PERSONAL COMPUTER BASED PROCESS CONTROL SIMULATOR SOFTWARE

A DISSERTATION

submitted in partial fulfilment of the
requirements for the award of the degree
of
MASTER OF ENGINEERING
in
ELECTRICAL ENGINEERING
(With Specialization in Systems Engineering and Operations Research)

By

VIJAY PANDE

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-247 667 (INDIA)

APRIL, 1990

D E D I C A T I N G

TO

MY

P A R E N T S

CANDIDATE'S DECLARATION

I hereby, certify that, the work done which is being presented in the dissertation entitled, "**Personal Computer Based Process Control Simulator Software**", in partial fulfilment of the requirements for the award of the degree of Master of Engineering in Electrical Engineering with specialization in "**Systems Engineering and Operations Research**, submitted in the Electrical Engineering Department, University of Roorkee, Roorkee (INDIA), is an authentic record of my own work carried out for a period of about six months from september, 1989 to March 1990, under the supervision of Prof. M.K.VASANTHA, Electrical Engineering Department, University of Roorkee, India.

The matter embodied in this dissertation has not been submitted by me for the award of any other degree.

Pande

Date: 6-4-90

[VIJAY PANDE]

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

M. Vasanthi 6.4.1990

[SHRI M.K.VASANTHA]
Professor
Electrical Engineering Deptt.
University of Roorkee,
Roorkee - 247 667,
INDIA.

ACKNOWLEDGEMENT

I am highly indebted to Shri M.K.VASANTHA, Professor, Electrical Engineering Department, for his valuable guidance, unending inspiration, constant encouragement and unceasing enthusiasm. I am also thankfull to Dr.D.K.MEHRA, Professor, Electronics and Computer Department, and Dr.B.MOHANTY, Lecturer, Chemical Engineering Department, for extending their support in the development of the software.

I extend my gratitude to Dr. J.D.SHARMA, Professor, Electrical Engineering Department for providing me with the ready to use software packages for my dissertation.

I am also thankfull to Dr.R.N.MISHRA, Reader, Electrical Engineering Department, for providing me with their suggestion specially in the area of system identification.

I extend my gratitude to Dr.R.B.SAXENA, Professor and Head, Electrical Engineering Department, Dr.A.K.PANT, and Shri RAVISH CHANDRA, Professor, Electrical Engineering Department for providing various facilities during the course of development of this work.

Also, I take this apportunity to express my sincere sense of obligation to my family member, friends and staff member who backed my interest and efforts by giving useful suggestions and possible support.

Pande

ROORKEE

VIJAY PANDE

ABSTRACT

Control system is an area which has got a great deal of mathematical complexity, and the physical significance is difficult to analyse. This has led to a wide gap in the understanding of the subject. To bridge up the gap in our understanding, an attempt has been made to come up with computer aided learning package of controls and process control under the heading of **personal computer based process control simulator software.**

The wide spectrum of control studies has been touched in this dissertation with the aid of readymate package, many new algorithms and existing subroutines. The area which has been touched in this dissertation is system identification, transfer function simulation, block diagram reductions using state space and FFT application program. A study package provided to us by Danes incorporate - U.S.A. for process control has also been interfaced.

Since each modules has some pitfall and limitations. Hence approach has also been presented in which direct development of new subroutine, subprograms is possible by invoking respective compilers and editor from the menu itself. This had made possible for this software to grow with a great deal of flexibility in near future.

CONTENTS

Contd. Contents

###

INTRODUCTION

Control systems is an area, which has traditionally been associated with a maze of mathematical complexity, where the physical significance has not been so forthcoming. This has led to a wide gap in our understanding of the subject and limited our creativity. In this dissertation the concept of a software learning package has been proposed, which incorporates powerful visual aids; this aids learning, and at the same time makes the mathematical computations and steps transparent to the the user.

The process control simulator (PCS) analyses the system by studying its input-output behaviour, and as a result outputs the identify of the system (SYSTEM IDENTIFICATION). Currently, this module limits itself to the identification of single-input, single output (SISO) systems. Subsequent to this analyses, the system thus identified can be verified for its I'-O' behaviour by feeding arbitrary input to this system.

In control system, block diagrams are an effective graphical representation of the systems is terms of their transfer-function (TF). Conventional system reduction techniques involve the use of Mason's gain formula, whose limitation stems from the fact, that at any arbitrary point in the block diagram, it is too tedious and difficult to get overall transfer function of the system specifically when

hidden forward paths and loops are involved.

The PCS overcomes this problem by computing the state-space matrix at any point, for a given input and thus being able to compute the TF of the system at any point where output is desired quickly and effectively. Furthermore, the PC also incorporates the flexibility to specify any point in the block diagram where input can be applied. This module (STATE SPACE ANALYSIS) is currently limited to SISO systems and does not offer any graphical aids.

Furthermore, the package offers a host of activities associated with transfer-function viz., simulating the output for a specified transfer function and input, both continuous and discrete, obtaining the inverse Laplacian of the given function and translation of T.F. to state-space. This is supported by the TRANSFER-FUNCTION SIMULATION module which also offers good graphical support.

This package also includes a PROCESS SIMULATION module, which is a ready to use package manufactured by Danes incorporates and has been added to enhance the versatility of the software. It basically allows the user to study the various kinds of controllers like PID, PI, etc.

The PCS is only an innovative concept, and therefore by no means complete. There are plenty of directions in which the package can be made to grow e.g. extending the system analysis from SISO to MIMO. Keeping this in mind,

this software has been interfaced with a standard mathematical package MATLAB: which allows user's to think in terms of matrices, naturally, and thus improve upon the existing package.

This PCS also make use of one of the key area of thrust in system studies that is Fast Fourier Transforms and it is used in evaluating autocorrelation, cross correlation and convolution function. It is also used in graphical study module of sampling and reconstruction of signal.

This dissertation report is categorised as follows. The Chapter one deals with the system identification which includes two methods for system identification. The Chapter two deal with transfer function simulation, which contains transfer function to state space conversion, output simulation for any input and inverse laplacian. Chapter three present a new approach for block diagram reduction based on state space approach. Chapter four deals with the fast fourier transform applications to reconstruction of signal, finding auto-correlation and cross-correlation functions. Chapter five include various modules which is very common but has been given a menu driven approach. They are matrix operation and least square approximation. Matrix operation has been given a menudriven approach which includes determinant, inverse, eigen value, eigen vector and gauss elimination programs. Similarly least square approximation

also includes five methods of getting a formulae of the interpolating curve. Chapter six gives a very brief description about the ready packages like process control package developed by Danes incorporate and Matlab of Maths work incorporates. Chapter seven gives the software structure and its strategic details. Then after conclusion about the work and subsequent to it is appendices which includes results and detailed source code program which is written in PASCAL.

ooooo

CHAPTER - I

SYSTEM IDENTIFICATION

SYSTEM IDENTIFICATION

1.1 INTRODUCTION:

The problem of system identification has attracted considerable attention because of a large number of applications in diverse field like chemical processes, biomedical systems, socioeconomic systems, transportation, ecology, communication, electric power system, hydrology aeronautics, etc. The model consist basically of mathematical equations which can be used for understanding the behaviour of the system, and where ever prediction and control is involved. Based upon the output (measurable effect) and input (measurable cause) to system (black box), system's equation could be determined. In system identification, we are concerned with the system models from record of system operation. The problem can be represented diagrammatically as shown in Fig. 1.1.

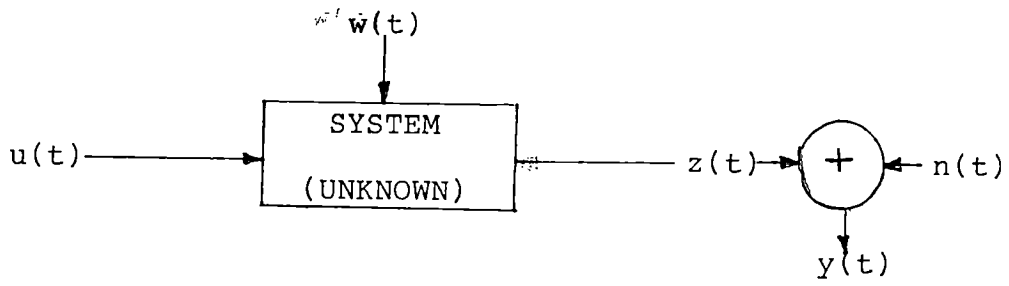


Fig.1

where,

u(t) is known inputvector of dimension n
z(t) is outputvector of dimension p
w(t) is the input disturbance vector
n(t) is the observation noise vector
y(t) is the measured output vector of dimension p.

Thus, the problem of system identification is the determination of the system model from record of u(t) and y(t). And as far as control system is concerned, many control system design algorithms and filtering algorithms in the text books assume knowledge of the parameters of the signal process model. But in practice a-priori-knowledge of those parameters alongwith the order of the system is rarely available and that is why there is a need to identify the model of the system. Completely by experimentation/simulation.

1.2 REVIEW:

The problems associated with the system identification are:

a) Determination of the order of the linear model.

b) Selection of a suitable criterion for determining the "accuracy" of the model.

c) Designing on input-signal which will maximize the accuracy of the estimates of the parameters of the model.

Many authors had worked on the identification, a brief outline of their work has been presented. For stationary stochastic time - series an autoregressive moving average (ARMA) model is frequently used, since it is the minimum parameter linear model of such time series. The book of N.K.Sinha and B.Kuszta [1] is probably the most complete book on the identification of stationary and

nonstationary systems. Durbin [2] has treated the identifying the auto-regressive (AR) parameter of an ARMA model given the moving average (MA) parameter (and vice versa). Lee and Gersch [3] also achieved the results for the problem of estimating AR parameter of a mixed ARMA model of given order assuming a knowledge of MA parameters. The major draw back with all the above existing methods was that, the order of the system comes by the assumptions which in general does not give the optimal order of the system. Mehra [4] has presented a method for identifying the state variable model of the gaussian process which can be executed in recurssive manner. His method is computationally convinient for estimating AR parameter of an ARMA model but is complex when dealing with MA parameters. Further Graupse, Krause and Moore gave a method of identifying ARMA parameter out of AR model [5], which infact was the only method to give both order of the system and estimated parameters but in this method model under consideration was not of the type of used by earlier investigators.

1.3 METHOD PRESENTED:

Here in this dissertation two sub models for identification are presented so that the program can be used as tool box for the future generation.

1.3.1 Method-1 [6]

LEAST SQUARE APPROXIMATION METHOD:

In this method the order of the system is assumed to be known a priory.

Consider a single input - single output model in discrete time, transfer function H(Z) (to be identified) of order n defined by equation (SI-1).

$$H(Z) = \frac{a_o + a_1 z^{-1} + a_2 z^{-2} + ... + a_n z^{-n}}{1 + b_1 z^{-1} + b_2 z^{-2} + ... + bnz^{-n}} \quad .. \text{(SI-1)}$$

where

$$H(Z) = \frac{y(z)}{u(z)}$$

and $y(z)$= discrete output response of the system.

and $u(z)$= discrete input to the system.

The eq.(SI-1) can be written as

$$y(K) = -b_1 y(K-1) - b_2 y(K-2) .... -b_n y(K-n)$$

$$+a_o u(K) + a_1 u(K-1).. + a_n u(K-n) + e(K) \quad .. \text{(SI-2)}$$

Here e(K) is a random variable which takes into account the uncertainty or noise in the model. We assume that [e(K)] is

a sequence of independent zero mean random variables having the same distribution.

Equation (SI-2) can be written in a matrix forms

$$\begin{bmatrix} y(K+1) \\ y(K+2) \\ \vdots \\ y(K+N) \end{bmatrix} = \begin{bmatrix} -y(K) & \cdots & -y(K-n+1) & u(K+1) & \cdots & u(K-n+1) \\ \vdots & & \vdots & \vdots & & \vdots \\ -y(K+N-1) & & & u(K+N) & & \end{bmatrix}_{N \times 2n+1} * \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \\ a_o \\ a_1 \\ \vdots \\ a_n \end{bmatrix}_{2n+1 \times} + \begin{bmatrix} e(K+1 \\ e(K+2 \\ \vdots \\ e(K+N \end{bmatrix}$$

or $\hat{Y}_N = H\,\hat{\theta} + e$

where N is number of samples taken. Hence $N > > n$

$$\hat{Y}_N = H\,\hat{\theta} + e \qquad \ldots \text{(SI-}$$

$\hat{Y}_N$ = output response vector of N elements.
$\hat{e}$ = error vector of N elements
$\hat{\theta}$ = parameter vector 2n+1 elements.
H = Nonsquare matrix of 2n+1 x N elements.

Noe eq. (SI-3) can be written as

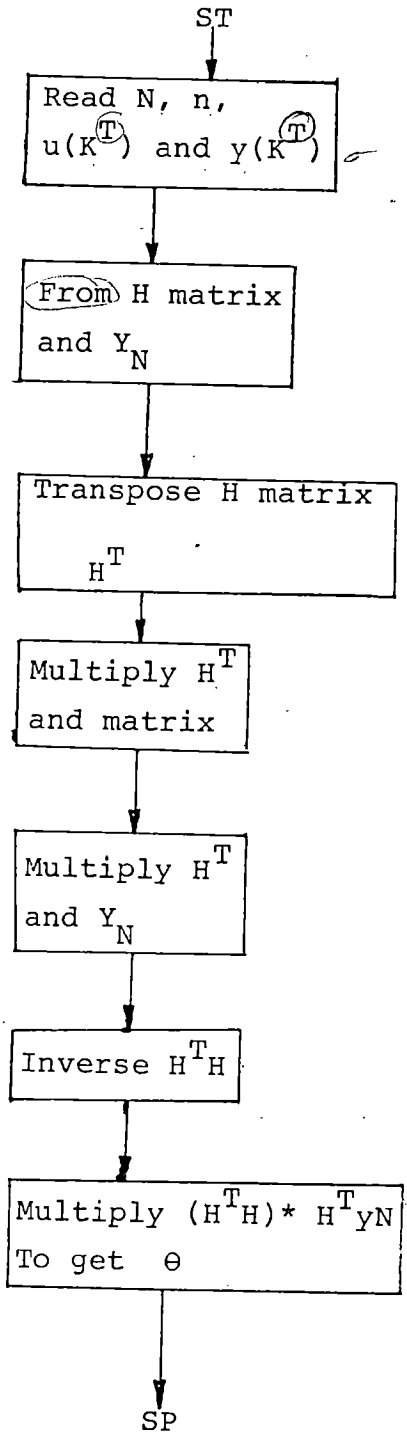
$$H^T \hat{Y}_N = (H^T H)\,\hat{\theta}$$

or $$(H^T H)^{-1} H^T \hat{Y}_N = \hat{\theta}$$

explain // Note that in the above process the error vector e vanish because of its zero mean property.

Hence $\hat{\theta} = (H^T H)^{-1} H^T Y_N$ .. (SI-4

Equation (SI-4) can easily be implemented on microcomputer t obtain a_i and b_i. Its flow chart is as given in Fig. 1.2.



Comment

Read n = order of system to beformed
N = nos. of samples
u(KT), y(KT) = discrete input & output condition N >> n.
form from H matrix based on eq.(SI-3)

Transpose H matrix - H^T

$\longrightarrow H^T H$

$\longrightarrow H^T Y_N$

$\longrightarrow (H^T H)^{-1}$

$\longrightarrow \hat{\theta} = (H^T H)^{-1} H^T Y_N$

Fig. 1.2

The draw back with this method is that order is not optimal But this method can be extended with [7] to give order of the system. The resulting programme written in PASCAL is given in Appendix 3.

1.3.2 Method-2

This is a method developed and tested by the author. In this method IIR model on ARMA model is basically derived from FIR model or MA model. Starting from equation (SI-1).

Let the system under consideration be of the form shown in Fig.1.3.

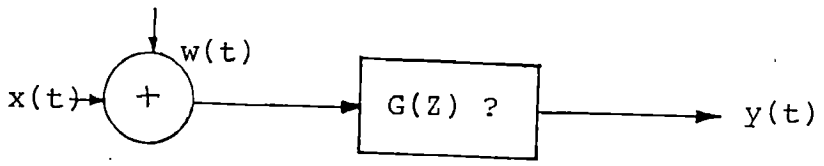


Fig.1.3

Now discretizing the input and the output

$$x(Z) \Rightarrow x(nT) \Rightarrow x(t) \quad \text{for } n = 0, 1, 2, \ldots$$
$$y(Z) \Rightarrow y(nT) \Rightarrow y(t)$$

.. (SI-5)

For sampled function X*(t) the auto correlation sequence under the erodicty assumption is defined as [8].

$$\phi_{xx}(nT) = \frac{1}{T} \lim_{K\to\infty} \frac{1}{2K+1} \sum_{i=-K}^{K} X(iT)\, X(iT+nT) = \frac{1}{T}\, \phi_{xx}(\tau)\,\Big|_{\tau = nT}$$

.. (SI-6)

Where T is the sampling time period.

Similarly cross correlation sequence between $x^*(t)$ and $y^*(t)$ is

$$\phi_{xy}(n\tau) = \frac{1}{T}\lim_{k \to} \frac{1}{2K+1} \sum_{i=-K}^{K} x(iT).\, y(iT+nT) = \frac{1}{T}\,\phi_{xy}(\tau)\Big|_{\tau = nT} \qquad .. \text{(SI-7)}$$

The fourier transform pair $\phi_{xx}(\tau)$ and $\phi_{xy}(\tau)$ represents statistical pairs of the sampled process. The pulse spectral density of the sampled function is

$$\Phi_{xx}(z) = \sum_{n=-} \phi^*_{xx}(nT).\, z^{-n} \qquad .. \text{(SI-8)}$$

Inverse Z-transform of $\Phi_{xx}(Z)$ is

$$\phi^*_{xx}(nT) = \frac{1}{2\pi j} \oint_{\Gamma} \Phi_{xx}(Z)\, Z^{n-1} dz \qquad .. \text{(SI-9)}$$

where the contour of integration Γ is the unit circle $|Z| = 1$. # for finite number of samples at regular sampling intervals,

$$\Phi_{xx}(Z) = \sum_{i=0}^{n} .\, x(i).\, x(i+n).Z^{-i} \qquad .. \text{(SI-10)}$$

$$\Phi_{xy}(Z) = \sum_{i=o}^{n} .\, x(i).\, y(i+n).Z^{-i} \qquad .. \text{(SI-11)}$$

and we also known [8] that

$$\Phi_{xy}(Z) = G(z).\Phi_{xx}(Z) \qquad .. \text{(SI-12)}$$

Hence,

$$G(Z) = \frac{\Phi_{xy}(z)}{\Phi_{xx}(z)} = \frac{[K_o, K_1, K_2, \ldots\ldots, K_m]}{[L_o, L_1, L_2, \ldots\ldots, L_m]} \quad .. \text{(SI-13)}$$

to get FIR model with a finite precision [9]

$$G(Z) = \text{deconvolute } (\Phi_{xy}(z); \Phi_{xx}(z))$$

or

$$G(Z) = \frac{[K_o/L_o, K_1/L_o, \ldots\ldots, K_m/L_o]}{[1, L_1/L_o, L_2/L_o, \ldots, L_m/L_o]} = \frac{[K'_o\ K'_1\ K'_2, \ldots, K'_m]}{[1, L'_1, L'_2, \ldots, L'_m]} \quad .. \text{(SI-14)}$$

on convolving above eq. we get

$$G_m = K'_m - \sum_{i=1}^{m} L'_i K'_{m-i} \quad .. \text{(SI-15)}$$

Where m varies from 0 to m.

So MA or FIR model of G(z) is

$$G(Z) = G_o + G_1 Z^{-1} + G_2 Z^{-2} + \ldots + G_m z^{-m} \quad .. \text{(SI-16)}$$

Now our aim is to evaluate a ARMA or IIR model out of this FIR model. From eq.(SI-1) and (SI.17).

Or,

$$H(z) = \frac{Y(Z)}{X(Z)} = G(Z)$$

Or

$$\frac{a_o + a_1 z^{-1} + \ldots + a_n z^{-n}}{1 + b_1 z^{-1} + \ldots + b_n z^{-n}} = G_o + G_1 z^{-1} + G_2 z^{-2} + \ldots \text{ m term}$$

.. (SI-18)

where $m >> n$

or

$$(a_o + a_1 z^{-1} + \ldots + a_n z^{-n}) = (1 + b_1 z^{-1} + \ldots + b_n z^{-n}) * (G_o + G_1 z^{-1} + G_2 z^{-2} + \ldots + G_m z^{-m})$$

$$\Rightarrow (a_o + a_1 z^{-1} + \ldots + a_n z^{-n}) = G_o + (G_1 + b_1 G_1) z^{-1} + (G_2 + G_1 b_1 + b_2 G_o) z^{-2} + (G_n + \sum_{i=1}^{n} b_i G_{n-i}) + \ldots\ldots\ldots + (G_m + \sum_{i=1}^{n} b_i G_{m-i}) z^{-n}$$

.. (SI-19)

equating the coefficients of like power of Z from both sides we get (n+1) term

$$\begin{aligned} a_o &= G_o \\ a_1 &= G_1 + b_i G_o \\ a_2 &= G_2 + b_1 G_1 + b_2 G_o \\ &\vdots \\ a_n &= G_n + \sum_{i=1}^{n} b_i G_{n-i} \end{aligned}$$

.. (SI-19)

Its matrix form will be

$$\begin{bmatrix} a_o \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 1 & & & \\ b_1 & 1 & & \\ b_2 & b_1 & 1 & \\ \vdots & \vdots & & \\ b_n & b_{n-1} & b_{n-2} \cdots & 1 \end{bmatrix} * \begin{bmatrix} G_o \\ G_1 \\ G_2 \\ \vdots \\ G_n \end{bmatrix} \qquad \text{.. (SI-21)}$$

Hence it is very easy to calculate the numerator term provided denominator term is known. In the L.H.S. of equation (SI-19) the term greater than Z^{-n} do not exist. Hence from R.H.S. of same equation

$$G_{n+1} + \sum_{i=1}^{n} b_i . G_{n+1-i} = 0$$

$$\text{or} \quad \sum_{i=1}^{n} b_i \, G_{n-i+1} = - G_{n+1} \qquad \text{.. (SI-22)}$$

So matrix form of above equation becomes.

$$\begin{bmatrix} G_1 & G_2 & G_3 & \cdots & G_n \\ G_2 & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ G_n & \cdot & \cdot & \cdot & G_{2n-1} \end{bmatrix} * \begin{bmatrix} b_n \\ b_{n-1} \\ \cdot \\ \cdot \\ b_1 \end{bmatrix} = \begin{bmatrix} -G_{n+1} \\ -G_{n+2} \\ \cdot \\ \cdot \\ -G_{2n} \end{bmatrix} \qquad \text{.. (SI-23)}$$

or in metrix notation, equn. (SI-23) can be written as

$$\Rightarrow \quad A\hat{b} = \hat{G} \qquad \ldots (SI\text{-}24)$$

N.K. SINHA & B. KUSZTA [1] has shown that rank of matrix A in equn. (SI-24) should be of full order and this full order becomes the order of the system and this matrix A is called Hankel Matrix [7]. Hence in other words (n+1) order of matrix A should result in $|A| = 0$ and hence n becomes the order of system. But some time matrix [A] does not converges to zero. So the method given in [1] is followed:

Let

$$A_n = \begin{matrix} \overleftarrow{\qquad n \qquad} \\ \end{matrix} \begin{matrix} n \\ \downarrow \\ \uparrow \end{matrix} \begin{bmatrix} G_1 & G_2 \ldots G_n \\ \vdots & \vdots \\ G_n \ldots\ldots\ldots & G_{2n-1} \end{bmatrix} \quad \text{and} \quad A_{n+1} = \begin{matrix} \overleftarrow{\qquad n+1 \qquad} \\ \end{matrix} \begin{bmatrix} G_1 & G_2 \ldots G_{n+1} \\ \vdots & \vdots \\ G_{n+} \ldots\ldots & G_{2n} \end{bmatrix}$$

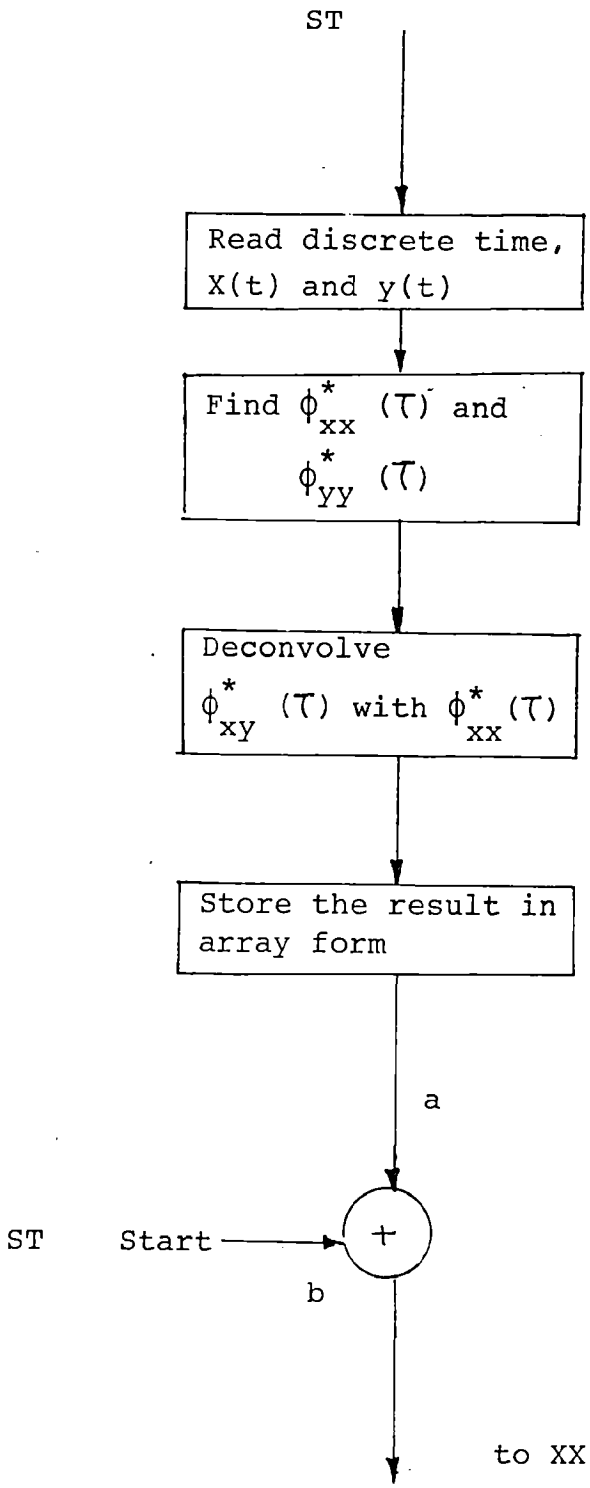
At each stage one can evaluate the element D(i) of the array.

$$D(n) = \frac{|A_i|}{|A_i + 1|}$$

for $i = 2,3,\ldots,n$ when $|A_i|$ is the determinent value of matrix A_i .

The value of i for which D(i) is the largest gives the order of the system. Once order is decided using gauss elimination method to equation (SI.23) b_i can be evaluator and after this using equation (SI.29) a_i can be evaluated and hence the system is identified in all respects.

The flow chart of this new method is shown in Fig. 1.4.



ARBITRARY INPUT ENTRY POINT

Read - KT, X(KT), Y(KT), and arrange in a array form.

Find auto and cross correlation function

deconvoluting $\phi^*_{xy}(\tau)$ with $\phi^*_{xx}(\tau)$ as given in eq.SI.13

At this point we get eq. SI.14.

If input X(t) is a unit step function then we can start it from this point.

The resulting programme in PASCAL is also listed in Appendix-2. This is also menu driven. The flow chart for this programme has two entry points, viz.(6) when the system input is a unit step function (b) when the input is any arbitrary for whose samples are known.

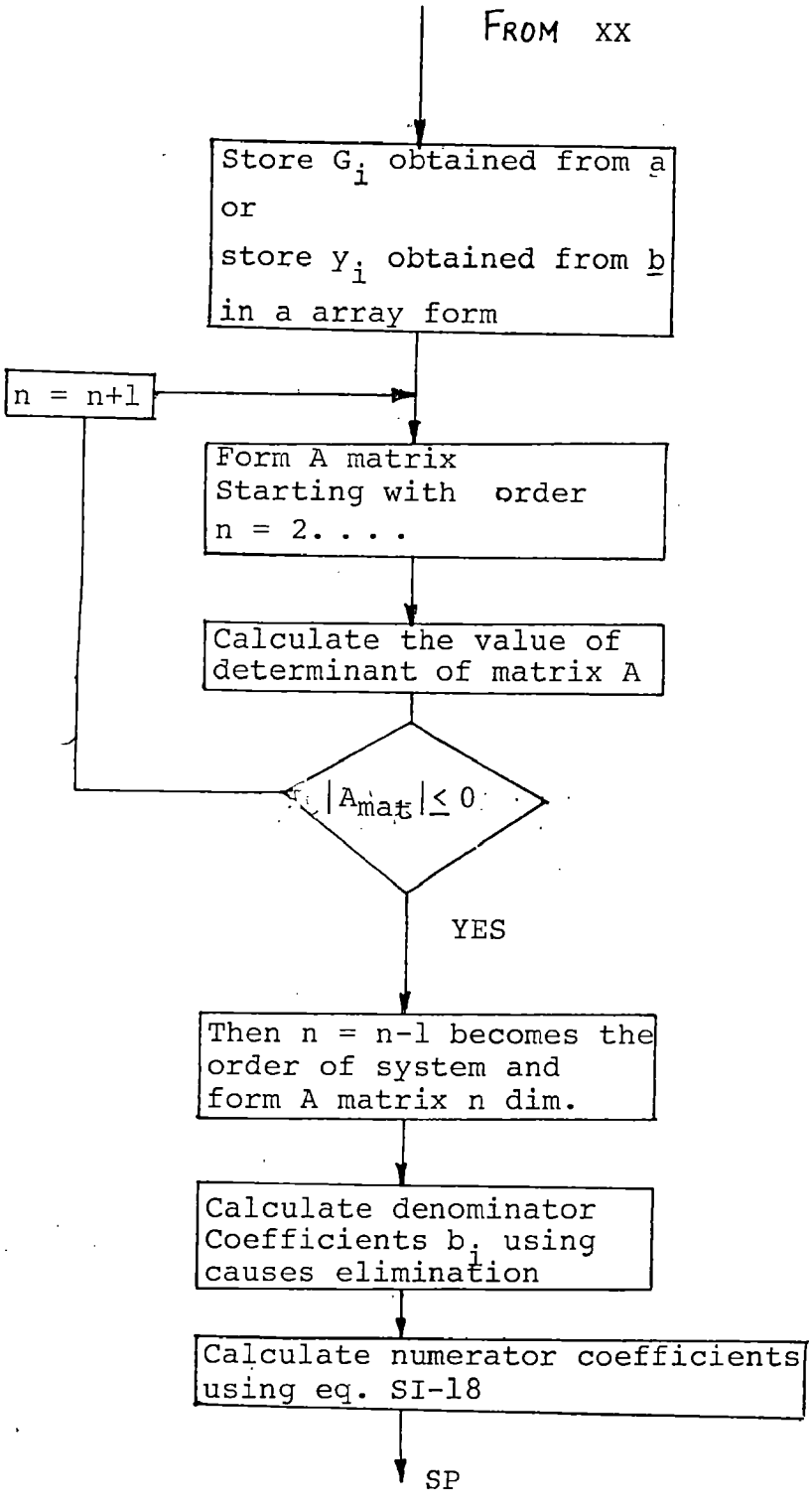


FIG.1.4

1.4 CONCLUSION

In this module an attempt has been made to make a tool box for system Identification. Two methods have been presented. The first being a known least square method and and the other developed by the author. The second method namely developed by author gives the optimal value for the order of system as well as the numerator and denominator coefficients of the ARMA model (I.I.R.). Whereas the first method assumes the order of the system. However the method developed by the author takes quite along time for arriving at the solution while first method is fast. Many problems as given in N.R.Sinha and B.Kuszta [1] have been tested using the identification software developed and gives identical result. Appendix-6 gives the result for the few of the problems.

