IEEE Transactions on Automatic Control, Vol. AC-12, pp.557-562, October 1967.

30. BROWN R.G.:"*Introduction to random signal analysis and Kalman filtering*", John Wiley & Sons, 1983.

31. VERHAEGEN M., VAN DOOREN P.:"*Numerical aspects of different Kalman filter implementations*", IEEE Transactions on Automatic Control, Vol. AC-31, pp.907-917, October 1986.

32. HANSEN A.G.:"*Simulating the normal distribution*", Byte, Vol.10 No.10, pp. 137-138, October 1985.

33. BHATTACHARYYA S., HOWZE J.:"Transfer function conditions for stability", IEEE Transactions on Automatic Control, Vol. AC-30, No.6,pp. 581-583, June 1985

34. OHTA Y., SILJAK D., MATSUMOTO T.: "*Decentralized control using quasi-block diagonal dominance of transfer function matrices*", IEEE Transactions on Automatic Control, Vol. AC-31, May 1986.

REFERENCES

APPENDICES

A) Open-loop Step Responses 176

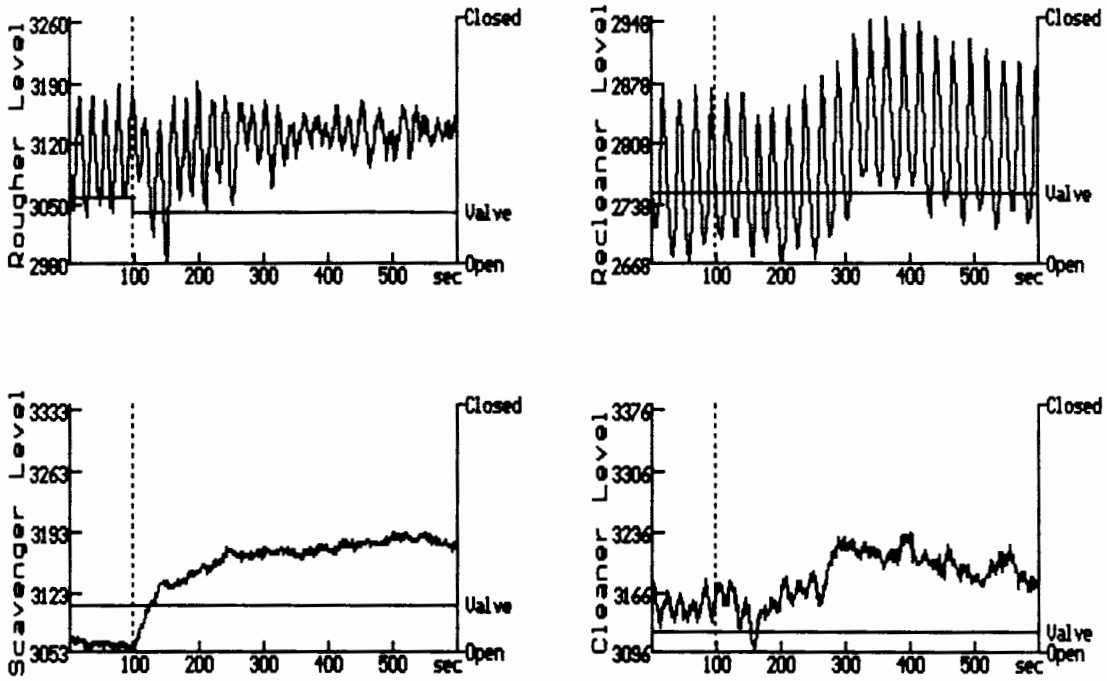
B) Closed-loop Step Response (INA Technique) 187

C) Closed-loop Step Response (Pole Assignment) 189

A) OPEN-LOOP STEP RESPONSES

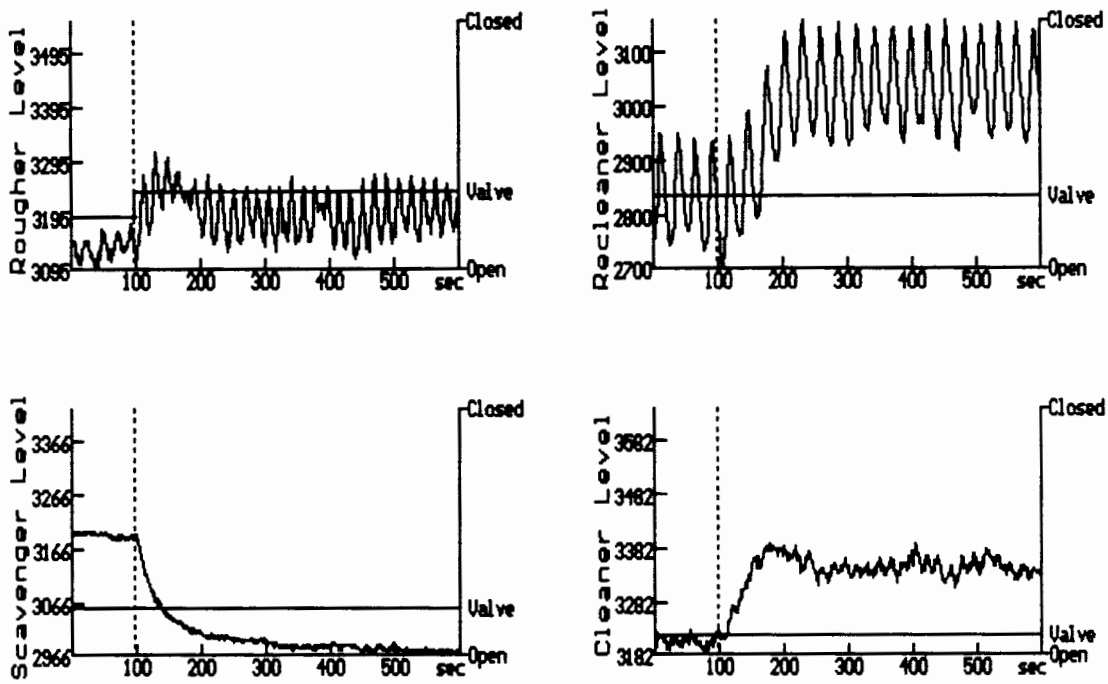
Flotation System Open-Loop Step Response

Test 1a) Rougher Step Size= -216 Units (-4%)



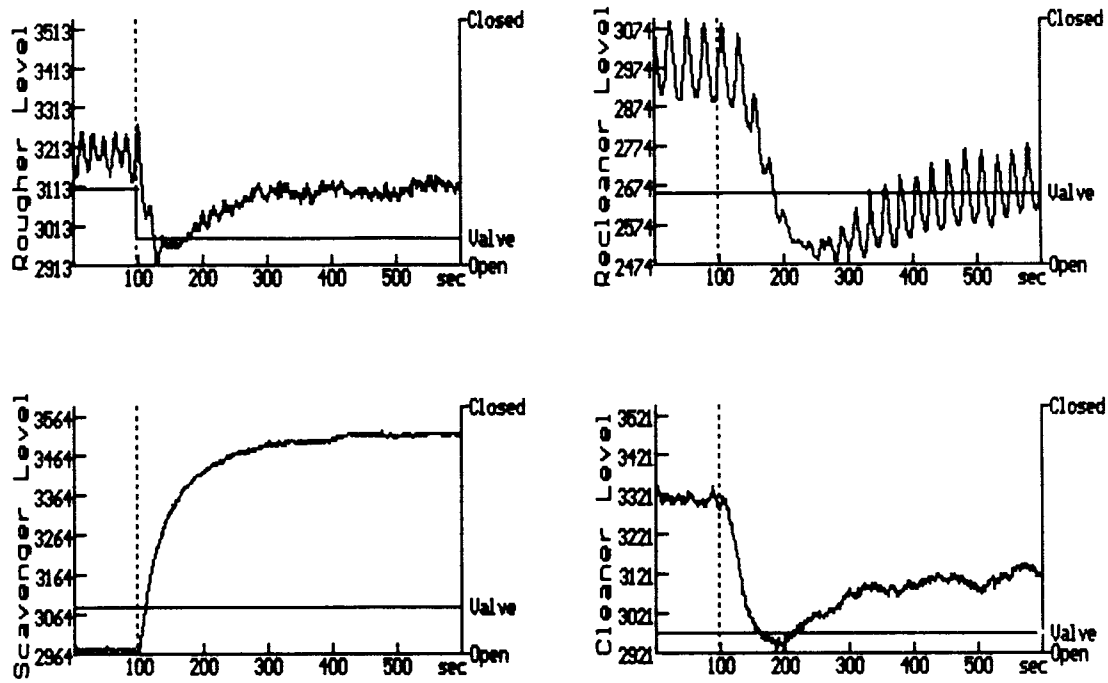
Flotation System Open-Loop Step Response

Test 1b) Rougher Step Size= 410 Units (10%)



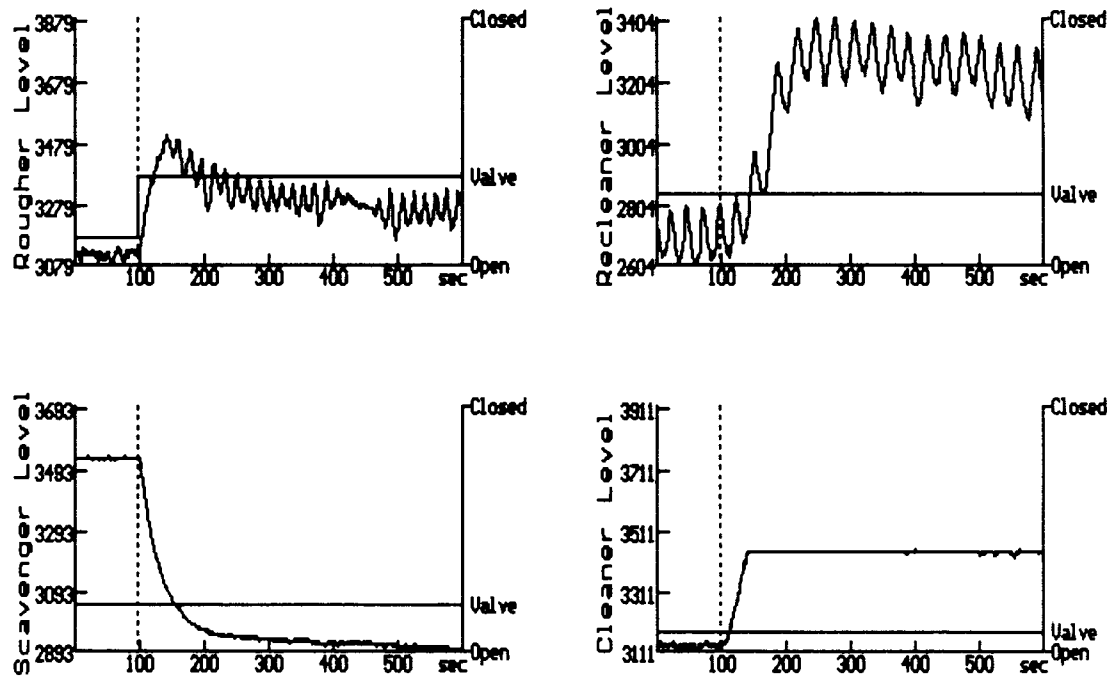
Flotation System Open-Loop Step Response

Test 1c) Rougher Step Size= -819 Units (-19%)



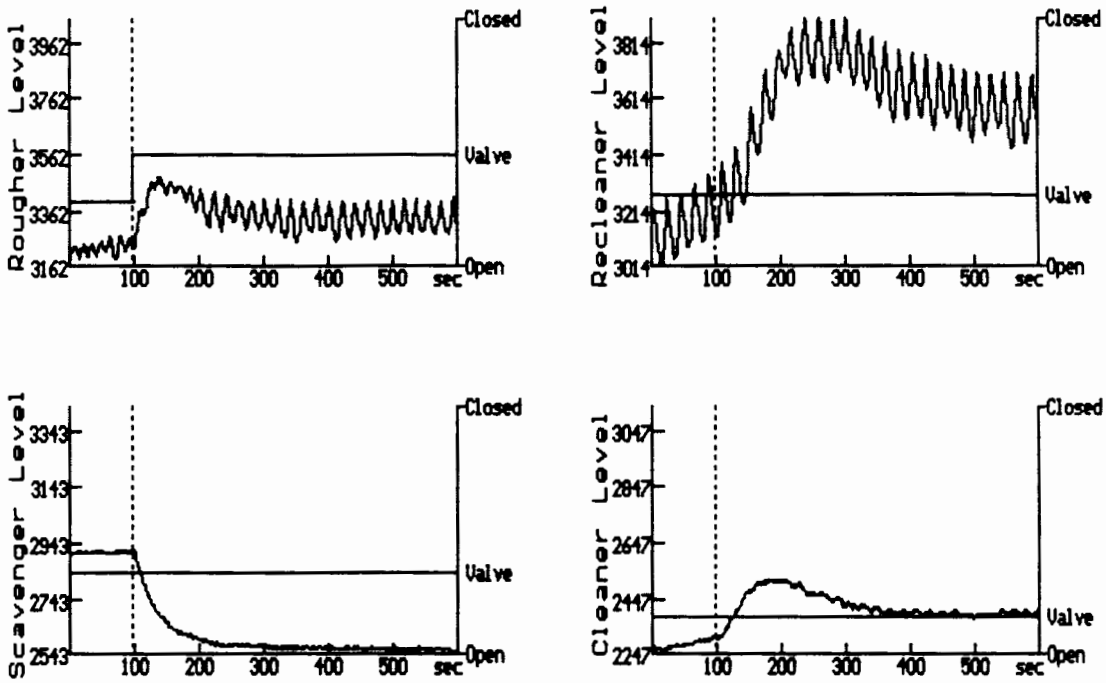
Flotation System Open-Loop Step Response

Test 1d) Rougher Step Size= 1024 Units (25%)



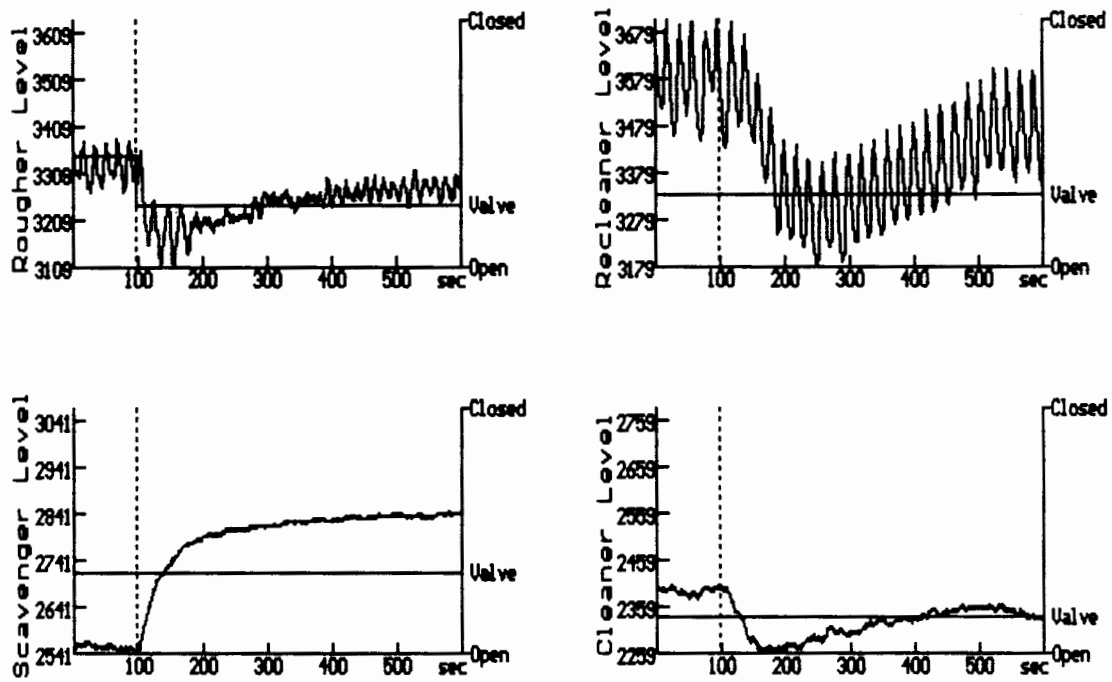
Flotation System Open-Loop Step Response

Test 1e) Rougher Step Size= 807 Units (20%)



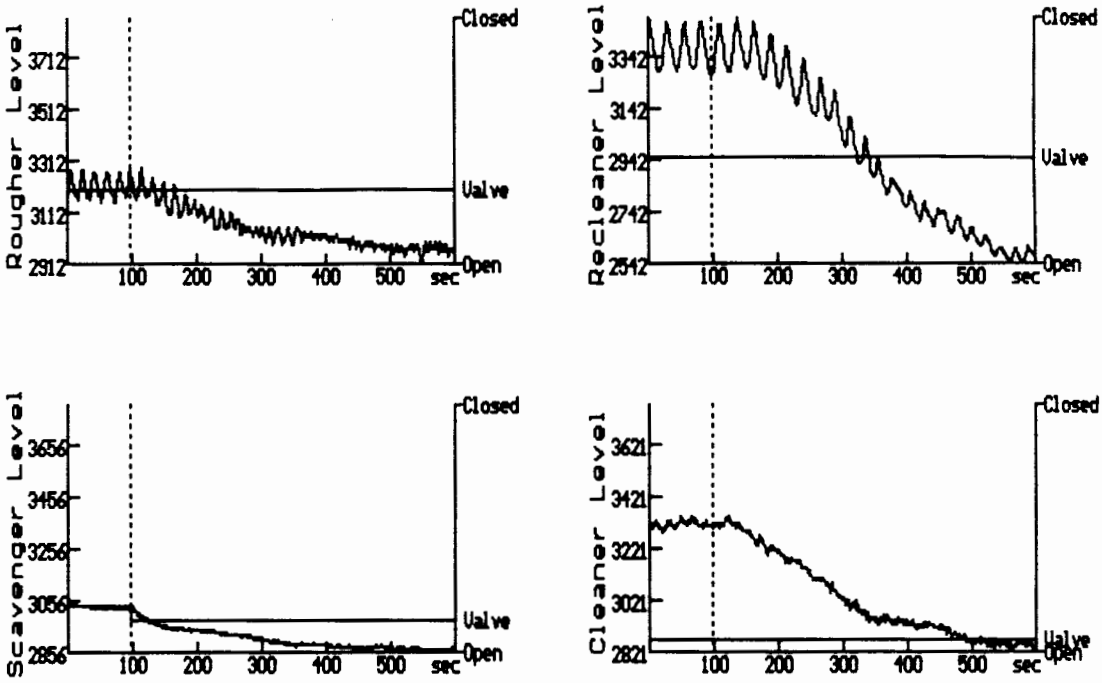
Flotation System Open-Loop Step Response

Test 1f) Rougher Step Size= -819 Units (-19%)



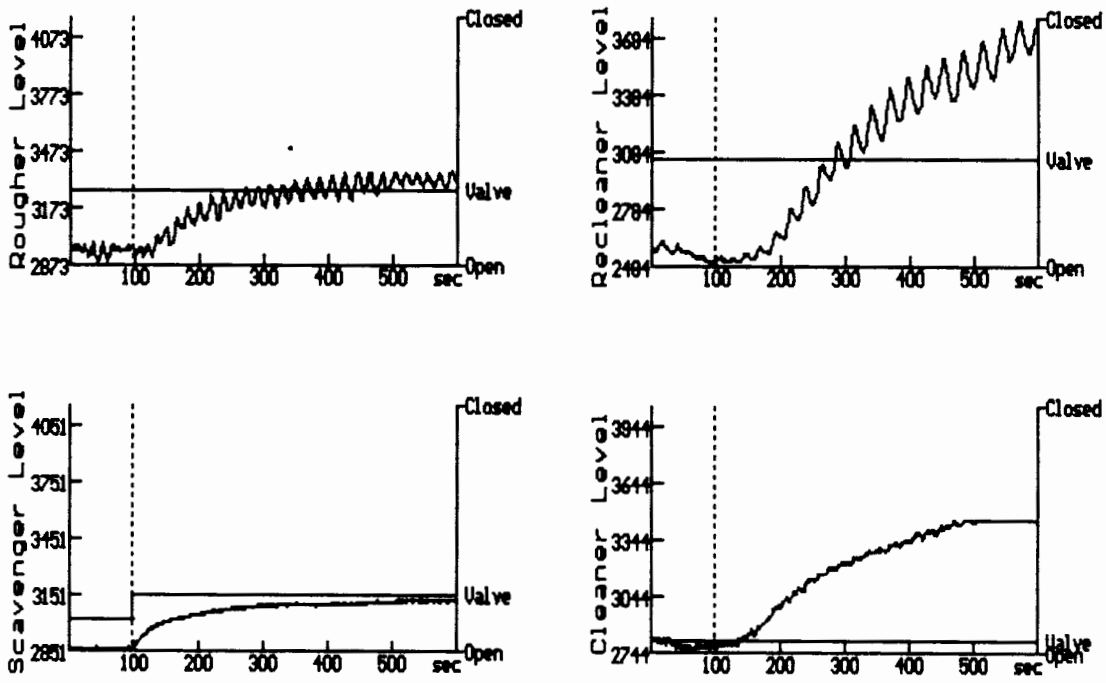
Flotation System Open-Loop Step Response

Test 2a) Scavenger Step Size= -224 Units (-4%)



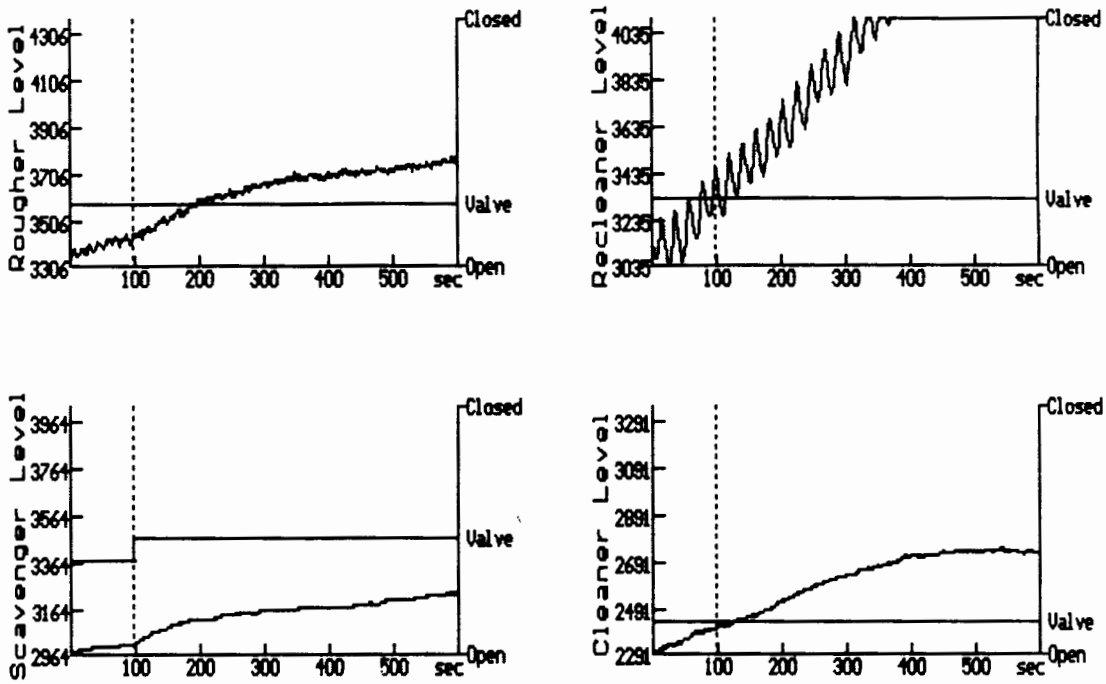
Flotation System Open-Loop Step Response

Test 2b) Scavenger Step Size= 409 Units (10%)



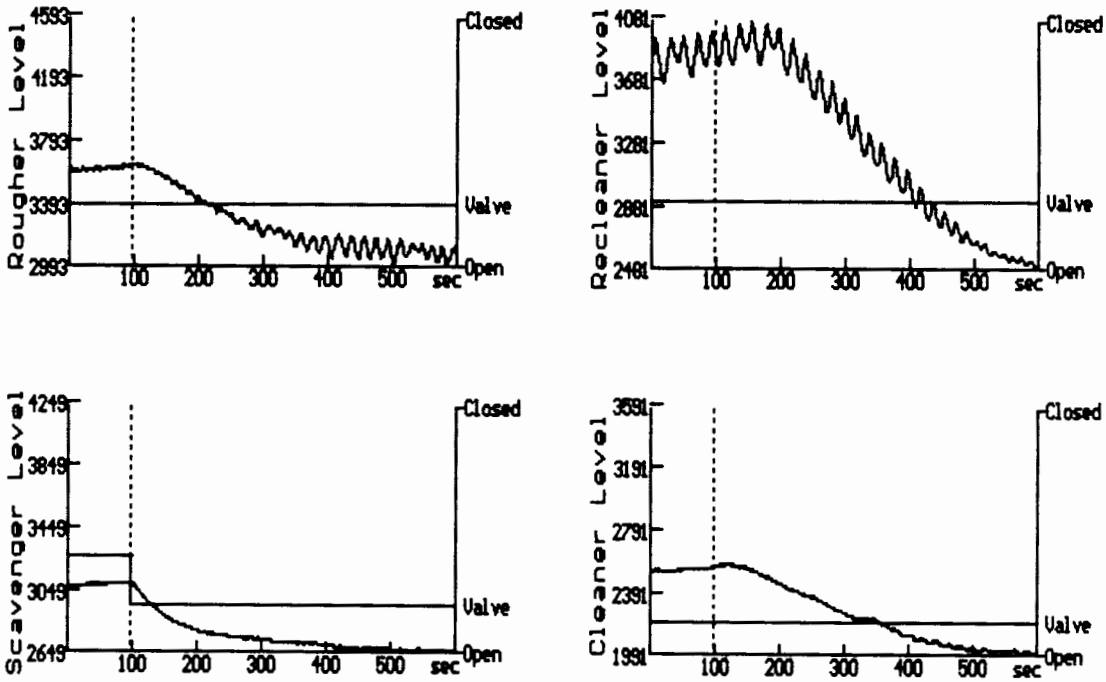
Flotation System Open-Loop Step Response

Test 2c) Scavenger Step Size= 408 Units (10%)



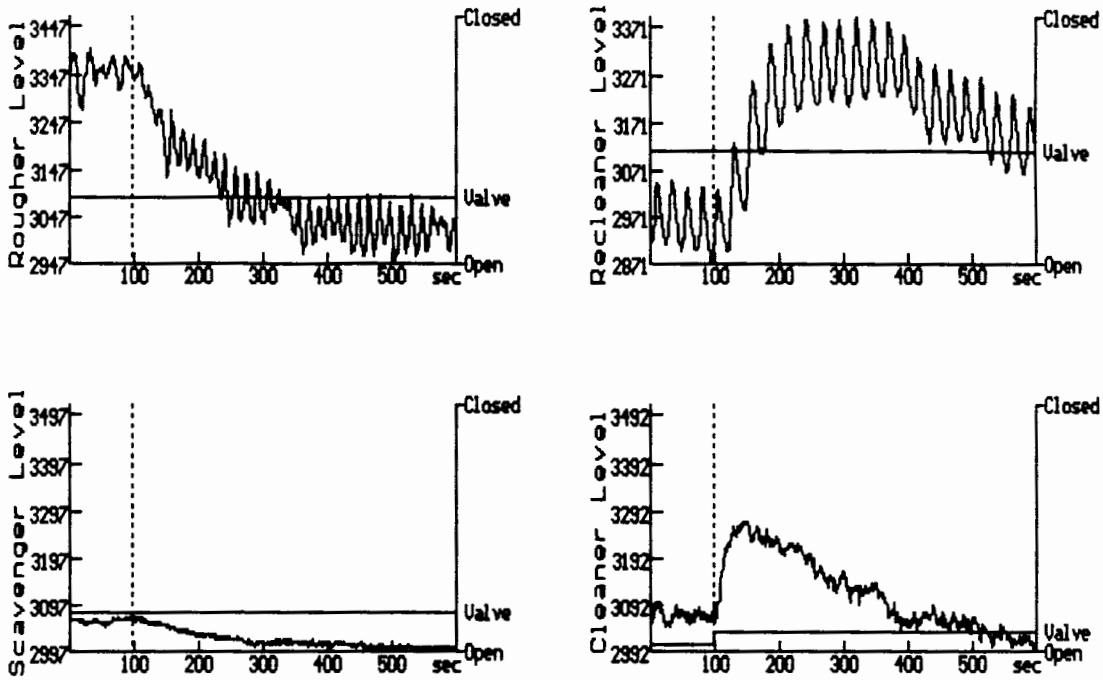
Flotation System Open-Loop Step Response

Test 2d) Scavenger Step Size= -814 Units (-19%)



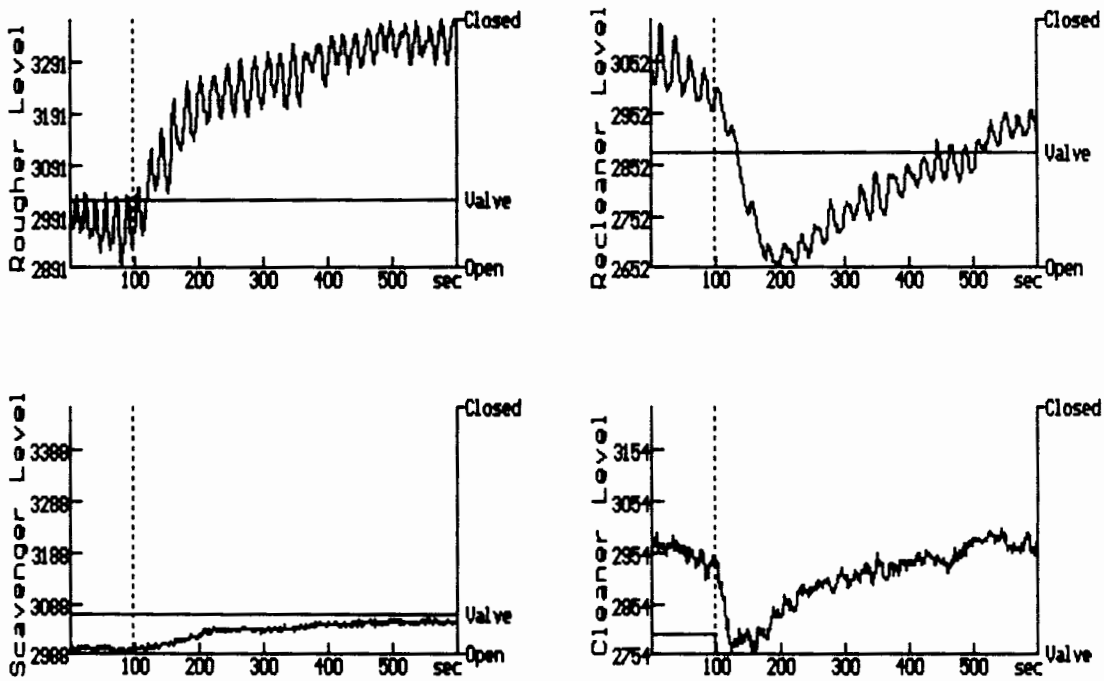
Flotation System Open-Loop Step Response

Test 3a) Cleaner Step Size= 218 Units (5%)



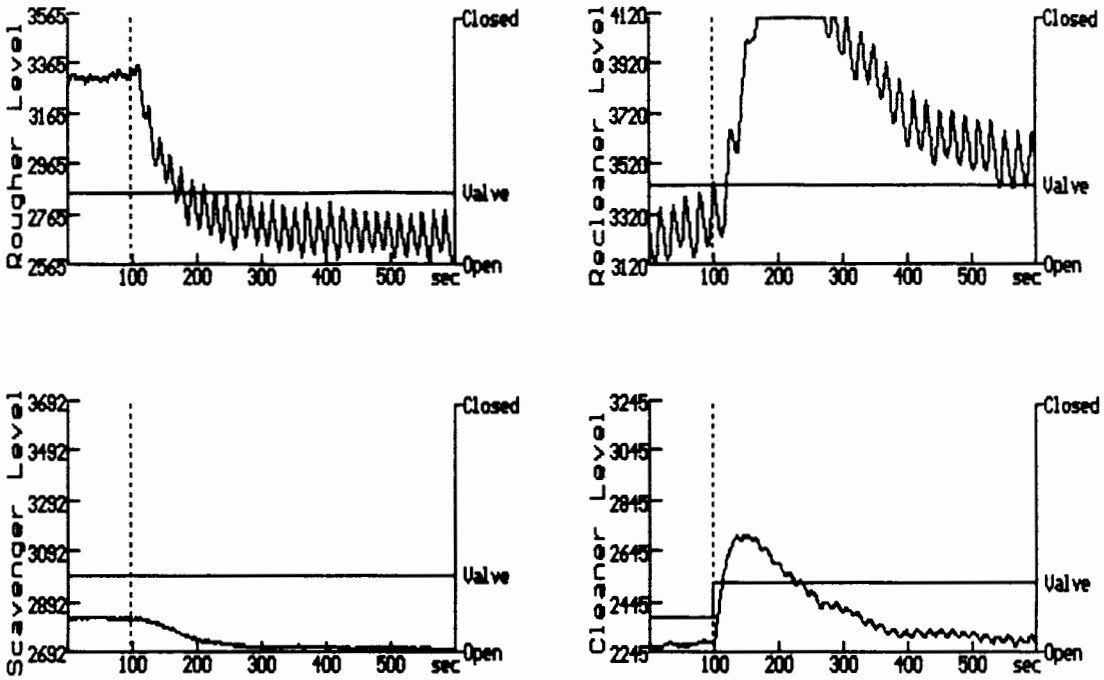
Flotation System Open-Loop Step Response

Test 3b) Cleaner Step Size= -368 Units (-8%)



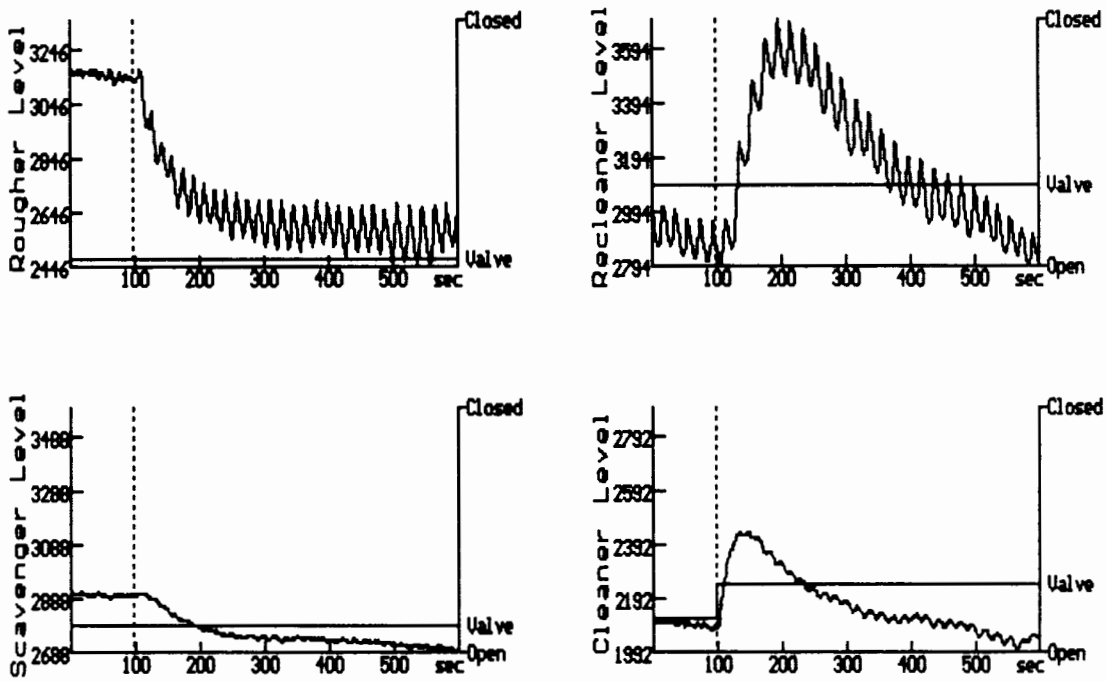
Flotation System Open-Loop Step Response

Test 3c) Cleaner Step Size= 611 Units (15%)



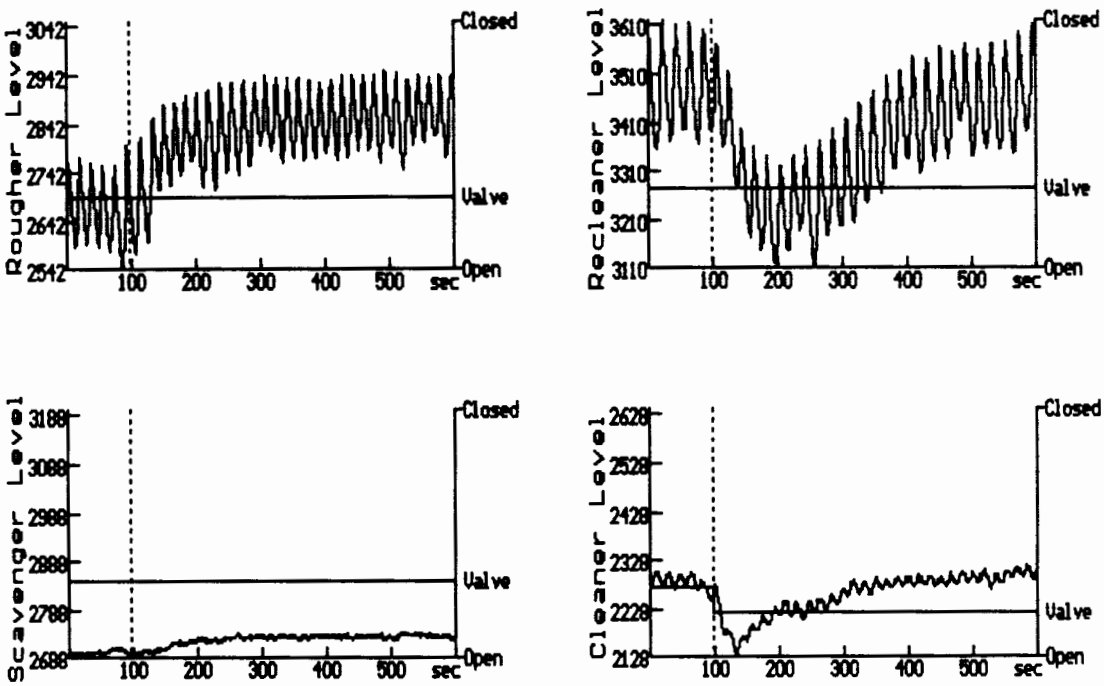
Flotation System Open-Loop Step Response

Test 3d) Cleaner Step Size= 604 Units (15%)



