

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

ON-LINE SYSTEM IDENTIFICATION FOR
CONTROL SYSTEM APPLICATIONS IN
PARTICLE ACCELERATORS

A Dissertation

Presented to

The Faculty of the Department of Applied Science
The College of William and Mary in Virginia

In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy

by

Mahesh Chowdhary

1996

UMI Number: 9805155

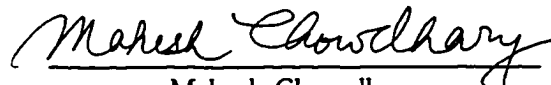
UMI Microform 9805155
Copyright 1997, by UMI Company. All rights reserved.

**This microform edition is protected against unauthorized
copying under Title 17, United States Code.**


UMI
300 North Zeeb Road
Ann Arbor, MI 48103

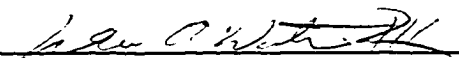
APPROVAL SHEET

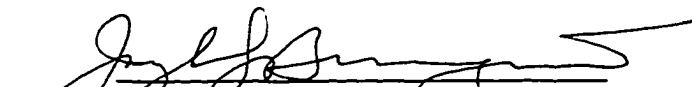
This dissertation is submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

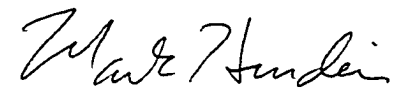

Mahesh Chowdhary

Approved, December 1996


Dennis M. Manos, Ph.D.
CSX Professor of Applied Science and Physics


William A. Watson, III, Ph.D.
Adjunct Professor of Applied Science


Joseph J. Bisognano, Ph.D.
Adjunct Professor of Applied Science and Physics


Mark K. Hinders, Ph.D.
Assistant Professor of Applied Science and Physics

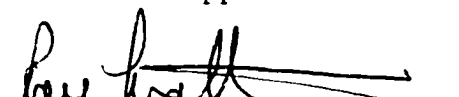

Roy C. Mathias, Ph.D.
Associate Professor of Mathematics

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	v
LIST OF FIGURES	vi
ABSTRACT	ix
CHAPTER I INTRODUCTION	2
CHAPTER II. FAST FEEDBACK SYSTEMS AT CEBAF	12
CHAPTER III APPLICATION OF ON-LINE SYSTEM IDENTIFICATION	40
CHAPTER IV ANALOG COMPUTER SIMULATION	65
CHAPTER V SUMMARY OF RESULTS AND CONCLUSIONS	87
APPENDIX A	92
APPENDIX B	95
ENDNOTES	147
BIBLIOGRAPHY	152

ACKNOWLEDGMENTS

Author wishes to gratefully acknowledge contributions of the following individuals to this work :

Dr. William A. Watson for his support to this work and his technical advice regarding real-time implementation issues. Dr. Jer-Nan Juang of NASA Langley Research Center for his technical advice regarding system identification algorithms. Dr. Dennis Manos for encouraging and supporting this work and providing inspiration. Dr. Andrew Hutton for providing an opportunity to pursue this research. Drs. Bisognano, Hinders, and Mathias for sparing their precious time to serve on the committee and providing constructive criticism and support.

LIST OF TABLES

Table	Page
1. Beam parameter variations observed at IPM1S10	20
2. Beam parameter variations observed using 500 Hz data acquisition system	21
A.1 Beam requirements - general characteristics	92

LIST OF FIGURES

Figure	Page
1. Schematic diagram of CEBAF.....	12
2. Power spectrum for X-position observed at IPM1S10.....	14
3. Time domain data for X-position observed at IPM1S10	15
4. Power Spectrum for beam energy variation observed at IPM1S10	16
5. Time domain data for beam energy variation observed at IPM1S10.....	17
6. Power Spectra for states X, X', Y, and Y' observed at IPM2A01	18
7. Power spectrum for $\Delta E/E$ variations at IPM2A01.....	19
8. Power spectrum for X-position at location IPM3C07.....	22
9. Phase variation of 60 Hz noise component observed at location IPM3C07 in Hall C line.....	23
10. Screen Capture image of GUI for Fast Energy Feedback System	36
11. $\Delta E/E$ with and without feedback correction (sample rate 60 Hz).	38
12. Power spectra of $\Delta E/E$ before and after correction signal was applied.....	39
13. Comparison between Output 1 and the difference between the estimated and actual output for the noise free case.	62
14. Estimation error for Output 1 for noise free case.	63
15. Comparison between output 1 and estimation error for the case where 10% random noise was added to input/output data.	64
16. Schematic diagram of the system simulated using an analog computer	70
17. Comparison between actual and identified output 1. The dynamics of simulated plant was changed at 4 instances by varying attenuators R11, and R12.....	72

18. Comparison between actual and identified output 2. The dynamics of simulated plant affecting state 2 was changed at 6 instances by varying attenuators R21 and R22.	73
19. Estimation error between actual and identified output 1.....	74
20. Disturbance suppression achieved by application of feedback correction using input 1 for the case where system dynamics was changed.....	75
21. Comparison between identified and actual output 1 for the case where 5% random noise was added to measurement and control input data and at points d11 and d12.	76
22. Estimation error between actual and identified output 1 for the case where 5% random noise was added to points d11 and d12.....	77
23. Disturbance suppression achieved after application of feedback correction using input 1 for the case where 5% random noise was added at points d11 and d12.	78
24. Comparison between actual and identified output 1 for the case where amplitude and frequency of noise signal applied at d11 and d12 was changed.	79
25. Estimation error between actual and identified output 1 for the case where amplitude and frequency of noise were changed.....	80
26. Disturbance suppression achieved by application of feedback correction using input 1 for the case where amplitude and frequency of noise were changed.....	81
27. Comparison between actual and identified output 1 for the case where dynamics of the system and noise characteristics were changed simultaneously.....	82
28. Estimation error between actual and identified output 1 where system dynamics and noise characteristics were varied simultaneously.....	83
29. Disturbance suppression achieved by application of feedback correction using input 1 for the case where system dynamics and noise characteristics were varied simultaneously.....	84
30. Loss function vs. Model order	86

31. CPU time/FTF iteration vs. Model order..... 89

ABSTRACT

Particle accelerators require a number of feedback systems in order to stabilize a variety of parameters. The Continuous Electron Beam Accelerator at Thomas Jefferson National Accelerator Facility presents a unique set of control and identification problems. This accelerator produces a continuous electron beam with energies between 0.5 and 4.0 GeV to be delivered to the experimental halls. In order to meet stringent beam quality requirements specified by the experimental halls, the position and the energy of the electron beam needs to be stabilized at various locations in the accelerator.

A number of noise measurement tests were conducted at various locations in the accelerator to obtain accurate information about the amplitude and the frequency of disturbances on the beam orbit and energy. Results of these measurements indicate that the line power harmonics were the primary source of disturbance on the beam orbit and energy.

A prototype fast feedback system was implemented in the injector and the East Arc regions of the accelerator to stabilize the beam position and energy at these locations. The scheme of implementation of these systems and measurements of their performance are presented here.

These feedback systems have to operate under conditions of varying noise characteristics and changing dynamics of the systems. For the feedback systems to

always perform optimally, the knowledge of time varying noise characteristics and changing system dynamics needs to be incorporated into the feedback strategy. The approach presented in this work is to perform on-line system identification using a formulation of Fast Transversal Filter (FTF) in order to extract the time varying information from input/output data of the feedback system.

A simulation test stand was developed using an analog computer to represent a continuous time system whose noise characteristics and dynamics could be changed in a controlled manner. An on-line system identification algorithm was implemented on a microprocessor similar to the ones used in the accelerator control system. Experience with the hardware-in-loop simulation for various cases of changing system dynamics and noise characteristics and the performance results of the on-line system identification algorithm operating under these conditions are presented in this dissertation.

Mahesh Chowdhary
Applied Science Department
Adviser: Dr. Dennis M. Manos
CSX Professor of Applied Science and Physics

ON-LINE SYSTEM IDENTIFICATION FOR CONTROL SYSTEM
APPLICATIONS IN PARICLE ACCELERATORS

CHAPTER I

INTRODUCTION

1.1 Background

The age of modern control theory was ushered in with the launching of first Sputnik in 1957. During the last several decades, developments in control theories have continued and applications of these theories in various aerospace, military and civilian industries have increased tremendously. Availability of powerful and inexpensive digital computers has been a crucial reason for this success. Digital computers have been used not only to implement control algorithms, but also to develop, validate and test control theory applications. The necessity of controlling large and complex physical plants, such as a major petroleum refinery, or a jumbo airliner, is another important reason for proliferation of control theories in various industries.

The core problem of controlling a physical plant is to determine appropriate control forces which can assure that the physical plant behaves in the desired fashion. For linear systems, the “state-feedback” strategy is the most common technique used in calculating the control forces. State information of a dynamic system is a set of physical quantities, the specification of which, in the absence

of external disturbances, completely determines the time evolution of the system. However, in general, the state information cannot be measured directly using available sensors. Hence, a technique for extracting state information from measured data is essential to a feedback control strategy.

Since the performance of a feedback system is dependent on the accuracy of reconstructed state, effective reconstruction of state from the measured data is very important. There are various factors involved in reconstructing state information from input/output data of a physical system. The measured data is almost always contaminated by noise from imperfect sensors. The number of sensors is usually less than the number of states of interest, so measured data at a given time alone is not sufficient to determine the state, and previous data has to be used. Since systems are usually affected by unpredictable, time-varying noises, uncertainty is introduced between the previous data and the current state. This problem is further complicated when the dynamics of the system also varies with time.

1.2 Literature Survey

From earliest times people have been concerned with estimating unknown quantities from observed data and making prediction. The least squares method has been an important milestone in the development of estimation theory. This well known method was apparently used by Gauss in 1795 in his studies of astronomy, though it was first published by Legendre in 1805. Since then, there

have been vast amounts of literature on various aspects of least squares method. A survey of such work on least square estimation for random variables has been presented by Sorenson [1].

First studies in applying least square estimation to stochastic process were done by Kolmogrov, Krein, and Wiener [2]-[3]. Kolmogrov applied the mean square theory to the prediction problem for discrete-time stationary process. A process is called stationary if all its statistical parameters are invariant to a translation in time. Kolmogrov and Krein's work did not focus on optimality of the predictor. Wiener [2] formulated the continuous-time linear prediction problem and derived an expression for optimal predictor. Various practical applications such as anti-aircraft fire control mechanism benefited from this work. Wiener developed the first explicit solutions of least square estimates of stochastic process [2]. Wiener's work was further extended by Van Trees [4]-[5], Stiffler [6], and Lindsey [7] into fields such as modulation theory.

Kalman changed the conventional least squares problem by developing a model [8] (commonly known as state space model) for a signal process. He described the signal process $y(t)$ with a system of equation described as

$$\begin{aligned}y(t) &= Cx(t) \\ \dot{x}(t) &= Ax(t) + Bu(t)\end{aligned}\tag{1}$$

In 1960 Kalman published his renowned method for sequential state estimation of discrete systems, known as the Kalman filter [9], using a state

space formulation. Two years later he published another version [10] of the Kalman filter for continuous time systems. Significant contribution in system realization theory in terms of concepts of controllability and observability of system were presented by Gilbert [11] and Kalman [12] in the same time period. Since then a large number of papers were published showing the importance of the Kalman filter, and at the same time revealing its unsatisfactory features.

A well known limitation in applying the conventional Kalman filter is its requirement of a priori knowledge about the system state space model and the covariances of process and measurement noises. This data in most cases is either partially known or completely unknown. Another drawback of the conventional Kalman filter is its inability to adjust itself to trace a changing environment or correct for the error caused by incorrect a priori information. After reaching steady state the filter “sleeps”, which means no matter how large the estimation error gets, the filter remains unaffected. This phenomenon is called filter divergence [13]-[16].

If the system state space model is known, but the noise covariances are unknown, then one must estimate noise statistics, or conduct a systematic analysis to provide noise covariances or filter gain in order to use the Kalman filter [17]-[21]. Adaptive filtering techniques [23]-[26] need to be used in order to improve upon the a priori assumptions made for the filter design. Adaptive Kalman filtering [25]-[26] uses the Kalman filter structure, but modifies the

scheme for computing the filter gain. The filter monitors the estimation error and improve its performance accordingly. Most existing adaptive Kalman filters and methods of estimating noise covariances or filter gain are complicated and are not suitable for on-line implementation. Furthermore, adaptive Kalman filtering methods are derived under the assumption that the system state space model is accurately known and that it does not change with time. The problem of adaptive Kalman filtering for unknown systems is more complicated. Goodwin [23] introduced a method of estimation for uncertain systems where the state vector is augmented by undetermined system parameters. Using this scheme the system parameters and states can be estimated at the same time. However, a nonlinear state estimation technique such as an extended Kalman filter has to be used for such systems because the system model becomes nonlinear due to state augmentation. To solve a nonlinear estimation problem, the system is usually linearized at each estimated state. The linearization is computationally inefficient for large order system and convergence of the estimate is not guaranteed.

To solve the problem of state estimation under unknown model and noise covariances, a system model needs to be identified before state estimation can be done. There are various cases where direct mathematical model generation is not possible. In some cases the knowledge of a system's mechanism is not completely known. In other cases the properties exhibited by the system may

change with time in an unpredictable manner. Further more, modeling process can be very time consuming and can lead to models that are unnecessarily complex. In many of the above mentioned cases the signals produced by the system can be measured and used for computation of mathematical models. This approach of system identification has been applied to solve many practical problems.

Astrom [27] and van Amerongen [28] have studied the problem of a ship-steering regulator. A ship's heading angle and position is controlled by its rudder angle. For a large ship, the position control is a fairly difficult problem because ship's response to a change in rudder angle is very slow and is affected by random components such as wave motion and wind. Most ships have a regulator that measures relevant variables and determines the rudder angle. The design of the regulator is based on the steering dynamics of the ship, which depend on a number of factors such as the shape and the size of ship, loading and trim, and water depth. Some of these factors vary during the journey; the disturbances from wind and waves may also change rapidly. Therefore, the regulator has to be constantly retuned to match the current dynamics of the system. This work is presented in references [27] and [28]. Many control problems in airplanes and missiles exhibit similar features where dynamic properties depend on speed, loading etc. and change with time. Machinery in paper-making plants is affected by many randomly changing factors. Researchers such as Landau [29] have

presented work in this area.

Short term prediction of power demand from an electricity generation system is a similar problem that requires good design of an adaptive predictor. There is a substantial random component in the power demand which depends on the time of the day and the day of the week, month or year. For efficient production of electricity a good prediction of demand a few hours ahead of time is very useful. Research in the area of power demand predictors is presented by Gupta and Yamada [30].

Channel equalization in communication networks is another problem where recursive techniques are very useful. Each channel in a communication network can be seen as a linear filter with a certain impulse response characteristic that differs from the ideal delta function response, distort the transmitted signal. The signal must be restored at the receiver using an equalizer, which is a filter whose impulse response resembles the inverse of that of the channel. However, in a communications network the channel between transmitter and receiver can be quite arbitrary and therefore it is desirable that the equalizer adapt itself to the actual properties of the chosen channel. Work done in this area has been published by Lucky [31] and Goddard [32].

Active control of aerospace structures is another complex problem. Large aerospace structures can accurately be represented by large mathematical models with dimensions in the order of hundreds. Besides, most aerospace structures

possess significant uncertainties and nonlinearities which make identification of mathematical model very difficult. A modal parameter identification algorithm for flexible structures known as Eigensystem Realization Algorithm (ERA) was developed by Juang [33] in 1984. A frequency-domain ERA and a recursive ERA were also developed by Juang [34].

1.3 Problem Statement

The Continuous Electron Beam Accelerator (CEBAF) at Jefferson Lab presents a unique set of control and identification problems. The accelerator produces a continuous electron beam, with energies between 0.5 and 4.0 GeV, to be delivered to three experimental halls. The beam quality requirements specified by the experimental halls are very stringent, (see table A.1 in Appendix A). In order to meet these requirements the position and energy of the electron beam needs to be stabilized at various locations in the accelerator against various disturbances. Feedback systems that regulate the position and energy of the beam at several locations in the accelerator have to operate under time-varying noise characteristics and changing dynamics of the system. A much clearer picture of the problem will be developed in Chapter 2 of this dissertation where details of the feedback system and noise measurement analysis and results are presented.

For the feedback systems to always perform optimally the knowledge of time-varying noise characteristics and changing system dynamics need to be incorporated

into the feedback strategy. One approach to solve this problem is to perform on-line system identification and extract the time-varying information from the input/output data of the feedback system. The theory, analysis, simulation, and implementation of this approach are presented in this dissertation.

1.4 Dissertation Outline

Chapter 2 presents with a description of the Continuous Electron Beam Accelerator. The relevant subsystems needed for beam orbit and energy stabilization are described in that chapter. Various noise measurements that were performed and various systems that were used to perform these measurements are described. The feedback algorithm and the performance measurements obtained with the feedback systems are described in that chapter.

Chapter 3 describes the theory and implementation of an on-line system identification algorithm. The mathematical formulation of the Fast Transversal Filter (FTF) is described there. Details about the implementation of FTF using VxWorks as the real-time operating system on a MVME167 VMEbus CPU is also described. The catchup technique, which is used to further improve the performance of the FTF implementation, is described.

Chapter 4 contains details about the simulation test stand that was used to test the implementation of FTF using an analog computer for hardware-in-the-loop

simulation. Description of an analog computer used to simulate a continuous-time plant whose dynamics and noise characteristics can be varied in a controlled manner is presented in that chapter. Results from performance measurements of the implementation of the FTF algorithm for on-line system identification of a continuous system simulated using the simulation test-stand are also described.

Chapter 5 presents a review and a summary of various results obtained and described in previous chapters. Conclusions from this study are also presented. Future directions of research from the present work are also presented.

CHAPTER II

FAST FEEDBACK SYSTEMS FOR CEBAF

2.1 Overview

The Continuous Electron Beam Accelerator (CEBAF) provides a continuous beam of electrons at any energy between 0.5 and 4.0 GeV. The CEBAF accelerator, Fig. 1, consists of a 45 MeV injector, two side-by-side superconducting linacs, and 9 arcs that recirculate the beam through the linacs up to 5 times for 4 GeV total energy.

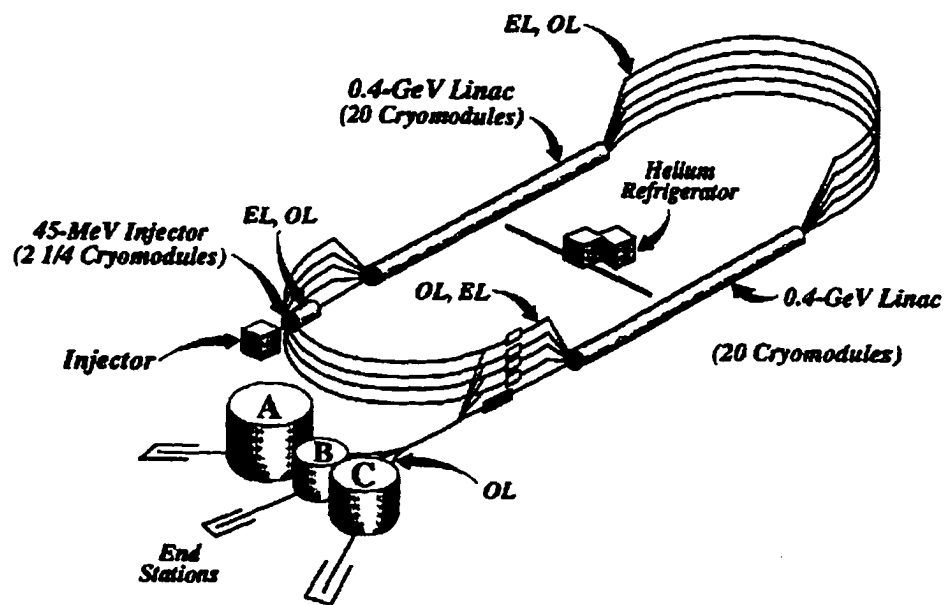


Fig. 1 Schematic Diagram of CEBAF

The key component of the superconducting linac is the superconducting accelerating cavity which allows for continuous acceleration of beam without excessive power loss in the cavity walls. Beams of different energies are separated at the first spreader and are transported through isochronous arcs to the recombiner at the entrance to the second linac. At the exit of the second linac, the beams of different energies are separated again to be sent to either one of the experimental halls or through the recirculation arcs.

The stability and quality of electron beam to be delivered to the Experimental Halls are important for experimenters. Few noise measurement exercises were conducted in order to determine the variation in orbit and energy of the beam at various locations in the accelerator.

2.2 Noise Measurements

A number of tests were conducted to obtain accurate information about the amplitude and frequency of disturbances on beam position and energy when the accelerator was operating in CW mode. These tests used Switched Electrode Electronics (SEE) Beam Position Monitors (BPM) [35] low level data acquisition software which can acquire beam position data at 119 kHz into 32k buffers / per antenna/ per BPM. A detailed description of the SEE BPM hardware and electronics is presented in section 2.5.1.2. When appropriately triggered, a routine in the SEE BPM low level data acquisition software acquires beam position data at 119kHz rate and stores it in on-board memory. Once the

required amount of data (32k buffer / per antenna/ per BPM) is collected, it is transferred to an ASCII file on a Unix host. The first series of tests were done using 5 SEE BPMs in the first pass line of the East Arc.

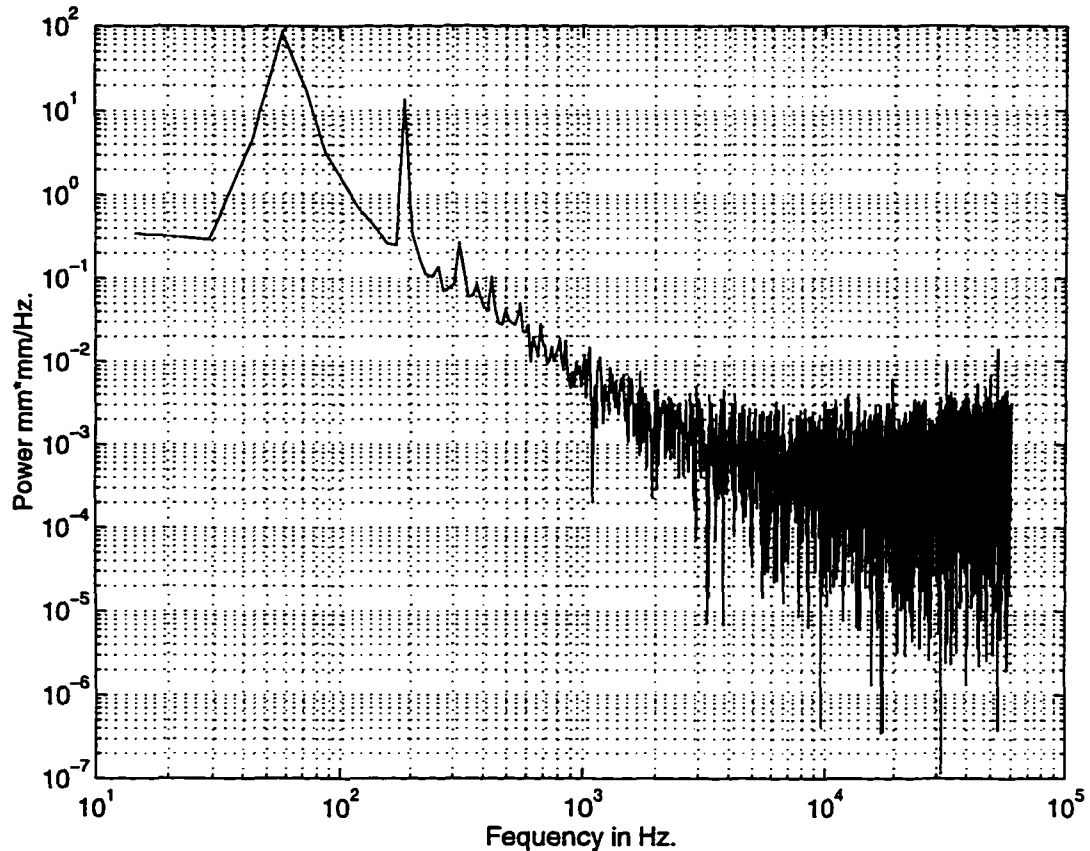


Fig. 2 Power Spectrum for X position observed at IPM1S10

The beam position data was later separated into beam orbit (position and angle in X and Y plane) and beam energy variations at a point in the vicinity of these five BPMs using a design model of this region of the accelerator. Fig. 2 shows a power spectrum of beam position variation in X direction in frequency domain. As seen from Fig. 2, the largest disturbance component in X position variation is centered at 60 Hz and the second peak is seen at 180 Hz.

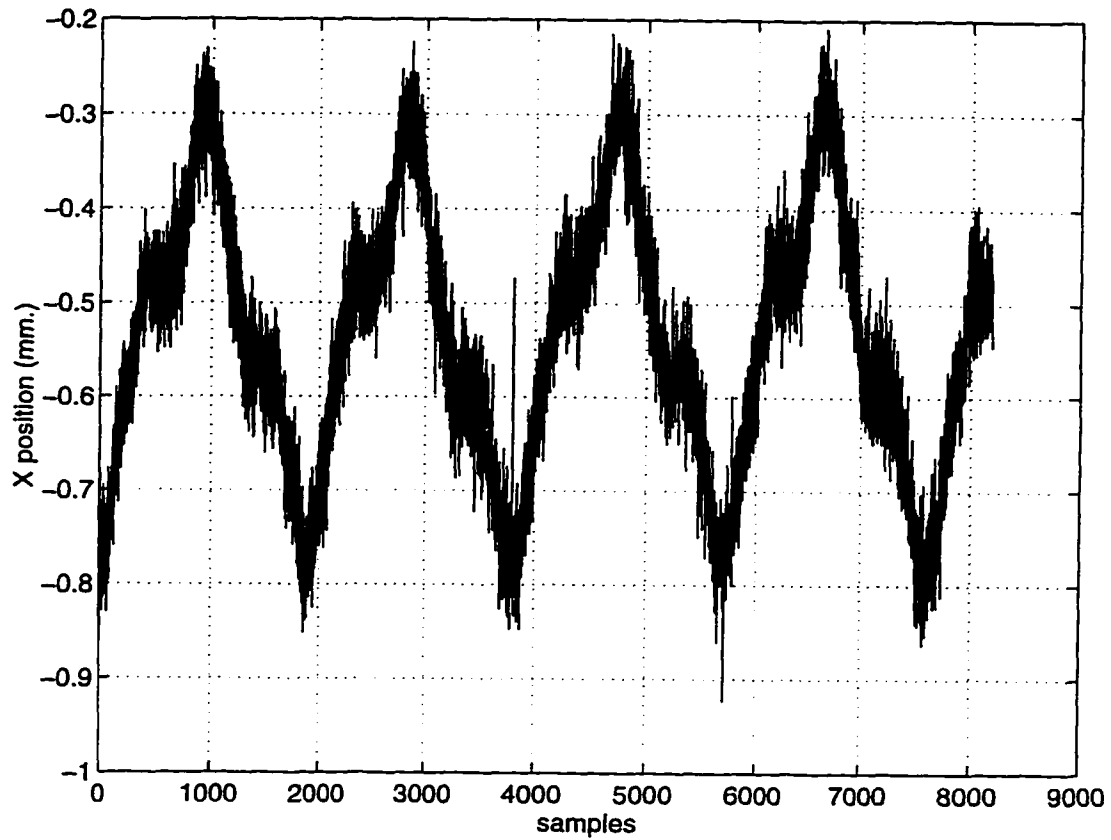


Fig. 3 Time Domain data for X-position observed at IPM1S10

Fig. 3 shows that the 60 Hz component of disturbance corresponds to a beam motion of approximately 0.6 mm in X plane. Time domain data and power spectrum for the beam energy variation are shown in Fig.(s) 4 and 5.

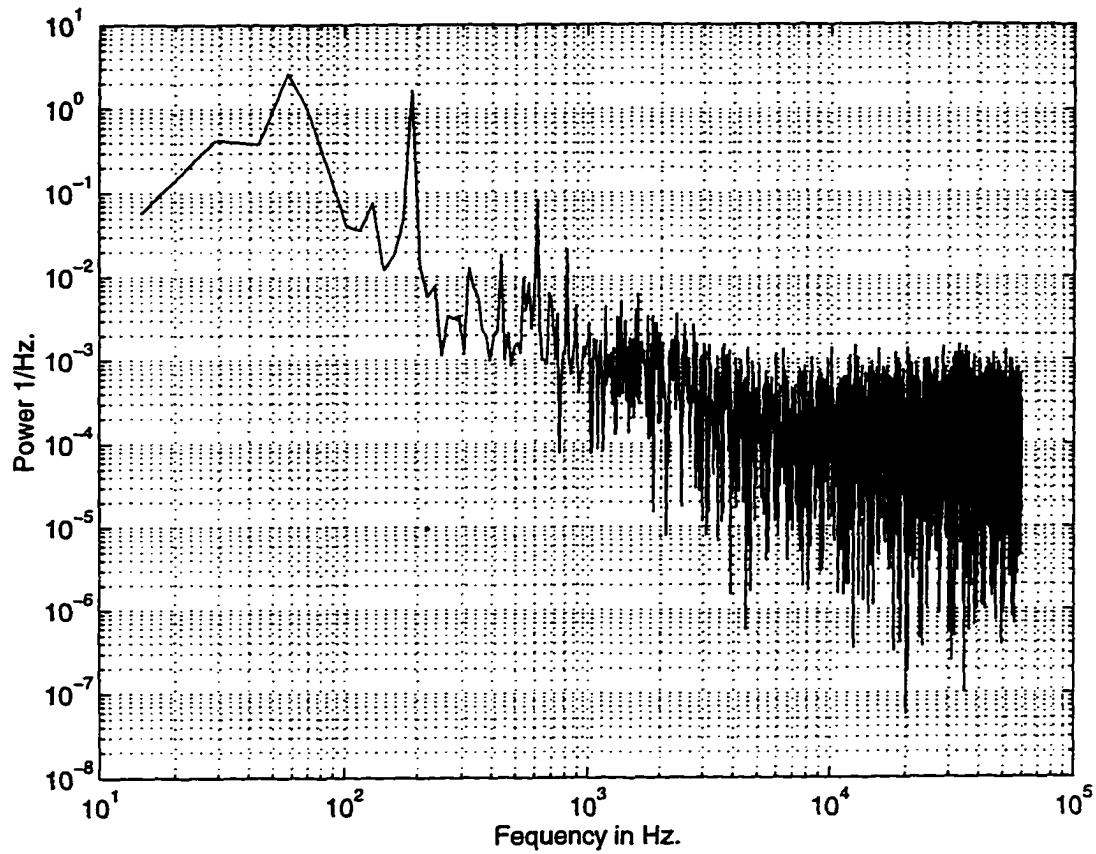


Fig. 4 Power Spectrum for beam energy variation observed at IPM1S10

The largest component of disturbance on beam energy again is centered at 60 Hz and the second largest component is located at 180 Hz. From Fig. 5 it can be seen that peak to peak energy variation is approximately 1.2×10^{-4} .