CHAPTER - II

TRANSFER FUNCTION SIMULATION

TRANSFER FUNCTION SIMULATION

2.1 INTRODUCTION

The main purpose of this to perform the host of activities associated with transfer function viz., transfer function to state space conversion, inverse laplace, output simulation of continuous transfer function and discrete transfer functions.

2.2 TRANSFER FUNCTION TO STATE-SPACE CONVERS

Transfer function to state space program basically converts transer function into Bush form of state space. Any transfer function

$$G(S) = \frac{b_o S^n - b_1 S^{n-1} + ... + b_n}{S^n + a_1 S^{n-1} + ... + a_n} = \frac{y(S)}{(S)}$$

for SISO system can be represented in a state space form by equation [10].

$$\dot{\hat{x}} = A\hat{x} + B u$$
$$y = C\hat{x} + D u$$

then $A =$ $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \ldots & 0 \\ 0 & 0 & 1 & \ldots & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \\ 0 & 0 & \ldots & \ldots & \ldots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \ldots & \ldots & a_1 \end{bmatrix}$ is n x n matrix

$$B = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 1 \end{bmatrix}$$

is n x 1 matrix

$$C = [\ (b_n - a_n b_o),\ (b_{n-1} - a_{n-1} b_o) \ . \ . \ . \ (b_1 - a_1 b_o)\ ]$$

is 1 x n matrix

$$D = [1 \times 1] = [b_o] .$$

and hence its signal flow graph's given as shown in Fig.2.1 .

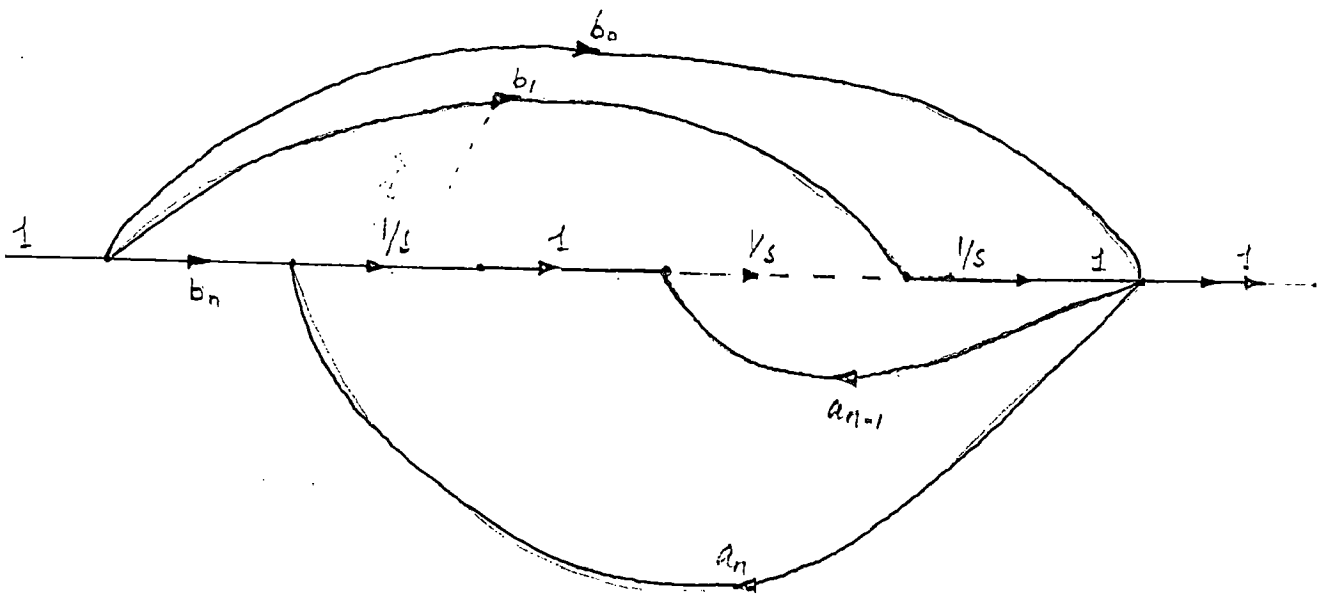


Fig.2.1

A, B, C, D matrix can be directly the moment G(s) is known. The programme, written is given in Appendix-2 and is menu

driven to implement bush transformation. At the same time this also serves as building block for state space approach to Block diagram reduction and analysis discussed in the next chapter.

2.3 INVERSE LAPLACE TRANSFORM

2.3.1 Introduction

The significance of numerical laplace inversion is obvious from the big range of applications. Well known is engg; Laplace transform methods are also used in order to solve differential and integral equations and to assist when other numerical methods are applied. Dubner and Abate [11] used fourier series for the numerical inversion of laplace transformation. Durbin [12] improved the same method. Further authors, Simon and Ks Crump [13] used different acceleration methods in order to speed up the convergence of the Fourier Series . The biggest disadvantage of the above mentioned methods is the dependance on the discretization, truncation error on the free parameter. At the same time method is a bit complex to implement through software. The laplace transform of a real function f: R $\longrightarrow$ $\mathbb{R}$ with f(t) = 0 for t < 0 and its inversion formulae

$$F(S) = L[f(t)] = \int_0^\infty e^{-st} f(t)\, dt. \qquad .. (2.1)$$

$$f(t) = L^{-1}[f(S)] = \frac{1}{2.\pi.i} \int_{v-i\infty}^{v+i\infty} e^{st} F(S)\, dS. \qquad .. (2.2)$$

with $S = V + i$; , $\in R$

$V \in R$ is arbitrary, but greater than the real parts of all the singurlarities of F(S). The integrals in (2.1) and (2.2) exist for $Re(S) > a \in R$ if

(a) f is locally integrable

(b) there exist a $t_o \geq 0$ and K, a $\in R$ such that

$$|f(t)| \leq Ke^{at} \text{ for all } t \geq t_o$$

(c) for all $t \in (0, \infty)$ there is a neighbourhood is which f is of bounded variation [14].

$$f(t) = \frac{e^{\gamma t}}{2\pi} \int_{-\infty}^{\infty} [Re(F(V+j\omega))\cos\omega t - Im(F(V+jw))\sin\omega t]d\omega . \quad .. (2.3)$$

— and on eq. (2.3) fourier transforms are applied in the existing methods.

2.3.2 Method Implemented

The method implemented for laplace transform is quite different from the conventional computer methods [11], [12], [13]. The method used in this dissertation is basically based on the conventional way of inverse laplace transform which has been studied in the class room. Only numerical method techniques are applied to it.

The method involves the classical way of finding partial fraction and then applying residue theorem [15]. Let us assume that a given rational function F(S) be written in the form [15].

$$F(S) = \frac{A(S)}{B(S)}$$

$$= \frac{a_o + a_1 S + a_2 S^2 + ... a_m S^m}{b_o + b_1 S + b_2 S^2 + ...b_n s^n} \qquad ..(2.4)$$

where S is a complex frequency variable, coefficient a_i and b_i are real quantities, and the degree of the numerator polynomial A(S) is less than degree of the denominator polynomial B(S) (i.e. m<n). The poles of the function F(S) [the roots of the polynomial B(S)] can be found using Newtons - Horners algorithms. The roots obtained either may be simple roots or complex roots. But first considering that we have a simple root located at the value of complex frequency variable p_i then

$$\lim_{S \to P_i} F(S) = \frac{K_i}{S - P_i}$$

The coefficient K_i in above equation is reffered to as the residue of the simple pole at P_i . If P_i is real, K_i will be real. In addition, since poles that are complex will always occur in conjugate pairs (this assumes that the b_i are real) it may be shown that the residues of such conjugate pair will also be conjugate [14]. The value of

the residue K_i may readily be determined directly from the function F(S).

$$K_i = \frac{A(P_i)}{B'(P_i)}$$

where p_i is a simple pole of F(S), $A(P_i)$ is the numerator polynomial of F(S) evaluated at $S = P_i$ and $B'(P_i)$ is the derivative of the denominator polynomial of F(S) (taken with respect to S), evaluated at $S = p_i$. The above relationship can easily be programmed for micro computer. In this way partial fraction expansion could be implemented on micro computer. After getting the partial fraction output, the next aim is to get its inverse laplace so as to get result in time domain. For this we proceed as follows. Let us assume that such a function has J simple real pole P_i and h pairs of complex conjugate pole located at P_i^c and P_i^c, where P_i^c is the complex conjugate of P_i^c. The function may then be expanded in the form

$$F(S) = \sum_{i=1}^{J} \frac{K_i}{S - P_i} + \sum_{i=1}^{h} \frac{K_i^c}{S-P_i^c} + \sum_{i=1}^{h} \frac{K_i^c}{S - P_i^c} \quad .. (2.5)$$

where K_i and K_i^c are residues of which K_i^c has a form $k_i^c = a_i + jb_i$, where a_i is the real part of the residue and b_i is the imaginary part. If we write $P_i^c = P_i^r + jP_i^i$ where P_i^r is the real part of the location of P_i^c and P_i^i

is the imaginary part then it may be shown that each pair of complex conjugate poles and the residues associated with them will have an inverse transform of the form

$$2\, e^{(t*P_i^r)}*[a_i^{\cdot} \cos (P_i^r.t) - (b_i \operatorname{Sin} (P_i^i.t)] \quad .. (2.6)$$

Thus the complete inverse transform for a function of the type is given by

$$f(t) = \sum_{i=1}^{J} K_i e^{P_i t} + 2 \sum_{i=1}^{h} e^{P_i^r.t} [a_i \cos(P_i^{(i)}.t) - b_i \sin (P_i^i.(t))] \quad .. (2.7)$$

And the above equation could easily be programmed on uc. The complete flow chart of this method is given as follows:

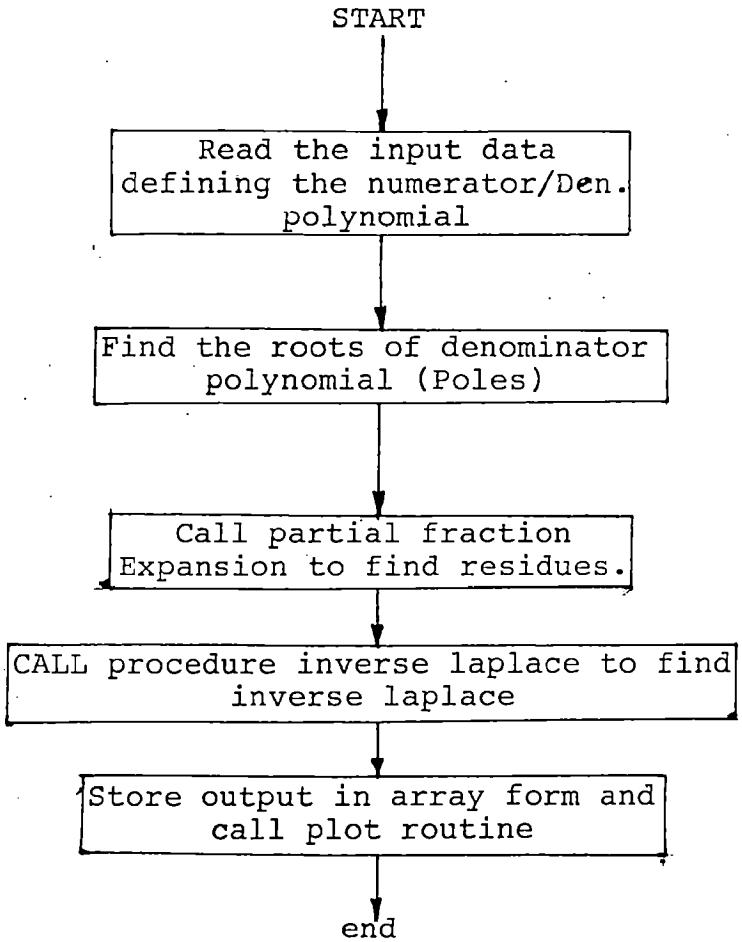


Fig. 2.2

Routine 'partial fraction' :-

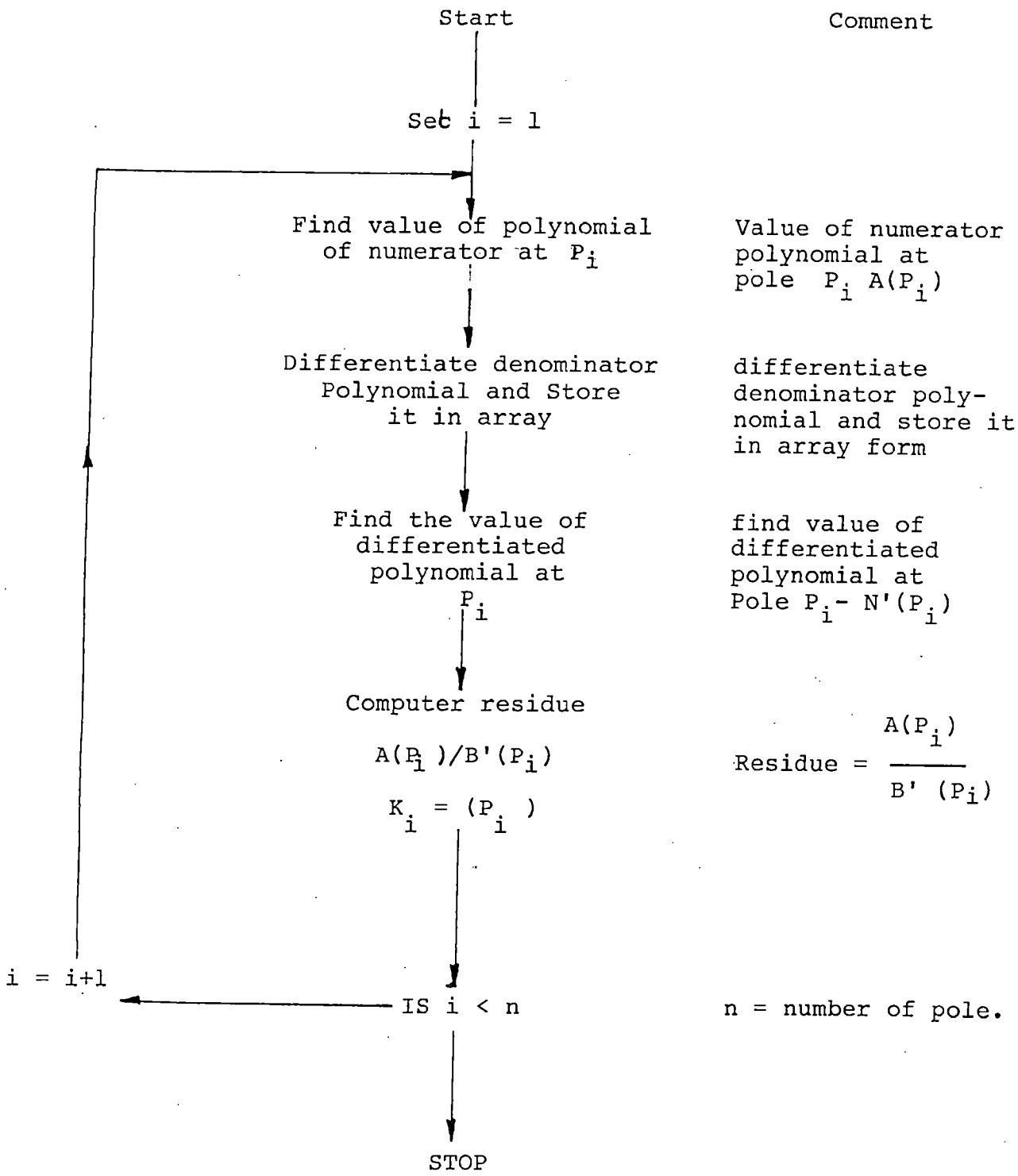


Fig. 2.3

ROUTINE OF INVERSE LAPLACE

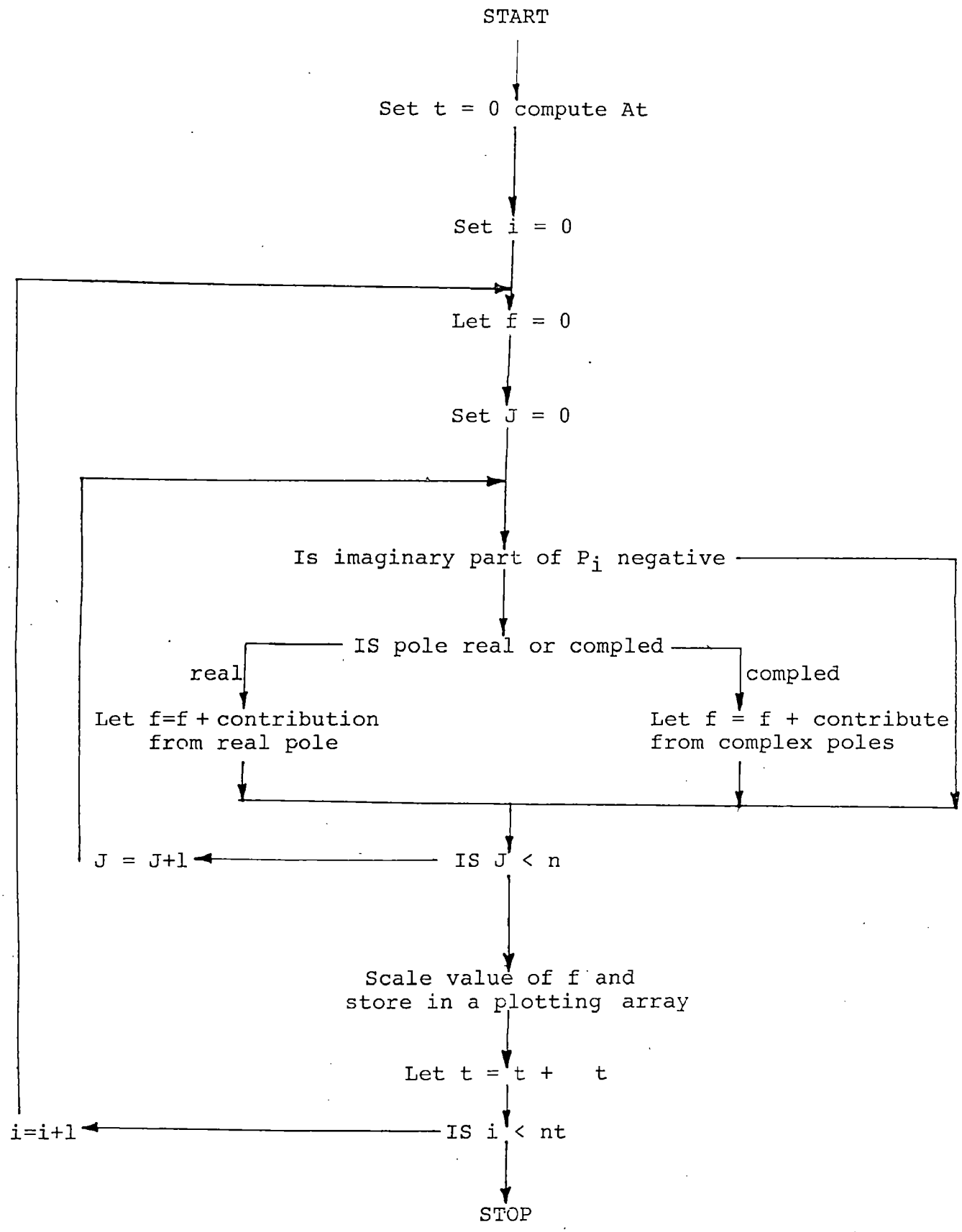


Fig.2.4

2.3.3 Limitation

The pitfall of this routine is that it cannot find the laplace inverse of critical cases i.e. it cannot evaluate inverse laplace of the cases in which e^{-ST} that is transportation log terms are involved.

2.4 CONTINUOUS TRANSFER FUNCTION SIMULATIONS

This method simulates the output for a given input and given T.F. .This method is extension of inverse Laplace transform. The discrete values of inverse laplace transform method is stored is an array. And the discrete values of input signals are also stored in a second array and the convolution is performed between these two array [9].

Consider that the discrete value after performining inverse laplace be represented as S(G) and discrete value of input signal be S(A), (provided sampling and range of time remain same for S(G) and S(A)).

then,

S(G) can be represented in a array as

$S(G) = [g_o, g_1, g_2, \ldots]$ and

$S(A) = [a_o, a_1, a_2, \ldots]$.

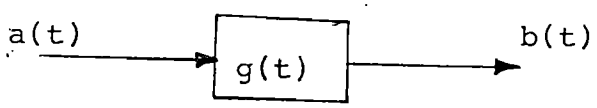


Fig. 2.5

then for continuous time

$$b(t) = \int_o^{\to t} a(x).\, g(t-x)\, dx$$

is the known as convolution or Dubmel integral. And for discrete time it is

$$b(K) = \sum_{n=0}^{K} a_n\, g_{k-n}$$

so in a array form it is like

$$S(A) = \boxed{a_o,\ a_1,\ a_2,\ \ldots\ldots}$$

$$\boxed{g_n, \ldots\ldots, g_3, g_2, g_1, g_o} = S(G)$$

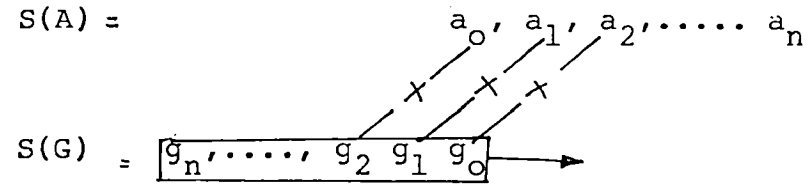
0^{th} term

$$\begin{array}{l} \qquad\qquad\qquad\qquad\qquad\qquad a_o,\ a_2,\ldots \\ \qquad\qquad\qquad\qquad\qquad\quad \times \\ \ldots\ g_3,\ g_2,\ g_1,\ g_o \\ \qquad\qquad\qquad\quad + \\ \hline S(B) = a_o g_o \end{array}$$

I^{st} term

$$\begin{array}{ll} S(A) & \qquad\qquad a_o,\ a_1,\ a_2, \ldots,\ a_n \\ & \qquad\qquad\ \times\ \ \times \\ S(G) = & \boxed{g_n, \ldots g_3, g_2, g_1, g_o} \longrightarrow \text{moving, performing addition} \\ & \qquad\qquad\quad + \\ \hline S(B) = & \qquad (a_o g_1 + a_1 g_o) \end{array}$$

2^{nd} term



$$S(B) = a_o g_2 + a_1 g_1 + a_2 g_o \ldots .$$

Hence S(B) array becomes

$$S(B) = a_o g_o,\ a_o g_1 + a_1 g_o, \ldots\ldots\ldots\ldots$$

Now once S(B) array is formed then it could be plotted against time array.

2.5 DISCRETE T.F. SIMULATION

Consider any discrete transfer function be represented by a rational polynomial.

$$H(Z) = \frac{b_o + b_1 Z^{-1} + b_2 Z^{-2} + \ldots . b^n Z^{-n}}{a_1 + a_1 Z^{-1} + a_2 Z^{-2} + \ldots\ldots .} \quad \ldots (2.7)$$

where

$H(Z)$ = $y(Z) / u(Z)$

$y(Z)$ = discrete output and

$u(Z)$ = discrete input .

then,

$$\frac{y(Z)}{u(Z)} = \frac{b_o + b_1 Z^{-1} + b_2 Z^{-2} + \ldots b^n Z^{-n}}{1 + a_1 Z^{-1} + a_2 Z^{-2} + \ldots . a^m Z^{-m}} \quad .. (2.8)$$

or,

$$y(Z) * (1 + a_1 Z - 4\, a_2\, Z^{-2} + \ldots + a_m Z^{-m})$$

$$= u(Z) * (b_o + b_1 Z^{-1} + \ldots\ldots b_n Z^{-n})$$

or,

$$y(K) = - a_1 y(K-1) - a_2(K-2), \ldots - a_m y(K-m) + b_o u(K) + b_1 u(K-1)$$

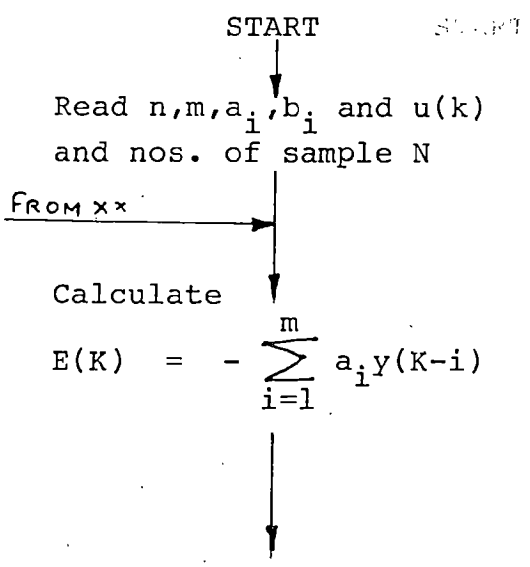
$$+ \ldots b_n\, u\, (K-n)$$

or,

$$y(K) = - \sum_{i=1}^{m} a_i y(K-i) + \sum_{i=0}^{n} b_i u(K-i) \quad .. (LT-10)$$

Now the above equation can be programmed for simulating its output.

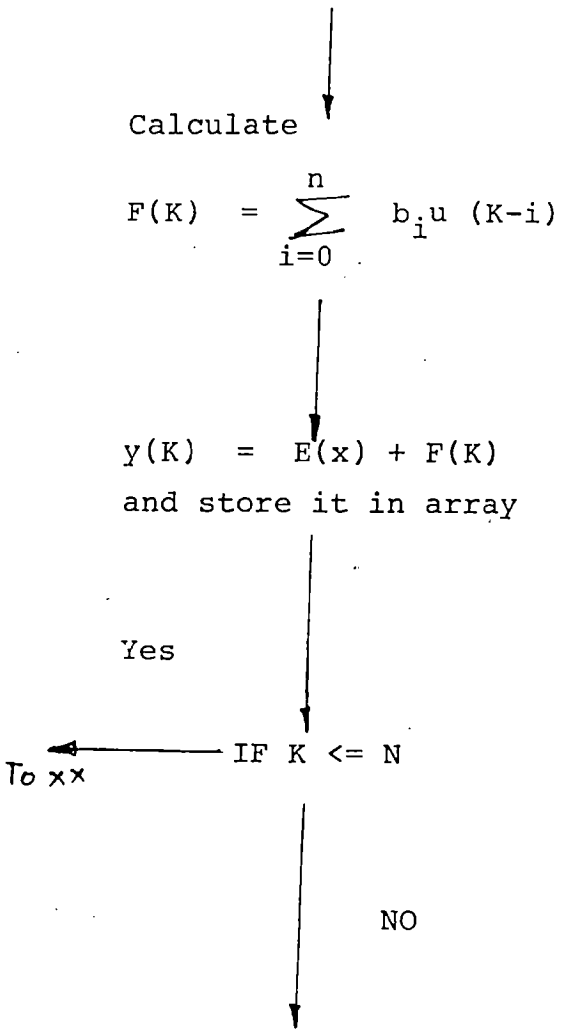
Hence its flow chart can be derived as follows:



COMMENT

Read n,m,a(i),b_i and u(t), N eith form ey board/file.

Contd..



Then print array of output that is y(K) .

2.6 Conclusion

The developed program is a menudriven and is given in Appendix-2. The data can be inputted either through the keyboard or file. The program for inverse laplace and output simulation is provided with a graphical facilities. The graphical routines are also written in PASCAL. With few of the limitations and pitfalls, the method provides a good graphical assistance.

CHAPTER - III

BLOCKDIAGRAM REDUCTION

STATE SPACE APPROACH FOR BLOCK DIAGRAM REDUCTION

3.1 INTRODUCTION

Block diagrams are very useful for representing control systems. To analyse the system block diagram is reduced as per conventional technique [1] but is restricted for simple block diagrams. An alternate approach is that of signal flow graphs developed by S.J. Mason, which does not require any reduction process of a flow graph gain formula which relates the input and output system variable for forward path gains and loop gains. But signal flow graph is not easy to implement as a whole on a microcomputer. Therefore, matrix approach is adopted to implement the block diagram on the P.C. Hence the state space approach to block diagram reduction method is adopted here as a solution. However the method at present is restricted to SISO systems. [34].

3.2 PROPOSED METHOD

For any SISO system, state model is given by [10].

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x} + Bu \\ y &= C\hat{x} + Du \end{aligned} \qquad \ldots(3.1)$$

$\hat{x}$ is n x 1 State Vector, u → u(t), A is n x n system matrix, B is n x 1 input matrix C is 1 x n output matrix D is 1 x 1 transmission matrix which is 'd' in this special case of SISO system.

Its block diagram could be represented as shown in Fig.3.1.