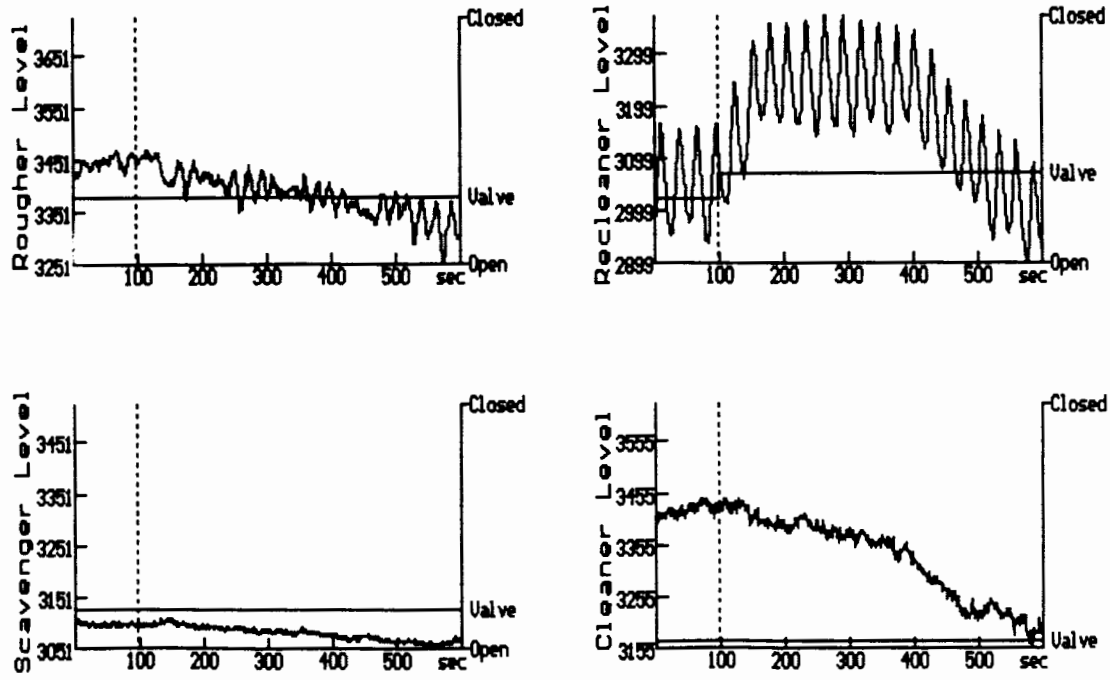
Flotation System Open-Loop Step Response

Test 3a) Cleaner Step Size= -409 Units (-9%)



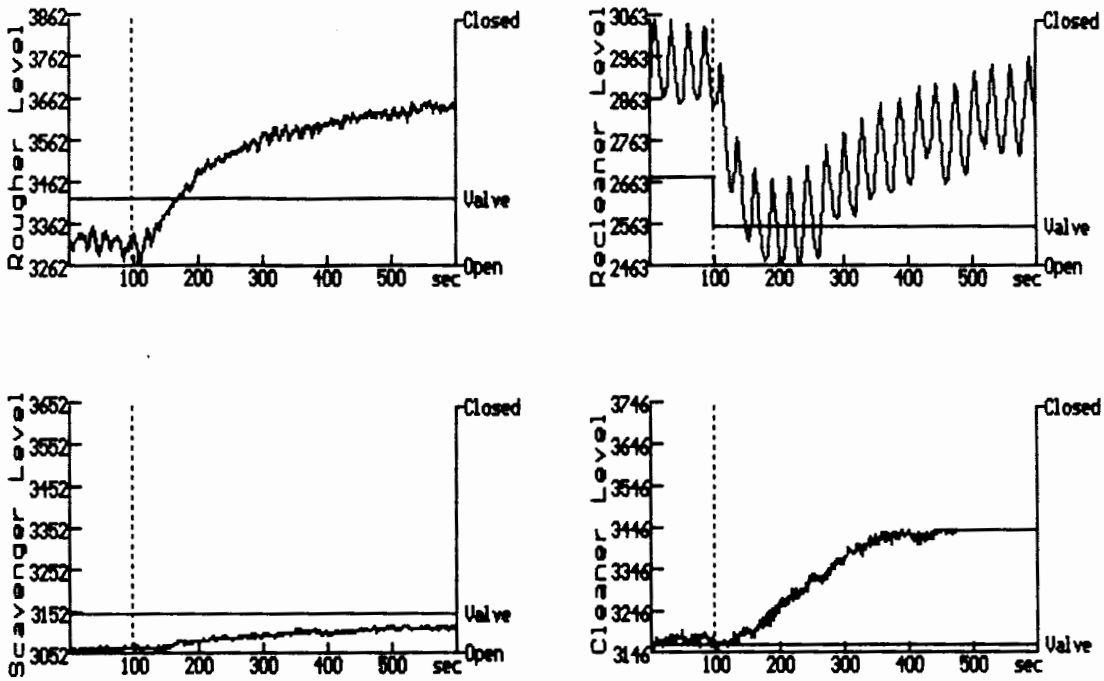
Flotation System Open-Loop Step Response

Test 4a) Recleaner Step Size= 416 Units (10%)



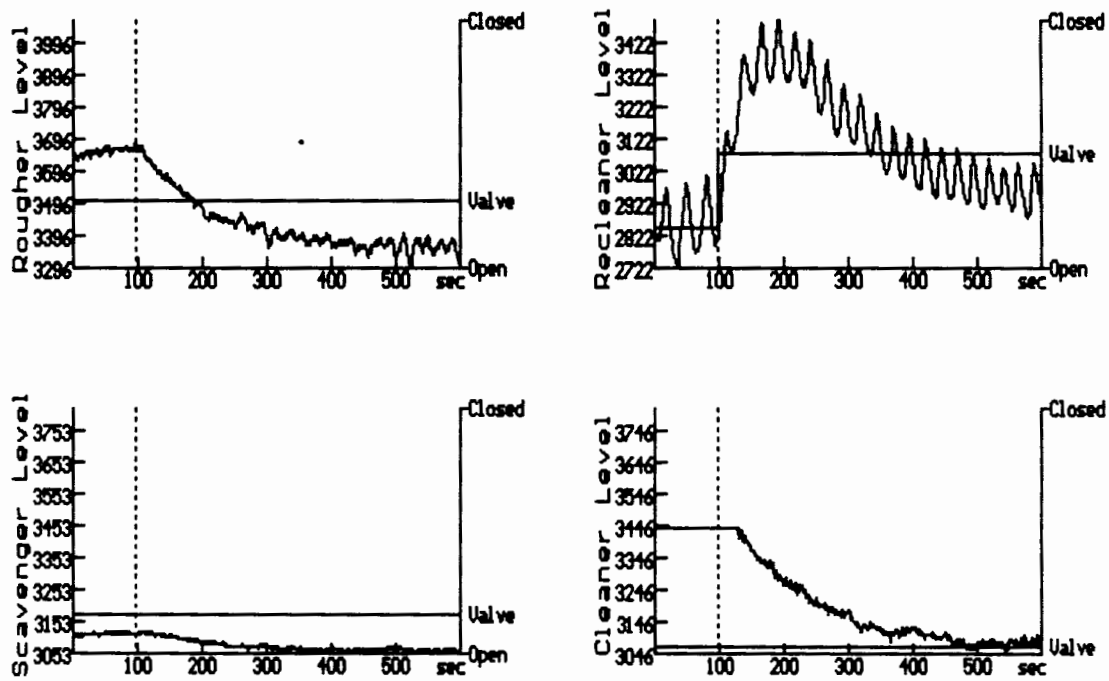
Flotation System Open-Loop Step Response

Test 4b) Recleaner Step Size= -819 Units (-19%)



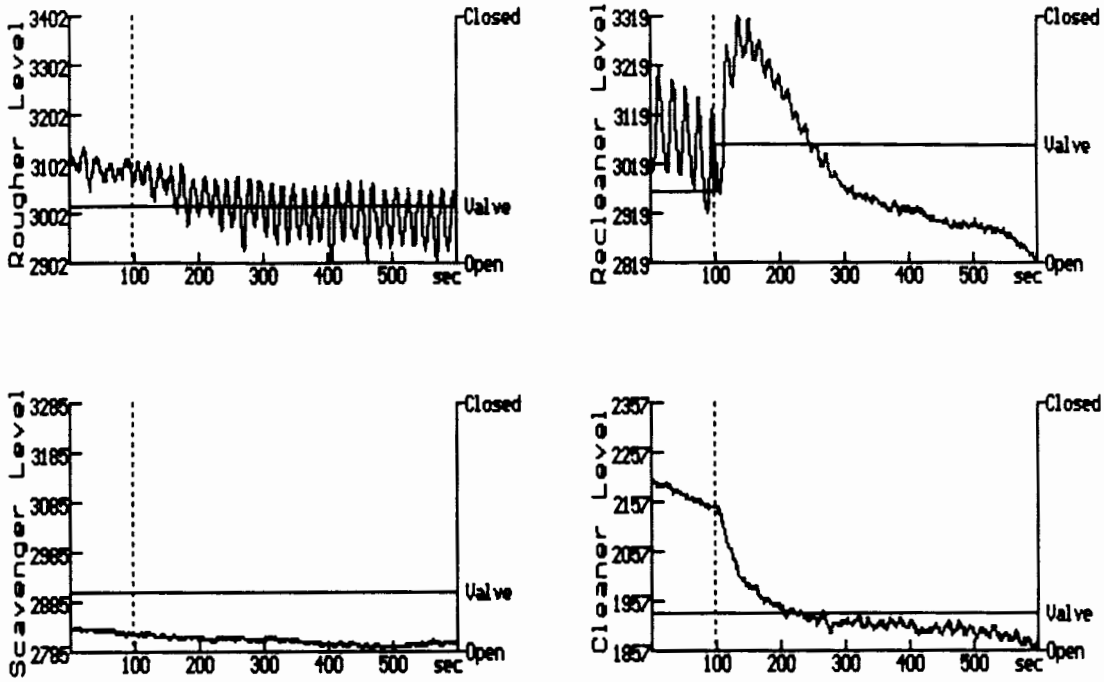
Flotation System Open-Loop Step Response

Test 4c) Recleaner Step Size= 1228 Units (30%)



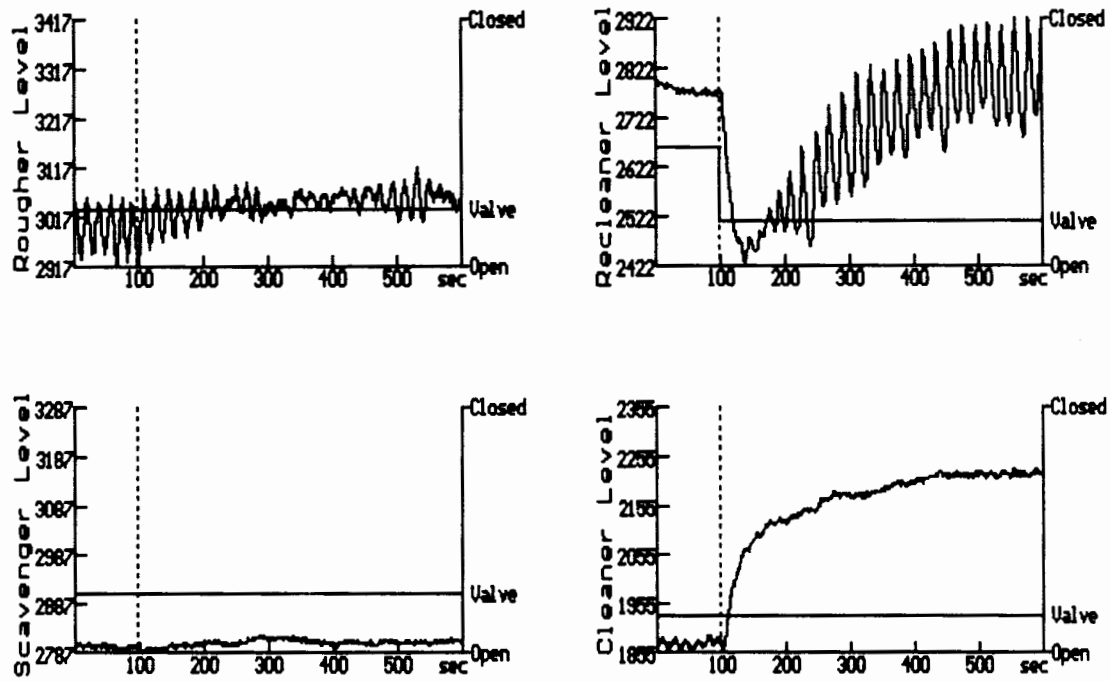
Flotation System Open-Loop Step Response

Test 4d) Recleaner Step Size= 812 Units (20%)



Flotation System Open-Loop Step Response

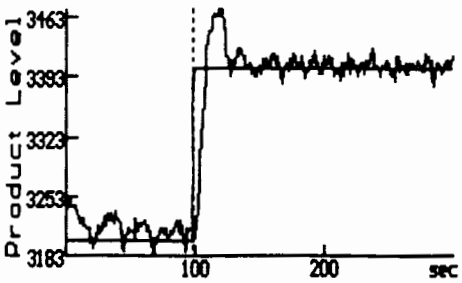
Test 4e) Recleaner Step Size= -1228 Units (-29%)



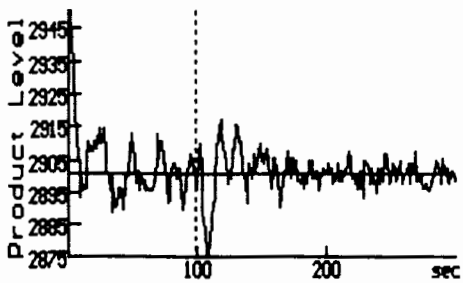
B) CLOSED-LOOP STEP RESPONSE (INA TECHNIQUE)

Closed-Loop Step Response

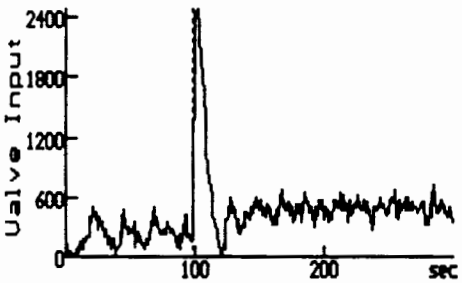
Rougher Step Size= 200 Units

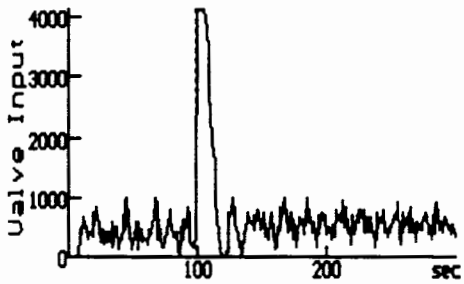


Rougher Tank



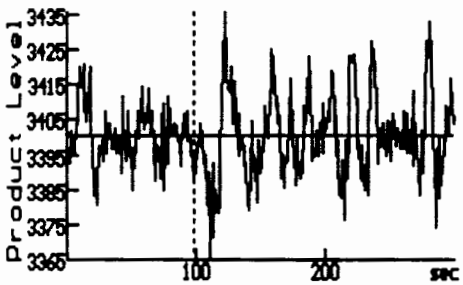
Scavenger Tank



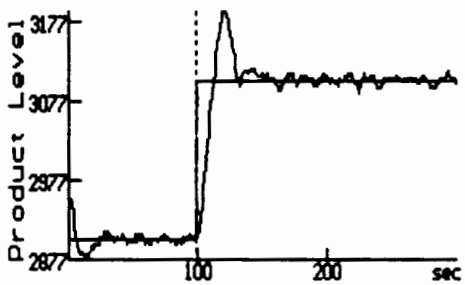


Closed-Loop Step Response

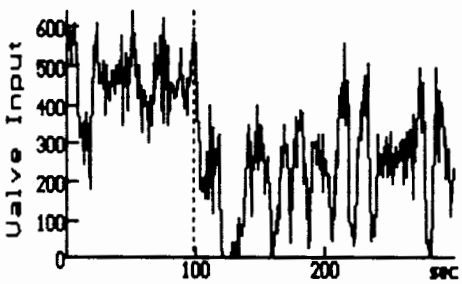
Scavenger Step Size= 200 Units

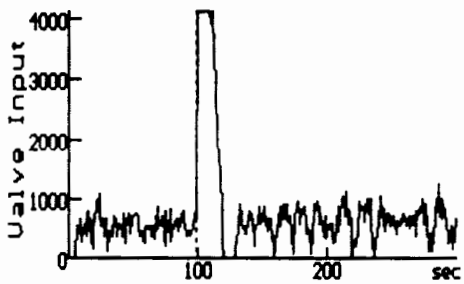


Rougher Tank



Scavenger Tank

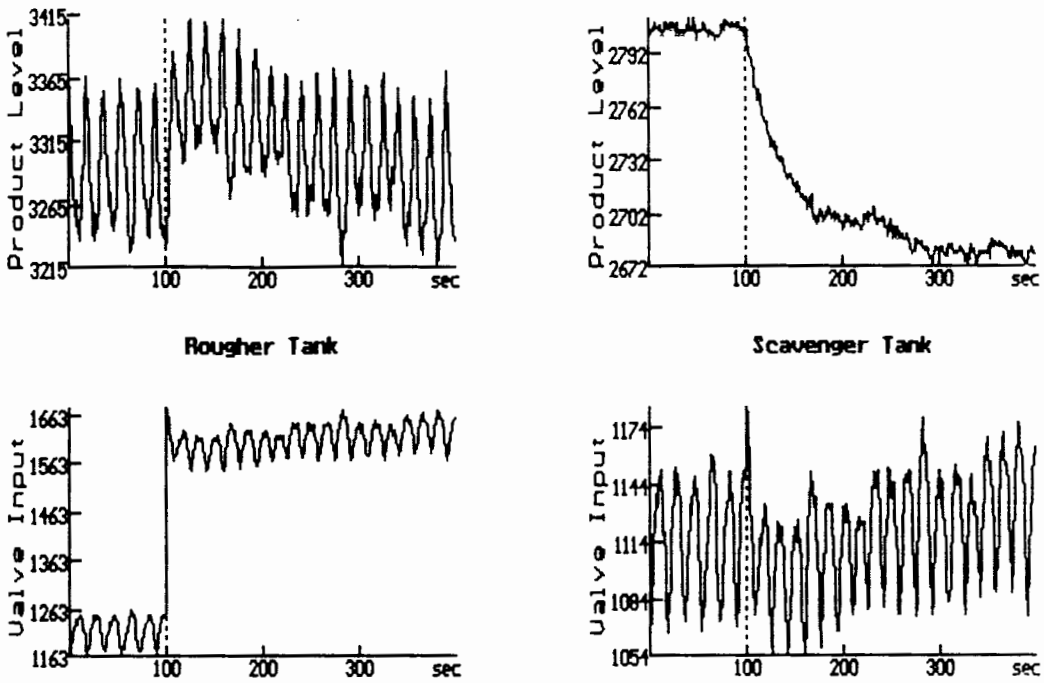




C) CLOSED-LOOP STEP RESPONSE (POLE ASSIGNMENT)

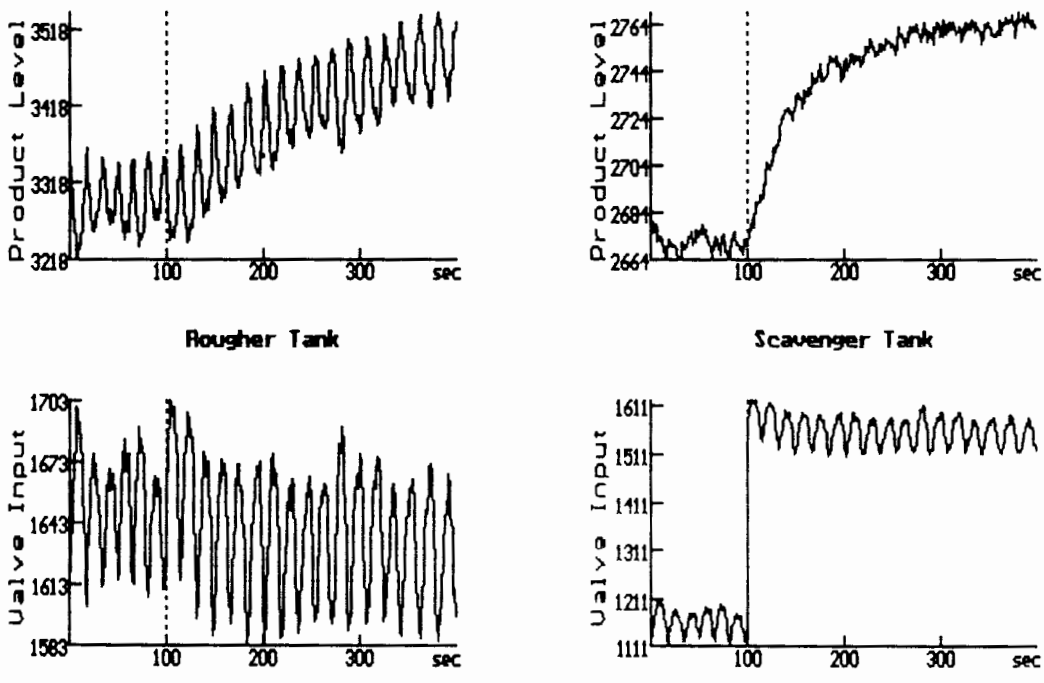
Flotation System Closed-Loop Step Response

Rougher Step Size= 410 Units (10%)



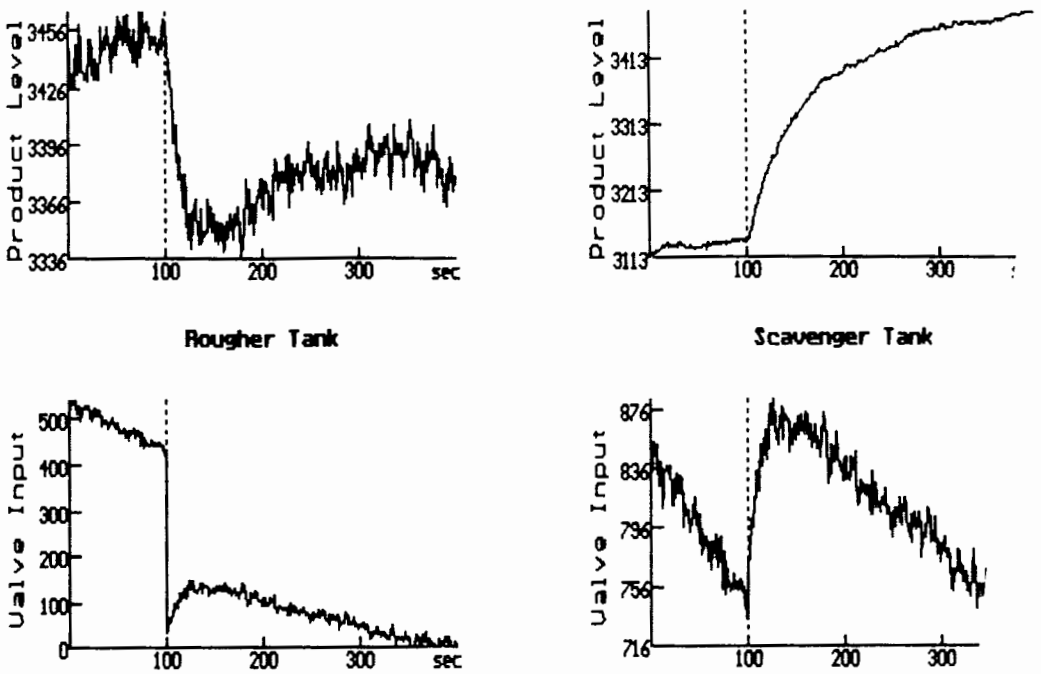
Flotation System Closed-Loop Step Response

Scavenger Step Size= 410 Units (10%)



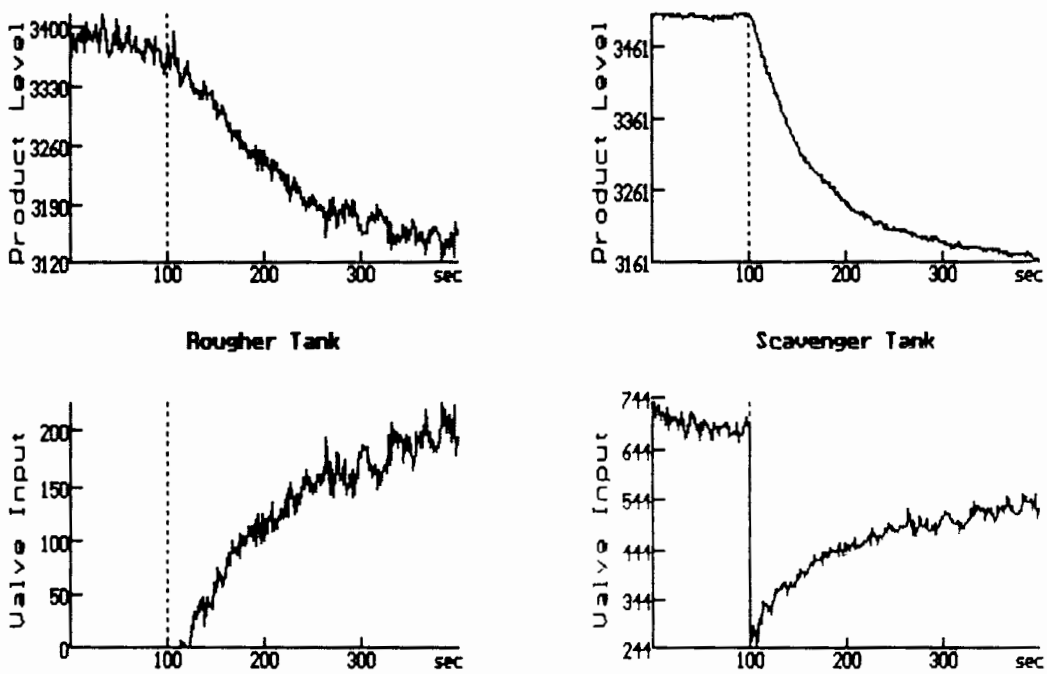
Flotation System Closed-Loop Step Response

Rougher Step Size= -410 Units (-9%)



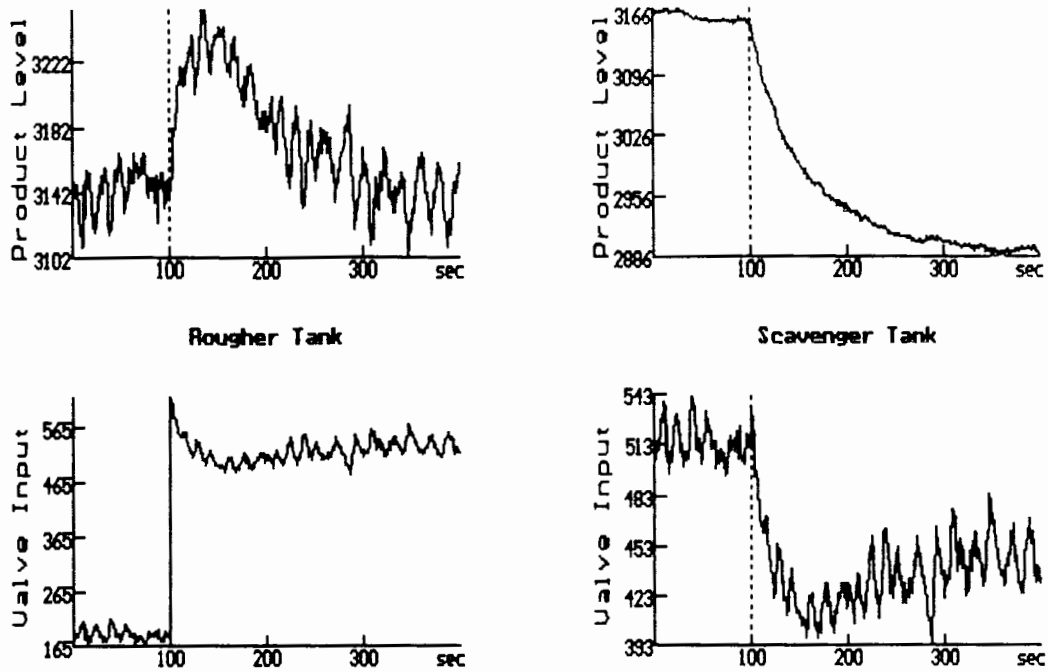
Flotation System Closed-Loop Step Response

Scavenger Step Size= -410 Units (-9%)



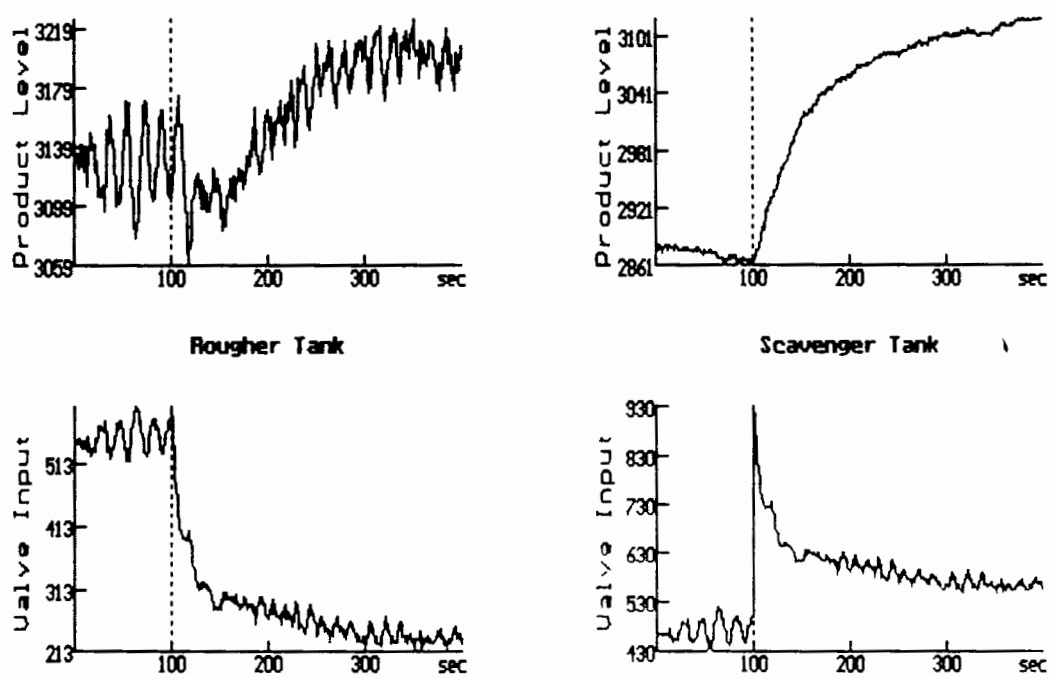
Flotation System Closed-Loop Step Response

Rougher Step Size= 410 Units (10%)



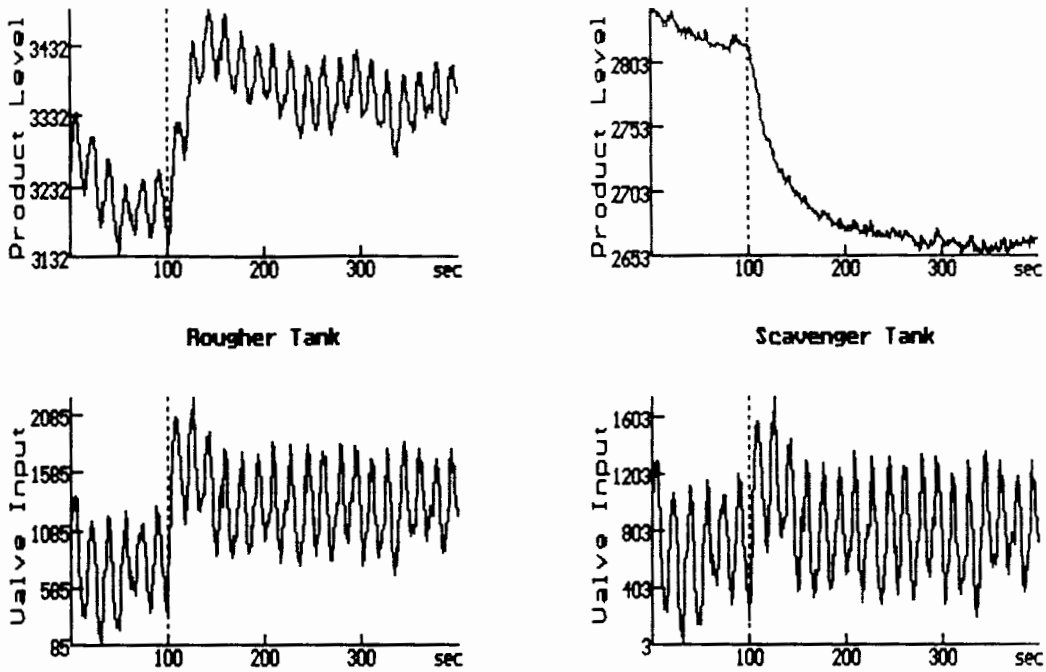
Flotation System Closed-Loop Step Response

Scavenger Step Size= 410 Units (10%)



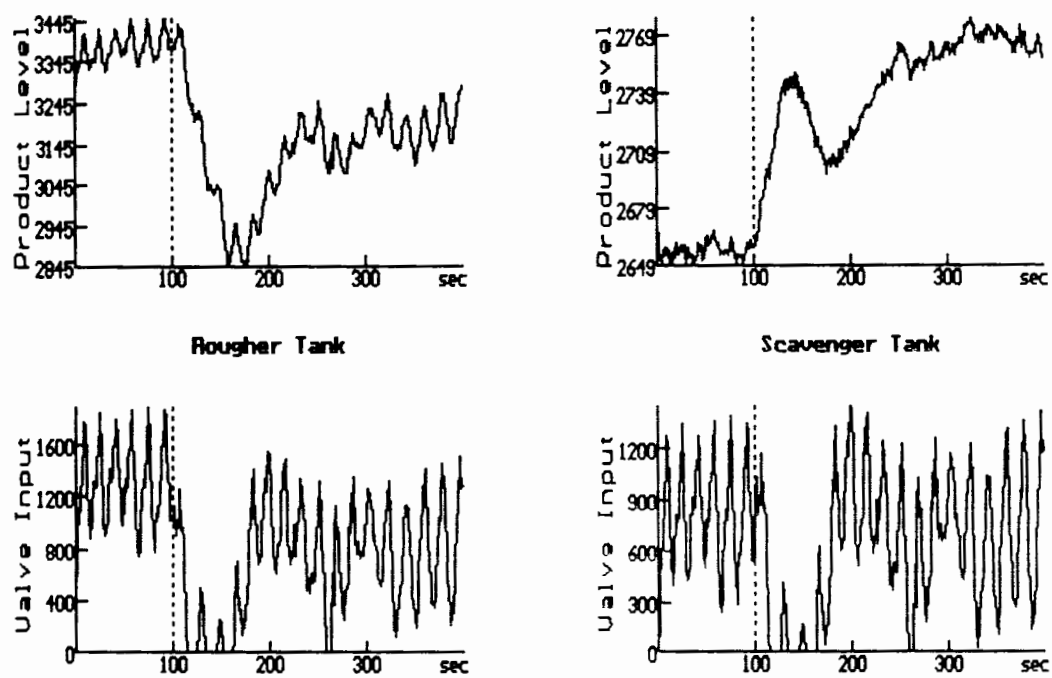
Flotation System Closed-Loop Step Response

Rougher Step Size= 410 Units (10%)



Flotation System Closed-Loop Step Response

Scavenger Step Size= 410 Units (10%)



APPENDIX C CLOSED-LOOP STEP RESPONSES (POLE ASSIGNMENT)

D) SOFTWARE REFERENCE

1. INACAD PROGRAM SOURCE LISTING

1.1 SUBROUTINE OVERVIEW

addelm	Add two functions in s
addpoly	Add two polynomials in s
calc2wf	Calculate frequency points
calcf	Calculate frequency points
calcpw	Calculate polynomial for given s
calcqi	Calculate Q(s) for s=jw
calctf	Calculate frequency reponse of function
calcwf	Calculate frequency points
clrbot	Clear bottom line of display
clrdsn	Clear all design parameters
clrdsp	Clear status line
clreol	Clear to end of line
clrtop	Clear second status line
cls	Clear text screen
cominv	Invert matrix
comm2	Display command menu
commnd	Display command menu
dcirc	Draw circle
deltmp	Delete temporary files
diagon	Diagonalisation menu
dimtf	Find dimension of transfer function
disk	Construct filenames
docol	Perform column operation
dorow	Perform row operation
dos	Enter DOS
drawc	Plot complex point
dsnrst	Initialise design parameters
editmlt	Edit multiplier
edittf	Edit transfer function
editwf	Edit frequency data file
erasel	Clear line of display
error	Error routine
frange	Select frequency range
getchc	Get character from keyboard
getname	Get filename
getord	Get process order
getqu2	Calculate multiplier for row/col multiply
getquot	Calculate multiplier for row/col operation
getscale	Get graph scale from operator
gotorc	Move cursor
help	Display help file
inacad	**Main program**
inadsn	INA design routine

inputtf	Input transfer function coefficients
invers	Display characters in inverse
kxcol	Multiply column by function
kxrow	Multiply row by function
lnwf	Natural logarithmic frequency table
loadrc	Get old cursor position
loadtf	Load transfer function from disk
loadwf	Load frequency table from disk
logwf	Logarithmic frequency table
mark	Draw cross at given point
matmlt	Multiply two matrices
mcircle	Draw M=1.3 circle
mltcol	Multiply column by function
mltrow	Multiply row by function
mnurst	Initialise menus
multpoly	Multiply two polynomials
nelm	Optimisation routine
normal	Print characters white on black
nyqstp	Nyquist diagram options
optim	Optimisation routine
pltax2	Plot axis
pltaxs	Plot axis
pltger	Plot Gershgorin circles
plthe2	Plot heading
plthed	Plot heading
pltina	Plot INA diagram
pltmlt	Plot multiplier
pltpts	Plot frequency points
polyxe	Expand e^x to series expansion
prcpar	Display design parameters
printtf	Print transfer function
prtml2	Print multiplier
prtmlt	Print multiplier
prtnyq	Print frequency reponse
prtplt	Print multiplier points
rcmlt	Perform row/column multiplication
readln	Read input
redokl	Perform operation
redowf	Recalculate frequency points
rowope	Do row operation
saverc	Save cursor position
savetf	Save transfer function to disk
savewf	Save frequency table to disk
setmde	Set text mode
setpfd	Select parameter file directory
shwpar	Show parameter selections
specwf	Calculate special frequency table
swop	Swop rows or columns
swopcol	Swop columns
swoprow	Swop rows
syscal	Perform DOS system call

```
undoch          Undo change
wait            Display wait message
```