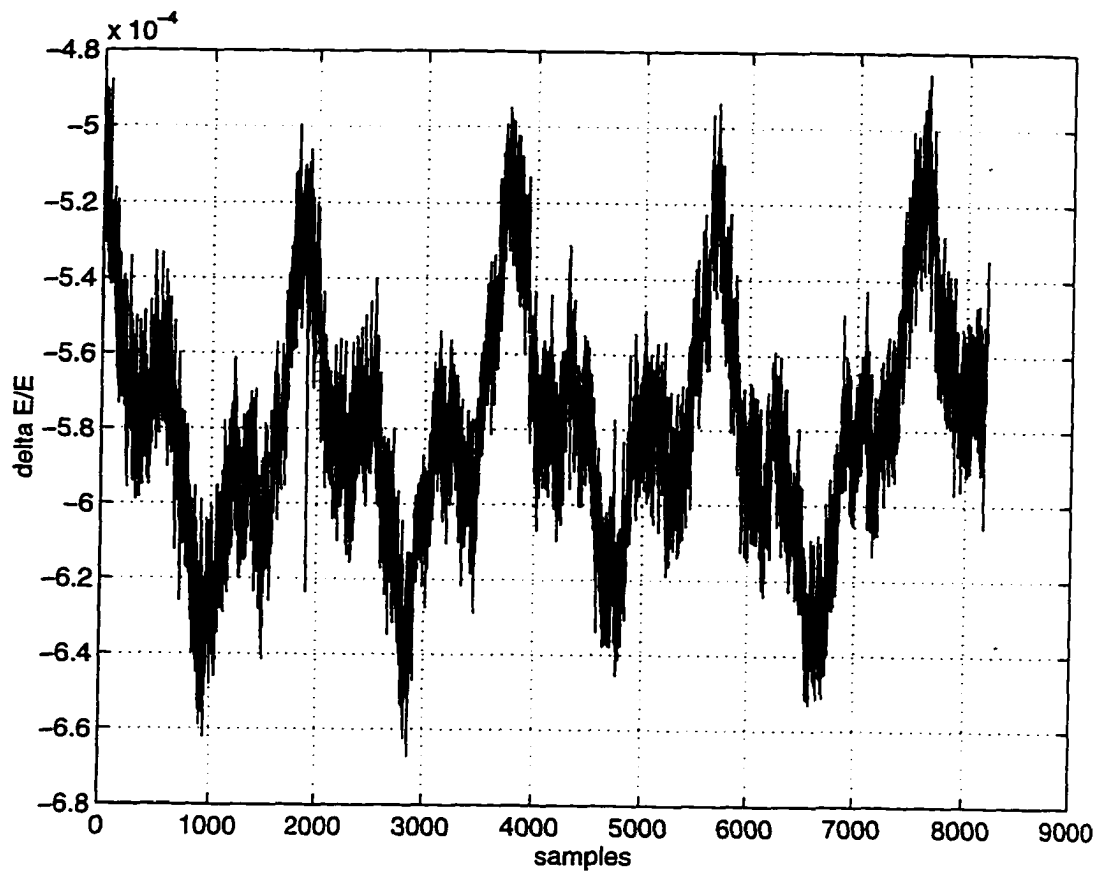


Fig. 5 Time domain data for beam energy variation observed at IPM1S10

Because the largest component of disturbance is observed at 180 Hz, another system known as 500 Hz system, was developed to simultaneously measure and quantify the noise on beam orbit and energy at different locations in the accelerator. This system samples the beam position at 500 Hz rate from various 4 channel electronics BPMs [36] at various location in the accelerator

simultaneously and stores the data in an ASCII file for later analysis. The analysis of the 500 Hz system data produced results similar to those from the SEE BPM system. The disturbance components associated with line power, at 60 Hz and 180 Hz, were observed in the data on beam orbit and energy. The power spectra and time domain data for one data set are shown Fig.(s) 6 and 7.

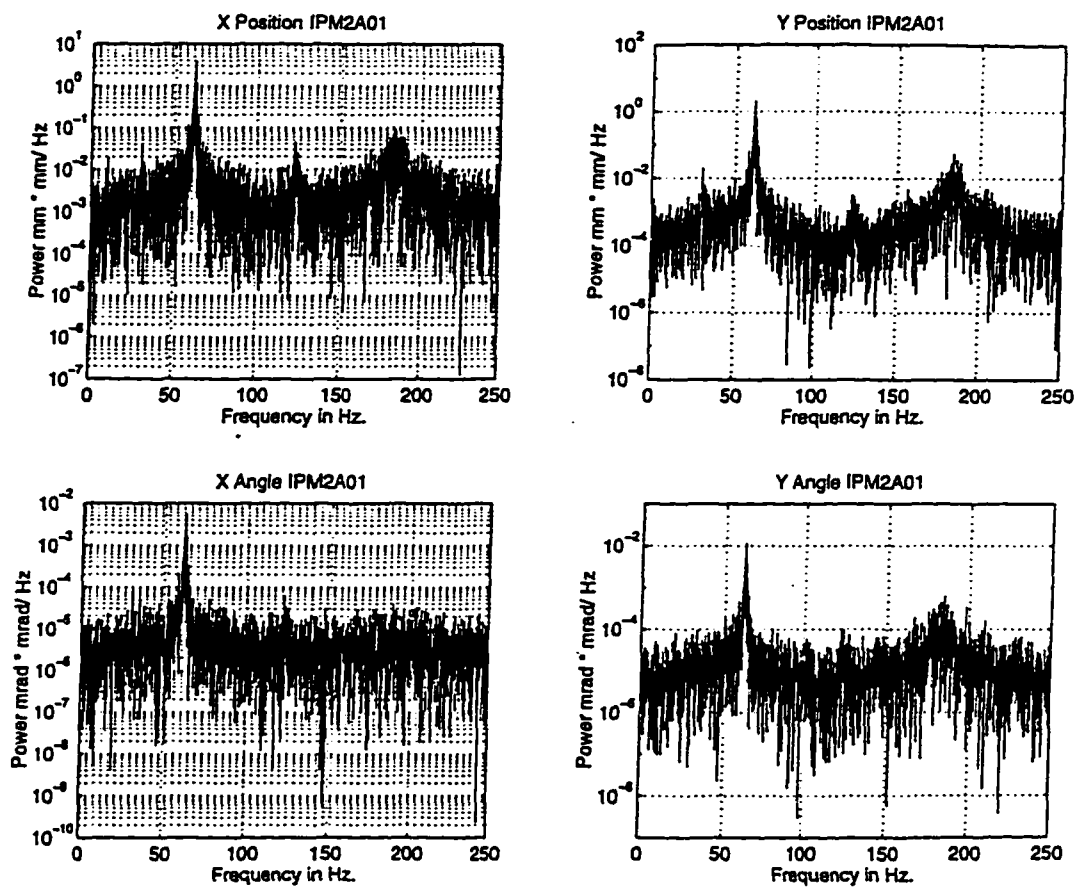


Fig. 6 Power Spectra for states X, X', Y, and Y' observed at IPM2A01

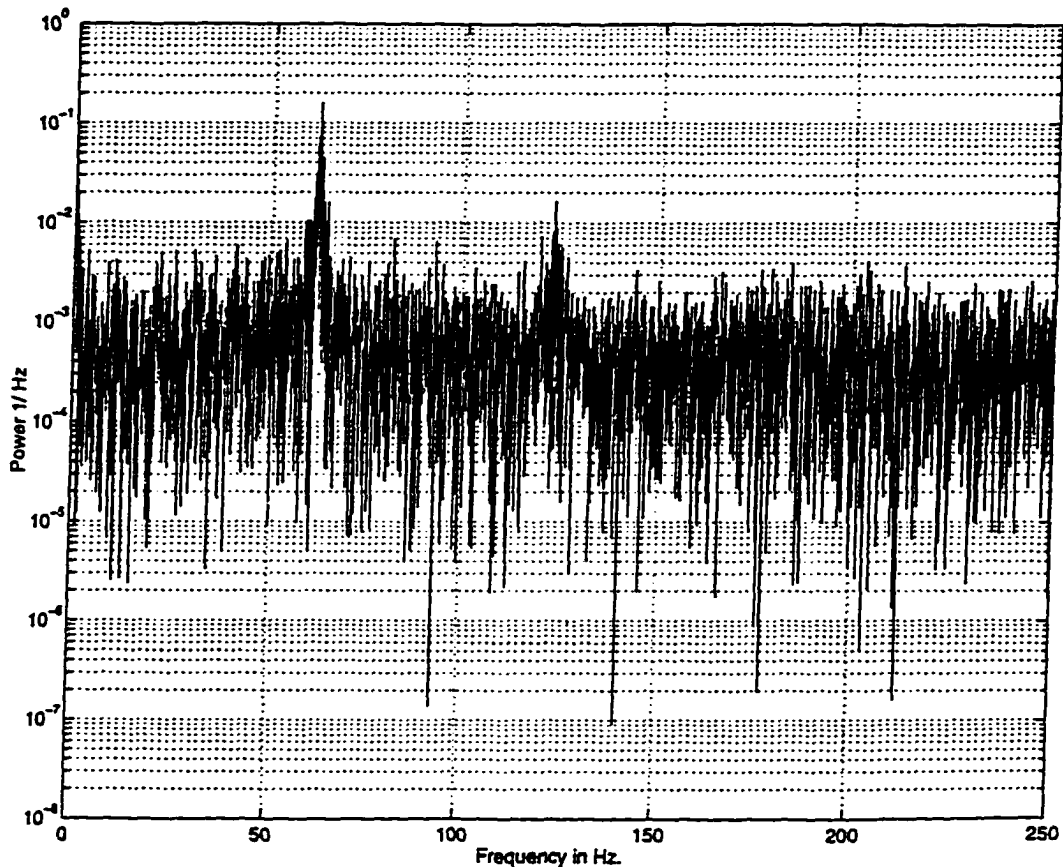


Fig. 7 Power spectrum for $\Delta E/E$ variation at IPM2A01

2.2.1 Variations in Noise Statistics

The noise components observed on the beam orbit and energy have exhibited variations in both amplitudes and frequencies for different sets of data. The variation in noise characteristics observed using five SEE BPMs in the first pass line of East Arc is shown in Table 1. The states X, X', Y, Y', $\Delta E/E$ displayed in Table 1 correspond to location IPM1S10 in the East Arc region. The fast acquisition data was collected using five SEE BPMs in the first pass line of East

Arc at a time interval of approximately 2 minutes. This data indicates that there is more than 20% variation in the amplitude of the states.

Table 1 BPM parameter variation observed at IPM1S10

Data Set	X (mm)	X' (mrad)	Y (mm)	Y' (mrad)	$\Delta E/E$
2	0.714	0.0914	0.2811	0.0916	3.51×10^{-4}
3	0.5616	0.0801	0.4421	0.1451	1.818×10^{-4}
4	0.5791	0.0772	0.1691	0.054	1.838×10^{-4}
5	0.5913	0.0774	0.2309	0.0683	2.026×10^{-4}
6	0.5857	0.0771	0.2113	0.0661	1.845×10^{-4}

Table 2 shows the noise statistics collected using the 4-channel electronics BPMs in various regions of the accelerator as indicated in the table. The amplitude variation in beam orbit (X, Y), beam trajectory (X', Y') and energy ($\Delta E/E$) are shown for various regions in the accelerator. This data was collect over a period of several days as indicated by time and date for each entry in the table

Table 2 Beam parameter variations observed using 500 Hz system

Location	X (mm)	X' (mrad)	Y (mm)	Y' (mrad)	$\Delta E/E$	Date
INJ0L10	1.9861	0.2659	0.9356	0.2119	1.1762×10^{-3}	4/1/96 15:15
INJ0L10	1.98	0.2532	0.8116	0.1956	1.018×10^{-3}	4/18/96 14:27
INJ0L10	2.134	0.3005	0.9025	0.2123	1.385×10^{-3}	5/15/96 13:40
EA1A01	0.6055	0.0425	0.2378	0.0926	0.2041×10^{-3}	3/29/96 16:31
EA1A01	0.4492	0.0321	0.1985	0.0907	0.2979×10^{-3}	4/1/96 15:15
EA1A01	0.3817	0.0266	0.2131	0.07	0.2703×10^{-3}	4/18/96 14:27
EA1A01	0.492	0.0474	0.2602	0.0675	0.3192×10^{-3}	5/15/96 13:40
WA2A01	0.4444	0.016	0.2713	0.0784	0.2021×10^{-3}	3/29/96 16:31
WA2A01	0.4348	0.0185	0.2876	0.0344	0.1586×10^{-3}	4/1/96 15:15

The power spectrum for X-position at location IPM3C07 is shown in Fig. 8. This power spectrum shows two distinct peaks around 60 Hz frequency. Upon further analysis, the cause for two peaks was found to be variation in the phase of 60 Hz noise component at approximately 1 Hz rate. Fig. 9 shows the phase variation of 60 Hz noise component observed in data acquired from 4-channel BPMs in the Hall C line.

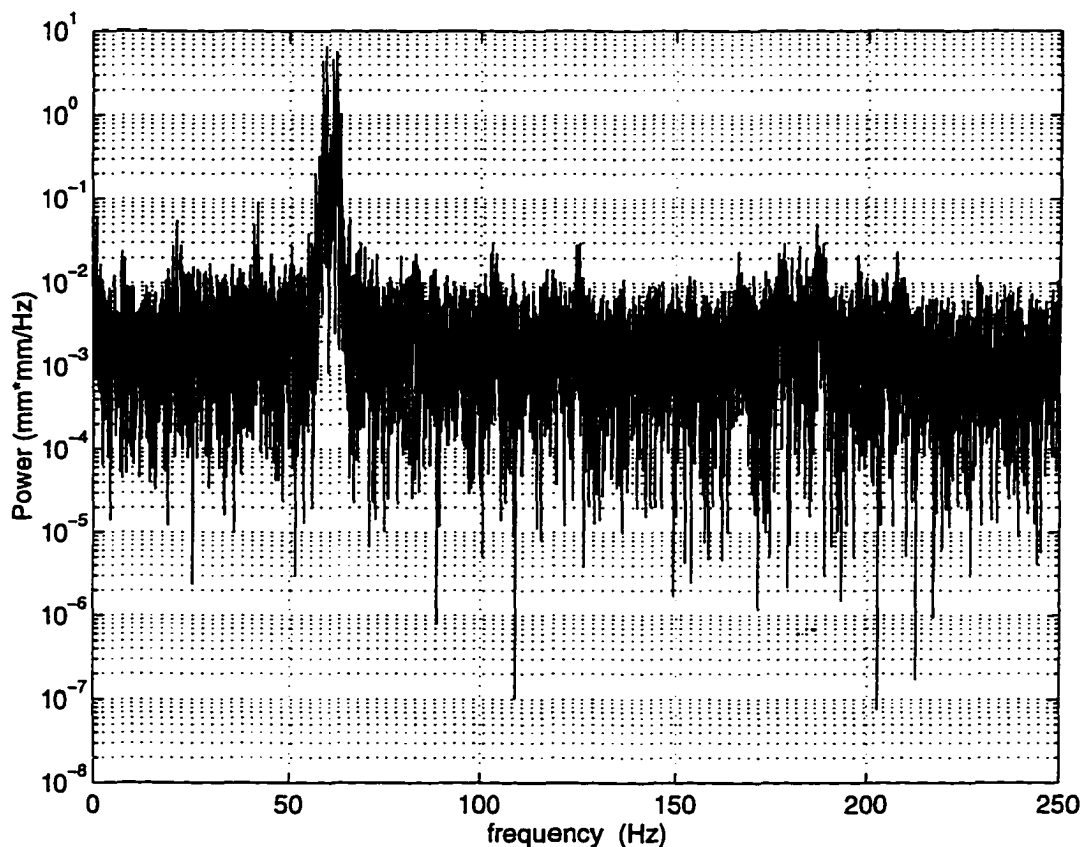


Fig. 8 Power spectrum for X-position at location IPM3C07

This phenomena of phase variation of 60 Hz component in the Hall C line is still under investigation. The data acquisition scheme of the 500 Hz system is

being investigated to eliminate timing problems as a possible source of this phase variation.

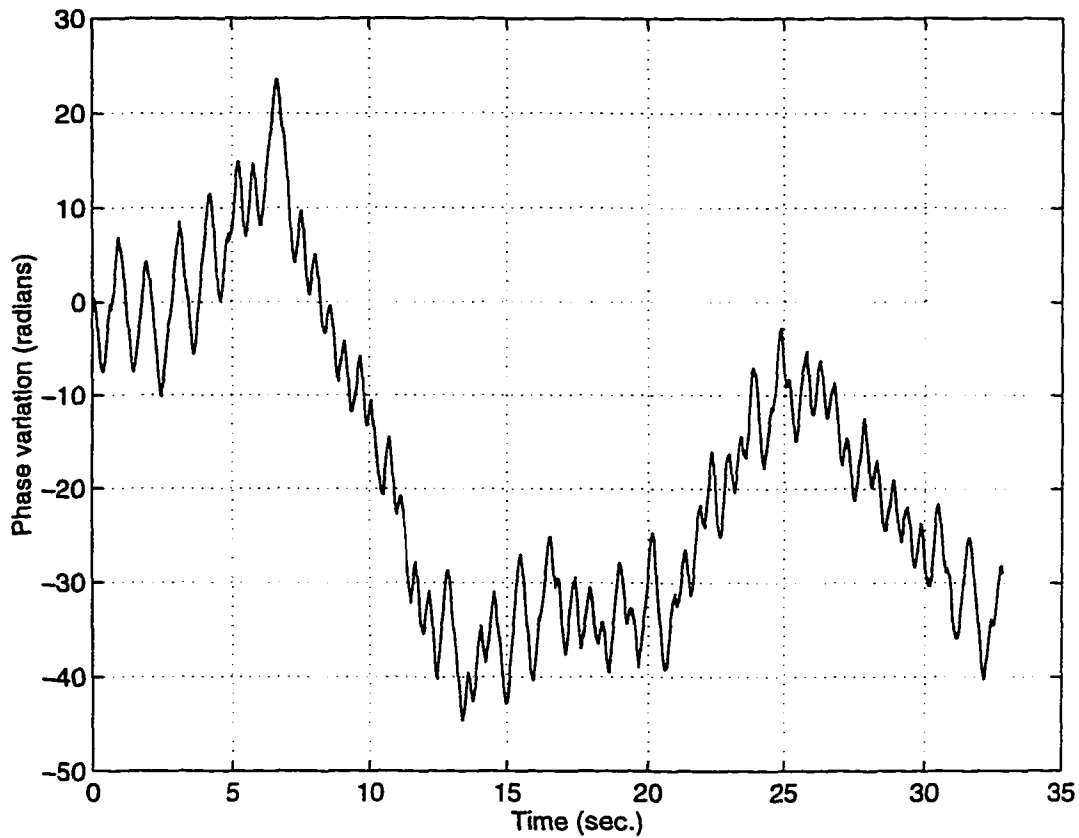


Fig. 9 Phase variation of 60 Hz noise component observed at location IPM3C07 in Hall C line

2.3 Fast Feedback Systems

The beam properties requirements specified by the three experimental halls are very stringent. The nominal value of beam emittance (momentum spread) is specified at $20\mu < \sigma_x, \sigma_y < 50 \mu$ in terms of r.m.s. spot size. The phase space area occupied by the beam is called as emittance. For a source of particles with width

w , from each point of which particles are produced within an angle θ , the phase space area (or the emittance) at the source will be $w\theta$. The nominal value of average beam energy ranges between 0.5 to 4.0 GeV with the energy variation in the range 3×10^{-4} to 1×10^{-3} . The relative energy spread specification for σ_E/E ranges between 5×10^{-5} to 5×10^{-4} with a stability requirement of 25% of the nominal value. Table A.1 in Appendix A describes the beam properties requirements specified by the experimental halls in detail. The results obtained from the noise measurement studies described in previous sections indicate that the beam orbit and energy variations are large when compared to the beam quality requirements specified by the experimental halls. Beam instabilities of the observed magnitude are not convenient for propagation through septa in the accelerator which have clearances of a few millimeters. Therefore feedback systems are needed to stabilize the beam orbit and energy.

2.4 Algorithms

The objective of using fast energy and orbit locks is to lock the beam energy and beam orbit at desired locations in the accelerator. Thus, the quantities of interest at the desired lock location are position, angle of trajectory of the beam in X and Y directions, and the energy variation. The variations in these quantities needs to be estimated from BPM measurements in the presence of sensor noise. Corrective action needs to be taken using actuators such as corrector magnets or

RF vernier cavities, in the presence of process noise. A suitable solution is to model the system in state space formalism of digital control theory [37] and to design of an optimal state estimator and an optimal controller. The description of the system in state space formalism is given by

$$x(k+1) = Ax(k) + Bu(k) + w(k) \quad (1)$$

$$y(k) = Cx(k) + Du(k) + v(k) \quad (2)$$

$x(k)$ is the state vector containing the attributes of the system that are dynamically significant. The state vector is $x = [X_0, X'_0, Y_0, Y'_0, \Delta E / E]$. A is the system dynamic matrix which takes system states from time instant k to $k+1$, B is the control input matrix which takes the control inputs to the state vector, $u(k)$ is the vector of control inputs to the system, $w(k)$ is the process noise vector, $y(k)$ is the vector of measurements, C is the measurement matrix which takes the states to measurements, and $v(k)$ is the measurement noise vector.

A Kalman filter is used to estimate the states from the BPM measurements. The measurement update from sample instant k is obtained using

$$\hat{x}(k) = \bar{x}(k) + L(y(k) - C\bar{x}(k) - Du(k)) \quad (3)$$

and the time update that takes the state vector from sample instant k to $k+1$ is obtained using

$$\bar{x}(k+1) = A\hat{x}(k) + Bu(k) \quad (4)$$

Here $\bar{x}(k)$ is the estimated state vector and $\bar{x}(k+1)$ is the predicted state vector for sample $k+1$ obtained from the estimated state at sample k . The controller equations that are used for the feedback loop can be obtained by combining the two equations above, described as

$$\bar{x}(k+1) = A\bar{x}(k) + Bu(k) + AL(y(k) - C\bar{x}(k) - Du(k)) \quad (5)$$

$$u(k) = -Kx(k) \quad (6)$$

L is the state estimator gain matrix and K is the controller gain matrix. Eq. (5) is used to propagate the state vector to sample $k+1$. The first term in Eq. (5) uses the system dynamic matrix A and the state vector at time k to calculate the new state. The second term uses the control input matrix B and incorporates the effect of actuator settings on the state. The third term is the correction term between estimated and actual states obtained from the measurements. Eq. (6) is used to calculate the actuator setting based on the current state estimate using negative state feedback through an optimal gain matrix K .

Matrices A , B , C , D , K and L that are used to compute control input $u(k)$ can be calculated from the analytically obtained model of the relevant subsystem of the accelerator and from an estimate of process and measurement noises. These matrices can also be extracted from input/output data from the feedback loop by applying system identification techniques which will be discussed in detail in next chapter.

2.5 Fast Energy Feedback System

The fast energy feedback system [38] is designed to stabilize the energy of the beam at a particular location in the accelerator against beam variations caused by effects such as phase and gradient fluctuations in the superconducting RF cavities of the accelerating system upstream. The beam energy variations for a particular lock location is determined from position measurements obtained from 5 BPMs in the vicinity of lock location. This set of 5 BPMs has at least one BPM in the dispersive region of the accelerator such that the position variation measured at this BPM can be related to energy variation at the lock location. A minimum set of 3 BPMs is needed to estimate 5 quantities describing the state of the beam at the lock location, namely $[X_o, X'_o, Y_o, Y'_o, \Delta E/E]$, from 6 position measurements (3 X and 3 Y positions). A set of 5 BPMs is used to maintain redundancy in case of failure of BPMs.

The energy correction signal computed by the feedback algorithm is applied using the vernier input available on the analog RF control module hardware. A differential signal applied to the vernier input changes the setpoint of individual analog feedback loop of the RF control system that regulates the amplitude and phase of the RF field in the cavity.

2.5.1 Beam Position Monitors (BPMs)

As mentioned in the previous sections there are two kinds of Beam position monitors currently being used in the accelerator. The mechanical hardware for both of these systems is essentially the same but the electronic hardware that processes the analog signal to produce the beam position is different. The mechanical hardware for a BPM consists of four thin-wire quarter wave pickup antennas which are symmetrically placed at the corners of a square in a plane perpendicular to the beam axis and centered on the beam axis. The pickup antennas are parallel to the direction of beam motion. Considering there are no errors in the system, and that the X+ and X- as well as Y+ and Y- signals are proportional to the amplitude of beam generated signal on the on each wire, then the beam position can be calculated using the expression

$$X = k \frac{X_{+-} - X_{-}}{X_{++} + X_{-}} \text{ and } Y = k \frac{Y_{+-} - Y_{-}}{Y_{++} + Y_{-}} \quad (7)$$

The sensitivity of each BPM, k , is measured and its nominal value lies within 1% of 18.5 mm.

The different electronics and the scheme of computation for Switched Electrode Electronics (SEE) and 4-channel electronics BPMs is described now.

2.5.1.1 4-channel Electronics BPMs

The electronics portion of this type of BPM is composed of a heterodyne preamplifier located in the accelerator tunnel and the synchronous amplitude detector located upstairs in the service buildings. The fundamental frequency of the beam is picked up by the four antennas and this signal is transmitted to the front end preamplifier known as B0005 electronics. The preamplifier amplifies these signals and then they are downconverted from 1.5 GHz to 1 MHz in the B005 electronics chassis. The downconverted signals are sent upstairs to B0007 electronics, resident in a CAMAC crate, and which consists of programmable gain amplifier, synchronous detector, and 12 bit ADC. The gain of the programmable gain amplifier is adjustable over a 30 dB range to be set according to the expected value of the beam current in the accelerator.

Eq. (7), which describes the calculation of beam position, cannot be directly applied for the 4-channel electronics BPMs because of the errors in the system which violate the assumptions for the calculation. There are two kinds of errors in the system. First, the amplitude gain in different channels, namely X^+ , X^- , Y^+ , Y^- , might be different. Second, there are offsets that exist for each of the four channels. Therefore the computation has to be modified as

$$X = k \frac{(X^+ - X_{off}^+) - \alpha_x(X^- - X_{off}^-)}{(X^+ - X_{off}^+) + \alpha_x(X^- - X_{off}^-)} \quad (8)$$

and similarly for Y position. The offsets and α_x , α_y are measured by the automatic calibration circuit. This computation of beam position still needs to be rotated by 45° to extract the beam position in physical coordinates used by accelerator physicists.

2.5.1.2 Switched Electrode Electronics (SEE) BPMs

There are several advantages that SEE BPMs provide over 4-channel electronics. The 4 channel system does not have sufficient dynamic range to operate outside the beam current range of 10A to 100 A. The 4-channel electronics have different drifting gains for X+, X- pair and Y+ and Y- pair of antennas. The 4-channel system does not have capability to detect multiple passes of beam through the same beam line in the linacs. The SEE BPMs were designed to overcome these difficulties.

SEE BPM electronics, for each channel, consists of a BPM detector, RF module located in the tunnel, IF module, timing module, data acquisition board resident in a VME crate. The timing module is used to synchronize the BPM system with the accelerator timing. The RF module accepts four inputs (X+, X-, Y+, Y-). The RF module switches between the plus and minus channels, amplifies the signal by 23 dB and downconverts the RF signal to 45 MHz before transmitting the signal to the IF module. The IF module amplifies the signal sent by the RF module and downconverts it to baseband signal such that it is ready to

be digitized by high speed commercial (VMIC 3115) 12 bit, data acquisition module. The beam position data is acquired at 248 kHz rate by the VMIC 3115 data acquisition module and is processed by the Motorola MV167 single board VME computer running a data processing routine in real-time, using VxWorks operating system [39]. This routine also regulates the digital gain for the linear operation of the video detectors in the IF module. The highest rate of processed beam position updates from the data acquisition and processing routine is currently limited to 60 Hz.

2.6 RF Control System

The RF control system [40] has to regulate the phase and gradient of the RF accelerating field in the cavities to a high degree of accuracy in order to achieve the stringent beam quality requirements. Microphonic noises in the form of mechanical vibrations modulate the resonance frequency of the cavity, and cause the phase of the accelerating field to fluctuate by as much as 20° and the gradient to fluctuate up to 5%. These fluctuations have to be suppressed by the RF control system by a factor of 100 for phase variations and by a factor of 1000 for gradient variations. The CEBAF design of the accelerating system uses a separate control system for individual cavity. The RF control system uses a heterodyne scheme to convert the cavity frequency of 1497 MHz to 70 MHz.

The major components of the RF control system include a high power amplifier (HPA), the power transmission system, the cryostat with superconducting cavity, and the low level RF control module. The HPA houses 8 klystrons, a common cathode power supply, and separate power supplies for the filaments and modulating anodes. Each klystron can deliver up to 5 kW of CW RF power to an individual cavity. The RF power from the klystron is provided to the cavity through a transmission system which consists of a waveguide (WR-650) with a circulator and directional couplers on the klystron side, and a higher-order-mode filter on the cavity side.