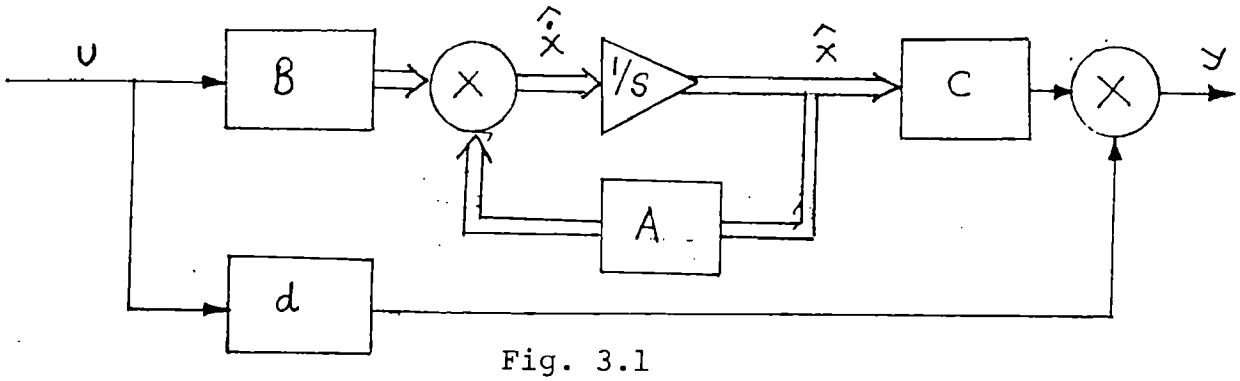


Fig. 3.1

Equivalently figures such as shown in Fig.3.1 can also be represent simply as shown in Fig.3.2.

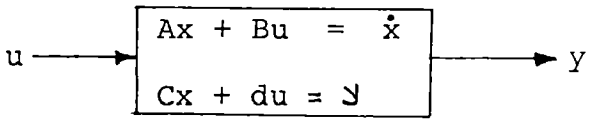
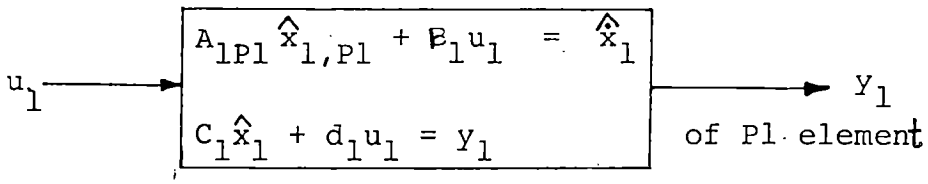


Fig. 3.2

Let A, B, C, d be of bush form type [10] for a given tansfer function. Now consider any block diagram which has got n blocks of single input and single output and may be interconnected in any fashion. So it is quite obvious that input of one block might be output of some other block or blocks. Consider that there are n blocks. Fig. 3.3 shows these n blocks with nomenclature shown there in for each block.



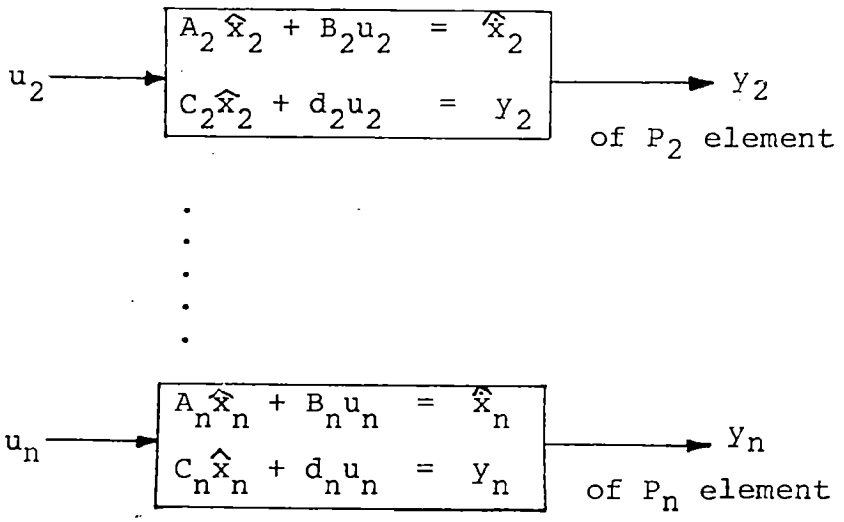


Fig.3.3

Now when each block is connected to each other in a regular fashion as given in a problem then it is quite obvious that the output of many of the blocks may be the input to the other block/blocks.

This input-output connection relationship can be characterised by the matrix equation:

$$u = K * y + u_r \qquad .. (3.2)$$

where K is the connection matrix consisting only of 0, -1, +1 having the following meanings:

0 = for no direct connection between two blocks
-1 = direction connection with negative gain
1 = Direct connection between two block with positive gain.
K = is thus a n x n connection matrix
U = is an input vector as seen in Fig. 3.3
y = is the output vector as seen in Fig.3.3
u_r = input vector.

The aim then is to write derivative terms of above blocks together so as to formulate a single state space model.

$$\begin{aligned}
\hat{\dot{x}}_{1(P_1)} &= A_{1,(P_1)}\, \hat{x}_{1,(P_1)} + \hat{B}_1 u_1 \\
\hat{\dot{x}}_{2(P_2)} &= A_{2,(P_2)}\, \hat{x}_{2,(P2)} + \hat{B}_2 u_2 \\
&\vdots \\
\hat{\dot{x}}_{n(P_1)} &= A_{n,(P_n)}\, \hat{x}_{n,(P_n)} + \hat{B}_n u_n
\end{aligned} \qquad .. (3.3)$$

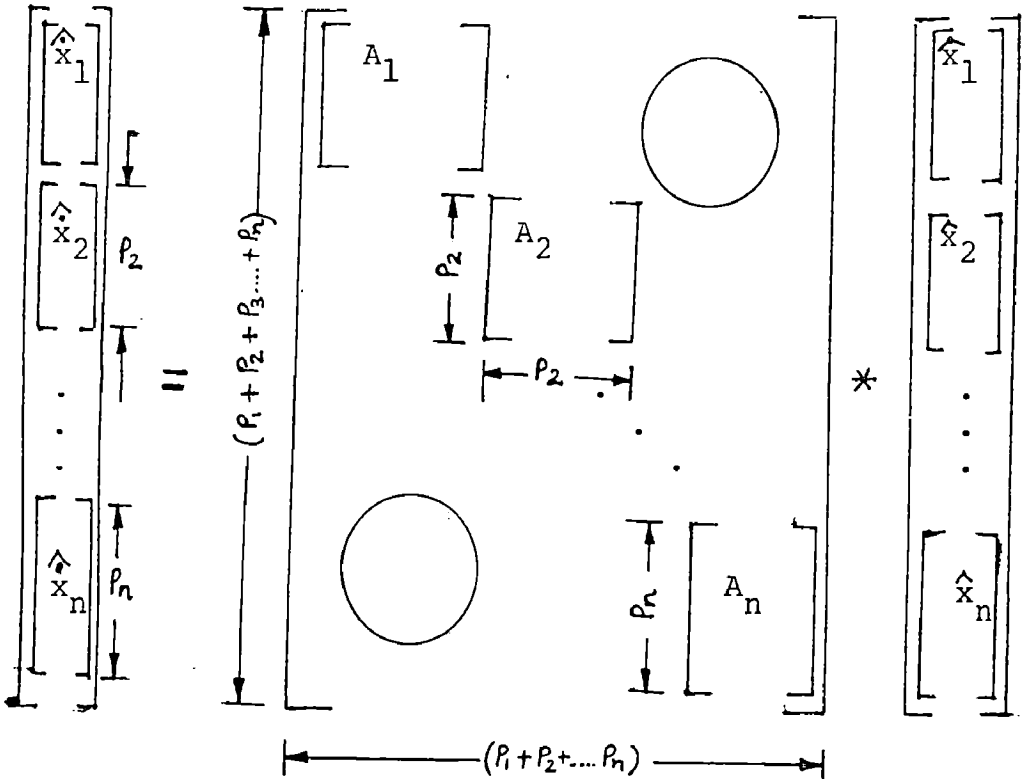
where P_1, P_2, ..., P_n is nos. of state variables in each block.

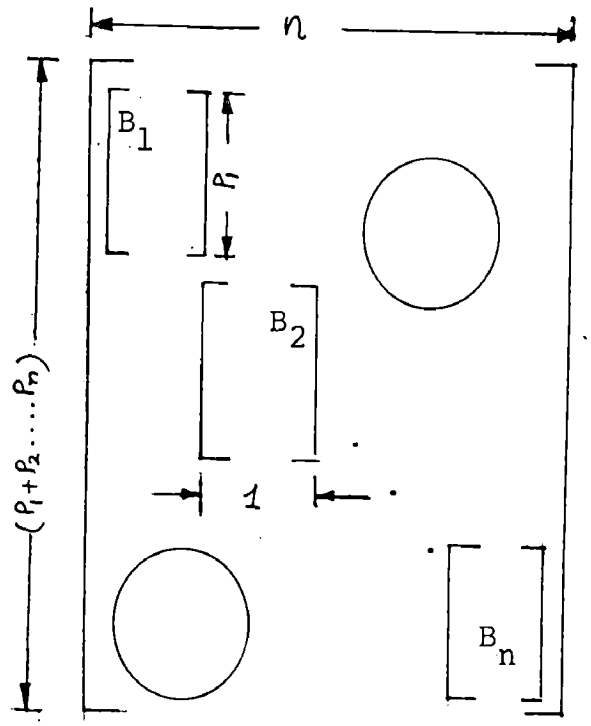
Equation 3.3 can be converted into a matrix equation of the

$$\begin{bmatrix} \hat{\dot{x}}_1 \\ \hat{\dot{x}}_2 \\ \vdots \\ \hat{\dot{x}}_n \end{bmatrix} \begin{matrix} \text{Pl element} \\ \text{P2 element} \\ \vdots \\ P_n \text{ element} \end{matrix} = \begin{bmatrix} [A_1] & & & \\ & [A_2] & & \\ & & [A_3] & \\ & & & [A_n] \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_n \end{bmatrix}$$

$$+ \begin{bmatrix} \hat{B}_1 & & & \\ & \hat{B}_2 & & \\ & & \hat{B}_3 & \\ & & & \hat{B}_n \end{bmatrix} * u \qquad .. (3.4)$$

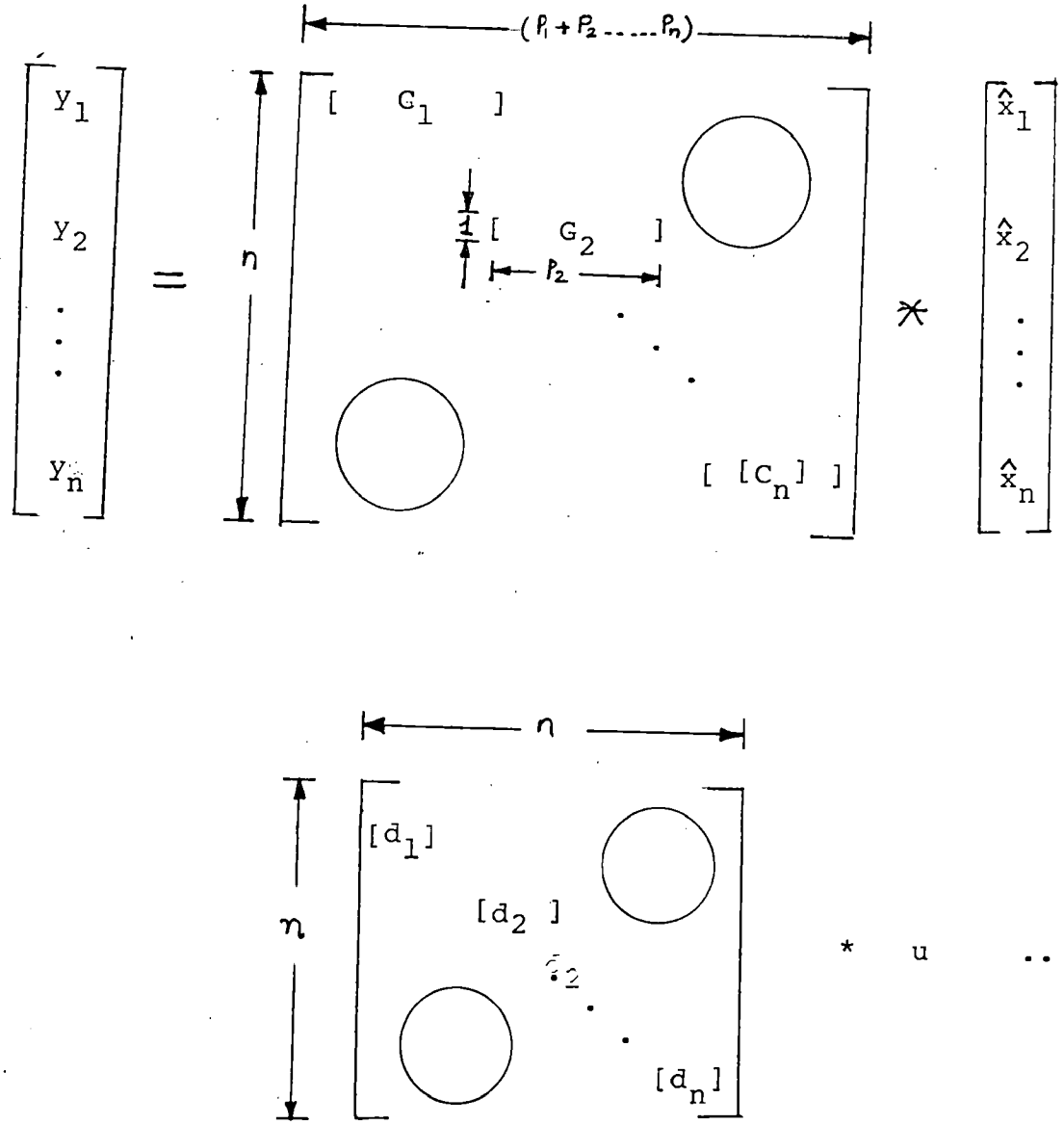
Equation (3.4) is equal to as given in equation 3.5.





* u .. (3.5)

Similarly output matrix is given as shown in equation (3.6).



Matrix given in equations (3.5) and (3.6). Now be called as A, B, C, D matrices so that these equations reduces to

$$\dot{\hat{x}} = A\hat{x} + Bu$$
$$\hat{y} = C\hat{x} + Du \qquad .. (3.7)$$

It is to be noted that the A, B, C, D matrices given in equation 3.7 are the matrices as formulated in equation 3.5 and equation 3.6 and has nothing to do with the system matrix

A, B, C, D of equation 3.1. As pointed out earlier let K be the connection matrix for n blocks shown in Fig. 3.3. Hence this matrix has got n x n dimension consisting of elements -1, +1 and 0 which defines the connection as shown in Fig. 3.4.

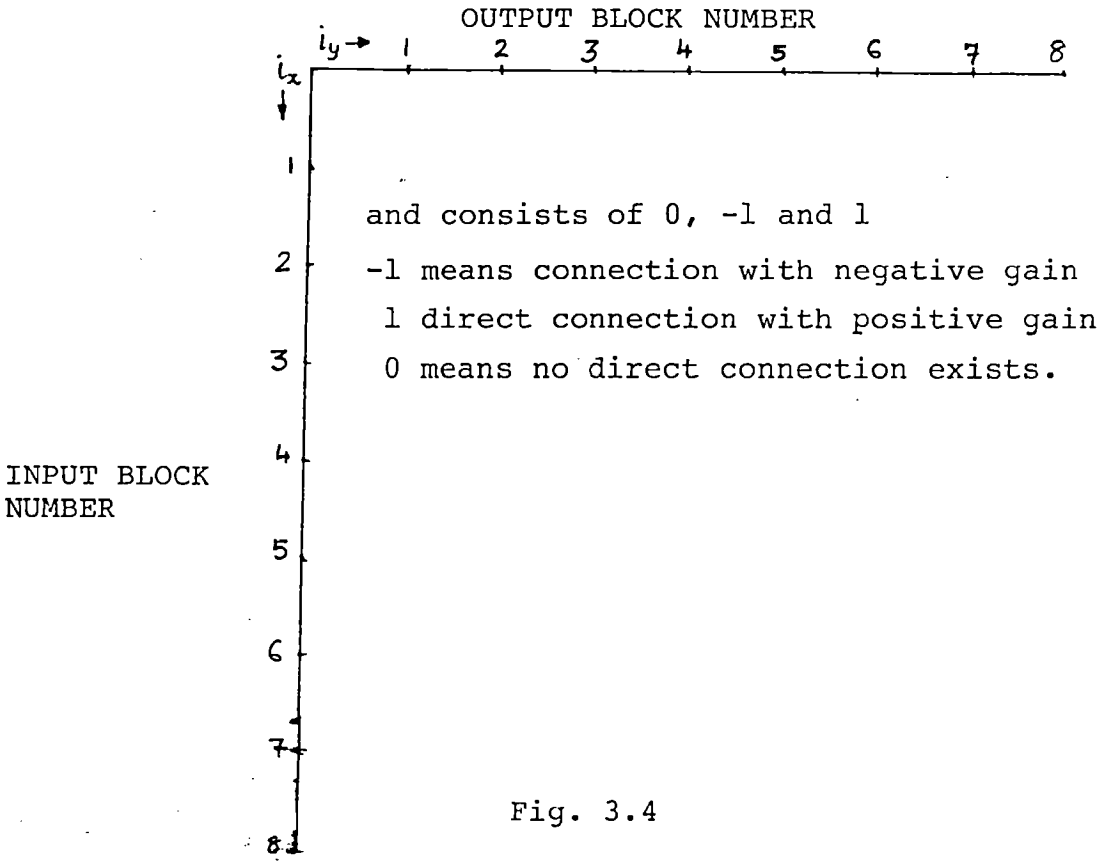


Fig. 3.4

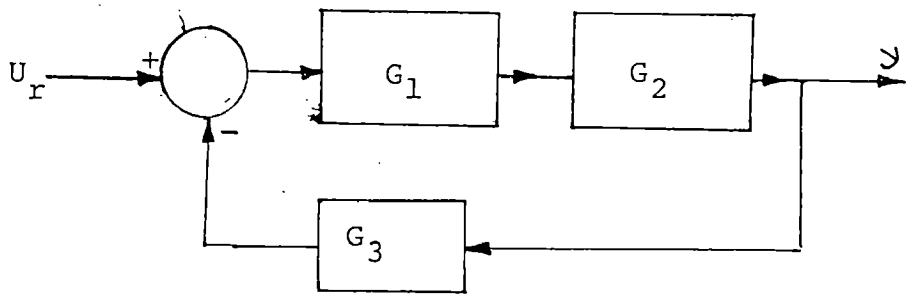


Fig. 3.5

For example, the K matrix for figure 3.5 is $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}$

The given system therefore is described by 3 basic matrix equation as given by equations (3.2) and (3.7) and with these equations one has to arrive at a relation for given input i_x and given output i_y.

$$\hat{x}_{iu} = AA\,\hat{x} + BB\,u_r \qquad .. (3.8)$$

$$\hat{y}_{iy} = CC\,\hat{x} + DD\,u_r \qquad .. (3.9)$$

Where AA, BB, CC, DD are the matrices identical to system matrices of equation (3.1).

From equation (3.7) and (3.2),

$$\hat{y} = C\hat{x} + D\,(K\hat{y} + Ur)$$

or

$$\hat{y} = C\,\hat{x} + D*K*\hat{y} + D\,u_r$$

or

$$(\hat{y} - D*K*\hat{y}) = C\hat{x} + D\,u_r$$

or

$$(I - D*K) * \hat{y} = C\hat{x} + D\,u_r$$

or

$$(I - D\,K)^{-1}\,(I - D*K) * \hat{y} = (I - D*K)^{-1} C\,\hat{X} + (I-D*K)^{-1} D u_r$$

$$\Rightarrow \hat{y} = (I - D*K)^{-1} C\hat{X} + (I-D*K)^{-1}\,D\,U_r \qquad .. (3.10)$$

Comparing the equation (3.4) with (3.9) one can say that

$$CC = (I - D*K)^{-1}\,C$$

$$DD = (I - D*K)^{-1}\,D \qquad .. (3.11)$$

The value of i_y specified will give the $y|_{iy}$ for the given input U_r .

i.e. $\hat{y}|_{iy}$ is output point.

CC (iy, $(P_1 + P_2 + \ldots + P_n)$ given the C matrix for that given output point.

Similarly, D (iy, ix) will give the value of D-matrix for given input and output point.

Again from equation (3.7) and (3.10) and putting them in first section of equation (3.7), one gets

$$AA = A + B * K * (I - D * K)^{-1} * C \quad \ldots (3.12)$$

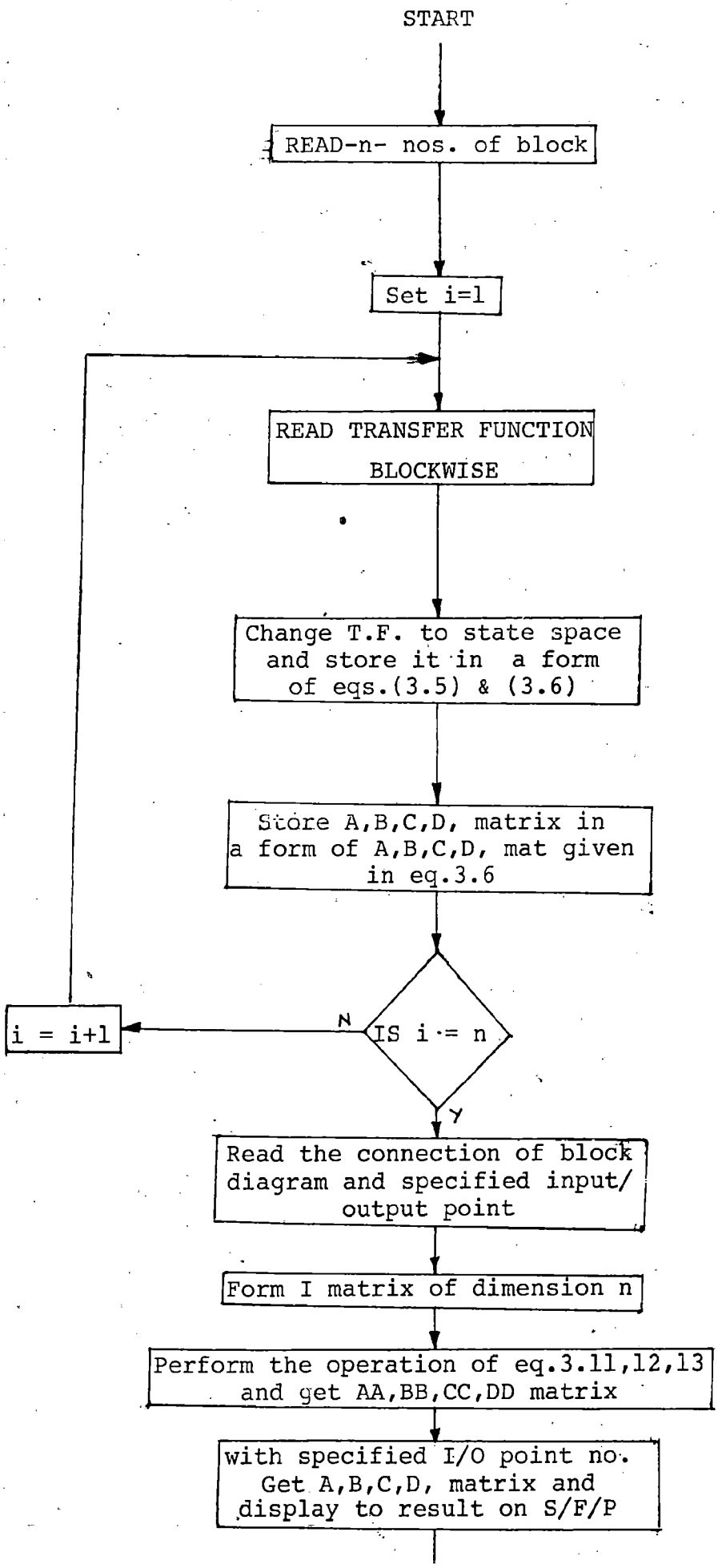
and $$BB = B * (I - K * D)^{-1} \quad \ldots (3.13)$$

... AA is state matrix and BB is input matrix hence BB $((P_1 + P_2 + \ldots + P_n)$, ix) will give the B matrix.

Hence equations (3.11), (3.12) and (3.13) can be programmed on a microcomputer to directly give the respective matrices A, B, C, D.

The flow chart of the above method is shown in Fig.3.6. It is to be noted that the each block diagram is described by its transfer function and the programme automatically gives the corresponding state space equations of each block.

The software programme written in PASCAL corresponding to flow chart of Fig.3.6 is given in Appendix .


START
READ-n- nos. of block
Set i=1
READ TRANSFER FUNCTION BLOCKWISE
Change T.F. to state space and store it in a form of eqs.(3.5) & (3.6)
Store A,B,C,D, matrix in a form of A,B,C,D, mat given in eq.3.6
IS i = n
N
i = i+1
Y
Read the connection of block diagram and specified input/ output point
Form I matrix of dimension n
Perform the operation of eq.3.11,12,13 and get AA,BB,CC,DD matrix
with specified I/O point no. Get A,B,C,D, matrix and display to result on S/F/P

3.3 CONCLUSION

An attempt has been made to apply state space approach for block diagram reduction and analysis. State space has got a great deal of advantage over conventional control system analysis like determination of controlability, observability and other methods can easily be incorporated directly to analyse the systems. The program developed as given in appendix-6 has been used successfully.

####

CHAPTER - IV

FFT - APPLICATIONS

FAST FOURIER TRANSFORM APPLICATIONS

4.1 INTRODUCTION

This module make use of Fast Fourier Transform (FFT) routine provided in turbopascal 5.5 integrated package, in a unit file (TPU file). They are FFTB2.PAS or FFT87B4.PAS and are compiled to form. TPU file. FFT is the Discrete fourier transform with a time complexity $n.lg_2 n$. The discrete fourier transform (DFT) is defined as [CH4-1].

$$H(K) = \sum_{n=0}^{N-1} h(n)\, e^{-j.2.\pi.Kn/N} \quad \text{for } 0 \leq K \leq N-1 \quad ..(FFT-1)$$

where N = number of discrete samples

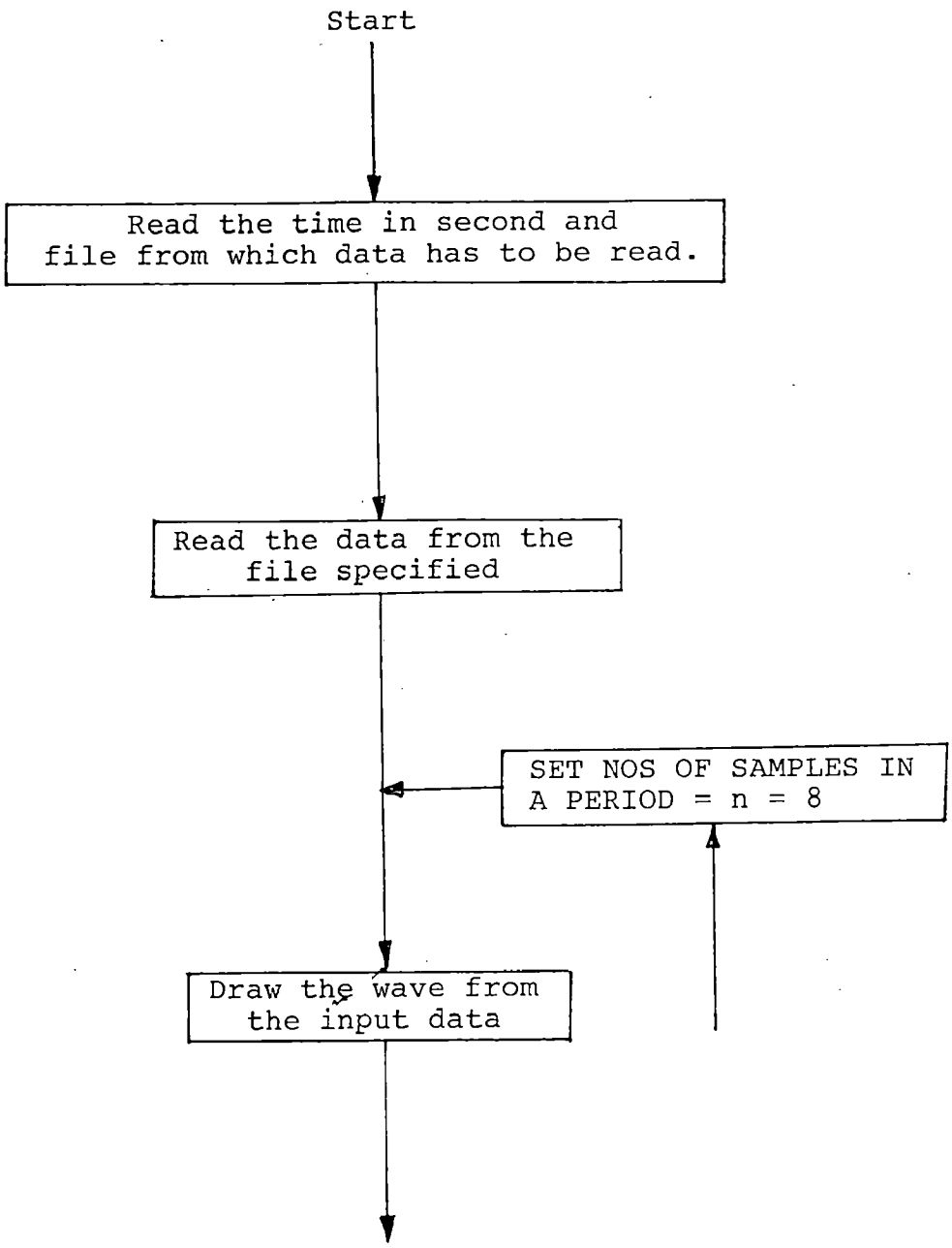
and the inverse DFT is given by

$$h(n) = \frac{1}{N} \sum_{K=0}^{N-1} H(K)\, e^{j\ 2.\pi.K.n/N} \quad \text{for } 0 \leq n \leq N-1,,,(FFT-2)$$

4.2 SIGNAL RECONSTRUCTION

In this subprogram FFT module is used in showing the effect of number of samples taken for the reconstruction of signal. The main aim of this program is to show the effect of change in sampling frequency and then reconstruction of signal. In this program any discrete time series samples of a signal is sampled at different sampling frequency and then it is reconstructed at that frequency using inverse FFT. This

subprogram gives the graphical display of what is done. This use of Turbopascalgraphical tool box. Here in this program, screen is divided in four windows. Two small windows is for transformation. Third one is instruction window. And fourth window which is the main window in for display. The discrete input is given through the file. The flow chart for the program is as follows.