1.2 SUBROUTINE INDEX


```
addelm(mltn,mltd,mltl,poly1n,poly1d,poly1l,          286
addpoly(result,poly1,poly2,mxsord)                   287
calc2wf(stdect,endect)                               209
calcf(n,x,errsum)                                    252
calcpw(result,poly,mxsord,w)                         206
calcqi(qw,calcg,calck,calcl,calcf,error,det)         206
calctf(w,noofw,kn,kd,kl,mxsord,order,                208
calcwf(stdect,endect)                                209
clrbot                                               205
clrdsn                                               238
clrdsp                                               204
clreol                                               205
clrtop                                               205
cls                                                  204
cominv(b,n,c)                                        211
comm2                                                214
commnd                                               214
dcirc(rad,i,j,qw,order,dim)                          262
deltmp                                               301
diagon                                               215
dimtf(order,mxsord,sorder,kn,kd)                     216
disk(concat,dir,name,ext)                            217
docol(rkn,rkd,rkl,knum,kden,klag,                    285
dorow(rkn,rkd,rkl,knum,kden,klag,                    286
dos                                                  301
drawc(i,j,qw,offset,order,dim)                       217
dsnrst                                               218
editmlt(order,sorder,kd,kn,kl,mxsord,change)         219
edittf(name,ext,ext2,arrn,arrd,arrl,exist)           222
editwf(w,noofw,wexist,filen)                         224
erasel                                               204
error(message)                                       227
frange                                               228
getchc(mnunb,escape)                                 228
getname(dir,name,ext,load,exist,escape)              230
getord                                               232
getqu2                                               233
getquot                                              234
getscale                                             235
gotorc(row,column)                                   204
help                                                 236
inadsn                                               237
inputtf(fname,ext,order,sorder,kd,kn,kl,             239
invers                                               205
kxcol(rkn,rkd,rkl,lnum,                              243
```

kxrow(rkn,rkd,rkl,knum, 242
lnwf(stdect,endect) 211
loadrc 204
loadtf(dir,name,ext,arrn,arrd,arrl, 243
loadwf(dir,name,ext,arr,noofw,exist) 245
logwf(stdect,endect) 210
mark(i,j) 218
matmlt(mxa,mxb,mxc,order) 246
mcircle(single,si,sj,order,inadim) 269
mltcol(mltn,mltd,mltl,rkn,rkd,rkl,knum, 287
mltrow(mltn,mltd,mltl,rkn,rkd,rkl,knum, 286
mnurst 246
multpoly(result,poly1,poly2,mxsord) 287
nelm(ix,x,h,n,f,eps,err) 249
normal 205
nyqstp 253
optim 257
pltax2 259
pltaxs(order) 260
pltger(qw,single,si,sj,order, 261
plthe2(start) 263
plthed(cls) 265
pltina 266
pltmlt 270
pltpts(qw,single,si,sj,order,dim,w) 272
polyxe(result,poly,t,mxsord) 288
prcpar 273
printtf(pfdir,name,ext,arrn,arrd,arrl,order,mxsord) 274
program inacad 201
prtml2 276
prtmlt 275
prtnyq 278
prtplt 280
rcmlt 289
readln(input,max,escape) 283
redokl 284
redowf 288
rowope 291
saverc 204
savetf(dir,name,ext,arrn,arrd,arrl, 294
savewf(dir,name,ext,arr,noofw,exist) 295
setmde 205
setpfd 295
shwpar(wait) 296
specwf(stdec,enddec) 210
swop 297
swopcol(knum,kden,klag,order,mxsord,ri,ruse) 300
swoprow(lnum,lden,llag,order,mxsord,rj,cuse) 300
syscal(command) 301
undoch 302
wait 206

1.3 INCLUDE FILE LISTING

file ADD.BLK - Debug option

```
$DEBUG or $NODEBUG
```

file DESDTA.BLK - Global temporary storage

```
        integer   noofw
        logical   ksaved,lsaved
        real      w(200)
        real      gnum(1000),gden(1000),glag(100)
        real      knum(1000),kden(1000),klag(100)
        real      lnum(1000),lden(1000),llag(100)
        real      fnum(1000),fden(1000),flag(100)
        common    /desdta/
     +            w,noofw,gnum,gden,glag,knum,kden,klag,
     +            lnum,lden,llag,fnum,fden,flag,
     +            ksaved,lsaved
```

file DESIGN.BLK - Design parameters

```
        integer      order,mxsord,si,sj,gerint,ostint
        logical      qexist,qinvrt,gexist,kexist,lexist,fexist
        logical      wexist,single
        real         inadim,xdim,ydim
        character*8  gfilen,kfilen,lfilen,ffilen,wfilen
        character*20 tfdir,pfdir,dsnnme
        common       /design/
     +               order,mxsord,si,sj,gerint,ostint,
     +               qexist,qinvrt,gexist,kexist,lexist,fexist,
     +               wexist,single,inadim,xdim,ydim,
     +               gfilen,kfilen,lfilen,ffilen,wfilen,dsnnme,
     +               tfdir,pfdir
```

file DIM.BLK - Graph axis scale selection

```
          real         xlo,xhi,ylo,yhi
          common/dim/  xlo,xhi,ylo,yhi
```

file MENU.BLK - Menu structure

```
        logical      escape
        integer      menunb
        integer      depth,oldmnu(10)
        integer      chcdfl(10)
        integer      top(2)
        integer      mnuinf(2,10)
        character*20 choice(60)
        character*45 messge(60)
        common       /menu/
     +               mnuinf,choice,messge,depth,
     +               chcdfl,top,oldmnu,menunb
```

file ROWOP.BLK - Row operation selections

```
      integer      ri,rj,ruse,cuse,rsord,wint
      real         rkd(21),rkl,rkn(21),stp,eps
      complex      quotient(200)
      common       /rowop/
     +             ri,rj,ruse,cuse,rsord,
     +             rkd,rkl,rkn,stp,eps,wint,
     +             quotient
```