The RF control module can be further classified into five components based on functionality: RF converter board, IF board, Analog Board, I/O board, and CPU board. The RF converter board transforms the 1497 MHz cavity field probe signal to a 70 MHz IF signal. The IF board contains a phase detector and controller for gradient and phase. The Analog Board provides gain stages for gradient and phase control. The I/O board provides 32 digital inputs, 32 digital outputs, 20 analog outputs, and 40 multiplexed analog inputs. The CPU board provides local intelligence and communicates with the hardware via the I/O board and the control computer. The CPU board has an Intel 80186 microprocessor which runs embedded software for data acquisition, signal calibration, and interlock functions.

2.7 Fast Orbit Feedback System

A fast orbit feedback system is designed to stabilize the orbit of the beam at a particular location in the accelerator against beam variations caused by various effects. The beam orbit variations for a particular lock location is determined from position measurements obtained from 5 BPMs in the vicinity of lock location.

The correction signal computed by the feedback algorithm is applied using the VME DAC that sends this signal to a modified trim (power supply). The trim card maintains a desired level of current flowing to the fast air core magnets in order to produce appropriate correction field against beam orbit variations.

2.7.1 Air Core Correctors

The air core corrector magnets [41] were designed to be able to suppress the line power harmonics disturbances at location in the East Arc. Based on noise measurement data presented in section 2.2, an estimate is made of the necessary field to provide the angular kick to be applied to these air core correctors without exceeding a ± 5 mm offset from the reference beam orbit. The determined angular kick requirement can be translated into integrated field of approximately 750 Gauss-cm for a beam energy of 445 MeV. The existing iron core magnets in the accelerator cannot be used to suppress line power harmonics disturbances. A test performed [41] on an iron core magnet indicated that when a 60 Hz, software-

generated signal was sent to the magnet, the field produced was highly distorted. This resulted in a decision to construct air core correctors that had a sufficient frequency response and field strength to correct for fast disturbances.

The six inch long fast air core correctors are designed to be mountable on a three inch beam pipe. The integrated field produced by these magnets is 970 Gauss-cm at 5 Amp current. Modified CEBAF trim cards are used for power supply to the fast air core correctors. The existing trim cards, initially tuned for slow response iron core correctors, were modified by changing a few resistive and inductive elements on the board in order to provide faster response. These cards accept a ± 3 Volt signal, through the backplane connector to control the ± 10 Amp output current going to correctors.

2.8 Performance of Feedback Systems

The fast energy and orbit feedback systems [38] described in section 2.5 and 2.7 have been successfully tested on the accelerator. The fast energy feedback system has been used to stabilize the beam energy in the injector for over a week without any problem. The fast orbit and energy feedback system in the East Arc have also been run for several days without interruptions.

The performance of feedback control system is generally expressed in terms of criteria such as stability, accuracy, transient response, residual noise, RMS error

criterion [42] etc. The exact specifications are usually dictated by the required system performance for individual system.

The transient response characteristics of fast energy feedback system in the injector can be observed in Fig. 10. This figure shows a screen capture image of the graphical user interface for the injector fast energy feedback system. The data for states X , X' , and $\Delta E/E$ acquired at 60 Hz rate is shown. The feedback loop was running in “compute only” mode up to sample instant 95. In this mode of operation the states and the feedback correction signal are computed but the correction is not applied to the system. It can be seen in Fig. 10 that while in “compute only” mode there was a DC error of approximately 0.005 in $\Delta E/E$. This DC error was corrected within 4 samples (66.67 msec.) after the correction signal was applied. Fast energy and orbit feedback systems in the East Arc display similar transient response behavior.

Fast Energy Lock: Instamatic Display of States 1-4

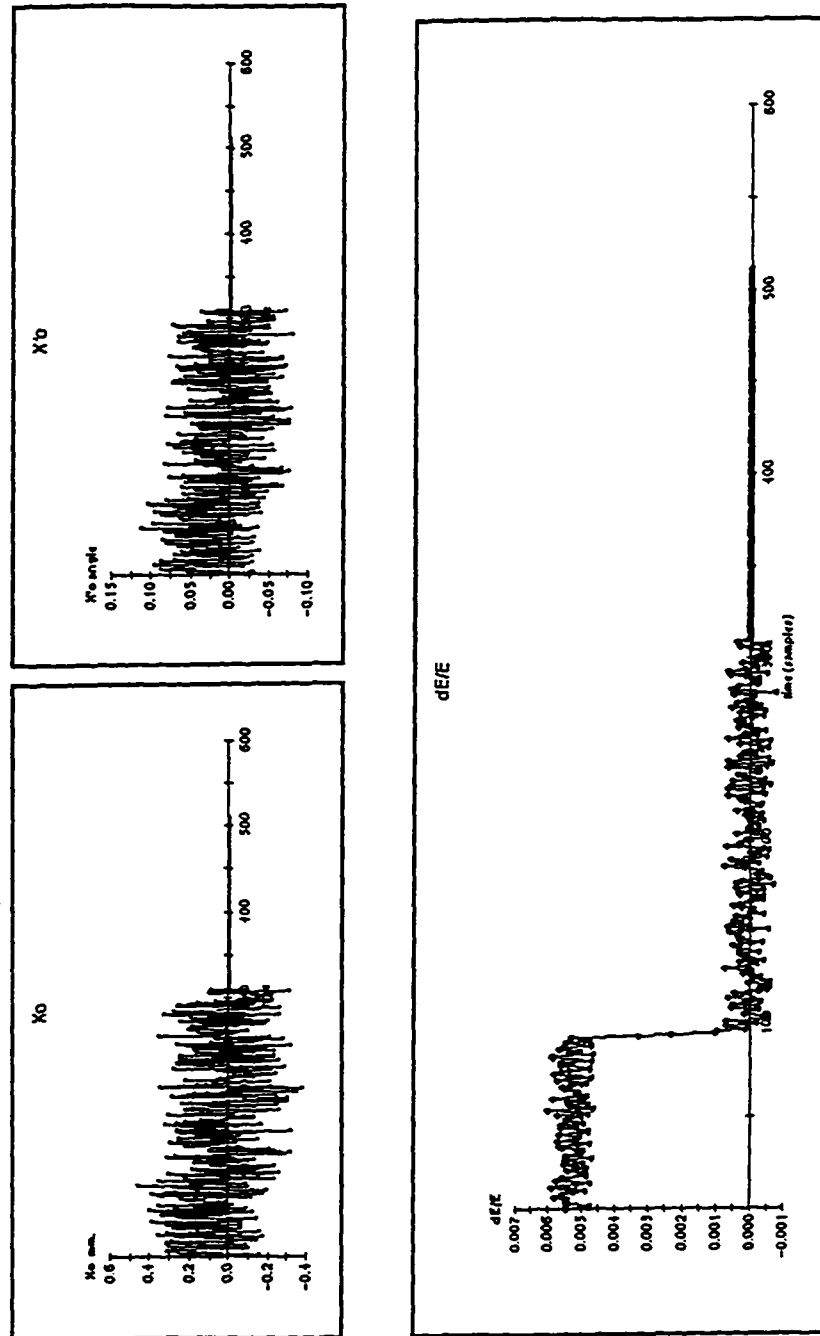


Fig. 10 Screen Capture image of GUI for Fast Energy Feedback System

RMS error criterion for the Injector fast energy feedback system was studied. This was accomplished by adding a disturbance on beam energy and running the feedback system to suppress the disturbance. Simultaneously the BPM position data was collected for analysis. The disturbance that was added to beam energy can be written as $A \sin(2\pi\omega t) + d(t)$. In this expression $d(t)$ is a uniformly distributed random noise component with an amplitude which was 6% of the amplitude of the deterministic component. Fig. 11 shows the state $\Delta E/E$ for a disturbance frequency, ω , of 6 Hz. Feedback correction was applied between samples 570 and 1300 approximately. The RMS error before correction 3.1×10^{-3} and it was reduced to 3.237×10^{-4} after application of feedback correction signal.

Fig 12 shows the power spectra of $\Delta E/E$ signal before and after application of correction signal. It can be seen from this figure that the deterministic noise component (6 Hz) has been reduced by over two orders of magnitude. The noise floor in the power spectrum for corrected signal is also lower than the uncorrected signal.

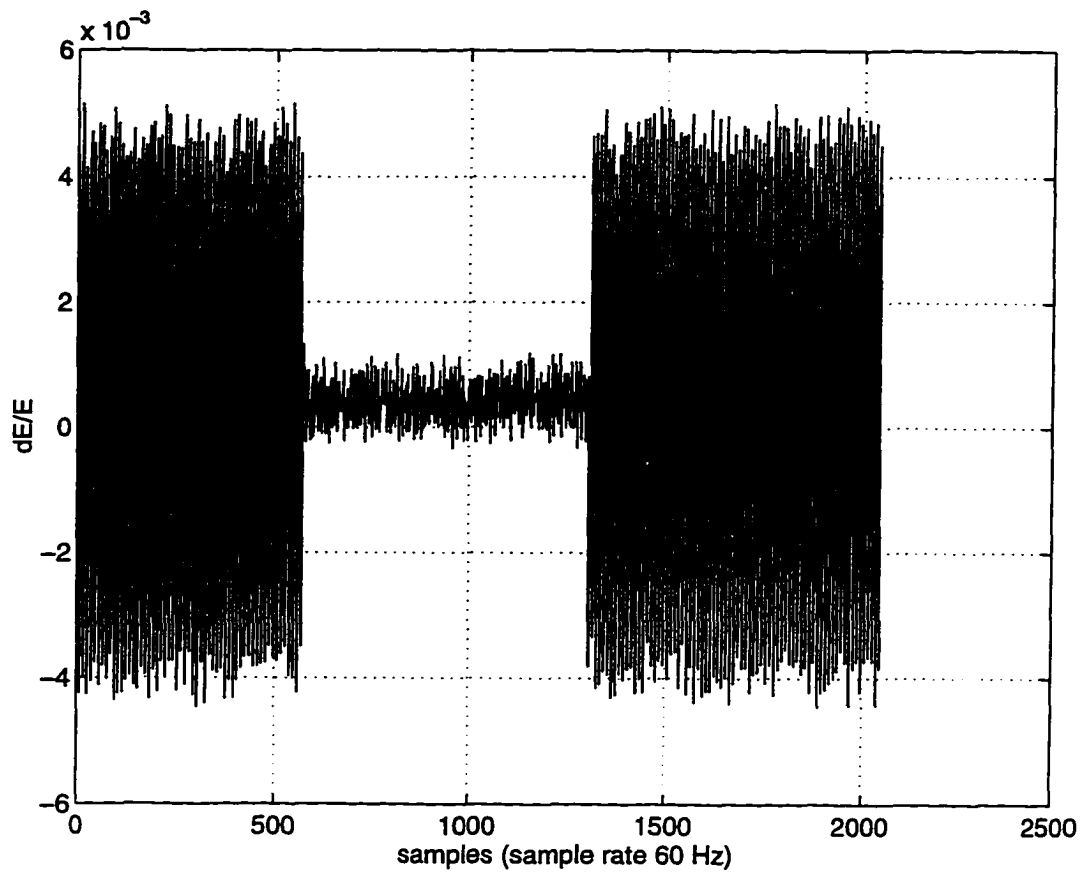


Fig. 11 $\Delta E/E$ with and without feedback correction (sample rate 60 Hz). When feedback correction is applied, the peak to peak variation in $\Delta E/E$ is reduced from $\pm 4 \times 10^{-3}$ to less than $\pm 1 \times 10^{-3}$

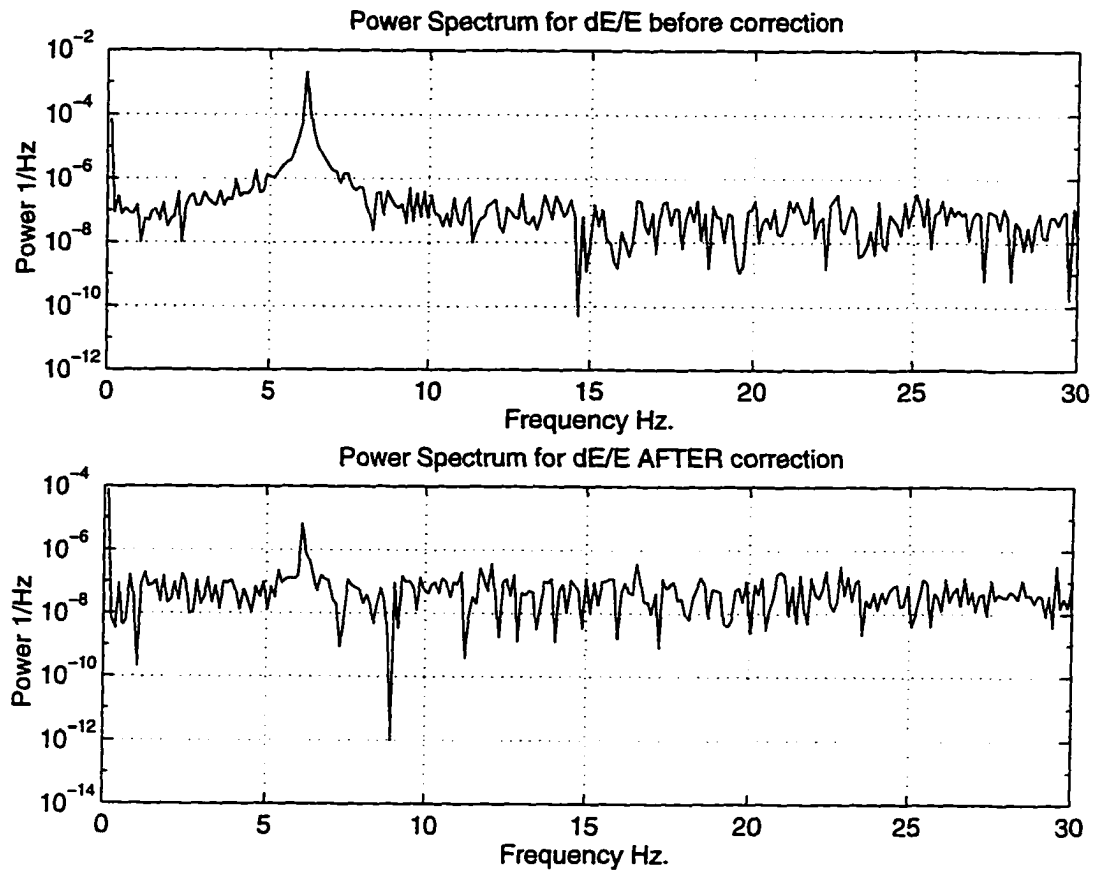


Fig. 12 Power spectra of $\Delta E/E$ before and after correction signal was applied. The power in disturbance component at 6 Hz is reduced by over two orders of magnitude by application of feedback correction.

CHAPTER III

APPLICATION OF ON-LINE SYSTEM IDENTIFICATION

3.1 Background Theory

The selection of mathematical formulation is an important decision in developing recursive system identification algorithms. The state space formalism, briefly described in section 2.4, is widely used and most convenient form for expressing the mathematical model for control systems engineering applications.

$$x(k+1) = Ax(k) + Bu(k) \quad (9)$$

$$y(k) = Cx(k) + Du(k) \quad (10)$$

Eq. (9) and (10) represent a discrete time state space model of a dynamic system. This form is useful for implementation of control algorithm on a digital computer because the input/output data for the physical system is discretized.

Starting with zero initial conditions ($x(0) = 0$) and solving for the outputs $y(k)$ in terms of previous inputs produces the following

$$x(0) = 0,$$

$$y(0) = Du(0)$$

$$x(1) = Bu(0)$$

$$y(1) = CBu(0) + Du(1)$$

$$x(2) = ABu(0) + Bu(1)$$

$$y(2) = CABu(0) + CBu(1) + Du(2)$$

and generalized summation series can be written for the expression of states and measurements at sample instant k as

$$x(k) = \sum_{i=1}^k A^{i-1} Bu(k-i) \quad (11)$$

$$y(k) = \sum_{i=1}^k CA^{i-1} Bu(k-i) + Du(k) \quad (12)$$

In the expression of $x(k)$ and $y(k)$ above if a unit pulse is applied as input at time instant $k=0$ such that $u_i(0) = 1$ for $i=1, 2, \dots, r$ and $u_i(k) = 0$ for $k=1, 2, \dots$ then the results can be assembled in a matrix form as

$$\theta_0 = D, \theta_1 = CB, \theta_2 = CAB, \theta_3 = CA^2 B, \dots, \theta_k = CA^{k-1} B$$

These matrices are known as system Markov parameters. Markov parameters are unique for a given system. The system matrices A , B , C , and D are contained in the system Markov parameters, therefore they can be used for system

identification. The system outputs at various instants $k = 0, 1, 2, \dots$ can be written in terms of Markov parameters and inputs as

$$y(k) = \sum_{i=0}^k Y_i u(k-i) \quad (13)$$

The state vector of system which relates to physical quantities is generally not accessible for direct measurement. If the system is observable then it is possible to use an observer (or state estimator) to estimate the state variables from input/output data. In some circumstances state variable estimates are preferable to their directly measured values because the error introduced by the sensors may be larger than the error in estimating these variables.

The equation for a state observer can be written as

$$\hat{x}(k+1) = A\hat{x}(k) + Bu(k) - L[y(k) - C\hat{x}(k) - Du(k)] \quad (14)$$

$$\hat{y}(k) = C\hat{x}(k) + Du(k) \quad (15)$$

where $\hat{x}(k)$ is the estimated state vector and $\hat{y}(k)$ is the estimated output.

The above equations can be rewritten as

$$\hat{x}(k+1) = (A + LC)\hat{x}(k) + (B + LD)u(k) - Ly(k)$$

$$y(k) = Cx(k) + Du(k) + v(k)$$

Now defining $\bar{A} = A + LC$, $\bar{B} = [B + LD \quad -L]$ and $\psi(k) = \begin{bmatrix} u(k) \\ y(k) \end{bmatrix}$ the

above equation can be rewritten as

$$\hat{x}(k+1) = \bar{A}\hat{x}(k) + \bar{B}\psi(k) \quad (16)$$

$$y(k) = Cx(k) + Du(k) \quad (17)$$

The pulse response characteristics of this observer system can be obtained using the same scheme as described above to obtain the system Markov parameters. Starting with zero initial conditions and solving for the estimated output, generalized summation series can be written for the expression of estimated states and estimated measurements at instant k as

$$\hat{x}(k) = \sum_{i=1}^k (A + LC)^{i-1} (B + LD)u(k-i) - \sum_{i=1}^k (A + LC)^{i-1} Ly(k-i) \quad (18)$$

$$\begin{aligned} \hat{y}(k) = & \sum_{i=1}^k C(A + LC)^{i-1} (B + LD)u(k-i) + Du(k) \\ & - \sum_{i=1}^k C(A + LC)^{i-1} Ly(k-i) \sum_{i=1}^k \end{aligned} \quad (19)$$

If a unit pulse is applied as input at instant $k=0$ such that $u_i(0) = 1$ for $i=1, 2, \dots, r$ and $u_i(k) = 0$ for $k=1, 2, \dots$ and then the results can be assembled in a matrix form as

$$\bar{\theta}_0 = D, \bar{\theta}_1 = C[B + LD \quad -L], \bar{\theta}_2 = C(A + LC)[B + LD \quad -L], \dots$$

$$\bar{\theta}_k = C(A + LC)^{k-1}[B + LD \quad -L] = [C(A + LC)^{k-1}(B + LD) \quad -C(A + LC)^{k-1}L]$$

These matrices written sequentially are called as observer Markov parameters.

They contain the system matrices A , B , C , D and estimator gain matrix L .

The equation for estimated output can be rewritten as

$$\hat{y}(k) + \sum_{i=1}^k C(A + LC)^{i-1} Ly(k-i) = \sum_{i=1}^k C(A + LC)^{i-1} (B + LD)u(k-i) + Du(k)$$

For system with an appropriate design of observer the difference between the estimated output and measured will approach zero after p ($k > p$) samples.

Therefore the above equation for estimated output can be written as

$$y(k) + \sum_{i=1}^p \bar{\theta}_i^{(1)} y(k-i) = \sum_{i=1}^p \bar{\theta}_i^{(2)} u(k-i) + Du(k) \quad (20)$$

where $\bar{\theta}_i^{(1)} = C(A + LC)^{i-1}$ and $\bar{\theta}_i^{(2)} = C(A + LC)^{i-1}(B + LD)$.

This equation is known as ARX model of order p . ARX model is used in developing recursive system identification techniques such as Fast Transversal Filter which is described later in this chapter.

3.2 Classical Least Squares Approach for System Identification

Consider the ARX model as described in Eq. (20) and rewrite it as

$$y(k) = \bar{\theta} \psi_p(k-1)$$

where $\psi_p(k-1)$ contains the input/output data and $\bar{\theta}$ contains the observer

Markov parameters written as

$$\psi_p(k-1) = [u(k) \quad y(k-1) \quad u(k-1) \quad \dots \quad y(k-p) \quad u(k-p)]^T$$

$$\bar{\theta} = [D \quad -\bar{\theta}_1^{(2)} \quad \bar{\theta}_1^{(1)} \quad \dots \quad -\bar{\theta}_p^{(2)} \quad \bar{\theta}_p^{(1)}]$$

Collecting all $y(k)$ for instants 0 through present above equation can be written in a matrix form as

$$Y(k) = \bar{\theta} \Psi_p(k-1) \quad (21)$$

$$\text{where } \Psi_p(k-1) = \begin{bmatrix} u(0) & u(1) & u(2) & \dots & u(p) & \dots & u(k) \\ & \psi(0) & \psi(1) & \dots & \psi(p-1) & \dots & \psi(k-1) \\ & & \psi(0) & \dots & \psi(p-2) & \dots & \psi(k-2) \\ & & & \ddots & \vdots & \dots & \vdots \\ & & & & \psi(0) & \dots & \psi(k-p) \end{bmatrix} \text{ and}$$

$$Y(k) = [y(0) \quad y(1) \quad y(2) \quad \dots \quad y(p) \quad \dots \quad y(k)]$$

$\Psi_p(k-1)$ is input/output data matrix with dimensions $[(m+r)p+r] \times (k+1)$ where m is the number of outputs, r is the number of inputs and p is the order of ARX model. $Y(k)$ contains the measurement data.

The observer Markov parameters, $\bar{\theta}$, can be obtained by applying a least squares solution to Eq (21) as

$$\hat{\theta}_p = Y(k)\Psi_p^T(k-1)[\Psi_p(k-1)\Psi_p^T(k-1)]^{-1} \quad (22)$$

The least squares solution can be obtained once sufficient number of data points, $k > ((r+m)p+r)$ for input/output data have been collected. The least squares solution for observer Markov parameters described using Eq. (22) has two drawbacks. First, the observer Markov parameters can only be obtained once all the data for k time instants has been collected. Second, it requires inverting a large covariance matrix $[\Psi_p(k-1)\Psi_p^T(k-1)]$ every time new data becomes available. Matrix inversion is a computationally intensive task and cannot be used in a solution for on-line implementation.

3.3 Recursive Technique

Recursive techniques can be applied in order to overcome the difficulties of least squares solution described in the previous section.

Eq. (22) can be written for time instant $k+1$ as

$$\hat{\theta}_p(k+1) = Y(k+1)\Psi_p^T(k)[\Psi_p(k)\Psi_p^T(k)]^{-1} \quad (23)$$

such that $Y(k+1) = [Y(k) \quad y(k+1)]$ and $\Psi_p(k) = [\Psi_p(k-1) \quad \psi_p(k)]$.

Now define the inverse of the input/output data covariance matrix as

$$P_p(k) = [\Psi_p(k)\Psi_p^T(k)]^{-1}$$

Further expanding the terms on the right hand side in this definition

$$\begin{aligned} P_p(k) &= \left[\begin{array}{cc} [\Psi_p(k-1) & \psi_p(k)] \begin{bmatrix} \Psi_p^T(k-1) \\ \psi_p^T(k) \end{bmatrix} \end{array} \right]^{-1} \\ &= \Psi_p(k-1)\Psi_p^T(k-1) + \psi_p(k)\psi_p^T(k) \end{aligned} \quad (24)$$

$$= [P_p^{-1}(k-1) + \psi_p(k)\psi_p^T(k)]^{-1} \quad (25)$$

Eq. (25) can be simplified by application of matrix inversion lemma (Sherman-Morrison formula) [43] as

$$P_p(k) = P_p(k-1) \left[I - \frac{\psi_p(k)\psi_p^T(k)P_p(k-1)}{(1 + \psi_p^T(k)P_p(k-1)\psi_p(k))} \right] \quad (26)$$

The Eq. (26) has been now been transformed in form useful for recursively updating the inverse of covariance matrix, $P_p(k)$, without having to perform a large matrix inversion.

$$\text{Define } G_p(k) = \frac{\psi_p^T(k)P_p(k-1)}{1 + \psi_p^T(k)P_p(k-1)\psi_p(k)}$$

Notice that the denominator in above equation is a scalar. Utilizing the recursive formulation for $P_p(k)$ and Eq. (23) an expression for updating the observer Markov parameters can be written as

$$\begin{aligned}\hat{\theta}_p(k+1) &= Y(k+1)\Psi_p^T(k)P_p(k) \\ &= [Y(k)\Psi_p^T(k-1) + y(k+1)\psi_p^T(k)]P_p(k-1)[I - \psi_p(k)G_p(k)]\end{aligned}$$

Now applying the expression $\hat{\theta}_p(k) = Y(k)\Psi_p^T(k-1)P_p(k-1)$ to above equation

$$\begin{aligned}\hat{\theta}_p(k+1) &= [\hat{\theta}_p(k) - \hat{\theta}_p(k)\psi_p(k)G_p(k) + y(k+1)\psi_p^T(k)P_p(k-1) \\ &\quad - y(k+1)\psi_p^T(k)P_p(k-1)\psi_p(k)G_p(k)]\end{aligned}$$

simplifying the third and fourth terms in the above expression

$$\hat{\theta}_p(k+1) = \hat{\theta}_p(k) + [y(k+1) - \hat{\theta}_p(k)\psi_p(k)]G_p(k) \quad (27)$$

The above equation presents an expression for obtaining the observer Markov parameters recursively. This expression can be simplified by defining the following

$$\hat{y}(k+1) = \hat{\theta}_p(k)\psi_p(k) \quad (28)$$

$$e(k+1) = y(k+1) - \hat{y}(k+1) \quad (29)$$

Finally, the recursive updating of the observer Markov parameters can be written as

$$\hat{\theta}_p(k+1) = \hat{\theta}_p(k) + e(k+1)G_p(k) \quad (30)$$

Formulas presented in Eqs. (26), (28), (29) and (30) constitute the Recursive Least Squares (RLS) algorithm [44]-[45].

In the above definitions $\hat{y}(k+1)$ is the predicted output at next time instant and $e(k+1)$ is the difference between estimated output and measured output at time instant $k+1$. $G_p(k)$ can be considered as the gain vector which determines how the output estimation error affects the update of observer Markov parameters $\hat{\theta}_p(k+1)$. The estimated output computed by $\hat{y}(k+1) = \hat{\theta}_p(k)\psi_p(k)$ can be called as *a priori* estimate since the $\hat{Y}_p(k)$ uses measurement at time instant k and not $k+1$. The expressions for *a priori* and *a posteriori* output estimates and errors can be written as

$$\hat{y}^-(k+1) = \hat{\theta}_p(k)\psi_p(k) \quad - \text{a priori output estimate}$$

$$\hat{y}^+(k+1) = \hat{\theta}_p(k+1)\psi_p(k) \quad - \text{a posteriori output estimate}$$

$$e^-(k+1) = y(k+1) - \hat{y}^-(k+1) \quad - \text{a priori estimation error}$$

$e^+(k+1) = y(k+1) - \hat{y}^+(k+1)$ - *a posteriori* estimation error

Based on these definitions the equation for parameter update can be written as

$$\hat{\theta}_p(k+1) = \hat{\theta}_p(k) + e^-(k+1)G_p(k) \quad (31)$$

Substituting the expression for *a posteriori* output estimate into *a posteriori* estimation

$$e^+(k+1) = y(k+1) - \hat{\theta}_p(k+1)\psi_p(k)$$

which upon simplification results in

$$e^+(k+1) = \frac{e^-(k+1)}{1 + \psi_p^T(k)P_p(k-1)\psi_p(k)}$$

This equation shows a recursive procedure of updating the *a posteriori* estimation error using the *a priori* estimation error.

As indicated earlier $Y(k)$ contains the various output measurements such that $Y(k) = [y(0) \ y(1) \ y(2) \ \dots \ y(p) \ \dots \ y(k)]$. The output estimation error corresponding to difference between $y(k)$ and $\hat{y}(k)$ has an analog known as equation error which is the difference between $Y(k)$ and predicted values as described by

$$E_p(k+1) = Y(k+1) - \hat{\theta}_p(k+1)\Psi_p(k)$$

Utilizing the definitions of *a priori* and *a posteriori* estimation errors, $Y(k)$ and $\Psi_p(k)$ the above equation can be rewritten as

$$E_p(k+1) = [E_p(k) - e_p^-(k+1)G_p(k+1)\Psi_p(k+1) \quad e_p^+(k+1)]$$

Now defining equation estimation square as

$$\check{E}_p(k+1) = E_p(k+1)E_p^T(k+1)$$

This equation can be simplified in terms of output estimation errors as

$$\check{E}_p(k+1) = E_p(k) + e_p^+(k+1)[e_p^-(k+1)]^T$$

The above equation shows how the equation estimation error squares can be recursively obtained from *a priori* and *a posteriori* errors.

3.4 Fast Transversal Filter (FTF)

The recursive least square algorithm described in the previous section can very well be implemented to perform on-line system identification, but, the number of mathematical operations that need to be performed for this algorithm place a constraint for a system with a large system order p and large number of inputs and outputs. The number of mathematical operations for recursive least squares

algorithm are proportional to $[(r + m)^2 p^2]$, therefore increase quadratically with the increase in system order p .

Computationally efficient versions of RLS known as Fast Transversal Filter (FTF) and Least Squares Lattice (LSL) filter were first presented by Cioffi and Kailath [46] and Lee and Morf [47]. Other formulations of FTF have been presented in references [44] and [45]. The implementation of on-line system identification algorithm developed and presented in this work is based on the formulation of FTF presented in reference [44].