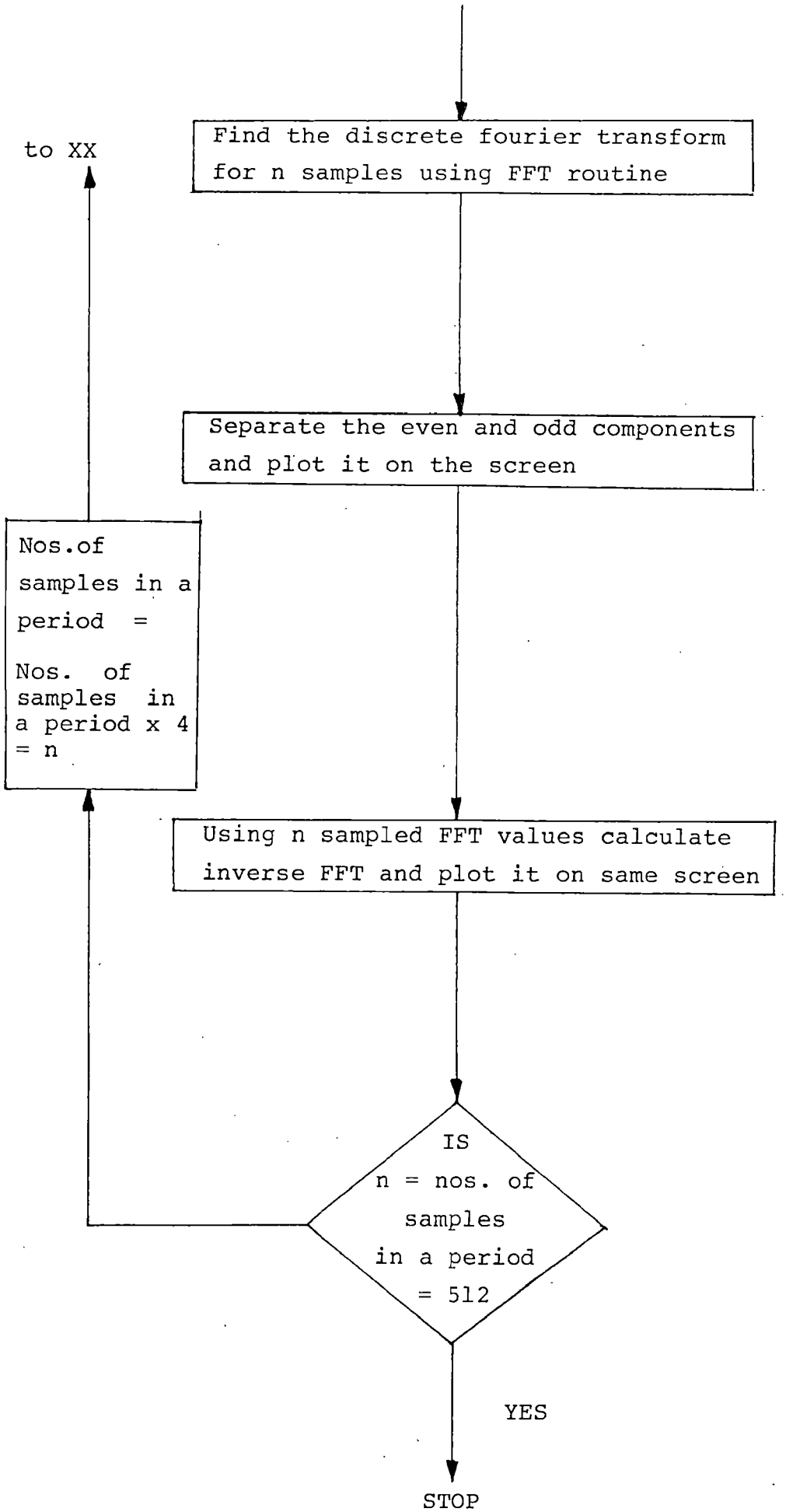

to XX
Find the discrete fourier transform for n samples using FFT routine
Separate the even and odd components and plot it on the screen
Nos.of samples in a period =
Nos. of samples in a period x 4 = n
Using n sampled FFT values calculate inverse FFT and plot it on same screen
IS n = nos. of samples in a period = 512
YES
STOP

Fig. 4.1

4.3 AUTOCORRELATION CROSSCORRELATION AND CONVOLUTION

In this submodule, FFT is used in evaluating autocorreation, cross correlation and convolution fucntions for both real and complex set of discrete data.

Auto correlation function is defined as [2]

$$\phi_{XX}(nT) = \frac{1}{T} \lim_{K\to\infty} \cdot \frac{1}{2K+1} \cdot \sum_{i=-K}^{K} X(iT).x(iT+nT) = \frac{1}{T} \phi_{XX}(\tau)|_{\tau=nt}$$

.... (FFT-3)

where T is the sampling period

X(iT) is the set of discrete data

X(iT+ nT) is the set of same discrete data displaced by nT samples.

Similarly, crosscorrelation is defined as [2]

$$\phi_{xy}(nT) = \frac{1}{T} \lim_{K\to\infty} \frac{1}{2K+1} \cdot \sum_{i=-K}^{K} x(iT).y(iT+nT) = \frac{1}{T} \phi_{xy}(\tau)|_{\tau=nT}$$

.... (FFT-4)

where y(iT+nT) is second set of data displaced by nT samples

Convolution is same as crosscorrleation fucntion.

Hence by equation - (FFT-1) we have

$$H(K) = \sum_{n=0}^{N-1} h(n)\, e^{-j\, 2\pi Kn/N} \quad \text{for} \quad 0 \le K \le N-1$$

and $$h(n) = \frac{1}{N} \sum_{K=0}^{N-1} H(K) e^{j\, 2\pi Kn/N} \quad \text{for} \quad 0 \le n \le N-1$$

Since discrete fourier transform series converts any discrete time series in discrete frequency series and vice a versa. It is known that convolution in time domain is a multiplication in frequency domain. [ ]

Hence for finding cross correlation of two discrete time series x(t) and y(t) we find their DFT which.

$$X(K) = \sum_{n=0}^{N-1} X(n)\, e^{-j2\pi . K . n/N} \qquad \text{for } 0 \leq n \leq N-1$$

$$Y(K) = \sum_{n=0}^{N-1} Y(n+p)\, e^{-j\, 2\pi . K(n+p)/N} \qquad \text{for } 0 \leq n \leq N-1$$

* *

(Where p is displacement.)

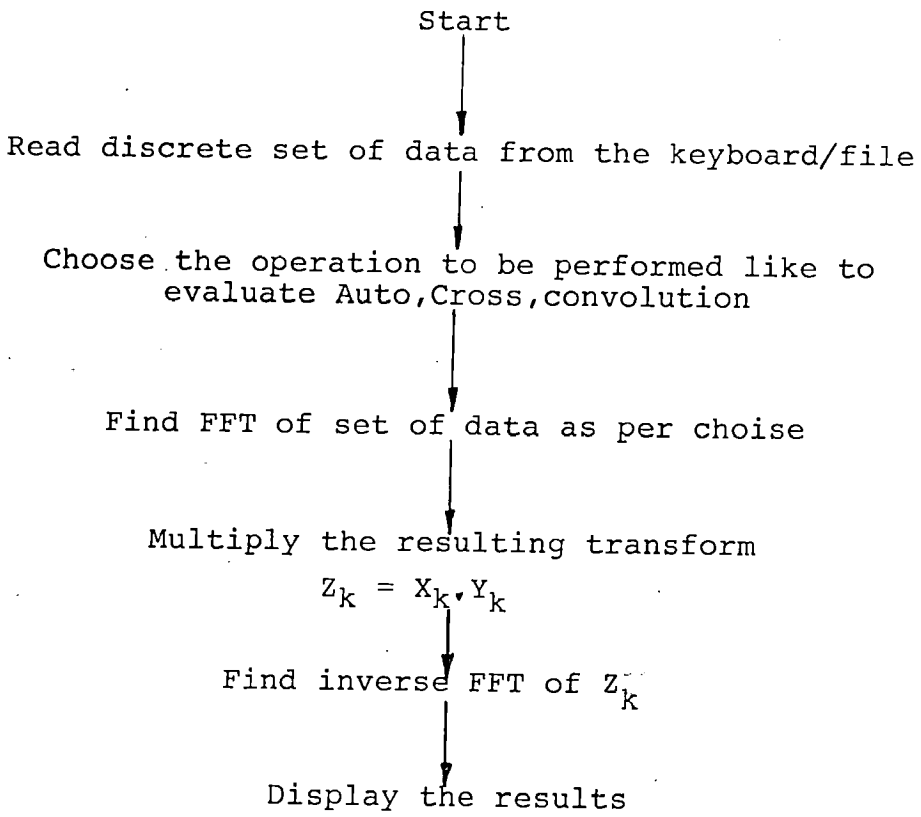
$$\phi^{F}_{xy}(K) = X(K) * Y(K) = \sum_{n=0}^{N-1} X(n) e^{-2\pi K\, n/N} * \sum_{n=0}^{N-1} Y(n+p) e^{-j\, 2\pi K(n+p)/N} \quad \ldots.(FFT\text{-}5)$$

which gives cross correlation in frequency domain. For finding its time domain value we have to take inverse discrete fourier transform eq. FFT-5 .

$\therefore \quad \phi_{xy}(nT) = \text{inverse DFT } (\phi^{F}_{xy}(K))$

$$\phi_{xy}(\tau)|_{(=n} = \frac{1}{N} \sum_{K=0}^{N-1} e^{j\,2\pi.K.n/N} \cdot \sum_{n=0}^{N-1} X(n)e^{-j2.\pi.K.\,n/N} \cdot \sum_{n=0}^{N-1} y(n+p)e^{-j2\,\pi.\,K.(n+p)/N} \quad \ldots\ldots(FFT\text{-}6)$$

Equation FFT-6 is programed for finding cross correlation, and convolution function. In equation (FFT-6) two forward FFT operations and one backward FFT operation is performed. If x(n) and y(n) are complex set of data then two more FFT operations are performed on the set of data. [17] The flow chart for the implementation is as follows:



Hence this module can work as a advance calculator for the student of systems studies.

CHAPTER - V

MATHEMATICAL FUNCTIONS

MATRIX APPLICATION

The main aim of creating this module is to give a menu touch to already existing subprogram for matrix operations like.

1) Determinant of the matrix
2) Inverse of the matrix
3) Eigen Value and Eigen Vector
4) Gauss Jorden method for solving simultaneous equation

By this sub modules program microcomputer can be used as calculator for matrix operations. This submenu has got four choices and depending upon the key pressed, microcomputer performs that operation and again comes to same submenu. This module uses three Turbopascal 5.5 units, they are DOS, Crt and common.

LEAST SQUARE OPERATIONS

This module deals basically with the curve fitting techniques based on the data paints (x,y). A suitable and well approximated curve can be derived out of a polynomial or logrithim, or exponential, or Fourier equations. The main aim of creating this module is to provide user with a menu driven touch to a already well tested existing subroutines so that microcomputer can be used as a calculator for finding the equation to a set of data. Four type of equations can be

drived are:

1) Polynomial

2) Exponential

3) Logarithm

4) Fourier

Depending upon the user's choice for fitting the data, program is executed. It input data either from key board or file and output the result to screen, file or printer depending upon the user's choice. This module uses three unit of Turbopascal 5.5 they are DOS, CRT, COMMON.

MATHS.INC: It is the include file provide with this package which is in the source code that is pascal. It includes advance mathematical procedures which are as follow.

(1) Matrix inversion routine - named as inverse { }.

(2) Determinant of matrix - Det ( );

(3) Eigenvalue of a matrix - Eigen ( );

(4) Gauss Jorden method of solving equation - Gauss_Jord ( );

(5) Least Square approximation - named as LSE_POLY ( ); for evaluating polynomial out of a set of data.

(6) LSE_EXPO( ); - for exponential function

(7) LSE_LOG( ); - for hegrathimic function

(8) LSE_FOUR( ); - for fourier function

Thus this math.Inc file can be used for further development of this package and can be enhanced from time to time.

CHAPTER - VI

The package also has an optional interfaces to MATLAB, a mathematical package, which allows for programs to be developed in a mathematical environment. The idea is to free the user from the monotonous job of developing simple routines which are in common usage, and allow for the development of a more abstract kind of thinking for more refined mathematical operations. This matlab is stand alone ready to use package developed by Maths work Inc. U.S.A. It is equipeted with following tool box

(i) Control System Tool Box

(ii) Signal Processing Tool Box

(iii) System Identification Tool Box.

For more detail see Matlab mannual.

Similarly an another ready to use package is interfaced developed by danes inc. This package provide bulk of activity associated with controllers. This package is completely assisted with graphical facility.

The main aim of setting up this features is just to provide the flexibility to switch over from one software to another by just remaining in main menu.

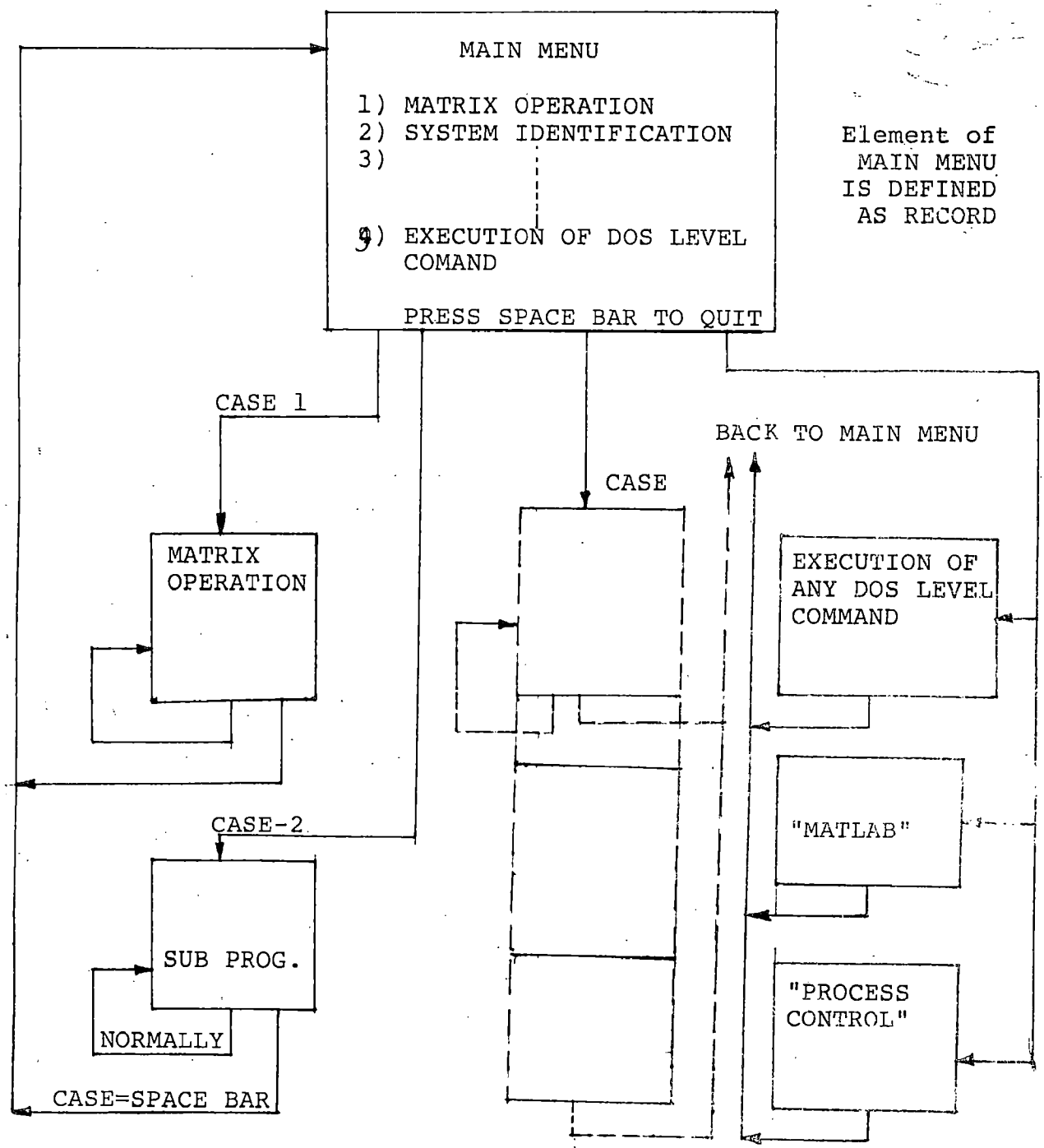
oooo

CHAPTER - VII

SOFTWARE STRUCTURE

STRUCTURE OF PACKAGE

This is the main program which automatically executes the subprogram described in earlier chapters. It is basically a premitive shell. The structure of Main menu is shown as follows.



When the power is first made on. The machine is under the control of MSDOS. The main menu of the developed software package is entered from MSDOS by executing the command MAIN.EXE. Once the PC is inside the developed software package one can go back to the MSDOS by pressing space bar once in main menu area.

Here the elements of main menu is defined as a record. Normally when no key is pressed, the main menu is displayed. As the key is pressed the respective sub menu is displayed. For example if key No.1 is pressed then menu of the subprogram is displayed. This operation is done using **case** statement of pascal. When the submenu is under display, with the respective choice depending upon the user, the corresponding program is executed. These programs are usually the '**exe**' version. And is directly executed using '**exec**' command provided in turbo pascal 5.5. Refer to Appendix-1 for the program of main menu. Hence menu program is a sort of premitive shell as shown in Fig.7.2.

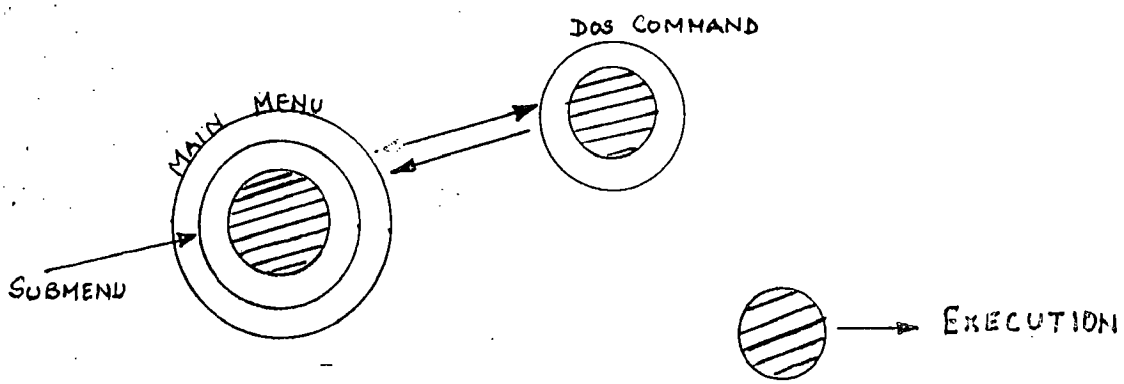


Fig. 7.2

Choice no.9 is kept for execution of any dos level command

directly from the main menu. Using this feature, editors can directly be loaded and further development is possible in the source code file if required, and can return to main menu after existing. This enhances the feature of this software for further development. Thus even turbopascal can be evoked from this menu. And after existing it again comes to main menu automatically.

This package uses many turbo pascal 5.5 unit files. The use of it is given in brief, as follow [10].

UNIT are

DOS The DOS unit implements a number of very useful operating system and file handling routines. The command associated with this unit is Exec, Swapvectors etc.

PRINTER The printer unit give a easy excess to the printer connected.

CRT The crt unit implements a range of powerfull routines that gives a full control to the PC's features, such as screen mode control, extended keyboard codes, colors, windows etc.

COMMON This unit is readily used in almost all the programs. It basically transfers the control while giving input either from keyboard or File and outputs the result to file, crt or printer.

FFTB2 This unit is used where fast fourier transforms are required. This unit basically evaluates farward and inverse FFT for the set of data.

Besides all this some of the turbopascal graphics units are also used like G-Driven, G-Shell, G-Kernel, Gwindow etc.[11].

In the source code listing in the Appendix the unit files used in that program.are listed under "USES".

Vijay.Inc. It is the file which contains the subroutine which was developed during this course of work and can further be used for future development.

The source code listing is presented in following way:

APPENDIX-I It is the source code listing of Menu program.

APPENDIX-II It is the source code listing of program of system identification.

APPENDIX-III It is the source code listing of transfer function simulation.

APPENDIX-IV It is the source code listing of FFT Application Program.

CONCLUSION

This package is developed with a view to provide a learning assistance to the students of control system. Each modules of this package has got its own limitations and pitfalls. Besides this, software engineering technique has not been used properly due to which exists the redundancy in source code, resulting in wastage of memory space. But their always exist the possibility of further development and optimization. This software package can be thought of as the stepping stone and a novel attempt in the field of computer aided learning of controls and process control operation. The package is always open for future development for its short comings and its enhancement from the comming generation of computer software engineers.

REFERENCE

1) N K Sinha and B.Kuszata: Text on Modelling and Identification of Dynamic System: VANNOSTRAND Publication, 1989.

2) Durbin: Fitting of Time Series Model, IEEE-Automatica, 1970.

3) Geusch:"Estimation of Autoregressive Parameter of a Mixed Autorequessive Moving Average Time Series: IEEE,Automatic Control, PP.83-589, Oct.1970.

4) R.K.Mehra: "On Line Identification of Linear Dynamic System with Application to Ralman Filtering", IEEE Automatic Comtrol Vol.16, PP.12-21 Feb.1971.

5) D.Graupe, D.J. Krause and J.B. Moore: "Identification of Auto Regressive Moving Average Parameter Time Series", IEEE, Automatic Control, Feb.1975.

6) M.G.Singh & M.S.Mahmand - Text on Large Scale System Modelling, (LIB).

7) Van Den Boom and Van Den Enden: "The determination of the Order of Process and Noise Dynamics, Automatica, Vol.10, PP. 245-256, 1974.

8) Newton Raiser and Gould: "Texton Stochactic Process" (LIB).

9) Cuenod & Durbing : Text on A Discrete Time Approach for System Analysis" (LIB).

10) Nagrath and Gopal: "A Test on Control System Engineering".

11) H.Dubner and D.J.Abate: "Numerical Inversion of Laplace Transforms by Relating them to Finite Fourier Transorm", J. ACM 15(1), 115-123, 1967.

12) F.Durbin: "Numerical Inversion of Laplace Transforms: An Effective Improvement of Dubner and Abate's Method", ACM, 17(4), 1973, 371-376.

13) K.S.Crump's Numerical Inversion of Laplace Transforms using a Fourier Series Approximations, J.AC,, 23(1), 1976, 89-96.

14) G.Honig & U.Hirdes: "A Method for Numerical Inversion of Laplace Transform Journal of Computational and Applied Mathematics", 10(1984), 113-132.

15) E.Krezig, "A Texton Engineering Mathematics" (LIB).

16) MATLAB MANNUAL

17) Roman Kuo:"A Text on Digital Signal Processing". [CH4-1]

18) Stanley: "A Text of Digital Signal Processing".

19) Turbo Pascal 5.5 - Mannual - of Borland Inc. 1989.

20) Turbo Pascal Graphics Tool Box - Mannual of Borland Inc. 1989.