1.4 SOURCE LISTING

```
$include:'add.blk'
*******************************************************************************
* INACAD * PROGRAM TO DO INA DESIGN                                           *
*                                                                             *
*          written by R.VENZKE        All rights reserved. (c)86,87,88        *
*                                                                             *
*******************************************************************************
* Descr:   Uses the Inverse Nyquist Array technique to help design            *
*          a controller for a multivariable control system.                   *
*                                                                             *
* Notes:   The program must be linked with FOR2GRPH.OBJ (Graphx Support)      *
*          and with MICROSOFT FORTRAN C LIBRARY for SYSTEM function and       *
*          C type strings.                                                    *
*                                                                             *
*          EXECUTION: Before INACAD can be used INT10 and HARDCOPY            *
*                     must be run for GRAPHX support.                         *
*                                                                             *
* File Support : INACAD.HLP - Online help fascility.                          *
*                                                                             *
* Calls:   ansisup,DSNRST,CLRDSN,MNURST,INADSN,DELTMP                         *
*******************************************************************************

      program inacad
      integer i
      character*30 copy

$INCLUDE:'DESIGN.BLK'
$INCLUDE:'MENU.BLK'
$INCLUDE:'DESDTA.BLK'

      copy='(c) RHE Venzke 86,87,88'
      write(*,*)' INACAD  Multi-Variable Control System',
     +' Design Aid  ',copy
      write(*,*)' '
      write(*,*)' '
      tfdir=' '
      pfdir='a:'
      open(1,file='default.cad',iostat=iocheck)
      read(1,'(a40)',err=4) tfdir
      read(1,'(a40)',err=4) pfdir
4     close(1)
      call dsnrst
      call clrdsn
      call mnurst
      open(1,file='default.ws',iostat=iocheck)
      read(1,'(i3)',err=2)noofw
      do 1 i=1,noofw
1       read(1,'(e13.7)',err=2)w(i)
```

```
        close(1)
        goto 3
2         close(1)
          noofw=51
          call calcwf(6,2)
3       inadim=20.
        multdim=2.
        wfilen='Default '
        call setmde
        call inadsn
        call deltmp
        open(1,file='default.cad',status='new')
        write(1,'(a40)',err=5) tfdir
        write(1,'(a40)',err=5) pfdir
5       close(1)
        call cls
        write(*,*)
     +   ' INACAD ',copy,' terminated.'
        end

c       Interface to Assemply routine QQGETCH

        integer function getch
        integer  qqgetch,i
        external qqgetch

        i=qqgetch()

c       Convert to lowercase

        if ((i.gt.64).and.(i.le.90)) i=i+32
        getch=i
        return
        end

c       Interface to Assemply routine QQREADCH

        integer function readch
        integer  qqreadch
        external qqreadch

        readch=qqreadch()
        return
        end


;*****************************************************************************
; IOSUP.ASM  I/O SUPPLEMENT - Assembly language                                *
;*****************************************************************************
; Descr:   The following Input routines were defined from Assembly language  *
;          to improve FORTRAN I/O.                                           *
;                                                                            *
;*****************************************************************************

TITLE I/O LIBRARY

         ;SUBTL Copyright 1986 by R.Venzke
         ;LAST REVISION: 27-9-1986
         ;NAME  I/O support for MS-Fortran

DATA     SEGMENT PUBLIC 'DATA'
```

```
DATA    ENDS

DGROUP  GROUP DATA

CODE    SEGMENT 'CODE'

ASSUME  CS:CODE
ASSUME  DS:DGROUP
ASSUME  SS:DGROUP

        ;QQReadch is a function returning ASCI code of key typed
        ;no echo
        ;INTEGER*4 READCH

PUBLIC  QQREAD
QQREAD  PROC FAR
        PUSH        BP
        MOV         BP,SP
        MOV         AH,7
        INT         21H
        MOV         AH,0
        MOV         DX,0
        MOV         SP,BP
        POP         BP
        RET
QQREAD  ENDP

        ;QQGetch flushes the input buffer and reads a key
        ;echo turned off
        ;INTEGER*4 GETCH

PUBLIC  QQGETC
QQGETC  PROC FAR
        PUSH        BP
        MOV         BP,SP
        MOV         AH,0CH
        MOV         AL,7
        INT         21H
        MOV         AH,0
        MOV         DX,0
        MOV         SP,BP
        POP         BP
        RET
QQGETC  ENDP

       ;Kbhit is a function to test if a key was pressed
       ; (returns TRUE if so)
       ;LOGICAL*4 KBHIT =0 if nokey was pressed

PUBLIC  KBHIT
KBHIT   PROC FAR
        PUSH        BP
        MOV         BP,SP
        MOV         AH,0BH
        INT         21H
        MOV         AH,AL
        MOV         DX,AX
        MOV         SP,BP
        POP         BP
        RET
KBHIT   ENDP
```

```
CODE    ENDS

        END


$include:'add.blk'
********************************************************************************
*  ANSISUP.FOR   ANSI.SYS SUPPLEMENT                                           *
********************************************************************************
* Descr: Screen interface subroutines. Uses ANSI.SYS driver to manipulate     *
*        screen output.                                                       *
*                                                                             *
* Calls:   none                                                               *
********************************************************************************


c Clear the screen

        subroutine cls
        write(*,'(/1x,A1,A3,\)')27,'[0m'
        write(*,'(A1,A3,\)')27,'[2J'
        return
        end

c Clear the bottom line

        subroutine clrdsp
        call normal
        call gotorc(25,1)
        call clreol
        return
        end

c Save cursor position

        subroutine saverc
        write(*,'(a1,a2,\)')27,'[s'
        return
        end

c Restore cursor position

        subroutine loadrc
        write(*,'(a1,a2,\)')27,'[u'
        return
        end

c Move cursor to ROW, COLUMN position

        subroutine gotorc(row,column)
        integer row,column
        write(*,'(A1,A1,I2.2,A1,I2.2,A1,\)')27,'[',row
     +,';',column,'H'
        return
        end

c Erase line

        subroutine erasel
        write(*,'(A1,A1,a1,\)')27,'[','K'
```

```
        return
        end

c Clear to end of line

        subroutine clreol
        write(*,'(A1,A1,a1,\)')27,'[','K'
        return
        end

c Set black and white 80x25 text mode

        subroutine setmde
        write(*,'(/1X,A1,A4,\)')27,'[=2h'
        return
        end

c Set foreground WHITE and background BLACK

        subroutine normal
        write(*,'(A1,A3,\)')27,'[0m'
        return
        end

c Set inverse character display

        subroutine invers
        write(*,'(A1,A3,\)')27,'[7m'
        return
        end

c Clear bottom line using inverse bar

        subroutine clrbot
        call gotorc(25,1)
        call invers
        call erasel
        return
        end

c Clear 24th row

        subroutine clrtop
        call gotorc(24,1)
        call normal
        call erasel
        return
        end

c Convert character to lower case

        integer function lowcase(i)
        integer i

        if ((i.ge.65).and.(i.le.90)) then
         i=i+32
        endif
        lowcase=i
        return
        end
```

```
c Display wait message

        subroutine wait
        call normal
        call gotorc(25,1)
        call clreol
        call gotorc(25,64)
        write(*,'(a\)') ' Please wait...'
        return
        end


$INCLUDE:'ADD.BLK'
*******************************************************************************
* CALCPW * CALCULATE POLYNOMIAL AT (0,jw)                                     *
*******************************************************************************
* Descr:   The polynomial specified by real coefficients of POLY is evalu-    *
*          ated at (0,jw) as given by W.                                      *
*                                                                             *
* Param:   RESULT is the complex result of the subroutine.                    *
*          MXSORD is the maximum order of s that can be stored in POLY.       *
*          W is the real coefficient of jw                                    *
*                                                                             *
* Calls:   none                                                               *
*******************************************************************************

        subroutine calcpw(result,poly,mxsord,w)
        integer    mxsord
        real       poly(*),w
        complex    result,jw
        complex*16 sum

        sum=cmplx(poly(mxsord+1),0.)
        jw=cmplx(0.,w)
        do 1 i=mxsord,1,-1
1         sum=sum*jw+cmplx(poly(i),0.)
        result=sum
        return
        end


$include:'add.blk'
*******************************************************************************
* CALCQI * CALCULTE INA MATRIX Q AT FREQUENCY WI                              *
*******************************************************************************
* Descr:   This subroutine calculates the INA response for one frequency      *
*          interval. File 1= temp.gs 2= temp.ks 3=temp.ls 4=temp.fs           *
*          are read from and result is written to file 9= temp.plt.           *
*                                                                             *
* Param:   ORDER order of matrices.                                           *
*          QW storage for resulting response.                                 *
*          CALCG,CALCK,CALCL,CALCF logical to signify if matrices exist       *
*          INVERSE logical set true for INVERSE Nyquist                       *
*          ERROR = true if a file access error occurred.                      *
*          DET is the determinant of the resulting mx.                        *
*                                                                             *
* Call.by: PLTINA                                                             *
* Calls:   none                                                               *
*******************************************************************************

        subroutine calcqi(qw,calcg,calck,calcl,calcf,error,det)
```

```
        complex  qw(*),mxb(100),det
        logical  calcg,calck,calcl,calcf,error
        integer  i,j,offset

$include:'design.blk'

        error=.false.
        if (calcg) then
          do 1 i=1,order*order
1            read(1,err=99) qw(i)
        else
          do 2 i=1,order
            offset=(i-1)*order
            do 2 j=1,order
              if (i.eq.j) then
                qw(j+offset)=cmplx(1.0,0.0)
              else
                qw(j+offset)=cmplx(0.0,0.0)
              endif
2         continue
        endif
        if (calcl) then
          do 3 i=1,order*order
3           read(3,err=99) mxb(i)
          call matmlt(mxb,qw,qw,order)
        endif
        if (calck) then
          do 4 i=1,order*order
4           read(2,err=99) mxb(i)
          call matmlt(qw,mxb,qw,order)
        endif
        if (qinvrt) then
          call cominv(qw,order,det)
        else
          det=cmplx(1,0)
        endif
        do 6 i=1,order*order
6         write(9,err=99) qw(i)
        goto 7
99        error=.true.
7       return
        end


$include:'add.blk'
*****************************************************************************
* CALCTF * CALCULATE TRANSFER FUNCTION RESPONSE                             *
*****************************************************************************
* Descr:   The transfer function as specified in KN,KD,KL is evaluated      *
*          at the frequency samples as given by W. The resulting complex    *
*          table is stored in the specified file.                           *
*                                                                           *
* Param:   MXSORD is the maximum order of s that can be handled by KN,etc.  *
*          KN are the numerator coefficients                                *
*          KD are the denominator coefficients                              *
*          KL is the lag                                                    *
*          W frequency table                                                *
*          ORDER is the order of the transfer function matrix nxn           *
*          ERROR is set to true if division by zero occurs.                 *
*          NOOFW contains the number of frequency points.                   *
```

```
*          TFDIR is the subdirectory to use for filename.                    *
*          NAME is the name for the file to write to.                        *
*          EXT is the file extension to use.                                 *
*          QEXIST flag to indicate if transfer function has been changed.    *
*                                                                            *
* Call.by: EDITTF                                                           *
* Calls:   CALCPW,DIMTF,DISK,ERROR                                          *
******************************************************************************

        subroutine calctf(w,noofw,kn,kd,kl,mxsord,order,
     +                    name,tfdir,ext,qexist)

        character*8  name
        character*20 tfdir
        character*4  ext
        character*32 filen
        logical      qexist,err
        integer      iocheck
        integer      noofw,mxsord,order,wi,row,col
        integer      ij,ijk,ijw,sorder
        real         kd(*),kl(*),kn(*),w(*),pi
        complex      kw,nw,dw,lw
        integer      getch,key
        external     getch
        character*40 message

        err=.false.
        qexist=.false.
        pi=3.1415926536
        call wait
        call dimtf(order,mxsord,sorder,kn,kd)
        call disk(filen,tfdir,name,ext)
95      open(1,file=filen,form='binary',status='NEW',iostat=iocheck)
        do 1 wi=1,noofw
          do 1 row=1,order
            do 1 col=1,order
              ij=col+order*(row-1)
              ijk=1+(mxsord+1)*(ij-1)
              call calcpw(nw,kn(ijk),sorder,w(wi))
              call calcpw(dw,kd(ijk),sorder,w(wi))
              if (cabs(dw).le.1.e-8) then
                err=.true.
                kw=(0,0)
              elseif (kl(ij).eq.0) then
                kw=nw/dw
              else
                lw=cmplx(0.,amod( kl(ij)*w(wi) , 2 * pi ))
                kw=nw/dw*cexp(lw)
              endif
              write(1,err=92)kw
1       continue
        if (err) then
          call error('Division by zero ignored!    ')
        endif
        close(1,iostat=iocheck)
        if (iocheck.ne.0) goto 92
        goto 93
92      message=name
        message(9:40)=' removed from design!'
        call clrbot
       write(*,'(2a\)')  name,' write error! Press key to continue or
```

```
     +ESC to abort'
        if (getch().ne.27) goto 95
        call error(message)
        name=' '
93      return
        end


$include:'add.blk'
*******************************************************************************
* CALCWF * CALCULATE  FREQUENCY SAMPLES                                       *
*******************************************************************************
* Descr:   The frequency sample table is initialised to the desired values.  *
*                                                                             *
* Param:   STDEC start decade                                                 *
*          ENDDEC end decade                                                  *
*                                                                             *
* Call.by: EDITWF,DSNRST                                                      *
* Calls:   none                                                               *
*******************************************************************************

c SQRT-LOG

        subroutine calcwf(stdect,endect)
        integer stdec,enddec,i,stdect,endect

$include:'desdta.blk'

        stdec=2*stdect
        enddec=endect*2
        do 3 i=0,noofw-1
          x=(stdec+enddec)*i/(noofw-1.)+1.
          if ((x-aint(x)).lt..1) x=.1+aint(x)
          if (x.ge.stdec) then
            w(i+1)=(x-aint(x))*10**(aint(x)-stdec)
          else
            w(i+1)=(x-aint(x))/10**(stdec-aint(x))
          endif
3       continue
        do 4 i=1,noofw
4         w(i)=sqrt(w(i))
        return
        end

c SQRT-SQRT-LOG
        subroutine calc2wf(stdect,endect)

        integer stdec,enddec,i,stdect,endect

$include:'desdta.blk'

        stdec=4*stdect
        enddec=endect*4
        do 3 i=0,noofw-1
          x=(stdec+enddec)*i/(noofw-1.)+1.
          if ((x-aint(x)).lt..1) x=.1+aint(x)
          if (x.ge.stdec) then
            w(i+1)=(x-aint(x))*10**(aint(x)-stdec)
          else
            w(i+1)=(x-aint(x))/10**(stdec-aint(x))
          endif
```

```
3       continue
        do 4 i=1,noofw
4         w(i)=sqrt(sqrt(w(i)))
        return
        end

c SPECIAL
        subroutine specwf(stdec,enddec)

        integer stdec,enddec,cnt
        real size(18),x,step

$include:'desdta.blk'


        size(1)=1.0
        size(2)=1.1
        size(3)=1.2
        size(4)=1.3
        size(5)=1.4
        size(6)=1.5
        size(7)=1.6
        size(8)=1.7
        size(9)=1.8
        size(10)=1.9
        size(11)=2.0
        size(12)=3.0
        size(13)=4.0
        size(14)=5.0
        size(15)=6.0
        size(16)=7.0
        size(17)=8.0
        size(18)=9.0
        cnt=0
        do 1 i=stdec,enddec-1
          x=10.0**i
          do 1 j=1,18
            cnt=cnt+1
            if (cnt.le.200) w(cnt)=x*size(j)
1       continue
        cnt=cnt+1
        if (cnt.le.200) w(cnt)=10.0**i
        if ((noofw.eq.1).or.(cnt.le.noofw)) then
          noofw=cnt
        else
          step=1.0/noofw*cnt
          do 2 i=1,noofw
           j=step*i
           w(i)=w(j)
2         continue
        endif
        return
        end

c LOG
        subroutine logwf(stdect,endect)
        integer stdec,enddec,i,stdect,endect

$include:'desdta.blk'

        stdec=stdect
```

```
        enddec=endect
        do 3 i=0,noofw-1
          x=(stdec+enddec)*i/(noofw-1.)+1.
          if ((x-aint(x)).lt..1) x=.1+aint(x)
          if (x.ge.stdec) then
            w(i+1)=(x-aint(x))*10.0**(aint(x)-stdec)
          else
            w(i+1)=(x-aint(x))/10.0**(stdec-aint(x))
          endif
3       continue
        return
        end

c LN
        subroutine lnwf(stdect,endect)
        integer stdec,enddec,i,stdect,endect

$include:'desdta.blk'

        stdec=nint(stdect*log(10.)/log(2.74))
        enddec=nint(endect*log(10.)/log(2.74))
        do 3 i=0,noofw-1
          x=(stdec+enddec)*i/(noofw-1.)+1.
          if ((x-aint(x)).lt..1) x=.1+aint(x)
          if (x.ge.stdec) then
            w(i+1)=(x-aint(x))*EXP(aint(x)-stdec)
          else
            w(i+1)=(x-aint(x))/EXP(stdec-aint(x))
          endif
3       continue
        return
        end


$include:'add.blk'
*******************************************************************************
* COMINV * COMPLEX MATRIX INVERSION                                           *
*******************************************************************************
* Descr:   Inverts a complex matrix using the Gauss-Jordan                    *
*           ellimination with complete pivoting                               *
*                                                                             *
* Param:   B = Complex matrix                                                 *
*          N = Dimension of A where A is nxn                                  *
*          C = Determinant                                                    *
*                    .                                                        *
* Call.by: EDITTF                                                             *
* Calls:   none                                                               *
*******************************************************************************

          SUBROUTINE COMINV(B,N,c)
          INTEGER    IR(10),IC(10)
C                    IR,IC are the pivot point of the matrix
          COMPLEX*16 a(100),D,TEMP,HOLD
C                    A(100) double precision
          complex    b(*),c
          INTEGER    I,N,J,K,L,II,JJ,III,M,IJ,IIJ,IT,KI,KJ,LP

          do 101 i=1,n*n
101         a(i)=b(i)
          DO 10 I=1,N
            IR(I)=I
```

```
            IC(I)=I
10        CONTINUE
          D=(1.,0.)
          M=-N
          DO 70 I=1,N
            M=M+N+1
            IF (I.EQ.N) GOTO 40
              BIG=cdABS(A(M))
              II=I
              JJ=I
              DO 20 K=I,N
                LP=(I-2)*N+K
                DO 20 L=I,N
                  LP=LP+N
                  IF (CdABS(A(LP)).LE.BIG) GOTO 20
                    BIG=CdABS(A(LP))
                    II=K
                    JJ=L
20              CONTINUE
22              IF (II.EQ.I) GOTO 30
                  IJ=I-N
                  IIJ=II-N
                  DO 25 J=1,N
                    IJ=IJ+N
                    IIJ=IIJ+N
                    TEMP=A(IJ)
                    A(IJ)=A(IIJ)
                    A(IIJ)=TEMP
25                CONTINUE
                  IT=IR(I)
                  IR(I)=IR(II)
                  IR(II)=IT
                  D=-D
30              IF (JJ.EQ.I) GOTO 40
                  IJ=(I-1)*N
                  IIJ=(JJ-1)*N
                  DO 35 J=1,N
                    IJ=IJ+1
                    IIJ=IIJ+1
                    TEMP=A(IJ)
                    A(IJ)=A(IIJ)
                    A(IIJ)=TEMP
35                CONTINUE
                  IT=IC(JJ)
                  IC(JJ)=IC(I)
                  IC(I)=IT
                  D=-D
40            if (cdabs(d*a(m)).lt.1.d-150) then
                c=(0,0)
                return
              endif
              D=D*A(M)
              TEMP=A(M)
              A(M)=(1.,0.)
              IJ=I-N
              DO 50 J=1,N
                IJ=IJ+N
                A(IJ)=A(IJ)/TEMP
50            CONTINUE
              KI=(I-1)*N
              DO 70 K=1,N
```

```
              KI=KI+1
              IF (K.EQ.I) GOTO 70
                TEMP=A(KI)
                A(KI)=(0.,0.)
                KJ=K-N
                IJ=I-N
                DO 60 J=1,N
                  KJ=KJ+N
                  IJ=IJ+N
                  HOLD=-TEMP*A(IJ)
                  A(KJ)=A(KJ)+HOLD
60              CONTINUE
70          CONTINUE
72          CONTINUE
            DO 80 I=1,N
              IF (IR(I).EQ.I) GOTO 80
              K=IR(I)
              JI=(I-1)*N
              JK=(K-1)*N
              DO 85 J=1,N
                JI=JI+1
                JK=JK+1
                TEMP=A(JI)
                A(JI)=A(JK)
                A(JK)=TEMP
85            CONTINUE
              IR(I)=IR(K)
              IR(K)=K
            GOTO 72
80          CONTINUE
82          CONTINUE
            DO 90 I=1,N
              IF(IC(I).EQ.I) GOTO 90
              K=IC(I)
              IJ=I-N
              KJ=K-N
              DO 75 J=1,N
                IJ=IJ+N
                KJ=KJ+N
                TEMP=A(IJ)
                A(IJ)=A(KJ)
                A(KJ)=TEMP
75            CONTINUE
              IC(I)=IC(K)
              IC(K)=K
              GOTO 82
90          CONTINUE
            do 100 i=1,n*n
100           b(i)=a(i)
            c=d
            RETURN
            END


$include:'add.blk'
*******************************************************************************
* COMM2  * SET UP COMMAND WINDOW                                              *
*******************************************************************************
* Descr:   The command window is set to the possible commands.                *
*          It is used when entering a transfer function.                      *
*                                                                             *
```

```
* Call.by: INPUTTF                                                              *
* Calls:   ansisup                                                              *
********************************************************************************

        subroutine comm2

        call invers
        call gotorc(5,62)
        write(*,'(a,\)')'    COMMANDS    '
        call gotorc(6,62)
        write(*,'(a,\)')'----------------'
        call gotorc(7,62)
        write(*,'(a,\)')' Select         '
        call gotorc(8,62)
        write(*,'(a,\)')'                '
        call gotorc(9,62)
        write(*,'(a,\)')'  <N>umerator   '
        call gotorc(10,62)
        write(*,'(a,\)')'                '
        call gotorc(11,62)
        write(*,'(a,\)')'  <D>enominator '
        call gotorc(12,62)
        write(*,'(a,\)')'                '
        call gotorc(13,62)
        write(*,'(a,\)')'  <L>ag         '
        call gotorc(14,62)
        write(*,'(a,\)')'                '
        call gotorc(15,62)
        write(*,'(a,\)')'  <P>ower of s  '
        call gotorc(16,62)
        write(*,'(a,\)')'                '
        call gotorc(17,62)
        write(*,'(a,\)')' <ESC> to Exit  '
        call gotorc(18,62)
        write(*,'(a,\)')'----------------'
        call normal
        return
        end


$include:'add.blk'
********************************************************************************
* COMMND * SET UP COMMAND WINDOW                                                *
********************************************************************************
* Descr:   The command window is set to the possible commands.                  *
*          It is used when entering a transfer function.                        *
*                                                                               *
* Call.by: INPUTTF                                                              *
* Calls:   ansisup                                                              *
********************************************************************************

        subroutine commnd

        call invers
        call gotorc(5,62)
        write(*,'(a,\)')'    COMMANDS    '
        call gotorc(6,62)
        write(*,'(a,\)')'----------------'
        call gotorc(7,62)
        write(*,'(a,\)')' Select         '
        call gotorc(8,62)
```

```
        write(*,'(a,\)')'                '
        call gotorc(9,62)
        write(*,'(a,\)')'  <N>umerator   '
        call gotorc(10,62)
        write(*,'(a,\)')'                '
        call gotorc(11,62)
        write(*,'(a,\)')'  <D>enominator '
        call gotorc(12,62)
        write(*,'(a,\)')'                '
        call gotorc(13,62)
        write(*,'(a,\)')'  <L>ag         '
        call gotorc(14,62)
        write(*,'(a,\)')'                '
        call gotorc(15,62)
        write(*,'(a,\)')'  <P>ower of s  '
        call gotorc(16,62)
        write(*,'(a,\)')'                '
        call gotorc(17,62)
        write(*,'(a,\)')'  <J>ump to R,C '
        call gotorc(18,62)
        write(*,'(a,\)')'                '
        call gotorc(19,62)
        write(*,'(a,\)')' <RTN> to Skip  '
        call gotorc(20,62)
        write(*,'(a,\)')'                '
        call gotorc(21,62)
        write(*,'(a,\)')' <ESC> to Exit  '
        call gotorc(22,62)
        write(*,'(a,\)')'----------------'
        call normal
        return
        end

$include:'add.blk'
********************************************************************************
* DIAGON * DO DIAGONALISATION DESIGN                                           *
********************************************************************************
* Descr:   Calls various INA design aids.                                      *
*          Displays the diagonalisation menu.                                  *
*                                                                              *
* Call.by: INADSN                                                              *
* Calls:   GETCHC,PLTINA,ROWOPE,SWOP,RCMLT,UNDOCH,PRTNYQ                       *
********************************************************************************

        subroutine diagon

$INCLUDE:'MENU.BLK'

1       menunb=4
        call getchc(menunb,escape)
        if (escape) goto 2
        if (chcdfl(menunb).eq.0) then
          call nyqstp
        elseif (chcdfl(menunb).eq.1) then
          call pltina
        elseif (chcdfl(menunb).eq.2) then
          call rowope
        elseif (chcdfl(menunb).eq.3) then
          call swop
        elseif (chcdfl(menunb).eq.4) then
```

```
          call rcmlt
        elseif (chcdfl(menunb).eq.5) then
          call undoch
        elseif (chcdfl(menunb).eq.6) then
          call prtnyq
        endif
        goto 1
2       return
        end


$include:'add.blk'
*******************************************************************************
* DIMTF  * FIND DIMENSIONS OF S IN TRANSFER FUNCTION                           *
*******************************************************************************
* Descr:   This procedure searches through the transfer function              *
*          coefficient array KN,KD to find the maximum power of s             *
*          used in the numerator and denominator elements.                    *
*                                                                             *
*        ! Automatically sets denominator to 1 if all coefficients equal 0    *
*                                                                             *
* Param:   ORDER  order of transfer function matrix                           *
*          MXSORD maximum allowed order for s.                                *
*          SORDER returns the maximum order of s used.                        *
*                                                                             *
* Call.by: CALCPW,INPUTTF                                                     *
* Calls:   none                                                               *
*******************************************************************************


        subroutine dimtf(order,mxsord,sorder,kn,kd)

        integer    order,mxsord,m,n,sorder
        integer    row,col,ij,ijk,power
        logical    first
        real       kn(*),kd(*)

        sorder=0
        do 2 row=1,order
          do 2 col=1,order
            ij=(col-1)*order+row
            ijk=(ij-1)*(mxsord+1)+1
            power=mxsord
4           if (kn(ijk+power).ne.0) goto 3
            power=power-1
            if (power.gt.sorder) goto 4
3           if (power.gt.sorder) sorder=power
            power=mxsord
6           if (kd(ijk+power).ne.0) goto 5
            power=power-1
            if (power.gt.0) goto 6
            if(kd(ijk).eq.0) kd(ijk)=1.
5           if (power.gt.sorder) sorder=power
2       continue
        return
        end


$include:'add.blk'
*******************************************************************************
* DISK   * FORM DISK FILE NAME                                                 *
```

```
*************************************************************************
* Descr:   Here the name for a file on disk is concastinated from       *
*          the directory DIR ,NAME and EXTension into CONCAT.           *
*                                                                       *
* Calls:   none                                                         *
*************************************************************************

        subroutine disk(concat,dir,name,ext)

        character*32 concat
        character*20 dir
        character*8  name
        character*4  ext
        integer      index,len

        index=1
2       if (index.eq.20) goto 3
        if (dir(index+1:index+1).eq.' ') goto 3
        index=index+1
        goto 2
3       len=1
4       if (len.eq.8) goto 5
        if (name(len+1:len+1).eq.' ') goto 5
        len=len+1
        goto 4
5       if (dir.eq.' ') then
          concat(1:len)=name(1:len)
          concat(len+1:len+4)=ext(1:4)
          concat(len+5:32)=' '
        else
          concat(1:index)=dir(1:index)
          concat(index+1:index+len)=name(1:len)
          concat(1+index+len:index+len+4)=ext(1:4)
          concat(index+len+5:32)=' '
        endif
        return
        end


$include:'add.blk'
*************************************************************************
* DRAWC  * DRAW COMPLEX VALUE                                           *
*************************************************************************
* Descr:    Plots a line between two given complex values at the corres-*
*           ponding axis.                                               *
*                                                                       *
*                                                                       *
* Param:    I,J index to axis.                                          *
*           QW(1) 1st complex point                                     *
*           QW(OFFSET) 2nd complex point                                *
*           ORDER INA matrix order                                      *
*           DIM dimension of axis                                       *
*                                                                       *
* Call.by:  PLOTPTS                                                     *
* Calls:    Hercules support DLINE.                                     *
*************************************************************************

          subroutine drawc(i,j,qw,offset,order,dim)

          integer    i,j,offset,order
          real       x,y,x2,y2,dim,xdiv,ydiv,xp,yp,xpl,ypl
```

```
          complex    qw(*)

          x=real(qw(1))
          y=aimag(qw(1))
          x2=real(qw(1+offset))
          y2=aimag(qw(1+offset))
          if ((abs(x).le.dim).and.(abs(y).le.dim)
     +       .and.(abs(x2).le.dim).and.(abs(y2).le.dim)) then
                xdiv=521./order/2.*(2*j-1)
                ydiv=347./order/2.*(2*i-1)
                xp1=xdiv+521.*x/order/2./dim
                yp1=ydiv-347.*y/order/2./dim
                call putpt(nint(xp1),nint(yp1))
                xp=xdiv+521.*x2/order/2./dim
                yp=ydiv-347.*y2/order/2./dim
                call dline(nint(xp),nint(yp))
c                if ((abs(nint(xp)-nint(xp1)).ge.12).or.
c     +              (abs(nint(yp)-nint(yp1)).ge.8))
c     +            call mark(nint(xp),nint(yp))
          endif
          return
          end

c Mark the position

          subroutine mark(i,j)
          integer i,j
          call putpt(i-3,j)
          call dline(i+3,j)
          call putpt(i,j-2)
          call dline(i,j+2)
          return
          end


$include:'add.blk'
*******************************************************************************
* DSNRST * DESIGN PARAMETERS RESET                                            *
*******************************************************************************
* Descr:   Sets the design enviroment to default.                             *
*                                                                             *
* Call.by: INACAD                                                             *
* Calls:   none                                                               *
*******************************************************************************

        subroutine dsnrst
        integer iocheck,i,runid

$INCLUDE:'DESIGN.BLK'
$INCLUDE:'DESdta.BLK'

c Copy protection scheme

        open(1,file='\ina.cad',iostat=iocheck)
        read(1,22,err=1)runid
22      format(i3)
        runid=runid+1
        rewind 1
        if (runid.gt.30000) runid=0
        write(1,22,err=1)runid
        close(1)
```

```
        goto 2
1       write(*,*)
        write(*,*)
        write(*,*) 'Unauthorised Duplicate.'
        stop

2       noofw=51
        call calcwf(3,3)
        order=1
        mxsord=20
        qexist=.false.
        wexist=.false.
        gexist=.false.
        kexist=.false.
        lexist=.false.
        fexist=.false.
        gfilen='        '
        kfilen='        '
        lfilen='        '
        ffilen='        '
        wfilen='DEFAULT '
        dsnnme=' '
        gerint=0
        ostint=0
        qexist=.false.
        qinvrt=.true.
        single=.false.
        inadim=0.
        multdim=0.
        call gmode
        call clrscr
        call tmode
        return
        end


$include:'add.blk'
*******************************************************************************
*EDITMLT * EDIT MULTIPLYING FACTOR                                            *
*******************************************************************************
* Descr:  The multiplication factor used for any row operation is             *
*         displayed and can be edited here.                                   *
*                                                                             *
* Param:  ORDER - Order of INA design.                                        *
*         SORDER - Highest power of s used.                                   *
*         KD - Denominator coefficients.                                      *
*         KN - Numerator coefficients.                                        *
*         KL - Lag                                                            *
*         MXSORD - Maximum order of s allowed                                 *
*         CHANGE - Set TRUE if multiplier is altered                          *
*                                                                             *
* Call.by: ROWOP,RCMLT                                                        *
* Calls:   ansisup,COMM2                                                      *
*******************************************************************************

          subroutine editmlt(order,sorder,kd,kn,kl,mxsord,change)

        character*20 input
        logical      escape,change
        integer      sorder,order,i,it,jt,j,k,ij,kij,mxsord
        real         kd(*),kn(*),kl
```

```
        real          num

        integer       getch,key
        external      getch

$include:'rowop.blk'

        i=1
        j=1
2       call cls
        call invers
        call gotorc(1,20)
        write(*,'(a\)')' Editing Multiplication Factor '
        call gotorc(1,62)
        call invers
        if (ruse.ne.0) then
          write(*,'(a\)')' Row  used: '
          write(*,'(i2.2\)')ruse
        else
          write(*,'(a\)')' Column used:'
          write(*,'(i2.2\)')cuse
        endif
        call gotorc(3,1)
        write(*,'(2a,2a1\)')
     +' Order of s   Numerator          Denominator          L',
     +'ag     ',13,10
        do 5 k=0,sorder
          write(*,'(i7,a,a1,a1\)')k,'     ',13,10
5       continue
        call comm2
4       call invers
        call gotorc(1,1)
        call normal
        write(*,'(a,i2,a,i2,3a1\)')'Reduce: (',ri,',',rj,')',13,10
        ij=(j-1)*order+i
        kij=(ij-1)*(mxsord+1)
        do 25 k=1,sorder+1
          call gotorc(3+k,13)
          write(*,111)kn(kij+k)
          write(*,112)kd(kij+k)
          if (k.eq.1) write(*,113)kl
25      continue
22      call normal
        call gotorc(7,62+8)
        key=lowcase(getch())
        if ((key.ne.13).and.(key.ne.27).and.(key.ne.110).and.(key.ne.100
     +).and.(key.ne.108).and.(key.ne.112)) goto 22
        if (key.eq.13)goto 7
        if (key.eq.27)goto 7
        if (key.eq.110) then
          k=0
10        k=k+1
            call invers
            call gotorc(3+k,13)
            write(*,111)kn(kij+k)
            call gotorc(3+k,13)
            call readln(input,14,escape)
            call normal
            if (escape) goto 11
            if (input.eq.'           ') goto 8
            read(input,110,err=9) num
```

```
            kn(kij+k)=num
            change=.true.
8           call gotorc(3+k,13)
            write(*,111)kn(kij+k)
            goto 16
9           write(*,'(a1\)')7
            call gotorc(3+k,13)
            write(*,111)kn(kij+k)
            k=k-1
16        if (k.lt.sorder+1) goto 10
          key=100
11        call gotorc(3+k,13)
          write(*,111)kn(kij+k)
        endif
        if (key.eq.100) then
          k=0
12        k=k+1
            call invers
            call gotorc(3+k,13+18)
            write(*,111)kd(kij+k)
            call gotorc(3+k,13+18)
            call readln(input,14,escape)
            call normal
            if (escape) goto 14
            if (input.eq.'           ') goto 13
            read(input,110,err=15) num
            kd(kij+k)=num
            change=.true.
13          call gotorc(3+k,13+18)
            write(*,111)kd(kij+k)
            goto 17
15          write(*,'(a1\)')7
            call gotorc(3+k,13+18)
            write(*,111)kd(kij+k)
            k=k-1
17        if (k.lt.sorder+1) goto 12
          key=108
14        call gotorc(3+k,13+18)
          write(*,111)kd(kij+k)
        endif
        if (key.eq.108) then
18          call invers
            call gotorc(4,13+18+16)
            write(*,111)k1
            call gotorc(4,13+18+16)
            call readln(input,14,escape)
            call normal
            if (escape) goto 19
            if (input.eq.'           ') goto 21
            read(input,110,err=20) num
            k1=num
            change=.true.
21          call gotorc(4,13+18+16)
            write(*,111)k1
            goto 19
20          write(*,'(a1\)')7
            call gotorc(4,13+18+16)
            write(*,111)k1
          goto 18
19        call gotorc(4,13+18+16)
          write(*,111)k1
```

```
        endif
        if (key.eq.112) then
1         call gotorc(25,1)
          write(*,'(15x,a,i1,a\)')' Change Power to (0-',mxsord,') : '
          call readln(input,2,escape)
          call normal
          call gotorc(25,1)
          call clreol
          if (.not.(escape.or.(input.eq.'          '))) then
            read(input,100,err=1) neworder
            if ((neworder.ge.0).and.(neworder.le.mxsord)) then
              sorder=neworder
            endif
          endif
        endif
        if (key.eq.112) goto 2
        goto 22
7       continue
        do 77 i=sorder+2,mxsord+1
          kd(i)=0
77        kn(i)=0
        return
100     format(bn,i2)
110     format(bn,e13.6)
111     format(1p,e14.7,\)
112     format(1p,e18.7,\)
113     format(1p,e16.7,\)
130     format(bn,2i2)
        end


$include:'add.blk'
*******************************************************************************
* EDITTF * EDIT TRANSFER FUNCTION MATRIX                                      *
*******************************************************************************
* Descr:   Allows the user to manipulate the process matrixes.                *
*                                                                             *
* Param:   DIR,NAME,EXT name of file containing data.                         *
*          EXIST logical set to true if this file has been altered.           *
*          ARRndl arrays containing coefficients                              *
*                                                                             *
* Call.by: PRCPAR                                                            *
* Calls:   ansisup,CALCTF,SAVCTF,LOADTF,DIMTF,INPUTTF,PRINTTF,GETCHC,RESETTF *
*******************************************************************************

        subroutine edittf(name,ext,ext2,arrn,arrd,arrl,exist)

        character*4  ext,ext2
        character*8  name,oldname
        character*32 filen
        logical      exist,change,dexist,found
        real         arrn(*),arrd(*),arrl(*)
        integer      getch,sorder,key
        external     getch


$INCLUDE:'DESDTA.BLK'
$INCLUDE:'DESIGN.BLK'
$INCLUDE:'MENU.BLK'

11      menunb=5
```

```
      call getchc(menunb,escape)
      if (escape) goto 111
      if (chcdfl(menunb).eq.0) then
        call getname(pfdir,name,ext,.true.,exist,escape)
        if (.not.escape) then
          call loadtf(pfdir,name,ext,arrn,arrd,arrl,
     +                order,mxsord,exist)
          if (name.ne.' ') then
            call calctf(w,noofw,arrn,arrd,arrl,mxsord,order,
     +                  name,tfdir,ext2,qexist)
          endif
        endif
      elseif (chcdfl(menunb).eq.1) then
        if (name.eq.' ') then
          call getname(pfdir,name,ext,.false.,exist,escape)
        endif
        if (.not.escape) then
          call dimtf(order,mxsord,sorder,arrn,arrd)
          if (sorder.eq.0) sorder=2
          call inputtf(name,ext,order,sorder,arrd,arrn,arrl,
     +                 (ext.eq.'.FS '),change,mxsord)
          if (change) then
10          call clrbot
            write(*,'(a\)')'    Press RETURN to calculate ...'
            if (getch().ne.13) then
              call inputtf(name,ext,order,sorder,arrd,arrn,arrl,
     +                     (ext.eq.'.FS '),change,mxsord)
              goto 10
            endif
            exist=.true.
            call calctf(w,noofw,arrn,arrd,arrl,mxsord,order,
     +                  name,tfdir,ext2,qexist)
          endif
        endif
      elseif (chcdfl(menunb).eq.2) then
        oldname=name
        call getname(pfdir,name,ext,.false.,exist,escape)
        if (.not.escape) then
          call savetf(pfdir,name,ext,arrn,arrd,arrl,
     +                order,mxsord,exist)
          if (oldname.ne.name) then
            call calctf(w,noofw,arrn,arrd,arrl,mxsord,order,
     +                  name,tfdir,ext2,qexist)
          endif
        endif
      elseif (chcdfl(menunb).eq.3) then
        if (name.ne.' ') then
          call printtf(pfdir,name,ext,arrn,arrd,arrl,order,mxsord)
        else
          call error('No matrix specified!          ')
        endif
      elseif (chcdfl(menunb).eq.4) then
        if (exist) then
          call clrbot
          write(*,'(4a\)')'       Warning: Old ',name,ext,
     +                  ' will be LOST!   Press <ESC> to abort...'
          if (getch().ne.27) exist=.false.
        endif
        if (.not.exist) then
          do 1 i=1,1000
            if (i.le.100) arrl(i)=0.
```

```
              arrn(i)=0.
              arrd(i)=0.
1           continue
            name=' '
            exist=.false.
          endif
        endif
        goto 11
111     return
        end


$include:'add.blk'
*******************************************************************************
* EDITWF *  EDIT FREQUENCY TABLE                                               *
*******************************************************************************
* Descr: The operator can specify the type of frequencies, the range          *
*        and the number of frequency points.                                  *
*                                                                             *
* Param: W - Frequency table                                                  *
*        NOOFW - Number of frequency points                                   *
*        WEXIST - Set TRUE if it is altered.                                  *
*        FILEN - Filename of frequency data                                   *
*                                                                             *
* Calls: REDOWF,CALCWF.                                                       *
*******************************************************************************

        subroutine editwf(w,noofw,wexist,filen)
        logical      wexist,change,escape
        integer      noofw,type
        real         w(*)
        integer      stdec,enddec,newnb,oldnfw,getch,key
        character*20 input
        character*8  filen
        external     getch

        type=1
        call normal
        call gotorc(4,25)
        write(*,'(2a\)')'Editing Frequency File : ',filen
        oldnfw=noofw

c       use W(2) rather than W(1) because W(1) may be equal to zero

        if (w(2).eq.0) then
          stdec=-9
        else
          stdec=nint(log10(w(2)))
        endif
        if (w(noofw).eq.0) then
          enddec=-9
        else
          enddec=nint(log10(w(noofw)))
        endif
        call gotorc(10,20)
        write(*,'(a\)')'Start Frequency   : 10^'
        write(*,'(i3,a\)')stdec,' Rad/s                    '
        call gotorc(12,20)
        write(*,'(a\)')'Final Frequency   : 10^'
        write(*,'(i3,a\)')enddec,' Rad/s'
        call gotorc(14,20)
```

```
        write(*,'(a\)')'Number of Samples :  '
        write(*,'(i3\)')noofw
        change=.false.
1       call gotorc(25,20)
        call invers
        write(*,'(a\)')' Press <ESC> to leave unchanged.'
        call normal
4       call gotorc(8,20)
        write(*,'(a\)')'Frequency Points  : '
        if (type.eq.1) call invers
        write(*,'(a\)')' SqrtLog '
        call normal
        if (type.eq.2) call invers
        write(*,'(a\)')' Log '
        call normal
        if (type.eq.3) call invers
        write(*,'(a\)')' Sqrt2Log '
        call normal
        if (type.eq.4) call invers
        write(*,'(a\)')' Ln '
        call normal
        if (type.eq.5) call invers
        write(*,'(a\)')' Special '
        call inverse
        call gotorc(9,22)
        write(*,'(a\)')' Press <SPACE BAR> to change selection. '
        key=getch()
        call normal
        call gotorc(9,20)
        call clreol
        if (key.eq.27) goto 999
        if (key.eq.32) then
          type=type+1
          if (type.eq.6) type=1
          change=.true.
          goto 4
        endif
2       call gotorc(10,20)
        write(*,'(a\)')'Start Frequency   : 10^'
        call saverc
        write(*,'(i3,a\)')stdec,' Rad/s'
        call invers
        call gotorc(11,22)
        write(*,'(a\)')' Enter new decade. '
        call normal
        call loadrc
        call readln(input,3,escape)
        call gotorc(11,20)
        call clreol
        if (escape) goto 999
        if (input.eq.' ') goto 3
        read(input,100,err=92) newnb
        if (newnb.ne.stdec) then
          if (newnb.lt.enddec) then
            stdec=newnb
            change=.true.
          endif
        endif
        goto 2
3       call gotorc(12,20)
        write(*,'(a\)')'Final Frequency   : 10^'
```

```
        call saverc
        write(*,'(i3,a\)')enddec,' Rad/s'
        call invers
        call gotorc(13,22)
        write(*,'(a\)')' Enter new decade. '
        call normal
        call loadrc
        call readln(input,3,escape)
        call gotorc(13,20)
        call clreol
        if (escape) goto 999
        if (input.eq.' ') goto 5
        read(input,100,err=94) newnb
        if (newnb.ne.enddec) then
          if (stdec.lt.newnb) then
            enddec=newnb
            change=.true.
          endif
        endif
        goto 3
5       call gotorc(14,20)
        write(*,'(a\)')'Number of Samples :  '
        call saverc
        write(*,'(i3\)')noofw
        call invers
        call gotorc(15,22)
        write(*,'(a\)')' Enter number from 2 to 200. '
        call normal
        call loadrc
        call readln(input,3,escape)
        call gotorc(15,20)
        call clreol
        if (escape) goto 999
        if (input.eq.' ') goto 7
        read(input,101,err=96) newnb
        if (newnb.ne.noofw) then
          if ((newnb.gt.1).and.(newnb.le.200)) then
            noofw=newnb
            change=.true.
          endif
        endif
        goto 5
7       if (.not.change) goto 1
        call invers
        call gotorc(17,20)
        write(*,'(a\)')' Press ''Y'' to accept change...'
        call normal
        if (getch().ne.121) then
          call gotorc(17,20)
          call clreol
          goto 1
        endif
        if (type.eq.2) then
          call logwf(-stdec,enddec)
        elseif (type.eq.3) then
          call calc2wf(-stdec,enddec)
        elseif (type.eq.4) then
          call lnwf(-stdec,enddec)
        elseif (type.eq.1) then
          call calcwf(-stdec,enddec)
        else
```

```
          call specwf(stdec,enddec)
        endif
        call redowf
        wexist=.true.
        goto 11


92      write(*,'(a1\)')7
        goto 2
94      write(*,'(a1\)')7
        goto 3
96      write(*,'(a1\)')7
        goto 5

999     escape=.true.
        noofw=oldnfw
11      return
100     format(bn,i3)
101     format(bn,i3)
        end


$include:'add.blk'
*******************************************************************************
* ERROR  * DISPLAY ERROR MESSAGE                                              *
*******************************************************************************
* Descr:   Displays an error message on the bottom line and waits             *
*          for a key to be pressed.                                           *
*                                                                             *
*                                                                             *
* Param:   MESSAGE - Error message                                            *
*                                                                             *
* Calls:   ansisup,                                                           *
*******************************************************************************

          subroutine error(message)

          character*30 message
          integer      key,getch
          external     getch

          call clrbot
          write(*,'(a1,3a\)') 7,'       ERROR: ',message,
     +     ' Press any key to continue...'
          key=getch()
          call normal
          call gotorc(25,1)
          call clreol
          return
          end


$include:'add.blk'
*******************************************************************************
* FRANGE * SET FREQUENCY RANGE                                                *
*******************************************************************************
* Descr:   Display menu to load, save or edit the frequency file.             *
*                                                                             *
* Call.by: PRCPAR                                                             *
* Calls:   ansisup,GETCHC,LOADWF,REDOWF,GETNAME,SAVEWF,EDITWF.                *
```

```
*******************************************************************************

        subroutine frange
        character*20 oldwname
        integer      getch
        external     getch

$include:'menu.blk'