Fast Transversal Filter (FTF) reduces the number of mathematical operations by utilizing the shifting property of input/output data that becomes available serially. The number of mathematical operations for FTF are proportional to $[(r + m)^3]$. The system order does not have any effect on the number of mathematical operations for FTF.

The functionality of FTF can be broken up into three parts: 1) Initialization of variables, 2) Forward time estimation, 3) Backward time estimation. Forward time estimation and backward time estimation share a common data matrix and update each other recursively. The computational steps of FTF are described in this section. Variables with \rightarrow correspond to the forward time estimation and those with \leftarrow correspond to backward time estimation.

Initialization :

The initialization begins when sufficient amount of input/output data has been collected such that $k > (r + m)p + r$. The vector $\psi_p(k-1)$ and input/output data matrix $\Psi_p(k-2)$ are formed as shown below in steps 1 and 2

$$1) \psi_p^T(k-1) = [\psi^T(k) \quad \psi^T(k-1) \quad \psi^T(k-2) \quad \dots \quad \psi^T(k-p)]$$

$$2) \Psi_p(k-2) = \begin{bmatrix} u(0) & u(1) & u(2) & \dots & u(p) & \dots & u(k-1) \\ & \psi(0) & \psi(1) & \dots & \psi(p-1) & \dots & \psi(k-2) \\ & & \psi(0) & \dots & \psi(p-2) & \dots & \psi(k-3) \\ & & & \ddots & \vdots & \dots & \ddots \\ & & & & \psi(0) & \dots & \psi(k-p-1) \end{bmatrix}$$

$$3) \Psi_p(k-1) = [\Psi_p(k-2) \quad \psi_p(k-1)]$$

Matrix $\Psi_{p+1}(k)$ can be partitioned in two different ways for use in forward prediction and backward prediction calculations. Matrices $Y_u(k-1)$ and $Y_u(-1)$ relevant to forward prediction calculation are obtained as shown in steps 4 and 5 below

$$4) Y_u(k-1) = \begin{bmatrix} u(1) & u(2) & u(3) & \dots & u(p+1) & \dots & u(k) \\ y(0) & y(1) & y(2) & \dots & y(p) & \dots & y(k-1) \end{bmatrix}$$

$$5) Y_u(-1) = \begin{bmatrix} u(0) \\ 0 \end{bmatrix}$$

$Y_u(k-1-p)$ used in backward prediction calculation is obtained as

$$6) Y_u(k-1-p) = \begin{bmatrix} 0 & 0 & 0 & \cdots & u(0) & \cdots & u(k-1-p) \\ 0 & 0 & 0 & \cdots & y(0) & \cdots & y(k-1-p) \end{bmatrix}$$

Initialization of Forward Time Estimation parameters is performed as follows:

$$7) P_p(k-2) = [\Psi_p(k-2)\Psi_p^T(k-2)]^{-1}$$

$$8) G_p(k) = \frac{\psi_p^T(k-1)P_p(k-2)}{1 + \psi_p^T(k-1)P_p(k-2)\psi_p(k-1)}$$

$$9) \hat{\theta}_p(k-1) = Y_u(k-1)\Psi_p^T(k-2)P_p(k-2)$$

$$10) \bar{E}(k-1) = Y_u(-1)Y_u^T(-1) + [Y_u(k-1) - \hat{\theta}_p(k-1)\Psi_p^T(k-2)] \\ \times [Y_u(k-1) - \hat{\theta}_p(k-1)\Psi_p^T(k-2)]^T$$

$$11) \gamma_p(k-1) = 1 - G_p(k-1)\psi_p(k-1)$$

Initialization of Backward-Time Estimation parameters is performed as :

$$12) P_p(k-1) = P_p(k-2)[I - \psi_p(k-1)G_p(k-1)]$$

$$13) \hat{\theta}_p(k-1) = Y_u(k-1-p)\Psi_p^T(k-1)P_p(k-1)$$

$$14) \bar{E}_p(k-1) = [Y_u(k-1-p) - \hat{\theta}_p(k-1)\Psi_p^T(k-1)] \times \\ [Y_u(k-1-p) - \hat{\theta}_p(k-1)\Psi_p^T(k-1)]^T$$

The recursion procedure begins with forward-time prediction with formation of vectors $\psi_p(k-1)$ and $Y_u(k)$ using last measured output $y(k)$ and input $u(k+1)$.

$$1) \psi_p(k-1) = \begin{bmatrix} u(k) \\ y(k-1) \\ u(k-1) \\ \vdots \\ y(k-p) \\ u(k-p) \end{bmatrix}$$

$$2) Y_u(k) = \begin{bmatrix} u(k+1) \\ y(k) \end{bmatrix}$$

$$3) e_p^-(k) = Y_u(k) - \hat{\theta}_p(k-1)\psi_p(k-1)$$

$$4) e_p^+(k) = \gamma_p(k-1)e_p^-(k)$$

$$5) \hat{\theta}_p(k) = \hat{\theta}_p(k-1) + e_p^+(k)G_p(k-1)$$

$$6) \bar{E}_p(k) = \bar{E}_p(k-1) + e_p^+(k)[e_p^+(k)]^T$$

$$7) \quad G_{p+1}(k) = [e_p^+(k)^T \bar{E}_p^{-1}(k) \quad G_p(k-1) - e_p^+(k)^T \bar{E}_p^{-1}(k) \hat{\theta}_p(k)]$$

$$8) \quad \gamma_{p+1}(k) = \gamma_p(k-1) - \bar{e}_p^+(k)^T \bar{E}_p^{-1}(k) \bar{e}_p^+(k)$$

The gain vector (shown in step 7) that is used by each of the forward and backward time estimations is partitioned $G_{p+1}(k) = [G_{p+1}^{(r)}(k) \quad G_{p+1}^{(a)}(k)]$ where

$$G_{p+1}^{(a)}(k) = \bar{e}_p^+(k)^T \bar{E}_p^{-1}(k).$$

The backward-time update recursion steps are as follows:

$$1) \quad \psi_p(k) = \begin{bmatrix} u(k+1) \\ y(k) \\ u(k-2) \\ \vdots \\ y(k-p+1) \\ u(k-p+1) \end{bmatrix}$$

$$2) \quad Y_u(k-p) = \begin{bmatrix} y(k-p) \\ u(k-p) \end{bmatrix}$$

$$3) \quad \bar{e}_p^-(k) = Y_u(k-p) - \hat{\theta}_p(k-1) \psi_p(k)$$

$$4) \quad \bar{e}_p^+(k) = \gamma_p(k-1) \bar{e}_p^-(k)$$

$$5) \quad G_p(k) = \frac{G_{p+1}^{(r)}(k) + G_{p+1}^{(a)} \hat{\theta}_p(k-1)}{1 - G_{p+1}^{(a)}(k) \bar{e}_p^-(k)}$$

$$6) \quad \hat{\theta}_p(k) = \hat{\theta}_p(k-1) + \bar{e}_p^-(k)G_p(k)$$

$$7) \quad \gamma_p(k) = \frac{\gamma_p(k-1)}{1 - G_{p+1}^{(a)}(k)\bar{e}_p^-(k)}$$

$$8) \quad \bar{e}_p^+(k) = \gamma_p(k)\bar{e}_p^-(k)$$

$$9) \quad \bar{E}_p(k) = \bar{E}_p(k-1) + \bar{e}_p^+(k)[\bar{e}_p^-(k)]^T$$

After performing the above the computational steps the forward time recursion again begins for time index k .

3.5 Implementation of FTF

The development of FTF as on-line system identification routine was planned and conducted with the goal of implementing it for control system applications (e.g. fast orbit and energy feedback systems) in the CEBAF control system. The accelerator has a distributed real-time control system [48], known as EPICS [49], which is implemented on more than 50 VMEbus CPUs (Motorola 68040 uP) located in service buildings around the racetrack shaped accelerator. These CPUs are connected via Ethernet. Each CPU runs a multitasking real-time operating system known as VxWorks [39]. Most of the development work for VxWorks target system is done on multi-user UNIX host system which provides tools such as text editors, make, source-code control etc.

Since the response time and data rate through the top level control system, ie. EPICS, is limited due to network constraints and indeterminacy of Unix, FTF routines have been implemented as VxWorks tasks, which can be triggered and monitored by routines of top level control system.

FTF routines are programmed in C and compiled with a C cross-compiler for combination of UNIX as host system and Motorola 68040 on MVME167 VMEbus single board computer running VxWorks as the target system. FTF routines also utilize some of the VxWorks libraries in order to use various VxWorks functions, such as circular (ring) buffers. Various modules of the FTF routines do not need to be linked with VxWorks system libraries or even with each other. The object modules are loaded directly onto the VxWorks system which using the symbol table contained in object modules dynamically resolves external symbol references. The C code for various routines of FTF is shown in Appendix B.

3.6 Catchup Technique

As described earlier, the functionality of FTF can be broken up into three parts : initialization, forward time estimation and backward time estimation. Once sufficient number of data points, ($k > (r + m)p + r$), become available, the initialization routine can be executed.

The initialization routine consumes the largest amount of time on the microprocessor. For a two input/two output system with model order of 5 the initialization routine consumed 19.661 msec. Each iteration of forward-time recursion takes 3.795 msec.

While the initialization is taking place, new input/output data keeps coming in. Before the forward time recursion can begin, a certain number of data points will be lost if recursion begins at the latest data point. In most cases this would not be a matter of concern. However, since the backward time recursion routines are sensitive to a loss of data between the initialization and the actual beginning of the recursive procedure, the performance of the FTF could be improved if a catchup technique were implemented. Such a facility can prevent the discontinuity between the initialization data and the set where recursion begins.

The VxWorks operating system [39] provides a facility of circular buffers (or ring buffers) which has been used to implement the catchup technique. In this scheme all the incoming input/output data goes into circular buffers. The initialization routine begins when sufficient number of data points have been accumulated in the circular buffer. While the initialization is taking place the new data keeps filling the circular buffer. Once the initialization process is complete the forward-time recursion begins by reading the data point in the circular buffer where the initialization routine left off. If the processor time consumption for

forward and backward time recursion is smaller than the sample rate then catchup can be completed within a finite number of time steps. The catchup technique will mostly be useful for the systems with a high levels of noise.

3.7 Performance of FTF routine

A two input, two output system was simulated in MATLAB to study the performance of FTF routine. The system matrices for the system that was simulated are as follows

$$A = \begin{bmatrix} 0 & 1 \\ 1 & -0.5 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 10 \\ 0.1 & 5 \end{bmatrix}, D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The simulated input/output data for this system for various test cases was recorded in text files and was used to test the performance of system identification routine.

The first test case involved identification with noise free data for inputs and outputs. In second case identification was performed with noisy data, where 10% (zero mean, uniformly distributed, random) noise was added to the input and output data.

The error between estimated output and measured output of the system can be used as a criterion to determine how well does the system algorithm perform. Fig.13 shows a comparison between output 1 and estimation error (residual) for

the noise free case. The estimation error is very small. Peak to peak variation is estimation error approximately 10^{-5} . Fig.14 shows a plot of estimation error (residual) for output 1. For $\pm 1V$ amplitude output1 signal, the RMS value of estimation error was 1.248×10^{-6} .

Fig. 15 shows a comparison between output 1 and estimation error for the case where 10% random noise was added to input/output data. For $\pm 1V$ amplitude output 1 signal, the RMS of estimation error for the 10% noise added to input/output data is 0.0392

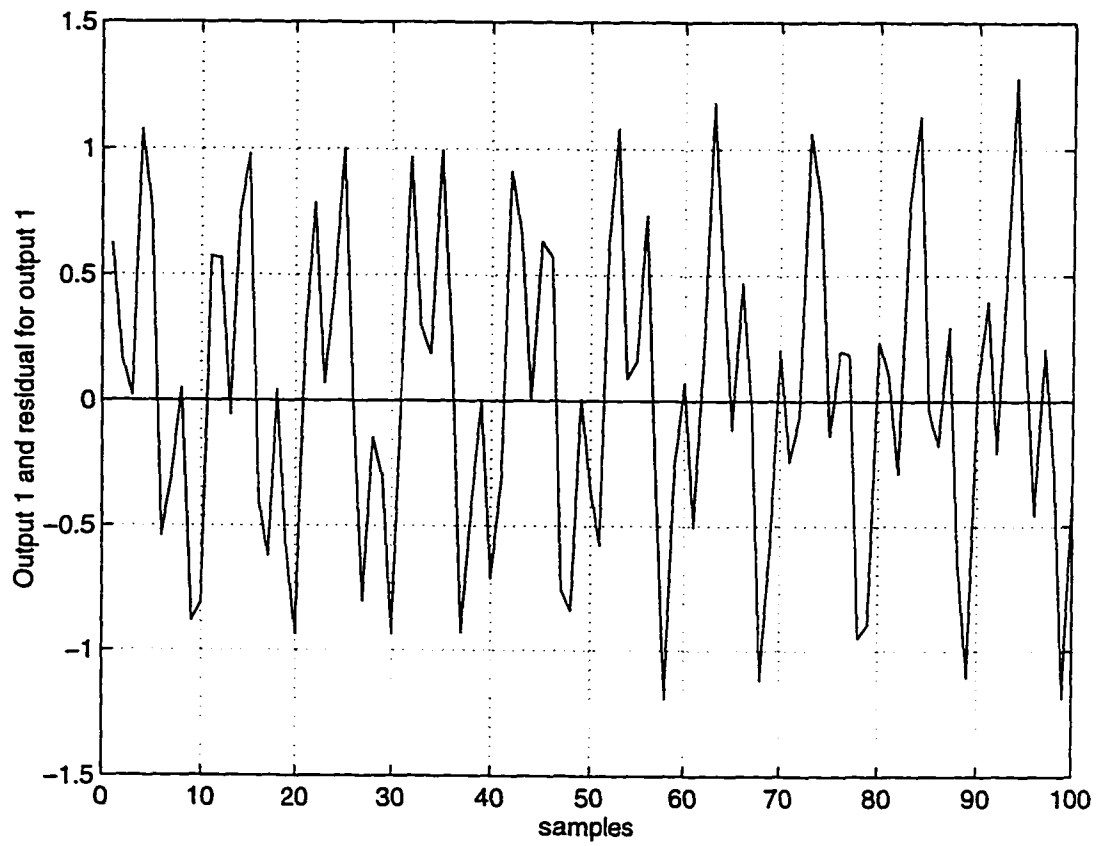


Fig. 13 Comparison between Output 1 and the difference between the estimated and actual output for the noise free case. Peak to peak variation in estimation error is 10^{-5}

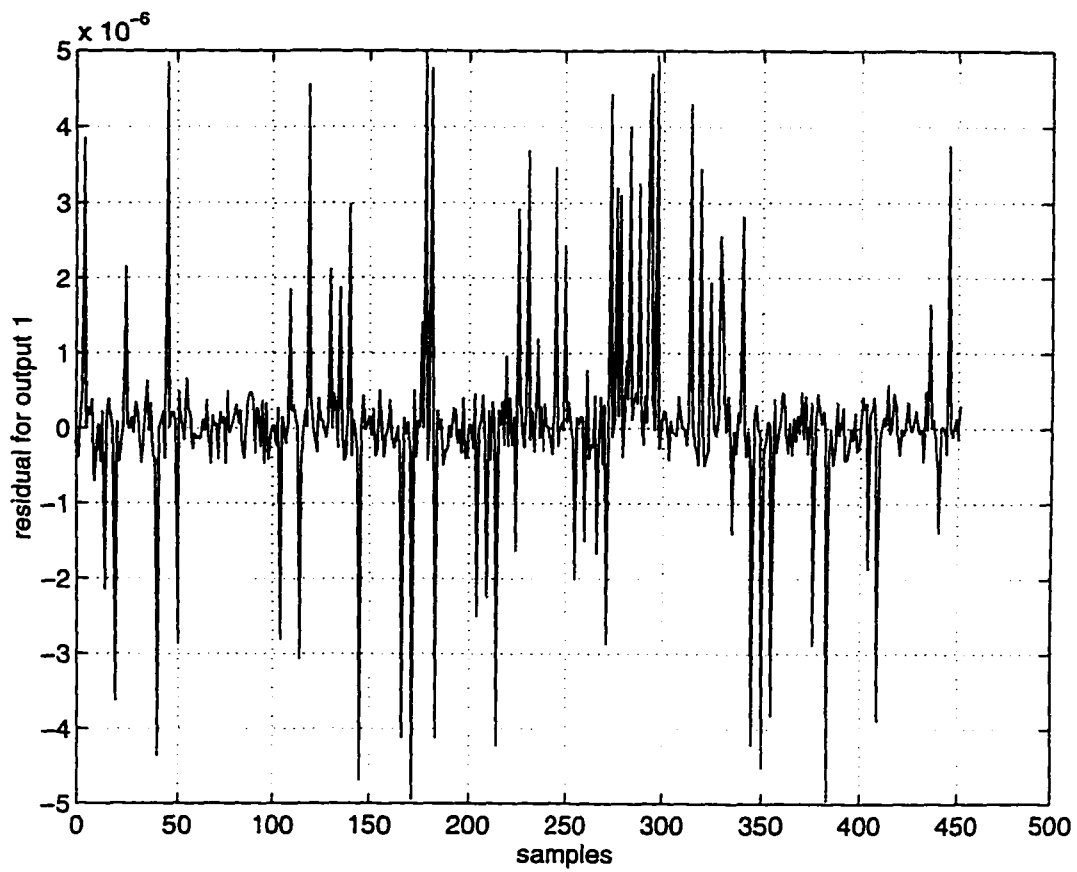


Fig. 14 Estimation error for Output 1 for noise free case. Peak to peak variation in estimation error is 10^{-5} . The RMS value of estimation error is 1.2486×10^{-6}

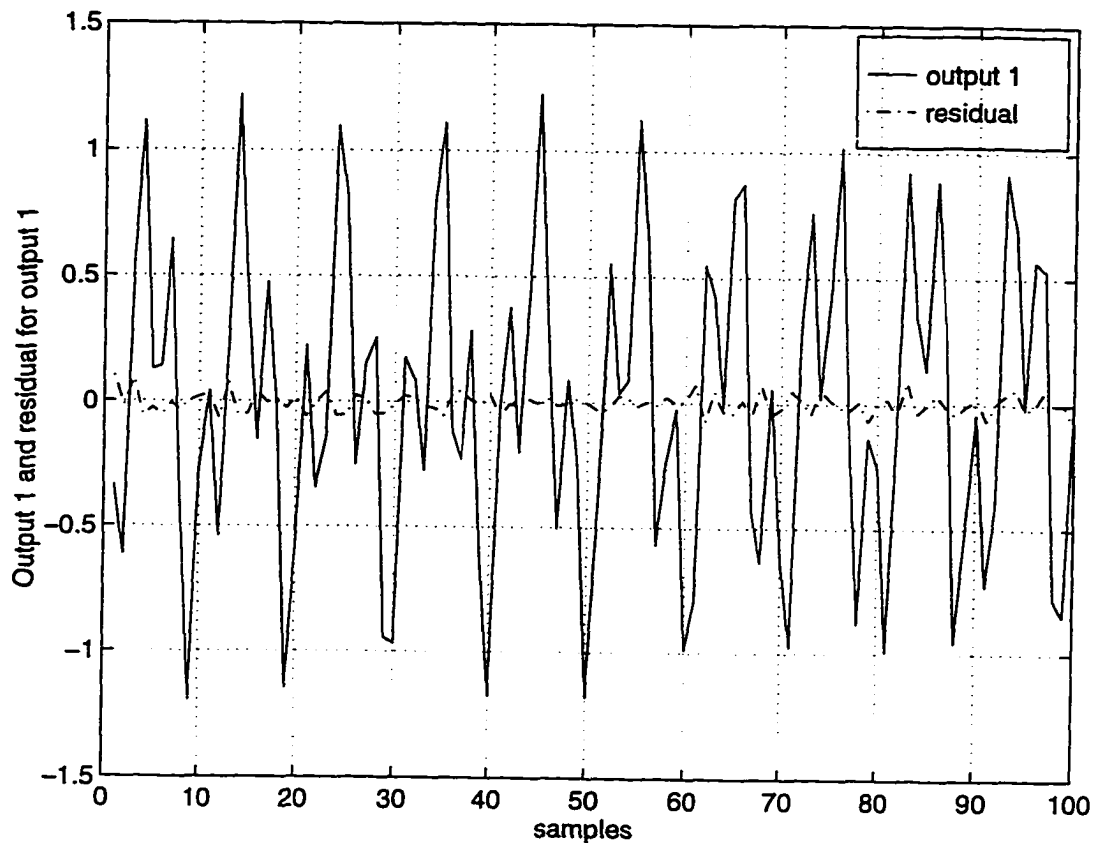


Fig. 15 Comparison between output 1 and estimation error for the case where 10% random noise was added to input/output data. Peak to peak variation in estimation error for output 1 is 0.1901. RMS of estimation error is 0.0392.

Next chapter describes a hardware-in-loop simulation that was performed using an analog computer and a teststand VMEbus CPU to test the functionality of on-line system identification algorithm under conditions of varying plant dynamics and process and measurement noise.

CHAPTER IV

ANALOG COMPUTER SIMULATIONS

4.1 Description of Analog Computer

An analog computer solves a real world problem by rendering the abstract mathematical system into electrical operations whose functions in reality are governed by the same equations as mathematical system. Two problems or systems are said to be analogous if certain or all of their respective measurable quantities obey the same mathematical laws. Analog computers benefit from using active electrical circuits as analogous system because they have no moving parts, they have a high speed of operation, yield good accuracy, and have a high degree of versatility. Active electrical networks consisting of resistors, capacitors, and op-amps connected together are capable of simulating any linear system. The forward voltage transfer characteristics of these networks are analogous to the basic linear mathematical operations encountered in the system's mathematical model. By using diode function generators and special circuits which have nonlinear voltage transfer characteristics, it is also possible to simulate some nonlinear systems on analog computers.

The input and output voltages of the analog computer are analogous to the corresponding mathematical variables of the problem. In some cases because of limitations of the analog computer or its associated input/output hardware, it becomes necessary to change the scale of the analog computer variables. It is important to realize that an analog computer solution is simply a voltage waveform whose time dependence is the same as that of the desired mathematical solution of the system equations the computer represents.

The procedure for simulating a system on an analog computer starts by determining the mathematical model describing the physical system of interest. An analog block diagram of the system is constructed to relate the sequence of mathematical operations and to aid in scaling the analog computer variables, if necessary. Using the analog block diagram, the electrical components are assembled together on the analog computer.

A typical simulation of a physical system involves a mathematical model of a system consisting a set of one or more differential equations and initial conditions on the variables. If the system is linear, the differential equations are linear. The operations required for a linear system are summation, sign inversion, multiplication by a constant, integration and differentiation. Each of these operations can be performed by different elements of the analog computer that

need to be assembled together in order to represent a complete differential equation.

4.2 Simulation Setup

A simulation test-stand was assembled to test the performance of an on-line system identification algorithm. A continuous-time plant whose dynamics and noise characteristics could be varied was simulated on an analog computer.

A Comadyna GP-10S [50] analog computer was used in this simulation test-stand. This analog computer has eight patch-panel operational amplifiers (op-amps). Each of these amplifiers have a provision of up to four integrating capacitors (two slow, two high speed). Amplifiers 1-6 have individual boards that connect 50K and 5K patch-panel input resistors. Amplifiers 1-4 have individual attenuators to adjust integrator initial conditions. There are eight potentiometers (5K variable resistance) available to be used as attenuators with any of the amplifiers. Besides these components, the analog computer has discrete modular elements such as multipliers, invertors etc.

The on-line system identification algorithms runs on a Motorola 68040 microprocessor mounted on MVME167 board resident in a VME crate. The input/output data from the plant, simulated by the analog computer is sampled using a 16 bit-16 channel Analog to Digital Convertor (ADC). The ADC used for

this simulation test stand is a Greenspring computers IndustryPack® IP-16ADC [51].

The IP-16ADC provides 16 single-ended input lines or eight differential input lines. The IP-16ADC provides 16 resolution. Software programmable gains of 1 or 0.5 and software programmable modes, unipolar or bipolar, enable the IP-16ADC to input ranges of 0 to 5 V, ± 5 V, 0 to 10V , and ± 10 V. The bipolar mode with a input range of ± 10 V was used for the simulation test stand. The conversion time for IP-16ADC is 8 μ seconds, which provides a usable throughput of 100000 conversions per second. The IP-16ADC can work in either single conversion or continuous conversion mode. In continuous mode the first conversion is initiated by writing to a unique address in I/O memory space. The subsequent conversions take place automatically after the previous conversion has been completed. In single conversion mode each conversion is started by writing to a unique I/O memory space. Data is ready to read after the Ready bit (SDL bit) has been set. The single conversion mode was selected for the test stand. A device driver was written in C programming language for this ADC. The source code for this device driver can be seen in Appendix B. The device driver performs all the operations needed to read the ADC data and make it available to the system identification algorithm running on the microprocessor.

The IP-16ADC IndustryPack is installed on a VME6U slave board VMESC5 [52] that can support five IndustryPacks. This board is manufactured by Systran Corporation.

The control signal generated by the algorithm running on the microprocessor is sent to the analog computer through a VME Digital to Analog Converter (DAC). The DAC card, DVME-628 [53], manufactured by Datel Corporation is used for this test stand. It provides eight channels of digital-to-analog conversion with 12 bit resolution. Each channel has its own D/A converter which settles to an overall accuracy of $\pm 0.05\%$ of full scale within 6 μ seconds. The DAC card can deliver the analog output signal between the ranges of 0 to 5V, 0 to 10 V, ± 2.5 V, ± 5.0 V, and ± 10 V. A device driver was written for this card which can perform all operations needed to generate an analog correction signal from the digital output generated by the microprocessor.

Two Tektronix 2400 digital oscilloscopes were used to observe the various input/output data generated during the course of simulation on the test stand. A Hewlett-Packard 3132A model function generator was used to generate the various disturbances and reference signals. A two input/two output system was configured on the analog computer to be simulated as the plant. A circuit diagram for the simulated system is shown in Fig. 16.

4.3 Simulation Cases

A two input/two output system was configured on the analog computer to be simulated as the plant.

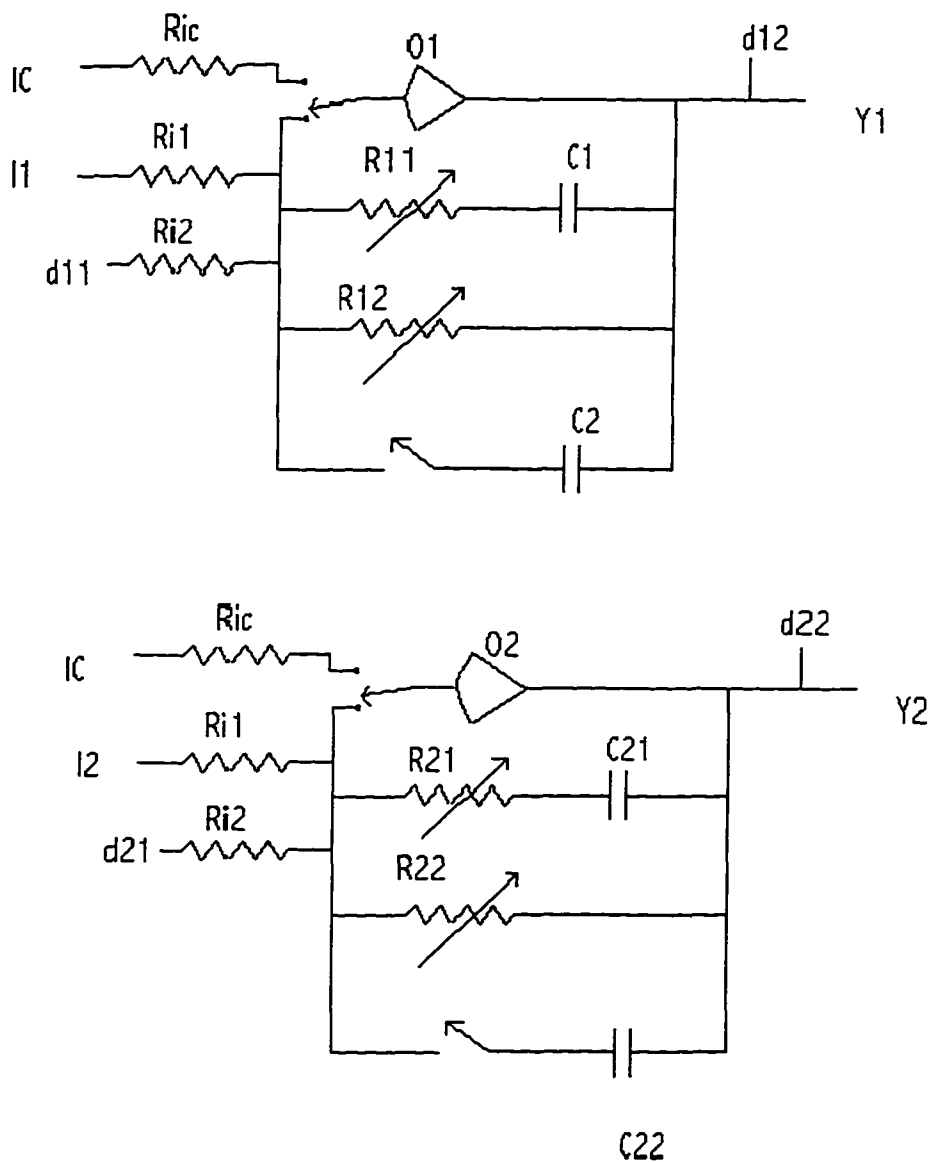


Fig. 16 Schematic diagram of the system simulated using an analog computer

The circuit diagram of the simulated system shown in figure 15 represents a two state system that is decoupled. The input points for this system are denoted as I1 and I2 and the output measurements are available at O1 and O2. The initial conditions can be applied at points denoted by IC. R11, R12, R21, and R22 represent the attenuators that can be varied to change the plant dynamics by changing the RC time constant and integrating action for each of the states. Various noise sources can be applied at points d11, d12, d21, and d22 to affect the process and measurement noise characteristics of the system.