oooo

APPENDICES

APPENDIX - I

PROGRAM FOR MAIN MENU

```
PROGRAM MAIN(INPUT,OUTPUT);

{-------------------------------------------------------------------}
{-      THE     MAIN     PROGRAM     FOR     MAIN     MENU          -}
{-------------------------------------------------------------------}

USES

  DOS,CRT,GDRIVER,GSHELL,GKERNEL,GWINDOW;

TYPE

  ANALYSIS = (MATRIXAPPLICATION,ROOTANALYSIS,LEASTSQUAREFIT,TRANSFERFSIMULA

              SYSTEMIDENTIFICATION, MATLABWORKSPACE,FFTAPPLICATIONS,
              LOOPANALYSIS,BLOCKDIAGSIMULA,PROCESSIMULA);

VAR

 WHICHANALYSIS              : ANALYSIS;
 PROGRAMNAME,CMDLINE        : STRING[79];
 CH                         : CHAR;
{-------------------------------------------------------------------------
}
PROCEDURE INTROD;
{-----------------------------------------------------}
{ PROCEDURE FOR INTRODUCTION OF THE SOFTWARE PACKAGE }
{-----------------------------------------------------}
 BEGIN
   INITGRAPHIC;
   DEFINEWINDOW(1,0,0,XMAXGLB,YMAXGLB);
   DEFINEWORLD(1,0,0,1000,1000);
   SELECTWORLD(1);
   SELECTWINDOW(1);
   DRAWBORDER;
   DRAWTEXTW(100,100,4,'  SYSTEM              IDENTIFICATION ');
   DRAWTEXTW(450,200,2,'            AND                      ');
   DRAWTEXTW(100,300,4,'  PROCESS             CONTROL        ');
   DRAWTEXTW(250,400,4,'        TOOL    BOX                  ');
   DRAWTEXTW(200,500,2,'BY');
   DRAWTEXTW(500,600,2,'VIJAY PANDE');
   DRAWTEXTW(200,700,2,'GUIDE');
   DRAWTEXTW(500,800,2,'PROF M.K.VASANTHA');
   DRAWTEXTW(200,900,2,'E.E.DEPT, UOR,  dt:14/1/1990');
   DRAWTEXTW(200,950,1,'PRESS ANY KEY TO CONTINUE ( EXCEPT ENTER KEY)');
   REPEAT UNTIL KEYPRESSED;
   LEAVEGRAPHIC;
   CLRSCR;
 END;
{-------------------------------------------------------------------------
-}
BEGIN {MAIN}
   INTROD;
   REPEAT
   INITGRAPHIC;
```

```
  DRAWBORDER;
  DEFINEWORLD(1,0,0,1000,1000);
  DEFINEWINDOW(1,0,0,1000,1000);
  DRAWTEXTW(100,100,3,'MAIN MENU');
  DRAWTEXTW(200,300,2,'1) MATRIX OPERATIONS                ');
  DRAWTEXTW(200,400,2,'2) LEAST SQUARE APPROXIMATION       ');
  DRAWTEXTW(200,450,2,'3) TRANSFER FUNCTION SIMULATION');
  DRAWTEXTW(200,500,2,'4) SYSTEM IDENTIFICATION            ');
  DRAWTEXTW(200,550,2,'5) EXECUTE DOS COMMANDS             ');
  DRAWTEXTW(200,600,2,'6) SS APROACH TO BLOCKD DIAGRAM');
  DRAWTEXTW(200,650,2,'7) PROCESS SIMULATION PACKAGE       ');
  DRAWTEXTW(200,700,2,'8) MATLAB WORK SPACE                ');
  DRAWTEXTW(200,750,2,'9) FFT APPLICATION PROGRAMS         ');
  DRAWTEXTW(100,850,2,'CHOOSE ANY OPTION BY PRESSING  THE KEY');
  DRAWTEXTW(50,950,2,'PRESS F1 FOR HELP  AND SPACE BAR TO QUIT');
    CH := READKEY;
    IF  KEYPRESSED THEN
       CH := READKEY;
    CASE CH OF
       '1' : BEGIN
               CLEARSCREEN;
               LEAVEGRAPHIC;
               WHICHANALYSIS := MATRIXAPPLICATION;
             END;
       '2' : BEGIN
               CLEARSCREEN;
               LEAVEGRAPHIC;
               WHICHANALYSIS := LEASTSQUAREFIT;
             END;
       '3' : BEGIN
               CLEARSCREEN;
               LEAVEGRAPHIC;
               WHICHANALYSIS := TRANSFERFSIMULA;
             END;
       '4' : BEGIN
               CLEARSCREEN;
               LEAVEGRAPHIC;
               WHICHANALYSIS := SYSTEMIDENTIFICATION;
             END;
       '5' : BEGIN
               CLEARSCREEN;
               LEAVEGRAPHIC;
               WHICHANALYSIS := LOOPANALYSIS;
             END;
       '6' : BEGIN
               CLEARSCREEN;
               LEAVEGRAPHIC;
               WHICHANALYSIS := BLOCKDIAGSIMULA;
             END;
       '7' : BEGIN
               CLEARSCREEN;
               LEAVEGRAPHIC;
               WHICHANALYSIS := PROCESSIMULA;
             END;
       '8' : BEGIN
```

```
                CLEARSCREEN;
                LEAVEGRAPHIC;
                WHICHANALYSIS := MATLABWORKSPACE;
              END;
        '9' : BEGIN
                CLEARSCREEN;
                LEAVEGRAPHIC;
                WHICHANALYSIS := FFTAPPLICATIONS;
              END;
    END; {CASE NO 1}
  UNTIL CH IN ['1'..'4'];
  WRITELN;
  CASE WHICHANALYSIS OF
    MATRIXAPPLICATION :BEGIN
                         REPEAT
                         WRITELN(' 1) DETERMINANT OF MATRIX');
                         WRITELN(' 2) INVERSE OF MATRIX');
                         WRITELN(' 3) EIGENVALUE/VECTOR OF MATRIX');
                         WRITELN(' 4) SOLUTION OF SIMULTANEOUS EQU');
                         WRITELN;
                         WRITE('SELECT A NUMBER(1-4)');
                            CH := READKEY;
                         UNTIL CH IN ['1'..'4'];
                         WRITELN;
                         CASE CH OF
                          '1' : BEGIN
                                  PROGRAMNAME := 'DET.EXE';
                                  CMDLINE := 'EXE';
                                END;
                          '2' : BEGIN
                                  PROGRAMNAME := 'INVERSE.EXE';
                                  CMDLINE := 'EXE';
                                END;
                          '3' : BEGIN
                                  PROGRAMNAME := 'WIELANDT.EXE';
                                  CMDLINE := 'EXE';
                                END;
                          '4' : BEGIN
                                  PROGRAMNAME := 'GAUSSIDL.EXE';
                                  CMDLINE := 'EXE';
                                END;
                         END;
                         SWAPVECTORS;
                         EXEC(PROGRAMNAME,CMDLINE);
                         SWAPVECTORS;
                        END;
    ROOTANALYSIS       : BEGIN
                         WRITELN(' 1)SECANT   METHOD      ');
                         WRITELN(' 2)LAGUERRE METHOD      ');
                         WRITELN(' 3)NEWTON-HORNER METHED');
                         WRITELN;
                         WRITE('SELECT A NUMBER(1-3)');
                         REPEAT
                            CH := READKEY;
                         UNTIL CH IN ['1'..'3'];
```

```
                    WRITELN;
                    CASE CH OF
                     '1' : BEGIN
                            PROGRAMNAME := 'SECANT.EXE';
                            CMDLINE := 'EXE';
                            SWAPVECTORS;
                            EXEC(PROGRAMNAME,CMDLINE);
                            SWAPVECTORS;
                           END;
                     '2' : BEGIN
                            PROGRAMNAME := 'LAGUERRE.EXE';
                            CMDLINE := 'EXE';
                            SWAPVECTORS;
                            EXEC(PROGRAMNAME,CMDLINE);
                            SWAPVECTORS;
                           END;
                     '3' : BEGIN
                            PROGRAMNAME := 'NEWTDEFL.EXE';
                            CMDLINE := 'EXE';
                            SWAPVECTORS;
                            EXEC(PROGRAMNAME,CMDLINE);
                            SWAPVECTORS;
                           END;
                    END;
                   END;
LEASTSQUAREFIT      :-BEGIN
                    REPEAT
                    WRITELN('1) INTERPOLATION/LAGRANGES');
                    WRITELN('2) INTERPOLATION/DIVIDE AND DIFFERENCE');
                    WRITELN('3) INTERPOLATION/CUBIC SPLINE');
                    WRITELN('4) INTERPOLATION/CLAMPED SPLINE');
                    WRITELN;
                    GOTOXY(20,10);
                    WRITELN('SELECT ANY NUMBER(1..4)');
                    CH := READKEY;
                    UNTIL CH IN ['1'..'4'];
                    WRITELN;
                    CASE CH OF
                     '1' : BEGIN
                             PROGRAMNAME := 'LAGRANGES.EXE';
                             CMDLINE := 'EXE';
                             SWAPVECTORS;
                             EXEC(PROGRAMNAME,CMDLINE);
                             SWAPVECTORS;
                           END;
                     '2' : BEGIN
                             PROGRAMNAME := 'DIVDIF.EXE';
                             CMDLINE := 'EXE';
                             SWAPVECTORS;
                             EXEC(PROGRAMNAME,CMDLINE);
                             SWAPVECTORS;
                           END;
                     '3' : BEGIN
                             PROGRAMNAME := 'CUBE_FRE.EXE';
                             CMDLINE := 'EXE';
```

```
                    SWAPVECTORS;
                    EXEC(PROGRAMNAME,CMDLINE);
                    SWAPVECTORS;
                  END;
          '4' : BEGIN
                    PROGRAMNAME := 'CUBE_CLA.EXE';
                    CMDLINE := 'EXE';
                    SWAPVECTORS;
                    EXEC(PROGRAMNAME,CMDLINE);
                    SWAPVECTORS;
                  END;
        END;
TRANSFERFSIMULA : BEGIN
        WRITELN(' 1) TRANSFER FUNC. TO S.SPACE');
        WRITELN(' 2) DISCRETE TF SIMULATION   ');
        WRITELN(' 3) CONTINOUS TF SIMULATION  ');
        WRITELN(' 4) INVERSE LAPLACE          ');
        WRITELN;
        WRITE('SELECT A NUMBER(1-4)');
        REPEAT
          CH := READKEY;
        UNTIL CH IN ['1'..'4'];
        WRITELN;
        CASE CH OF
          '1' : BEGIN
                  PROGRAMNAME := 'AJAY18.EXE';
                  CMDLINE := 'EXE';
                  SWAPVECTORS;
                  EXEC(PROGRAMNAME,CMDLINE);
                  SWAPVECTORS;
                END;
          '2' : BEGIN
                  PROGRAMNAME := 'AJAY15.EXE';
                  CMDLINE := 'EXE';
                  SWAPVECTORS;
                  EXEC(PROGRAMNAME,CMDLINE);
                  SWAPVECTORS;
                END;
          '3' : BEGIN
                  PROGRAMNAME := 'VIJAY.EXE';
                  CMDLINE := 'EXE';
                  SWAPVECTORS;
                  EXEC(PROGRAMNAME,CMDLINE);
                  SWAPVECTORS;
                END;
          '4' : BEGIN
                  PROGRAMNAME := 'VIJAY.EXE';
                  CMDLINE := 'EXE';
                  SWAPVECTORS;
                  EXEC(PROGRAMNAME,CMDLINE);
                  SWAPVECTORS;
                END;
        END;
      END;
YSTEMIDENTIFICATION : BEGIN
```

```
                    WRITELN(' 1)LSE METHOD (GSINGH)' );
                    WRITELN(' 2)FIR METHOD (VPANDE)' );
                    WRITELN(' 3)NEW METHOD (VPANDE)' );
                    WRITELN;
                    WRITE('SELECT A NUMBER(1-3)');
                    REPEAT
                      CH := READKEY;
                    UNTIL CH IN ['1'..'3'];
                    WRITELN;
                    CMDLINE := 'EXE';
                    CASE CH OF
                    '1' : BEGIN
                            PROGRAMNAME := 'AJAY10.EXE';
                          END;
                    '2' : BEGIN
                            PROGRAMNAME := 'AJAY30.EXE';
                          END;
                    '3' : BEGIN
                            PROGRAMNAME := 'VIJAY27.EXE';
                          END;
                    END;
                    SWAPVECTORS;
                    EXEC(PROGRAMNAME,CMDLINE);
                  END;
BLOCKDIAGSIMULA   : BEGIN
                    REPEAT
                    GOTOXY(5,5);
                    WRITELN('THIS METHOD FINDS       ');
                    WRITELN('STATE SPACE EQUIVALENT ');
                    WRITELN('AT ANY POINT IN A       ');
                    WRITELN('BLOCK DIAGRAM FOR A     ');
                    WRITELN('GIVEN INPUT AND OUTPUT ');
                    WRITELN;
                    WRITELN;
                    WRITELN('PRESS ANY KEY');
                    UNTIL KEYPRESSED;
                    PROGRAMNAME := 'VIJAY9.EXE';
                    CMDLINE := 'EXE';
                    SWAPVECTORS;
                    EXEC(PROGRAMNAME,CMDLINE);
                    SWAPVECTORS;
                  END;
LOOPANALYSIS      : BEGIN
                    PROGRAMNAME := 'AJAY6.EXE';
                    CMDLINE := 'EXE';
                    SWAPVECTORS;
                    EXEC(PROGRAMNAME,CMDLINE);
                    SWAPVECTORS;
                  END;
PROCESSIMULA      : BEGIN
                    PROGRAMNAME := 'START.BAT';
                    CMDLINE := 'BAT';
                    SWAPVECTORS;
                    EXEC(PROGRAMNAME,CMDLINE);
                    SWAPVECTORS;
```

```
                    END;
   MATLABWORKSPACE  : BEGIN
                       PROGRAMNAME := 'MATLAB';
                       CMDLINE := 'EXE';
                       SWAPVECTORS;
                       EXEC(PROGRAMNAME,CMDLINE);
                       SWAPVECTORS;
                      END;
   FFTAPPLICATIONS  : BEGIN
                       GOTOXY(5,5);
                       WRITELN('1). SHANNON RECONSTRUCTION');
                       WRITELN('2). FFT APPLICATIONS');
                       REPEAT
                        CH := READKEY;
                       UNTIL CH IN ['1'..'2'];
                       WRITELN;
                       CASE CH OF
                      '1' : BEGIN
                             PROGRAMNAME := 'VIJAYDEMO.EXE';
                             CMDLINE := 'EXE';
                             SWAPVECTORS;
                             EXEC(PROGRAMNAME,CMDLINE);
                             SWAPVECTORS;
                            END;
                      '2' : BEGIN
                             PROGRAMNAME := 'VIJAYPROGS.EXE';
                             CMDLINE := 'EXE';
                             SWAPVECTORS;
                             EXEC(PROGRAMNAME,CMDLINE);
                             SWAPVECTORS;
                            END;
                       END;
                      END;
                    END;
   LEAVEGRAPHIC;
END.
```

APPENDIX - II

PROGRAM FOR T.F.SIMULATION

```
PROGRAM TRANSFER_TO_SS(input,output);

uses
   dos,crt,common;

 type
  TNvector  = array[1..20] of real;
  TNmatrix  = array[1..20] of TNvector;

 VAR
   mord,p,q,r,s,t,u,v : integer;
   num,den : tnvector;
   Amat,Bmat,Cmat,Dmat : TNmatrix;

procedure GetData(var mord : integer;
                  var num,den  : TNvector);

{-------------------------------------------------------}
{- Output: mord,num,den                                 -}
{-                                                      -}
{- This procedure sets the value of order,denomi.,numer}
{-of TF from either keyboard input or file input        -}
{-------------------------------------------------------}

var
  Ch : char;

procedure GetDataFromKeyboard(var mord : integer;
                              var num,den  : TNvector);

{---------------------------------------}
{- Output: order,num,den                -}
{-                                      -}
{- This procedure sets the value of     -}
{- p and u from keyboard input -         }
{---------------------------------------}

var
  i : integer;

begin
  Writeln;
    repeat
       write('ORDER OF TRANSFER FUNCTION "MORD":');
       READLN(mord);
       IOcheck;
    until not IOerr;
    writeln;
    writeln('VALUES OF COIFFIECENT OF NUM AND DEN IN DECENDING POWER OF S')
;
    writeln;
    for i := 1 to mord+1 do
       repeat
          write('NUMERATOR[',i-1,']*S^(',i-1,') : ');
          readln(num[i]);
```

```
          IOcheck;
        until not IOerr;
        writeln;
    for i := 1 to mord+1 do
        repeat
            write('DENOMINATOR[',i-1,']*S^(',i-1,') : ');
            readln(den[i]);
            IOcheck;
        until not IOerr;
end; { procedure GetDataFromKeyboard }

procedure GetDataFromFile(var mord : integer;
                          var num,den  : TNvector);

{--------------------------------------}
{- Output : p , u                     -}
{-                                    -}
{- This procedure sets the value of   -}
{- Dimen and Data from file input     -}
{--------------------------------------}

var
  FileName : string[255];
  InFile : text;
   i : integer;

begin
  Writeln;
  repeat
    writeln;
    repeat
      Write('File name? ');
      Readln(FileName);
      Assign(InFile, FileName);
      Reset(InFile);
      IOCheck;
    until not IOerr;
    Readln(InFile,mord);
    for i := mord+1 to 1 do
      begin
         readln(infile,num[i],den[i]);
      end;
     IOCheck;
  until not IOerr;
end; { procedure GetDataFromFile }

begin { procedure GetData }
  case InputChannel('Input Data From') of
    'K' : GetDataFromKeyboard(mord,num,den);
    'F' : GetDataFromFile(mord,num,den);
  end;
  GetOutputFile(OutFile);
end; { procedure GetData }
{-------------------------------------------------------------------}
PROCEDURE tftoss(num,den : tnvector;mord : integer;
```

```
                var Amat,Bmat,Cmat,Dmat:Tnmatrix;
                var p,q,r,s,t,u,v :integer);
{-------------------------------------------------------}
{PROCEDURE to change TF to State space transformation }
{Method is based on the on algorithim of J.N.Little   }
{num = array of numerator;den = array of denominator  }
{nord,mord = order of numerator and denominator resp. }
{Amat,Bmat,Cmat,Dmat :  state space matrix             }
{p,q = matrix dimension of Amat                        }
{r,s = matrix dimension of Bmat                        }
{t,u = matrix dimension of Cmat                        }
{v   = dimension of Dmat                               }
{LIMITATION  Applicable only for SISO systems          }
{-------------------------------------------------------}
VAR
  i,j : integer;
  dden : TNvector;
BEGIN
  if mord = 0 then
   begin
      Amat[1,1] := 0;
      Bmat[1,1] := 0;
      Cmat[1,1] := 0;
      Dmat[1,1] := num[1]/den[1];
      p := 1;
      q := 1;
      r := 1;
      s := 1;
      t := 1;
      u := 1;
      v := 1;
   end
   else
   begin
    for i := 1 to mord do
     begin
        dden[i] := den[i]/den[mord+1];
     end;
    for i := 1 to mord do
     begin
        for j := 1 to mord do
          if i = mord
          then Amat[i,j] := -dden[j] else
          if j = i+1
          then Amat[i,j] := 1
          else
          Amat[i,j] := 0;
     end;
    for i := 1 to mord do
     begin
        if i = mord then
         Bmat[i,1] := 1
        else
         Bmat[i,1] := 0;
     end;
```

```
    for i := 2 to mord+1 do
     begin
        Cmat[1,i-1] := num[i-1]-num[mord+1]*dden[i-1];
     end;
    Dmat[1,1] := num[mord+1];
    p := mord;
    q := p;
    r := q;
    s := 1;
    t := 1;
    u := p;
    v := 1;
   end;
end; {end of procedure}
{-------------------------------------------------------------------------
--}
procedure Results(var p,q,r,s,t,u,v : integer;
                  var Amat,bmat,cmat,Dmat :tnmatrix);


{-------------------------------------------------------------------}
{- This procedure outputs the results to the device OutFile -}
{-------------------------------------------------------------------}

var
   i,j : integer;

begin
  Writeln(OutFile);
  Writeln(OutFile);
  Writeln(OutFile, 'ORDER OF A - MATRIX : ',P,'X',Q );
  writeln;
  writeln(outfile, 'A - MATRIX :');
  WRITELN;
  for i := 1 to p do
  begin
     for j := 1 to q do
         write(outfile,Amat[i,j] : 18:8);
    Writeln(OutFile);
  end;
  writeln(outfile, 'ORDER OF B - MATRIX : ',R,'X',S );
  writeln(outfile);
  writeln(outfile,'B - MATRIX;');
  writeln(outfile);
  for i := 1 to r do
    begin
      write(outfile,Bmat[i,s]);
      writeln(outfile);
    end;
  writeln(outfile,'C - MATRIX :');
  writeln(outfile);
  writeln(outfile,'ORDER OF C - MATRIX : ',T,'X',U);
  writeln(outfile);
  for i := 1 to u do
```