$include:'design.blk'
$include:'desdta.blk'

11      menunb=6
        call getchc(menunb,escape)
        if (escape) goto 111
        if (chcdfl(menunb).eq.0) then
          oldwname=wfilen
          call getname(pfdir,wfilen,'.WS ',.true.,wexist,escape)
          if (.not.escape) then
            call loadwf(pfdir,wfilen,'.WS ',w,noofw,wexist)
            if (wfilen.eq.' ') then
              wfilen=oldwname
            else
              call redowf
            endif
          endif
        elseif (chcdfl(menunb).eq.2) then
          call getname(pfdir,wfilen,'.WS ',.false.,wexist,escape)
          if (.not.escape)
     +      call savewf(pfdir,wfilen,'.WS ',w,noofw,wexist)
        elseif (chcdfl(menunb).eq.1) then
          call editwf(w,noofw,wexist,wfilen)
        endif
        goto 11
111     return
        end


$include:'add.blk'
*******************************************************************************
* GETCHC * GET MENU CHOICE                                                    *
*******************************************************************************
* Descr:   This routine displays the menu as specified by MENUNB and          *
*          returns the selected item in CHCDFL(MENUNB). If escape is          *
*          pressed instead the logical ESCAPE becomes true.                   *
*                                                                             *
* Calls:   ansisup                                                            *
*******************************************************************************

        subroutine getchc(mnunb,escape)

        integer    i,ch,getch,readch
        integer    msgrow,mnunb
        logical    done
        external   getch,readch

$INCLUDE:'MENU.BLK'

        menunb=mnunb
        done=.false.
        escape=.false.
        ch=32
```

```
      if(oldmnu(depth).ne.menunb)depth=depth+1
      call cls
      if (menunb.eq.2) then
        call gotorc(1,1)
      write(*,'(3a\)')'        INACAD Multi-Variable Control System',
     +' Design Aid',
     +' (c)RHE Venzke 1986,87,88'
      else
        call invers
        call gotorc(top(1),top(2)+4)
        write(*,'(a,\)')choice(mnuinf(1,oldmnu(depth-1))
     +          +chcdfl(oldmnu(depth-1)))
      endif
      call gotorc(1,1)
      call invers
      write(*,'(A,\)')' Menu '
      do 3 i=2,depth
        write(*,'(I1,A,\)')chcdfl(oldmnu(i-1))+1,'.'
3     continue
      write(*,'(A1,A4,\)')27,'[1D '
      call invers
      call gotorc(2+top(1),top(2)-9)
      write(*,'(a,\)')' Select:'
      call normal
      do 1 i=mnuinf(1,menunb),mnuinf(2,menunb)-1
        call gotorc(4+top(1)+2*(i-mnuinf(1,menunb)),top(2)-5)
        write(*,'(A,I1,A,A,\)')'f',1+i-mnuinf(1,menunb)
     +,'   ',choice(i)
1     continue
      call gotorc(4+top(1)+2*(i-mnuinf(1,menunb)),top(2)-6)
      write(*,'(2a,\)')'Esc   ',choice(i)
      msgrow=6+top(1)+2*(i-mnuinf(1,menunb))
      call invers
      call gotorc(msgrow,top(2)-9)
      write(*,'(a,\)')' Choice: '
2     continue
      call invers
      call gotorc(4+top(1)+2*chcdfl(menunb),top(2))
      write(*,'(a,\)') choice(chcdfl(menunb)+mnuinf(1,menunb))
      call normal
      call gotorc(msgrow,top(2))
      write(*,'(a,\)') messge(chcdfl(menunb)+mnuinf(1,menunb))
      ch=getch()
      if (ch.eq.0) then
        ch=readch()
        if ((ch.eq.77).or.(ch.eq.80).or.(ch.eq.79).or.(ch.eq.81)) then
          call gotorc(3+top(1)+2*chcdfl(menunb),top(2))
          write(*,'(a,\)')'                    '
          call gotorc(4+top(1)+2*chcdfl(menunb),top(2))
          write(*,'(a,\)') choice(chcdfl(menunb)+mnuinf(1,menunb))
          chcdfl(menunb)=chcdfl(menunb)+1
          if ((ch.eq.79).or.(ch.eq.81))chcdfl(menunb)=mnuinf(2,menunb)
     +-mnuinf(1,menunb)
          if (chcdfl(menunb).gt.(mnuinf(2,menunb)
     +-mnuinf(1,menunb))) then
            chcdfl(menunb)=0
          endif
       elseif ((ch.eq.75).or.(ch.eq.72).or.(ch.eq.71).or.(ch.eq.73)) then
          call gotorc(3+top(1)+2*chcdfl(menunb),top(2))
          write(*,'(a,\)')'                    '
          call gotorc(4+top(1)+2*chcdfl(menunb),top(2))
```

```
            write(*,'(a,\)') choice(chcdfl(menunb)+mnuinf(1,menunb))
            chcdfl(menunb)=chcdfl(menunb)-1
            if ((ch.eq.71).or.(ch.eq.73)) chcdfl(menunb)=0
            if (chcdfl(menunb).lt.0) then
              chcdfl(menunb)=mnuinf(2,menunb)-mnuinf(1,menunb)
            endif
          elseif ((ch.ge.59).and.((ch-59).lt.
     +            (mnuinf(2,menunb)-mnuinf(1,menunb)))) then
            chcdfl(menunb)=ch-59
            done=.true.
          endif
        elseif (ch.eq.13) then
          done=.true.
        elseif (ch.eq.27) then
          done=.true.
          escape=.true.
        endif
        if (.not.done) goto 2
        if (escape.or.(chcdfl(menunb).eq.(mnuinf(2,menunb)-
     +                              mnuinf(1,menunb)))) then
          depth=depth-1
          escape=.true.
        else
          oldmnu(depth)=menunb
          call cls
          call invers
          call gotorc(top(1),top(2)+4)
          write(*,'(a,\)')choice(mnuinf(1,oldmnu(depth))
     +          +chcdfl(oldmnu(depth)))
          call gotorc(1,1)
          call invers
          write(*,'(A,\)')' Menu '
          do 4 i=2,depth+1
            write(*,'(I1,A,\)')chcdfl(oldmnu(i-1))+1,'.'
4         continue
          write(*,'(A1,A4,\)')27,'[1D '
          call normal
        endif
        return
        end

$include:'add.blk'
*******************************************************************************
* GETNAME* GET FILENAME FROM USER                                             *
*******************************************************************************
* Descr:   Here the user is prompted for a filename. This routine is used     *
*          both for saving and loading. It checks whether the file exists.    *
*                                                                             *
*                                                                             *
* Param:   DIR,NAME,EXT Complete filename.                                    *
*          LOAD Logical false if used for saving else for loading.            *
*          NAME = ' ' if used for loading but file not found.                 *
*          ESCAPE logical true if escape was pressed.                         *
*          EXIST  logical true if old file needs to be saved.                 *
*                                                                             *
* Call.by: EDITTF,LOADTF,SAVTF                                                *
* Calls:   ansisup,SYSCAL                                                     *
*******************************************************************************

        subroutine getname(dir,name,ext,load,exist,escape)
```

```
      character*32 fname
      character*20 dir
      character*8  name,newname,oldname,tname
      character*4  ext
      character*40[c] command
      logical      load,escape,found,exist
      integer      i,getch,key
      external     getch

      oldname=name
1     call normal
      call gotorc(4,20)
      write(*,'(a\)')'File directory= '
      if (dir.eq.' ') then
        write(*,'(a\)')'default'
      else
        write(*,'(a\)')dir
      endif
      call gotorc(6,20)
      call clreol
      call gotorc(6,20)
      write(*,'(a\)')'Enter filename: '
      call saverc
      write(*,'(4a\)')name,'[',ext,']                '
      if (exist.and.load) then
        call clrbot
        write(*,'(10x,4a\)')'WARNING: Altered ',name,ext,
     +' will be LOST!    Press <ESC> to abort.'
        call normal
      endif
      call inverse
      call gotorc(8,20)
      write(*,'(a\)')' Press <Enter> for default filename.'
      call normal
      call loadrc
      call readln(newname,8,escape)
      if (escape) name=oldname
      if (escape) goto 100
      if (newname.ne.' ') name=newname
      call disk(fname,dir,name,ext)
      inquire(file=fname,err=99,exist=found)
      call gotorc(6,51)
      call inverse
      if (found) then
        write(*,'(a\)')' FOUND '
        if ((.not.load).and.(oldname.ne.name)) then
          call clrbot
          write(*,'(2a\)')'        WARNING: Old file will be lost! ',
     +     '   Press <ESC> to abort...'
          if (getch().eq.27) then
            call normal
            call gotorc(25,1)
            call clreol
            goto 1
          endif
        endif
      elseif (.not.load) then
        write(*,'(a\)')' NEW FILE '
        do 2 i=1,2000
2       continue
```

```
        else
          write(*,'(a,a1\)')' NOT FOUND ',7
          call normal
          write(tname,'(2a)')'*',ext
          call disk(fname,dir,tname,' /w ')
          call gotorc(10,1)
          write(command,'(2a)')'dir ',fname
          call syscal(command)
          write(*,'(2a1\)')13,10
          write(*,'(a\)')'Press any key to continue...'
          key=getch()
          call cls
        endif
        if (.not.found.and.load) goto 1
        if (name.eq.' ') goto 1
        goto 100

99      call error('Disk fault!                    ')
        goto 1

100     return
        end


$include:'add.blk'
*******************************************************************************
* GETORD * GET MATRIX ORDER                                                    *
*******************************************************************************
* Descr:   Sets the order of the transfer matrices.                            *
*          Calculates MXSORD which contains the maximum order of s that        *
*          can be stored in the available data space.                          *
*                                                                              *
* Call.by: PRCPAR                                                             *
* Calls:   READLN,DSNRST,ansisup                                               *
*******************************************************************************

        subroutine getord

        character*20 input
        logical      escape
        integer      num

$INCLUDE:'DESIGN.BLK'

1       call clrdsp
        call gotorc(4,10)
        write(*,'(A\)')'Select transfer matrices order (NxN)'
        call gotorc(5,11)
        write(*,'(a\)')'where N ranges from 1 to 10.'
        call gotorc(7,10)
        write(*,'(a\)')'Enter order N: '
        call saverc
        write(*,'(i2.2\)')order
        call gotorc(9,10)
        write(*,'(a,i2\)')'Maximum order of s of polynomials= ',mxsord
        call loadrc
        call readln(input,2,escape)
        if (escape) goto 10
        if (input.eq.' ') goto 10
        read(input,100,err=11)num
        if ((num.gt.10).or.(num.le.0)) goto 11
```

```
        order=num
        mxsord=1000/order/order-1.5
        if (mxsord.gt.20) mxsord=20
        goto 1
11      write(*,'(a1\)') 7
        goto 1
10      return
100     format(bn,i2)
        end


$include:'add.blk'
********************************************************************************
* GETQU2 * GET QUOTIENT                                                        *
********************************************************************************
* Descr:   Get quotient from INA diagram.                                      *
*          Used by row/ column multiplication routine.                         *
*                                                                              *
*                                                                              *
* Param:   none                                                                *
*                                                                              *
* Call.by: ROWMLT                                                              *
* Calls:   ansisup.                                                            *
********************************************************************************

          subroutine getqu2

          character*32 filen
          integer      iocheck,ci,cj,wi,row,col
          complex      kw
          integer      getch,key
          external     getch

$include:'rowop.blk'
$include:'design.blk'
$include:'desdta.blk'

          if (.not.qexist) then
            call error('No Nyquist Diagram exists!    ')
            do 3 i=1,noofw
3             quotient(i)=(0,0)
            goto 10
          endif
          call wait
          if (ruse.ne.0) then
            ci=ruse
            cj=rj
          else
            ci=ri
            cj=cuse
          endif
          call disk(filen,tfdir,'nyquist ','.~jw')
2         open(1,file=filen,form='binary',iostat=iocheck)
          do 1 wi=1,noofw
            do 1 row=1,order
              do 1 col=1,order
                read(1,err=92)kw
                if ((row.eq.ci).or.(col.eq.cj))
     +             quotient(wi)=1/kw
1         continue
          close(1)
```

```
          goto 10
92        call error('INA diagram file corrupted!    ')
          close(1)
10        return
          end


$include:'add.blk'
*******************************************************************************
* GETQUOT* GET QUOTIENT                                                       *
*******************************************************************************
* Descr:   Gets the specified quotient from the INA diagram to be used        *
*          by row or column operations.                                       *
*                                                                             *
* Param:   Given by ROWOP.BLK                                                 *
*                                                                             *
* Call.by: ROWOP                                                              *
* Calls:   ansisup.                                                           *
*******************************************************************************

          subroutine getquot

          character*32 filen
          integer      iocheck,ci,cj,wi,row,col
          complex      kw
          integer      getch,key
          external     getch

$include:'rowop.blk'
$include:'design.blk'
$include:'desdta.blk'

          if (.not.qexist) then
            call error('No Nyquist Diagram exists!    ')
            do 3 i=1,noofw
3             quotient(i)=(0,0)
            goto 10
          endif
          call wait
          if (ruse.ne.0) then
            ci=ruse
            cj=rj
          else
            ci=ri
            cj=cuse
          endif
          call disk(filen,tfdir,'nyquist ','.~jw')
2         open(1,file=filen,form='binary',iostat=iocheck)
          do 1 wi=1,noofw
            quotient(wi)=(-1,0)
            do 1 row=1,order
              do 1 col=1,order
                read(1,err=92)kw
                if ((row.eq.ci).and.(col.eq.cj)) then
                  if (cabs(kw).gt.1.e-38) then
                    quotient(wi)=quotient(wi)/kw
                  else
                    call clrbot
        write(*,'(8x,a,1p,e13.7\)')'WARNING: Division by zero at w= ',
     +w(wi)
                    quotient(wi)=(0,0)
```

```
                endif
              endif
              if ((row.eq.ri).and.(col.eq.rj))
     +           quotient(wi)=quotient(wi)*kw
1         continue
          close(1)
          goto 10
92        call error('INA diagram file corrupted!    ')
          close(1)
10        return
          end


$include:'add.blk'
*******************************************************************************
*GETSCALE* GET SCALE                                                          *
*******************************************************************************
* Descr:   The operator can enter scales to be used for the multiplier        *
*          diagram.                                                           *
*                                                                             *
* Param:   Set by DIM.BLK                                                     *
*                                                                             *
* Call.by: ROWOP,ROWMLT                                                       *
* Calls:   ansisup,                                                           *
*******************************************************************************

          subroutine getscale
          character*20 input
          logical      escape

$include:'dim.blk'

          call gotorc(4,20)
          write(*,'(a\)')'Multiplier Diagram Options'
          call gotorc(6,15)
         write(*,'(a\)')'Select real values for min. & maximum scale,'
          call gotorc(7,15)
          write(*,'(a\)')'or enter zero for automatic scaling.'
          call gotorc(24,20)
          call invers
          write(*,'(a\)')' Press <ESC> to exit options. '
          call normal
          call gotorc(11,20)
          write(*,'(a\)')' Y-Axis Range: '
          write(*,'(1p,e10.3,a,1p,e10.3,\)') ylo,'>',yhi
1         call gotorc(9,20)
          write(*,'(a\)')' X-Axis Range: '
          write(*,'(1p,e10.3,a,1p,e10.3,\)') xlo,'>',xhi
          call gotorc(9,35)
          call readln(input,10,escape)
          if (escape) goto 100
          if (input.ne.' ') then
            read(input,333,err=1) xlo
            goto 1
          endif
11        call gotorc(9,46)
          write(*,'(1p,e10.3\)') xhi
          call gotorc(9,46)
          call readln(input,10,escape)
          if (escape) goto 100
          if (input.ne.' ') then
```

```
            read(input,333,err=11) xhi
            goto 11
          endif
2         call gotorc(11,20)
          write(*,'(a\)')' Y-Axis Range: '
          write(*,'(1p,e10.3,a,1p,e10.3,\)') ylo,'>',yhi
          call gotorc(11,35)
          call readln(input,10,escape)
          if (.not.escape.and.input.ne.' ') then
            read(input,333,err=2) ylo
            goto 2
          endif
22        call gotorc(11,46)
          write(*,'(1p,e10.3\)') yhi
          call gotorc(11,46)
          call readln(input,10,escape)
          if (escape) goto 100
          if (input.ne.' ') then
            read(input,333,err=22) yhi
            goto 22
          endif
          goto 1
333       format(e10.3)
100       return
          end


$include:'add.blk'
*******************************************************************************
* HELP   * ONLINE HELP FASCILITY                                               *
*******************************************************************************
* Descr:   The file INACAD.HLP is displayed page by page on the screen.       *
*                                                                              *
* Call.by: INADSN                                                             *
* Calls:   ansisup.                                                           *
*******************************************************************************

          subroutine help

          logical      dummy
          character*20 filen
          character    ch
          logical      kbhit
          integer      getch,iocheck
          external     getch,kbhit

          call gotorc(7,1)
          filen='inacad.hlp'
          open(1,file=filen,form='binary',iostat=iocheck)
          if (iocheck.ne.0) then
            close(1)
            call error('Help file not found!          ')
            goto 99
          endif
1         read(1,err=99)ch
          if (kbhit()) then
            if (getch().eq.27) goto 99
          endif
          if (ch.eq.'{') then
            call invers
          elseif (ch.eq.'}') then
```

```
            call normal
          elseif (ch.eq.'|') then
            if (getch().eq.27) goto 99
          elseif (ch.ne.10) then
            write(*,12,err=99)ch
          endif
          goto 1
12        format(a1\)
99        close(1)
100       return
          end


$include:'add.blk'
*******************************************************************************
* INADSN * PERFORM INA DESIGN                                                 *
*******************************************************************************
* Descr:   Displays the top level menu and allows the user to jump to        *
*            a specific part of the design.                                   *
*                                                                             *
* Call.by: INACAD                                                            *
* Calls:   ansisup,GETCHC,DSNPAR,PRCPAR,NYQPAR,DIAGON                        *
*******************************************************************************

          subroutine inadsn

          integer   key,getch
          external  getch
          logical   dsnclr
          external  dsnclr

$INCLUDE:'MENU.BLK'
$include:'design.blk'

11        menunb=2
          call getchc(menunb,escape)
          if (escape) goto 10
          if (chcdfl(menunb).eq.5) then
            call shwpar(.false.)
            call clrbot
            write(*,'(8x,2a\)')'WARNING: Old design parameters will ',
     +        'be lost!   Press <ESC> to abort...'
            if (getch().ne.27) then
              call gotorc(25,1)
              call clreol
              write(*,'(20x,a\)')'Clearing Design...'
              call clrdsn
            endif
            goto 11
          elseif (chcdfl(menunb).eq.0) then
            call dos
          elseif (chcdfl(menunb).eq.1) then
            call help
          elseif (chcdfl(menunb).eq.2) then
            call shwpar(.true.)
          elseif (chcdfl(menunb).eq.3) then
            call prcpar
          elseif (chcdfl(menunb).eq.4) then
            call diagon
          endif
          goto 11
```

```
10          if (gexist.or.kexist.or.lexist.or.fexist.or.wexist) then
              call gotorc(24,21)
             write(*,'(a1,a\)')7,'Changes were made ! Quit anyway (y/n)?'
              if (getch().eq.121) goto 111
              depth=depth+1
              goto 11
            else
              call gotorc(24,21)
             write(*,'(a1,a\)')7,'Quit INACAD (y/n)?'
              if (getch().eq.121) goto 111
              depth=depth+1
              goto 11
            endif
111         return
            end

            logical function dsnclr()
            logical result

$include:'design.blk'

            if ((gfilen.eq.' ').and.(kfilen.eq.' ').and.
     +      (lfilen.eq.' ').and.(ffilen.eq.' ')) then
              result=.true.
            else
              result=.false.
            endif
            dsnclr=result
            return
            end

            subroutine clrdsn()
            integer i

$include:'design.blk'
$include:'rowop.blk'
$include:'desdta.blk'

            dsnnme=' '
            gfilen=' '
            gexist=.false.
            kfilen=' '
            kexist=.false.
            lfilen=' '
            lexist=.false.
            ffilen=' '
            fexist=.false.
            qexist=.false.
            ri=1
            rj=1
            ruse=1
            cuse=0
            rsord=1
            xdim=0
            ydim=0
            eps=1.e-8
            wint=10
            do 1 i=1,21
              rkd(i)=0.0
1             rkn(i)=0.0
            rkd(1)=1.
```

```
          rkl=0.
          stp=10.
          do 3 i=1,1000
            gnum(i)=0.
            knum(i)=0.
            lnum(i)=0.
            fnum(i)=0.
            gden(i)=0.
            kden(i)=0.
            lden(i)=0.
3           fden(i)=0.
          do 4 i=1,100
            glag(i)=0.
            klag(i)=0.
            llag(i)=0.
4           flag(i)=0.
          ksaved=.false.
          lsaved=.false.
          return
          end


$include:'add.blk'
*******************************************************************************
* INPUTTF* INPUT TRANSFER FUNCTION                                            *
*******************************************************************************
* Descr:   The transfer function can be edited in this routine.               *
*                                                                             *
* Param:   FNAME filename of definition.                                      *
*          EXT   filename extesion                                            *
*          ORDER matrix order (nxn)                                           *
*          SORDER maximum order of s used.                                    *
*          MXSORD maximum allowed order of s.                                 *
*          KNT,KDT,KLT contains matrix specifications.                        *
*          DIAGON logical true if only diagonal entries are allowed i.e. F(s)*
*          CHANGE logical true if changes were made to the data.              *
*                                                                             *
* Call.by: EDITTF                                                             *
* Calls:   READLN,ansisup,COMMND,CLRCOM                                       *
*******************************************************************************

       subroutine inputtf(fname,ext,order,sorder,kd,kn,kl,
     +diagon,change,mxsord)

       character*4  ext
       character*8  fname
       character*20 input
       character*32 filen
       logical      escape,change,diagon
       integer      sorder,order,i,it,jt,j,k,ij,kij,mxsord
       real         kd(*),kl(*),kn(*)
       real         num

       integer      getch,key
       external     getch

       i=1
       j=1
       change=.false.
2      call cls
       call invers
```

```
        call gotorc(1,25)
        write(*,'(a,a,a\)')' Editing: ',fname,ext
        call gotorc(1,67)
        call invers
        write(*,'(a\)')' Mx Order:'
        write(*,'(i2.2\)')order
        call gotorc(3,1)
        write(*,'(2a,2a1\)')
     +' Order of s    Numerator         Denominator          ',
     +'Lag     ',13,10
        do 5 k=0,sorder
          write(*,'(i7,a,a1,a1\)')k,'     ',13,10
5       continue
        call commnd
4       call invers
        call gotorc(1,1)
        if (diagon) then
          write(*,'(a,\)')' Diag. Element:'
          i=j
        else
          write(*,'(a,\)')' Element: '
        endif
        call normal
        write(*,'(a,i2,a,i2,3a1\)')' (',i,',',j,')',13,10
        ij=(i-1)*order+j
        kij=(ij-1)*(mxsord+1)
        do 25 k=1,sorder+1
          call gotorc(3+k,13)
          write(*,111)kn(kij+k)
          write(*,112)kd(kij+k)
          if (k.eq.1) write(*,113)kl(ij)
25      continue
22      call normal
        call gotorc(25,1)
        call clreol
        call gotorc(7,62+8)
        key=getch()
        if ((key.ne.13).and.(key.ne.27).and.(key.ne.110).and.(key.ne.100
     +).and.(key.ne.108).and.(key.ne.112).and.(key.ne.106)) goto 22
        if (key.eq.13) then
          j=j+1
          if (j.gt.order) then
            j=1
            i=i+1
            if (i.gt.order) i=1
          endif
        endif
        if (key.eq.27)goto 7
        if (key.eq.110) then
          k=0
10        k=k+1
            call invers
            call gotorc(3+k,13)
            write(*,111)kn(kij+k)
            call gotorc(3+k,13)
            call readln(input,14,escape)
            call normal
            if (escape) goto 11
            if (input.eq.'           ') goto 8
            read(input,110,err=9) num
            kn(kij+k)=num
```

```
            change=.true.
8           call gotorc(3+k,13)
            write(*,111)kn(kij+k)
            goto 16
9           write(*,'(a1\)')7
            call gotorc(3+k,13)
            write(*,111)kn(kij+k)
            k=k-1
16        if (k.lt.sorder+1) goto 10
          key=100
11        call gotorc(3+k,13)
          write(*,111)kn(kij+k)
        endif
        if (key.eq.100) then
          k=0
12        k=k+1
            call invers
            call gotorc(3+k,13+18)
            write(*,111)kd(kij+k)
            call gotorc(3+k,13+18)
            call readln(input,14,escape)
            call normal
            if (escape) goto 14
            if (input.eq.'           ') goto 13
            read(input,110,err=15) num
            kd(kij+k)=num
            change=.true.
13          call gotorc(3+k,13+18)
            write(*,111)kd(kij+k)
            goto 17
15          write(*,'(a1\)')7
            call gotorc(3+k,13+18)
            write(*,111)kd(kij+k)
            k=k-1
17        if (k.lt.sorder+1) goto 12
          key=108
14        call gotorc(3+k,13+18)
          write(*,111)kd(kij+k)
        endif
        if (key.eq.108) then
18          call invers
            call gotorc(4,13+18+16)
            write(*,111)kl(ij)
            call gotorc(4,13+18+16)
            call readln(input,14,escape)
            call normal
            if (escape) goto 19
            if (input.eq.'           ') goto 21
            read(input,110,err=20) num
            kl(ij)=num
            change=.true.
21          call gotorc(4,13+18+16)
            write(*,111)kl(ij)
            goto 19
20          write(*,'(a1\)')7
            call gotorc(4,13+18+16)
            write(*,111)kl(ij)
          goto 18
19        call gotorc(4,13+18+16)
          write(*,111)kl(ij)
        endif
```

```
        if (key.eq.112) then
1         call clrbot
          write(*,'(15x,a,i2,a\)')' Change Power to (0-',mxsord,') : '
          call readln(input,2,escape)
          call clrbot
          if (.not.(escape.or.(input.eq.'          '))) then
            read(input,100,err=1) neworder
            if ((neworder.ge.0).and.(neworder.le.mxsord)) then
              sorder=neworder
            endif
          endif
        endif
        if (key.eq.106) then
23        call clrbot
          write(*,'(20x,a,\)')'JUMP to (row col):'
          call readln(input,5,escape)
          call normal
          if (escape.or.(input.eq.'          ')) goto 24
          call clrbot
          read(input,130,err=23)it,jt
          if((it.gt.order).or.(it.le.0).or.(jt.gt.order).or.(jt.le.0)
     +) goto 23
          i=it
          j=jt
24      endif
        if (key.eq.106) goto 4
        if (key.eq.13) goto 4
        if (key.eq.112) goto 2
        goto 22
7       continue
        return
100     format(bn,i2)
110     format(bn,e13.6)
111     format(1p,e14.7,\)
112     format(1p,e18.7,\)
113     format(1p,e16.7,\)
130     format(bn,2i2)
        end


$include:'add.blk'
*******************************************************************************
* KXCOL  * MULTIPLY K(s) * COLUMN OF MATRIX                                   *
* KXROW  * MULTIPLY K(s) * ROW OF MATRIX                                      *
*******************************************************************************
* Descr:   Multiplies the specified row or column o fa transfer matrix by a   *
*          desired function.                                                  *
*                                                                             *
*                                                                             *
* Param:  RKN,RKD,RKL - Function specifications                               *
*         KNUM,KDEN,KLAG - Matrix coefficients                                *
*         ORDER - Matrix order                                                *
*         MXSORD - Highest power of s used.                                   *
*         RUSE/CUSE - Row / column to be multiplied                           *
*                                                                             *
* Call.by: ROWOP                                                              *
* Calls:   MLTROW,MLTCOL                                                      *
*******************************************************************************
          subroutine kxrow(rkn,rkd,rkl,knum,
     +                     kden,klag,order,mxsord,ruse)
          real       rkn(*),rkd(*),rkl,knum(*)
```

```
      real       kden(*),klag(*),mltn(140),mltd(140),mltl(10)
      integer    i,j,order,mxsord,ruse

      call mltrow(mltn,mltd,mltl,rkn,rkd,rkl,knum,
     +              kden,klag,order,mxsord,ruse)
      do 1 i=1,order
        klag(i+order*(ruse-1))=mltl(i)
        do 1 j=1,mxsord+1
          knum(j+(mxsord+1)*((i-1)+order*(ruse-1)))=
     +          mltn(j+(mxsord+1)*(i-1))
          kden(j+(mxsord+1)*((i-1)+order*(ruse-1)))=
     +          mltd(j+(mxsord+1)*(i-1))
1     continue
      return
      end

      subroutine kxcol(rkn,rkd,rkl,lnum,
     +                 lden,llag,order,mxsord,cuse)
      real       rkn(*),rkd(*),rkl,lnum(*)
      real       lden(*),llag(*),mltn(140),mltd(140),mltl(10)
      integer    i,j,order,mxsord,cuse

      call mltcol(mltn,mltd,mltl,rkn,rkd,rkl,lnum,
     +              lden,llag,order,mxsord,cuse)
      do 1 i=1,order
        llag(cuse+order*(i-1))=mltl(i)
        do 1 j=1,mxsord+1
          lnum(j+(mxsord+1)*((cuse-1)+order*(i-1)))=
     +          mltn(j+(mxsord+1)*(i-1))
          lden(j+(mxsord+1)*((cuse-1)+order*(i-1)))=
     +          mltd(j+(mxsord+1)*(i-1))
1     continue
      return
      end


$include:'add.blk'
*******************************************************************************
* LOADTF * LOAD TRANSFER FUNCTION MATRIX                                      *
*******************************************************************************
* Descr:   Here the transfer function data is loaded from disk.               *
*          NAME  set to ' ' if file is in error.                              *
*          If the file was saved compacted, it is decompacted automatically.  *
*                                                                             *
* Param:   DIR,NAME,EXT - filename specifications                             *
*          ARRn,d,l refers to parameter arrays                                *
*          ORDER is the matrix order (n*n)                                    *
*          MXSORD is the maximum order of s of the saved file                 *
*          EXIST set false                                                    *
*                                                                             *
* Call.by: EDITTF                                                            *
* Calls:   ansisup,DISK,READLN,(AUTOORD)                                      *
*******************************************************************************

      subroutine loadtf(dir,name,ext,arrn,arrd,arrl,
     +                  order,mxsord,exist)

      character*4  ext
      character*8  name
      character*20 dir
      character*32 filen
```

```
      character*30 message
      real         arrn(*),arrd(*),arrl(*)
      logical      exist
      integer      key,order,mxtord,mxsord,tlorder,t2order,i,j
      integer      getch,iocheck
      external     getch
      logical      autoord
      external     autoord


1     call disk(filen,dir,name,ext)
      open(1,file=filen,iostat=iocheck)
      read(1,'(3i2)',err=6)tlorder,t2order,mxtord
      mxtord=mxtord-1
      if (tlorder.ne.order) then
        if (autoord()) then
          order=tlorder
          mxsord=1000/order/order-1.5
          if (mxsord.gt.20) mxsord=20
        endif
      endif
      if((t2order.ne.order).or.(tlorder.ne.order).or.(mxsord.lt.
     +   mxtord)) goto 10
      do 3 i=1,order
        do 3 j=1,order
          do 3 k=1,mxsord+1
            ijk=k+(mxsord+1)*((j-1)+order*(i-1))
            if (k.le.mxtord+1) then
              read(1,'(e13.7)',err=6)arrn(ijk)
            else
              arrn(ijk)=0.
            endif
3     continue
      do 4 i=1,order
        do 4 j=1,order
          do 4 k=1,mxsord+1
            ijk=k+(mxsord+1)*((j-1)+order*(i-1))
            if (k.le.mxtord+1) then
              read(1,'(e13.7)',err=6)arrd(ijk)
            else
              arrd(ijk)=0.
            endif
4     continue
      do 5 i=1,order*order
5       read(1,'(e13.7)',err=6)arrl(i)
      exist=.false.
      close(1,iostat=iocheck)
      if (iocheck.ne.0) goto 6
      goto 100

6     call error('File corrupted!                ')
      name=' '
      close(1)
      goto 100
10    write(message,'(a,i2.2)')'Order incorrect!   Set to : ',
     +     tlorder
      call error(message)
      name=' '
100   return
      end
```

```
c       Check if no parameters have been specified before
c       then update process order

        logical function autoord()
        logical result

$include:'design.blk'

         if ((kfilen.eq.' ').and.
     +    (lfilen.eq.' ').and.(ffilen.eq.' ')) then
           result=.true.
         else
           result=.false.
         endif
         autoord=result
        return
        end


$include:'add.blk'
*******************************************************************************
* LOADWF * LOAD FRQUENCY SAMPLE POINTS                                        *
*******************************************************************************
* Descr:   Here the frequency sample data  is loaded from disk.               *
*                                                                             *
*                                                                             *
* Param:   DIR,NAME,EXT - Filename specifications.                            *
*          EXIST - set to logical false                                       *
*          ARR   -  refers to parameter array                                 *
*          NOOFW  -  the number of samples                                    *
*                                                                             *
* Call.by: EDITWF                                                            *
* Calls:   ERROR                                                              *
*******************************************************************************
        subroutine loadwf(dir,name,ext,arr,noofw,exist)

        character*8  name
        character*20 dir,input
        character*4  ext
        real         arr(*)
        logical      exist,found
        character*32 filen
        integer      getch,key,noofw,iocheck,i,j
        external     getch

        call disk(filen,dir,name,ext)
1       open(1,file=filen,iostat=iocheck)
        read(1,'(i3)',err=2)noofw
        do 3 i=1,noofw
3       read(1,'(e13.7)',err=2)arr(i)
        close(1)
        exist=.false.
        goto 100

2       call error('Frequency file corrupted!     ')
        noofw=1
        arr(1)=1
        close(1)
100     return
        end
```

```
$include:'add.blk'
*******************************************************************************
* MATMLT * MATRIX MULTIPLICATION                                              *
*******************************************************************************
* Descr:   This routine multiplies two complex matrices.                      *
*                                                                             *
* Param:   MXC = MXA x MXB                                                    *
*          ORDER is the matrix order nxn.                                     *
*                                                                             *
* Call.by: CALCQI                                                             *
* Calls:   none.                                                              *
*******************************************************************************

        subroutine matmlt(mxa,mxb,mxc,order)

        integer    order,dx,ax,bx,offset
        complex    mxa(*),mxb(*),mxc(*)

c       use double precision in calculations
        complex*16 mxd(100)

        do 1 i=1,order
          offset=(i-1)*order
          do 1 j=1,order
            dx=j+offset
            mxd(dx)=(0,0)
            do 1 k=1,order
1             mxd(dx)=mxd(dx)+mxa(k+offset)*mxb(j+(k-1)*order)
        do 2 i=1,order*order
2         mxc(i)=mxd(i)
        return
        end


$include:'add.blk'
*******************************************************************************
* MNURST * MENU RESET                                                         *
*******************************************************************************
* Descr:   Initialises the menu data with the necessary data.                 *
*                                                                             *
* Call.by: INACAD                                                             *
* Calls:   ansisup                                                            *
*******************************************************************************

            function ip(i)
c           Used as a counter to fill menu items
            integer  i

            i=i+1
            ip=i
            return
            end

        subroutine mnurst

        integer    i

$INCLUDE:'MENU.BLK'
```

```
i=0
top(1)=1
top(2)=20
oldmnu(1)=1
depth=1
mnuinf(1,1)=i+1
choice(ip(i))=' INA DESIGN PROGRAM '
messge(i)=' '
mnuinf(2,1)=i
chcdfl(1)=0
mnuinf(1,2)=i+1
choice(ip(i))=' DOS Command       '
messge(i)=' Execute a DOS Command.                '
choice(ip(i))=' Help              '
messge(i)=' Get help on updated information.      '
choice(ip(i))=' Design Status     '
messge(i)=' Show design setup & enter design name. '
choice(ip(i))=' Process Parameters'
messge(i)=' Select process paramters.             '
choice(ip(i))=' Diagonalisation   '
messge(i)=' Diagonalise Process Response.         '
choice(ip(i))=' Clear design      '
messge(i)=' Clear design parameters.              '
choice(ip(i))=' Quit Program      '
messge(i)=' Return To DOS .                       '
mnuinf(2,2)=i
chcdfl(2)=3
mnuinf(1,3)=i+1
choice(ip(i))=' File Directories  '
messge(i)=' Temporary & parameter file directories.'
choice(ip(i))=' Process Order     '
messge(i)=' Change order of the process matrices.  '
choice(ip(i))=' Frequency Range   '
messge(i)=' Select frequency sample range.        '
choice(ip(i))=' Plant Matrix G(s) '
messge(i)=' Edit/Load/Save/Print plant matrix G(s).'
choice(ip(i))=' Controller Mx K(s) '
messge(i)=' Edit/Load/Save/Print matrix K(s).     '
choice(ip(i))=' Controller Mx L(s) '
messge(i)=' Edit/Load/Save/Print matrix L(s).     '
choice(ip(i))=' Feedback Mx F(s)   '
messge(i)=' Edit/Load/Save/Print feedback mx F(s). '
choice(ip(i))=' Return '
messge(i)=' Return To TOP LEVEL Menu.'
mnuinf(2,3)=i
chcdfl(3)=3
mnuinf(1,4)=i+1
choice(ip(i))=' Nyquist Setup     '
messge(i)=' Select Nyquist Diagram options.       '
choice(ip(i))=' Plot Nyquist      '
messge(i)=' Show resulting Nyquist Diagram.       '
choice(ip(i))=' Row/Col. operation '
messge(i)=' Perform a row or column operation.    '
choice(ip(i))=' Swop Rows /Columns '
messge(i)=' Interchange rows/columns of Q(s).     '
choice(ip(i))=' Multiply Row/Col.  '
messge(i)=' Multiply a row or column with function.'
choice(ip(i))=' Undo Operation    '
messge(i)=' Undo alteration of K(s) or L(s).      '
choice(ip(i))=' Print Response    '
messge(i)=' Print frequency response of process.  '
```

```
choice(ip(i))=' Return '
messge(i)=' Return To TOP LEVEL Menu.                '
mnuinf(2,4)=i
chcdfl(4)=1
mnuinf(1,5)=i+1
choice(ip(i))=' Load Transfer Fn    '
messge(i)=' Load transfer function file from disk. '
choice(ip(i))=' Edit Transfer Fn    '
messge(i)=' Create or edit transfer function.      '
choice(ip(i))=' Save Transfer Fn    '
messge(i)=' Save transfer function file to disk.   '
choice(ip(i))=' Print Transfer Fn   '
messge(i)=' Print Coefficient Matrix.'
choice(ip(i))=' Remove Parameter    '
messge(i)=' Delete parameter matrix from Design.   '
choice(ip(i))=' Return '
messge(i)=' Return To PROCESS PARAMETERS Menu.     '
mnuinf(2,5)=i
chcdfl(5)=0
mnuinf(1,6)=i+1
choice(ip(i))=' Load Frequency Data'
messge(i)=' Load frequency range from Disk.        '
choice(ip(i))=' Edit Frequency Data'
messge(i)=' Create or change frequency range.          '
choice(ip(i))=' Save Frequency Data'
messge(i)=' Save frequency range to Disk.          '
choice(ip(i))=' Return              '
messge(i)=' Return To PROCESS PARAMETERS Menu.     '
mnuinf(2,6)=i
chcdfl(6)=0
mnuinf(1,7)=i+1
choice(ip(i))=' Operation Options   '
messge(i)=' Select element to ellim. & row to use. '
choice(ip(i))=' Edit Multiplier     '
messge(i)=' Enter multiplication factor.           '
choice(ip(i))=' Optimise Multiplier'
messge(i)=' Do optimisation of multiplier function.'
choice(ip(i))=' Plot Multiplier     '
messge(i)=' Plot multiplication factor & function. '
choice(ip(i))=' Adjust Scale        '
messge(i)=' Select scale for plot of multiplier.   '
choice(ip(i))=' Perform Operation   '
messge(i)=' Update altered K(s) or L(s) matrix.    '
choice(ip(i))=' Print Operation     '
messge(i)=' Print operation performed on Printer.  '
choice(ip(i))=' Print Plot          '
messge(i)=' Print plotted points on Printer.       '
choice(ip(i))=' Return              '
messge(i)=' Return To PERFORM INA DESIGN Menu.     '
mnuinf(2,7)=i
chcdfl(7)=0
mnuinf(1,8)=i+1
mnuinf(2,8)=i
chcdfl(8)=0
mnuinf(1,9)=i+1
choice(ip(i))=' Operation Options   '
messge(i)=' Select Row or Column to multiply.      '
choice(ip(i))=' Edit Multiplier     '
messge(i)=' Edit multiplication funtion.           '
choice(ip(i))=' Optimise Multiplier'
messge(i)=' Do optimisation of multiplier function.'
```

```
        choice(ip(i))=' Plot Multiplier    '
        messge(i)=' Plot element inverse & multiplier.     '
        choice(ip(i))=' Adjust Scale       '
        messge(i)=' Select scale for plot of multiplier.   '
        choice(ip(i))=' Perform Operation  '
        messge(i)=' Update altered K(s) or L(s) matrix.    '
        choice(ip(i))=' Print Operation    '
        messge(i)=' Print operation on printer.            '
        choice(ip(i))=' Print Plot         '
        messge(i)=' Print plotted points on Printer.       '
        choice(ip(i))=' Return             '
        messge(i)=' Return To PERFORM INA DESIGN Menu.     '
        mnuinf(2,9)=i
        chcdfl(9)=0
        mnuinf(1,10)=i+1
        choice(ip(i))=' Swop Rows        '
        messge(i)=' Select rows to interchange.            '
        choice(ip(i))=' Swop Columns     '
        messge(i)=' Select columns to interchange.         '
        choice(ip(i))=' Perform Operation  '
        messge(i)=' Update altered K(s) or L(s) matrix.    '
        choice(ip(i))=' Print Operation  '
        messge(i)=' Print operation on printer.            '
        choice(ip(i))=' Return           '
        messge(i)=' Return To PERFORM INA DESIGN Menu.     '
        mnuinf(2,10)=i
        chcdfl(10)=0
        return
        end


$include:'add.blk'
*******************************************************************************
* NELM   * NONLINEAR PARAMETER ADJUSTMENT METHOD                               *
*******************************************************************************
* Descr:   This routine performs a nonlinear parameter adjustment             *
*          of a function to reduce the sum of error^2.                        *
*                                                                             *
*                                                                             *
* Param:  N - Number of parameters to adjust (max.21)                         *
*         X(N) - Parameters to adjust.                                        *
*         F    - Sum of square of error                                       *
*         EPS  - Convergence crriterion specifications:                       *
*                end IF SQR((F(i)-F(lowest))/(N+1)) < EPS*N                   *
*         P(N*(N+2)) - temporary storage                                      *
*         FP(N+1)    -      "                                                *
*         XS(N)      -      "                                                *
*         H(N)       - Step size corresponding to X(N)                        *
*         IX         - set to 0 when routine first entered                    *
*                                                                             *
* Call.by:                                                                    *
* Calls:   CALCF     - Function calculating Sum of error squared.             *
*******************************************************************************

        subroutine nelm(ix,x,h,n,f,eps,err)

        real       x(*),f,fs,eps,p(483),fp(22),xs(21),h(*)
        integer    ix,i,j,k,k1,nn,n,n1,np,xn,ih,is,k0,il,key
        logical    kbhit,err
        integer    getch
        external   kbhit,getch
```

```
          err=.false.
          call saverc
          n1=n+1
          np=n*(n+2)
          nn=n*(n+1)
          call calcf(n,x,f)
          fp(1)=f
          if (ix.eq.0) then
            do 1 i=1,n
              k=i
              do 1 j=1,n1
                p(k)=x(i)
                if (i.eq.j-1) p(k)=x(i)+h(i)
1           k=k+n
          endif
2         k=1+n
          do 3 i=2,n1
            do 4 j=1,n
              x(j)=p(k)
              k=k+1
4           continue
            call calcf(n,x,f)
            fp(i)=f
3         continue
          if (fp(1)-fp(2)) 303,303,5
303       ih=2
          il=1
          go to 6
5         ih=1
          il=2
6         do 7 i=3,n1
          if (fp(i)-fp(ih)) 304,304,8
304       if (fp(i)-fp(il)) 305,7,7
305       il=i
          go to 7
8         ih=i
7         continue
          xn=n
50        k1=nn
          do 9 i=1,n
            k=i
            s=0.0
            do 10 j=1,n1
              if (j.ne.ih) s=s+p(k)
              k=k+n
10          continue
            k1=k1+1
            p(k1)=s/xn
9         continue
          k=nn+1
          do 11 i=1,n
            x(i)=p(k)
11        k=k+1
          call calcf(n,x,f0)
          s=0.0
          do 12 i=1,n1
12        s=s+(fp(1)-f0)**2
          s=s/xn
          if (sqrt(s)-eps) 100,100,307
307       if (kbhit()) goto 100
```

```
          call loadrc
          write(*,'(1p,e14.6,a,1p,e14.6\)') fp(il),'     ',sqrt(s)
          if (ih-1) 308,13,308
308       is=1
          go to 14
13        is=2
14        continue
          do 15 i=1,n1
            if (i.ne.ih) then
              if (fp(i).lt.fp(is)) is=i
            endif
15        continue
c         REFLECTION
          k=(ih-1)*n+1
          k0=nn+1
          do 16 i=1,n
            x(i)=2.*p(k0)-p(k)
            k=k+1
            k0=k0+1
16        continue
          k=k-n
          call calcf(n,x,f)
c         EXPANSION
          if (f-fp(il))311,20,20
311       k0=nn+1
          do 17 i=1,n
            xs(i)=2.*x(i)-p(k0)
17        k0=k0+1
          call calcf(n,xs,fs)
          if (fs-fp(il))312,18,18
312       do 19 i=1,n
            p(k)=xs(i)
19        k=k+1
          fp(ih)=fs
          il=ih
          ih=is
          go to 30
18        il=ih
          ih=is
          fp(il)=f
21        do 22 i=1,n
            p(k)=x(i)
22        k=k+1
          go to 30
20        if(f-fp(is))313,23,23
313       fp(ih)=f
          ih=is
          go to 21
23        if(f-fp(ih))314,25,25
314       do 24 i=1,n
            p(k)=x(i)
24        k=k+1
          fp(ih)=f
c         CONTRACT
          k=k-n
25        k0=nn+1
          do 26 i=1,n
            xs(i)=.5*(p(k)+p(k0))
            k=k+1
26        k0=k0+1
          k=k-n
```

```
          call calcf(n,xs,fs)
          if (fs-fp(ih))315,40,40
315       do 27 i=1,n
            p(k)=xs(i)
27        k=k+1
          fp(ih)=fs
          if(fp(1)-fp(2)) 316,316,28
316       ih=2
          go to 29
28        ih=1
29        continue
          do 31 i=3,nl
            if (fp(i).gt.fp(ih)) ih=i
31        continue
30        go to 50
40        fp(1)=fp(il)
          if (il-1) 318,43,318
318       k=(il-1)*n
          do 41 i=1,n
            k=k+1
            x(i)=p(k)
            p(k)=p(i)
41        p(i)=x(i)
          il=1
43        k=n
          do 42 i=2,nl
            do 42 j=1,n
              k=k+1
42        p(k)=.5*(p(k)+p(j))
          go to 2
100       k=(il-1)*n
          do 101 i=1,n
            k=k+1
101       x(i)=p(k)
          f=fp(il)
222       if (kbhit()) then
            call error('Optimisation ABORTED!         ')
            err=.true.
          endif
          return
          end

c         Calculate resulting sum of error^2

          subroutine calcf(n,x,errsum)
          integer      n
          real         x(*),errsum
          complex      result
          integer      wi
          real         pi
          complex      kw,nw,dw,lw

$include:'desdta.blk'
$include:'rowop.blk'


        errsum=0.
        pi=3.1415926536
        do 67 wi=1,noofw,wint
          call calcpw(nw,x(1),rsord,w(wi))
          call calcpw(dw,x(rsord+2),rsord,w(wi))
```

```
          if (cabs(dw).le.1.e-8) then
            result=(0,0)
          else
            if (mod(n,2).ne.0) then
              lw=cmplx(0.,amod( x(n)*w(wi) , 2 * pi ))
              result=nw/dw*cexp(lw)
            else
              result=nw/dw
            endif
          endif
        errsum=errsum+cabs(quotient(wi)-result)**2
67      continue
69      return
        end


$include:'add.blk'
*******************************************************************************
* NYQSTP * NYQUIST DIAGRAM SETUP                                              *
*******************************************************************************
* Descr:   In this subroutine the Nyquist Diagram Parameters can be changed. *
*                                                                             *
* Call.by: DIAGON                                                             *
* Calls:   ansisup,                                                           *
*******************************************************************************

            subroutine nyqstp

            logical      change,escape
            character*20 input
            integer      i,j,key,getch,pos
            external     getch

$include:'design.blk'
$include:'desdta.blk'

            pos=1
            call normal
            call gotorc(4,17)
          write(*,'(a\)')'Press <RTN> to move cursor to desired option.'
            call gotorc(25,17)
            write(*,'(a,\)')'Press <ESC> to exit Nyquist options menu.'
1           call normal
            call gotorc(6,20)
            if (single) then
              write(*,'(a\)') 'ELEMENT(s) to display on plot: '
              write(*,'(a,i2.2,a,i2.2,a,\)')'(',si,',',sj,')'
            else
              write(*,'(a\)') 'ELEMENT(s) to display on plot: ( ALL )'
            endif
            call gotorc(9,20)
            write(*,'(a,a1\)')'SCALE  plot to axis range of :',241
            write(*,'(1p,e14.7\)') inadim
            call gotorc(12,20)
            write(*,'(a\)')   'TYPE of Nyquist Array Diagram '
            if (qinvrt) call invers
            write(*,'(a,\)')' INVERSE '
            call normal
            write(*,'(a\)')' or '
            if (.not.qinvrt) call invers
            write(*,'(a,\)')' DIRECT '
```

```