4.3.1 Variation in dynamics of the plant

The first simulated case involved observing the performance of the system identification algorithm under conditions of time-varying dynamics. This was accomplished by varying attenuators R11 and R12 for state 1 and R21 and R22 for state 2 during the course of simulation while the system identification algorithm was running.

Fig. 17 shows a comparison between the actual output and the estimated output generated from the identified system for state 1. The estimated output of the system is computed from identified observer Markov parameters using

$\hat{y}(k+1) = \hat{\theta}_p(k)\psi_p(k)$. Fig. 17 shows the same comparison for output 2.

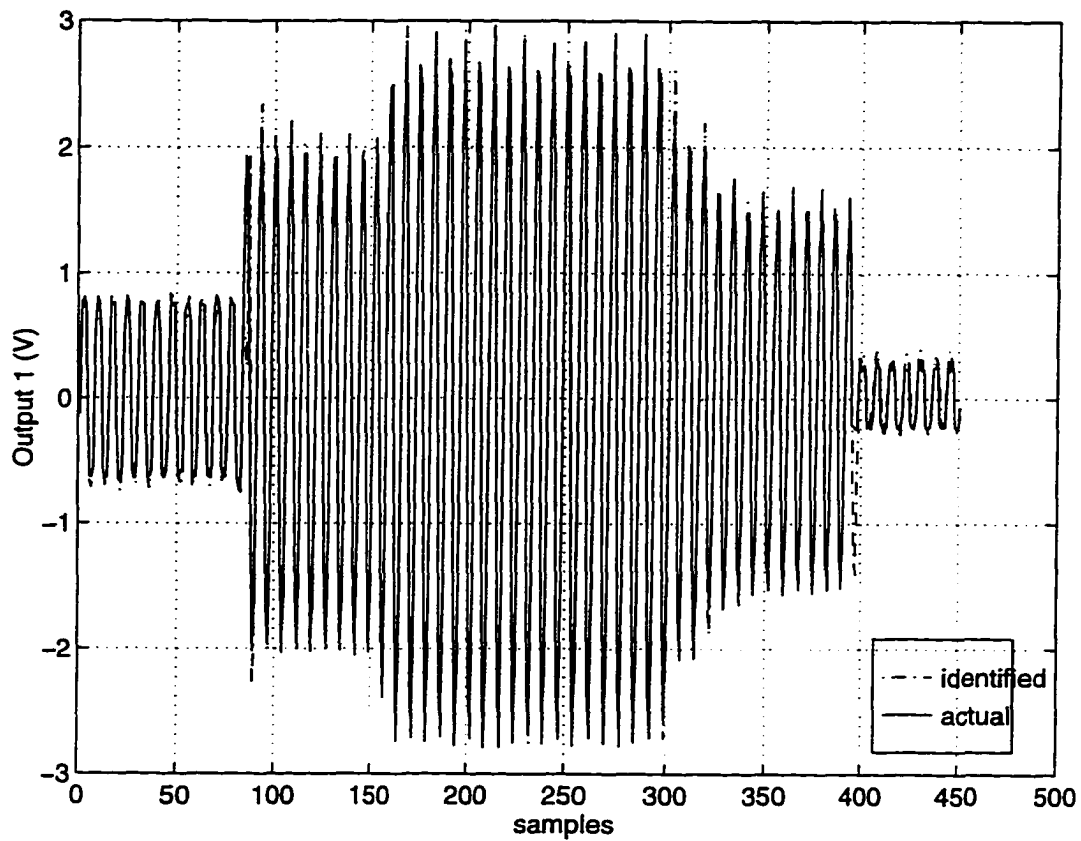


Fig. 17 Comparison between actual and identified output 1. The dynamics of simulated plant was changed at 4 instances by varying attenuators R11, and R12. First change was introduced at sample 90, 2nd at sample 155, 3rd at sample 300 and 4th at sample 395.

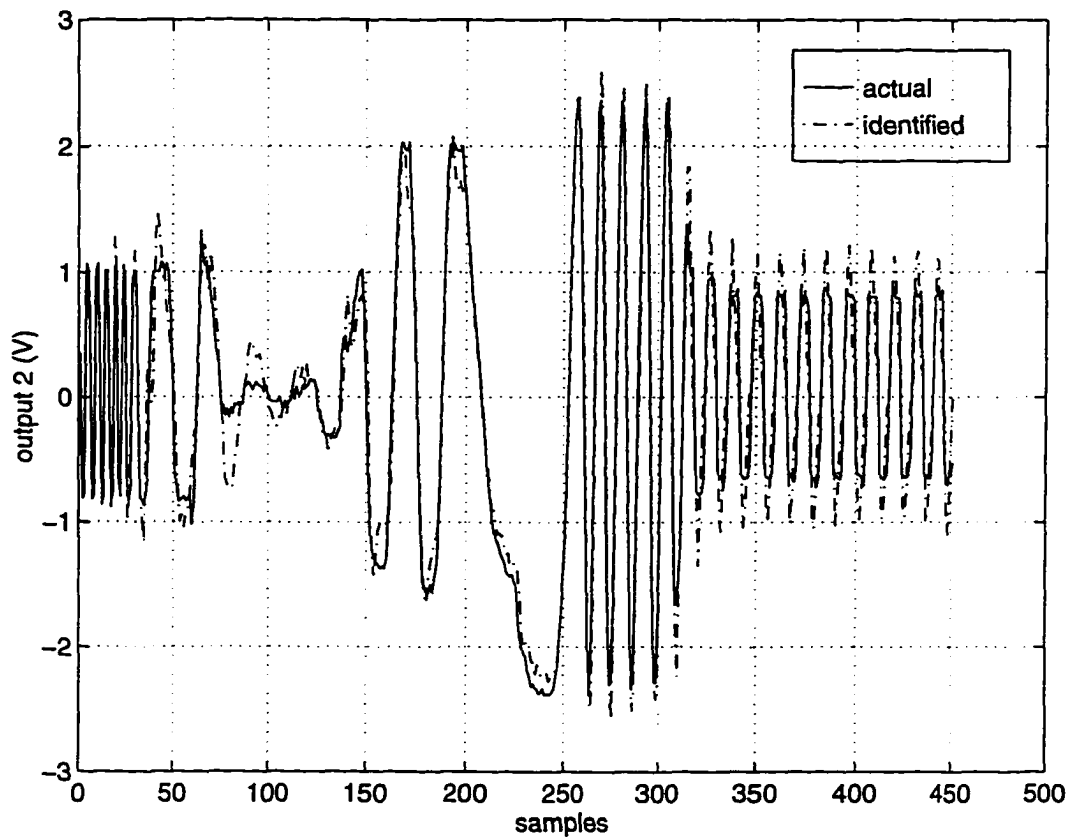


Fig. 18 Comparison between actual and identified output 2. The dynamics of simulated plant affecting state 2 was changed at 6 instances by varying attenuators R21 and R22. First change was introduced at sample 25, 2nd at sample 75, 3rd at sample 145, 4th at sampler 200, 5th at sample 260 and 6th at sample 310

Fig. 19 shows a plot of error between identified and actual output for state 1. This plot shows that estimation error is large at the moment that system dynamics is changed, however the error rapidly drops to a nominal value within 2-3 cycles. The dynamics of the system was changed using attenuators R12, R22. The effect of this change is apparent at sample instant 90 in this plot.

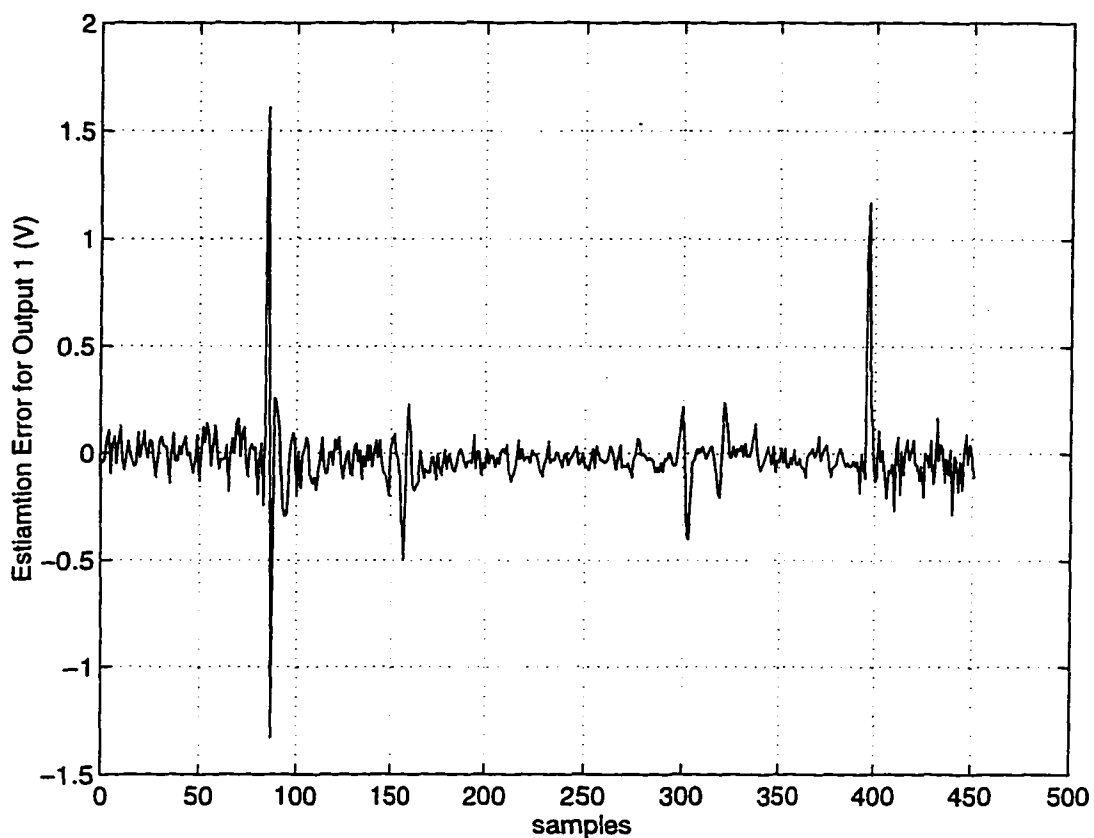


Fig. 19 Estimation error between actual and identified output 1. Spikes at samples 90, 155, 300 and 395 correspond to instances when the dynamics of system affecting state 1 was changed.

Fig. 20 shows disturbance suppression achieved by applying the correction signal generated by the system identification algorithm. This figure shows that disturbance is suppressed to within 4% most of the time. However it rises to 8% momentarily when the dynamics is changed.

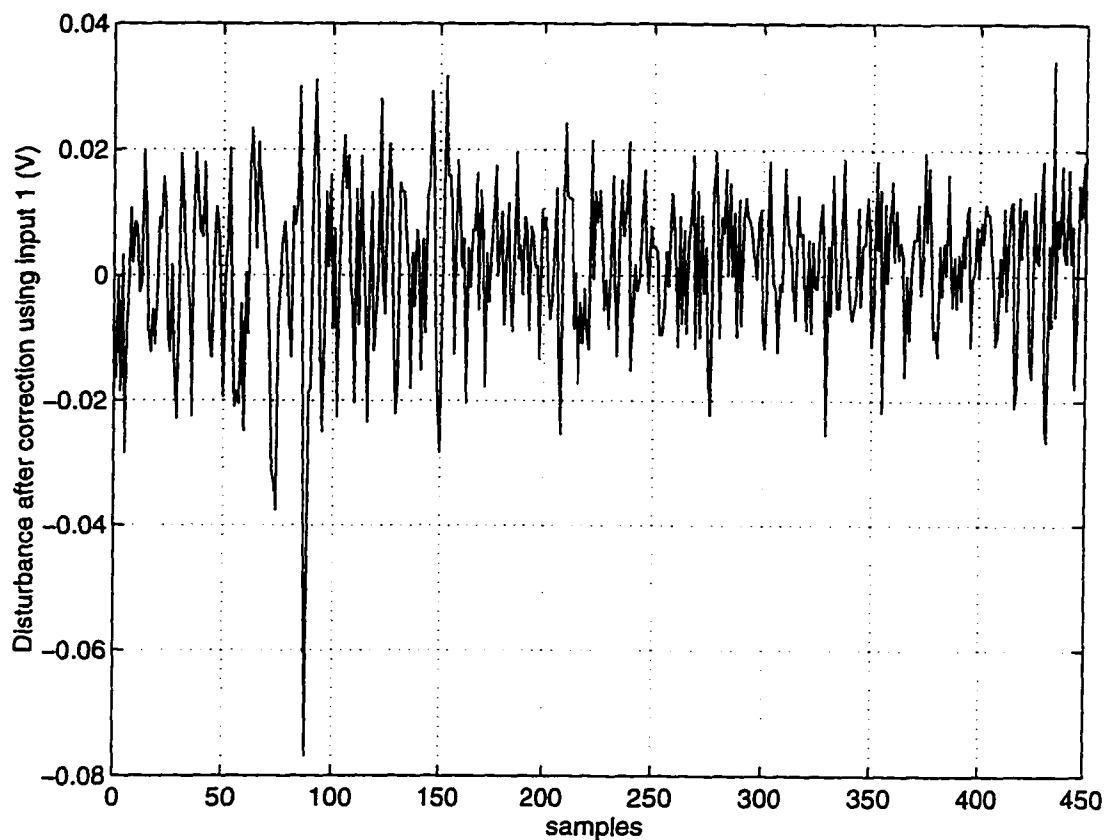


Fig. 20 Disturbance suppression achieved by application of feedback correction using input 1 for the case where system dynamics was changed. Spike observed at sample 90 corresponds to the first instance where system dynamics was changed.

4.3.2 Variation in noise characteristics

The first test involved adding 5% random noise to input d11 and d12 and varying the input signal during the course of simulation. The system dynamics was not changed. Fig. 21 shows a comparison between actual and identified output 1.

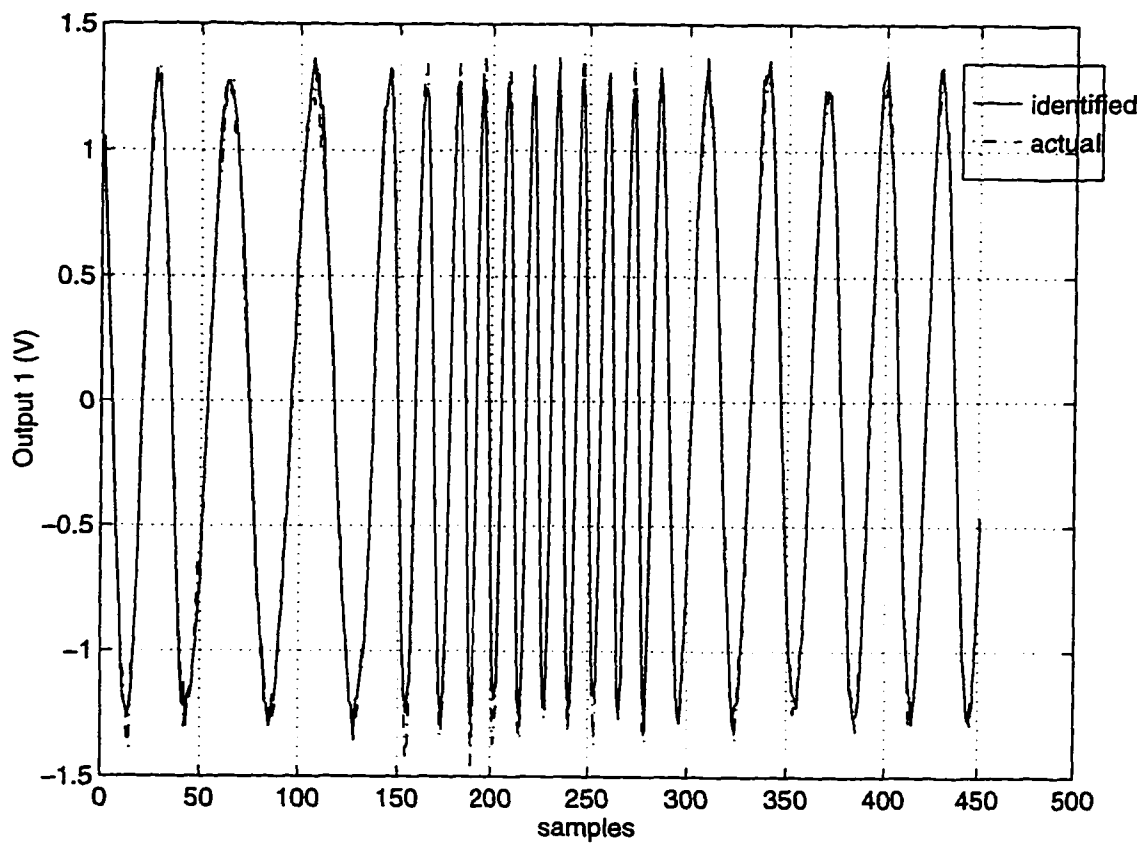


Fig. 21 Comparison between identified and actual output 1 for the case where 5% random noise was added to measurement and control input data and at points d11 and d12.

The estimation error for output 1 shown in Fig. 22 indicates that the nominal value of estimation error is larger compared to the noise-free case where system dynamics was changed as shown in Fig. 19.

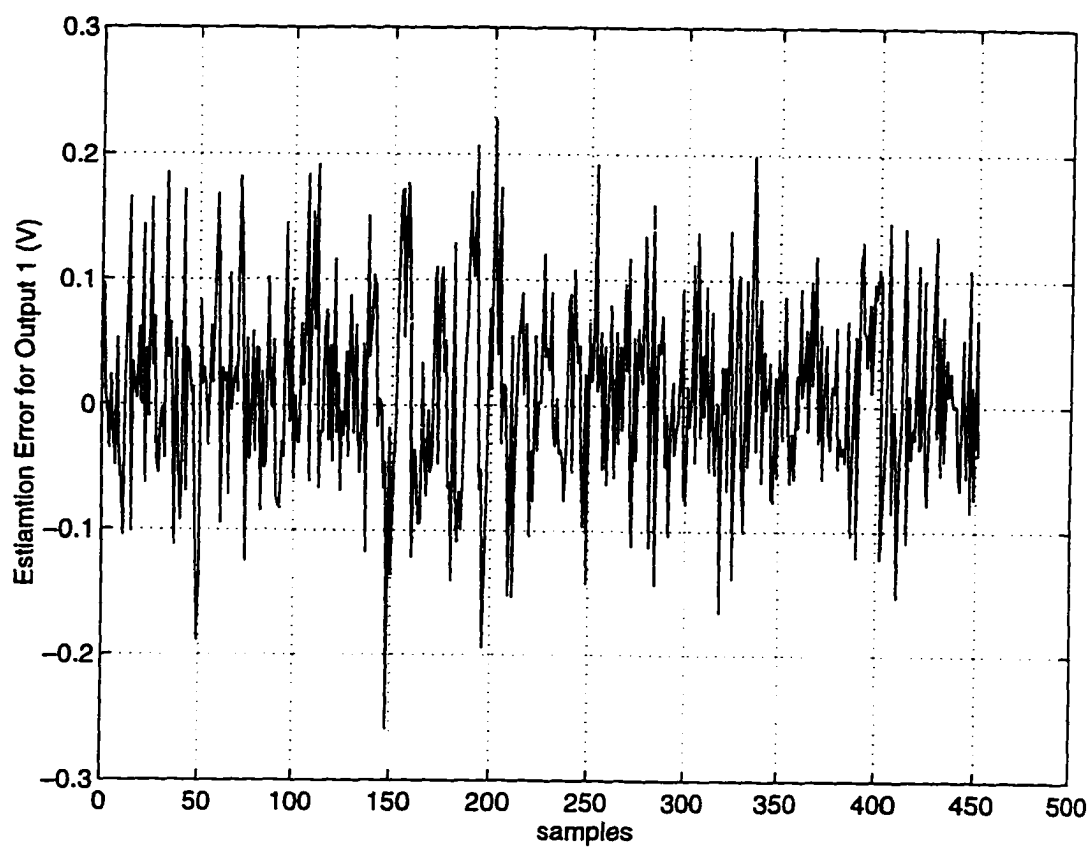


Fig. 22 Estimation error between actual and identified output 1 for the case where 5% random noise was added to points d11 and d12.

Fig. 23 shows that, with application of feedback correction signal generated for the identified system, the disturbance is suppressed to within 8-10% for this case.

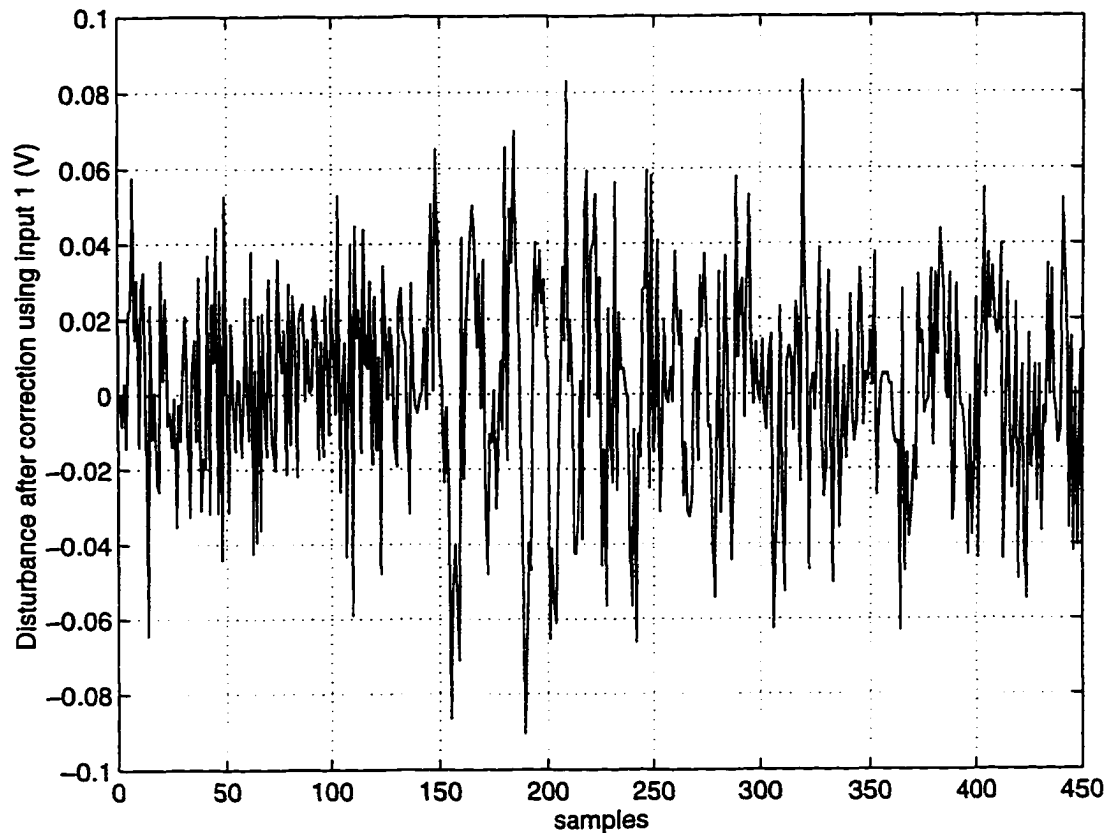


Fig. 23 Disturbance suppression achieved after application of feedback correction using input 1 for the case where 5% random noise was added at points d11 and d12.

In the second simulation case, the process noise characteristics were varied by changing the amplitude and frequency of the disturbance source applied to point

d11 and d12 shown in circuit diagram in Fig. 16. A comparison of the actual output with the estimated output 1 is shown in Fig. 24.

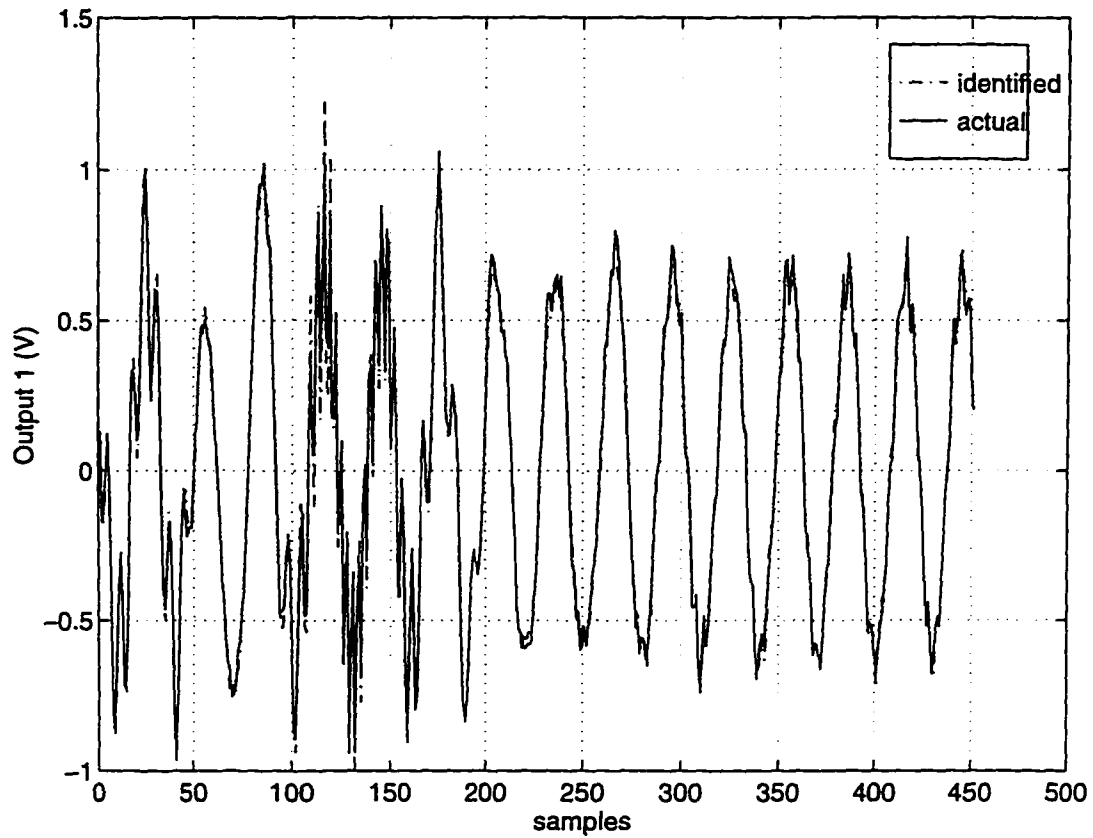


Fig. 24 Comparison between actual and identified output 1 for the case where amplitude and frequency of noise signal applied at d11 and d12 was changed.

Fig. 25 shows the estimation error between actual and identified output. The estimation error becomes large momentarily when the amplitude and frequency of the disturbance is increased.

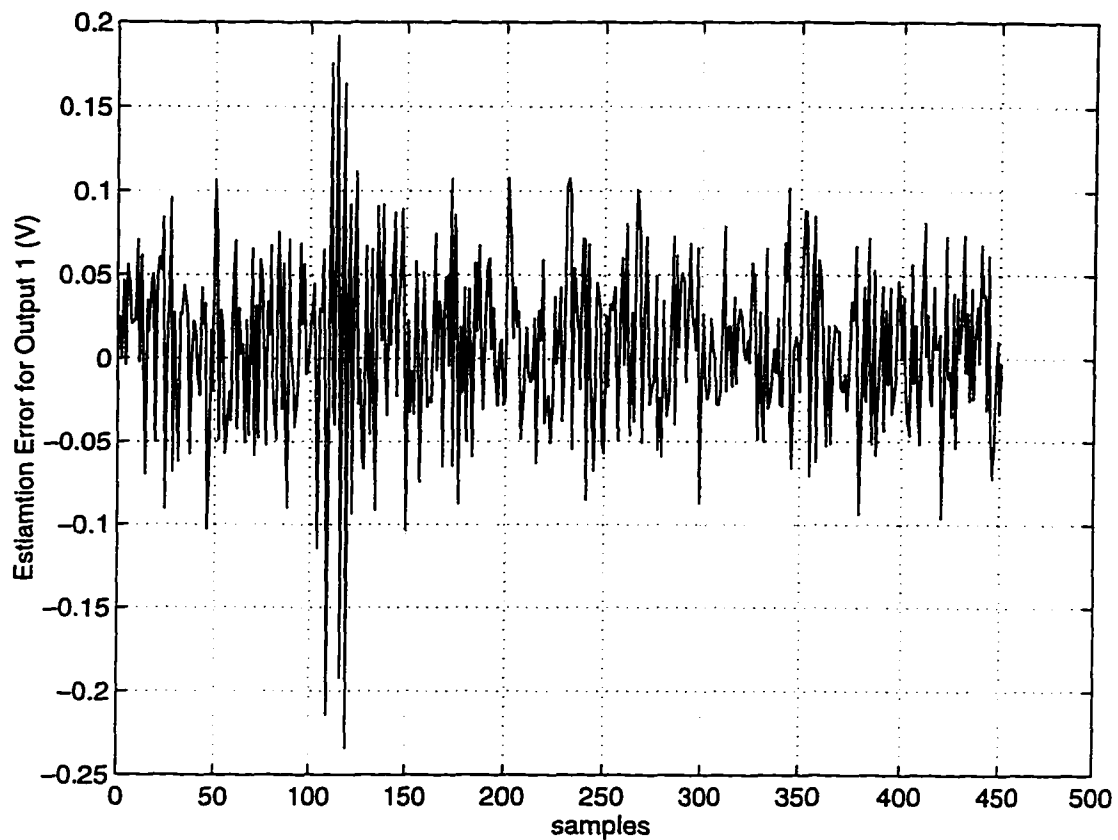


Fig. 25 Estimation error between actual and identified output 1 for the case where amplitude and frequency of noise was changed. Error becomes large at sample 110 when amplitude and frequency of noise are increased.

Fig. 26 shows that by applying a feedback correction signal generated by system identification algorithm, the disturbance is suppressed to within 6% most of the time. However the residual error rises to 8% momentarily when the noise characteristics are changed.