JAY18.PAS

```
  begin
    write(outfile,Cmat[t,i]);
  end;
  writeln(outfile);
  writeln(outfile,'D - MATRIX');
  writeln(outfile,'ORDER OF D - MATRIX : ',V);
  writeln(outfile,Dmat[1,1]);
  writeln(outfile);
end;
{-------------------------------------------------------------------------}
-}
begin {main}
  getdata(mord,num,den);
  tftoss(num,den,mord,Amat,Bmat,Cmat,Dmat,p,q,r,s,t,u,v);
  results(p,q,r,s,t,u,v,Amat,Bmat,Cmat,Dmat);
  close(outfile);
end.
```

```
PROGRAM DISC_SIMU_FOR_CONT_SYS;
USES
  DOS,CRT,COMMON,GDRIVER,GKERNEL,GWINDOW,GSHELL,PRINTER;

TYPE
   COMPLEX      = RECORD
                  RE,IM : REAL
                  END;
   INvector     = ARRAY[0..50] OF REAL;
   CARRAY       = ARRAY[1..20] OF COMPLEX;
   FLOAT        = REAL;
   ININTVECTOR = ARRAY[0..30] OF INTEGER;
CONST
  INNEARLYZERO = 1E-04;

VAR
  I,M,N,DEGREE,NUMROOTS,P                    : INTEGER;
  NUM,DEN,POLY,ROOT,IMAG,VALUE,DERIV         : INvector;
  BBA,AA,U,UU,YOUTPUT                        : INVECTOR;
  POLES,RESID                                : CARRAY;
  A,B                                        : PLOTARRAY;
  ITER                                       : ININTVECTOR;
  ERROR                                      : BYTE;
  RANGEOFTIME                                : REAL;
  CH,H                                       : CHAR;
  FALSE                                      : BOOLEAN;
{-------------------------------------------------------------------}
PROCEDURE add(x,y:complex;VAR z:complex);
            BEGIN
               z.re := x.re+y.re;
               z.im := x.im+y.im
            END;
{-------------------------------------------------------------------}
PROCEDURE sub(x,y:complex;VAR z:complex);
            BEGIN
               z.re := x.re-y.re;
               z.im := x.im-y.im
            END;
{-------------------------------------------------------------------}
PROCEDURE mul(x,y:complex;VAR z:complex);
            VAR q:complex;
            BEGIN
               q.re:= x.re*y.re - x.im*y.im;
               q.im:= x.re*y.im + x.im*y.re;
               z := q
            END;
{-------------------------------------------------------------------}
PROCEDURE dvd(x,y:complex;VAR z:complex);
            VAR q:complex;
                d:real;
            BEGIN
               d:=sqr(y.re)+sqr(y.im);
               q.re:=(x.re*y.re+x.im*y.im)/d ;
               q.im:=(x.im*y.re-x.re*y.im)/d ;
               z := q
```

```
          END;
{-------------------------------------------------------------------------}
FUNCTION mag(x:complex):real;
          BEGIN
             mag := sqrt(sqr(x.re)+sqr(x.im));
          END;
{-------------------------------------------------------------------------}

PROCEDURE valpoly(n:integer;var a:INvector;s:complex;var val:complex);
  {-------------------------------------------------------------------}
  {Procedure for finding the complex value of polynomial of the type}
  {given below                                                        }
  {   n = Degree of polynomial                                        }
  {   a = Array of coefficients of polynomial                         }
  {   s = complex value of argument of polynomial                     }
  { val = value of polynomial                                         }
  { The polynomial is of the type                                     }
  { a(0)+a(1)*s+a(2)*s^2+_ _ _ _ _ _ _ _ _ _ _ a(n)*s^n               }
  {-------------------------------------------------------------------}
  VAR
    i : integer;
    z : complex;
  BEGIN
    z.im := 0.0;
    val.re := a[n];
    val.im := 0.0;
    i := n;
    if n > 0 then
       repeat
         begin
           i := i-1;
           z.re := a[i];
           mul(val,s,val);
           add(val,z,val);
         end
       until i = 0
  end;
{-------------------------------------------------------------------------}


PROCEDURE difpoly(n:integer;var a:INvector;var m:integer;var b:INvector);

 {-------------------------------------------------------------------}
 {  Procedure for differentiating the polynomial                     }
 {  n  - Degree of ploynomial                                        }
 {  a  - Array of coefficients                                       }
 {  m  - Degree of differentated polynomial                          }
 {  b  - Array of corfficients of differentiated polynomial          }
 {  a(0)+a(1)*s+a(2)*s^2+----------a(n)*s^n                          }
 {-------------------------------------------------------------------}
  var
    i : integer;
  begin
    m := n-1;
    for i := 1 to n do
```

```
        b[i-1] := i*a[i];
  end;
{-------------------------------------------------------------------------
}

PROCEDURE parfracexp(m : integer; var a : INvector;n : integer;
                     var b : INvector;var p,r : carray);

 {-------------------------------------------------------------------------}
 {  m - Degree of numerator a(s)                                           }
 {  a - Array of coefficients of a(s)                                      }
 {  n - Degree of denominator b(s)                                         }
 {  b - Array of coefficients b(s)                                         }
 {  p - Complex array of poles of F(s)                                     }
 {  r - Complex array of residues                                          }
 {-------------------------------------------------------------------------}
 var
   i,k : integer;
   w,z : complex;
   c   : INvector;
 begin
   for i := 1 to n do
     begin
       valpoly(m,a,p[i],w);
       difpoly(n,b,k,c);
       valpoly(k,c,p[i],z);
       dvd(w,z,r[i])
     end;
 end;
{-------------------------------------------------------------------------
-}

PROCEDURE expc( x:complex;VAR z:complex);
{-----------------------------------------------}
{ POCEDURE for COMPLEX EXPONENTIAL FUNCTION     }
{-----------------------------------------------}
var r : real;
BEGIN
   r := exp(x.re);
   z.re := r*cos(x.im);
   z.im := r*sin(x.im);
END;

{-------------------------------------------------------------------------
--}

PROCEDURE INVLAPLACE(nt:integer; tr:real; n:integer;
                     var p,r:carray;var a:plotarray;VAR BBA,AA:INVECTOR);
{-----------------------------------------------------------------}
{ PROCEDURE for finding inverse laplace of rational function      }
{ nt - Number of time increments in the plot                      }
{ tr - Range of value of time                                     }
{ n  - degree of denominator polynomial of F(s)                   }
{ p  - complex array of poles locations                           }
{ r  - complex array of residues                                  }
```

```
a  - plotting array
-------------------------------------------------------------------------}
                                                                         }
ir
,dt,f : real;
,j    : integer;
t,ex  : complex;

GIN
 t  := 0.0;
 dt := tr/nt;
 for i := 0 to nt do
   begin
      f := 0.0;
      for j := 1 to n do
        begin
           if p[j].im > -1.0E-5 then
           if abs(p[j].im) < 1.0E-5
            then
            f := f+r[j].re*exp(p[j].re*t)
           else
            begin
               pt.re := p[j].re*t;
               pt.im := p[j].im*t;
               expc(pt,ex);
               mul(ex,r[j],ex);
               f := f+2.0*ex.re;
            end;
        end;
   a[i,1] := t;
   BBA[I] := t;
   a[i,2] := f;
   AA[I]  := f;
   t := t+dt;
 end;
;
```

```
edure Newt_Horn_Defl(InitDegree : integer;
                     InitPoly   : TNvector;
                     Guess      : FLOAT;
                     Tol        : FLOAT;
                     MaxIter    : integer;
                 var Degree     : integer;
                 var NumRoots   : integer;
                 var Poly       : TNvector;
                 var Root       : TNvector;
                 var Imag       : TNvector;
                 var Value      : TNvector;
                 var Deriv      : TNvector;
                 var Iter       : TNIntVector;
                 var Error      : byte);
```

```
  Denominator, Discrim : Float;

begin
  Denominator := 2 * A;
  ReRoot1 := -B / Denominator;
  ReRoot2 := ReRoot1;
  Discrim := B * B - 4 * A * C;
  if Discrim < 0 then
    begin
      ImRoot1 := -Sqrt(-Discrim) / Denominator;
      ImRoot2 :=  Sqrt(-Discrim) / Denominator;
    end
  else
    begin
      if B < 0 then  { Choose ReRoot1 to have the greatest absolute value }
        ReRoot1 := ReRoot1 + Sqrt(Discrim) / Denominator
      else
        ReRoot1 := ReRoot1 - Sqrt(Discrim) / Denominator;
      ReRoot2 := C / (A * ReRoot1);    { The product of the 2 roots is C/A }
      ImRoot1 := 0;
      ImRoot2 := 0;
    end;
end;     { procedure QuadraticFormula }

procedure TestData(InitDegree : integer;
                   InitPoly   : INvector;
                   Tol        : Float;
                   MaxIter    : integer;
                var Degree    : integer;
                var Poly      : INvector;
                var NumRoots  : integer;
                var Roots     : INvector;
                var yRoots    : INvector;
                var Iter      : INIntVector;
                var Error     : byte);

{-------------------------------------------------------------}
{- Input:  InitDegree, InitPoly, Tol, MaxIter                -}
{- Output: Degree, Poly, NumRoots, Roots, yRoots,            -}
{-         Iter, Error                                       -}
{-                                                           -}
{- This procedure sets the initial value of the above        -}
{- variables.  This procedure also tests the tolerance       -}
{- (Tol), maximum number of iterations (MaxIter), and        -}
{- Degree for errors and returns the appropriate error       -}
{- code.  Finally, it examines the coefficients of Poly.     -}
{- If the constant term is zero, then zero is one of the     -}
{- roots and the polynomial is deflated accordingly. Also    -}
{- if the leading coefficient is zero, then Degree is        -}
{- reduced until the leading coefficient is non-zero.        -}
{-------------------------------------------------------------}

var
  Term : integer;
```

```
begin
  Error := 0;
  NumRoots := 0;
  Degree := InitDegree;
  Poly := InitPoly;
  if Tol <= 0 then
    Error := 4;
  if MaxIter < 0 then
    Error :=5;
  { Reduce Degree until leading coefficient <> zero }
  while (Degree > 0) and (ABS(Poly[Degree]) < TNNearlyZero) do
    { Reduce Degree until leading coefficient <> zero }
    Degree  :=  Pred(Degree);
  if Degree <= 0 then
    Error := 3;
  { Deflate polynomial until the constant term <> zero }
  while (ABS(Poly[0]) < TNNearlyZero) and (Degree > 0) do
  begin
    NumRoots := Succ(NumRoots);
    Roots[NumRoots] := 0;
    yRoots[NumRoots] := 0;
    Iter[NumRoots] := 0;
    Degree := Pred(Degree);
    for Term := 0 to Degree do
      Poly[Term] := Poly[Term + 1];
  end;
nd; { procedure TestData }

rocedure FindValueAndDeriv(Degree : integer;
                           Poly   : TNvector;
                           X      : Float;
                       var Value  : Float;
                       var Deriv  : Float);

{-------------------------------------------------------------------------
- Input:  Degree, Poly, X                                                -}
- Output: Value, Deriv                                                   -}
-                                                                        -}
-                                                                        -}
- This procedure applies the technique of synthetic division to         -}
- determine both the Value and derivative of the polynomial, Poly,      -}
- at X.  The 0th element of the first synthetic division is the         -}
- value of the polynomial at X, and the 1st element of the second       -}
- synthetic division is the derivative of the polynomial at X.          -}
{-------------------------------------------------------------------------}

ar
 Poly1, Poly2 : TNvector;
gin
 SynDiv(Degree, Poly, X, Poly1);
 Value := Poly1[0];
 SynDiv(Degree, Poly1, X, Poly2);
 Deriv := Poly2[1];
d; { procedure FindValueAndDeriv }

ocedure FindOneRoot(Degree  : integer;
```

```
                    Poly     : TNvector;
                    Guess    : Float;
                    Tol      : Float;
                    MaxIter  : integer;
                var Root     : Float;
                var Value    : Float;
                var Deriv    : Float;
                var Iter     : integer;
                var Error    : byte);

{-------------------------------------------------------------------}
{- Input:  Degree, Poly, Guess, Tol, MaxIter                       -}
{- Output: Root, Value, Deriv, Iter, Error                         -}
{-                                                                 -}
{- A single root of the polynomial Poly.  The root must be         -}
{- approximated within MaxIter iterations to a tolerance of Tol.   -}
{- The root, value of the polynomial at the root (Value), and the  -}
{- value of the derivative of the polynomial at the root (Deriv),  -}
{- and the number of iterations (Iter) are returned. If no root    -}
{- is found, the appropriate error code (Error) is returned.       -}
{-------------------------------------------------------------------}

var
  Found : boolean;
  OldX, OldY, OldDeriv,
  NewX, NewY, NewDeriv : Float;

procedure CheckSlope(Slope : Float;
                 var Error : byte);

{-------------------------------------------------}
{- Input:  Slope                                  -}
{- Output: Error                                  -}
{-                                                -}
{- This procedure checks the slope to see if it is -}
{- zero.  The Newton Raphson algorithm may not be  -}
{- applied at a point where the slope is zero.     -}
{-------------------------------------------------}

begin
  if ABS(Slope) <= TNNearlyZero then
    Error := 2;
end; { procedure CheckSlope }

procedure Initial(Degree   : integer;
                  Poly     : TNvector;
                  Guess    : Float;
              var OldX     : Float;
              var OldY     : Float;
              var OldDeriv : Float;
              var Found    : boolean;
              var Iter     : integer;
              var Error    : byte);

{-------------------------------------------------------------}
```

```
{- Input:  Degree, Poly, Guess                                    -}
{- Output: OldX, OldY, OldDeriv, Found, Iter, Error               -}
{-                                                                -}
{- This procedure sets the initial values of the above            -}
{- variables. If OldY is zero, then a root has been               -}
{- found and Found = TRUE.                                        -}
{-----------------------------------------------------------------}

begin
  Found := false;
  Iter := 0;
  Error := 0;
  OldX := Guess;
  FindValueAndDeriv(Degree, Poly, OldX, OldY, OldDeriv);
  if ABS(OldY) <= TNNearlyZero then
    Found := true
  else
    CheckSlope(OldDeriv, Error);
end; { procedure Initial }

Function TestForRoot(X, OldX, Y, Tol : Float) : boolean;

{-----------------------------------------------------------------}
{ These are the stopping criteria.  Four different ones are       -}
{ provided.  If you wish to change the active criteria, simply    -}
{ comment off the current criteria (including the preceding OR)   -}
{ and remove the comment brackets from the criteria (including    -}
{ the following OR) you wish to be active.                        -}
{-----------------------------------------------------------------}

begin
  TestForRoot :=                          {---------------------------}
    (ABS(Y) <= TNNearlyZero)              {- Y=0                      -}
                                          {-                          -}
          or                              {-                          -}
                                          {-                          -}
    (ABS(X - OldX) < ABS(OldX*Tol))       {- Relative change in X     -}
                                          {-                          -}
 (*       or                          *) {-                          -}
 (*                                   *) {-                          -}
 (* (ABS(OldX - X) < Tol)             *) {- Absolute change in X     -}
 (*                                   *) {-                          -}
 (*       or                          *) {-                          -}
 (*                                   *) {-                          -}
 (* (ABS(Y) <= Tol)                   *) {- Absolute change in Y     -}
                                          {---------------------------}

{-----------------------------------------------------------------------}
{- The first criteria simply checks to see if the value of the         -}
{- function is zero.  You should probably always keep this criteria    -}
{- active.                                                             -}
{-                                                                     -}
{- The second criteria checks the relative error in X. This criteria   -}
{- evaluates the fractional change in X between interations. Note      -}
{- that X has been multiplied through the inequality to avoid divide   -}
```

```
 by zero errors.                                                    -}
                                                                    -}
 The third criteria checks the absolute difference in X between     -}
 iterations.                                                        -}
                                                                    -}
 The fourth criteria checks the absolute difference between         -}
 the value of the function and zero.                                -}
-------------------------------------------------------------------}

d; { procedure TestForRoot }

gin { procedure FindOneRoot }
Initial(Degree, Poly, Guess, OldX, OldY, OldDeriv, Found, Iter, Error);
while not(Found) and (Error = 0) and (Iter<MaxIter) do
begin
  Iter := Succ(Iter);
  NewX := OldX - OldY / OldDeriv;
  FindValueAndDeriv(Degree, Poly, NewX, NewY, NewDeriv);
  Found := TestForRoot(NewX, OldX, NewY, Tol);
  OldX := NewX;
  OldY := NewY;
  OldDeriv := NewDeriv;
  if not(Found) then
    CheckSlope(OldDeriv, Error);
end;
Root := OldX;
Value := OldY;
Deriv := OldDeriv;
if not(Found) and (Error = 0) and (Iter >= MaxIter) then
  Error := 1;
if Found then
  Error := 0;
d; { procedure FindOneRoot }

ocedure ReducePolynomial(var Degree : integer;
                         var Poly   : INvector;
                             Root   : Float);

-------------------------------------------------------}
 Input:  Degree, Poly, Root                            -}
 Output: Degree, Poly                                  -}
                                                       -}
 This procedure deflates the polynomial Poly by        -}
 factoring out the Root.  Degree is reduced by one.    -}
-------------------------------------------------------}

-
NewPoly : INvector;
Term : integer;
gin
SynDiv(Degree, Poly, Root, NewPoly);
Degree := Pred(Degree);
For Term := 0 to Degree do
  Poly[Term] := NewPoly[Term+1];
d; { procedure ReducePolynomial }
```

```
begin  { procedure Newt_Horn_Def1 }
  TestData(InitDegree, InitPoly, Tol, MaxIter,
           Degree, Poly, NumRoots, Root, Value, Iter, Error);
  while (Error=0) and (Degree>2) do
  begin
    FindOneRoot(Degree, Poly, Guess, Tol, MaxIter, Root[NumRoots+1],
                Value[NumRoots+1], Deriv[NumRoots+1], Iter[NumRoots+1], Err
or);
    if Error = 0 then
    begin
      NumRoots := Succ(NumRoots);
      {-------------------------------------------------------}
      {- The next statement refines the approximate root by -}
      {- plugging it into the original polynomial.  This    -}
      {- eliminates a lot of the round-off error            -}
      {- accumulated through many iterations                -}
      {-------------------------------------------------------}
      if NumRoots > 1 then
      begin
        Iter1 := 0;
        FindOneRoot(InitDegree, InitPoly, Root[NumRoots],
                    Tol, MaxIter, Root[NumRoots], Value[NumRoots],
                    Deriv[NumRoots], Iter1, Error);
        Iter[NumRoots] := Iter[NumRoots] + Iter1;
      end;
      ReducePolynomial(Degree, Poly, Root[NumRoots]);
      Guess := Root[NumRoots];
    end;
  end;
  case Degree of
    1 : begin            { Solve this linear }
          Degree := 0;
          NumRoots := Succ(NumRoots);
          Root[NumRoots] := -Poly[0] / Poly[1];
          FindOneRoot(InitDegree, InitPoly, Root[NumRoots], Tol,
                      MaxIter, Root[NumRoots], Value[NumRoots],
                      Deriv[NumRoots], Iter[NumRoots], Error);
        end;

    2 : begin            { Solve this quadratic }
          Degree := 0;
          NumRoots := Succ(Succ(NumRoots));
          QuadraticFormula(Poly[2], Poly[1], Poly[0],
                           Root[NumRoots - 1], Imag[NumRoots - 1],
                           Root[NumRoots], Imag[NumRoots]);
          if ABS(Imag[NumRoots]) < TNNearlyZero then
            { if the roots are real, they can be     }
            { made more accurate using Newton-Horner }
            begin
              FindOneRoot(InitDegree, InitPoly, Root[NumRoots-1], Tol,
                          MaxIter, Root[NumRoots-1], Value[NumRoots-1],
                          Deriv[NumRoots-1], Iter[NumRoots-1], Error);

              FindOneRoot(InitDegree, InitPoly, Root[NumRoots], Tol,
```

```
                              MaxIter, Root[NumRoots], Value[NumRoots],
                              Deriv[NumRoots], Iter[NumRoots], Error);
            end
          else
            { If the roots are complex, then assign    }
            { the value to be zero (which is true,     }
            { except for some roundoff error) and the  }
            { derivative to be zero (which is usually  }
            { FALSE; the derivative is usually complex }
            begin
              Value[NumRoots-1] := 0;     Value[NumRoots] := 0;
              Deriv[NumRoots-1] := 0;     Deriv[NumRoots] := 0;
              Iter[NumRoots-1] := 0;      Iter[NumRoots] := 0;
            end;
        end;
  end; { case }
end; { procedure Newt_Horn_Def1 }

{-------------------------------------------------------------------------}
procedure GetData( p : integer;
                   var u  : INvector);

{-------------------------------------------------------}
{- Output: Dimen, Data                                  -}
{-                                                      -}
{- This procedure sets the value of Dimen and Data      -}
{- from either keyboard input or file input             -}
{-------------------------------------------------------}

var
  Ch : char;

procedure GetDataFromKeyboard( p : integer;
                               var u  : INvector);

{---------------------------------------}
{- Output: p , u                       -}
{-                                     -}
{- This procedure sets the value of    -}
{- p and u from keyboard input -       }
{---------------------------------------}

var
  i : integer;

begin
  Writeln;
  for i := 0 to p do
      repeat
        Write('U[',i,'] :');
        Readln(u[i]);
        IOCHECK;
      until not IOerr;
   end; { procedure GetDataFromKeyboard }
```

```
rocedure GetDataFromFile( p : integer;
                          var u  : TNvector);

-----------------------------------------}
- Output : p , u                        -}
-                                       -}
- This procedure sets the value of      -}
- Dimen and Data from file input        -}
-----------------------------------------}

ar
 FileName : string[255];
 InFile : text;
  i : integer;

gin
Writeln;
repeat
  Writeln;
  repeat
    Write('File name? ');
    Readln(FileName);
    Assign(InFile, FileName);
    Reset(InFile);
    IOCheck;
  until not IOerr;
    for i := 0 to p do
    begin
      Read(InFile,u[i]);
      IOCheck;
    end;
      iocheck;
ntil not IOerr;
l; { procedure GetDataFromFile }

in { procedure GetData }
ase InputChannel('Input Data From') of
  'K' : GetDataFromKeyboard(p,u);
  'F' : GetDataFromFile(p,u);
nd;
; { procedure GetData }
-------------------------------------------------------------------------------}
cedure  CONVOLUTION(b,U : tnvector;m,q : integer;var y : tnvector);
-----------------------------------------------}
PROCEDURE for filter implementation (generalised)}
a,b - array of coefficients of numerator and     }
                                        denominator}
n,m - order of numerator and denominator         }
u[k]- discret input array to be read             }
y[k]- discret output array                       }
H(z) = b(0)+b(1)*z^-1+b(2)*z^-2+----b(m)*z^-m    }
       ---------------------------------------   }
        1 + a(1)*z^-1+a(2)*z^-2+----a(n)*z^-n    }
                                                 }
CAUTION : Transfer function should of above      }
```

```
{                    Type.
{-------------------------------------------------------}
AR
 k,i : integer;
   r : tnvector;
egin
   for k := 0 to q do
     begin
        r[k] := 0.0;
        for i := 0 to m do
          begin
            r[k] := r[k] + b[i]*u[k-i+q];
          end;
        y[k] := r[k];

     end;
end;
-------------------------------------------------------------------------------}
ocedure SPLINE_INT2(A,C:PLOTARRAY;N:INTEGER);
r
X, Temp : Float;
1, Dx, Dy, I, J,  Lines, Scale : integer;
X1, Y1, X2, Y2 : integer;
3 : PlotArray;
gin
:learScreen;
:etColorWhite;
:otoXY(50, 25);
lx := -8;
ly := 7;
.1 := 3;
'1 := 5;
2 := 25;
2 := 10;
ines := 0;
cale := 0;
efineWindow(1, 0, 0, XMAXGLB,YMAXGLB);
efineWindow(2, 0,0,XMAXGLB,YMAXGLB);
EFINEWORLD(1,0,0,1000,1000);
efineHeader(2, ' INPUT-OUTPUT TIME RESPONSE');
:THeaderON;
 := 64;
oline(A, N, A[2, 1], A[N - 1, 1], B, M);
.ndWorld(2, B, M, 1, 1.08);
:lectWindow(2);
'awBorder;
:TLINESTYLE(1);
awAxis(Dx, Dy, X1, Y1, X2, Y2, Lines, Scale, false);
awPolygon(A, 2, N - 1, 7, 2, 0);
TLINESTYLE(0);
AWAXIS(0,0,X1,Y1,X2,Y2,0,0,FALSE);
AWPOLYGON(B,1,-M,0,0,0);
TLINESTYLE(1);
AWAXIS(DX,DY,X1,Y1,X2,Y2,0,0,FALSE);
AWPOLYGON(C,2,N-1,7,2,0);
```

```
  SELECTWORLD(1);
  SELECTWINDOW(1);
  DrawTextW(730, 50, 2,'TIME');
  DRAWTEXTW(750,100,2,'RESPONSE');
  DRAWTEXTW(730,200,1,'THE SCALE IS');
  DrawTextW(730, 250,1,'X-AXIS IS TIME AXIS');
  DRAWTEXTW(730,300,1,'Y-AXIS IS MAGNITUDE');
  DRAWTEXTW(730,450,2,'...IS INPUT');
  DRAWTEXTW(730,550,2,'--IS OUTPUT');
  DRAWTEXTW(730,800,1,'PRESS ANY KEY OR');
  DRAWTEXTW(730,900,1,'PRESS H FOR HARDCOPY');
 END;
{-------------------------------------------------------------------------
-}
procedure SPLINE_INT1(A:PLOTARRAY;N:INTEGER);
var
  X, Temp : Float;
  M, Dx, Dy, I, J,  Lines, Scale : integer;
  X1, Y1, X2, Y2 : integer;
  B : PlotArray;
begin
  ClearScreen;
  SetColorWhite;
  GotoXY(50, 25);
  Dx := -8;
  Dy := 7;
  X1 := 3;
  Y1 := 5;
  X2 := 25;
  Y2 := 10;
  Lines := 0;
  Scale := 0;
  DefineWindow(1, 0, 0, XMAXGLB,YMAXGLB);
  DefineWindoW(2, 0,0,XMAXGLB,YMAXGLB);
  DEFINEWORLD(1,0,0,1000,1000);
  DefineHeader(2, ' INVERSE LAPLACE');
  SETHeaderON;
  M := 64;
  Spline(A, N, A[2, 1], A[N - 1, 1], B, M);
  FindWorld(2, B, M, 1, 1.08);
  SelectWindow(2);
  DrawBorder;
  SETLINESTYLE(1);
  DrawAxis(Dx, Dy, X1, Y1, X2, Y2, Lines, Scale, false);
  DrawPolygon(A, 2, N - 1, 7, 2, 0);
  SETLINESTYLE(0);
  DRAWAXIS(0,0,X1,Y1,X2,Y2,0,0,FALSE);
  DRAWPOLYGON(B,1,-M,0,0,0);
  SELECTWORLD(1);
  SELECTWINDOW(1);
  DrawTextW(730, 100, 2,'TIME');
  DRAWTEXTW(740,200,2,'RESPONSE');
  DRAWTEXTW(730,260,1,'THE SCALE IS');
  DrawTextW(730, 300,1,'X-AXIS IS TIME AXIS');
  DRAWTEXTW(730,400,1,'Y-AXIS IS MAGNITUDE');
```

```
  DRAWTEXTW(730,500,1,'..THE INVERSE LAPLACE');
  DrawTextW(730, 600, 1, 'THE TIME FOR OUTPUT SIMULATION');
  DrawTextW(730,700,1,'IS THE RANGE OF TIME');
  DRAWTEXTW(730,800,1,'PRESS ANY KEY EXCEPT ENTER KEY');
  DRAWTEXTW(730,900,1,'PRESS H FOR HARD COPY ');
 END;
{-------------------------------------------------------------------------
}
BEGIN {MAIN}
   INITGRAPHIC;
   DEFINEWINDOW(1,0,0,XMAXGLB,YMAXGLB);
   DEFINEWORLD(1,0,0,1000,1000);
   SELECTWORLD(1);
   SELECTWINDOW(1);
   DRAWBORDER;
   DRAWTEXTW(200,100,4,'  TRANSFER    FUNCTION');
   DRAWTEXTW(250,200,4,'    SIMULATION');
   DRAWTEXTW(100,300,4,'FOR   ARBITRARY   INPUT');
   DRAWTEXTW(200,400,2,'BY');
   DRAWTEXTW(500,500,2,'VIJAY PANDE');
   DRAWTEXTW(200,600,2,'GUIDE');
   DRAWTEXTW(500,700,2,'PROF M.K.VASANTHA');
   DRAWTEXTW(200,800,2,'E.E.DEPT, UOR,  dt:14/1/1990');
   DRAWTEXTW(200,900,1,'PRESS ANY KEY TO CONTINUE ( EXCEPT ENTER KEY)');
   REPEAT UNTIL KEYPRESSED;
   LEAVEGRAPHIC;
   CLRSCR;
   WRITELN;
   WRITE('RANGE OF TIME FOR WHICH SIMULATION TO BE CARRIED ');
   READLN(RANGEOFTIME);
   WRITELN('NUMBER OF DISCRETE POINTS YOU WANT TO EVALUATE : ');
   WRITE('               SHOULD BE INRANGE OF 10 TO 30      : ');
   READLN(P);
   WRITE('ORDER OF NUMERATOR m? :');
   READLN(M);
   FOR I := M DOWNTO 0  DO
    BEGIN
       WRITE('COEFFICIENT OF NUMERATOR S^',I,' IS: ');
       READLN(NUM[I]);
    END;
   WRITELN;
   WRITE('ORDER OF DENOMINATOR n ? :');
   READLN(N);
   FOR I := N DOWNTO 0 DO
    BEGIN
       WRITE('COEFFICIENT OF S^',I,' OF DENOMINATOR IS ?:');
       READLN(DEN[I]);
    END;
   NEWT_HORN_DEFL(N,DEN,0,1E-03,80,DEGREE,NUMROOTS,POLY,ROOT,IMAG,VALUE,
                  DERIV,ITER,ERROR);
   WRITELN('*****THE POLES ARE*****');
   FOR I := 1 TO N DO
    BEGIN
       POLES[I].RE := ROOT[I];
       POLES[I].IM := IMAG[I];
```

```
        WRITE(POLES[I].RE,'+J',POLES[I].IM);
        WRITELN;
      END;
    PARFRACEXP(M,NUM,N,DEN,POLES,RESID);
    WRITELN;
    WRITELN('******THE RESIDUES ARE******');
    FOR I := 1 TO N DO
     BEGIN
        WRITE(RESID[I].RE,'+J',RESID[I].IM);
        WRITELN;
     END;
    INVLAPLACE(P,RANGEOFTIME,N,POLES,RESID,A,BBA,AA);
    INITGRAPHIC;
    SPLINE_INT1(A,P);
    REPEAT UNTIL KEYPRESSED;
    CH := READKEY;
    IF (CH = H) THEN HARDCOPY(FALSE,4);
    LEAVEGRAPHIC;
    CLRSCR;
    GOTOXY(5,5);
    WRITELN('NOW YOU HAVE TO ENTER INPUT DATA U(t)');
    GETDATA(P,U);
    FOR I := 0 TO 2*P DO
     BEGIN
        IF I <= P THEN
        UU[I] := U[I]
        ELSE
        UU[I] := U[I-P];
     END;
   CLRSCR;
   CONVOLUTION(AA,UU,P,P,YOUTPUT);
   FOR I := 0 TO P DO
    BEGIN
      A[I+1,2] := YOUTPUT[I];
      B[I+1,1] := BBA[I];
      B[I+1,2] := U[I]/DEN[0];
    END;
   INITGRAPHIC;
   SPLINE_INT2(A,B,P);
   REPEAT UNTIL KEYPRESSED;
   CH := READKEY;
   IF (CH = H) THEN HARDCOPY(FALSE,4);
   LEAVEGRAPHIC;
   FOR I := 0 TO P DO
    BEGIN
      WRITELN(A[I+1,1],' ',A[I+1,2],' ',B[I+1,2]);
    END;
END.
```

```
PROGRAM filter(input,output);

 uses
   dos,crt,common;

 type
  TNvector  = array[1..50] of real;
  var
   p,n,m : integer;
    u,a,b,y : tnvector;

)rocedure GetData(var p,n,m : integer;
                  var u,a,b  : TNvector);

{-------------------------------------------------------}
{- Output: Dimen, Data                                  -}
{-                                                      -}
{- This procedure sets the value of Dimen and Data      -}
{- from either keyboard input or file input             -}
{-------------------------------------------------------}

ar
 Ch : char;

rocedure GetDataFromKeyboard(var p,n,m : integer;
                             var u,a,b  : TNvector);

{---------------------------------------}
{- Output: p , u                       -}
{-                                     -}
{- This procedure sets the value of    -}
{- p and u from keyboard input -       }
{---------------------------------------}

ir
 i : integer;

gin
Writeln;
repeat
   Write('input number of samples p ');
   Readln(p);
   writeln('order of numerator (m)');
   write('m = ');
   readln(m);
   writeln('coefficients of numerator');
   for i := 1 to m+1 do
   begin
      write('b[',i-1,']=');
      readln(b[i]);
   end;
   writeln('order of denominator(n):');
       write('n=');
       readln(n);
```

```
   writeln('coefficients of denominator');
   for i := 1 to n do
   begin
     write('a[',i,']=');
     readln(a[i]);
   end;

   IOCheck;
 until (not IOerr);
 Writeln;
 for i := 1 to p+10 do
     repeat
       Write('U[',i,'] :');
       Readln(u[i]);
       IOCheck;
     until not IOerr;
nd; { procedure GetDataFromKeyboard }

rocedure GetDataFromFile(var p,n,m : integer;
                         var u,a,b  : TNvector);

-----------------------------------------}
- Output : p , u                         -}
-                                        -}
- This procedure sets the value of       -}
- Dimen and Data from file input         -}
-----------------------------------------}

ar
 FileName : string[255];
 InFile : text;
  i : integer;

egin
 Writeln;
 repeat
   Writeln;
   repeat
     Write('File name? ');
     Readln(FileName);
     Assign(InFile, FileName);
     Reset(InFile);
     IOCheck;
   until not IOerr;
   Read(InFile,p);
   read(infile,m);
   for i := 1 to m+1 do
     begin
        readln(infile,b[i]);
     end;
   read(infile,n);
   for i := 1 to n do
   begin
      readln(infile,a[i]);
   end;
```

```
    IOCheck;
    while (not IOerr) and (n>m) do
    begin
      for i := 1 to p+10 do
      begin
        Read(InFile,u[i]);
        IOCheck;
      end;
    end;
  until not IOerr;
end; { procedure GetDataFromFile }

begin { procedure GetData }
  case InputChannel('Input Data From') of
    'K' : GetDataFromKeyboard(p,m,n,b,a,u);
    'F' : GetDataFromFile(p,m,n,b,a,u);
  end;
  GetOutputFile(OutFile);
end; { procedure GetData }
{-------------------------------------------------------------------------}
procedure Results( q    : integer;
                   var U,Y   : INvector);


{-------------------------------------------------------------}
{- This procedure outputs the results to the device OutFile -}
{-------------------------------------------------------------}

var
   Row : integer;

begin
  Writeln(OutFile);
  Writeln(OutFile);
  Writeln(OutFile,'    THE INPUT      AND          OUTPUT');
  for Row := 10 to q+10 DO
  begin
        Write(OutFile,U[Row] ,Y[ROW] :18:8);
    Writeln(OutFile);
  end;
end;
{-------------------------------------------------------------------------}
procedure dfilter(a,b : tnvector; q,n,m : integer;var u,y : tnvector);
 {-----------------------------------------------}
 { PROCEDURE for filter implementation (generalised)}
 { a,b - array of coefficients of numerator and     }
 {                                  denominator}
 { n,m - order of numerator and denominator         }
 { u[k]- discret input array to be read             }
 { y[k]- discret output array                       }
 { H(z) = b(0)+b(1)*z^-1+b(2)*z^-2+----b(m)*z^-m    }
 {        ---------------------------------         }
 {         1 + a(1)*z^-1+a(2)*z^-2+----a(n)*z^-n    }
 {                    - -                           }
```

```
  { CAUTION : Transfer function should of above       }
  {                   Type.                           }
  {---------------------------------------------------}
VAR
  k,i : integer;
  p,r : tnvector;
 begin
    for k := 10 to q+10 do
      begin
         p[k] := 0.0;
         for i := 1 to n do
           begin
              p[k] := p[k]-a[i]*y[k-i];
           end;
         r[k] := 0.0;
         for i := 1 to m+1 do
           begin
              r[k] := r[k] + b[i]*u[k-i+1];
           end;
         y[k] := r[k] +p[k];
      end;
  end;

begin {main}
   getdata(p,m,n,b,a,u);
   dfilter(a,b,p,n,m,u,y);
   Results(p,u,y);
   close(outfile);
end.
```