            call normal
            call gotorc(15,20)
          write(*,'(a,i3\)')'Number of Gershgorin circles :',abs(gerint)
            call gotorc(18,20)
          write(*,'(a,i3\)')'Number of Ostrowski  circles :',abs(ostint)
            call gotorc(21,20)
            write(*,'(a\)')   'DOMINANCE use to plot circles '
            if (gerint+ostint.lt.0) call invers
            write(*,'(a,\)')' COLUMN '
            call normal
            write(*,'(a\)')' or '
            if (gerint+ostint.ge.0) call invers
            write(*,'(a,\)')'  ROW   '
            call inverse
            if (pos.eq.1) then
              call gotorc(7,20)
              write(*,'(2a,\)')' Select element as ''row,column''',
     +         ' or enter 0 for ALL.'
2             call gotorc(6,52)
              call normal
              call readln(input,5,escape)
              if (escape) goto 100
              if (input.ne.' ') then
                read(input,122,err=1) i,j
122             format(bn,2i2)
              if ((i.ge.0).and.(i.le.order).and.(j.ge.0).and.(j.le.order))
     +               then
                  si=i
                  sj=j
                  if ((si.ne.0).and.(sj.ne.0)) then
                    single=.true.
                  else
                    single=.false.
                  endif
                  qexist=.false.
                else
                  write(*,'(a1\)')7
                  goto 2
                endif
              endif
              pos=2
              call gotorc(7,20)
              call normal
              call clreol
              call inverse
            endif
            if (pos.eq.2) then
              call gotorc(10,20)
              write(*,'(2a,\)')' Select maximum axis range value,',
     +'or 0 for automatic scaling.'
3             call gotorc(9,51)
              call saverc
              call normal
              call readln(input,20,escape)
              if (escape) goto 100
              if (input.ne.' ') then
                read(input,222,err=3) inadim
                inadim=abs(inadim)
                call loadrc
                write(*,'(1p,e14.7\)') inadim
                qexist=.false.
```

```
            endif
222         format(e13.7e2)
            pos=3
            call gotorc(10,20)
            call normal
            call clreol
            call inverse
          endif
          if (pos.eq.3) then
            call gotorc(13,23)
            write(*,'(a,\)')' Press <SPACE BAR> to change type.'
4           call gotorc(12,50)
            call saverc
            key=getch()
            if (key.eq.27) goto 100
            if (key.eq.32) then
              qinvrt=.not.qinvrt
              qexist=.false.
              call loadrc
              call normal
              if (qinvrt) call invers
              write(*,'(a,\)')' INVERSE '
              call normal
              write(*,'(a\)')' or '
              if (.not.qinvrt) call invers
              write(*,'(a,\)')' DIRECT '
              call normal
            endif
            if (key.ne.13) goto 4
            pos=4
            call gotorc(13,20)
            call normal
            call clreol
            call inverse
          endif
          if (pos.eq.4) then
            call gotorc(16,23)
            write(*,'(a,i3,a\)')' Select amount from 0 to ',noofw,'.'
5           call gotorc(15,50)
            call normal
            call readln(input,3,escape)
            if (escape) goto 100
            if (input.ne.' ') then
              read(input,444,err=5) i
444           format(bn,i3)
              if ((i.lt.0).or.(i.gt.noofw)) then
                write(*,'(a1\)')7
                goto 5
              endif
              if ((ostint.lt.0).or.(gerint.lt.0)) then
                gerint=-i
              else
                gerint=i
              endif
              qexist=.false.
            endif
            pos=5
            call gotorc(16,20)
            call normal
            call clreol
            call inverse
```

```
            endif
            if (pos.eq.5) then
               call gotorc(19,23)
               write(*,'(a,i3,a\)')' Select amount from 0 to ',noofw,'.'
6              call gotorc(18,50)
               call normal
               call readln(input,3,escape)
               if (escape) goto 100
               if (input.ne.' ') then
                 read(input,444,err=6) i
                 if ((i.lt.0).or.(i.gt.noofw)) then
                   write(*,'(a1\)')7
                   goto 6
                 endif
                 if ((ostint.lt.0).or.(gerint.lt.0)) then
                   ostint=-i
                 else
                   ostint=i
                 endif
                 qexist=.false.
               endif
               pos=6
               call gotorc(19,20)
               call normal
               call clreol
               call inverse
             endif
             if (pos.eq.6) then
               if (ostint+gerint.eq.0) then
                 goto 8
               endif
               call gotorc(22,23)
               write(*,'(a,\)')' Press <SPACE BAR> to change dominance.'
7              call gotorc(21,50)
               call saverc
               key=getch()
               if (key.eq.27) goto 100
               if (key.eq.32) then
                 ostint=-ostint
                 gerint=-gerint
                 qexist=.false.
                 call loadrc
                 call normal
                 if (gerint+ostint.lt.0) call invers
                 write(*,'(a,\)')' COLUMN '
                 call normal
                 write(*,'(a\)')' or '
                 if (gerint+ostint.ge.0) call invers
                 write(*,'(a,\)')'  ROW   '
                 call normal
               endif
               if (key.ne.13) goto 7
8              pos=1
               call gotorc(22,20)
               call normal
               call clreol
             endif
             goto 1
100          return
             end
```

```
$include:'add.blk'
*******************************************************************************
* OPTIM  * OPTIMISATION                                                       *
*******************************************************************************
* Descr:   Here the options for the optimisation routine are selected.        *
*                                                                             *
* Call.by: ROWOP                                                              *
* Calls:   ansisup,PLTMLT,NELM                                                *
*******************************************************************************

          subroutine optim

          real         rkd2(10),rkl2,rkn2(10)
          real         x(21),stpn(21),sumerr
          logical      escape,change,err
          integer      i,getch,key
          character*20 input
          external     getch

$include:'rowop.blk'
$include:'desdta.blk'

123         do 1 i=1,10
              rkd2(i)=rkd(i)
1             rkn2(i)=rkn(i)
            rkl2=rkl
            call gotorc(4,20)
            write(*,'(a\)')'Nonlinear Parameter Adjustment Procedure'
            call gotorc(6,15)
            write(*,'(2a\)')'This routine attempts to optimise ',
     +'the entered function.'
            call gotorc(9,15)
            write(*,'(2a\)')'Enter the starting stepsize ',
     +'to optimise all coefficients'
            call gotorc(10,15)
            write(*,'(a\)')'as a percentage of its initial value.'
            call gotorc(11,15)
            write(*,'(a\)')' (N.B. Coefficients = 0 will remain 0.)'
            call gotorc(25,20)
            write(*,'(a\)')'Press <ESC> to exit optimisation.'
39          call gotorc(13,15)
            write(*,'(a\)')'  Stepsize : '
            call saverc
            write(*,'(f6.2,a\)')stp,'%'
            call loadrc
            call readln(input,6,escape)
            if (escape) goto 100
            if (input.eq.' ') goto 40
            read(input,331,err=39) stp
            if (stp.eq.0) then
              write(*,'(a1\)')7
            endif
            goto 39
331         format(bn,e6.2)
40          call gotorc(15,15)
          write(*,'(a,i3,a\)')'Select frequency sample interval from 1-',
     + noofw,'.'
            if (wint.gt.noofw) wint=noofw
            if (wint.eq.0) wint=1
            call gotorc(16,15)
```

```
            write(*,'(a\)')' Interval: ',
            call saverc
            write(*,'(i3\)')wint
            call loadrc
            call readln(input,3,escape)
            if (escape) goto 100
            if (input.eq.' ') goto 41
            read(input,333,err=40) wint
            if ((wint.le.0).or.(wint.gt.noofw)) then
              write(*,'(a1\)')7
            endif
            goto 40
333         format(bn,i3)
41          call gotorc(18,15)
            write(*,'(a\)')'Enter Convergence Epsilon : '
            call saverc
            write(*,'(1p,e13.6\)')eps
            call loadrc
            call readln(input,13,escape)
            if (escape) goto 100
            if (input.eq.' ') goto 441
            read(input,332,err=40) eps
            if (eps.lt.0) then
              write(*,'(a1,\)')7
              eps=0.
            endif
            goto 41
332         format(bn,f13.6)
441         call gotorc(20,20)
            write(*,'(2a1,3a\)')' ',228,' Error**2   ',' ',
     +'   Epsilon    '
            call gotorc(25,20)
            call invers
            write(*,'(a\)')' Press key to START optimization. '
            if (getch().eq.27) goto 100
            call clRbot
            write(*,'(2a\)')' Press any ',
     +  'key to stop optimisation . . . '
            call normal
33          do 3 i=1,rsord+1
              x(i)=rkn(i)
              x(i+rsord+1)=rkd(i)
              stpn(i)=stp*rkn(i)/100.
3             stpn(i+rsord+1)=stp*rkd(i)/100.
            change=.true.
            n=2*rsord+2
            if (rkl.ne.0) then
              n=n+1
              x(n)=rkl
              stpn(n)=stp*rkl/100.
            endif
            ct=0
            call gotorc(21,18)
            call nelm(ct,x,stpn,n,sumerr,eps,err)
            if (err) then
              call gotorc(20,1)
              call clreol
              call gotorc(21,1)
              call clreol
            endif
            do 4 i=1,rsord+1
```

```
              rkn(i)=x(i)
4             rkd(i)=x(i+rsord+1)
            if (mod(n,2).ne.0) rkl=x(n)
            call invers
            call gotorc(23,20)
      write(*,'(a1,a,1p,e15.6,a\)')7,' Sum of Errors**2 = ',sumerr,' '
            call normal
            call gotorc(25,1)
            call clreol
            call gotorc(25,20)
            call invers
            write(*,'(a\)')'  Press key for PLOT...  '
            call normal
            if (getch().eq.27) goto 99
            call pltmlt
            call cls
            call invers
            call gotorc(15,15)
            write(*,'(a\)')' Press key to accept ALTERED Function,'
            call gotorc(16,15)
            write(*,'(a\)')' or press <ESC> to UNDO optimisation. '
            call normal
            if (getch().ne.27) goto 100
99          do 2 i=1,10
              rkn(i)=rkn2(i)
2             rkd(i)=rkd2(i)
            rkl=rkl2
100         return
            end


$include:'add.blk'
*******************************************************************************
* PLTAX2 * PLOT AXIS FOR MLT DIAGRAM                                          *
*******************************************************************************
* Descr:   Here the axis for the ina plot is drawn.                           *
*                                                                             *
* Call.by: PLTMLT                                                            *
* Calls:   Hercules Support                                                   *
*******************************************************************************

        subroutine pltax2

        integer    i,j,k,order,m
        real       delatx,deltay,x,y,dely,delx,mlt,xdim,ydim

$include:'dim.blk'

        order=1
        call clrscr
        call putpt(0,0)
        call dline(521,0)
        call dline(521,347)
        call dline(0,347)
        call dline(0,0)
        deltax=0.9*521./order/2.
        delx=521./order/20.
        deltay=0.9*347./order/2.
        dely=347./order/20.
        ydiv=347./2.
        xdiv=521./2.
```

```
          xdim=xhi-xlo
          ydim=yhi-ylo
          x=-521.*xlo/xdim
          y=347+347.*ylo/ydim
          call circ(nint(x),nint(y),3)
          if (x.gt.521) x=521
          if (y.gt.347) y=347
          if (x.lt.0) x=0
          if (y.lt.0) y=0
          call putpt(0,nint(y))
          call dline(521,nint(y))
          call putpt(nint(x),0)
          call dline(nint(x),347)
              do 1 k=1,20
                if (mod(k,5).eq.0) then
                  mlt=2.
                else
                  mlt=5.
                endif
                call putpt(nint(x+k*delx),nint(y+dely/mlt))
                call  dline(nint(x+k*delx),nint(y-dely/mlt))
                call putpt(nint(x-k*delx),nint(y+dely/mlt))
                call  dline(nint(x-k*delx),nint(y-dely/mlt))
                call putpt(nint(x+delx/mlt),nint(y+k*dely))
                call  dline(nint(x-delx/mlt),nint(y+k*dely))
                call putpt(nint(x+delx/mlt),nint(y-k*dely))
1               call  dline(nint(x-delx/mlt),nint(y-k*dely))
          return
          end

$include:'add.blk'
*******************************************************************************
* PLTAXS * PLOT AXIS FOR INA DIAGRAM                                          *
*******************************************************************************
* Descr:   Here the axis for the ina plot is drawn.                           *
*                                                                             *
* Param. : ORDER - order of diagram i.e. nxn set of axis                      *
*                                                                             *
* Call.by: PLTINA                                                            *
* Calls:   Hercules Support                                                   *
*******************************************************************************

          subroutine pltaxs(order)

          integer    i,j,k,order
          real       delatx,deltay,x,y,dely,delx,mlt

          call clrscr
          call putpt(0,0)
          call dline(521,0)
          call dline(521,347)
          call dline(0,347)
          call dline(0,0)
          deltax=0.9*521./order/2.
          delx=521./order/20.
          deltay=0.9*347./order/2.
          dely=347./order/20.
          do 1 i=1,order
            y=347.*(2*i-1)/order/2.
            do 1 j=1,order
```

```
              x=521.*(2*j-1)/order/2.
c              call putpt(nint(x-deltax),nint(y))
c              call dline(nint(x+deltax),nint(y))
c              call putpt(nint(x),nint(y-deltay))
c              call dline(nint(x),nint(y+deltay))
              do 1 k=1,9
                if (k.eq.5) then
                  mlt=2.
                else
                  mlt=5.
                endif
                call putpt(nint(x+k*delx),nint(y+dely/mlt))
                call  dline(nint(x+k*delx),nint(y-dely/mlt))
                call putpt(nint(x-k*delx),nint(y+dely/mlt))
                call  dline(nint(x-k*delx),nint(y-dely/mlt))
                call putpt(nint(x+delx/mlt),nint(y+k*dely))
                call  dline(nint(x-delx/mlt),nint(y+k*dely))
                call putpt(nint(x+delx/mlt),nint(y-k*dely))
1               call  dline(nint(x-delx/mlt),nint(y-k*dely))
          return
          end


$include:'add.blk'
*******************************************************************************
* PLTGER * PLOT GERSHGORIN AND OSTROWSKI CIRCLES                               *
*******************************************************************************
* Descr:   The Gershgorin and Ostrowski circles are plotted                    *
*          at a single frequency point.                                        *
*                                                                              *
*                                                                              *
* Param:   QW - Array containing response.                                     *
*          SINGLE - Set true if only one set of axis is used.                  *
*          SI,SJ  - Index of single element to plot.                           *
*          ORDER  - INA diagram order.                                         *
*          DIM    - Extents of axis                                            *
*          GERINT,OSTINT - Determine row or column dominace.                   *
*          FHANDLE - File handle of feedback matrix.                           *
*          ERROR - Flag indicating file access error.                          *
*                                                                              *
* Call.by: PLTINA.
*
* Calls:   ansisup,hercules support,DCIRC.                                     *
*******************************************************************************
          subroutine pltger(qw,single,si,sj,order,
     +                      dim,gerint,ostint,fhandle,error)

          complex qw(*),f(10),dummy
          real*8  div
          real    dim
          integer si,sj,order,gerint,ostint,fhandle,i,j,dom,index,indexs
          logical error,single
          real*8  gerrad(10),ostrad(10),max

          indexs=sj+(si-1)*order
          dom=gerint
          if (dom.eq.0) dom=ostint
          do 1 i=1,order
            gerrad(i)=0.
            do 1 j=1,order
```

```
            if (dom.gt.0) then
            if (j.ne.i) gerrad(i)=gerrad(i)+cabs(qw(j+(i-1)*order))
            else
            if (j.ne.i) gerrad(i)=gerrad(i)+cabs(qw(i+(j-1)*order))
            endif
1         continue
          if (ostint.ne.0) then
            if (fhandle.ne.0) then
              do 2 i=1,order
              do 2 j=1,order
                if (i.eq.j) then
                  read(fhandle,err=99)f(i)
                else
                  read(fhandle,err=99)dummy
                endif
2             continue
            else
              do 3 i=1,order
3               f(i)=(1,0)
            endif
            do 4 i=1,order
              max=0
              do 5 j=1,order
                if (i.ne.j) then
                  div=gerrad(j)/cabs(f(j)+qw(j+(j-1)*order))
c                  div=gerrad(j)/cabs(qw(j+(j-1)*order))
                  if (max.lt.div) max=div
                endif
5             continue
4             ostrad(i)=gerrad(i)*max
              do 7 i=1,order
                index=i+(i-1)*order
                if (single) then
                  if (index.eq.indexs)
     +              call dcirc(ostrad(i),1,1,qw(index),1,dim)
                else
                  call dcirc(ostrad(i),i,i,qw(index),order,dim)
                endif
7             continue
            endif
            if (gerint.ne.0) then
              do 6 i=1,order
                index=i+(i-1)*order
                if (single) then
                  if (index.eq.indexs)
     +              call dcirc(gerrad(i),1,1,qw(index),1,dim)
                else
                  call dcirc(gerrad(i),i,i,qw(index),order,dim)
                endif
6             continue
            endif
            goto 100
99          error=.true.
100         return
            end


c         Draw circle of radius RAD at I,J with center at QW

          subroutine dcirc(rad,i,j,qw,order,dim)
```

```
          integer    i,j,order,radius
          real       x,y,dim,xdiv,ydiv,xp,yp
          real*8     rad
          complex    qw

          x=real(qw)
          y=aimag(qw)
          radius=nint(rad*521./order/dim/2.)
          if ((abs(x).le.dim).and.(abs(y).le.dim)
     +         .and.(radius.gt.1)) then
            xdiv=521./order/2.*(2*j-1)
            ydiv=347./order/2.*(2*i-1)
            xp=xdiv+521.*x/order/2./dim
            yp=ydiv-347.*y/order/2./dim
            if (rad.le.2.*dim) then
              call circ(nint(xp),nint(yp),radius)
            else
              call putpt(nint(xp-6),nint(yp))
              call dline(nint(xp+6),nint(yp))
              call putpt(nint(xp),nint(yp-4))
              call dline(nint(xp),nint(yp+4))
            endif
          endif
          return
          end


$include:'add.blk'
*******************************************************************************
* PLTHE2 * PLOT HEADING FOR MULTIPLIER DIAGRAM                                 *
*******************************************************************************
* Descr:   Adds information to the Multiplier plot.                            *
*                                                                              *
* Call.by: PLTMLT.                                                             *
* Calls:   Hercules Support                                                    *
*******************************************************************************

          subroutine plthe2(start)

          character*20 str
          character*8 dfilen
          logical start

$include:'design.blk'
$include:'desdta.blk'
$include:'rowop.blk'
$include:'dim.blk'

          if (start) then
            call gmode
            call clrscr
            call level(1)
            call pltax2
          else
          call level(0)
          call blkfil(522,346,197,346)
          call level(1)
          call putpt(522,0)
          call dline(719,0)
          call dline(719,347)
          call dline(522,347)
```

```
        call dline(522,0)
        call putpt(1,1)
        call dline(1,346)
        call dline(718,346)
        call dline(718,1)
        call dline(1,1)
        call textf(522+9,14,20,' MULTIPLIER DIAGRAM ')
        call textf(522+9,14*2,20,'--------------------')
        call textf(522+9,14*3,20,dsnnme)
        call textf(522+9,14*4,20,' G(s) :             ')
        call textf(522+9*10,14*4,8,gfilen)
        call textf(522+9,14*5,20,' K(s) :             ')
        call textf(522+9*10,14*5,8,kfilen)
        call textf(522+9,14*6,20,' L(s) :             ')
        call textf(522+9*10,14*6,8,lfilen)
        call textf(522+9,14*7,20,' F(s) :             ')
        call textf(522+9*10,14*7,8,ffilen)
        call textf(522+9,14*8,20,' W(i) :             ')
        call textf(522+9*10,14*8,8,wfilen)
        call textf(522+9,14*10,20,' Process order   x  ')
        write(str,'(i2.2)')order
        call textf(522+9*16,14*10,2,str)
        call textf(522+9*19,14*10,2,str)
        call textf(522+9,14*12,20,' Element  :(   ,   )  ')
        write(str,'(i2.2)')ri
        call textf(522+9*13,14*12,2,str)
        write(str,'(i2.2)')rj
        call textf(522+9*16,14*12,2,str)
        if (cuse.ne.0) then
          write(str,'(a,i2.2,a)')' Use column:[',cuse,']'
          call textf(522+9,14*13,16,str)
        else
          write(str,'(a,i2.2,a)')' Use row  : [',ruse,']'
          call textf(522+9,14*13,16,str)
        endif
        call textf(522+9,14*15,20,' Frequency Range:   ')
        call textf(522+9,14*16,4,'  1:')
        write(str,'(1p,E10.3,a)') w(1),' Rad/s'
        call textf(522+9*5,14*16,16,str)
        write(str,'(i3,a,1p,E10.3,a)') noofw,':',w(noofw),' Rad/s'
        call textf(522+9,14*17,20,str)
        write(str,'(a,1p,E10.3)')' X min: ',xlo
        call textf(522+9,14*19,20,str)
        write(str,'(a,1p,E10.3)')'   max: ',xhi
        call textf(522+9,14*20,20,str)
        write(str,'(a,1p,E10.3)')' Y min: ',ylo
        call textf(522+9,14*21,20,str)
        write(str,'(a,1p,E10.3)')'   max: ',yhi
        call textf(522+9,14*22,20,str)
        if (qinvrt) then
          call textf(522+9,14*23,20,' Operating on 1/Q(s)')
        else
          call textf(522+9,14*23,20,' Operating on Q(s)  ')
        endif
        call textf(522+9,347-5,20, 'INACAD (c)R.Venzke  ')
        endif
        return
        end

$include:'add.blk'
```

```
*******************************************************************************
* PLTHED * PLOT HEADING FOR INA DIAGRAM                                       *
*******************************************************************************
* Descr:   Adds information to the INA plot.                                  *
*                                                                             *
* Param. : CLS - Flag set if information window is to be cleared first.       *
*                                                                             *
* Call.by: PLTINA.                                                           *
* Calls:   Hercules Support                                                   *
*******************************************************************************

          subroutine plthed(cls)

          logical      cls
          character*20 str
          character*8  dfilen

$include:'design.blk'
$include:'desdta.blk'

          if (cls) then
            call level(0)
            call blkfil(522,346,197,346)
          endif
          call level(1)
          call putpt(522,0)
          call dline(719,0)
          call dline(719,347)
          call dline(522,347)
          call dline(522,0)
          call putpt(1,1)
          call dline(1,346)
          call dline(718,346)
          call dline(718,1)
          call dline(1,1)
          if (qinvrt) then
            call textf(522+9,14,20,'   INVERSE NYQUIST  ')
          else
            call textf(522+9,14,20,'   DIRECT  NYQUIST  ')
          endif
          call textf(522+9,14*2,20,' ---------------- ')
          call textf(522+9,14*3,20,dsnnme)
          call textf(522+9,14*5,20,' G(s) :            ')
          call textf(522+9*10,14*5,8,gfilen)
          call textf(522+9,14*6,20,' K(s) :            ')
          call textf(522+9*10,14*6,8,kfilen)
          call textf(522+9,14*7,20,' L(s) :            ')
          call textf(522+9*10,14*7,8,lfilen)
          if (qinvrt) then
            call textf(522+9,14*8,20,' F(s) :              ')
            call textf(522+9*10,14*8,8,ffilen)
          endif
          call textf(522+9,14*9,20,' W(i) :            ')
          call textf(522+9*10,14*9,8,wfilen)
          call textf(522+9,14*11,20,' Process order   x  ')
          write(str,'(i2.2)')order
          call textf(522+9*16,14*11,2,str)
          call textf(522+9*19,14*11,2,str)
          if (single) then
            call textf(522+9,14*13,20,' Element : (   ,   )  ')
            write(str,'(i2.2)')si
```

```
            call textf(522+9*13,14*13,2,str)
            write(str,'(i2.2)')sj
            call textf(522+9*16,14*13,2,str)
          endif
          call textf(522+9,14*15,20,' Frequency Range:    ')
          call textf(522+9,14*16,4,'  1:')
          write(str,'(1p,E10.3,a)') w(1),' Rad/s'
          call textf(522+9*5,14*16,16,str)
          write(str,'(i3,a,1p,E10.3,a)') noofw,':',w(noofw),' Rad/s'
          call textf(522+9,14*17,20,str)
          write(str,'(a,a1,1p,E10.3)')' Axis: ',241,inadim
          call textf(522+9,14*19,20,str)
          if (gerint.ne.0)
     +      call textf(522+9,14*21,20,' Gershgorin Circles ')
          if (ostint.ne.0)
     +      call textf(522+9,14*22,20,' Ostrowski Circles  ')
          if ((gerint.gt.0).or.(ostint.gt.0)) then
            call textf(522+9,14*23,20,'   Row  Dominance   ')
          elseif ((gerint.ne.0).or.(ostint.ne.0)) then
            call textf(522+9,14*23,20,'  Column Dominance  ')
          endif
          call textf(522+9,347-5,20, 'INACAD(c)RHE Venzke ')
          return
          end


$include:'add.blk'
*******************************************************************************
* PLTINA * PLOT INA DIAGRAM                                                   *
*******************************************************************************
* Descr:   Here the inverse Nyquist diagram is calculated and then plotted.  *
*                                                                             *
* Call.by: DIAGON                                                            *
* Calls:   ansisup,DISK,PLTAXS,CALCQI,PLTPTS,PLTGER,Hercules support         *
*******************************************************************************

          subroutine pltina

          integer      i,j,k,offset,iocheck,readch,getch,t,fhandle,key
          integer      gerstp
          logical      err,calcg,calck,calcl,calcf,deterr
          logical      abort,kbhit,escape,trace,autoscale
          complex      det,qw(200),qwi
          character*32 filen
          external     kbhit,readch,getch
          character*20 input

$include:'desdta.blk'

$include:'design.blk'

          gerstp=0
          if (ostint+gerint.ne.0) then
            if (abs(ostint).gt.abs(gerint)) then
              gerstp=abs(ostint)
            else
              gerstp=abs(gerint)
            endif
            gerstp=nint(1.0*noofw/gerstp)
          endif
          call gmode
```

```
          call clrscr
22        autoscale=.false.
c         if (.not.qexist) then
          if (.true.) then
            if (inadim.eq.0) then
              autoscale=.true.
              inadim=2.
            endif
            deterr=.false.
            abort=.false.
            trace=.false.
            t=1
            offset=order*order
            err=.false.
              calcg=(gfilen.ne.' ')
              calck=(kfilen.ne.' ')
              calcl=(lfilen.ne.' ')
              calcf=(ffilen.ne.' ')
              if (calcg) then
                call disk(filen,tfdir,gfilen,'.~gw')
                open(1,file=filen,form='binary',iostat=iocheck)
              endif
              if (calck) then
                call disk(filen,tfdir,kfilen,'.~kw')
                open(2,file=filen,form='binary',iostat=iocheck)
              endif
              if (calcl) then
                call disk(filen,tfdir,lfilen,'.~lw')
                open(3,file=filen,form='binary',iostat=iocheck)
              endif
              call disk(filen,tfdir,'nyquist ','.~jw')
              open(9,file=filen,form='binary',status='NEW',
     +                                        iostat=iocheck)
              fhandle=0
              if (calcf) then
                call disk(filen,tfdir,ffilen,'.~fw')
                open(4,file=filen,form='binary',iostat=iocheck)
                call disk(filen,tfdir,ffilen,'.~f2')
                open(5,file=filen,form='binary',status='NEW',
     +                                         iostat=iocheck)
                do 12 i=1,noofw*offset
                  read(4,err=999) qwi
12                write(5,err=999) qwi
                rewind 5
                rewind 4
                fhandle=5
              endif
              if (single) then
                call pltaxs(nint(1.))
              else
                call pltaxs(order)
              endif
              call plthed(.false.)
              call level(0)
              call textf(522+9,14*23,20,' Press ESC to abort.')
              call level(1)
              call calcqi(qw,calcg,calck,calcl,calcf,err,det)
              if (cabs(det).lt.1.e-8) then
                do 1 j=1,order*order
1                 qw(j)=(0,0)
                deterr=.true.
```

```
              endif
              do 2 i=1,noofw-1
                if (trace) then
155               t=mod(t+1,2)
                  call level(t)
                  call textf(522+9,14*23,20,' Hit KEY to continue')
                  call level(1)
                  if (.not.kbhit()) goto 155
                endif
                if (kbhit()) then
                  key=readch()
                  if (key.eq.27) then
                    abort=.true.
                    goto 998
                  endif
                  if (lowcase(key).eq.116) then
                    trace=.not.trace
                    if (.not.trace) then
                      call level(0)
                      call textf(522+9,14*23,20,' Press ESC to abort.')
                      call level(1)
                    endif
                  endif
                endif
                if (.not.(err.or.abort)) then
                    call calcqi(qw(offset+1),calcg,calck,calcl,
     +                                 calcf,err,det)
                    if (cabs(det).lt.1.e-8) then
                      do 4 j=1,order*order
4                        qw(j+offset)=(0,0)
                      call level(t)
                      t=mod(t+1,2)
                    call textf(522+9,14*19,20,' Determinant *ERROR*')
                      call level(1)
                      deterr=.true.
                    endif
                    call pltpts(qw,single,si,sj,order,inadim,w(i))
            if ((gerint+ostint.ne.0).and.(mod(i,abs(gerstp)).eq.0))then
                    call pltger(qw,single,si,sj,order,inadim,gerint,
     +                    ostint,fhandle,err)
                endif
                    do 3 j=1,order*order
3                     qw(j)=qw(j+offset)
                endif
2             continue
              close(9,iostat=iocheck)
              if (iocheck.ne.0) goto 999
              if (.not.autoscale) then
                if (gerstp.ne.0) plthed(.true.)
              else
                call level(0)
                call textf(522+9,14*23,20,'REPLOT -> Rescaling ')
                call level(1)
                call disk(filen,tfdir,'nyquist ','.~jw')
            open(9,file=filen,form='binary',status='OLD',iostat=iocheck)
                inadim=0.
                do 35 i=1,offset*noofw
                  read(9,err=999) qwi
                  qwi=cmplx(abs(real(qwi)),abs(aimag(qwi)))
                  if (real(qwi).gt.inadim) inadim=real(qwi)
                  if (aimag(qwi).gt.inadim) inadim=aimag(qwi)
```

```
35              continue
                write(*,'(a1\)')7
                close(9)
              endif
              call textf(522+9,14*23,20,'                    ')
              goto 998
999           err=.true.
998           qexist=.true.
              if (err) then
                call tmode
                call error('File corrupted - Plot aborted!')
                qexist=.false.
              else if (abort) then
                write(*,'(a1\)')7
                i=getch()
                call tmode
                call clrbot
                write(*,'(8x,a,a\)')' USER INTERRUPT:  Plot aborted!',
     +            '  Press any key to continue...'
                qexist=.false.
              else if (deterr) then
                i=getch()
                call tmode
                call error('Determinant = zero !          ')
              else if (.not.autoscale) then
                i=getch()
                if (i.eq.109) call mcircle(single,si,sj,order,inadim)
                if (i.eq.109) i=getch()
              endif
              if (calcg) close(1)
              if (calck) close(2)
              if (calcl) close(3)
              if (calcf) then
                close(4)
                close(5)
              endif
            endif
            if (autoscale.and.qexist) then
              qexist=.false.
              goto 22
            endif
            call tmode
            return
            end

            subroutine mcircle(single,si,sj,order,inadim)
            integer si,sj,order
            logical single
            real    inadim,re,r,cx,m
            complex circ(50)

            m=1.3
            r=abs(m/(1-m*m))
            x=m*m/(1-m*m)
            do 1 i=1,50
              re=r*sin(i/50.0*2*3.1416)+x
              cx=r*cos(i/50.0*2*3.1416)
1           circ(i)=cmplx(re,cx)
            if (single) then
              do 3 i=1,49
3               call drawc(nint(1.),nint(1.),circ(i),nint(1.),
```

```
     +            nint(1.),inadim)
          else
            do 2 i=1,order
              do 2 j=1,49
                  call drawc(i,i,circ(j),nint(1.),order,inadim)
2           continue
          endif
          do 4 i=1,50
            re=sin(i/50.0*2*3.1416)
            cx=cos(i/50.0*2*3.1416)
4         circ(i)=cmplx(re,cx)
          if (single) then
            do 5 i=1,49
5             call drawc(nint(1.),nint(1.),circ(i),nint(1.),
     +            nint(1.),inadim)
          else
            do 6 i=1,order
              do 6 j=1,49
                  call drawc(i,i,circ(j),nint(1.),order,inadim)
6           continue
          endif
          return
          end


$include:'add.blk'
*******************************************************************************
* PLTMLT * PLOT MULTIPLIER FUNCTION AND QUOTIENT                               *
*******************************************************************************
* Descr:   This routine plots the multiplier function and                      *
*          the quotient on a set of axis.                                      *
*                                                                              *
* Call.by: ROWOP                                                               *
* Calls:   PLTHE2,ansisup,hercules graphic support.                            *
*******************************************************************************

          subroutine pltmlt

          logical      error,autoscale,user
          complex      result(100)
          real         pi,xp,yp
          real         x,y,x2,y2,temp
          complex      nw,dw,lw
          integer      i,getch,key,j,k
          external     getch

$include:'desdta.blk'
$include:'rowop.blk'
$include:'design.blk'
$include:'dim.blk'

          user=.false.
          if ((xhi.eq.0).and.(yhi.eq.0).and.
     +        (xlo.eq.0).and.(ylo.eq.0)) then
            autoscale=.true.
          else
            autoscale=.false.
          endif
          pi=3.1415926536
          do 1 i=1,noofw
            call calcpw(nw,rkn,rsord,w(i))
```

```
            call calcpw(dw,rkd,rsord,w(i))
            lw=cmplx(0.,amod( rkl*w(i) , 2 * pi ))
            if (cabs(dw).le.1.e-8) then
              result(i)=(0,0)
            else
              result(i)=nw/dw*cexp(lw)
            endif
            if (cabs(result(i)).ne.0.) user=.true.
1         continue
          if (autoscale) then
            xlo=real(quotient(1))
            ylo=aimag(quotient(1))
            xhi=real(quotient(1))
            yhi=aimag(quotient(1))
            do 3 i=2,noofw
      if (real(quotient(i)).gt.xhi) xhi=real(quotient(i))
      if (aimag(quotient(i)).gt.yhi) yhi=aimag(quotient(i))
      if (real(quotient(i)).lt.xlo) xlo=real(quotient(i))
      if (aimag(quotient(i)).lt.ylo) ylo=aimag(quotient(i))
            if (user) then
      if (real(result(i)).lt.xlo) xlo=real(result(i))
      if (aimag(result(i)).lt.ylo) ylo=aimag(result(i))
      if (real(result(i)).gt.xhi) xhi=real(result(i))
      if (aimag(result(i)).gt.yhi) yhi=aimag(result(i))
            endif
3           continue
          endif
          xdim=xhi-xlo
          ydim=yhi-ylo
          if (xdim.lt.0) then
            temp=xhi
            xhi=xlo
            xlo=temp
            xdim=-xdim
          endif
          if (ydim.lt.0) then
            temp=yhi
            yhi=ylo
            ylo=temp
            ydim=-ydim
          endif
          if (xdim.lt.1.e-4) then
            xlo=xlo-0.5
            xhi=xhi+0.5
            xdim=xhi-xlo
          endif
          if (ydim.lt.1.e-4) then
            ylo=ylo-0.5
            yhi=yhi+0.5
            ydim=yhi-ylo
          endif
          call plthe2(.true.)
          do 2 i=1,noofw-1
            x=real(quotient(i))
            y=aimag(quotient(i))
            x2=real(quotient(i+1))
            y2=aimag(quotient(i+1))
            xp=521.*(x-xlo)/xdim
            yp=347.-347.*(y-ylo)/ydim
            call putpt(nint(xp),nint(yp))
            call circ(nint(xp),nint(yp),3)
```

```
            xp=521.*(x2-xlo)/xdim
            yp=347.-347.*(y2-ylo)/ydim
            call dline(nint(xp),nint(yp))
            x=real(result(i))
            y=aimag(result(i))
            x2=real(result(i+1))
            y2=aimag(result(i+1))
            xp=521.*(x-xlo)/xdim
            yp=347.-347.*(y-ylo)/ydim
            call putpt(nint(xp+4),nint(yp+3))
            call dline(nint(xp-4),nint(yp-3))
            call putpt(nint(xp-4),nint(yp+3))
            call dline(nint(xp+4),nint(yp-3))
            call putpt(nint(xp),nint(yp))
            xp=521.*(x2-xlo)/xdim
            yp=347.-347.*(y2-ylo)/ydim
            call dline(nint(xp),nint(yp))
2           continue
            call plthe2(.false.)
            key=getch()
            if (key.eq.32) then
              do 4 i=1,noofw
              x=real(quotient(i))
              y=aimag(quotient(i))
              x2=real(result(i))
              y2=aimag(result(i))
              xp=521.*(x-xlo)/xdim
              yp=347.-347.*(y-ylo)/ydim
              call putpt(nint(xp),nint(yp))
              xp=521.*(x2-xlo)/xdim
              yp=347.-347.*(y2-ylo)/ydim
              call dline(nint(xp),nint(yp))
4             continue
              call plthe2(.false.)
              key=getch()
            endif
            call tmode
100         return
            end


$include:'add.blk'
*******************************************************************************
* PLTPTS * PLOT FREQUENCY POINTS FOR INA                                       *
*******************************************************************************
* Descr:   The data for the INA is plotted for one frequency sample.          *
*                                                                             *
* Param:   QW array containing ORDER*ORDER*2 frequency responses.             *
*             *2 because lines are plotted to next value                      *
*                                                                             *
* Call.by: PLTINA                                                             *
* Calls:   DRAWC.                                                             *
*******************************************************************************

            subroutine pltpts(qw,single,si,sj,order,dim,w)

            complex      qw(*)
            real         dim,w(*)
            integer      i,j,index,indexs,offset,si,sj,order,readch
            logical      single
            character*20 str
```

```
          offset=order*order
          indexs=sj+(si-1)*order
          do 1 i=1,order
            do 1 j=1,order
              index=j+(i-1)*order
              if (single) then
                if (index.eq.indexs)
     +            call drawc(nint(1.),nint(1.),qw(index),offset,
     +            nint(1.),dim)
              else
                call drawc(i,j,qw(index),offset,order,dim)
              endif
1         continue

c         display current frequency

          write(str,'(1p,E10.3)') w(2)
          call textf(522+9*5,14*18,10,str)
          return
          end



$include:'add.blk'
*******************************************************************************
* PRCPAR * PROCESS PARAMETERS                                                  *
*******************************************************************************
* Descr:   Shows menu to view and edit process matrices.                       *
*                                                                              *
* Call.by: INADSN                                                              *
* Calls:   ansisup,GETORD,FRANGE,EDITTF,GETCHC                                 *
*******************************************************************************

        subroutine prcpar

        integer    getch,key
        logical    dsnclr
        external   dsnclr,getch

$INCLUDE:'MENU.BLK'
$INCLUDE:'DESIGN.BLK'
$INCLUDE:'DESDTA.BLK'

11      menunb=3
        call getchc(menunb,escape)
        if (escape) goto 111
        if (chcdfl(menunb).eq.0) then
          if (dsnclr()) then
            call setpfd
          else
            call error('Design must be cleared first!')
          endif
        elseif (chcdfl(menunb).eq.1) then
          if (dsnclr()) then
            call getord
          else
            call error('Design must be cleared first!')
          endif
        elseif (chcdfl(menunb).eq.2) then
          call frange
```

```
        elseif (chcdfl(menunb).eq.3) then
          call edittf(gfilen,'.GS ','.~gw',gnum,gden,glag,gexist)
        elseif (chcdfl(menunb).eq.4) then
          call edittf(kfilen,'.KS ','.~kw',knum,kden,klag,kexist)
        elseif (chcdfl(menunb).eq.5) then
          call edittf(lfilen,'.LS ','.~lw',lnum,lden,llag,lexist)
        elseif (chcdfl(menunb).eq.6) then
          call edittf(ffilen,'.FS ','.~fw',fnum,fden,flag,fexist)
        endif
        goto 11
111     return
        end


$include:'add.blk'
*******************************************************************************
* PRINTTF* PRINT TRANSFER FUNCTION                                            *
*******************************************************************************
* Descr:   Prints selected transfer function matrix specification             *
*          on a printer or to the screen.                                     *
*                                                                             *
*                                                                             *
* Param:  PFDIR,NAME,EXT - Filename specifications.                           *
*         ARRN,ARRD,ARRL - matrix coefficients.                               *
*         ORDER - process order.                                              *
*         MXSORD -  maximum power of s used.                                  *
*                                                                             *
* Call.by: PRCPAR                                                            *
* Calls:   ansisup.                                                           *
*******************************************************************************

          subroutine printtf(pfdir,name,ext,arrn,arrd,arrl,order,mxsord)

          character*20 pfdir
          character*8  name
          character*4  ext
          character*32 filen
          real         arrn(*),arrd(*),arrl(*)
          integer      order,mxsord,i,k,j,sorder,ijk
          integer      getch,key
          external     getch

          call clrbot
          write(*,'(20x,a\)')' Printer READY (y/n) ?'
          key=getch()
          if (key.eq.27) goto 100
          if (lowcase(key).eq.121) then
            call clrbot
            write(*,'(a\)')' Printing . . .'
            open(2,file='lpt1:')
          else
            call cls
            open(2,file='con:')
          endif
          call dimtf(order,mxsord,sorder,arrn,arrd)
          call disk(filen,pfdir,name,ext)
          write(2,'(1h ,a)')' '
          write(2,'(1h ,2a)')' Process Matrix: ',filen
          write(2,'(1h ,a)') '============================ '
          write(2,'(1h ,a)')' '
          write(2,'(1h ,a)')
```

```
     +' Element     Order      Numerator      Denominator        Lag   '
          write(2,'(1h ,a)')
     +'---------   -------    -----------    -------------      -----  '
          do 3 i=1,order
            do 3 j=1,order
              write(2,'(1h ,a)')' '
              write(2,'(1h ,a,i2.2,a,i2.2,a\)')' (',i,',',j,') '
              do 3 k=1,sorder+1
                write(2,'(a,i2.2,a\)')'    s^',k-1,'   '
                ijk=k+(mxsord+1)*((j-1)+order*(i-1))
                write(2,'(a,1p,e13.6,a\)')'  ',arrn(ijk),' '
                write(2,'(a,1p,e13.6,a\)')'  ',arrd(ijk),' '
       if (k.eq.1) write(2,'(a,1p,e13.6,a\)')' ',arr1(j+(i-1)*order),' '
                write(2,'(a)')' '
                write(2,'(1h ,a\)')'          '
3         continue
          write(2,'(1h )')
          write(2,'(1h \)')
          call clrbot
          write(*,'(a\)')' Press <key> to continue . . .'
          key=getch()
          close(2)
100       return
          end


$include:'add.blk'
*******************************************************************************
* PRTMLT * PRINT MULTIPLIER AND ROW OPERATION DATA                             *
*******************************************************************************
* Descr:   Print selected or last row or column operation                      *
*          specifications. It displays the selected multiplier                 *
*          too.                                                                *
*                                                                              *
* Call.by: ROWOP                                                               *
* Calls:   ansisup.                                                            *
*******************************************************************************

          subroutine prtmlt

          integer      i,getch,key
          external     getch

$include:'rowop.blk'
$include:'design.blk'


          call clrbot
          write(*,'(20x,a\)')' Printer READY (y/n) ?'
          key=getch()
          if (key.eq.27) goto 100
          if (lowcase(key).eq.121) then
            call clrbot
            write(*,'(a\)')' Printing operation . . .'
            open(2,file='lpt1:')
          else
            call cls
            open(2,file='con:')
          endif
          write(2,'(1h ,a)')' '
          write(2,'(1h ,a)')'****************************************'
```

```
          if (qinvrt) then
            if (cuse.eq.0) then
            write(2,'(1h ,a)')' Row  Operation  on  1/[Q(s)] -> [K(s)]'
            else
            write(2,'(1h ,a)')' Column Operation on 1/[Q(s)] -> [L(s)]'
            endif
          else
            if (cuse.eq.0) then
            write(2,'(1h ,a)')' Row   Operation   on  [Q(s)] -> [L(s)]'
            else
            write(2,'(1h ,a)')' Column Operation  on  [Q(s)] -> [K(s)]'
            endif
          endif
          write(2,'(1h ,a)')'****************************************'
          write(2,'(1h ,a)')' '
          write(2,'(1h ,a,i2.2,a,i2.2,a)')'Element reduced: (',
     +                                    ri,',',rj,')  '
          if (cuse.eq.0) then
            write(2,'(1h ,a,i2.2)')'Row  used      :  ',ruse
          else
            write(2,'(1h ,a,i2.2)')'Column  used   :      ',cuse
          endif
          write(2,'(1h ,a)')' '
          write(2,'(1h ,a)')
     +      ' Multiplier:     Numerator      Denominator        Lag   '
          write(2,'(1h ,a)')
     +      ' ===========     ---------      -----------       -----  '
          do 1 i=1,rsord+1
            write(2,'(1h ,a,i2.2\)')'    s^',i-1
            write(2,'(6x,e13.7,\)')rkn(i)
            write(2,'(3x,e13.7,\)')rkd(i)
            if (i.eq.1) then
              write(2,'(2x,e13.7)')rk1
            else
              write(2,'(1h )')
            endif
1         continue
          write(2,'(1h0)')
          write(2,'(1h \)')
100       call clrbot
          write(*,'(a\)')' Press <key> to continue . . . '
          key=getch()
          return
          end

          subroutine prtm12

          integer      i,getch,key
          external     getch

$include:'rowop.blk'
$include:'design.blk'


          call clrbot
          write(*,'(20x,a\)')' Printer READY (y/n) ?'
          key=getch()
          if (key.eq.27) goto 100
          if (lowcase(key).eq.121) then
            call clrbot
```

```
            write(*,'(a\)')' Printing operation . . .'
            open(2,file='lpt1:')
          else
            call cls
            open(2,file='con:')
          endif
          write(2,'(1h ,a)')' '
          write(2,'(1h ,a)')'****************************************'
          if (qinvrt) then
            if (cuse.eq.0) then
            write(2,'(1h ,a)')' Row  Operation  on  1/[Q(s)] -> [K(s)]'
            else
            write(2,'(1h ,a)')' Column Operation on 1/[Q(s)] -> [L(s)]'
            endif
          else
            if (cuse.eq.0) then
            write(2,'(1h ,a)')' Row   Operation   on  [Q(s)] -> [L(s)]'
            else
            write(2,'(1h ,a)')' Column Operation  on  [Q(s)] -> [K(s)]'
            endif
          endif
          write(2,'(1h ,a)')'****************************************'
          write(2,'(1h ,a)')' '
          if (cuse.eq.0) then
            write(2,'(1h ,a,i2.2,a)')
     +     ' Multiply  Row  [',ruse,']  by Function'
          else
            write(2,'(1h ,a,i2.2,a,a)')
     +     ' Multiply Column [',cuse,'] by Function'
          endif
          write(2,'(1h ,a)')' '
          write(2,'(1h ,a)')
     +      ' Multiplier:     Numerator      Denominator        Lag   '
          write(2,'(1h ,a)')
     +      ' ===========     =========      ===========       =====  '
          do 1 i=1,rsord+1
            write(2,'(1h ,a,i2.2\)')'    s^',i-1
            write(2,'(6x,e13.7,\)')rkn(i)
            write(2,'(3x,e13.7,\)')rkd(i)
            if (i.eq.1) then
              write(2,'(2x,e13.7)')rkl
            else
              write(2,'(1h )')
            endif
1         continue
          write(2,'(1h0)')
          write(2,'(1h \)')
100       call clrbot
          write(*,'(a\)')' Press <key> to continue . . . '
          key=getch()
          return
          end


$include:'add.blk'
*****************************************************************************
* PRTNYQ * PRINT PROCESS FREQUENCY RESPONSE                                  *
*****************************************************************************
* Descr:   Loads the process (INA Plot)response given by INA.PLT  in         *
*          the TFDIR, temporary file directory. The expected number          *
*          of samples is given by NOOFW. The ORDER specifies the order       *
```

```
*          nxn of the matrix. The response is then printed.                  *
*                                                                            *
* Call.by: DIAGON                                                           *
* Calls:   ansisup,ERROR.                                                   *
*****************************************************************************

          subroutine prtnyq

          integer      iocheck
          complex      kw
          character*32 filen,user
          character*8  name
          integer      i,getch,key,j,k,l,m,readch
          logical      kbhit,escape
          external     getch,kbhit,readch


$include:'design.blk'
$include:'desdta.blk'


          if (.not.qexist) then
            call error('No Nyquist Diagram exists !    ')
            return
          endif
          write(*,'(20x,a\)')' Printer READY (y/n) ?'
          key=getch()
          if (key.eq.27) goto 100
          call disk(filen,tfdir,'NYQUIST ','.~JW')
          open(1,file=filen,form='binary',iostat=iocheck)