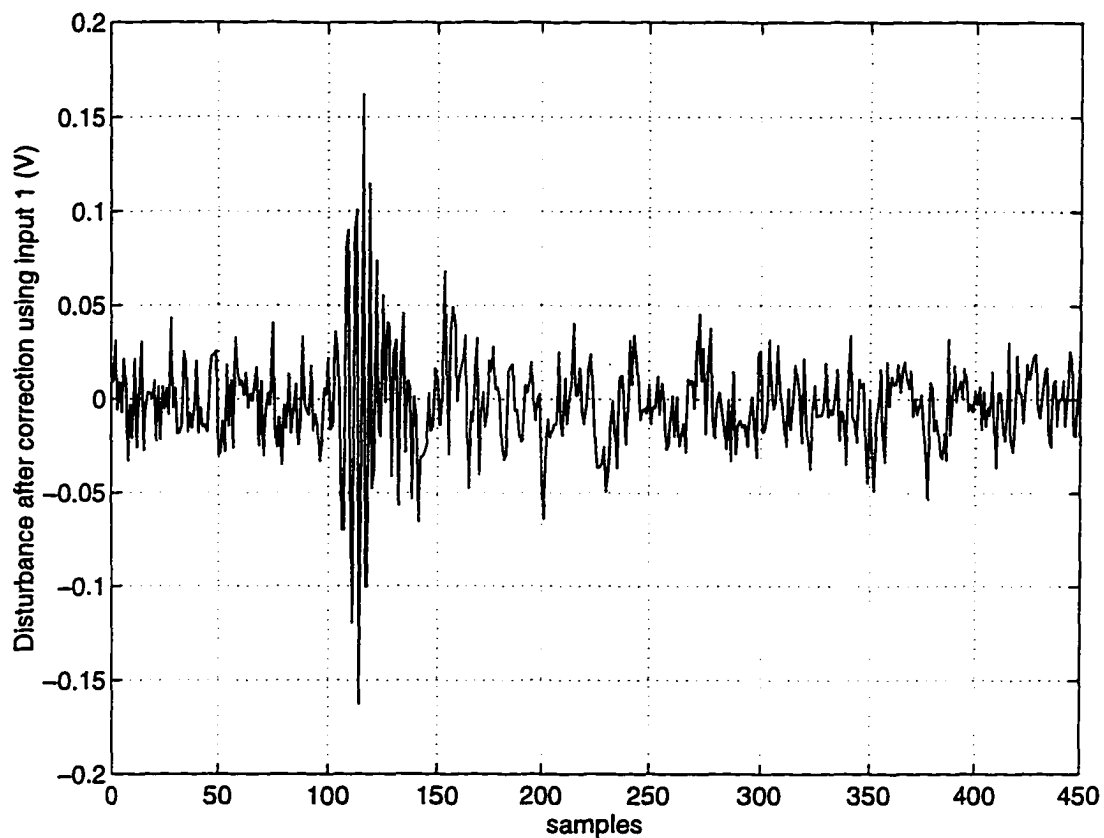


Fig. 26 Disturbance suppression achieved by application of feedback correction using input 1 for the case where amplitude and frequency of noise are changed. Suppression reduces at sample 110 when amplitude and frequency of are increased.

4.3.3 Variation in noise characteristics and dynamics

In this simulation both the noise characteristics and the system dynamics were changed while the performance of the system identification algorithm was observed. The system dynamics were varied by changing the attenuator R12 and adding capacitor C12 in the feedback circuit Fig. 27 shows a comparison between the actual and the identified output.

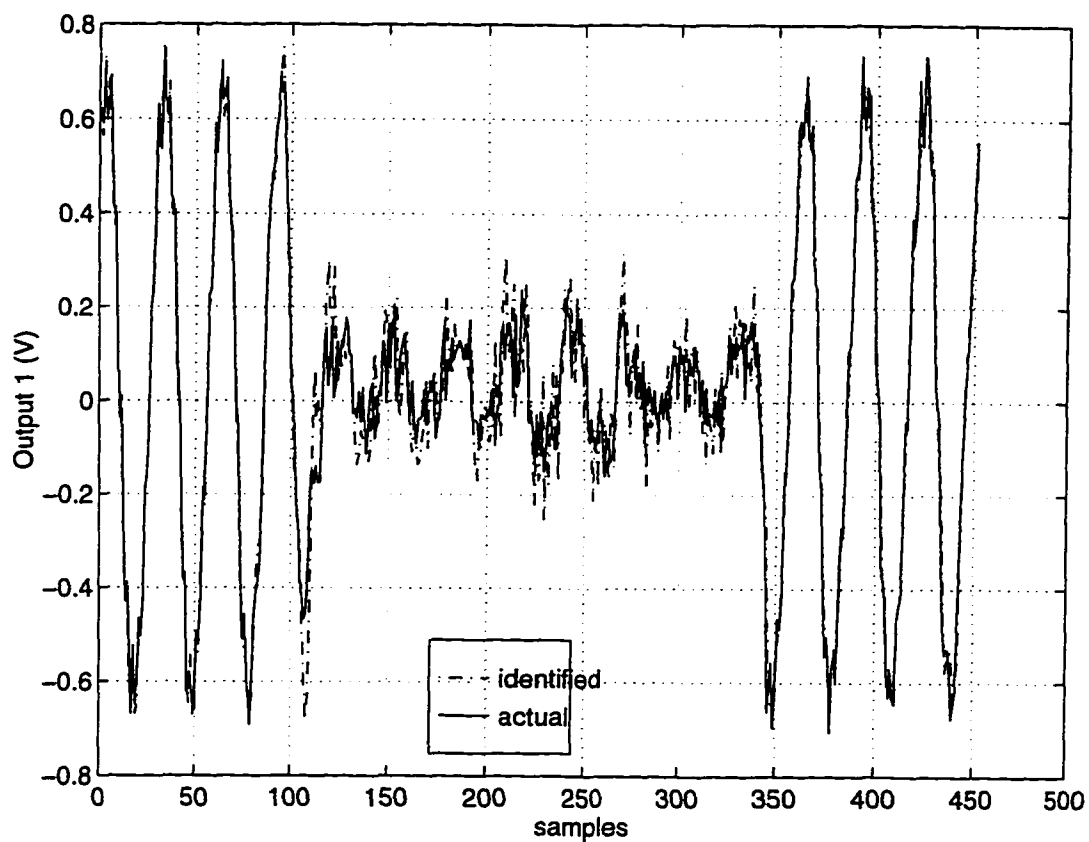


Fig. 27 Comparison between actual and identified output 1 for the case where dynamics of the system and noise characteristics were changed simultaneously.

. Noise characteristics were changed by changing the frequency and amplitude of disturbance applied at points d11 and d12. Fig. 27 shows the estimation error between identified and actual output 1. Estimation error becomes large when the system dynamics and noise characteristics are changed. The error drops to nominal value within 3 cycles. The estimation error observed for this case is the larger than the cases where either the dynamics or noise were changed separately.

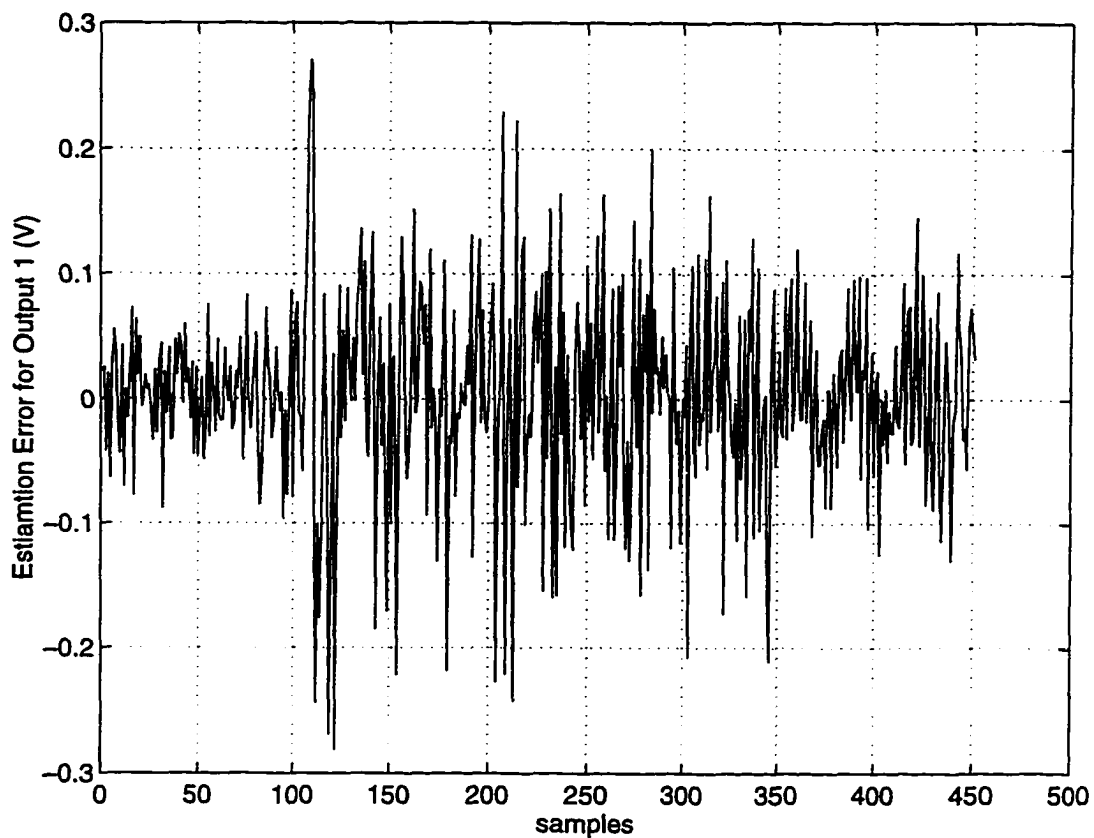


Fig. 28 Estimation error between actual and identified output 1 where system dynamics and noise characteristics were varied simultaneously. Spike at sample 110 corresponds to the instance where dynamics of system was changed.

Fig. 29 show the disturbance suppression achieved by application of feedback correction in this case. It can be observed from Fig.(s) 27 and 29 that the disturbance suppression is smaller when the output measurement signal amplitude is decreased.

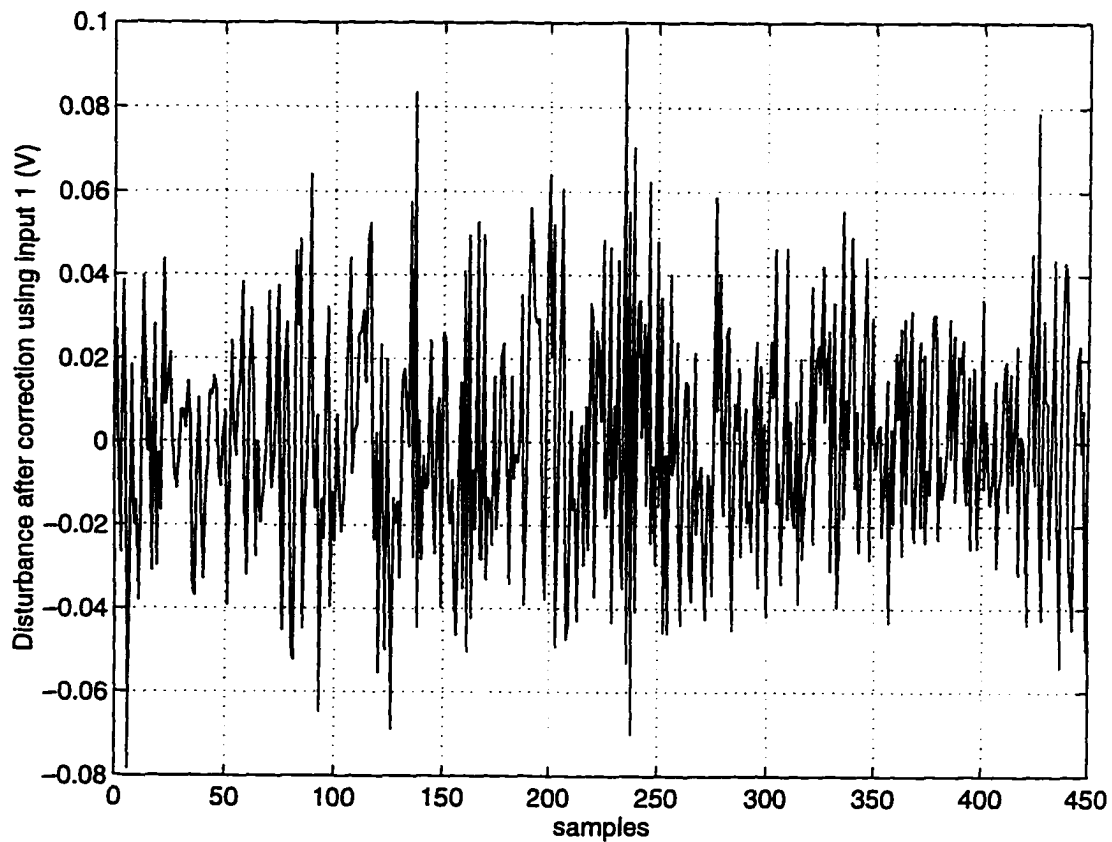


Fig. 29 Disturbance suppression achieved by application of feedback correction using input 1 for the case where system dynamics and noise characteristics were varied simultaneously.

4.4 Model Order Selection

Model order for a system may not be known accurately *a priori*. A higher model order often results in a better fit between identified and actual system model. However, for some system identification algorithms a higher model order would result in a disproportionately large computational load. FTF has an advantage in this regard for on-line implementation.

Increasing the model order beyond a certain point will not result in any more improvement in model fit. There are various criteria that can be applied to determine the model order which would result in the best model fit. The one applied in this work is described as

$$\text{tr}[E_{\pi}(k)E_p^T(k)] = \text{tr}[(Y(k) - \bar{\theta}_p \Psi_p(k-1))(Y(k) - \bar{\theta}_p \Psi_p(k-1))^T]$$

which is the trace of the equation error squared. This is also known as the “loss function”. The experimental data collected from the analog computer test stand indicates that the model order greater than 5 will not result in any added improvement in the model fit. Fig. 30 shows a graph of loss function plotted against model order p .

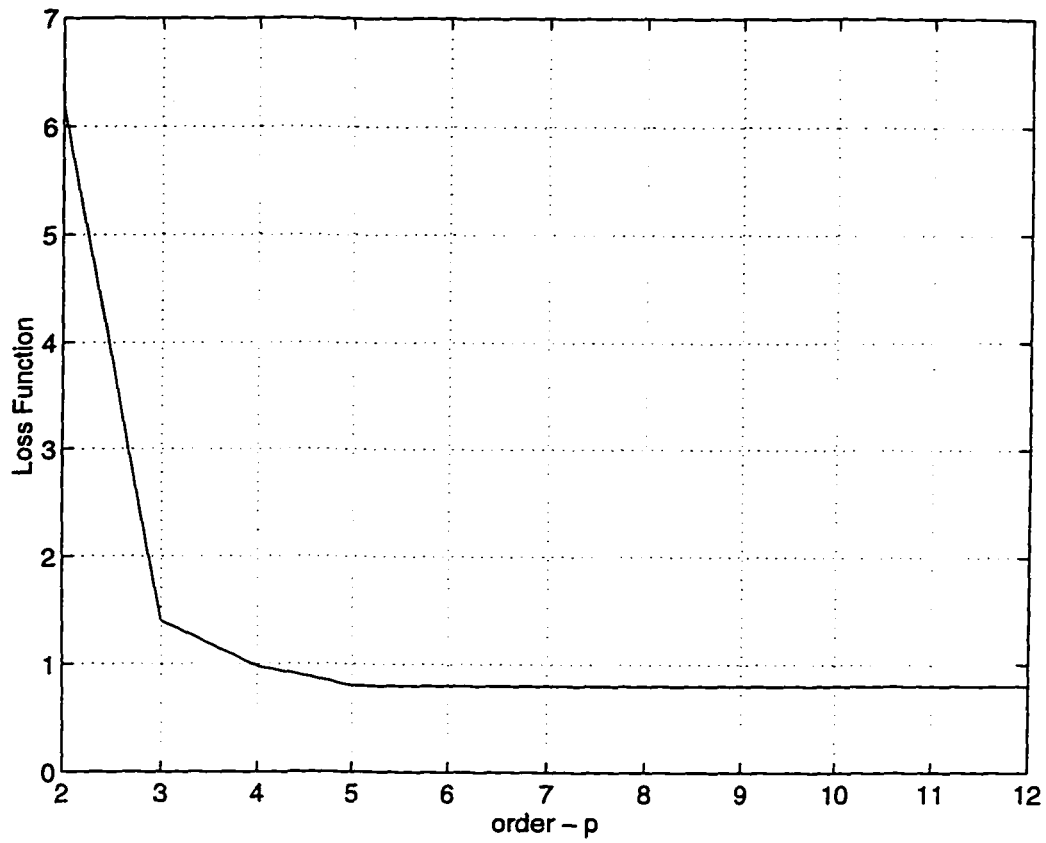


Fig. 30 Loss function vs. Model order

CHAPTER V

SUMMARY OF RESULTS AND CONCLUSIONS

In this chapter a review of various results presented in previous chapters and conclusion drawn from this study are presented. Possible extensions of this research are also presented.

5.1 Summary and Conclusions

Results of various noise measurements conducted on the accelerator were presented in Chapter 2. These results indicate that line power harmonics are the primary sources of disturbance on beam orbit and energy. The strongest disturbance component corresponds to 60 Hz frequency and the largest frequency component observed is at 180 Hz. The amplitude of the 60 Hz component for beam position variation can be as large as 1.98 mm in the X plane, observed at BPM locations in the injector. The largest amplitude of the beam energy variation $\Delta E/E$ was observed in the injector. Its amplitude was 1.38×10^{-3} . These disturbance components vary with time. Table 1 displays variations in amplitude of the 60 Hz component at various locations in the accelerator. Table 2 shows the

variation in these disturbance components observed from data collected using SEE BPM low level data acquisition software.

The performance data of the prototype fast feedback system that has been implemented in the injector and East Arc region of the accelerator was described in chapter 2. The transient response characteristics of fast orbit and energy lock systems, sampling at 60 Hz rate, indicate that these systems can correct a DC error within 4 samples (66.67 msec.). The disturbance suppression observed for fast energy lock indicates that the RMS error can be reduced by a factor of 10 by application of feedback control. The power spectra for the $\Delta E/E$ observed before and after application of a correction signal indicates that the power of deterministic disturbance signal (6 Hz) was reduced by over two orders of magnitude by application of the correction signal.

The theory and implementation of an on-line system identification algorithm, Fast Transversal Filter, was described in chapter 3. Computation efficiency of FTF makes it attractive for on-line implementation. Timing studies performed on the algorithm indicate that model order p does not affect the CPU computational load of FTF for a given number of inputs and outputs. Fig. 31 shows a plot between CPU time and model order. Increasing the number of inputs and outputs does affect the time required for one FTF iteration, which is approximately proportional to $(r + m)^3$, where r is the number of inputs and m is the number of outputs. A

catchup technique has been implemented to further improve the performance of the FTF algorithm for on-line implementation.

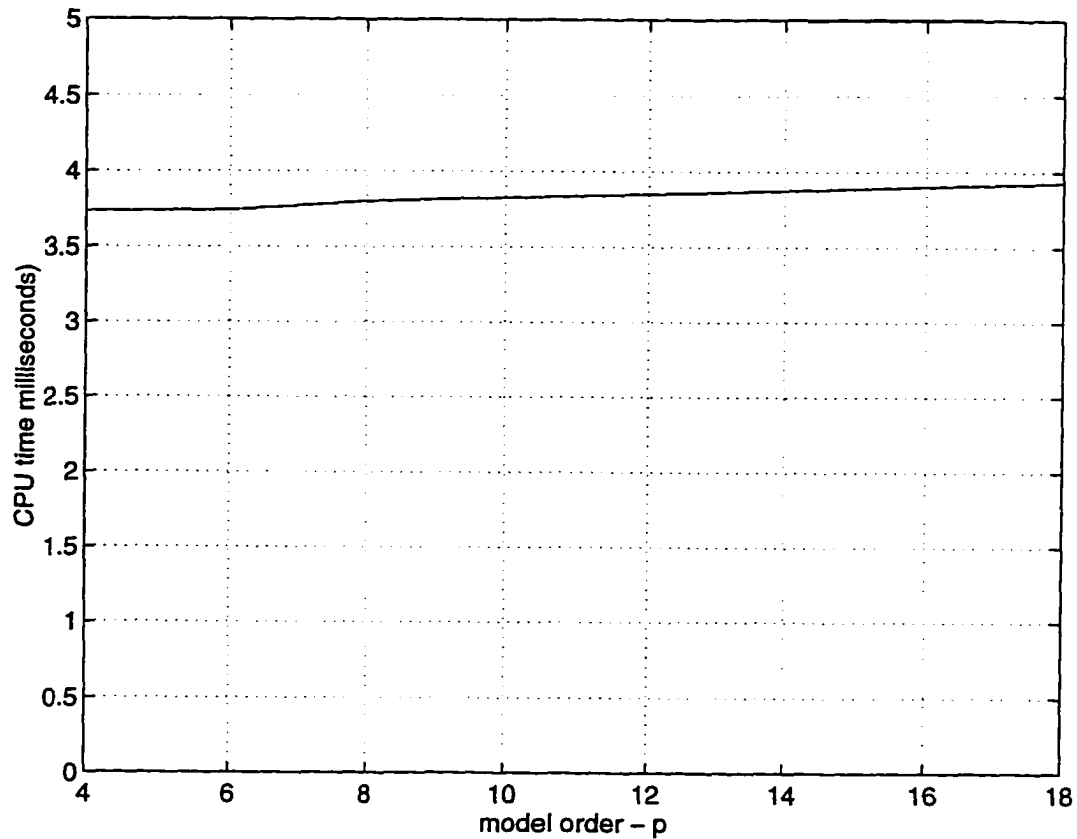


Fig. 31 CPU time/FTF iteration vs. Model order

Performance of the implementation of the FTF algorithm for on-line system identification was verified on a simulation test-stand whose detailed description was presented in chapter 4. An analog computer was used to simulate a continuous-time plant. The FTF algorithm was executed on the test-stand microprocessor, which is similar to CEBAF control system microprocessors. Various test cases that were studied included on-line identification under

conditions of varying plant dynamics, disturbance signal, and noise characteristics. Results from a simulation where the plant dynamics and the noise characteristics both vary with time indicate that the disturbance can be suppressed to within 8% nominally. Observations of processor time consumption indicate that one iteration of FTF for a two input/two output system with model order of 5 consumes 3.795 milliseconds.

The FTF algorithm implementation described in this dissertation, with added exception handling and minor modifications can be used to monitor and improve the performance of the fast orbit and energy feedback systems currently installed in the CEBAF control system.

5.2 Future Work

The processor time consumption per iteration of FTF for a system with large number of inputs and outputs can further be reduced by applying a recursive scheme for obtaining the matrix inverse of \bar{E}_p , described in step 7 of forward time estimation computation procedure.

On-line FTF implementation as described in this work could further be enhanced to perform the task of on-line system diagnostics and fault prediction. Since the observer Markov parameters have in them embedded all the system

matrices relevant to a feedback loop they can provide a wealth of information about the state of various actuators and sensors.

FTF algorithm could be implemented on a specialized microprocessor, with access to ADCs and DACs that interface with the dynamic system, to solve a general class of control problems. By applying a known excitation to available inputs of the plant and monitoring the outputs, FTF can construct the information about the system model and start generating control action for a desired performance objective thereafter. It can also tune itself to varying conditions of plant dynamics and noise characteristics by constantly observing the input/output data from the system. The processor power will be the limiting factor for the sample rate, and the number of inputs and outputs that could be considered for such a solution.

APPENDIX A

Table A.1: Beam Requirements - general characteristics

Parameter	Nominal Value and Range	Stability ^d (for hours)
Beam Emittance: rms spot size for achromatic beam tune (1σ) (> 1 Hz)	<u>Hall A</u> ^{b,c} : $20\ \mu\text{m} < \sigma_x < 50\ \mu\text{m}$ $20\ \mu\text{m} < \sigma_y < 50\ \mu\text{m}$ <u>Hall B</u> : $20\ \mu\text{m} < \sigma_x < 70\ \mu\text{m}$ $20\ \mu\text{m} < \sigma_y < 70\ \mu\text{m}$ <u>Hall C</u> : $50\ \mu\text{m} < \sigma_x < 100\ \mu\text{m}$ $50\ \mu\text{m} < \sigma_y < 100\ \mu\text{m}$	25% of value
Beam Emittance: angular divergence (1σ) (> 1 Hz) ^a	$\sigma_x, \sigma_y < 100\ \mu\text{r}$	25% of value
Beam position (< 1 Hz) ^a	0 μm (relative to monitor axis)	rms deviation is less than 25% of the beam spot's rms radius
Beam direction (< 1 Hz) ^a	0 μr (relative to monitor axis)	rms deviation is less than 25% of the beam angular divergence rms $\frac{1}{2}$ cone angle
Energy Spectrum (1σ) (> 1 Hz) ^a	<u>Hall A</u> : $\sigma_E/E < 5\ \text{E-5}$ <u>Hall B</u> : $\sigma_E/E < 4.0\ \text{E-4}$ <u>Hall C</u> : $\sigma_E/E < 2.5\ \text{E-4}$	25% of value
Energy (average) (< 1 Hz) ^a	0.5 - 4 GeV	<u>Hall A</u> : $< 3\ \text{E-4}$ <u>Hall B</u> : $< 1\ \text{E-3}$ <u>Hall C</u> : $< 1\ \text{E-3}$ (also $< 3\ \text{E-3}$ over days for all)
Background (Beam Halo)	$< 1\ \text{E-6}$ of Total Current at 5σ (with diagnostic to be provided by the experiment)	any value within nominal range
Current (dc average) (< 1 Hz) ^{a,c} (Note: any single hall is	<u>Hall A</u> : 40 nA - 180 μA <u>Hall B</u> : 1 nA - 10 μA <u>Hall C</u> : 40 nA - 180 μA	within 10% of value requested by experimenter

restricted to < 120 μA unless it has exclusive use of the beam, and total current delivered to all 3 halls must be less < 180 μA)		
Polarization (current range to be determined by agreement between Physics and Accelerator Divisions)	>35% (from bulk Gas, with expected currents of order 100 μA) > 75% (from strained cathodes, with currents of order 30 μA expected)	< 10% of value
Effective Duty Factor	> 90% (lower values may be negotiated with the Accelerator Division)	any value within nominal range (90%-100%)
Proper Impingement on Beam Dump (raster)	rastered beam spot size > 100 μm stability of position < 1 cm (not including rastering)	

^a Note: We include in the definitions of beam emittance and energy and energy spectrum all components of the beam emittance at frequencies above 1 Hz; components at frequencies below 1 Hz are considered part of the beam position and direction (for transverse phase space) and as part of energy average (for longitudinal phase space)

^b Note: Hall A also requires an achromatic tune in which the beam spot is 2 mm < σ_x < 3mm and 20 μm < σ_y < 50 μm

^c Note: Hall A and C also require a dispersive tune which results in larger spot sizes on the target depending on the energy spread in the beam

^d Note: The limits identified in all stability specifications are to be considered as “windows” on the measured values of the quantities

^e Note: The high frequency (> 1 Hz) fluctuations in the beam current are specified through the effective duty factor

APPENDIX B

SOURCE CODE FOR THE ON-LINE SYSTEM

IDENTIFICATION ROUTINES

```

/* DAQff.c */

/* ----- */
/*
/* Module : DAQff.c
/* Function :
/* This program collects the input/output data by reading the ADC. Input/
/* output data is filled in separate ring buffers which are accessed by the
/* Initialization and FTF forward prediction routines for implementation
/* of catchup technique
/*
/* Rel   Date       Author           Comments
/* 0     8 Aug96    Mahesh Chowdhary  Initial Release
/* ----- */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <rngLib.h>
#include <taskLib.h>

#define BUFFERSIZE 1000

/* Id for Ring Buffers to store Input/Output data */

RING_ID mgU1, mgU2, mgY1, mgY2;

/* variables for the IP ADC */

```

```

/* Control and Status register */
unsigned short *cnsr = (unsigned short *)0xffff0000;

/* Data register */
unsigned short *dreg = (unsigned short *)0xffff0004;

/* Trigger ADC Conversion Register */
unsigned short *trig = (unsigned short *)0xffff000c;

unsigned short data, ccnsr;

int points=0;
int FTFDone=1;

DAQfft( )
{

FILE *fpinput, *fpoutput;
void resetADC( );
void setChan1( );
void setChan2( );
void setChan3( );
void setChan4( );
void readADC( );

int i, status;
double tmp, U1[5000], U2[5000], Y1[5000], Y2[5000];

/* create ring buffers to store input/output data */

rngU1 = rngCreate(BUFFERSIZE*sizeof(double));
rngU2 = rngCreate(BUFFERSIZE*sizeof(double));
rngY1 = rngCreate(BUFFERSIZE*sizeof(double));
rngY2 = rngCreate(BUFFERSIZE*sizeof(double));

/* Open files to store input/output data for later analysis */

if((fpinput = fopen("input","w"))==0)
{
printf("cannot open the input file \n");
exit(1);
}

```

```

if((fpoutput = fopen("output","w"))==0)
{
    printf("cannot open the output file \n");
    exit(1);
}

/* perform the software reset of ADC */
resetADC( );

while(!FTFDone)
{
    setChan1( );
    readADC( );
    tmp = (data - 32768)*(10.0/65536);
    U1[points]= tmp;
    status = rngBufPut(rngU1, (char *)&tmp, 8);

    setChan2( );
    readADC( );
    tmp = (data - 32768)*(10.0/65536);
    U2[points] = tmp;
    status = rngBufPut(rngU2, (char *)&tmp, 8);

    setChan3( );
    readADC( );
    tmp = (data - 32768)*(10.0/65536);
    Y1[points] = tmp;
    status = rngBufPut(rngY1, (char *)&tmp, 8);

    setChan4( );
    readADC( );
    tmp = (data - 32768)*(10.0/65536);
    Y2[points] = tmp;
    status = rngBufPut(rngY2, (char *)&tmp, 8);

    points++;
    if(points > 5000) points = 0; /* don't need > 5000 points for analysis */
}

taskPrioritySet(taskNameToId("DAQftf"),185);

for(i=0;i<points;i++)