APPENDIX - III

PROGRAM FOR SYSTEM IDENTIFICATION

```
program identil(input,output);
 {-------------------------------------------------------------------}
 { Programm for identification of parameter using LSE approximation }
 { The input is Read from a file                                     }
 { The inputs are                                                    }
 {             1. Number of samples       N                        }
 {             2. Input to the system     U(k)                     }
 {             3. Output of the system    Y(k)                     }
 { No error handling is done, So be careful while inputing          }
 {-------------------------------------------------------------------}
const
   TNNearlyzero = 1E-07;
type
   TNvector = array[1..30] of real;
   TNmatrix = array[1..30] of TNvector;
   Float    = real;
var
   m,n,p,r : integer;
   y,u,yp,upr  : TNvector;
   i,j,k   : integer;
   hmat,pmat,zmat,Ymat : TNmatrix;
   zmata,zmatb,zmatc,zzmat :TNmatrix;
   filename  : string[255];
   infile    : text;
   error     : byte;
{-------------------------------------------------------------------------
---}
procedure Inverse(Dimen : integer;
                  Data   : TNmatrix;
              var Inv    : TNmatrix;
              var Error  : byte);


procedure Initial(Dimen : integer;
              var Data  : TNmatrix;
              var Inv   : TNmatrix;
              var Error : byte);

{-------------------------------------------------------}
{- Input: Dimen, Data                                  -}
{- Output: Inv, Error                                  -}
{-                                                     -}
{- This procedure test for errors in the value of Dimen -}
{-------------------------------------------------------}

var
  Row : integer;

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    begin
      { First make the inverse-to-be the identity matrix }
```

```
      FillChar(Inv, SizeOf(Inv), 0);
      for Row := 1 to Dimen do
        Inv[Row, Row] := 1;
      if Dimen = 1 then
        if ABS(Data[1, 1]) < INNearlyZero then
          Error := 2   { Singular matrix }
        else
          Inv[1, 1] := 1 / Data[1, 1];
    end;
end; { procedure Initial }

procedure EROdiv(Divisor : Float;
                 Dimen   : integer;
             var Row     : INvector);

{-------------------------------------------------------------}
{- Input: Divisor, Dimen, Row                                  -}
{-                                                             -}
{- elementary row operation - dividing by a constant          -}
{-------------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    Row[Term] := Row[Term] / Divisor;
end; { procedure EROdiv }

procedure EROswitch(var Row1 : INvector;
                    var Row2 : INvector);

{-------------------------------------------------------------}
{- Input: Row1, Row2                                           -}
{- Output: Row1, Row2                                          -}
{-                                                             -}
{- Elementary row operation - switching two rows              -}
{-------------------------------------------------------------}

var
 DummyRow : INvector;

begin
 DummyRow := Row1;
 Row1 := Row2;
 Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : Float;
                     Dimen        : integer;
                 var ReferenceRow : INvector;
                 var ChangingRow  : INvector);

{-------------------------------------------------------------}
 Input: Multiplier, Dimen, ReferenceRow, ChangingRow          -}
```

```
{- Output: ChangingRow                                                  -}
{-                                                                      -}
{- Row operation - adding a multiple of one row to another              -}
{-----------------------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
end; { procedure EROmultAdd }


procedure Invert(Dimen : integer;
                 var Data  : TNmatrix;
                 var Inv   : TNmatrix;
                 var Error : byte);

{-----------------------------------------------------------------------}
{- Input: Dimen, Data                                                  -}
{- Output: Inv, Error                                                  -}
{-                                                                     -}
{- This procedure computes the inverse of the matrix Data              -}
{- and stores it in the matrix Inv.  If the matrix Data                -}
{- is singular, then Error = 2 is returned.                            -}
{-----------------------------------------------------------------------}

var
  Divisor, Multiplier : Float;
  Row, ReferenceRow : integer;

procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Data         : TNmatrix;
            var Inv          : TNmatrix;
            var Error        : byte);

{-----------------------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Data, Inv                               -}
{- Output: Data, Inv, Error                                            -}
{-                                                                     -}
{- This procedure searches the ReferenceRow column of                  -}
{- the Data matrix for the first non-zero element below                -}
{- the diagonal. If it finds one, then the procedure                   -}
{- switches rows so that the non-zero element is on the                -}
{- diagonal. This same operation is applied to the Inv                 -}
{- matrix. If no non-zero element exists in a column, the              -}
{- matrix is singular and no inverse exists.                           -}
{-----------------------------------------------------------------------}

var
  NewRow : integer;

begin
```

```
  Error := 2;  { No inverse exists }
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do
  { Try to find a        }
  { row with a non-zero }
  { diagonal element    }
  begin
    NewRow := Succ(NewRow);
    if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Data[NewRow], Data[ReferenceRow]);
      { Switch these two rows }
      EROswitch(Inv[NewRow], Inv[ReferenceRow]);
      Error := 0;
    end;
  end; { while }
end; { procedure Pivot }

begin { procedure Inver }
  { Make Data matrix upper triangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Check to see if the diagonal element is zero }
    if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Data, Inv, Error);
    if Error = 0 then
    begin
      Divisor := Data[ReferenceRow, ReferenceRow];
      EROdiv(Divisor, Dimen, Data[ReferenceRow]);
      EROdiv(Divisor, Dimen, Inv[ReferenceRow]);
      for Row := 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if (Row <> ReferenceRow) and
           (ABS(Data[Row, ReferenceRow]) > TNNearlyZero) then
        begin
          Multiplier := -Data[Row, ReferenceRow] /
                         Data[ReferenceRow, ReferenceRow];
          EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
          EROmultAdd(Multiplier, Dimen, Inv[ReferenceRow], Inv[Row]);
        end;
    end;
  end;
end; { procedure Inver }

begin { procedure Inverse }
  Initial(Dimen, Data, Inv, Error);
  if Dimen > 1 then
    Inver(Dimen, Data, Inv, Error);
end; { procedure Inverse }
{-------------------------------------------------------------------}
procedure multi(dr1,dc1,dr2,dc2:integer;
                vij1:TNmatrix;vij2:TNmatrix;var vij3:TNmatrix);
 var
```

```
    i,j,k : integer;

    {-------------------------------------------------------------------}
    {-Input :di=number of Row of vij1:dr=number of column of vij2       }
    {-Output:Matrix vij3 of di row and dr column                        }
    {                                                                   }
    { this procedure evalutes the multiplication of two matrix          }
    {                                                                   }
    { Be careful -  error dealing is not their                          }
    {-------------------------------------------------------------------}
    begin
      for i := 1 to dr1 do
        begin
          vij3[i,j] := 0.0;
          for j := 1 to dc2 do
           begin
            for k := 1 to dr2 do
              vij3[i,j] := vij3[i,j]+vij1[i,k]*vij2[k,j];
            end;
          end;
     end;
{-------------------------------------------------------------------}
{-------------------------------------------------------------------}
begin {main program begins now}
 writeln('input number of samples using - N');
 writeln('order of zeros you want to have - m');
 writeln('order of system you want to have - n');
 readln(N,m,n);
 writeln('N= ',p,'m= ',m,'n= ',n);
 writeln('Read Sampled data from file');
 writeln('filename?  ');
 readln(filename);
 assign(infile,filename);
 reset (infile);
 for r := 1 to p do
  begin
      readln(infile,yp[r],upr[r]);
  end;
  close (infile);
 for r := 1 to 2*p do
  begin
     if r <= p then
     begin
        y[r] := 0.0;
        u[r] := 0.0;
     end
     else
     begin
        y[r] := yp[r-p];
        u[r] := upr[r-p];
     end;
   end;
 for i := 1 to p DO
   begin
     for j := 1 to (n+m+1) DO
```

```
        begin
          if j <= n then
           begin
             hmat[i,j] := -y[p-j+i];
             pmat[j,i] := hmat[i,j];
           end
          ELSE
           BEGIN
             hmat[i,j] := u[i-j+1+p-n];
             pmat[j,i] := hmat[i,j];
           END;
        end;
    end;
 for i := 1 to p DO
    begin
      Ymat[i,1] := y[i+1+k];
    end;
multi(n+m+1,p,p,n+m+1,pmat,hmat,zmata);
multi(n+m+1,p,p,1,pmat,Ymat,zmatb);
inverse(m+n+1,zmata,zmatc,error);
multi(n+m+1,n+m+1,n+m+1,1,zmatc,zmatb,zzmat);
writeln('coefficients = ');
for i := 1 to n+m+1 do
  BEGIN
   writeln(zzmat[i,1]:18:3);
   IF I <= N THEN
     WRITELN('A[',I-1,'] = ',zzmat[I,1])
   ELSE
     WRITELN('B[',I-N,'] = ',zzmat[I,1]);
  END;
end.
```

```
PROGRAM  IDENTI_VIJAY_COMBINED(INPUT,OUTPUT);

{-------------------------------------------------------}
{- THIS IS MY OWN METHOD FOR IDENTIFING THE SYSTEM IN   -}
{- DISCRETE DOMAIN. IT UTILISES BOTH THE CLASSICAL AND  -}
{- DECONVOLUTION TECHNIQUE FOR THE NOISE FREE SYSTEMS   -}
{- AND FOR NORMALLY DISTRIBUTED NOISE IT USES BOTH      -}
{- AUTOCORRELATION AND CROSSCORELATION TECHNIQUE AND    -}
{- DECONVOLUTION IS PERFORMED AND AFTER THAT CLASSICAL  -}
{- TECHNIQUE IS APPLIED                                 -}
{- FOR MORE DETAIL REFER DISSERTATION REPORT OF         -}
{- VIJAY PANDE EE DEPT. U.O.R ROORKEE                   -}
{-------------------------------------------------------}

USES
  DOS,CRT,COMMON;

TYPE


  INVECTOR              = ARRAY[0..50] OF REAL;
  INVECTOR1             = ARRAY[0..20] OF REAL;
  INMATRIX              = ARRAY[0..20] OF INVECTOR1;
  INVECTOR2             = ARRAY[1..20] OF REAL;
  INMATRIX1             = ARRAY[1..20] OF INVECTOR2;
  FLOAT                 = REAL;
CONST

  INNEARLYZERO          = 1E-04;

VAR
  P,ORDER,PP,I,K,MN,XYZ          : INTEGER;
  RANGEOFTIME,VALUE,XVALUE,QQ,YY: REAL;
  WELEMENT                       : INVECTOR;
  BROWVECT                       : INVECTOR1;
  BMATRIX,WARRAY,AVECTOR         : INMATRIX;
  WVECTOR,BARRAY,DVALUE,HVALUE   : INVECTOR2;
  HANKELMAT                      : INMATRIX1;
  ERROR1,ERROR2                  : BYTE;
  OUTFILE                        : TEXT;
{-------------------------------------------------------------------------}
PROCEDURE  DECONVOLUTION(BROW,AROW:INVECTOR;N,M:INTEGER;VAR GROW:INVECTOR);

{-------------------------------------------------}
{ PROCEDURE FOR DECONVOLUTION OF TWO ROW VECTOR -}
{-b(0),b(1),b(2),------ is first row vector     -}
{- of N coefficients                            -}
{-a(0),a(1),a(2),------ is second row vector    -}
{- of M coefficients                            -}
{-------------------------------------------------}

VAR
  I,J     : INTEGER;
  GGROW   : INVECTOR;
  PP      : REAL;
```

```
BEGIN
   FOR I := 0 TO N+M DO
   BEGIN
      IF I <= N THEN
      BROW[I] := BROW[I]/AROW[0]
      ELSE
      BROW[I] := 0.0;
   END;
   FOR I := 0 TO M+N DO
   BEGIN
      IF I <= M THEN
      AROW[I] := AROW[I]/AROW[0]
      ELSE
      AROW[I] := 0.0;
   END;
   GROW[0] := BROW[0];
   FOR I := 1 TO N DO
   BEGIN
      PP := 0.0;
      FOR J := 1 TO I DO
      BEGIN
         GGROW[J] := AROW[J]*GROW[I-J]+PP;
         PP := GGROW[J];
      END;
      GROW[I] := BROW[I]-PP ;
   END;
END;
{-------------------------------------------------------------------
--}
PROCEDURE  HANKELMATRIX(ORDER:INTEGER;GROW:INVECTOR;VAR HMAT:INMATRIX1;
                                VAR WMAT:INVECTOR2);

{-------------------------------------------------------------------}
{- CREATING HANKEL MATRIX , AND W MATRIX DERIVED AFTER DECONV -}
{-------------------------------------------------------------------}

VAR
  I,K,J    : INTEGER;

BEGIN
   FOR I := 1 TO ORDER DO
   BEGIN
      K := 0;
      FOR J := 1 TO ORDER DO
      BEGIN
         HMAT[I,J] := GROW[I+K];
         K := K+1 ;
      END;
   END;
   FOR I := 1 TO ORDER DO
   BEGIN
      WMAT[I] := -GROW[ORDER+I];
   END;
END;
```

```
{-------------------------------------------------------------------------}
PROCEDURE  NUMARRAYMAT(ORDER:INTEGER;BROWVECT:INVECTOR1;GROW:INVECTOR;
                       VAR  BMATRIX,WVECTORMATRIX:INMATRIX);

{-----------------------------------------------------}
{- THIS PROCEDURE FORMS BMATRIX AND W VECTOR        -}
{-----------------------------------------------------}

VAR
  I,J       : INTEGER;

  BEGIN
   FOR I := 0 TO ORDER DO
   BEGIN
      FOR J := 0 TO ORDER DO
      BMATRIX[I,J] :=0.0;
   END;
   FOR J := 0 TO ORDER DO
   BEGIN
      FOR I := 0 TO ORDER DO
      BEGIN
         IF I-J < 0 THEN
         BMATRIX[I,J] := 0.0
         ELSE
         BMATRIX[I,J] := BROWVECT[I-J];
      END;
  END;
  FOR I := 0 TO ORDER DO
  BEGIN
     WVECTORMATRIX[I,0] := GROW[I];
  END;
END;
{-------------------------------------------------------------------------}
procedure multi(dr1,dc1,dr2,dc2:integer;
                vij1:INmatrix;vij2:INmatrix;var vij3:INmatrix);
 var
   i,j,k : integer;

   {-----------------------------------------------------------------}
   {-Input :di=number of Row of vij1;dr=number of column of vij2      }
   {-Output:Matrix vij3 of di row and dr column                       }
   {                                                                  }
   { This procedure evalutes the multiplication of two matrix         }
   {                                                                  }
   { Be careful -  error dealing is not their                         }
   {-----------------------------------------------------------------}
   begin
     for i := 0 to dr1 do
      begin
        vij3[i,j] := 0.0;
        for j := 0 to dc2 do
         begin
          for k := 0 to dr2 do
           vij3[i,j] := vij3[i,j]+vij1[i,k]*vij2[k,j];
          end;
```

```
        end;
    end;
{-------------------------------------------------------------------------}
procedure Determinant(Dimen : integer;
                      Data  : TNmatrix1;
                  var Det   : Float;
                  var Error : byte);

procedure Initial(Dimen : integer;
              var Data  : TNmatrix1;
              var Det   : Float;
              var Error : byte);

{---------------------------------------------------------}
{- This procedure tests for errors in the value of Dimen -}
{---------------------------------------------------------}

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    if Dimen = 1 then
      Det := Data[1, 1];
end; { procedure Initial }

procedure EROswitch(var Row1 : TNvector2;
                    var Row2 : TNvector2);

{-------------------------------------------------}
{- Elementary row operation - switching two rows -}
{-------------------------------------------------}

var
  DummyRow : TNvector2;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : Float;
                     Dimen        : integer;
                 var ReferenceRow : TNvector2;
                 var ChangingRow  : TNvector2);

{-----------------------------------------------------------}
{- Row operation - adding a multiple of one row to another -}
{-----------------------------------------------------------}

var
  Term : integer;

begin
```

```
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier * ReferenceRow[Term
];
end; { procedure EROmultAdd }

Function Deter(Dimen : integer;
           var Data  : TNmatrix1) : Float;

{-------------------------------------------------------}
{- Input: Dimen, Data                                   -}
{- Output: Deter                                        -}
{-                                                      -}
{- Function returns the determinant of the Data matrix -}
{-------------------------------------------------------}

var
  PartialDeter, Multiplier : Float;
  Row, ReferenceRow : integer;
  DetEqualsZero : boolean;

procedure Pivot(Dimen          : integer;
                ReferenceRow   : integer;
            var Data           : TNmatrix1;
            var PartialDeter   : Float;
            var DetEqualsZero  : boolean);

{-------------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Data, PartialDeter           -}
{- Output: Data, PartialDeter, DetEqualsZero                -}
{-                                                          -}
{- This procedure searches the ReferenceRow column of the   -}
{- matrix Data for the first non-zero element below the     -}
{- diagonal. If it finds one, then the procedure switches   -}
{- rows so that the non-zero element is on the diagonal.    -}
{- Switching rows changes the determinant by a factor of    -}
{- -1; this change is returned in PartialDeter.             -}
{- If it doesn't find one, the matrix is singular and the   -}
{- Determinant is zero (DetEqualsZero = true is returned).  -}
{-------------------------------------------------------------}

var
  NewRow : integer;

begin
  DetEqualsZero := true;
  NewRow := ReferenceRow;
  while DetEqualsZero and (NewRow < Dimen) do  { Try to find a row  }
                                               { with a non-zero    }
                                               { element in this    }
                                               { column             }
  begin
    NewRow := Succ(NewRow);
    if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Data[NewRow], Data[ReferenceRow]);
```

```
      { Switch these two rows }
      DetEqualsZero := false;
      PartialDeter := -PartialDeter;  { Switching rows changes }
                                      { the determinant by a   }
                                      { factor of -1           }
    end;
  end;
end; { procedure Pivot }

begin  { function Deter }
  DetEqualsZero := false;
  PartialDeter := 1;
  ReferenceRow := 0;
  { Make the matrix upper triangular }
  while not(DetEqualsZero) and (ReferenceRow < Dimen - 1) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { If diagonal element is zero then switch rows }
    if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Data, PartialDeter, DetEqualsZero);
    if not(DetEqualsZero) then
      for Row := ReferenceRow + 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if ABS(Data[Row, ReferenceRow]) > TNNearlyZero then
        begin
          Multiplier := -Data[Row, ReferenceRow] /
                        Data[ReferenceRow, ReferenceRow];
          EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
        end;
    { Multiply the diagonal Term into PartialDeter }
    PartialDeter := PartialDeter * Data[ReferenceRow, ReferenceRow];
  end;
  if DetEqualsZero then
    Deter := 0
  else
    Deter := PartialDeter * Data[Dimen, Dimen];
end; { function Deter }

begin { procedure Determinant }
  Initial(Dimen, Data, Det, Error);
  if Dimen > 1 then
    Det := Deter(Dimen, Data);
end; { procedure Determinant }
{-------------------------------------------------------------------------}
procedure Gaussian_Elimination(Dimen        : integer;
                               Coefficients : TNmatrix1;
                               Constants    : TNvector2;
                           var Solution     : TNvector2;
                           var Error        : byte);

procedure Initial(Dimen        : integer;
              var Coefficients : TNmatrix1;
              var Constants    : TNvector2;
              var Solution     : TNvector2;
              var Error        : byte);
```

```
{-------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                     -}
{- Output: Solution, Error                                   -}
{-                                                           -}
{- This procedure test for errors in the value of Dimen.     -}
{- This procedure also finds the solution for the            -}
{- trivial case Dimen = 1.                                   -}
{-------------------------------------------------------------}

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    if Dimen = 1 then
      if ABS(Coefficients[1, 1]) < TNNearlyZero then
        Error := 2
      else
        Solution[1] := Constants[1] / Coefficients[1, 1];
end; { procedure Initial }

procedure EROswitch(var Row1 : TNvector2;
                    var Row2 : TNvector2);

{-----------------------------------------------}
{- Input: Row1, Row2                           -}
{- Output: Row1, Row2                          -}
{-                                             -}
{- elementary row operation - switching two rows -}
{-----------------------------------------------}

var
  DummyRow : TNvector2;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier    : Float;
                     Dimen         : integer;
                 var ReferenceRow  : TNvector2;
                 var ChangingRow   : TNvector2);

{-------------------------------------------------------------}
{- Input: Multiplier, Dimen, ReferenceRow, ChangingRow       -}
{- Output: ChangingRow                                       -}
{-                                                           -}
{- row operation - adding a multiple of one row to another   -}
{-------------------------------------------------------------}

var
  Term : integer;
```

```
begin
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
end; { procedure EROmultAdd }

procedure UpperTriangular(Dimen        : integer;
                      var Coefficients : TNmatrix1;
                      var Constants    : TNvector2;
                      var Error        : byte);

{-------------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                           -}
{- Output: Coefficients, Constants, Error                          -}
{-                                                                 -}
{- This procedure makes the coefficient matrix upper triangular.   -}
{- The operations which perform this are also performed on the     -}
{- Constants vector.                                               -}
{- If one of the main diagonal elements of the upper triangular    -}
{- matrix is zero, then the Coefficients matrix is singular and    -}
{- no solution exists (Error = 2 is returned).                     -}
{-------------------------------------------------------------------}

var
  Multiplier        : Float;
  Row, ReferenceRow : integer;

procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Coefficients : TNmatrix1;
            var Constants    : TNvector2;
            var Error        : byte);

{-------------------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Coefficients                        -}
{- Output: Coefficients, Constants, Error                          -}
{-                                                                 -}
{- This procedure searches the ReferenceRow column of the          -}
{- Coefficients matrix for the first non-zero element below        -}
{- the diagonal. If it finds one, then the procedure switches      -}
{- rows so that the non-zero element is on the diagonal.           -}
{- It also switches the corresponding elements in the              -}
{- Constants vector. If it doesn't find one, the matrix is         -}
{- singular and no solution exists (Error = 2 is returned).        -}
{-------------------------------------------------------------------}

var
  NewRow : integer;
  Dummy : Float;

begin
  Error := 2;           { No solution exists }
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do    { Try to find a      }
                                               { row with a non-zero }
```

```
                                                    { diagonal element    }
  begin
    NewRow := Succ(NewRow);
    if ABS(Coefficients[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Coefficients[NewRow], Coefficients[ReferenceRow]);
      { Switch these two rows }
      Dummy := Constants[NewRow];
      Constants[NewRow] := Constants[ReferenceRow];
      Constants[ReferenceRow] := Dummy;
      Error := 0;     { Solution may exist }
    end;
  end;
end; { procedure Pivot }

begin { procedure UpperTriangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen - 1) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Check to see if the main diagonal element is zero }
    if ABS(Coefficients[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Coefficients, Constants, Error);
    if Error = 0 then
      for Row := ReferenceRow + 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if ABS(Coefficients[Row, ReferenceRow]) > TNNearlyZero then
        begin
          Multiplier := -Coefficients[Row, ReferenceRow] /
                         Coefficients[ReferenceRow,ReferenceRow];
          EROmultAdd(Multiplier, Dimen,
                     Coefficients[ReferenceRow], Coefficients[Row]);
          Constants[Row] := Constants[Row] +
                            Multiplier * Constants[ReferenceRow];
        end;
  end; { while }
  if ABS(Coefficients[Dimen, Dimen]) < TNNearlyZero then
    Error := 2;     { No solution }
end; { procedure UpperTriangular }

procedure BackwardsSub(Dimen        : integer;
                   var Coefficients : TNmatrix1;
                   var Constants    : TNvector2;
                   var Solution     : TNvector2);

{-------------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                           -}
{- Output: Solution                                                -}
{-                                                                 -}
{- This procedure applies backwards substitution to the upper      -}
{- triangular Coefficients matrix and Constants vector. The        -}
{- resulting vector is the solution to the set of equations and    -}
{- is returned in the vector Solution.                             -}
{-------------------------------------------------------------------}
```

```
var
  Term, Row : integer;
  Sum : Float;

begin
  Term := Dimen;
  while Term >= 1 do
  begin
    Sum := 0;
    for Row := Term + 1 to Dimen do
      Sum := Sum + Coefficients[Term, Row] * Solution[Row];
    Solution[Term] := (Constants[Term] - Sum) / Coefficients[Term, Term];
    Term := Pred(Term);
  end;
end; { procedure BackwardsSub }

begin { procedure Gaussian_Elimination }
  Initial(Dimen, Coefficients, Constants, Solution, Error);
  if Dimen > 1 then
  begin
    UpperTriangular(Dimen, Coefficients, Constants, Error);
    if Error = 0 then
      BackwardsSub(Dimen, Coefficients, Constants, Solution);
  end;
end; { procedure Gaussian_Elimination }
{-------------------------------------------------------------------------
}
procedure GetData(var p : integer;VAR T : REAL;
                  var Y  : TNvector);

{-------------------------------------------------}
{- Output: Dimen, Data                           -}
{-                                               -}
{- This procedure sets the value of Dimen and Data -}
{- from either keyboard input or file input      -}
{-------------------------------------------------}

var
  Ch : char;

procedure GetDataFromKeyboard(var p : integer;VAR T : REAL;
                              var Y  : TNvector);

{---------------------------------------}
{- Output: p , u                       -}
{-                                     -}
{- This procedure sets the value of    -}
{- p and u from keyboard input -       }
{---------------------------------------}

var
  i : integer;

begin
  Writeln;
```

```
  repeat
    Write('Input number of samples p : ');
    Readln(p);
    write('RANGE OF TIME FROM t = 0 secs TO : ');
    readln(T);
    IOCheck;
  until (not IOerr);
  Writeln;
  for i := 0 to p do
      repeat
        Write('AT TIME = ',T*I/P, '  Y[',i,'] : ');
        Readln(Y[i]);
        IOCheck;
      until not IOerr;
end; { procedure GetDataFromKeyboard }

procedure GetDataFromFile(var p : integer;VAR T : REAL;
                          var Y  : TNvector);

{---------------------------------------}
{- Output : p , u                      -}
{-                                     -}
{- This procedure sets the value of    -}
{- Dimen and Data from file input      -}
{---------------------------------------}

var
  FileName : string[255];
  InFile : text;
   i : integer;

begin
  Writeln;
  repeat
    Writeln;
    repeat
      Write('File name? ');
      Readln(FileName);
      Assign(InFile, FileName);
      Reset(InFile);
      IOCheck;
    until not IOerr;
    Read(InFile,p);
    read(inFile,T);
    IOCheck;
    while (not IOerr)  do
    begin
      for i := 0 to p do
      begin
        Read(InFile,Y[i]);
        IOCheck;
      end;
    end;
  until not IOerr;
end; { procedure GetDataFromFile }
```

```
)egin { procedure GetData }
  case InputChannel('Input Data From') of
    'K' : GetDataFromKeyboard(p,T,Y);
    'F' : GetDataFromFile(p,T,Y);
  end;
)nd; { procedure GetData }
{-------------------------------------------------------------------------}
)rocedure Results(  q    : integer;
                var A    : INMATRIX;VAR B : INVECTOR1);



{-------------------------------------------------------------------}
{- This procedure outputs the results to the device OutFile -}
{-------------------------------------------------------------------}

)ar
   Row : integer;

)egin
  Writeln(OutFile);
  Writeln(OutFile);
  Writeln(outfile,'THE ORDER OF SYSTEM IS',Q);
  Writeln(OutFile,'  THE NUMERATOR COEFF.   AND  THE DENOMINATOR COEFF.');
  for Row := 0 to q DO
  begin
        WriteLN(OutFile,'Z^',ROW,' ',A[Row,0],'   ',B[ROW] );
    Writeln(OutFile);
  end;
)nd;
{-------------------------------------------------------------------------}
BEGIN {MAIN}
   CLRSCR;
   GOTOXY(5,5);
   WRITELN( 'THIS METHOD IS FOR THE SYSTEMS EXCITED BY UNIT STEP INPUT ');
   GETDATA(P,RANGEOFTIME,WELEMENT);
   CLRSCR;
   ORDER := 1;
   K := 1;
   DVALUE[1] := WELEMENT[1];
   REPEAT
      ORDER := ORDER+1 ;
      HANKELMATRIX(ORDER,WELEMENT,HANKELMAT,WVECTOR);
      DETERMINANT(ORDER,HANKELMAT,VALUE,ERROR1);
      DVALUE[ORDER] := VALUE;
      IF (ORDER > 2) AND (VALUE > 0.0) THEN
      BEGIN
        HVALUE[K] := DVALUE[ORDER-1]/DVALUE[ORDER];
      END;
      K := K+1;
   UNTIL (VALUE <= 0) AND (ORDER <= INT((P+1)/2)) AND (ORDER <= 20);
   XVALUE := HVALUE[1];
   FOR I := 2 TO K-1 DO
   BEGIN
```

```
        IF HVALUE[I] > XVALUE THEN
        BEGIN
          XVALUE := HVALUE[I];
          MN     := I;
        END;
    END;
    IF ORDER = 20 THEN
      WRITELN(' NO CONVERGENCE IS FOUND,SO CHECK THE INPUTS MADE')
    ELSE
    BEGIN
      ORDER := MN ;
      HANKELMATRIX(ORDER,WELEMENT,HANKELMAT,WVECTOR);
      GAUSSIAN_ELIMINATION(ORDER,HANKELMAT,WVECTOR,BARRAY,ERROR2);
      BROWVECT[0] := 1.0;
      FOR I := 1 TO ORDER DO
      BEGIN
         BROWVECT[I] := BARRAY[ORDER+1-I];
      END;
      NUMARRAYMAT(ORDER,BROWVECT,WELEMENT,BMATRIX,WARRAY);
      MULTI(ORDER,ORDER,ORDER,0,BMATRIX,WARRAY,AVECTOR);
      GETOUTPUTFILE(OUTFILE);
      RESULTS(ORDER,AVECTOR,BROWVECT);
      CLOSE(OUTFILE);
    END;
END.
```

APPENDIX - IV

PROGRAM FOR BLOCKDIGRAM REDUCTION

```
PROGRAM BLOCK_DIAGRAM_SIMULATION;
USES
  DOS,CRT,GRAPH,COMMON,GKERNEL,GWINDOW,GDRIVER,GSHELL;
const
   INNearlyzero = 1E-07;
type
   INvector = array[1..10] of real;
   INmatrix = array[1..10] of INvector;
   Float    = real;
var
PP,P,SS,VV,I,J,AXY,NBLOCKS,MORD : INTEGER;
Q,R,S,T,U,V                     : INTEGER;
INPUTNUMBER,OUTPUTNUMBER        : INTEGER;
NUM,DEN                         : INVECTOR;
KMAT,AMAT,BMAT,CMAT,DMAT,EMAT   : INMATRIX;
FMAT,GMAT,HMAT,TMAT,TTMAT       : INMATRIX;
AAMAT,BBMAT,CCMAT,DDMAT         : INMATRIX;
BTMAT,BITMAT,BIMAT              : INMATRIX;
ERROR                           : BYTE;
{-------------------------------------------------------------------------}
PROCEDURE tftoss(num,den : tnvector;mord,pp,ss,vv : integer;
                 var Amat,Bmat,Cmat,Dmat:Inmatrix;
                 var p,q,r,s,t,u,v :integer);
 {-------------------------------------------------------}
 {PROCEDURE to change TF to State space transformation }
 {Method is based on the on algorithim of J.N.Little   }
 {num = array of numerator;den = array of denominator  }
 {nord,mord = order of numerator and denominator resp. }
 {Amat,Bmat,Cmat,Dmat :  state space matrix             }
 {p,q = matrix dimension of Amat                       }
 {r,s = matrix dimension of Bmat                       }
 {t,u = matrix dimension of Cmat                       }
 {v   = dimension of Dmat                              }
 {LIMITATION  Applicable only for SISO systems         }
 {-------------------------------------------------------}
VAR
  i,j : integer;
  dden : INvector;
 BEGIN
   if mord = 0 then
    begin
       Amat[1+pp,1+pp] := 0;
       Bmat[1+pp,1+ss] := 0;
       Cmat[1+ss,1+pp] := 0;
       Dmat[1+vv,1+vv] := num[1]/den[1];
       p := 1+pp;
       q := 1+pp;
       r := 1+pp;
       s := 1+ss;
       t := 1+ss;
       u := 1+pp;
       v := 1+vv;
   end
   else
```

```
   begin
    for i := 1 to mord do
     begin
        dden[i] := den[i]/den[mord+1];
     end;
    for i := 1 to mord do
     begin
        for j := 1 to mord do
          if i = mord
          then Amat[i+pp,j+pp] := -dden[j] else
          if j = i+1
          then Amat[i+pp,j+pp] := 1
          else
          Amat[i+pp,j+pp] := 0;
     end;
    for i := 1 to mord do
     begin
        if i = mord then
         Bmat[i+pp,1+ss] := 1
        else
         Bmat[i+pp,1+ss] := 0;
     end;
    for i := 2 to mord+1 do
     begin
        Cmat[1+ss,i-1+pp] := num[i-1]-num[mord+1]*dden[i-1];
     end;
    Dmat[1+vv,1+vv] := num[mord+1];
    p := mord+pp;
    q := p;
    r := q;
    s := 1+ss;
    t := 1+ss;
    u := p;
    v := 1+vv;
   end;
end; {end of procedure}

{-------------------------------------------------------------------------
---}

PROCEDURE CONNECTIONMAT(NBLOCKS:INTEGER;VAR KMAT:INMATRIX;
                        VAR INPUTNUMBER,OUTPUTNUMBER:INTEGER);
 {-----------------------------------------------------}
 {procedure for formation of connection matrix         }
 {K is connection matrix                               }
 {This matrix consist of 1,-1 or 0                     }
 {If their are 10 blocks then it is a 10X10 matrix     }
 { OF 1,-1,0                                           }
 { APPLICABLE ONLY FOR SISO SYSTEMS                    }
 {-----------------------------------------------------}
VAR
 I,J : INTEGER;

BEGIN
  CLRSCR;
```

```
  GOTOXY(4,3);
  WRITE('THE ENTRY SHOULD BE 1,-1 OR 0');
  GOTOXY(15,13);
  WRITE('IF PATH IS FORWARD THEN ENTRY SHOULD BE : 1');
  GOTOXY(16,13);
  WRITE('IF PATH IS NEGATIVE THEN ENTRY IS : -1');
  GOTOXY(17,13);
  WRITE('IF NO DIRECT CONNECTION EXIST THEN IT IS : 0');
  CLRSCR;
  FOR I := 1 TO NBLOCKS DO
   BEGIN
     FOR J := 1 TO NBLOCKS DO
     BEGIN
       GOTOXY(3,3);
       WRITE('BLOCK NO: ',I,' IS GETTING INPUT FROM BLOCK NO: ',J,'IS(1/-1/
0) ');
       READLN(KMAT[I,J]);
       CLRSCR;
     END;
   END;
  CLRSCR;
  GOTOXY(8,8);
  WRITE('GIVE THE INPUTNUMBER ');
  READLN(INPUTNUMBER);
  WRITELN;
  WRITE('GIVE THE OUTPUTNUMBER ');
  READLN(OUTPUTNUMBER);
END;
{-------------------------------------------------------------------------
-}
procedure multi(dr1,dc1,dr2,dc2:integer;
                vij1:TNmatrix;vij2:TNmatrix;var vij3:TNmatrix);
var
  i,j,k : integer;

  {-------------------------------------------------------------------}
  {-Input :di=number of Row of vij1;dr=number of column of vij2       }
  {-Output:Matrix vij3 of di row and dr column                        }
  {                                                                   }
  { This procedure evalutes the multiplication of two matrix          }
  {                                                                   }
  { Be careful -  error dealing is not their                          }
  {-------------------------------------------------------------------}
  begin
    for i := 1 to dr1 do
      begin
        vij3[i,j] := 0.0;
        for j := 1 to dc2 do
         begin
          for k := 1 to dr2 do
           vij3[i,j] := vij3[i,j]+vij1[i,k]*vij2[k,j];
         end;
      end;
  end;
-------------------------------------------------------------------}
```

```
PROCEDURE SUBTRACT(ABC:INTEGER;MAT1,MAT2:INMATRIX;VAR MAT3:INMATRIX);
 {-------------------------------------------------}
 { PROCEDURE FOR SUBSTRACTION OF TO MATRIX          }
 {-------------------------------------------------}
VAR
  I,J : INTEGER:
BEGIN
   FOR I := 1 TO ABC DO
   BEGIN
      FOR J := 1 TO ABC DO
      MAT3[I,J] := MAT1[I,J]-MAT2[I,J];
   END;
END;
{-------------------------------------------------------------------------------}
PROCEDURE ADDMAT(ABC:INTEGER;MAT1,MAT2:INMATRIX;VAR MAT3:INMATRIX);
 {-------------------------------------------------}
 { PROCEDURE FOR ADDITION OF MATRIX                 }
 {-------------------------------------------------}
VAR
 I,J : INTEGER:
BEGIN
   FOR I := 1 TO ABC DO
   BEGIN
      FOR J := 1 TO ABC DO
      MAT3[I,J] := MAT1[I,J]+MAT2[I,J];
   END;
END;
{-------------------------------------------------------------------------------}
PROCEDURE EYE(ND : INTEGER;VAR IDMAT : INMATRIX);

{-------------------------------------------------}
{ PROCEDURE TO FORM IDENTITY MATRIX OF SIZE ND     }
{-------------------------------------------------}
VAR
 I,J : INTEGER;
BEGIN
  FOR I := 1 TO ND DO
  BEGIN
    FOR J := 1 TO ND DO
    BEGIN
      IF I = J THEN
      IDMAT[I.J] := 1
      ELSE
      IDMAT[I,J] := 0;
    END;
  END;
END;
{-------------------------------------------------------------------------------}

procedure Results(var pp,vv,INPUTNUMBER,OUTPUTNUMBER : integer;
              var AAMAT,BBMAT,CCMAT,DDMAT :tnmatrix);



{-----------------------------------------------------------}
```

```
{- This procedure outputs the results to the device OutFile -}
{-------------------------------------------------------------}

var
   i,j : integer;

begin
  Writeln(OutFile);
  Writeln(OutFile);
  Writeln(OutFile, 'ORDER OF A - MATRIX : ',PP,'X',PP);
  writeln;
  writeln(outfile, 'A - MATRIX :');
  WRITELN;
  for i := 1 to PP do
  begin
     for j := 1 to PP do
         write(outfile,AAMAT[i,j] : 18:8);
    Writeln(OutFile);
  end;
  writeln(outfile, 'ORDER OF B - MATRIX : ',PP,' X 1' );
  writeln(outfile);
  writeln(outfile,'B - MATRIX;');
  writeln(outfile);
  for i := 1 to PP do
    begin
      write(outfile,BBMAT[i,INPUTNUMBER]);
      writeln(outfile);
    end;
  writeln(outfile,'C - MATRIX :');
  writeln(outfile);
  writeln(outfile,'ORDER OF C - MATRIX : 1 X ',PP);
  writeln(outfile);
  for i := 1 to PP do
   begin
     write(outfile,CCMAT[OUTPUTNUMBER,i]);
   end;
   writeln(outfile);
   writeln(outfile,'D - MATRIX');
   writeln(outfile,'ORDER OF D - MATRIX : ',V);
   writeln(outfile,DDMAT[OUTPUTNUMBER,INPUTNUMBER]);
   writeln(outfile);
end;
{-------------------------------------------------------------------------
-}
procedure Inverse(Dimen : integer;
                  Data  : INmatrix;
              var Inv   : INmatrix;
              var Error : byte);


procedure Initial(Dimen : integer;
              var Data  : INmatrix;
              var Inv   : INmatrix;
              var Error : byte);
```

```
{-------------------------------------------------------}
{ Input: Dimen, Data                                    }
{ Output: Inv, Error                                    }
{                                                       }
{ This procedure test for errors in the value of Dimen  }
{-------------------------------------------------------}

ar
 Row : integer;

egin
 Error := 0;
 if Dimen < 1 then
   Error := 1
 else
   begin
     { First make the inverse-to-be the identity matrix }
     FillChar(Inv, SizeOf(Inv), 0);
     for Row := 1 to Dimen do
       Inv[Row, Row] := 1;
     if Dimen = 1 then
       if ABS(Data[1, 1]) < TNNearlyZero then
         Error := 2   { Singular matrix }
       else
         Inv[1, 1] := 1 / Data[1, 1];
   end;
nd; { procedure Initial }

rocedure EROdiv(Divisor : Float;
                Dimen   : integer;
            var Row     : TNvector);

{-------------------------------------------------------}
{ Input: Divisor, Dimen, Row                            }
{                                                       }
{ elementary row operation - dividing by a constant    }
{-------------------------------------------------------}

ar
 Term : integer;

egin
 for Term := 1 to Dimen do
   Row[Term] := Row[Term] / Divisor;
nd; { procedure EROdiv }

rocedure EROswitch(var Row1 : TNvector;
                   var Row2 : TNvector);

{-------------------------------------------------------}
{ Input: Row1, Row2                                     }
{ Output: Row1, Row2                                    }
{                                                       }
{ Elementary row operation - switching two rows         }
{-------------------------------------------------------}
```

```
ar
 DummyRow : INvector;

egin
 DummyRow := Row1;
 Row1 := Row2;
 Row2 := DummyRow;
nd; { procedure EROswitch }

rocedure EROmultAdd(Multiplier    : Float;
                    Dimen         : integer;
                var ReferenceRow  : INvector;
                var ChangingRow   : INvector);

{-------------------------------------------------------------}
{ Input: Multiplier, Dimen, ReferenceRow, ChangingRow         }
{ Output: ChangingRow                                         }
{                                                             }
{ Row operation - adding a multiple of one row to another     }
{-------------------------------------------------------------}

ar
 Term : integer;

egin
 for Term := 1 to Dimen do
   ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
nd; { procedure EROmultAdd }


rocedure Inver(Dimen : integer;
           var Data  : INmatrix;
           var Inv   : INmatrix;
           var Error : byte);

{-------------------------------------------------------------}
{ Input: Dimen, Data                                          }
{ Output: Inv, Error                                          }
{                                                             }
{ This procedure computes the inverse of the matrix Data      }
{ and stores it in the matrix Inv.  If the matrix Data        }
{ is singular, then Error = 2 is returned.                    }
{-------------------------------------------------------------}

ar
 Divisor, Multiplier : Float;
 Row, ReferenceRow : integer;

rocedure Pivot(Dimen  - -    : integer;
               ReferenceRow  : integer;
           var Data          : INmatrix;
           var Inv           : INmatrix;
           var Error         : byte);
```

```
{-------------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Data, Inv                     -}
{- Output: Data, Inv, Error                                  -}
{-                                                           -}
{- This procedure searches the ReferenceRow column of        -}
{- the Data matrix for the first non-zero element below      -}
{- the diagonal. If it finds one, then the procedure         -}
{- switches rows so that the non-zero element is on the      -}
{- diagonal. This same operation is applied to the Inv       -}
{- matrix. If no non-zero element exists in a column, the    -}
{- matrix is singular and no inverse exists.                 -}
{-------------------------------------------------------------}

var
  NewRow : integer;

begin
  Error := 2;  { No inverse exists }
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do
  { Try to find a       }
  { row with a non-zero }
  { diagonal element    }
  begin
    NewRow := Succ(NewRow);
    if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Data[NewRow], Data[ReferenceRow]);
      { Switch these two rows }
      EROswitch(Inv[NewRow], Inv[ReferenceRow]);
      Error := 0;
    end;
  end; { while }
end; { procedure Pivot }

begin { procedure Inver }
  { Make Data matrix upper triangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Check to see if the diagonal element is zero }
    if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Data, Inv, Error);
    if Error = 0 then
    begin
      Divisor := Data[ReferenceRow, ReferenceRow];
      EROdiv(Divisor, Dimen, Data[ReferenceRow]);
      EROdiv(Divisor, Dimen, Inv[ReferenceRow]);
      for Row := 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if (Row <> ReferenceRow) and
           (ABS(Data[Row, ReferenceRow]) > TNNearlyZero) then
        begin
          Multiplier := -Data[Row, ReferenceRow] /
```

```
                        Data[ReferenceRow, ReferenceRow];
          ERUmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
          ERUmultAdd(Multiplier, Dimen, Inv[ReferenceRow], Inv[Row]);
        end;
    end;
  end;
end; { procedure Inver }

begin { procedure Inverse }
  Initial(Dimen, Data, Inv, Error);
  if Dimen > 1 then
    Inver(Dimen, Data, Inv, Error);
end; { procedure Inverse }
{-------------------------------------------------------------------------}

BEGIN {main}
  INITGRAPHIC;
  DEFINEWINDOW(1,0,0,XMAXGLB,YMAXGLB);
  DEFINEWORLD(1,0,0,1000,1000);
  SELECTWORLD(1);
  SELECTWINDOW(1);
  DRAWBORDER;
  DRAWTEXTW(100,100,4,'BLOCKDIAGRAM SIMULATION');
  DRAWTEXTW(200,400,2,'BY');
  DRAWTEXTW(500,500,2,'VIJAY PANDE');
  DRAWTEXTW(200,600,2,'GUIDE');
  DRAWTEXTW(500,700,2,'PROF M.K.VASANTHA');
  DRAWTEXTW(200,800,2,'E.E.DEPT, UOR,  dt:10/1/1990');
  DRAWTEXTW(400,900,1,'PRESS ANY KEY TO CONTINUE');
  REPEAT UNTIL KEYPRESSED;
  LEAVEGRAPHIC;
  CLRSCR;
  FOR I := 1 TO 10 DO
  BEGIN
     FOR J := 1 TO 10 DO
     BEGIN
        AMAT[I,J] := 0;
        BMAT[I,J] := 0;
        CMAT[I,J] := 0;
        DMAT[I,J] := 0;
     END;
  END;
  PP := 0;
  SS := 0;
  VV := 0;
  CLRSCR;
  GOTOXY(2,2);
  WRITE('NUMBER OF BLOCKS: ');
  READLN(NBLOCKS);
  FOR AXY := 1 TO NBLOCKS DO
  BEGIN
     WRITE('ORDER OF TRANSFER FUNCTION OF BLOCKNO: ',AXY,' IS : ');
     READLN(MORD);
     GOTOXY(3,5);
     WRITE('THE BLOCK NUMBER IS ',AXY);
```

```
      WRITELN;
      CLRSCR;
      GOTOXY(3,5);
      WRITELN('ENTER THE VALUES OF COEFF. OF NUM,DEN OF BLOCK NO: ',AXY);
      FOR I := 1 TO MORD+1 DO
      BEGIN
         CLRSCR;
         GOTOXY(5,2);
         WRITE('NUMERATOR [',I-1,'] X S^',I-1,' : ');
         READLN(NUM[I]);
         WRITELN;
         CLRSCR;
         GOTOXY(6,2);
         WRITE('DENOMINATOR [',I-1,'] X S^',I-1,' : ');
         READLN(DEN[I]);
         WRITELN;
         CLRSCR;
      END;
      TFTOSS(NUM,DEN,MORD,PP,SS,VV,AMAT,BMAT,CMAT,DMAT,P,Q,R,S,T,U,V);
      PP := P;
      SS := S;
      VV := V;
   END;
CLRSCR;
CONNECTIONMAT(NBLOCKS,KMAT,INPUTNUMBER,OUTPUTNUMBER);
CLRSCR;
GETOUTPUTFILE(OUTFILE);
EYE(VV,GMAT);
MULTI(NBLOCKS,NBLOCKS,VV,VV,KMAT,DMAT,BIMAT);
SUBTRACT(VV,GMAT,BIMAT,BIMAT);
INVERSE(VV,BIMAT,BITMAT,ERROR);
MULTI(PP,VV,VV,VV,BMAT,BITMAT,BBMAT);
MULTI(NBLOCKS,NBLOCKS,VV,PP,KMAT,CMAT,EMAT);
MULTI(PP,VV,VV,PP,BBMAT,EMAT,FMAT);
ADDMAT(PP,AMAT,FMAT,AAMAT);
MULTI(VV,VV,NBLOCKS,NBLOCKS,DMAT,KMAT,HMAT);
SUBTRACT(VV,GMAT,HMAT,TMAT);
INVERSE(VV,TMAT,TTMAT,ERROR);
MULTI(VV,VV,VV,PP,TTMAT,CMAT,CCMAT);
MULTI(VV,VV,VV,VV,TTMAT,DMAT,DDMAT);
RESULTS(PP,VV,INPUTNUMBER,OUTPUTNUMBER,AAMAT,BBMAT,CCMAT,DDMAT);
CLOSE(OUTFILE);
END.
```