          if (lowcase(key).eq.121) then
            call clrbot
            write(*,'(8x,2a\)')'Printing Frequency Response...',
     +      ' Press <ESC> to abort.'
            open(2,file='lpt1:')
          else
            call gotorc(10,10)
            write(*,'(a\)')' (Use CON: for screen output)'
            call gotorc(11,10)
            write(*,'(a\)')' Enter filename [.PRN]:'
            call readln(input,8,escape)
            if (escape) goto 199
            call disk(user,pfdir,input,'.prn')
            if ((input.eq.'con:     ').or.(input.eq.'CON:     '))then
              open(2,file='con:')
              call cls
            else
              open(2,file=user,status='new')
            endif
          endif
          write(2,'(1h ,a)')' '
          if (qinvrt) then
            write(2,'(1h ,a)')'    Process Response: INVERSE'
          else
            write(2,'(1h ,a)')'    Process Response: DIRECT '
          endif
            write(2,'(1h ,a)')'   ==========================='
          write(2,'(1h ,a)')' '
          write(2,'(1h ,2a)')'    Design : ',dsnnme
          if (gfilen.ne.' ')
     +      write(2,'(1h ,2a)')'    [G(s)] : ',gfilen
```

```
      if (kfilen.ne.' ')
     +  write(2,'(1h ,2a)')'    [K(s)] : ',kfilen
      if (lfilen.ne.' ')
     +  write(2,'(1h ,2a)')'    [L(s)] : ',lfilen
      if (ffilen.ne.' ')
     +  write(2,'(1h ,2a)')'    [F(s)] : ',ffilen
      write(2,'(1h ,2a)')'     W(i)  : ',wfilen
      write(2,'(1h ,a)')' '
      write(2,'(1h ,a)')
     +  '  ELEMENT      FREQUENCY          REAL         IMAGINARY'
      write(2,'(1h ,a)')
     +  ' ---------    -----------        ------      -----------'
      write(2,'(1h ,a)')' '
      do 1 i=1,order
        do 1 j=1,order
          write(2,'(1h )')
          write(2,'(1h \)')
          close(1)
          open(1,file=filen,form='binary',iostat=iocheck)
          do 1 k=1,noofw
            if (k.eq.1) then
              write(2,'(a,i2.2,a,i2.2,a\)')'  (',i,',',j,')   '
            else
              write(2,'(13x\)')
            endif
            write(2,'(1h ,1p,e13.6,a\)')w(k),' '
            do 2 l=1,order
              do 2 m=1,order
                if (kbhit()) then
                  if (readch().eq.27) goto 199
                endif
                read(1,err=99)kw
                if ((l.eq.i).and.(m.eq.j)) then
                  write(2,'(1p,e13.6,a\)') real(kw),'  '
                  write(2,'(1p,e13.6)') aimag(kw)
                endif
2             continue
1       continue
        write(2,'(1h )')
        write(2,'(1h \)')
        call clrbot
        write(*,'(25x,a\)')' Press any key to continue...'
        goto 100

99      call clrbot
        write(*,'(2a\)')' *FILE ERROR*',filen
        goto 100

199     call clrbot
        write(*,'(8x,a\)')'USER INTERRUPT: Printout aborted. '

100     key=getch()
        write(2,'(1h )')
        close(1)
        close(2)
        return
        end

$include:'add.blk'
*******************************************************************************
```

```
* PRTPLT * PRINT PLOT                                                          *
*******************************************************************************
* Descr:   Prints the multiplier and qoutient data on a printer.              *
*          If no printer is used displays output on screen.                   *
*                                                                             *
* Calls:   ansisup.                                                           *
*******************************************************************************

          subroutine prtplt

          complex      result
          real         pi
          complex      nw,dw,lw
          logical      kbhit
          integer      i,getch,key,j,k,readch
          external     getch,readch,kbhit

$include:'desdta.blk'
$include:'rowop.blk'

          call clrbot
          write(*,'(20x,a\)')' Printer READY (y/n) ?'
          key=getch()
          if (key.eq.27) goto 100
          if (lowcase(key).eq.121) then
            call clrbot
            write(*,'(8x,2a\)')'Printing multiplier response. ',
     +        'Press <ESC> to abort.'
            open(2,file='lpt1:')
          else
            call cls
            write(*,'(a,3a1\)')'Press <ESC> to abort.',13,10,10
            open(2,file='con:')
          endif
          write(2,'(1h )')
          write(2,'(1h ,2a)')
     +'                        APPROXIMATION                  ',
     +'DATA TO BE FITTED'
          write(2,'(1h ,2a)')
     +'  FREQUENCY          real         imaginary ',
     +'          real         imaginary'
          write(2,'(1h ,2a)')
     +' -----------        ------       -----------',
     +'         ------       -----------'
          write(2,'(1h )')
          do 1 i=1,noofw
            if (kbhit()) then
              if (readch().eq.27) goto 199
            endif
            write(2,'(1h ,e13.7,\)')w(i)
            call calcpw(nw,rkn,rsord,w(i))
            call calcpw(dw,rkd,rsord,w(i))
            if (cabs(dw).le.1.e-8) then
              result=(0,0)
              write(2,'(a30\)') '   *** Division by zero ***     '
            else
              if (lw.ne.0) then
                lw=cmplx(0.,amod( rkl*w(i) , 2 * pi ))
                result=nw/dw*cexp(lw)
              else
                result=nw/dw
```

```
            endif
          write(2,'(2x,1p,e13.6,\)') real(result)
          write(2,'(2x,1p,e13.6,\)') aimag(result)
          endif
          write(2,'(4x,1p,e13.6,\)') real(quotient(i))
          write(2,'(2x,1p,e13.6)') aimag(quotient(i))
1         continue
          write(*,'(1h )')
          write(*,'(1h \)')
          call clrbot
          write(*,'(25x,a\)')'Press any key to continue...'
          goto 100

199       call clrbot
          write(*,'(8x,a\)')'USER INTERRUPT: Printout aborted.'
100       key=getch()
          close(2)
          return
          end


$include:'add.blk'
*******************************************************************************
* PSEUDO * PSEUDO-DIAGONALISATION                                             *
*******************************************************************************
* Descr: This program calculates a K matrix to pseudo diagonalize             *
*        the frequency response of a tranfer function matrix.                 *
*                                                                             *
*        This program is designed to be used in conjunction with              *
*        INACAD.                                                              *
*                                                                             *
*        Operating Instructions                                               *
*        ----------------------                                               *
*                                                                             *
*        Use INACAD to plot the Inverse Nyquist Diagram of the                *
*        process. This will result in a temporary file being written          *
*        to disk : NYQUIST.JW.                                                *
*                                                                             *
*        This file in addition to the following information entered            *
*        at run time will result in a parameter file : PSEUDO.KS              *
*        which can directly be used by INACAD.                                *
*                                                                             *
*        Additional input:                                                    *
*                                                                             *
*          Process order. &                                                   *
*          Number of frequency points to use.                                 *
*                                                                             *
*******************************************************************************
*  Support routines                                                           *
*                                                                             *
*        SREVAL - Calculate eigenvalues and eigenvectors.                     *
*                                                                             *
*        COMINV - Matrix inversion routine.                                   *
*                                                                             *
*******************************************************************************

      program pseudo
      integer n,ierr
      complex gw1(100),u(100),d
      real    a(100),eval(10),evctr(100),aik(100)
```

```
900   format(a\)
      open(1,file='nyquist.jw',form='binary')
      open(2,file='pseudo.ks',status='new')
      write(*,900) ' Enter order:'
      read(*,'(i3)') n
      write(*,900) ' Enter no of frequency points:'
      read(*,'(i3)') nw

      do 7 jr=1,n
        do 8 ij=1,n*n
8         a(ij)=0.0
        do 9 iw=1,nw
          do 10 i=1,n
            do 10 j=1,n
              ij=i+(j-1)*n
10            read(1,err=99) gw1(ij)
          do 11 i=1,n
            do 11 l=i,n
              il=i+(l-1)*n
              do 12 k=1,n
                if (k.eq.jr) goto 12
                ik=i+(k-1)*n
                lk=l+(k-1)*n
                a(il)=a(il)+real(gw1(ik))*real(gw1(lk))+
     +            aimag(gw1(ik))*aimag(gw1(lk))
12            continue
              li=l+(i-1)*n
11        a(li)=a(il)
9       continue
        rewind(1)
        call sreval(a,n,eval,evctr,aik,ierr)
        if (ierr.ne.0) then
          write(*,*) 'Eval routine error'
          stop
        endif
        emn=1.e30
        do 13 i=1,n
          if (eval(i).gt.emn) goto 13
          emn=eval(i)
          imn=i
13      continue
        sum=0.0
        do 14 j=1,n
          jrj=jr+(j-1)*n
          jimn=j+(imn-1)*n
          u(jrj)=cmplx(evctr(jimn),0.0)
14      sum=sum+real(u(jrj))
        if (sum.gt.0) goto 7
        do 19 j=1,n
          jrj=jr+(j-1)*n
19      u(jrj)=-u(jrj)
7     continue
      close(1)
      call cominv(u,n,d)
      write(2,'(3i2)') n,n,1
      do 15 i=1,n
        do 15 j=1,n
15      write(2,'(e13.7)') real(u(i+(j-1)*n))
      do 16 i=1,n*n
16      write(2,'(e13.7)') 1.0
      do 17 i=1,n*n
```

```
17      write(2,'(e13.7)') 0.0
      close(2)
      goto 100
99    write(*,*) 'File access error'
100   continue
      stop
      end


$include:'add.blk'
*******************************************************************************
* READLN * READ LINE INPUT                                                    *
*******************************************************************************
* Descr:   Read a string input of specified length MAX.                       *
*                                                                             *
* Param:   INPUT returned string.                                             *
*          ESCAPE logical true if the escape key was pressed.                 *
*                                                                             *
* Calls:   ansisup.                                                           *
*******************************************************************************

        subroutine readln(input,max,escape)

        character*20 input
        character*2  keych
        integer      readch,key,count
        integer      max
        external     readch
        logical      escape,start
        integer      line(20),i

        escape=.false.
        start=.true.
        call saverc
        count=0
        input='                    '
1       key=readch()
        if (key.eq.27) escape=.true.
        if (key.eq.27) goto 5
        if (key.eq.13) goto 5
        if (key.ge.32) then
          if (start) then
            do 4 k=1,max
4             write(*,'(a\)')' '
            start=.false.
            call loadrc
          endif
          if (count.le.max-1) then
            count=count+1
            keych=char(key)
            write(*,'(al,\)') keych
            line(count)=key
          else
            write(*,'(al,\)')7
         endif
        endif
        if ((count.ne.0).and.(key.eq.8)) then
          count=count-1
          call loadrc
          do 2 k=1,max
```

```
2           write(*,'(a\)')' '
          call loadrc
          do 21 k=1,count
21          write(*,'(a\)')char(line(k))
        endif
        goto 1
5       continue
        do 211 k=1,count
211       input(k:k)=char(line(k))
        return
        end


$include:'add.blk'
*******************************************************************************
* REODKL * RECALCULATE K AND L MATRIX                                         *
*******************************************************************************
* Descr:   The specified row or column operation is performed on              *
*          the required row or column of the K or L matrix. The               *
*          frequency response of the matrix is recalculated and               *
*          written to disk.                                                   *
*                                                                             *
* Call.by: ROWOP                                                              *
* Calls:   ansisup.(DOCOL),(DOROW)                                            *
*******************************************************************************

          subroutine redokl

          logical    exist
          integer    getch,key,i
          external   getch

$include:'rowop.blk'
$include:'design.blk'
$include:'desdta.blk'

          if (qinvrt) then
          if (((kfilen.eq.' ').and.(ruse.ne.0)).or.
     +        ((lfilen.eq.' ').and.(ruse.eq.0))) then
            call error(' No K(s) or L(s) file exists !      ')
            goto 100
          endif
          else
          if (((kfilen.eq.' ').and.(ruse.eq.0)).or.
     +        ((lfilen.eq.' ').and.(ruse.ne.0))) then
            call error(' No K(s) or L(s) file exists !      ')
            goto 100
          endif
          endif
          exist=.true.
          call savetf(tfdir,'backup  ','.~ks',knum,kden,klag,
     +                order,mxsord,exist)
          if (exist) goto 100
          exist=.true.
          call savetf(tfdir,'backup  ','.~ls',lnum,lden,llag,
     +                order,mxsord,exist)
          if (exist) goto 100
          qexist=.false.
          ksaved=.false.
          lsaved=.false.
          call clrtop
```

```
          if (qinvrt) then
            do 1 i=1,rsord+1
1             rkn(i)=-rkn(i)
            if (ruse.ne.0) then
              write(*,'(30x,a\)')'  Updating K(s). . .'
              call docol(rkn,rkd,rkl,knum,kden,klag,
     +                   order,mxsord,ruse,ri)
              call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                    kfilen,tfdir,'.~kw',qexist)
              kexist=.true.
              ksaved=.true.
            else
              write(*,'(30x,a\)')'  Updating L(s). . .'
              call dorow(rkn,rkd,rkl,lnum,lden,llag,
     +                   order,mxsord,cuse,rj)
              call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                    lfilen,tfdir,'.~lw',qexist)
              lexist=.true.
              lsaved=.true.
            endif
            do 2 i=1,rsord+1
2             rkn(i)=-rkn(i)
          else
            write(*,'(8x,a\)')'WARNING: operating on DIRECT Nyquist!'
            if (ruse.ne.0) then
              write(*,'(a\)')'  Updating L(s). . .'
              call dorow(rkn,rkd,rkl,lnum,lden,llag,
     +                   order,mxsord,ri,ruse)
              call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                    lfilen,tfdir,'.~lw',qexist)
              lexist=.true.
              lsaved=.true.
            else
              write(*,'(a\)')'  Updating K(s). . .'
              call docol(rkn,rkd,rkl,knum,kden,klag,
     +                   order,mxsord,rj,cuse)
              call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                    kfilen,tfdir,'.~kw',qexist)
              ksaved=.true.
              kexist=.true.
            endif
          endif
100       return
          end

c         Perform column operation on matrix

          subroutine docol(rkn,rkd,rkl,knum,kden,klag,
     +                     order,mxsord,cchange,cuse)
          real       rkn(*),rkd(*),rkl,knum(*),kden(*),klag(*)
          real       mltn(140),mltd(140),mltl(10)
          integer    order,mxsord,cchange,cuse,i,offst1,offst2

          call mltcol(mltn,mltd,mltl,rkn,rkd,rkl,knum,
     +                kden,klag,order,mxsord,cuse)
          do 1 i=1,order
            offst1=1+(mxsord+1)*((cchange-1)+order*(i-1))
            offst2=1+(mxsord+1)*(i-1)
            call addelm(knum(offst1),kden(offst1),
     +                  klag(cchange+(i-1)*order),
     +                  mltn(offst2),mltd(offst2),mltl(i),
```

```
     +                  knum(offst1),kden(offst1),
     +                  klag(cchange+(i-1)*order),mxsord)
1         continue
          return
          end

c         Perform row operation on matrix

          subroutine dorow(rkn,rkd,rkl,knum,kden,klag,
     +                     order,mxsord,rchange,ruse)
          real       rkn(*),rkd(*),rkl,knum(*),kden(*),klag(*)
          real       mltn(140),mltd(140),mltl(10)
          integer    order,mxsord,rchange,ruse,i,offst1,offst2

          call mltrow(mltn,mltd,mltl,rkn,rkd,rkl,knum,
     +               kden,klag,order,mxsord,ruse)
          do 1 i=1,order
            offst1=1+(mxsord+1)*((i-1)+order*(rchange-1))
            offst2=1+(mxsord+1)*(i-1)
            call addelm(knum(offst1),kden(offst1),
     +                  klag(i+(rchange-1)*order),
     +                  mltn(offst2),mltd(offst2),mltl(i),
     +                  knum(offst1),kden(offst1),
     +                  klag(i+(rchange-1)*order),mxsord)
1         continue
          return
          end

c         Add two elements.

          subroutine addelm(mltn,mltd,mltl,poly1n,poly1d,poly1l,
     +                      poly2n,poly2d,poly2l,mxsord)
          real       mltn(*),mltd(*),mltl,poly1n(*),poly1d(*)
          real       poly1l,poly2n(*),poly2d(*),poly2l,t,temp1(21)
          real       temp2(21)
          integer    mxsord,i

          if ((poly1l.ne.0).or.(poly2l.ne.0)) then
            if (poly1l.ge.poly2l) then
              mltl=poly1l
              call polyxe(temp1,poly2n,poly2l-poly1l,mxsord)
              call multpoly(temp1,temp1,poly1d,mxsord)
              call multpoly(temp2,poly1n,poly2d,mxsord)
            else
              mltl=poly2l
              call polyxe(temp1,poly1n,poly1l-poly2l,mxsord)
              call multpoly(temp1,temp1,poly2d,mxsord)
              call multpoly(temp2,poly2n,poly1d,mxsord)
            endif
          else
            mltl=0.
            call multpoly(temp1,poly1n,poly2d,mxsord)
            call multpoly(temp2,poly2n,poly1d,mxsord)
          endif
          call addpoly(mltn,temp1,temp2,mxsord)
          call multpoly(mltd,poly1d,poly2d,mxsord)
          return
          end
c         Multiply row by function

          subroutine mltrow(mltn,mltd,mltl,rkn,rkd,rkl,knum,
```

```
     +                kden,klag,order,mxsord,rchange)

          real       mltn(*),mltd(*),mltl(*),rkn(*),rkd(*),rkl
          real       knum(*),kden(*),klag(*)
          integer    order,mxsord,rchange,i

          do 1 i=1,order
1           mltl(i)=rkl+klag(i+order*(rchange-1))
          do 2 i=1,order
2           call multpoly(mltn(1+(mxsord+1)*(i-1)),rkn,
     +         knum(1+(mxsord+1)*((i-1)+order*(rchange-1))),mxsord)
          do 3 i=1,order
3           call multpoly(mltd(1+(mxsord+1)*(i-1)),rkd,
     +         kden(1+(mxsord+1)*((i-1)+order*(rchange-1))),mxsord)
          return
          end

c         Multiply column by function

          subroutine mltcol(mltn,mltd,mltl,rkn,rkd,rkl,knum,
     +                kden,klag,order,mxsord,cchange)

          real       mltn(*),mltd(*),mltl(*),rkn(*),rkd(*),rkl
          real       knum(*),kden(*),klag(*)
          integer    order,mxsord,cchange,i

          do 1 i=1,order
1           mltl(i)=klag(cchange+order*(i-1))+rkl
          do 2 i=1,order
2             call multpoly(mltn(1+(mxsord+1)*(i-1)),rkn,
     +       knum(1+(mxsord+1)*((cchange-1)+order*(i-1))),mxsord)
          do 3 i=1,order
3             call multpoly(mltd(1+(mxsord+1)*(i-1)),rkd,
     +       kden(1+(mxsord+1)*((cchange-1)+order*(i-1))),mxsord)
          return
          end

c         Multiply two polynomials

          subroutine multpoly(result,poly1,poly2,mxsord)

          real       result(*),poly1(*),poly2(*),sum
          integer    mxsord,i,j

          do 1 i=mxsord+1,1,-1
            sum=0.
            do 2 j=1,i
2             sum=sum+poly1(j)*poly2(i-j+1)
1         result(i)=sum
          return
          end

c         Add two polynomials

          subroutine addpoly(result,poly1,poly2,mxsord)

          real       result(*),poly1(*),poly2(*)
          integer    mxsord,i,j

          do 1 i=1,mxsord+1
1           result(i)=poly1(i)+poly2(i)
```

```
          return
          end

c         Construct series of e^ts

          subroutine polyxe(result,poly,t,mxsord)
          real    result(*),poly(*),coeff(21),fact,t
          integer mxsord,i
          external fact

          do 1 i=1,mxsord+1
1           coeff(i)=t/fact(i)
          call multpoly(result,poly,coeff,mxsord)
          return
          end

c         Calculate Factorial

          real function fact(x)
          integer x,i
          real product

          product=1.
          if (x.gt.1) then
            do 1 i=2,x
1             product=product*i
          endif
          fact=product
          return
          end


$include:'add.blk'
*******************************************************************************
* REDOWF * RE-CALCULATE TRANSFER MATRICES                                     *
*******************************************************************************
* Descr:   This subroutine is called is the frequency range                   *
*          was altered. The transfer matrices are re-calculated.              *
*                                                                             *
* Call.by: EDITWF                                                             *
* Calls:   CALCTF.                                                            *
*******************************************************************************

        subroutine redowf

$include:'design.blk'
$include:'desdta.blk'

        if (gfilen.ne.' ')
     +    call calctf(w,noofw,gnum,gden,glag,mxsord,order,
     +                gfilen,tfdir,'.~gw',qexist)
        if (kfilen.ne.' ')
     +    call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                kfilen,tfdir,'.~kw',qexist)
        if (lfilen.ne.' ')
     +    call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                lfilen,tfdir,'.~lw',qexist)
        if (ffilen.ne.' ')
     +    call calctf(w,noofw,fnum,fden,flag,mxsord,order,
     +                ffilen,tfdir,'.~fw',qexist)
        return
```

```
        end


$include:'add.blk'
*******************************************************************************
* RCMLT * ROW/COLUMN MULTIPLICATION BY A FACTOR                                *
*******************************************************************************
* Descr:  Multiply a row or column by a function.                              *
*                                                                              *
* Call.by: DIAGON                                                              *
* Calls:   ansisup,SAVETF,CALCTF,KXROW,KXCOL,EDITMLT                           *
*******************************************************************************
          subroutine rcmlt

          integer      i,ruset,cuset,key,j,rit,rjt
          real         mltn(140),mltd(140),mltl(10)
          character*20 input
          integer      getch
          external     getch
          logical      change,exist

$INCLUDE:'MENU.BLK'
$include:'rowop.blk'
$include:'design.blk'
$include:'desdta.blk'

          change=.true.
11        menunb=9
          call getchc(menunb,escape)
          if (escape) goto 111
          if (chcdfl(menunb).eq.0) then
1           call gotorc(4,20)
            write(*,'(a\)')'Select an array element that is to be '
            call gotorc(5,20)
            write(*,'(a\)')'multiplied by the multiplier function.'
            call gotorc(8,20)
            write(*,'(a\)')' Element: ('
            call saverc
            write(*,'(i2.2,a,i2.2,a\)') ri,',',rj,')'
            call loadrc
            call readln(input,5,escape)
            if (input.eq.' ') goto 12
            if (escape) goto 12
            read(input,222,err=199) rit,rjt
            if ((rit.gt.order).or.(rit.le.0)) goto 199
            if ((rjt.gt.order).or.(rjt.le.0)) goto 199
            change=.true.
            ri=rit
            rj=rjt
            goto 1
222         format(bn,2i2)
199          write(*,'(a\)') 7
            goto 1
12          ruse=ri
            cuse=0
2           call gotorc(10,20)
            write(*,'(a\)')'Select row or column multiplication.'
            call gotorc(14,20)
            write(*,'(a\)')'Press key to change and <ESC> to exit.'
            call gotorc(12,20)
              write(*,'(a\)')'Multiply '
```

```
            if (cuse.eq.0) call invers
            write(*,'(a\)')' Row ['
            write(*,'(i2.2,a\)')ruse,'] '
            call normal
            if (ruse.eq.0) call invers
            write(*,'(a\)')' Column ['
            write(*,'(i2.2,a\)')cuse,'] '
            call normal
            if (getch().eq.27) goto 11
            if (ruse.eq.0) then
              ruse=ri
              cuse=0
            else
              cuse=rj
              ruse=0
            endif
            change=.true.
            goto 2
          elseif (chcdfl(menunb).eq.1) then
            call editmlt(1,rsord,rkd,rkn,rkl,9,change)
          elseif (chcdfl(menunb).eq.2) then
            if (change) call getqu2
            change=.false.
            call optim
          elseif (chcdfl(menunb).eq.3) then
            if (change) call getqu2
            change=.false.
            call pltmlt
          elseif (chcdfl(menunb).eq.4) then
            call getscale
          elseif (chcdfl(menunb).eq.5) then
            goto 113
          elseif (chcdfl(menunb).eq.6) then
            call prtml2
          elseif (chcdfl(menunb).eq.7) then
            if (change) call getqu2
            change=.false.
            call prtplt
          endif
          goto 11
113       call gotorc(4,15)
          write(*,'(a\)')'Perform Operation (y/n)?'
          if (getch().ne.121) goto 100
          write(*,'(a\)')' YES'
          if (qinvrt) then
          if (((kfilen.eq.' ').and.(ruse.ne.0)).or.
     +        ((lfilen.eq.' ').and.(ruse.eq.0))) then
            call error(' No K(s) or L(s) file exists !      ')
            goto 100
          endif
          else
          if (((kfilen.eq.' ').and.(ruse.eq.0)).or.
     +        ((lfilen.eq.' ').and.(ruse.ne.0))) then
            call error(' No K(s) or L(s) file exists !      ')
            goto 100
          endif
          endif
          exist=.true.
          call savetf(tfdir,'backup  ','.~ks',knum,kden,klag,
     +                order,mxsord,exist)
          if (exist) goto 100
```

```
          exist=.true.
          call savetf(tfdir,'Backup  ','.~ls',lnum,lden,llag,
     +               order,mxsord,exist)
          if (exist) goto 100
          call clrtop
          ksaved=.false.
          lsaved=.false.
          if (qinvrt) then
            if (ruse.ne.0) then
              write(*,'(30x,a\)')'  Updating K(s). . .'
              call kxcol(rkd,rkn,-rkl,knum,kden,klag,
     +                   order,mxsord,ruse)
              call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                    kfilen,tfdir,'.~kw',qexist)
              kexist=.true.
              ksaved=.true.
            else
              write(*,'(30x,a\)')'  Updating L(s). . .'
              call kxrow(rkd,rkn,-rkl,lnum,lden,llag,
     +                   order,mxsord,cuse)
              call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                    lfilen,tfdir,'.~lw',qexist)
              lexist=.true.
              lsaved=.true.
            endif
          else
            write(*,'(8x,a\)')'WARNING: operating on DIRECT Nyquist!'
            if (ruse.ne.0) then
              write(*,'(a\)')'  Updating L(s). . .'
              call kxrow(rkn,rkd,rkl,lnum,lden,llag,
     +                   order,mxsord,ruse)
              call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                    lfilen,tfdir,'.~lw',qexist)
              lexist=.true.
              lsaved=.true.
            else
              write(*,'(a\)')'  Updating K(s). . .'
              call kxcol(rkn,rkd,rkl,knum,kden,klag,
     +                   order,mxsord,cuse)
              call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                    kfilen,tfdir,'.~kw',qexist)
              kexist=.true.
              ksaved=.true.
            endif
          endif
100       goto 11
111       return
          end


$include:'add.blk'
*******************************************************************************
* ROWOPE * ROW/COLUMN OPERATION                                               *
*******************************************************************************
* Descr:   Here the options for a row/column operation are selected.          *
*                                                                             *
* Call.by: DIAGON                                                             *
* Calls:   ansisup,GETCHC,READLN,EDITMLT,PRTMLT,OPTIM,REDOKL.                 *
*******************************************************************************

          subroutine rowope
```

```
          integer      i,rit,rjt,ruset,cuset
          character*20 input
          integer      getch
          external     getch
          logical      change

$INCLUDE:'MENU.BLK'
$include:'rowop.blk'
$include:'design.blk'
$include:'dim.blk'

          change=.true.
          xlo=0
          xhi=0
          ylo=0
          yhi=0
11        menunb=7
          call getchc(menunb,escape)
          if (escape) goto 111
          if (chcdfl(menunb).eq.0) then
1           call gotorc(4,15)
            write(*,'(a\)')'Select an array element to reduce to zero.'
            call gotorc(6,15)
            write(*,'(a\)')'           Element: ('
            call saverc
            write(*,'(i2.2,a,i2.2,a\)')ri,',',rj,')'
            call loadrc
            call readln(input,5,escape)
            if (input.eq.' ') goto 12
            if (escape) goto 12
            read(input,222,err=99) rit,rjt
            if ((rit.gt.order).or.(rit.le.0)) goto 99
            if ((rjt.gt.order).or.(rjt.le.0)) goto 99
            change=.true.
            ri=rit
            rj=rjt
            cuse=0
            ruse=1
            goto 1
222         format(bn,2i2)
99          write(*,'(a\)') 7
            goto 1
12          call gotorc(8,15)
            write(*,'(a\)')'Select a row or column to use in reduction.'
            call gotorc(12,20)
            write(*,'(a\)')'Press <RTN> to change Row with Column.'
            call gotorc(13,20)
            write(*,'(a\)')'or press <ESC> to exit.'
2           call gotorc(10,25)
            if (cuse.eq.0) then
              write(*,'(a\)')'Use  Row   ['
              call saverc
              write(*,'(i2.2,a\)')ruse,']'
            else
              write(*,'(a\)')'Use Column ['
              call saverc
              write(*,'(i2.2,a\)')cuse,']'
            endif
            call loadrc
            call readln(input,2,escape)
```

```
            if (escape) goto 11
            if (input.eq.' ') then
              if (cuse.eq.0) then
                cuse=ruse
                ruse=0
              else
                ruse=cuse
                cuse=0
              endif
              goto 2
            endif
            read(input,2222,err=999) ruset
            if ((ruset.gt.order).or.(ruset.le.0)) goto 999
            if ((ruset.eq.rj).and.(ruse.eq.0)) goto 999
            if ((ruset.eq.ri).and.(cuse.eq.0)) goto 999
            if (cuse.eq.0) then
              ruse=ruset
            else
              cuse=ruset
              ruse=0
            endif
            change=.true.
            goto 2
999         write(*,'(a\)') 7
            goto 2
2222        format(bn,i2)
          elseif (chcdfl(menunb).eq.1) then
            call editmlt(1,rsord,rkd,rkn,rkl,9,change)
          elseif (chcdfl(menunb).eq.2) then
            if (change) call getquot
            change=.false.
            call optim
          elseif (chcdfl(menunb).eq.3) then
            if (change) call getquot
            change=.false.
            call pltmlt
          elseif (chcdfl(menunb).eq.4) then
            call getscale
          elseif (chcdfl(menunb).eq.5) then
            call gotorc(4,20)
            write(*,'(a\)')'Perform Operation (y/n)?'
            if (getch().eq.121) then
              write(*,'(a\)')' YES'
              call redokl
            endif
          elseif (chcdfl(menunb).eq.6) then
            call prtmlt
          elseif (chcdfl(menunb).eq.7) then
            if (change) call getquot
            change=.false.
            call prtplt
          endif
          goto 11
111       return
          end

$include:'add.blk'
*******************************************************************************
* SAVETF * SAVE TRANSFER FUNCTION MATRIX                                      *
*******************************************************************************
```

```
* Descr:   The transfer function as specified in arrd,arrd,arrl with dimen-  *
*          sion given by order and mxsord (maximum order of s possible) is   *
*          saved.                                                            *
*                                                                            *
* Param:   ASK tells it whether to prompt the user for the save name.        *
*          EXIST is an indication if the data was altered from last save     *
*            if SAVETF was successful EXIST is set to FALSE                  *
*                                                                            *
* Call.by: EDITTF                                                           *
* Calls:   READLN,DISK                                                       *
******************************************************************************

        subroutine savetf(dir,name,ext,arrn,arrd,arrl,
     +                    order,mxsord,exist)

        character*8  name
        character*20 dir,input
        character*4  ext
        real         arrn(*),arrd(*),arrl(*)
        logical      exist,escape,found
        character*32 filen
        integer      getch,key,order,mxsord,sorder,iocheck
        external     getch

1       call disk(filen,dir,name,ext)
        open(1,file=filen,status='new')
        call dimtf(order,mxsord,sorder,arrn,arrd)
        write(1,'(3i2)',err=2)order,order,sorder+1
        do 3 i=1,order
          do 3 j=1,order
            do 3 k=1,sorder+1
              ijk=k+(mxsord+1)*((j-1)+order*(i-1))
3             write(1,'(e13.7)',err=2)arrn(ijk)
        do 4 i=1,order
          do 4 j=1,order
            do 4 k=1,sorder+1
              ijk=k+(mxsord+1)*((j-1)+order*(i-1))
4             write(1,'(e13.7)',err=2)arrd(ijk)
        do 5 i=1,order*order
5         write(1,'(e13.7)',err=2)arrl(i)
        close(1,iostat=iocheck)
        if (iocheck.ne.0) goto 2
        exist=.false.
        goto 100
2       call error('Disk write error!              ')
100     return
        end


$include:'add.blk'
******************************************************************************
* SAVEWF * SAVE FREQUENCY FILE                                               *
******************************************************************************
* Descr:   Save frequency sample table with specified name.                  *
*                                                                            *
* Param:   DIR,NAME,EXT - filename specifications.                           *
*          ARR - array containg frequency points.                            *
*          NOOFW - number of points used.                                    *
*          EXIST - Set false if successfully saved.                          *
*                                                                            *
* Call.by: EDITWF                                                           *
```

```
* Calls:   DISK,ERROR                                                          *
*******************************************************************************

        subroutine savewf(dir,name,ext,arr,noofw,exist)

        logical      exist
        character*8  name
        character*20 dir,input
        character*4  ext
        real         arr(*)
        character*32 filen
        integer      getch,key,noofw,iocheck
        external     getch

1       call disk(filen,dir,name,ext)
        open(1,file=filen,status='new',iostat=iocheck)
        write(1,'(i3)',err=2)noofw
        do 3 i=1,noofw
3         write(1,'(e13.7)',err=2)arr(i)
        close(1,iostat=iocheck)
        if (iocheck.ne.0) goto 2
        exist=.false.
        goto 100
2       call error('Disk write error!             ')
100     return
        end


$include:'add.blk'
*******************************************************************************
* SETPFD * SET PARAMETER FILE DIRECTORY                                        *
*******************************************************************************
* Descr:   Change the parameter and temporary file directories.                *
*                                                                              *
* Calls:   ansisup.                                                            *
*******************************************************************************

        subroutine setpfd

        character*20 input
        logical      escape

$INCLUDE:'DESIGN.BLK'

        call gotorc(4,20)
        write(*,'(a,\)')'Select data file directories, '
        call gotorc(5,20)
        write(*,'(a,\)')'or enter '':'' for none.'
        call gotorc(15,20)
        write(*,'(a,\)')'Press <ESC> to exit.'
        call gotorc(7,20)
        write(*,'(2a,\)')'Parameter file directory: ',pfdir
        call gotorc(9,20)
        write(*,'(2a,\)')'Temporary file directory: ',tfdir
        call gotorc(7,20)
        write(*,'(a,\)')'Parameter file directory: '
        call readln(input,20,escape)
        if(escape) goto 100
        if(input.ne.' ') pfdir=input
        if(input.eq.':') pfdir=' '
        call gotorc(9,20)
```

```
        write(*,'(a,\)')'Temporary file directory: '
        call readln(input,20,escape)
        if(escape) goto 100
        if(input.ne.' ') tfdir=input
        if (input.eq.':') tfdir=' '
100     return
        end


$include:'add.blk'
*******************************************************************************
* SHWPAR * SHOW DESIGN PARAMETERS                                             *
*******************************************************************************
* Descr:   Displays current design parameter settings.                        *
*          Also displays if parameters have not yet been updated.             *
*                                                                             *
* Param:   WAIT logical true if key must be pressed to continue.              *
*                                                                             *
* Calls:   ansisup.                                                           *
*******************************************************************************

          subroutine shwpar(wait)

          character*20 input
          logical      wait
          integer      key,readch
          external     readch

$INCLUDE:'DESIGN.BLK'

          call gotorc(3,20)
          write(*,'(a,\)')' DESIGN PROGRAM STATUS '
          call gotorc(5,15)
          call normal
          write(*,'(a\)')' Current Design Name      : '
          write(*,'(2a,\)')' ',dsnnme
          call gotorc(7,15)
          write(*,'(a,\)')' Process Matrix Order    : '
          write(*,'(I3,\)')order
          call gotorc(9,15)
          write(*,'(a,\)')' Frequency Range File    : '
          write(*,'(3a,\)')' ',wfilen,'[.WS ]'
          if (wexist) write(*,'(a\)')' SAVE!'
          call gotorc(11,15)
          write(*,'(a,\)')' Plant Matrix G(s)       : '
          write(*,'(3a,\)')' ',gfilen,'[.GS ]'
          if (gexist) write(*,'(a\)')' SAVE!'
          call gotorc(13,15)
          write(*,'(a,\)')' Controller Matrix K(s)  : '
          write(*,'(3a,\)')' ',kfilen,'[.KS ]'
          if (kexist) write(*,'(a\)')' SAVE!'
          call gotorc(15,15)
          write(*,'(a,\)')' Controller Matrix L(s)  : '
          write(*,'(3a,\)')' ',lfilen,'[.LS ]'
          if (lexist) write(*,'(a\)')' SAVE!'
          call gotorc(17,15)
          write(*,'(a,\)')' Feedback Matrix F(s)    : '
          write(*,'(3a,\)')' ',ffilen,'[.FS ]'
          if (fexist) write(*,'(a\)')' SAVE!'
          call gotorc(19,15)
          write(*,'(a,\)')' Parameter File Directory : '
```

```
          write(*,'(2a,\)')' ',pfdir
          call gotorc(21,15)
          write(*,'(a,\)')' Temporary File Directory : '
          write(*,'(2a,\)')' ',tfdir
          if (wait) then
            call clrtop
        write(*,'(15x,a\)')' Enter New  Design  Name :'
            call readln(input,19,escape)
            if (input.ne.' ') dsnnme=input
          endif
          return
          end


$include:'add.blk'
*******************************************************************************
* SWOP   *  SWOP ROW OR COLUMNS OF INA                                         *
*******************************************************************************
* Descr:   Here the rows or columns of the INA can be swopped.                 *
*          Swopping rows of the INA changes rows    of K                       *
*          Swopping columns of the INA changes columns of L                   *
*                                                                             *
* Call.by: DIAGON.                                                            *
* Calls:   ansisup,SAVETF,CALCTF,(SWOPROW),(SWOPCOL)                          *
*******************************************************************************

          subroutine swop

          integer      rit,rjt
          character*20 input
          logical      change,exist
          real         temp
          integer      key,getch
          external     getch

$INCLUDE:'MENU.BLK'
$include:'rowop.blk'
$include:'design.blk'
$include:'desdta.blk'

11      menunb=10
        call getchc(menunb,escape)
        if (escape) goto 111
        if (chcdfl(menunb).eq.0) then
1         call gotorc(4,20)
          write(*,'(a\)')'Select rows to be swopped.'
          call gotorc(6,20)
          write(*,'(a\)')' Rows '
          call saverc
          write(*,'(i2.2,a,i2.2\)') ri,',',ruse
          call loadrc
          call readln(input,5,escape)
          if (input.eq.' ') goto 11
          if (escape) goto 11
          read(input,222,err=1) rit,rjt
          if ((rit.gt.order).or.(rit.le.0)) goto 1
          if ((rjt.gt.order).or.(rjt.le.0)) goto 1
          change=.true.
          ri=rit
          ruse=rjt
          cuse=0
```

```
          goto 1
222       format(bn,2i2)
        elseif (chcdfl(menunb).eq.1) then
2         call gotorc(4,20)
          write(*,'(a\)')'Select columns to be swopped.'
          call gotorc(6,20)
          write(*,'(a\)')' Columns '
          call saverc
          write(*,'(i2.2,a,i2.2\)') rj,',',cuse
          call loadrc
          call readln(input,5,escape)
          if (input.eq.' ') goto 11
          if (escape) goto 11
          read(input,222,err=2) rit,rjt
          if ((rit.gt.order).or.(rit.le.0)) goto 2
          if ((rjt.gt.order).or.(rjt.le.0)) goto 2
          change=.true.
          rj=rit
          cuse=rjt
          ruse=0
          goto 2
        elseif (chcdfl(menunb).eq.2) then
          goto 113
        elseif (chcdfl(menunb).eq.3) then
          call clrtop
          write(*,'(20x,a\)')' Printer READY (y/n) ?'
          key=getch()
          if (key.eq.27) goto 11
          if (lowcase(key).eq.121) then
            call clrbot
            write(*,'(a\)')' Printing operation . . .'
            open(2,file='lpt1:')
          else
            call cls
            open(2,file='con:')
          endif
          write(2,'(1h ,a)')' '
          write(2,'(1h ,a)')'****************************************** '
          if (qinvrt) then
            if (cuse.eq.0) then
            write(2,'(1h ,a)')' Row  operation  on  1/[Q(s)] -> [K(s)]'
            else
            write(2,'(1h ,a)')' Column operation on 1/[Q(s)] -> [L(s)]'
            endif
          else
            if (cuse.eq.0) then
            write(2,'(1h ,a)')' Row  operation   on   [Q(s)] -> [L(s)]'
            else
            write(2,'(1h ,a)')' Column  operation  on [Q(s)] -> [K(s)]'
            endif
          endif
          write(2,'(1h ,a)')'****************************************** '
          write(2,'(1h )')
          if (cuse.eq.0) then
            write(2,'(1h ,a,i2.2,a,i2.2)')
     +       'Row   : ',ri,' swopped   with  row : ',ruse
          else
            write(2,'(1h ,a,i2.2,a,i2.2)')
     +       'Column: ',rj,' swopped with  column: ',cuse
          endif
          write(2,'(1h0,a)')' '
```

```
          write(2,'(1h ,a\)')' '
          call clrbot
          write(*,'(20x,a\)')' Press <key> to continue . . .'
          key=getch()
          close(2)
          goto 11
        endif
113     call gotorc(4,15)
        write(*,'(25x,a\)')'Perform Operation (y/n)?'
        if (lowcase(getch()).ne.121) goto 100
        write(*,'(a\)')' YES'
        if (qinvrt) then
          if (((kfilen.eq.' ').and.(ruse.ne.0)).or.
     +        ((lfilen.eq.' ').and.(ruse.eq.0))) then
            call error(' No K(s) or L(s) file exists !      ')
            goto 100
          endif
          else
          if (((kfilen.eq.' ').and.(ruse.eq.0)).or.
     +        ((lfilen.eq.' ').and.(ruse.ne.0))) then
            call error(' No K(s) or L(s) file exists !      ')
            goto 100
          endif
          endif
          exist=.true.
          call savetf(tfdir,'backup  ','.~ks',knum,kden,klag,
     +                order,mxsord,exist)
          if (exist) goto 100
          exist=.true.
          call savetf(tfdir,'backup  ','.~ls',lnum,lden,llag,
     +                order,mxsord,exist)
          if (exist) goto 100
          ksaved=.false.
          lsaved=.false.
          call clrtop
          if (qinvrt) then
            if (ruse.ne.0) then
              write(*,'(30x,a\)')'  Updating K(s). . .'
              call swopcol(knum,kden,klag,order,mxsord,ri,ruse)
              call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                    kfilen,tfdir,'.~kw',qexist)
              kexist=.true.
              ksaved=.true.
            else
              write(*,'(30x,a\)')'  Updating L(s). . .'
              call swoprow(lnum,lden,llag,order,mxsord,rj,cuse)
              call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                    lfilen,tfdir,'.~lw',qexist)
              lexist=.true.
              lsaved=.true.
            endif
          else
            write(*,'(8x,a\)')'WARNING: operating on DIRECT Nyquist!'
            if (ruse.ne.0) then
              write(*,'(a\)')'  Updating L(s). . .'
              call swoprow(lnum,lden,llag,order,mxsord,ri,ruse)
              call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                    lfilen,tfdir,'.~lw',qexist)
              lexist=.true.
              lsaved=.true.
            else
```

```
              write(*,'(a\)')'  Updating K(s). . .'
              call swopcol(knum,kden,klag,order,mxsord,rj,cuse)
              call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                    kfilen,tfdir,'.~kw',qexist)
              kexist=.true.
              ksaved=.true.
            endif
          endif
100       goto 11
111       return
          end

c         Swop columns

          subroutine swopcol(knum,kden,klag,order,mxsord,ri,ruse)
          real       knum(*),kden(*),klag(*),temp
          integer    order,mxsord,ri,ruse,i,j

          do 1 i=1,order
            temp=klag(ruse+order*(i-1))
            klag(ruse+order*(i-1))=
     +        klag(ri+order*(i-1))
            klag(ri+order*(i-1))=temp
            do 1 j=1,mxsord+1
              temp=knum(j+(mxsord+1)*((ruse-1)+order*(i-1)))
              knum(j+(mxsord+1)*((ruse-1)+order*(i-1)))=
     +          knum(j+(mxsord+1)*((ri-1)+order*(i-1)))
              knum(j+(mxsord+1)*((ri-1)+order*(i-1)))=temp
              temp=kden(j+(mxsord+1)*((ruse-1)+order*(i-1)))
              kden(j+(mxsord+1)*((ruse-1)+order*(i-1)))=
     +          kden(j+(mxsord+1)*((ri-1)+order*(i-1)))
1             kden(j+(mxsord+1)*((ri-1)+order*(i-1)))=temp
          return
          end

c         Swop rows

          subroutine swoprow(lnum,lden,llag,order,mxsord,rj,cuse)
          real       lnum(*),lden(*),llag(*),temp
          integer    order,mxsord,rj,cuse,i,j

          do 1 i=1,order
            temp=llag(i+order*(cuse-1))
            llag(i+order*(cuse-1))=
     +        llag(i+order*(rj-1))
            llag(i+order*(rj-1))=temp
            do 1 j=1,mxsord+1
              temp=lnum(j+(mxsord+1)*((i-1)+order*(cuse-1)))
              lnum(j+(mxsord+1)*((i-1)+order*(cuse-1)))=
     +          lnum(j+(mxsord+1)*((i-1)+order*(rj-1)))
              lnum(j+(mxsord+1)*((i-1)+order*(rj-1)))=temp
              temp=lden(j+(mxsord+1)*((i-1)+order*(cuse-1)))
              lden(j+(mxsord+1)*((i-1)+order*(cuse-1)))=
     +          lden(j+(mxsord+1)*((i-1)+order*(rj-1)))
1             lden(j+(mxsord+1)*((i-1)+order*(rj-1)))=temp
          return
          end


$include:'add.blk'
*******************************************************************************
```

```
* SYSCAL * SYSTEM CALL                                                          *
*********************************************************************************
* Descr:   Execute child process specified.                                     *
*                                                                               *
* Param:   COMMAND - String containing DOS command.                             *
*                                                                               *
* Call.by:                                                                      *
* Calls:   SYSTEM - routine in CEXEC.LIB                                        *
*********************************************************************************

c       Construct interface specification to C library routine

        interface to integer*2 function system [c]
     +          (string[reference])
        character*1 string
        end

        subroutine syscal(command)

        integer*2       i,system
        character*30[c] command

        i = system(command)
        if (i.eq.-1) call error('No COMMAND.COM file found!    ')
        end

c       Perform DOS command selection

        subroutine dos

        integer         key,getch
        external        getch
        character*30[c] command
        logical         success

        call gotorc(4,1)
        write(*,'(a\)')'Enter DOS command or press <RTN> to exit.'
        call gotorc(6,1)
1       write(*,'(a\)')'C>'
        read(*,'(a)') command
        if (command.eq.' ') goto 100
        call syscal(command)
        write(*,'(3al\)')' ',13,10
        goto 1
100     return          .
        end

c       Delete all temporary files that were created.

        subroutine deltmp
        character*30[c] command

        command='erase *.~??'
        call syscal(command)
        return
        end


$include:'add.blk'
*********************************************************************************
* UNDOCH * UNDO ALTERATION    OF K OR L MATRIX                                  *
```

```
*************************************************************************
* Descr:   Loads a saved copy of the K and L matrix from disk.          *
*          Subroutine is called to reverse the effect of a row          *
*          operation.                                                   *
*                                                                       *
* Call.by: ROWOP                                                        *
* Calls:   ansisup,CALCTF,LOADTF.                                       *
*************************************************************************

          subroutine undoch

          logical      exist
          character*8  name
          integer      getch,key
          external     getch

$include:'rowop.blk'
$include:'design.blk'
$include:'desdta.blk'

          if (kfilen.eq.' ') ksaved=.false.
          if (lfilen.eq.' ') lsaved=.false.
          if ((.not.ksaved).and.(.not.lsaved)) then
            call error('No change pending !                ')
            goto 100
          endif
          call gotorc(4,20)
          if (lsaved.and.ksaved) then
            write(*,'(a\)')'Undo Change to K(s) and L(s) matrix (y/n)?'
          elseif (lsaved) then
            write(*,'(a\)')'Undo Change to L(s) matrix (y/n)?'
          else
            write(*,'(a\)')'Undo Change to K(s) matrix (y/n)?'
          endif
          if (getch().eq.121) then
            write(*,'(a\)')' YES'
            exist=.false.
            name='backup  '
            if (ksaved) then
              call loadtf(tfdir,name,'.~ks',knum,kden,klag,
     +                order,mxsord,exist)
              call calctf(w,noofw,knum,kden,klag,mxsord,order,
     +                kfilen,tfdir,'.~kw',qexist)
            endif
            if (lsaved) then
              call loadtf(tfdir,name,'.~ls',lnum,lden,llag,
     +                order,mxsord,exist)
              call calctf(w,noofw,lnum,lden,llag,mxsord,order,
     +                   lfilen,tfdir,'.~lw',qexist)
            endif
            ksaved=.false.
            lsaved=.false.
          endif
100       return
          end
```