```

```

    {
        fprintf(fpinput,"%g %g \n", U1[i], U2[i]);
        fprintf(fpoutput,"%g %g \n", Y1[i], Y2[i]);
    }

fclose(fpinput);
fclose(fpoutput);
}

/* function to perform a software reset of ADC */
void resetADC( )
{
    *cnsr = 0x2168;
    ccnsr = 0;
    ccnsr = *cnsr;
    while((ccnsr & 0x2000) == 0)
    {
        ccnsr = *cnsr;
        printf("Reset the ADC, cnsr = %x \n", ccnsr);
    }
}

/* select ADC channel 1 to be read */
void setChan1( )
{
    /* write to bits 3, 5, and 6 of Control& stat reg for +/- 10 V for chan 1 */
    *cnsr = 0x2168;
    ccnsr = 0;
    ccnsr = *cnsr;
}

/* select ADC channel 2 to be read */
void setChan2( )
{
    /* write to bits 4, 5, and 6 of Control& stat reg for +/- 10 V for chan 2 */
    *cnsr = 0x2170;
    ccnsr = 0;
    ccnsr = *cnsr;
}

/* select ADC channel 3 to be read */
void setChan3( )

```

```
{
*cnsr = 0x2169;
ccnsr = 0;
ccnsr = *cnsr;
}

/* select ADC channel 4 to be read */
void setChan4()
{
*cnsr = 0x2171;
ccnsr = 0;
ccnsr = *cnsr;
}

/* perform an ADC read on the selected channel */
void readADC()
{
/* trigger the adc to convert data */
*trig = 0xffff;

/* delay for 16 msec */
taskDelay(1);

/* read the sdl register */
ccnsr = *cnsr;
data = *dreg;
}
```

```

/* runfft.c */

/* ----- */
/*
/* Module : runfft.c
/* Function : Reads the input/output data from ring buffers in order
/*             implement catchup technique. First, initialization
/*             routine (ftfinit) is called after  $k > (r+m)p+r$  data
/*             points have been collected. ftf routine is called
/*             thereafter for each new measurement.
/*             The vector and matrix memory allocations and matrix inversion
/*             routines used here are from the book:
/*             Numerical Recipes in C by Press, W.H., W.H., Teukolsky, S.A.,
/*             Vetterling, W.T., and Flannery, B.P, Cambridge University
/*             Press, 1983
/*
/* Rel      Date      Author      Comments
/* 0        8 Aug96   Mahesh Chowdhary  Initial Release
/* ----- */

#include <stdio.h>
#include <stdlib.h>
#include "nrutil.h"
#include <string.h>
#include <rngLib.h>
#include <taskLib.h>

/* global variables declared in DAQfft.o */

extern int FTFDone;
extern int points;

/* ring buffers for input/out data (declared in DAQfft.o) */

extern RING_ID rngU1, rngU2, rngY1, rngY2;

runfft()
{

FILE *fp;

```

```

int i=0,j,n=500,init_pts, status;
int m=2,r=2,p=5,SizeX,rmm;
float **utemp,**ytemp,*u,*y,*uk1,**Up,**Yp;
float **uInit,**yInit;
double uInp[2],yMeas[2];
double **ThetaF,**ThetaB,**EsqF,**EsqB,*error;
double *Gammak,*Gpk;
double *GpA,*GpR,GmP[1],*Yh;

```

```

SizeX=(r+m)*p+r;
rmm=r+m;
init_pts=2*((r+m)*p+r);

```

```

/* memory allocation for vector */

```

```

u=vector(0,r-1);
y=vector(0,m-1);
Yh=dvector(0,rmm-1);
error=dvector(0,rmm-1);
Gpk=dvector(0,SizeX-1);
GpA=dvector(0,rmm-1);
GpR=dvector(0,SizeX-1);
Gammak=dvector(0,1);

```

```

/* memory allocation for matrices */

```

```

utemp=matrix(0,r-1,0,n);
ytemp=matrix(0,m-1,0,n-1);
Up=matrix(0,r-1,0,p-1);
Yp=matrix(0,m-1,0,p-1);
uInit=matrix(0,r-1,0,init_pts-1);
yInit=matrix(0,m-1,0,init_pts-1);
uk1=vector(0,r-1);
Up=matrix(0,r-1,0,p-1);
Yp=matrix(0,m-1,0,p-1);
ThetaF=dmatrix(0,rmm-1,0,SizeX-1);
ThetaB=dmatrix(0,rmm-1,0,SizeX-1);
EsqF=dmatrix(0,rmm-1,0,rmm-1);
EsqB=dmatrix(0,rmm-1,0,rmm-1);

```

```

if((fp = fopen("results","w"))==0)
{

```

```

printf("cannot open the results file \n");
exit(1);
}

/* read in the data */

/* Check for sufficient number of data points for FTF initialization */
/*  $k > (r+m)*p+r$  */

if(points < init_pts)
{
printf("ERROR: not sufficient number of DATA points for initialization \n");
printf("FTF Initialization could not begin \n");
exit(1);
}
for(j=0;j<=n-1;j++)
{
status = rngBufGet(rngU1, (char *)&utemp[0][i], 8);
status = rngBufGet(rngU2, (char *)&utemp[1][i], 8);
status = rngBufGet(rngY1, (char *)&ytemp[0][i], 8);
status = rngBufGet(rngY2, (char *)&ytemp[1][i], 8);
}

/* prepare input/output data matrices for FTF initialization */

for(i=0;i<=r-1;i++)utemp[i][n]=0;

for(i=0;i<=init_pts-1;i++)
{
for(j=0;j<=r-1;j++)
{
uInit[j][i]=utemp[j][i];
}

for(j=0;j<=m-1;j++)
{
yInit[j][i]=ytemp[j][i];
}
}

/* prepare matrices for ftf iteration */

```

```

    for(i=0;i<=p-1;i++)
    {
        for(j=0;j<=r-1;j++)
    {
        Up[j][i]=uInit[j][(init_pts-1-p+i)];
    }
    }

    for(j=0;j<=r-1;j++)
    {
        u[j]=utemp[j][init_pts-1];
    }

    for(i=0;i<=p-1;i++)
    {
        for(j=0;j<=m-1;j++)
        {
            Yp[j][i]=yInit[j][(init_pts-1-p+i)];
        }
    }

/* the FTF initialization routine */

ftfInit(r,m,p,(init_pts-1),SizeX,uInit,yInit,ThetaF,ThetaB,EsqF,EsqB,Gammak,
Gpk);

/* FTF forward and backward time iterations until FTFDone */

while(!FTFDone)
{
/* read the data from ring buffers */

/* The i/o data comes from the ring buffers starting at the last time
index used by ftfInit initialization routine. There is no loss of
data. estimated Yh and Theta's computed by the ftf will lag in time
in the beginning but catchup will take only a few iterations */

    if(rngIsEmpty(rngU1))
    {
        while(rngIsEmpty(rngU1))
        {
            mv167Delay(300);

```

```

    }
  }
  status = rngBufGet(rngU1, (char *)&uInp[0], 8);

  if(rngIsEmpty(rngU2))
  {
    while(rngIsEmpty(rngU2))
    {
mv167Delay(300);
    }
  }
  status = rngBufGet(rngU2, (char *)&uInp[1], 8);

  if(rngIsEmpty(rngY1))
  {
    while(rngIsEmpty(rngY1))
    {
mv167Delay(300);
    }
  }
  status = rngBufGet(rngY1, (char *)&yMeas[0], 8);

  if(rngIsEmpty(rngY2))
  {
    while(rngIsEmpty(rngY2))
    {
mv167Delay(300);
    }
  }
  status = rngBufGet(rngY2, (char *)&yMeas[1], 8);

/* fill in the new i/o data */

for(j=0;j<=r-1;j++)
{
  uk1[j]=uInp[j];
}
for(j=0;j<=m-1;j++)
{
  y[j]=yMeas[j];
}

```

```

/* call ftf routine */

ftf(r,m,p,SizeX,uk1,u,y,Up,Yp,ThetaF,ThetaB,EsqF,error,EsqB,Gammak,GpR,
GpA,Gpk,GmP,Yh);

fprintf(fp,"%g %g %g %g %g %g\n", Yh[2], Yh[3], error[2],
error[3],Yh[0],Yh[1]);
/* prepare i/o matrices for next iteration */

    for(i=0;i<=(p-2);i++)
        {
            for(j=0;j<=r-1;j++)
                {
                    Up[j][i]=Up[j][i+1];
                }
        }

    for(j=0;j<=r-1;j++)
        {
            u[j]=uk1[j];
        }

    for(i=0;i<=(p-2);i++)
        {
            for(j=0;j<=m-1;j++)
                {
                    Yp[j][i]=Yp[j][i+1];
                }
        }
}

fclose(fp);

/* free the memory allocation for vectors */

free_vector(y,0,m-1);
free_vector(u,0,r-1);
free_vector(uk1,0,r-1);
free_dvector(error,0,mm-1);
free_dvector(GpR,0,SizeX-1);
free_dvector(GpA,0,mm-1);
free_dvector(Gpk,0,SizeX-1);

```

```
free_dvector(Gammak,0,1);

/* free the memory allocation for matrices */
free_matrix(utemp,0,r-1,0,n);
free_matrix(ytemp,0,m-1,0,n-1);
free_matrix(Up,0,r-1,0,p-1);
free_matrix(Yp,0,m-1,0,p-1);
free_matrix(uInit,0,r-1,0,init_pts-1);
free_matrix(yInit,0,m-1,0,init_pts-1);
free_dmatrix(ThetaF,0,rnm-1,0,SizeX-1);
free_dmatrix(ThetaB,0,rnm-1,0,SizeX-1);
free_dmatrix(EsqF,0,rnm-1,0,rnm-1);
free_dmatrix(EsqB,0,rnm-1,0,rnm-1);
}
```

```

/* ftfInit.c */

/* ----- */
/*
/* Module : ftfInit.c
/* Function : Performs initialization for FTF forward and backward time
/* recursion routines. Reads the input/out data and computes
/* forward and backward time OMPs, gain vector at k-1, gamma at
/* k-1 and the inverse of covariance matrix Pp at k-1, k-2
/* The vector and matrix memory allocations and matrix inversion
/* routines used here are from the book:
/* Numerical Recipes in C by Press, W.H., W.H., Teukolsky, S.A.,
/* Vetterling, W.T., and Flannery, B.P, Cambridge University
/* Press, 1983
/*
/* Rel      Date      Author      Comments
/* 0        8 Aug96   Mahesh Chowdhary  Initial Release
/* ----- */

```

```

#include <stdio.h>
#include <stdlib.h>
#include "nrutil.h"

```

```

#define N 100

```

```

float **unt, **ynt;
double *xip, **Xip1, **Xip2, **yn2, *yn1, **Yuk1, **temp1, **temp2,
        *numGp, denGp, temp, **Ppk, **temp3, **tPp;

```

```

void ftfInit(int r, int m, int p, int n, int SizeX, float **uInit, float **yInit,
            double **ThetaF, double **ThetaB, double **EsqF, double **EsqB,
            double *Gammak, double *Gpk)

```

```

/*

```

```

Variables:

```

```

r - number of inputs
m - number of outputs
p - model order

```

n - number of data points used for initialization
SizeX - $(r+m)*p + r$
uInit - input data matrix
yInit - measurement data matrix
ThetaF - Forward time OMPs estimated
ThetaB - Backward time OMPs estimated
EsqF - Forward time Equation estimation-error squares
EsqB - Backward time Equation estimation-error squares
Gammak - conversion factor
Gpk - gain vector that weighs the update for estimation

*/

```

{
  int i, j, l, rnm, k, il, ill_cond[1];
  denGp=1;rnm=r+m;

/* Memory allocation for vectors */

  yn1=dvector(0,rnm-1);
  xip=dvector(0,SizeX-1);
  numGp=dvector(0,SizeX-1);

/* Memory allocation for vectors */

  unt=matrix(0,r,0,n+1+p);
  ynt=matrix(0,m,0,n+1+p);
  Xip1=dmatrix(0,SizeX-1,0,N);
  Xip2=dmatrix(0,SizeX-1,0,N);
  yn2=dmatrix(0,rnm-1,0,N);
  Yuk1=dmatrix(0,rnm-1,0,N);
  tPp=dmatrix(1,SizeX,1,SizeX);
  Ppk=dmatrix(0,SizeX-1,0,SizeX-1);
  temp1=dmatrix(0,N,0,SizeX-1);
  temp2=dmatrix(0,rnm-1,0,N);
  temp3=dmatrix(0,SizeX-1,0,SizeX-1);

  for(j=0;j<=r-1;j++)
  {
    yn1[j]=uInit[j][0];
  }

```

```

for(i=0;i<=n;i++)
{
  i1=i+p;

  for(j=0;j<=m-1;j++)
  {
    ynt[j][i+p]=yInit[j][i];
  }
  for(j=0;j<=r-1;j++)
  {
    unt[j][i+p]=uInit[j][i];
  }
}

i1=0;

/* prepre I/O data matrix Xip and regressor xip */

for(k=p;k<=n+p;k++)
{
  for(i=0;i<=r-1;i++)
  {
    xip[i]=unt[i][k];
  }
  for(j=0;j<=p-1;j++)
  {
    for(i=0;i<=m-1;i++)
    {
      xip[(rnm*j)+r+i]=ynt[i][k-1-j];
    }
    for(i=0;i<=r-1;i++)
    {
      xip[(rnm*j)+rnm+i]=unt[i][k-1-j];
    }
  }
}

for(i=0;i<=SizeX-1;i++)
{
  Xip1[i][i1]=xip[i];
  Xip2[i][i1]=xip[i];
}
i1=i1+1;

```

```

    }
    for(i=0;i<=SizeX-1;i++)
    {
        Xip1[i][n]=xip[i];
    }

for(i=0;i<=n-1;i++)
{
    for(j=0;j<=m-1;j++)
    {
        yn2[r+j][i]=yInIt[j][i];
    }
    for(j=0;j<=r-1;j++)
    {
        yn2[j][i]=uInIt[j][i+1];
    }
}

for(i=0;i<=n-p-1;i++)
{
    for(j=0;j<=m-1;j++)
    {
        Yuk1[j][i+1+p]=yInIt[j][i];
    }
    for(j=0;j<=r-1;j++)
    {
        Yuk1[m+j][i+1+p]=uInIt[j][i];
    }
}

temp=0;

/* Prepare covariance matrix Pp */

for(i=0;i<=SizeX-1;i++)
{
    for(j=0;j<=SizeX-1;j++)
    {
        for(l=0;l<=n-1;l++)
        {
            temp=temp+Xip2[i][l]*Xip2[j][l];
        }
    }
}

```

```

    Ppk[i][j]=temp;
    tPp[i+1][j+1]=Ppk[i][j];
    temp=0;
  }
}

/* compute the inverse of covariance matrix */
/* The matrix inversion routine used here based on svdcmp
   routine from the book: Numerical Recipes in C by
   Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P,
   Cambridge University Press, 1983    */

matInv(tPp,SizeX,SizeX,ill_cond);

if(ill_cond[0]=0)
{
  printf("ERROR: ill conditioned matrix \n");
  exit(1);
}

for(i=1;i<=SizeX;i++)
{
  for(j=1;j<=SizeX;j++)
  {
    Ppk[i-1][j-1]=tPp[i][j];
  }
}

for(j=0;j<=SizeX-1;j++)
{
  for(i=0;i<=SizeX-1;i++)
  {
    numGp[j]=xip[i]*Ppk[i][j]+numGp[j];
  }
  denGp=denGp+xip[j]*numGp[j];
}

/* compute the gain vector Gp used to weigh the estimation update */

for(i=0;i<=SizeX-1;i++)
{
  Gpk[i]=numGp[i]/denGp;
}

```

```

}

/* compute the conversion factor gamma k */

Gammak[0]=1;
for(i=0;i<=SizeX-1;i++)
{
  Gammak[0]=Gammak[0]-Gpk[i]*xip[i];
}

temp=0;
for(i=0;i<=n-1;i++)
{
  for(j=0;j<=SizeX-1;j++)
  {
    for(l=0;l<=SizeX-1;l++)
    {
      temp=temp+Xip2[l][i]*Ppk[l][j];
    }
    temp1[i][j]=temp;
    temp=0;
  }
}
temp=0;

/* compute the forward time OMPs, ThetaF */

for(i=0;i<=mm-1;i++)
{
  for(j=0;j<=SizeX-1;j++)
  {
    for(l=0;l<=n-1;l++)
    {
      temp=temp+yn2[i][l]*temp1[l][j];
    }
    ThetaF[i][j]=temp;
    temp=0;
  }
}
temp=0;
for(i=0;i<=mm-1;i++)
{

```

```

    for(j=0;j<=n-1;j++)
    {
        for(l=0;l<=SizeX-1;l++)
        {
            temp=temp+ThetaF[i][l]*Xip2[l][j];
        }
        temp2[i][j] = yn2[i][j] - temp;
        temp=0;
    }
}
temp=0;

/* compute the Equation forward-time Error Squares */

for(i=0;i<=rnm-1;i++)
{
    for(j=0;j<=rnm-1;j++)
    {
        for(l=0;l<=n-1;l++)
        {
            temp=temp+temp2[j][l]*temp2[i][l];
        }
        EsqF[i][j]=temp+yn1[j]*yn1[i];
        temp=0;
    }
}
for(j=0;j<=SizeX-1;j++)
{
    for(i=0;i<=SizeX-1;i++)
    {
        temp3[j][i]=-1.0*xip[j]*Gpk[i];
    }
}
for(i=0;i<=SizeX-1;i++)
{
    temp3[i][i]=1+temp3[i][i];
}

/* Pp for backward time iteration */

temp=0;
for(i=0;i<=SizeX-1;i++)

```

```

{
  for(j=0;j<=SizeX-1;j++)
  {
    for(l=0;l<=SizeX-1;l++)
    {
      temp=temp+tPp[i+1][l+1]*temp3[l][j];
    }
    Ppk[i][j]=temp;
    temp=0;
  }
}
temp=0;
for(i=0;i<=n;i++)
{
  for(j=0;j<=SizeX-1;j++)
  {
    for(l=0;l<=SizeX-1;l++)
    {
      temp=temp+Xip1[l][i]*Ppk[l][j];
    }
    temp1[i][j]=temp;
    temp=0;
  }
}
temp=0;

/* compute the Backward time OMPs ThetaB*/

for(i=0;i<=nm-1;i++)
{
  for(j=0;j<=SizeX-1;j++)
  {
    for(l=0;l<=n;l++)
    {
      temp=temp+Yuk1[i][l]*temp1[l][j];
    }
    ThetaB[i][j]=temp;
    temp=0;
  }
}
temp=0;
for(i=0;i<=nm-1;i++)

```

```

{
  for(j=0;j<=n;j++)
  {
    for(l=0;l<=SizeX-1;l++)
    {
      temp=temp+ThetaB[i][l]*Xip1[l][j];
    }
    temp2[i][j]=Yuk1[i][j]-temp;
    temp=0;
  }
}
temp=0;

/* compute the backward time Equation error squares */

for(i=0;i<=rnm-1;i++)
{
  for(j=0;j<=rnm-1;j++)
  {
    for(l=0;l<=n;l++)
    {
      temp=temp+temp2[j][l]*temp2[i][l];
    }
    EsqB[i][j]=temp;
    temp=0;
  }
}

/* free the memory allocation for vectors */

free_dvector(xip,0,SizeX-1);
free_dvector(numGp,0,SizeX-1);
free_dvector(yn1,0,rnm-1);

/* free the memory allocation for matrices */

free_matrix(unt,0,r,0,n+1+p);
free_matrix(ynt,0,m,0,n+1+p);
free_dmatrix(Xip1,0,SizeX-1,0,N);
free_dmatrix(Xip2,0,SizeX-1,0,N);
free_dmatrix(yn2,0,rnm-1,0,N);
free_dmatrix(Yuk1,0,rnm-1,0,N);

```

```

free_dmatrix(tPp,1,SizeX,1,SizeX);
free_dmatrix(Ppk,0,SizeX-1,0,SizeX-1);
free_dmatrix(temp1,0,N,0,SizeX-1);
free_dmatrix(temp2,0,rnm-1,0,N);
free_dmatrix(temp3,0,SizeX-1,0,SizeX-1);
}

/* ftf.c */

/* ----- */
/*
/* Module : ftf.c
/* Function : Performs one recursion of FTF forward time estimation and
/*           calls the backward time recursion routine. Updates the
/*           Observer Markov Parameters (OMPs), Equation error squares,
/*           gain Gp, and conversion factor gamma k. This routine is based
/*           on formulation of Fast Transversal Filter presented in the book:
/*           Applied System Identification by Jer-Nan Juang, PTR Prentice
/*           Hall, 1994
/*           The vector and matrix memory allocations and matrix inversion
/*           routines used here are from the book:
/*           Numerical Recipes in C by Press, W.H., W.H., Teukolsky, S.A.,
/*           Vetterling, W.T., and Flannery, B.P, Cambridge University
/*           Press, 1983
/*
/* Rel      Date      Author      Comments
/* 0        8 Aug96   Mahesh Chowdhary  Initial Release
/* ----- */

#include<stdio.h>
#include<stdlib.h>
#include "nrutil.h"

void ftf(int r, int m, int p, int SizeX, float *uk1, float *u, float *y, float **Up,
float **Yp, double **ThetaF, double **ThetaB, double **EsqF,
double *error, double **EsqB, double *Gammak, double *GpR,
double *GpA, double *Gpk, double GmP[], double *Yh)

/*

```

Variables:

r - number of inputs
 m - number of outputs
 p - model order
 SizeX - $(r+m)*p + r$
 u - input time at k
 y - output time at k
 uk1 - input time at k+1
 Up - input time at k-1
 Yp - output time at k-1
 ThetaF - Forward time OMPs estimated
 ThetaB - Backward time OMPs estimated
 EsqF - Forward time Equation estimation error squares
 EsqB - Backward time Equation estimation error squares
 error - difference between estimated and measure output
 Gammak - conversion factor
 GpR - first $\langle 1 \times \text{SizeX} \rangle$ elements of the augmented gain vector
 GpA - remaining elements of the augmented gain vector
 GmP - update of conversion factor
 Gpk - gain vector
 Yh - estimated output at time k

```

    */
    {
    int i,j,k,l,rnm,ill_cond[1];
    double *Yf, *xip, *epriF, *epstF, *GainP, *Gta1, *Gtr1;
    double **EsqInvF;
    double temp;
    ill_cond[0]=0;

    rnm=r+m;

    /* Memory allocation for vectors */

    Yf=dvector(0,rnm-1);
    xip=dvector(0,SizeX-1);
    epriF=dvector(0,rnm-1);
    epstF=dvector(0,rnm-1);
    GainP=dvector(0,SizeX-1+rnm);
    Gta1=dvector(0,rnm-1);
    Gtr1=dvector(0,SizeX-1);
  
```

```

/* Memory allocation for matrices */

EsqInvF=dmatrix(1,mm,1,mm);

/* prepare the i/o data matrices */

for(i=0;i<=m-1;i++)
{
  Yf[r+i]=y[i];
}

for(i=0;i<=r-1;i++)
{
  xip[i]=u[i];
  Yf[i]=uk1[i];
}

/* form the input/output data matrices for forward time recursion */

for(j=0;j<=p-1;j++)
{
  for(i=0;i<=m-1;i++)
  {
    xip[(mm*j)+r+i]=Yp[i][p-1-j];
  }
}

for(j=0;j<=p-1;j++)
{
  for(i=0;i<=r-1;i++)
  {
    xip[(mm*j)+mm+i]=Up[i][p-1-j];
  }
}
temp=0;

/* Compute the estimated output, the a priori forward-time estimation error
and the a posteriori forward-time estimation error */

for(i=0;i<=mm-1;i++)

```

```

{
for(l=0;l<=SizeX-1;l++)
{
temp=temp+ThetaF[i][l]*xip[l];
Yh[i]=temp;
}
error[i]=epriF[i]=Yf[i]-temp;
temp=0;
epstF[i]=Gammak[0]*epriF[i];
Gtal[i]=0;
}

/* compute the forward time Equation error squares matrix */

for(i=0;i<=rnm-1;i++)
{
for(j=0;j<=rnm-1;j++)
{
EsqF[i][j]=EsqF[i][j]+epstF[i]*epriF[j];
}
}

/* Prepare EsqF for inversion */

for(i=1;i<=rnm;i++)
{
for(j=1;j<=rnm;j++)
{
EsqInvF[i][j]=EsqF[i][j];
}
}

/* Use the matrixc inversion routine from Numerical Recipes in C */

matInv(EsqInvF,rnm,rnm,ill_cond);

if(ill_cond[0]=0)
{
printf("ERROR: ill conditioned matrix \n");
exit(1);
}

```

```

/* compute the forward time OMPs */

for(i=0;i<=rnm-1;i++)
{
  for(j=0;j<=SizeX-1;j++)
  {
    ThetaF[i][j]=ThetaF[i][j]+epriF[i]*Gpk[j];
  }
}

/* compute the updated gain matrix */

GmP[0]=Gammak[0];

for(i=0;i<=rnm-1;i++)
{
  for(j=0;j<=rnm-1;j++)
  {
    Gta1[i]=epstF[j]*EsqInvF[j+1][i+1]+Gta1[i];
  }
  GainP[i]=Gta1[i];
  GmP[0]=GmP[0]-Gta1[i]*epstF[i];
}
temp=0;

/* partition of the gain matrix */

for(i=0;i<=SizeX-1;i++)
{
  for(j=0;j<=rnm-1;j++)
  {
    temp=temp+Gta1[j]*ThetaF[j][i];
  }
  Gtr1[i]=Gpk[i]-temp;
  GainP[rnm+i]=Gtr1[i];
  GpR[i]=GainP[i];
  temp=0;
}

for(i=0;i<=rnm-1;i++)
{

```

```

    GpA[i]=GainP[SizeX+i];
}

/* call the backward time FTF recursion */

ftfB(r,m,p,SizeX,uk1,u,y,Up,Yp,ThetaB,GpA,GpR,Gpk,Gammak,GmP,EsqB);

/* free memory from vectors */

free_dvector(Yf,0,rnm-1);
free_dvector(xip,0,SizeX-1);
free_dvector(epriF,0,rnm-1);
free_dvector(epstF,0,rnm-1);
free_dvector(GainP,0,SizeX-1+rnm);
free_dvector(Gta1,0,rnm-1);
free_dvector(Gtr1,0,SizeX-1);

/* free memory from matrices */
free_dmatrix(EsqInvF,1,rnm,1,rnm);
}

/* ftfB.c */

/* ----- */
/* Module : ftfB.c */
/* Function : Performs one recursion of FTF backward time estimation */
/*           Updates the backward time OMPs, Equation error squares, gain */
/*           Gp, and the conversion factor gamma k. This routine is based on */
/*           formulation of Fast Transversal Filter presented in the book: */
/*           Applied System Identification by Jer-Nan Juang, PTR Prentice */
/*           Hall, 1994 */
/*           The vector and matrix memory allocations routines used here */
/*           are from the book: */
/*           Numerical Recipes in C by Press, W.H., W.H., Teukolsky, S.A., */
/*           Vetterling, W.T., and Flannery, B.P, Cambridge University */
/*           Press, 1983 */
/* ----- */
/* Rel   Date       Author           Comments */
/* 0     8 Aug96    Mahesh Chowdhary  Initial Release */
/* ----- */

```

```

#include <stdio.h>
#include <stdlib.h>
#include "nrutil.h"

void fitB(int r, int m, int p, int SizeX, float *uk1, float *u, float *y, float **Up,
          float **Yp, double **ThetaB, double *GpA, double *GpR,
          double *GpK, double *Gammak, double Gmp[], double **EsqB)

/*
Variables:
r      - number of inputs
m      - number of outputs
p      - model order
SizeX  - (r+m)*p + r
u      - input time at k
y      - output time at k
uk1    - input time at k+1
Up     - input time at k-1
Yp     - output time at k-1
ThetaB - Backward time OMPs estimated
EsqB   - Backward time Equation estimation error squares
error  - difference between estimated and measured output
Gammak - conversion factor
GpR    - first <1 x SizeX> elements of the augmented gain vector
GpA    - remaining elements of the augmented gain vector
GmP    - update of conversion factor
Gpk    - gain vector

*/

{
int i,j,l,rmm;
double *Yb, *xip,*epriB,*epstB;
double temp1, temp2;
rmm=r+m;

/* Memory allocation for temporary vectors and matrices */

Yb=dvector(0,rmm-1);
xip=dvector(0,SizeX-1);
epriB=dvector(0,rmm-1);

```

```

epstB=dvector(0,mm-1);

/* prepare i/o data matrices */

for(i=0;i<=m;i++)
{
  Yb[i]=Yp[i][0];
}

for(i=0;i<=r;i++)
{
  xip[i]=uk1[i];
  Yb[m+i]=Up[i][0];
}

for(j=0;j<=m;j++)
{
  for(i=0;i<=p-2;i++)
  {
    Yp[j][i]=Yp[j][i+1];
  }
  Yp[j][p-1]=y[j];
}

for(j=0;j<=r;j++)
{
  for(i=0;i<=p-2;i++)
  {
    Up[j][i]=Up[j][i+1];
  }
  Up[j][p-1]=u[j];
}

/* form the input/output data matrix for backward time recursion */

for(j=0;j<=p-1;j++)
{
  for(i=0;i<=m;i++)
  {
    xip[(mm*j)+r-1+i]=Yp[i][p-1-j];
  }
}

```

```

for(j=0;j<=p-1;j++)
{
for(i=0;i<=r;i++)
{
xip[(rnm*j)+rnm+i]=Up[i][p-1-j];
}
}
temp1=0;

/* compute the backward time a priori and a posteriori estimation errors */

temp2=1;

for(i=0;i<=rnm-1;i++)
{
for(l=0;l<=SizeX-1;l++)
{
temp1=temp1+ThetaB[i][l]*xip[l];
}
epriB[i]=Yb[i]-temp1;
temp2=temp2-GpA[i]*epriB[i];
temp1=0;
}
temp1=0;

/* compute the updated gain matrix */

for(i=0;i<=SizeX-1;i++)
{
for(j=0;j<=rnm-1;j++)
{
temp1=temp1+GpA[j]*ThetaB[j][i];
}
GpK[i]=(GpR[i]+temp1)/temp2;
temp1=0;
}

/* compute the backward time OMPs ThetaB */

for(i=0;i<=rnm-1;i++)
{

```

```

for(j=0;j<=SizeX-1;j++)
{
  ThetaB[i][j]=ThetaB[i][j]+epriB[i]*GpK[j];
}
}
Gammak[0]=Gmp[0]/temp2;

/* compute the backward time Equation error equares */

for(i=0;i<=rnm-1;i++)
{
  epstB[i]=Gammak[0]*epriB[i];
  for(j=0;j<=rnm-1;j++)
  {
    EsqB[i][j]=EsqB[i][j]+epstB[i]*epriB[j];
  }
}

free_dvector(Yb,0,rnm-1);
free_dvector(xip,0,SizeX-1);
free_dvector(epriB,0,rnm-1);
free_dvector(epstB,0,rnm-1);
}

```

```

/* ----- */
/* Module: mv167Clk.c */
/* Function: Provides a variable length delay for mv167 */
/*           code. Uses the auxillary clock to delay the */
/*           required time period, and then returns the */
/*           function mv167Delay(). The delay can be set */
/*           between 1 second and 200 usec. */
/* ----- */
/* Rel   Date       Authors           Comments. */
/* 0     8 Sep 94   D.Barker, M.Chowdhary  First Release. */
/* ----- */

/* include files */
#include <vxWorks.h>
#include <semLib.h>

SEM_ID mv167DelaySem;

void mv167DelayInit();
void mv167Delay();
void mv167DelayISR();

/* ----- */
/* External Procedure: mv167DelayInit() */
/* Function: Initialises the delay system by creating the */
/*           semaphore needed for delay pending. Connects */
/*           the mv167 Aux Clk to the mv167DelayISR. */
/* ----- */
void mv167DelayInit()
{
    mv167DelaySem = semBCreate(0,SEM_EMPTY);
    sysAuxClkConnect(mv167DelayISR);
}

/* ----- */
/* External Procedure: mv167Delay() */
/* Function: Delays for set period of time. Time period in */
/*           the range 333333 to 200 micro seconds are */
/*           allowed. */
/* ----- */

```

```

void mv167Delay(delay)
long delay;
{
int result;
long rate;

    if(delay>333333 || delay<200)
    {
        printf("mv167Delay: Delay out of bounds.\n");
/*      exit(); */
    }

    rate=1000000/delay;
    sysAuxClkRateSet (rate);
    sysAuxClkEnable();
    /* pend this task on taking the semaphore */
    result = semTake(mv167DelaySem, WAIT_FOREVER);
    sysAuxClkDisable();
}

/* ----- */
/*
/* Internal Procedure: mv167DelayISR()
/*
/* Function: Interrupt Service Routine for aux clock
/*      interrupts. Gives the mv167DelaySem semaphore.
/*
/* ----- */
void mv167DelayISR()
{
    semGive(mv167DelaySem);
}

```

```
/* nrutil.h */
```

```
/*
```

This header file declares the data types of the memory allocation routines used in the programs.