APPENDIX - V

PROGRAM FOR FFT APPLICATIONS

```
PROGRAM FFTPROGRAMS;

{------------------------------------------------------------------------
--}
{-
 -}
{-      Purpose:  This procedure provides I/O routines for using the
 -}
{-                fast Fourier transform, convolution and correlation
 -}
{-                routines.
 -}
{-
 -}
{-      Unit   :  vij.... TPU       procedure MakeSinCosTable
 -}
{-                                            BitInvert
 -}
{-                                            FFT
 -}
{-                                  procedure RealFFT
 -}
{-                                  procedure RealConvolution
 -}
{-                                  procedure RealCorrelation
 -}
{-                                  procedure ComplexFFT
 -}
{-                                  procedure ComplexConvolution
 -}
{-                                  procedure ComplexCorrelation
 -}
{-
 -}
{------------------------------------------------------------------------
--}

{$I-}      { Disable I/O error trapping  }

uses
  FFTB2, Dos, Crt, Common;

type
  Analyses = (RF, RN, RC, RA, CF, CN, CC, CA);

var
  WhichAnalysis : Analyses;          { Indicates which application  }
                                     { will be run                  }
  NumPoints : integer;               { Number of points  }
  XReal, XImag : INvectorPtr;        { One set of complex data points  }
  HReal, HImag : INvectorPtr;        { Another set of complex data points  }
  Inverse : boolean;                 { False ==> forward Fourier transform
}
                                     { True ===> inverse Fourier transform
}
```

```
  Auto : boolean;                    { False ==> crosscorrelation  }
                                     { True ===> autocorrelation   }
  Error : byte;                      { Flags if something went wrong  }

procedure Initialize(var NumPoints : integer;
                     var XReal     : TNvectorPtr;
                     var XImag     : TNvectorPtr;
                     var HReal     : TNvectorPtr;
                     var HImag     : TNvectorPtr;
                     var Error     : byte);

{-------------------------------------------------------------}
{- Output: NumPoints, XReal, XImag, HReal, HImag, Error      -}
{-                                                           -}
{- This procedure initializes the above variables to zero    -}
{-------------------------------------------------------------}

begin
  NumPoints := 0;
  New(XReal);
  New(XImag);
  New(HReal);
  New(HImag);
  FillChar(XReal^, SizeOf(XReal^), 0);
  FillChar(XImag^, SizeOf(XImag^), 0);
  FillChar(HReal^, SizeOf(HReal^), 0);
  FillChar(HImag^, SizeOf(HImag^), 0);
  Error := 0;
end; { procedure Initialize }

procedure GetData(var NumPoints     : integer;
                  var WhichAnalysis : Analyses;
                  var Auto          : boolean;
                  var XReal         : TNvectorPtr;
                  var XImag         : TNvectorPtr;
                  var HReal         : TNvectorPtr;
                  var HImag         : TNvectorPtr);

{-------------------------------------------------------------}
{- Output: NumPoints, WhichAnalysis, Auto, XReal, XImag,     -}
{-         HReal, HImag                                      -}
{-                                                           -}
{- This procedure reads in data from either the keyboard     -}
{- or a data file.                                           -}
{-------------------------------------------------------------}

var
  Ch : char;
  NumPoints1 : integer;
  NumPoints2 : integer;

function TestForPowersOfTwo(NumPoints : integer) : boolean;

{-------------------------------------------------------------}
{- Input: NumPoints                                          -}
```

```
{- Output: TestForPowersOfTwo                                    -}
{-                                                               -}
{- This procedure checks if NumPoints is a power of two.  -}
{- It returns True if it is. False if it isn't.                  -}
{-----------------------------------------------------------------}

type
  ShortArray = array[1..13] of integer;

const
  PowersOfTwo : ShortArray = (2, 4, 8, 16, 32, 64, 128, 256,
                              512, 1024, 2048, 4096, 8192);

var
  Term : integer;
  Test : boolean;

begin
  Test := false; { Assume NumPoints not a power of two  }
  Term := 1;
  while (Term <= 13) and (not Test) do
  begin
    if NumPoints = PowersOfTwo[Term] then
      Test := true; { NumPoints is a power of two  }
    Term := Succ(Term);
  end;
  TestForPowersOfTwo := Test;
end; { function TestForPowersOfTwo }

procedure GetRealVectorFromFile(var NumPoints : integer;
                                var XReal     : TNvectorPtr);

{-------------------------------------------------}
{- Output: NumPoints, X                          -}
{-                                               -}
{- This procedure reads in a real vector of      -}
{- data points from a data file.                 -}
{-------------------------------------------------}

var
  FileName : string[255];
  InFile : text;

begin
  Writeln;
  repeat
    Write('File name? ');
    Readln(FileName);
    Assign(InFile, FileName);
    Reset(InFile);
    IOCheck;
  until not IOerr ;
  NumPoints := 0;
  while not EOF(InFile)  do
  begin
```

```
    Readln(InFile, XReal^[NumPoints]);
    NumPoints := Succ(NumPoints);
    IOCheck;
  end;
  Close(InFile);
end; { procedure GetRealVectorFromFile }

procedure GetComplexVectorFromFile(var NumPoints : integer;
                                   var XReal     : TNvectorPtr;
                                   var XImag     : TNvectorPtr);

{-----------------------------------------------}
{- Output: NumPoints, XReal, XImag             -}
{-                                             -}
{- This procedure reads in a complex vector    -}
{- of data points from a data file.            -}
{-----------------------------------------------}

var
  FileName : string[255];
  InFile : text;

begin
  Writeln;
  repeat
    Write('File name? ');
    Readln(FileName);
    Assign(InFile, FileName);
    Reset(InFile);
    IOCheck;
  until not IOerr;
  NumPoints := 0;
  while not EOF(InFile) do
  begin
    Readln(InFile, XReal^[NumPoints], XImag^[NumPoints]);
    NumPoints := Succ(NumPoints);
    IOCheck;
  end;
  Close(InFile);
end; { procedure GetComplexVectorFromFile }

procedure GetRealVectorFromKeyboard(var NumPoints : integer;
                                    var XReal     : TNvectorPtr);

{-----------------------------------------------}
{- Output: NumPoints, X                        -}
{-                                             -}
{- This procedure reads in a real vector of    -}
{- data points from the keyboard.              -}
{-----------------------------------------------}

var
  Term : integer;

begin
```

```
  NumPoints := 0;
  Writeln;
  repeat
    Write('Number of points (a power of two between 2^', INArraySize + 1, ')
? ');
    Readln(NumPoints);
    IOCheck;
    IOerr := not TestForPowersOfTwo(NumPoints);
  until (NumPoints >= 2) and (NumPoints <= INArraySize + 1) and not IOerr;
  Writeln;
  Writeln('Type in the X values:');
  for Term := 0 to NumPoints - 1  do
  repeat
    Write('Real(X[', Term, ']): ');
    Readln(XReal^[Term]);
    IOCheck;
  until not IOerr;
end; { procedure GetRealVectorFromKeyboard }

procedure GetComplexVectorFromKeyboard(var NumPoints : integer;
                                       var XReal     : INvectorPtr;
                                       var XImag     : INvectorPtr);

{-------------------------------------------------}
{- Output: NumPoints, XReal, XImag               -}
{-                                               -}
{- This procedure reads in a complex vector of   -}
{- data points from the keyboard.                -}
{-------------------------------------------------}

var
  Term : integer;

begin
  NumPoints := 0;
  repeat
    Write('Number of points (a power of two between 2^', INArraySize + 1, ')
? ');
    Readln(NumPoints);
    IOCheck;
    IOerr := not TestForPowersOfTwo(NumPoints);
  until (NumPoints >= 2) and (NumPoints <= INArraySize + 1) and not IOerr;
  Writeln;
  Writeln('Type in the X values:');
  for Term := 0 to NumPoints - 1  do
  begin
    repeat
      Write('Real(X[', Term, ']): ');
      Readln(XReal^[Term]);
      IOCheck;
    until not IOerr;
    repeat
      Write('Imaginary(X[', Term, ']): ');
      Readln(XImag^[Term]);
      IOCheck;
```

```
    until not IOerr;
  end;
end; { procedure GetComplexVectorFromKeyboard }

procedure GetInverse(var Inverse : boolean);

var
  Ch : char;

begin
  Writeln;
  Write('(F)orward or (I)nverse transform? ');
  repeat
    Ch := UpCase(ReadKey);
  until Ch in ['F', 'I'];
  Writeln(Ch);
  if Ch = 'F' then
    Inverse := false  { Forward transform  }
  else
    Inverse := true;  { Inverse transform  }
end; { procedure GetInverse }

begin { procedure GetData }
  Writeln('   1) Real Fast Fourier Transform');
  Writeln('   2) Real Convolution');
  Writeln('   3) Real Autocorrelation');
  Writeln('   4) Real Crosscorrelation');
  Writeln('   5) Complex Fast Fourier Transform');
  Writeln('   6) Complex Convolution');
  Writeln('   7) Complex Autocorrelation');
  Writeln('   8) Complex Crosscorrelation');
  Writeln;
  Write('   Select a number (1-8): ');
  repeat
    Ch := ReadKey;
  until Ch in ['1'..'8'];

  Writeln;
  case Ch of
    '1' : begin
            Writeln('********* Real Fast Fourier transform *********');
            GetInverse(Inverse);
            WhichAnalysis := RF;
          end;

    '2' : begin
            Writeln('********* Real Convolution *********');
            WhichAnalysis := RN;
          end;

    '3' : begin
            Writeln('********* Real Autocorrelation *********');
            Auto := true;              { Autocorrelation  }
            WhichAnalysis := RA;
          end;
```

```
    '4' : begin
            Writeln('********** Real Crosscorrelation **********');
            Auto := false;              { Crosscorrelation  }
            WhichAnalysis := RC;
          end;

    '5' : begin
            Writeln('********** Complex Fast Fourier transform **********');
            GetInverse(Inverse);
            WhichAnalysis := CF;
          end;

    '6' : begin
            Writeln('********** Complex Convolution **********');
            WhichAnalysis := CN;
          end;

    '7' : begin
            Writeln('********** Complex Autocorrelation **********');
            Auto := true;               { Autocorrelation  }
            WhichAnalysis := CA;
          end;

    '8' : begin
            Writeln('********** Complex Crosscorrelation **********');
            Auto := false;              { Crosscorrelation  }
            WhichAnalysis := CC;
          end;
  end; { case }

  Writeln;
  Ch := InputChannel('Input Data From');
  if Ch = 'K' then
    repeat  { Until NumPoints1 := NumPoints2  }
      NumPoints1 := NumPoints2;
      case WhichAnalysis of
        RF, RA : GetRealVectorFromKeyboard(NumPoints, XReal);

        RN, RC : begin
                   Writeln;
                   Writeln('The first function:');
                   GetRealVectorFromKeyboard(NumPoints1, XReal);
                   Writeln;
                   Writeln('The 2nd function:');
                   GetRealVectorFromKeyboard(NumPoints2, HReal);
                   if NumPoints1 <> NumPoints2 then
                   begin
                     Writeln;
                     Write('The number of points in these two files');
                     Writeln(' are different.');
                   end;
                   NumPoints := NumPoints1;
                 end;
```

```
        CF, CA : GetComplexVectorFromKeyboard(NumPoints, XReal, XImag);

        CN, CC : begin
                   Writeln;
                   Writeln('The first function:');
                   GetComplexVectorFromKeyboard(NumPoints1, XReal, XImag);
                   Writeln;
                   Writeln('The 2nd function:');
                   GetComplexVectorFromKeyboard(NumPoints2, HReal, HImag);
                   if NumPoints1 <> NumPoints2 then
                   begin
                     Writeln;
                     Write('The number of points in these two files');
                     Writeln(' are different.');
                   end;
                   NumPoints := NumPoints1;
                 end;
      end { case }
    until NumPoints1 = NumPoints2
  else { Ch = 'F' }
    repeat
      NumPoints2 := NumPoints1;
      case WhichAnalysis of
        RF, RA : GetRealVectorFromFile(NumPoints, XReal);

        RN, RC : begin
                   Writeln;
                   Writeln('The first function:');
                   GetRealVectorFromFile(NumPoints1, XReal);
                   Writeln;
                   Writeln('The 2nd function:');
                   GetRealVectorFromFile(NumPoints2, HReal);
                   if NumPoints1 <> NumPoints2 then
                   begin
                     Writeln;
                     Write('The number of points in these two files');
                     Writeln(' are different.');
                   end;
                   NumPoints := NumPoints1;
                 end;

        CF, CA : GetComplexVectorFromFile(NumPoints, XReal, XImag);

        CN, CC : begin
                   Writeln;
                   Writeln('The first function:');
                   GetComplexVectorFromFile(NumPoints1, XReal, XImag);
                   Writeln;
                   Writeln('The 2nd function:');
                   GetComplexVectorFromFile(NumPoints2, HReal, HImag);
                   if NumPoints1 <> NumPoints2 then
                   begin
                     Writeln;
                     Write('The number of points in these two files');
                     Writeln(' are different.');
```

```
                    end;
                    NumPoints := NumPoints1;
                  end;
      end; { case }
    until NumPoints1 = NumPoints2;
  GetOutputFile(OutFile);
end; { procedure GetData }

procedure Results(NumPoints     : integer;
                  WhichAnalysis : Analyses;
              var XReal         : TNvectorPtr;
              var XImag         : TNvectorPtr;
                  Error         : byte);

{-------------------------------------------------------------}
{- This procedure outputs the results to the device OutFile -}
{-------------------------------------------------------------}

var
  Index : integer;

begin
  Writeln(OutFile);
  Writeln(OutFile);
  if Error >= 1 then
    DisplayError;

  case Error of
    0 : begin
          case WhichAnalysis of
            RF : Writeln(OutFile, 'Results of real Fourier transform:');
            RN : Writeln(OutFile, 'Results of real convolution:');
            RC : Writeln(OutFile, 'Results of real crosscorrelation:');
            RA : Writeln(OutFile, 'Results of real autocorrelation:');
            CF : Writeln(OutFile, 'Results of complex Fourier transform:');
            CN : Writeln(OutFile, 'Results of complex convolution:');
            CC : Writeln(OutFile, 'Results of complex crosscorrelation:');
            CA : Writeln(OutFile, 'Results of complex autocorrelation:');
          end; { case }

          for Index := 0 to NumPoints - 1 do
            Writeln(OutFile, XReal^[Index], '    ', XImag^[Index]);
          Writeln;
        end;

    1 : begin
          Writeln(OutFile, 'The number of data points: ', NumPoints);
          Writeln(OutFile, 'There are too few data points.');
        end;

    2 : begin
          Writeln(OutFile, 'The number of data points: ', NumPoints);
          Writeln(OutFile,
                  'The number of data points must be a power of two for a')
;
```

```
          Writeln(OutFile,
              'radix-2 transform or a power of four for a radix-4 transform.
');
        end;
  end; { case }
end; { procedure Results }
{-------------------------------------------------------------------------}
PROCEDURE INTROD;
{-------------------------------------------------------}
{ PROCEDURE FOR INTRODUCTION OF THE SOFTWARE PACKAGE }
{-------------------------------------------------------}
 BEGIN
    INITGRAPHIC;
    DEFINEWINDOW(1,0,0,XMAXGLB,YMAXGLB);
    DEFINEWORLD(1,0,0,1000,1000);
    SELECTWORLD(1);
    SELECTWINDOW(1);
    DRAWBORDER;
    DRAWTEXTW(100,100,3,'THIS    MODULE    OF    PROGRAM    USES ');
    DRAWTEXTW(450,200,3,'     FFT TECHNIQUE FOR          ');
    DRAWTEXTW(100,300,3,'EVALUATING FUNCTIONS LIKE AUTO,CROSS' );
    DRAWTEXTW(250,400,3,'CORRELATION, AND ITS  INVERSE          ');
    DRAWTEXTW(200,500,2,'BY');
    DRAWTEXTW(500,600,2,'VIJAY PANDE');
    DRAWTEXTW(200,700,2,'GUIDE');
    DRAWTEXTW(500,800,2,'PROF M.K.VASANTHA');
    DRAWTEXTW(200,900,2,'E.E.DEPT, UOR,  dt:16/2/1990');
    DRAWTEXTW(200,950,1,'PRESS ANY KEY TO CONTINUE ( EXCEPT ENTER KEY)');
    REPEAT UNTIL KEYPRESSED;
    LEAVEGRAPHIC;
    CLRSCR;
 END;
{-------------------------------------------------------------------------
}
begin { program FFTPrograms }
  introd;
  ClrScr;
  Initialize(NumPoints, XReal, XImag, HReal, HImag, Error);
  GetData(NumPoints, WhichAnalysis, Auto, XReal, XImag, HReal, HImag);

  case WhichAnalysis of
    RF : RealFFT(NumPoints, Inverse, XReal, XImag, Error);
    RN : RealConvolution(NumPoints, XReal, XImag, HReal, Error);
    RC : RealCorrelation(NumPoints, Auto, XReal, XImag, HReal, Error);
    RA : RealCorrelation(NumPoints, Auto, XReal, XImag, XReal, Error);
    CF : ComplexFFT(NumPoints, Inverse, XReal, XImag, Error);
    CN : ComplexConvolution(NumPoints, XReal, XImag, HReal, HImag, Error);
    CC : ComplexCorrelation(NumPoints, Auto, XReal, XImag,
                            HReal, HImag, Error);
    CA : ComplexCorrelation(NumPoints, Auto, XReal, XImag,
                            XReal, XImag, Error);
  end; { case }

  Results(NumPoints, WhichAnalysis, XReal, XImag, Error);
  Close(OutFile);
```

APPENDIX - VI

RESULTS

ProcessPlus TABLE OF CONTENTS

DATA SCREENS

A - This table of contents
P - Process
C - Controller and PID values
M - Model-predictive controller
T - calculated PID controller Tuning
U - process identifier (oscillation)
S - process identifier (Step data)
O - plotting Options
R - disk file Read-write
D - System (to do DOS comman
X - Exit ProcessPlus
H - (or F1) Help

TIME RESPONSE PLOTS

1 - Setpoint
2 - Setpoint ramp
3 - Load before process
4 - Load after process

FREQUENCY RESPONSE PLOTS (Open Loop)

5 - Process with Controller
6 - Controller with inverse proc
7 - Combined
8 - Robustness plot and stabilit

PRESS THE "H" KEY NOW FOR HELP WITH THE DEMO

Press a letter (for data screen), number (for plot), H or F1 for hel

PROCESS

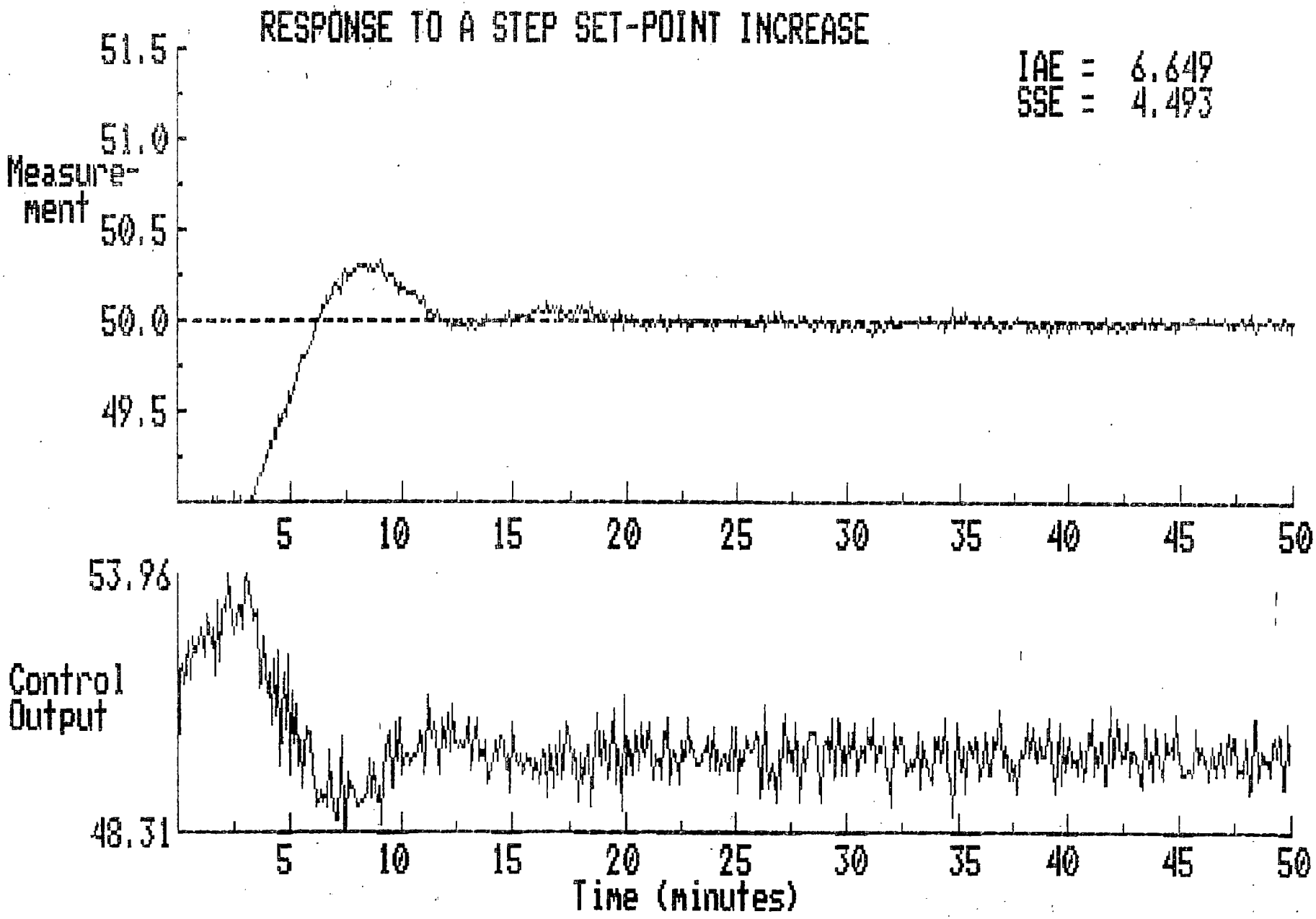
$$\text{Process} = \frac{GP\,(LTs + 1)\,e^{-DTs}}{(C0 + C1s + C2s^2 + C3s^3)\,(T1s + 1)(T2s + 1)(T3s + 1)(T4s +}$$

Gain: GP= .9285 Dead time: DT= 3.1
Lead time: LT= 0.
Lag time(s): T1= 0. T2= 0. T3= 0. T4= 0.
Polynomial coefficients: C0= 1. C1= 10. C2= 0. C3= 0.

Sample interval= .1 Time units (1=sec, 2=min, 3=hr):
oad size= 1.4 3Load noise= 0. 3Measurement noise= .02 3% Hysteresis

PRESS THE "H" KEY NOW FOR HELP WITH THE DEMO

move3Type value then DY3Esc -abort3F1 or H -help3q or PgDn -exit:


RESPONSE TO A STEP SET-POINT INCREASE
IAE = 6.649
SSE = 4.493
Measure-
ment
51.5
51.0
50.5
50.0
49.5
5
10
15
20
25
30
35
40
45
50
53.96
Control
Output
48.31
5
10
15
20
25
30
35
40
45
50
Time (minutes)