1.5 MENU TREE STRUCTURE

The menu tree structure of the INACAD program is constructed as follows:

1.1 Dos Command

Run a DOS program or execute DOS command.

1.2 Help

Display help message.

1.3 Design Status

Displays current design program status and allows entry of design description.

1.4 Process Parameters

1.4.1 File Directories

Edit file directories.

1.4.2 Process Order

Edit process order.

1.4.3 Frequency Range

1.4.3.1 Load Frequency Data

Load frequency data from disk.

1.4.3.2 Edit Frequency Data

Edit the frequency type& range.

1.4.3.3 Save Frequency Data

Save frequency data to disk.

1.4.4 Plant matrix G(s)

1.4.4.1 Load Transfer Fn

Load data file from disk.

1.4.4.2 Edit Transfer Fn

Edit matrix data.

1.4.4.3 Save Transfer Fn

Save data file to disk.

1.4.4.4 Print Transfer Fn

Print matrix on printer.

1.4.4.5 Remove Parameter

Remove parameter from design.

1.4.5 Controller mx K(s)

1.4.5.1 Load Transfer Fn

Load data file from disk.

1.4.5.2 Edit Transfer Fn

Edit matrix data.

1.4.5.3 Save Transfer Fn

Save data file to disk.

1.4.5.4 Print Transfer Fn

Print matrix on printer.

1.4.5.5 Remove Parameter

Remove parameter from design.

1.4.6 Controller mx L(s)

1.4.6.1 Load Transfer Fn

Load data file from disk.

1.4.6.2 Edit Transfer Fn

Edit matrix data.

1.4.6.3 Save Transfer Fn

Save data file to disk.

1.4.6.4 Print Transfer Fn

Print matrix on printer.

1.4.6.5 Remove Parameter

Remove parameter from design.

1.4.7 Feedback mx F(s)

1.4.7.1 Load Transfer Fn

Load data file from disk.

1.4.7.2 Edit Transfer Fn

Edit matrix data.

1.4.7.3 Save Transfer Fn

Save data file to disk.

1.4.7.4 Print Transfer Fn

Print matrix on printer.

1.4.7.5 Remove Parameter

Remove parameter from design.

1.5 Diagonalisation

1.5.1 Nyquist Setup

Enter options for Nyquist diagram.
i.e. scale, inverse/direct, Gershgorin/
Ostrowski circles, Row/Column dominance.

1.5.2 Plot Nyquist

Display diagram on the screen.

1.5.3 Row/Col. Operation

1.5.3.1 Operation Options

Specify element to eliminate, and element to use.

1.5.3.2 Edit Multiplier

Enter multiplier function for the elimination.

1.5.3.3 Optimize Multiplier

Optimize multiplier function for best elimination result.

1.5.3.4 Plot Multiplier

Display multiplier function and quotient on the screen.

1.5.3.5 Adjust Scale

Specify the scale of the plot.

1.5.3.6 Perform Operation

Calculate new K or L matrix.

1.5.3.7 Print Operation

Print operation on printer.

1.5.3.8 Print Plot

Print multiplier and quotient response on printer.

1.5.4 Swop Rows/Columns

1.5.4.1 Swop Rows

Select which rows to swop.

1.5.4.2 Swop Columns

Select which Columns to swop.

1.5.4.3 Perform Operation

Calculate new K or L matrix.

1.5.4.4 Print Operation

Print operation on printer.

1.5.5 Multiply Row/Col.

1.5.5.1 Operation Options

Specify element to multiply by the function.

1.5.5.2 Edit Multiplier

Enter function to use for the multiplication.

1.5.5.3 Optimize Multiplier

Optimize function so that result is close to unity.

1.5.5.4 Plot Multiplier

Display function and quotient on the screen.

1.5.5.5 Adjust Scale

Specify the scale of the plot.

1.5.5.6 Perform Operation

Calculate new K or L matrix.

1.5.5.7 Print Operation

Print operation on printer.

1.5.5.8 Print Plot

Print multiplier and quotient response on printer.

1.5.6 Undo Operation

Restore K or L matrix.

1.5.7 Print Response

Print process response on printer.

1.6 Clear Design

Clear design specifications.

1.7 Quit Program

Terminate INACAD and return to DOS.

APPENDIX D SOFTWARE REFERENCE

2. POLE ASSIGNMENT ROUTINES

2.1 SUBROUTINE OVERVIEW

calcabk	Calculate matrix (A-BK)
calcf	Calculate feedback vector f
calck	Calculate feedback matrix K
calcm	Calculate solution matrix M
calcr	Calculate closed-loop coefficient vector r
control	Check controllability
faddev	Perform FADDEEV algorithm
getdata	Read input data from file
multpoly	Multiply polynomials
mxbyvec	Multiply matrix by vector
mximlt	Multiply matrices
mxinv	Invert matrix
mxmult	Multiply matrices
poleval	Find matrix eigenvalues
print	Print results
pacad	Main program
sume	Sum coefficients e for FADDEEV algorithm

2.1.1 Implementation of BORRIE method

Included in the listing are the following routines, which can be used to apply the controllable companion form algorithm instead of the FADDEEV method or as a check.

To implement note that $T=M^T$ and T^{-1} is calculated by *calct*.

calcac	Calculate open-loop characteristic coeff.
calct	Calculate matrix T^{-1} by BORRIE method
vecbymx	Multiply vector by matrix

2.2 SUBROUTINE INDEX

calcabk(n,a,b,k,t) 314
calcac(n,a,trinv,tr,tm,tm2,ac) 326
calcf(n,m,r,f) 315
calck(n,q,f,k) 315
calcm(n,g,bb,m) 315
calcr(n,pole,e,rv,r) 314
calct(n,p,a,tm,tm2,trinv) 325
control(n,a,u,q,t,tm,tm2,p,bb,det) 316

faddev(n,a,b,e,p) 317
getdata(N,A,b,pole,q) 319
multpoly(mxsord,poly1,poly2) 323
mxbyvec(n,a,q,t) 320
mximlt(n,a,b,i,p) 318
mxinv(n,c,b) 320
mxmult(n,a,b,p) 319
pacad 312
poleval(n,pole,poly) 323
print(N,det,g,E,m,p,r,k,f,rv,id) 324
sume(n,e,p,b,i) 318
vecbymx(n,v,a,b) 326

2.4 SOURCE LISTING

```
*****************************************************************************
* PACAD  * Pole Assignment Computer Aided Design                             *
*****************************************************************************
* Descr: This program applies the dyadic approach to the design of          *
*        a controller for a multivariable control system by pole            *
*        assignment.                                                        *
*                                                                           *
*          System description:                                              *
*                                                                           *
*           dx                              A is nxn                        *
*          ---- = A x(t) + B u(t)          B is nxn max.                   *
*           dt                                                              *
*                                                                           *
*          choose Q for pole assignment          Q is a real n-vector      *
*             "   POLE i.e. closed loop poles  POLE is a complex n-vector   *
*                                                                           *
* Input File Format:                                                        *
*                                                                           *
*         Line Col:12345678901234567890                                     *
*           1     >Process Order max 25                                     *
*           2     >2                                                        *
*           3     >System matrix A e.g. Identity matrix                     *
*           4     >1.0       0.0                                            *
*           5     >0.0       1.0                                            *
*           6     >System matrix B e.g. Identity matrix                     *
*           7     >1.0       0.0                                            *
*           8     >0.0       1.0                                            *
*           9     >Required Roots   Real part Imaginary part                *
*          10     >-1.0      0.0       -1.0      0.0                        *
*          11     >Q vector                                                 *
*          12     >1.0       0.0                                            *
*                                                                           *
*       Format used I2, E10.4                                               *
*                                                                           *
*       Note - In this examples lines 1,3,6,9 & 11 will be skipped          *
*                                                                           *
*              and may contain any comment.                                 *
*                                                                           *
*****************************************************************************
```

```
      program pacad

c     input data

      integer order,pass
      character*64 ifnam,ofnam,comment
      real*8 a(25,25),b(25,25),q(25)
      complex*16 pole(25)

c     calculation data

      real*8 det
      real*8 g(25,25,25),e(25),p(25,25),m(25,25),k(25,25),f(25),r(25)
      real*8 t(25),bb(25),id(25,25),rv(25),tm(25,25),tm2(25,25)

c     G used for for faddeev method
c     E coeff. of characteristic function for closed loop system
c     P temporary matrix used by faddeev and retains M before inversion
c     M matrix identified with G(s)
c     K controller matrix
c     F feedback control vector
c     DET determinant used to test controllability
c     R solution vector
c     T temporary vector used by CONTROL
c     TM,TM2 temporary matrices used by CONTROL
c     BB vector containing b=Uq
c     ORDER process order max. 25

      write(*,911) '#Multivariable Pole Assignment Design Program'
      write(*,911) '#(C) Copyright R.Venzke 1988'
      write(*,911) '#'


      write(*,900)
900   format('#Data filename : '\)
      read(*,911) ifnam
      write(*,901)
901   format('#Output filename : '\)
      read(*,911) ofnam
910   format(a\)
911   format(a)

      open(3,file=ifnam)
      open(4,file=ofnam,status='new')


      write(*,910) '#Reading, '
      read(3,'(a)') comment
      read(3,500) order
500   format(bn,i7)

c     read input data from file
      call getdata(order,a,b,pole,q)

      write(*,910) 'processing, control ,'
c     check controllablity
      call control(order,a,b,q,t,tm,tm2,p,bb,det)

      write(*,910) 'faddeev, '
c     get matrix G from faddeev algorithm
      call faddev(order,a,g,e,p)
```

```
      write(*,910) 'Matrix M, '
c     calculate matrix identified with G(s)
      call calcm(order,g,bb,m)

      write(*,910) 'Matrix M, '
c     get a copy of M before inversion
      call calcm(order,g,bb,p)

c     calculate inverse of m to solve simultaneous eqs.
      call mxinv(n,det,m)
      if (det.eq.0) then
        write(*,*) ' ERROR:Inversion of M not possible!'
      endif

      write(*,910) 'vector R, '
c     calculate solution vector R
      call calcr(order,pole,e,rv,r)

      write(*,910) 'Vector F, '
c     calculate vector F
      call calcf(order,m,r,f)

      write(*,910) 'check inversion ,'
c     check inversion of m
      call mxmult(order,m,p,id)

      write(*,910) 'Matrix K, '
c     calculate controller matrix K
      call calck(order,q,f,k)

      write(*,910) 'Matrix A-BK, '
c     calculate closed loop A matrix
      call calcabk(order,a,b,k,t)

      write(*,910) 'writing, '

c     print results to output file
      call print(order,det,g,e,m,p,r,k,f,rv,id)

      write(*,911) 'done.'
      close(3)
      close(4)
      end


*****************************************************************************
* CALCABK* Calculate A-BK matrix                                             *
*****************************************************************************
* Descr:  Calculate the closed loop A matrix after state feedback.           *
*         The results are written to a file ABK.DAT sothat                   *
*         this matrix can be used to check where the resulting system        *
*         poles are placed after pole assignment.                            *
*                                                                            *
* Param:  N - order of matrix                                                *
*         A - A matrix                                                       *
*         B - B matrix                                                       *
*         K - K matrix                                                       *
*         T - temporary nxn matrix                                           *
*                                                                            *
*****************************************************************************
```

```
      subroutine calcabk(n,a,b,k,t)
      integer n,i,j,l
      real*8 a(n,n),b(n,n),k(n,n),t(n,n)

      do 1 i=1,n
        do 1 j=1,n
          t(i,j)=a(i,j)
          do 1 l=1,n
1           t(i,j)=t(i,j)-b(i,l)*k(l,j)

      open(8,file='abk.dat',status='new')
      write(8,'(i2)') n
      do 2 i=1,n
2       write(8,'(25e10.4)')  (t(i,j),j=1,n)
      close(8)
      return
      end

*****************************************************************************
* CALCR  * Calculate resulting r-vector                                      *
*****************************************************************************
* Descr:   Subtract the open loop characteristic coefficient vector          *
*          from the desired closed loop characteristic equation and store    *
*          result in the solution vector-r.                                  *
*                                                                            *
* Param:   N    - system order                                               *
*          POLE - array containing desired pole positions                    *
*          E    - open loop characteristic equation from Faddeev algorithm   *
*          RV   - desired characteristic eqn for closed loop system          *
*          R    - solution vector                                            *
*                                                                            *
* Call.by: PACAD                                                             *
* Calls  : POLEVAL                                                           *
*****************************************************************************

      subroutine calcr(n,pole,e,rv,r)
      integer n
      real*8 e(n),r(n),t,rv(n)
      complex*16 pole(n)

      call poleval(n,pole,rv)
      do 1 i=1,n
1       r(i)=rv(i)-e(i)
      return
      end

*****************************************************************************
* CALCM  * CALCULATE M MATRIX                                                *
*****************************************************************************
* Descr:   Here the m(s)-matrix containing the product of the faddeev        *
*          equivalent single input system g(s) and the b=Bq vector is        *
*          calculated.                                                       *
*                                                                            *
* Param:   N - system order                                                  *
*          G - Faddeev g(s) nxnxn matrix                                     *
*          BB - b-vector                                                     *
*          M - resulting nxn matrix                                          *
*                                                                            *
* Call.by: PACAD                                                             *
* Calls:   none.                                                             *
```

```
**************************************************************************

          subroutine calcm(n,g,bb,m)
          integer n,i,j,k
          real*8 g(n,n,n),bb(n),m(n,n)

          do 1 i=1,n
            do 1 j=1,n
              m(n-i+1,j)=0.
              do 1 k=1,n
1               m(n-i+1,j)=m(n-i+1,j)+g(i,j,k)*bb(k)
          return
          end

**************************************************************************
* CALCF  * CALCULATE FEEDBACK VECTOR                                     *
**************************************************************************
* Descr:   This routine solves the system of linear equations:           *
*                                                                        *
*                 M x = r   for x.                                       *
*                                                                        *
* Param:  N - system order                                               *
*         M - nxn matrix                                                 *
*         R - resulting vector                                           *
*         F - solution x-vector                                          *
*                                                                        *
* Call.by:PACAD                                                          *
* Calls:  MXINV,MXBYVEC.                                                 *
**************************************************************************

          subroutine calcf(n,m,r,f)
          integer n
          real*8 m(n,n),r(n),f(n)

          call mxbyvec(n,m,r,f)
          return
          end

**************************************************************************
* CALCK  * CALCULATE FEEDBACK MATRIX K                                   *
**************************************************************************
* Descr:   Here the state feedback matrix K is computed by calculating   *
*          the product of vector q and feedback vector f.                *
*                                                                        *
* Param:   N - system order                                              *
*          Q - q-vector                                                  *
*          F - feedback vector                                           *
*          K - resulting feedback matrix                                 *
*                                                                        *
* Call.by: PACAD                                                         *
* Calls:   ansisup,                                                      *
**************************************************************************

          subroutine calck(n,q,f,k)
          integer n
          real*8 q(n),f(n),k(n,n)

          do 1 i=1,n
             do 1 j=1,n
1              k(i,j)=q(i)*f(j)
          return
```

```
      end

*************************************************************************
* CONTROL* Check controllability                                        *
*************************************************************************
* Descr:   This routine checks whether the choice of vector Q           *
*          for the pole assignment technique gives the necessary        *
*          controllability of the feedback system, by inverting         *
*          matrix W and finding its determinant.                        *
*                                                                       *
*                          2         n                                  *
*          P = [ b, Ab, A b, .... A b  ] , where b= Bq                  *
*                                                                       *
* Param:   DET - determinant of P                                       *
*          A   - system matrix nxn                                      *
*          N   - process order                                          *
*          U   - input matrix B                                         *
*          Q   - design n-vector q                                      *
*          P   - controllabilty matrix nxn                              *
*          T   - temporary n-vector                                     *
*          TM  - temporary nxn matrix                                   *
*          TM2 - temporary nxn matrix                                   *
*          BB  - resulting n-vector b                                   *
*                                                                       *
*************************************************************************

      subroutine control(n,a,u,q,t,tm,tm2,p,bb,det)
      integer n
      real*8 a(n,n),u(n,n),q(n),p(n,n),t(n),det
      real*8 bb(n),tm(n,n),tm2(n,n)

c     t=u*q
      call mxbyvec(n,u,q,t)

c     p(1)=t
      do 1 i=1,n
1       p(i,1)=t(i)

      if (n.eq.1) goto 100

c     put matrix A into temporary storage
      do 2 i=1,n
        do 2 j=1,n
2         tm(i,j)=a(i,j)

c     calculate A*t
      call mxmult(n,a,t,bb)

c     store in p(2)  => p(2)=At
      do 3 i=1,n
3       p(i,2)=bb(i)

      do 4 k=3,n

c       calculate A * A * ....
        call mxmult(n,tm,a,tm2)
        do 5 i=1,n
          do 5 j=1,n
5           tm(i,j)=tm2(i,j)

        call mxbyvec(n,tm,t,bb)
```

```
            do 6 i=1,n
6             p(i,k)=bb(i)

4         continue
100   continue

c     get bb vector from p(i,1)
      do 7 i=1,n
7       bb(i)=p(i,1)

c     Check inversion of matrix p
      call mxinv(n,det,p)

      if (det.eq.0.0) then
        write(*,*) ','
        write(*,*) 'WARNING: Choice of q-vector may be invalid'
        write(*,*) ' '
      endif
      return
      end

*****************************************************************************
* FADDEV * Perform Faddev algorithm                                         *
*****************************************************************************
* Descr: FADDEV evaluates the matrices B and vector e given matrix A        *
*                                                                           *
*                           n-1      n-2                                    *
*                          s  * B + s   *B  +....+ sB   + B         B  = I   *
*              -1               1        2          n-1   n      ,  1       *
*         (sI-A)    =  ------------------------------------------------     *
*                                    n       n-1                            *
*                          |sI-A| = s + e * s   +...+ e * s + e            *
*                                      1           n-1      n              *
*                                                                           *
*                                                                           *
*               N order of matrix A(nxn)                                    *
*               P matrix nxn temporary storage                              *
*                                                                           *
* Call.by: PACAD                                                            *
* Calls:   MXIMLT,SUME,TRACE                                                *
*****************************************************************************

          subroutine faddev(n,a,b,e,p)
          integer  n,i,j,k
          real*8   a(n,n),b(n,n,n),e(n),p(n,n),trace,max
          external trace

          do 1 j=1,n
            do 1 k=1,n
              if (j.ne.k) then
                b(1,j,k)=0.0
              else
                b(1,j,k)=1.0
              endif
1         continue
          do 2 i=1,n
            if (i.ne.1) call sume(n,e(n-i+2),p,b,i)
            call mximlt(n,a,b,i,p)
            e(n-i+1)=-trace(n,p)/i
2         continue
c         Add new coefficient e to AB should sum to zero
```

```
      max=0
      do 21 j=1,n
        do 21 k=1,n
          if (j.eq.k) then
            max=dmax1(max,p(j,k)+e(1))
          elseif (i.ne.1) then
            max=dmax1(max,p(j,k))
          endif
21    continue
      write(*,'(a,e12.4,a\)') 'Check= ',max,'=zero '
      return
      end

c     Do summation for characteristic eqn

      subroutine sume(n,e,p,b,i)
      integer n,j,k,i
      real*8  p(n,n),e,b(n,n,n)

c     Add new coefficient e to AB

      do 21 j=1,n
        do 21 k=1,n
          if (j.eq.k) then
            b(i,j,k)=p(j,k)+e
          elseif (i.ne.1) then
            b(i,j,k)=p(j,k)
          endif
21    continue
      return
      end

c     Calculate the trace of a matrix

      real*8 function trace(n,b)
      integer n
      real*8    b(n,n),t

      t=0.
      do 2 i=1,n
2       t=t+b(i,i)
      trace=t
      return
      end


c     Multiply matrix A by matrix B[i]

      subroutine mximlt(n,a,b,i,p)
      integer n,i,j,l,k
      real*8  a(n,n),b(n,n,n),p(n,n)

      do 10 l=1,n
        do 10 j=1,n
          p(l,j)=0.
          do 10 k=1,n
10          p(l,j)=p(l,j)+a(l,k)*b(i,k,j)
      return
      end

c     Multiply matrix A by matrix B[i]
```

```
c         used to check inversion routine

          subroutine mxmult(n,a,b,p)
          integer n,j,l,k
          real*8  a(n,n),b(n,n),p(n,n)

          do 10 l=1,n
            do 10 j=1,n
              p(l,j)=0.
              do 11 k=1,n
11              p(l,j)=p(l,j)+a(l,k)*b(k,j)
c             write(*,*) ' ',p(l,j)
10        continue
          return
          end
```

```
*******************************************************************************
*GETDATA* Get Program Data                                                    *
*******************************************************************************
* Descr: Reads ORDER,matrix A, matrix B, POLE positions and vector Q          *
*        from file #4. The read data is then written to file #3 to            *
*        check the input data.                                                *
*                                                                             *
* Param: N process order                                                      *
*        A,B nxn matrix                                                       *
*        POLE n-vector containing pole positions                              *
*        Q   design n-vector                                                  *
*        File #4 - INPUT                                                      *
*        File #3 - OUTPUT                                                     *
*                                                                             *
* Call.by: PACAD                                                              *
* Calls: none                                                                 *
*******************************************************************************

      SUBROUTINE getdata(N,A,b,pole,q)
      INTEGER N
      REAL*8 A(n,n),b(n,n),q(n)
      COMPLEX*16 pole(n)
      INTEGER I,J

501   format(bn,25e10.4)
502   format(1p,50e12.4)
503   format(bn,50e10.4)
      write(4,400) N
400   format(' Process Order = ',i7)

      write(4,'(a)') ' Matrix A : '
      read(3,'(a)') comment
      do 99 i = 1,N
         read(3,501) (A(i,j), j=1,N)
         write(4,'(1p,25e12.4)') (A(i,j), j = 1,N)
99    continue
      write(4,'(a)') ' '

      write(4,'(a)') ' Matrix B : '
      read(3,'(a)') comment
      do 199 i = 1,N
         read(3,501) (b(i,j), j=1,N)
         write(4,'(1p,25e12.4)') (b(i,j), j = 1,N)
199    continue
      write(4,'(a)') ' '
```

```
      write(4,'(a)') ' Design Poles : '
      read(3,'(a)') comment
      read(3,503) (pole(i), i =1,n)
      write(4,502) (pole(i), i = 1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Design Q vector : '
      read(3,'(a)') comment
      read(3,501) (q(i), i =1,n)
      write(4,'(1p,25e12.4)') (q(i), i = 1,N)
      write(4,'(a)') ' ************************************************'
      write(4,'(a)') ' '
      RETURN
      end

*****************************************************************************
*MXBYVEC * Matrix by vector multiplication                                  *
*****************************************************************************
* Descr: MXBYVEC mutliplies matrix A by a vector Q and stores the           *
*        result in vector T                                                 *
*                                                                           *
* Param: A is nxn matrix                                                    *
*        Q is n vector                                                      *
*        T is resulting n vector                                            *
*        N is the order n                                                   *
* Call.by:                                                                  *
* Calls:  none                                                              *
*****************************************************************************

          subroutine mxbyvec(n,a,q,t)
          integer n
          real*8 a(n,n),q(n),t(n)

          do 1 i=1,n
            t(i)=0.
            do 1 j=1,n
1             t(i)=t(i)+a(i,j)*q(j)
          return
          end

*****************************************************************************
* MXINV  * REAL MATRIX INVERSION                                            *
*****************************************************************************
* Descr:   Inverts a real matrix using the Gauss-Jordan                     *
*          ellimination with complete pivoting.                             *
*                                                                           *
*          Note: It is an adaptation of a complex matrix inversion routine  *
*                                                                           *
* Param:   B =         matrix                                               *
*          N = Dimension of B where B is nxn                                *
*          C = determinant of B                                             *
*                                                                           *
* Call.by: CONTROL,CALCM                                                    *
* Calls:   none.                                                            *
*****************************************************************************

          SUBROUTINE mxinv(n,c,b)

          INTEGER    IR(30),IC(30)
C                    IR,IC are the pivot point of the matrix
```

```
      real*8     a(900),D,TEMP,HOLD
      real*8     b(n,n),c
      INTEGER    I,N,J,K,L,II,JJ,III,M,IJ,IIJ,IT,KI,KJ,LP

      do 101 i=1,n
      do 101 j=1,n
101     a((i-1)*n+j)=b(i,j)
      DO 10 I=1,N
        IR(I)=I
        IC(I)=I
10    CONTINUE
      D=1.0d0
      M=-N
      DO 70 I=1,N
        M=M+N+1
        IF (I.EQ.N) GOTO 40
          BIG=dABS(A(M))
          II=I
          JJ=I
          DO 20 K=I,N
            LP=(I-2)*N+K
            DO 20 L=I,N
              LP=LP+N
              IF (dABS(A(LP)).LE.BIG) GOTO 20
                BIG=dABS(A(LP))
                II=K
                JJ=L
20          CONTINUE
22          IF (II.EQ.I) GOTO 30
              IJ=I-N
              IIJ=II-N
              DO 25 J=1,N
                IJ=IJ+N
                IIJ=IIJ+N
                TEMP=A(IJ)
                A(IJ)=A(IIJ)
                A(IIJ)=TEMP
25            CONTINUE
              IT=IR(I)
              IR(I)=IR(II)
              IR(II)=IT
              D=-D
30          IF (JJ.EQ.I) GOTO 40
              IJ=(I-1)*N
              IIJ=(JJ-1)*N
              DO 35 J=1,N
                IJ=IJ+1
                IIJ=IIJ+1
                TEMP=A(IJ)
                A(IJ)=A(IIJ)
                A(IIJ)=TEMP
35            CONTINUE
              IT=IC(JJ)
              IC(JJ)=IC(I)
              IC(I)=IT
              D=-D
40        if (dabs(d*a(m)).lt.1.d-150) then
            c=0.0d0
            return
          endif
          D=D*A(M)
```

```
            TEMP=A(M)
            A(M)=1.0d0
            IJ=I-N
            DO 50 J=1,N
              IJ=IJ+N
              A(IJ)=A(IJ)/TEMP
50          CONTINUE
            KI=(I-1)*N
            DO 70 K=1,N
              KI=KI+1
              IF (K.EQ.I) GOTO 70
                TEMP=A(KI)
                A(KI)=0.0d0
                KJ=K-N
                IJ=I-N
                DO 60 J=1,N
                  KJ=KJ+N
                  IJ=IJ+N
                  HOLD=-TEMP*A(IJ)
                  A(KJ)=A(KJ)+HOLD
60              CONTINUE
70          CONTINUE
72          CONTINUE
            DO 80 I=1,N
              IF (IR(I).EQ.I) GOTO 80
              K=IR(I)
              JI=(I-1)*N
              JK=(K-1)*N
              DO 85 J=1,N
                JI=JI+1
                JK=JK+1
                TEMP=A(JI)
                A(JI)=A(JK)
                A(JK)=TEMP
85            CONTINUE
              IR(I)=IR(K)
              IR(K)=K
            GOTO 72
80          CONTINUE
82          CONTINUE
            DO 90 I=1,N
              IF(IC(I).EQ.I) GOTO 90
              K=IC(I)
              IJ=I-N
              KJ=K-N
              DO 75 J=1,N
                IJ=IJ+N
                KJ=KJ+N
                TEMP=A(IJ)
                A(IJ)=A(KJ)
                A(KJ)=TEMP
75            CONTINUE
              IC(I)=IC(K)
              IC(K)=K
              GOTO 82
90          CONTINUE
            do 100 i=1,n
            do 100 j=1,n
100           b(i,j)=a((i-1)*n+j)
            c=d
            RETURN
```

```
          END

*****************************************************************************
*POLEVAL * Polynomial Evaluation                                            *
*****************************************************************************
* Descr: POLEVAL evaluates a polynomial from given roots.                   *
*                                                                           *
* Param: N no. of roots                                                     *
*        POLE array containing roots                                        *
*        POLY array containing subscripts of polynomial                     *
*        eqn= s^(n-1)*poly(n)+....s*poly(2)+poly(1)                         *
*                                                                           *
*****************************************************************************

          subroutine poleval(n,pole,poly)
          integer    n,i,j
          real*8     poly(n)
          complex*16 pole(n),polyt,polyc(25)

          do 2 i=1,n
2           polyc(i)=(0.0,0.0)
          polyc(1)=(1.0,0.0)
          do 1 i=1,n
            polyt=-pole(i)
            call multpoly(n,polyc,polyt)
1         continue
          do 3 i=1,n
            poly(i)=dreal(polyc(i))
            if (cabs(polyc(i)).ne.abs(poly(i))) then
              write(*,'(a\)') ' Coeff= '
              write(*,'(1p,50e12.4)')polyc(i)
              write(*,*)' WARNING:Complex poles are not conjugates.'
            endif
3         continue

          return
          end

*****************************************************************************
*MULTPOLY* Multiply Polynomials                                             *
*****************************************************************************
* Descr: Multiply polynomial of form:                                       *
*             s^(n-1)*poly(n)+....s*poly(2)+poly(1)     * ( s + poly2 )     *
*                                                                           *
* Param: MXSORD maximum order of s                                          *
*        POLY1 array containing polynomials of s                            *
*        POLY2 -pole to add                                                 *
*        POLY1  resulting polynomial of order MXSORD                        *
*                                                                           *
*****************************************************************************

          subroutine multpoly(mxsord,poly1,poly2)
          complex*16 poly1(*),poly2
          integer    mxsord,i,j

          do 2 i=mxsord,2,-1
2           poly1(i)=POLY1(I-1)+poly1(i)*poly2
          poly1(1)=poly1(1)*poly2
          return
          end
```

```
*************************************************************************
* PRINT  * Print data                                                   *
*************************************************************************
* Descr:   Prints the intermediate results of the pole assignment       *
*          technique and the resulting controller.                      *
*                                                                       *
* Call.by: PACAD                                                        *
* Calls:   none                                                         *
*************************************************************************

      subroutine print(N,det,g,E,m,p,r,k,f,rv,id)
      INTEGER N,l
      REAL*8 g(n,n,n),e(n),m(n,n),p(n,n),r(n),rv(n),k(n,n),det,f(n)
      real*8 id(n,n)

      write(4,'(a)') ' '
      write(4,'(a\)') ' Controllability determinant :'
      write(4,'(1p,25e12.4)') det
      write(4,'(a)') ' '

      write(4,'(a)') ' Open Loop Characteristic polynomial : '
      write(4,'(1p,25e12.4)') (e(i), i=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Faddev Matrix'
      do 1 i=1,n
        write(4,'(a,i2)') ' s^',n-i
        do 1 j=1,n
1         write(4,'(1p,25e12.4)') (g(i,j,l), l=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Closed Loop Characteristic polynomial : '
      write(4,'(1p,25e12.4)') (rv(i), i=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Matrix M identified with G(s) : '
      do 2 j=1,n
2       write(4,'(1p,25e12.4)') (p(j,l), l=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Inverse of M : '
      do 3 j=1,n
3       write(4,'(1p,25e12.4)') (m(j,l), l=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Identity of M * 1/M : '
      do 7 j=1,n
7       write(4,'(1p,25e12.4)') (id(j,l), l=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Resulting vector : '
      write(4,'(1p,25e12.4)') (r(i), i=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Feedback vector : '
      write(4,'(1p,25e12.4)') (f(l), l=1,N)
      write(4,'(a)') ' '

      write(4,'(a)') ' Resulting controller : '
      do 4 j=1,n
4       write(4,'(1p,25e12.4)') (k(j,l), l=1,N)
```

```
      write(4,'(a)') ' '
      write(4,'(a)') ' ************************************************'
      open(8,file='kmatrix.dat',status='new')
      do 5 j=1,n
5       write(8,'(1p,25e12.4)') (k(j,l), l=1,N)
      close(8)
      RETURN
      END


*************************************************************************
* CALCT  * CALCULATE T MATRIX                                           *
*************************************************************************
* Descr:   This routine calculates the transfomation matrix T^-1 for    *
*                                                                       *
*                x̄ = T z̄   from the controllabilty matrix P^-1          *
*                                                                       *
* Param:   N - system order                                             *
*          P - controllabilty matrix inverse                            *
*          A - system A matrix                                          *
*          TM,TM2 - temporary nxn matrices                              *
*          TRINV - Resulting inverse T matrix                           *
*                                                                       *
* Call.by: BORRIE                                                       *
*                                                                       *
* Calls:   MXMULT,VECBYMX,                                              *
*************************************************************************

      subroutine calct(n,p,a,tm,tm2,trinv)
      integer n,i,j,k
      real*8    p(n,n),a(n,n),trinv(n,n),pm(25),pmnew(25)
      real*8    tm(n,n),tm2(n,n)

      do 4 i=1,n
        do 4 j=1,n
          if (i.eq.j) then
            tm(i,j)=1.0
          else
            tm(i,j)=0.0
          endif
4     continue
      do 2 i=1,n
        pm(i)=p(n,i)
2       trinv(1,i)=pm(i)

      do 1 k=2,n
        call mxmult(n,a,tm,tm2)
        do 5 i=1,n
          do 5 j=1,n
5           tm(i,j)=tm2(i,j)
        call vecbymx(n,pm,tm,pmnew)
        do 1 i=1,n
          trinv(k,i)=pmnew(i)
1     continue

      write(4,'(a)') 'Tinv matrix:'
      do 11 i=1,n
11      write(4,600) (trinv(i,j), j=1,n)

600   format(1p,25e12.4)
      return
```

```
      end

c     Calculate vector by matrix multiplication

      subroutine vecbymx(n,v,a,b)
      integer n,i,j
      real*8 v(n),a(n,n),b(n)

      do 1 i=1,n
        b(i)=0.0
        do 1 j=1,n
1         b(i)=b(i)+v(j)*a(j,i)
      return
      end


*************************************************************************
* CALCAC * CALCULATE AC VECTOR                                          *
*************************************************************************
* Descr:   This routine calculates the closed-loop character-           *
*          istic equation from T and T-1 from the controllable          *
*          companion method.                                            *
*                                                                       *
* Param:   N - system order                                             *
*          TRINV - controllabilty matrix inverse                        *
*          TR    - controllabiliy matrix                                *
*          A - system A matrix                                          *
*          TM,TM2 - temporary nxn matrices                              *
*          AC    - Resulting vector                                     *
*                                                                       *
* Call.by: BORRIE                                                       *
*
* Calls:   MXMULT,VECBYMX,                                              *
*************************************************************************
      subroutine calcac(n,a,trinv,tr,tm,tm2,ac)
      integer n
      real*8 a(n,n),trinv(n,n),tr(n,n),tm(n,n)
      real*8 tm2(n,n),ac(n)

      call mxmult(n,trinv,a,tm)
      call mxmult(n,tm,tr,tm2)
      do 2 i=1,n
2       ac(i)=-tm2(n,i)
      write(4,'(a)') 'Ac vector:'
      write(4,600) (ac(j), j=1,n)
600   format(1p,25e12.4)

      return
      end
```

APPENDIX D SOFTWARE REFERENCE