This header file was obtained from the book:

Numerical Recipes in C

by, W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery
Cambridge University Press, 1983

```
*/
```

```
#ifndef _NR_UTILS_H_
```

```
#define _NR_UTILS_H_
```

```
static double sqrg;
```

```
#define SQR(a) ((sqrg=(a)) == 0.0 ? 0.0 : sqrg*sqrg)
```

```
static double dsqrg;
```

```
#define DSQR(a) ((dsqrg=(a)) == 0.0 ? 0.0 : dsqrg*dsqrg)
```

```
static double dmaxarg1,dmaxarg2;
```

```
#define DMAX(a,b) (dmaxarg1=(a),dmaxarg2=(b),(dmaxarg1) > (dmaxarg2) ?\
    (dmaxarg1) : (dmaxarg2))
```

```
static double dminarg1,dminarg2;
```

```
#define DMIN(a,b) (dminarg1=(a),dminarg2=(b),(dminarg1) < (dminarg2) ?\
    (dminarg1) : (dminarg2))
```

```
static double maxarg1,maxarg2;
```

```
#define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ?\
    (maxarg1) : (maxarg2))
```

```
static double minarg1,minarg2;
```

```
#define FMIN(a,b) (minarg1=(a),minarg2=(b),(minarg1) < (minarg2) ?\
    (minarg1) : (minarg2))
```

```
static long lmaxarg1,lmaxarg2;
```

```
#define LMAX(a,b) (lmaxarg1=(a),lmaxarg2=(b),(lmaxarg1) > (lmaxarg2) ?\
    (lmaxarg1) : (lmaxarg2))
```

```
static long lminarg1,lminarg2;
```

```

#define LMIN(a,b) (lminarg1=(a),lminarg2=(b),(lminarg1) < (lminarg2) ?\
    (lminarg1) : (lminarg2))

static int imaxarg1,imaxarg2;
#define IMAX(a,b) (imaxarg1=(a),imaxarg2=(b),(imaxarg1) > (imaxarg2) ?\
    (imaxarg1) : (imaxarg2))

static int iminarg1,iminarg2;
#define IMIN(a,b) (iminarg1=(a),iminarg2=(b),(iminarg1) < (iminarg2) ?\
    (iminarg1) : (iminarg2))

#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

#if defined(__STDC__) || defined(ANSI) || defined(NRANSI) /* ANSI */

void nrerror(char error_text[]);
double *vector(long nl, long nh);
int *ivector(long nl, long nh);
unsigned char *cvector(long nl, long nh);
unsigned long *lvector(long nl, long nh);
double *dvector(long nl, long nh);
double **matrix(long nrl, long nrh, long ncl, long nch);
double **dmatrix(long nrl, long nrh, long ncl, long nch);
int **imatrix(long nrl, long nrh, long ncl, long nch);
double **submatrix(double **a, long oldrl, long oldrh, long oldcl, long oldch,
    long newrl, long newcl);
double **convert_matrix(double *a, long nrl, long nrh, long ncl, long nch);
double ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh);
void free_vector(double *v, long nl, long nh);
void free_ivector(int *v, long nl, long nh);
void free_cvector(unsigned char *v, long nl, long nh);
void free_lvector(unsigned long *v, long nl, long nh);
void free_dvector(double *v, long nl, long nh);
void free_matrix(double **m, long nrl, long nrh, long ncl, long nch);
void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch);
void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch);
void free_submatrix(double **b, long nrl, long nrh, long ncl, long nch);
void free_convert_matrix(double **b, long nrl, long nrh, long ncl, long nch);
void free_f3tensor(double ***t, long nrl, long nrh, long ncl, long nch,
    long ndl, long ndh);

#else /* ANSI */

```

```
/* traditional - K&R */

void nerror();
double *vector();
double **matrix();
double **submatrix();
double **convert_matrix();
double ***f3tensor();
double *dvector();
double **dmatrix();
int *ivector();
int **imatrix();
unsigned char *cvector();
unsigned long *lvector();
void free_vector();
void free_dvector();
void free_ivector();
void free_cvector();
void free_lvector();
void free_matrix();
void free_submatrix();
void free_convert_matrix();
void free_dmatrix();
void free_imatrix();
void free_f3tensor();

#endif /* ANSI */

#endif /* _NR_UTILS_H_ */
```

```

/* nutil.c */

/*
This file describes all the memory allocation routines used in various routines
for FTF implementation.

This functions were obtained from the book:
Numerical Recipes in C
by: W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery.
Cambridge University Press, 1983
*/

#if defined(__STDC__) || defined(ANSI) || defined(NRANSI) /* ANSI */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(char error_text[])
/* Numerical Recipes standard error handler */
{
    fprintf(stderr,"run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

double *vector(long nl, long nh)
/* allocate a double vector with subscript range v[nl..nh] */
{
    double *v;

    v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl+NR_END;
}

int *ivector(long nl, long nh)
/* allocate an int vector with subscript range v[nl..nh] */
{

```

```

int *v;

v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
if (!v) perror("allocation failure in ivector()");
return v-nl+NR_END;
}

unsigned char *cvector(long nl, long nh)
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
    unsigned char *v;

    v=(unsigned char *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(unsigned
char)));
    if (!v) perror("allocation failure in cvector()");
    return v-nl+NR_END;
}

unsigned long *lvector(long nl, long nh)
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
    unsigned long *v;

    v=(unsigned long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
    if (!v) perror("allocation failure in lvector()");
    return v-nl+NR_END;
}

double *dvector(long nl, long nh)
/* allocate a double vector with subscript range v[nl..nh] */
{
    double *v;

    v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
    if (!v) perror("allocation failure in dvector()");
    return v-nl+NR_END;
}

double **matrix(long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1, ncol=nch-ncl+1;

```

```

double **m;

/* allocate pointers to rows */
m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
if (!m) nrerror("allocation failure 1 in matrix()");
m += NR_END;
m -= nrl;

/* allocate rows and set pointers to them */
m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
m[nrl] += NR_END;
m[nrl] -= ncl;

for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

/* return pointer to array of pointers to rows */
return m;
}

double **dmatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
double **m;

/* allocate pointers to rows */
m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
if (!m) nrerror("allocation failure 1 in matrix()");
m += NR_END;
m -= nrl;

/* allocate rows and set pointers to them */
m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
m[nrl] += NR_END;
m[nrl] -= ncl;

for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

/* return pointer to array of pointers to rows */
return m;
}

```

```

}

int **imatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1, ncol=nch-ncl+1;
    int **m;

    /* allocate pointers to rows */
    m=(int **) malloc((size_t)((nrow+NR_END)*sizeof(int*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    /* allocate rows and set pointers to them */
    m[nrl]=(int *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}

double **submatrix(double **a, long oldrl, long oldrh, long oldcl, long oldch,
    long newrl, long newcl)
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
    long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
    double **m;

    /* allocate array of pointers to rows */
    m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
    if (!m) nrerror("allocation failure in submatrix()");
    m += NR_END;
    m -= newrl;

    /* set pointers to rows */
    for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

```

```

/* return pointer to array of pointers to rows */
return m;
}

double **convert_matrix(double *a, long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
    long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
    double **m;

    /* allocate pointers to rows */
    m=(double **) malloc((size_t) ((nrow+NR_END)*sizeof(double*)));
    if (!m) nrerror("allocation failure in convert_matrix()");
    m += NR_END;
    m -= nrl;

    /* set pointers to rows */
    m[nrl]=a-ncl;
    for(i=1,j=nrl+1;i<nrow;i++j++) m[j]=m[j-1]+ncol;
    /* return pointer to array of pointers to rows */
    return m;
}

double ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh)
/* allocate a double 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh] */
{
    long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
    double ***t;

    /* allocate pointers to pointers to rows */
    t=(double ***) malloc((size_t)((nrow+NR_END)*sizeof(double**)));
    if (!t) nrerror("allocation failure 1 in f3tensor()");
    t += NR_END;
    t -= nrl;

    /* allocate pointers to rows and set pointers to them */
    t[nrl]=(double **) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double*)));
    if (!t[nrl]) nrerror("allocation failure 2 in f3tensor()");

```

```

t[nrl] += NR_END;
t[nrl] -= ncl;

/* allocate rows and set pointers to them */
t[nrl][ncl]=(double *)
malloc((size_t)((nrow*ncol*ndep+NR_END)*sizeof(double));
if (!t[nrl][ncl]) nrerror("allocation failure 3 in f3tensor()");
t[nrl][ncl] += NR_END;
t[nrl][ncl] -= ndl;

for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
for(i=nrl+1;i<=nrh;i++) {
    t[i]=t[i-1]+ncol;
    t[i][ncl]=t[i-1][ncl]+ncol*ndep;
    for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
}

/* return pointer to array of pointers to rows */
return t;
}

void free_vector(double *v, long nl, long nh)
/* free a double vector allocated with vector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_ivec(int *v, long nl, long nh)
/* free an int vector allocated with ivec() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_cvec(unsigned char *v, long nl, long nh)
/* free an unsigned char vector allocated with cvec() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_lvec(unsigned long *v, long nl, long nh)
/* free an unsigned long vector allocated with lvec() */
{

```

```

    free((FREE_ARG) (v+n1-NR_END));
}

void free_dvector(double *v, long n1, long nh)
/* free a double vector allocated with dvector() */
{
    free((FREE_ARG) (v+n1-NR_END));
}

void free_matrix(double **m, long n1, long nrh, long ncl, long nch)
/* free a double matrix allocated by matrix() */
{
    free((FREE_ARG) (m[n1]+ncl-NR_END));
    free((FREE_ARG) (m+n1-NR_END));
}

void free_dmatrix(double **m, long n1, long nrh, long ncl, long nch)
/* free a double matrix allocated by dmatrix() */
{
    free((FREE_ARG) (m[n1]+ncl-NR_END));
    free((FREE_ARG) (m+n1-NR_END));
}

void free_imatrix(int **m, long n1, long nrh, long ncl, long nch)
/* free an int matrix allocated by imatrix() */
{
    free((FREE_ARG) (m[n1]+ncl-NR_END));
    free((FREE_ARG) (m+n1-NR_END));
}

void free_submatrix(double **b, long n1, long nrh, long ncl, long nch)
/* free a submatrix allocated by submatrix() */
{
    free((FREE_ARG) (b+n1-NR_END));
}

void free_convert_matrix(double **b, long n1, long nrh, long ncl, long nch)
/* free a matrix allocated by convert_matrix() */
{
    free((FREE_ARG) (b+n1-NR_END));
}

```

```

void free_f3tensor(double ***t, long nrl, long nrh, long ncl, long nch,
    long ndl, long ndh)
/* free a double f3tensor allocated by f3tensor() */
{
    free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
    free((FREE_ARG) (t[nrl]+ncl-NR_END));
    free((FREE_ARG) (t+nrl-NR_END));
}

#else /* ANSI */
/* traditional - K&R */

#include <stdio.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(error_text)
char error_text[];
/* Numerical Recipes standard error handler */
{
    void exit();

    fprintf(stderr,"run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

double *vector(nl,nh)
long nh,nl;
/* allocate a double vector with subscript range v[nl..nh] */
{
    double *v;

    v=(double *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(double)));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl+NR_END;
}

int *ivector(nl,nh)
long nh,nl;
/* allocate an int vector with subscript range v[nl..nh] */

```

```

{
  int *v;

  v=(int *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(int)));
  if (!v) perror("allocation failure in ivector()");
  return v-nl+NR_END;
}

unsigned char *cvector(nl,nh)
long nh,nl;
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
  unsigned char *v;

  v=(unsigned char *)malloc((unsigned int)
((nh-nl+1+NR_END)*sizeof(unsigned char)));
  if (!v) perror("allocation failure in cvector()");
  return v-nl+NR_END;
}

unsigned long *lvector(nl,nh)
long nh,nl;
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
  unsigned long *v;

  v=(unsigned long *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(long)));
  if (!v) perror("allocation failure in lvector()");
  return v-nl+NR_END;
}

double *dvector(nl,nh)
long nh,nl;
/* allocate a double vector with subscript range v[nl..nh] */
{
  double *v;

  v=(double *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(double)));
  if (!v) perror("allocation failure in dvector()");
  return v-nl+NR_END;
}

```

```

double **matrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
  long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
  double **m;

  /* allocate pointers to rows */
  m=(double **) malloc((unsigned int)((nrow+NR_END)*sizeof(double*)));
  if (!m) nrerror("allocation failure 1 in matrix()");
  m += NR_END;
  m -= nrl;

  /* allocate rows and set pointers to them */
  m[nrl]=(double *)
  malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(double)));
  if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
  m[nrl] += NR_END;
  m[nrl] -= ncl;

  for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

  /* return pointer to array of pointers to rows */
  return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
  long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
  double **m;

  /* allocate pointers to rows */
  m=(double **) malloc((unsigned int)((nrow+NR_END)*sizeof(double*)));
  if (!m) nrerror("allocation failure 1 in matrix()");
  m += NR_END;
  m -= nrl;

  /* allocate rows and set pointers to them */
  m[nrl]=(double *)
  malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(double)));

```

```

if (!m[nrl]) perror("allocation failure 2 in matrix()");
m[nrl] += NR_END;
m[nrl] -= ncl;

for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

/* return pointer to array of pointers to rows */
return m;
}

int **imatix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    int **m;

    /* allocate pointers to rows */
    m=(int **) malloc((unsigned int)((nrow+NR_END)*sizeof(int*)));
    if (!m) perror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    /* allocate rows and set pointers to them */
    m[nrl]=(int *) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(int)));
    if (!m[nrl]) perror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}

double **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
double **a;
long newcl,newrl,oldch,oldcl,oldrh,oldrl;
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
    long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;

```

```

double **m;

/* allocate array of pointers to rows */
m=(double **) malloc((unsigned int) ((nrow+NR_END)*sizeof(double*)));
if (!m) nrerror("allocation failure in submatrix()");
m += NR_END;
m -= newrl;

/* set pointers to rows */
for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

/* return pointer to array of pointers to rows */
return m;
}

double **convert_matrix(a,nrl,nrh,ncl,nch)
double *a;
long nch,ncl,nrh,nrl;
/* allocate a double matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
double **m;

/* allocate pointers to rows */
m=(double **) malloc((unsigned int) ((nrow+NR_END)*sizeof(double*)));
if (!m) nrerror("allocation failure in convert_matrix()");
m += NR_END;
m -= nrl;

/* set pointers to rows */
m[nrl]=a-ncl;
for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;
/* return pointer to array of pointers to rows */
return m;
}

double ***f3tensor(nrl,nrh,ncl,nch,ndl,ndh)
long nch,ncl,ndh,ndl,nrh,nrl;
/* allocate a double 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh] */

```

```

{
  long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
  double ***t;

  /* allocate pointers to pointers to rows */
  t=(double ***) malloc((unsigned int)((nrow+NR_END)*sizeof(double**)));
  if (!t) nrerror("allocation failure 1 in f3tensor()");
  t += NR_END;
  t -= nrl;

  /* allocate pointers to rows and set pointers to them */
  t[nrl]=(double **)malloc((unsigned int)
((nrow*ncol+NR_END)* sizeof(double*)));

  if (!t[nrl]) nrerror("allocation failure 2 in f3tensor()");
  t[nrl] += NR_END;
  t[nrl] -= ncl;

  /* allocate rows and set pointers to them */
  t[nrl][ncl]=(double *)
malloc((unsigned int)((nrow*ncol*ndep+NR_END)*sizeof(double)));
  if (!t[nrl][ncl]) nrerror("allocation failure 3 in f3tensor()");
  t[nrl][ncl] += NR_END;
  t[nrl][ncl] -= ndl;

  for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
  for(i=nrl+1;i<=nrh;i++) {
    t[i]=t[i-1]+ncol;
    t[i][ncl]=t[i-1][ncl]+ncol*ndep;
    for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
  }

  /* return pointer to array of pointers to rows */
  return t;
}

void free_vector(v,nl,nh)
double *v;
long nh,nl;
/* free a double vector allocated with vector() */
{
  free((FREE_ARG) (v+nl-NR_END));
}

```

```

}

void free_ivector(v,nl,nh)
int *v;
long nh,nl;
/* free an int vector allocated with ivector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_cvector(v,nl,nh)
long nh,nl;
unsigned char *v;
/* free an unsigned char vector allocated with cvector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_lvector(v,nl,nh)
long nh,nl;
unsigned long *v;
/* free an unsigned long vector allocated with lvector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_dvector(v,nl,nh)
double *v;
long nh,nl;
/* free a double vector allocated with dvector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_matrix(m,nrl,nrh,ncl,nch)
double **m;
long nch,ncl,nrh,nrl;
/* free a double matrix allocated by matrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

```

```

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
long nch,ncl,nrh,nrl;
/* free a double matrix allocated by dmatrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
long nch,ncl,nrh,nrl;
/* free an int matrix allocated by imatrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

void free_submatrix(b,nrl,nrh,ncl,nch)
double **b;
long nch,ncl,nrh,nrl;
/* free a submatrix allocated by submatrix() */
{
    free((FREE_ARG) (b+nrl-NR_END));
}

void free_convert_matrix(b,nrl,nrh,ncl,nch)
double **b;
long nch,ncl,nrh,nrl;
/* free a matrix allocated by convert_matrix() */
{
    free((FREE_ARG) (b+nrl-NR_END));
}

void free_f3tensor(t,nrl,nrh,ncl,nch,ndl,ndh)
double ***t;
long nch,ncl,ndh,ndl,nrh,nrl;
/* free a double f3tensor allocated by f3tensor() */
{
    free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
    free((FREE_ARG) (t[nrl]+ncl-NR_END));
}

```

```
    free((FREE_ARG) (t+nrl-NR_END));  
}  
  
#endif /* ANSI */
```

NOTES

1. Sorenson, H.W., Least-squares estimation: from Gauss to Kalman, IEEE Spectrum, vol. 7, pp. 63-68, July 1970
2. Weiner, N, Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications, New York Technical Press and Wiley, 1949
3. Kailath, T., A view of three decades of linear filtering theory, IEEE Transactions of Information Theory, vol IT-20, pp. 146-181, 1974
4. Van Trees, H.L., Detection, Estimation, and Modulation Theory, Part I, IEEE Transactions of Information Theory, vol. IT-14, pp. 612-613, 1968
5. Van Trees, H.L., Detection, Estimation, and Modulation Theory, Part II, IEEE Transactions of Information Theory, vol. IT-18, pp. 450-451, 1972
6. Stiffler, J. J., Theory of Synchronous Communication, IEEE Transactions of Information Theory (book rev.), vol. IT-18, pp. 218-219, 1972
7. Lindsey, W.C., Synchronization Systems in Communication and Control, Englewood Cliffs, NJ, Prentice-Hall, 1972
8. Kalman, R. E., A New Approach to Linear Filtering and Prediction Problems, Journal of Basic Engineering, vol. 82 D, pp. 34-45, 1960
9. Kalman, R. E., On the General Theory of Control Systems, Proceedings of First International Congress, IFAC, Moscow, USSR, pp. 481-492, 1960
10. Kalman, R. E. and Bucy, R. S., New results in linear filtering and prediction theory, Transactions of ASME Journal of Basic Engineering, vol. 83, pp. 95-108, 1961
11. Gilbert, E.G., Controllability and observability in multivariable control systems," SIAM Journal on Control, vol. 1, no.2, pp. 128-151, 1963

12. Kalman, R. E., Mathematical description of linear dynamical systems,” SIAM Journal on Control, vol. 1, no.2, 152-192, 1963
13. Sorenson, H. W., Kalman Filter: Theory and Application, IEEE, Inc., New York, 1985
14. Schlee, F.H., Divergence in Kalman Filter, AIAA Journal, vol 5, pp.1114-1120, 1976
15. Fitzgerald, Divergence of Kalman Filter, IEEE Transactions on Automatic Controls, Dec. 1991
16. Sangsuk-Iam, Bullock, T.E., Analysis of continuous time Kalman filter under incorrect noise covariances, Automatica, vol. 24, No.5, pp. 659-669, 1988
17. Sangsuk-Iam, Bullock, T. E., Behavior of the discrete time Kalman filter under incorrect noise covariances, Proceedings 26th IEEE Conference on Decision and Control, Los Angeles, CA
18. Lee, T.T., A direct approach to identify the noise covariances for Kalman filtering, IEEE Transactions on Automatic Control, AC-25, pp. 841-842, 1980
19. Belanger, P. R., Estimation of noise covariances matrices for a linear time varying stochastic system, Automatica, 10, pp. 267-275
20. Sangsuk-Iam, Bullock, T. E., Direct estimation of noise covariances, Proceedings. of ACC Conference., pp. 1289-1294, Atlanta, GA 1988
21. Carew, B., Belanger, P. R., Identification of optimal filter steady-state gain for systems with unknown noise covariances, IEEE Transactions on Automatic Control, AC-18, No.6, pp. 582-587, 1973
22. Tajima, K., Estimation of steady state Kalman filter gain, IEEE Transactions on Automatic Control, AC-23, pp. 944-945, 1978
23. Goodwin, G.C., Sin, K. S., Adaptive filtering, prediction, and control, Prentice-Hall, Englewood Cliffs, NJ 07632, 1984
24. Haykin, S., Adaptive filter theory, Prentice-Hall, Englewood Cliffs, NJ, 07632, 1986

25. Mehra, R.K., Approach to adaptive filtering, IEEE Transactions on Automatic Control, AC-17, pp. 693-698, 1972
26. Mehra, R.K., On the identification of variances and adaptive Kalman filtering, IEEE Transactions on Automatic Control, AC-15, pp. 175-184, 1970
27. Astrom, K. J., Why use adaptive technique for steering large tankers?, International Journal of Control, vol. 32, pp. 689-708, 1979
28. van Amerongen, J., A model reference adaptive auto-pilot for ships -practical results, Proceedings of 8th IFAC World Congress, Kyoto, Japan 1981
29. Landau, I. D., Adaptive control - The model reference approach, Dekker, New York, 1979
30. Gupta, P. C., Yamada, K., Adaptive short-time forecasting of hourly load using weather information, IEEE Transactions on Power Apparatus and Systems, vol. PAS-91, pp. 2085-2095
31. Lucky, R., Automatic equalization for digital communication," Bell Systems and Technology Journal, vol. 44, pp. 547-588, 1965
32. Goddard, D., Channel equalization using a Kalman filter for fast data transmission, IBM Journal of Research and Development, vol. 18, pp. 267-273, 1974
33. Juang, J.-N., Pappa, R. S., An Eigensystem Realization Algorithm (ERA) for modal parameter identification and model reduction, Journal of Guidance, Control, and Dynamics, vol. 9, no. 3, pp. 294-303, 1986
34. Juang, J.-N., Mathematical correlation of modal parameter identification methods via system realization theory, International Journal of Analytical and Experimental Modal Analysis, vol. 2, no. 1, pp. 1-18, 1987
35. Powers, T., Doolittle, L., Ursic, R., and Wagner, J., Design, Commissioning and Operational Results of Wide Dynamic Range BPM Switched Electrode Electronics, Proceedings of Beam Instrumentation Workshop, Advanced Photon Source, IL, May 96

36. Hofler, A.S. et al, Performance of CEBAF Arc Beam Position Monitors, Proceedings of the Particle Accelerator Conference, pp. 2298-2300, Washington DC, 1993
37. Kuo, B. C., Digital Control Systems, 2nd Edition, HBJ College Publisher, Orlando, FL, 1992
38. Chowdhary, M, Krafft, G.A., Shoaee, H., and Watson, III, W.A., A Fast Feedback System for CEBAF, Proceedings of 1995 International Conference on Accelerator and Large Experimental Physics Control Systems, pp-Th4B-e, Chicago, IL, 1995
39. VxWorks Programmer's Guide 5.1, Wind River Systems, 101 Atlantic Avenue, Alameda, CA, 94501, 1993
40. Simrock, S.N., The RF Control System for CEBAF, Proceedings of 1990 US Particle Accelerator Conference, San Francisco, 1990
41. Legg, R., Chowdhary, M., Karn, J., Merz, W., Fast feedback corrector system, TJNAF Tech Note 96-005, Newport News, VA, 1996
42. Shinnars, S.M., Control System Design, John Wiley & Sons, NY, 1964
43. Goulab, G.H. and Van Voan, C.F., Matrix Computations, The Johns Hopkins University Press, Baltimore, Maryland, 1983
44. Juang, J-N, Applied System Identification, PTR Prentice Hall, Englewoods Cliff, New Jersey, 1994
45. Ljung, L. and Soderstrom, T., Theory and Practice of Recursive Identification, The MIT Press, Cambridge, Massachusetts, 1987
46. Cioffi, J.M. and Kailath, T., Fast Recursive-Least Squares Transversal Filter for Adaptive Filtering, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. ASSP-32, pp. 304-337
47. Lee, D.T, and Morf, M., Recursive Least-Squares Ladder Estimation Algorithm, IEEE Transactions on Circuits and System, vol. CAS-28, pp. 467-481

48. Watson, III, W.A., The CEBAF Control System, Proceedings of 1995 US Particle Accelerator Conference, Dallas, TX, pp. 2167-2171, 1995
49. Delesio, L.R. et al, The EPICS Architecture, Proceedings of 1991 International Conference on Accelerator and Large Experimental Physics Control Systems, pp. 278, 1991
50. Operators and Maintenance Manual for GP10-S Analog Computer, Comdyna, Inc. , 305 Devonshire Road, Barrington, Illinois,60010, 1996
51. User Manual for IP-16 ADC, Greensprings Computers, 1204 O'Brien Drive, Menlo Park, California, 94025, 1994
52. User Manual for VMESC5, SYSTRAN Corporation, 4126 Linden Avenue, Dayton, Ohio, 45432, 1995
53. User Manual for DVME-628, Datel, Inc., 11 Cabot Boulevard, Mansfield, Massachusetts, 02048, 1995

BIBLIOGRAPHY

1. B. Friedland, Control System Design: An Introduction to State-Space Methods, McGraw-Hill Book Company, New York, 1986
2. J-N Juang, Applied System Identification, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1994
3. B. C. Kuo, Digital Control Systems, 2nd Edition, Harcourt Brace Jovanovich, Publishers, Orlando, Florida, 1992
4. L. Ljung and T. Soderstrom, Theory and Practice of Recursive Identification, The MIT Press, Cambridge, Massachusetts, 1987
5. J. Van De Veget, Feedback Control Systems, 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1990
6. A. Papoulis, Probability, Random Variables, and Stochastic Process, John Wiley & Soncs, New York ,1965
7. W.H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, Numerical Recipes in C, Cambridge University Press, 1988

VITA

Mahesh Chowdhary

Born in India on May 26, 1966. Obtained a Bachelor of Engineering (Mechanical) degree from Regional Engineering College, Surat, India in August 1988. Obtained a Master of Science in Mechanical Engineering degree from Old Dominion University, Norfolk, Virginia, USA in December 1990. Successfully defended the dissertation for a Ph. D. in Applied Science in December 1996.