

*Final
Version*

Evaluation of Various Multi-Input Multi-Output Recursive Algorithms for On-line System Identification

by

José Luis Maldonado-Salgado

B.S. Computer Engineering, May 1992, University of Puerto Rico

A Thesis submitted to

The Faculty of

The School of Engineering and Applied Science
of The George Washington University in partial satisfaction
of the requirements for the degree of Master of Science

July, 1994

This research was conducted at NASA LaRC

Abstract

Analytical models for structural dynamics are constructed for a number of purposes and in spite of laborious modeling efforts by experienced engineers, the resulting models often have limited fidelity. To complement these analytical models, measurement models are estimated using data from experiments. Both on-line and off-line methods have received considerable attention, and for on-line estimation, a number of recursive procedures have been proposed. The work presented in this paper analyzes three recursive estimation algorithms: Recursive Least Squares, Fast Transversal Filter and Lattice Filter. All three algorithms were implemented using the control computer (IBM RS/6000) supporting the EOS AM-1 testbed to evaluate computational speed and convergence rate. Results from these studies include (1) the Recursive Least Squares is the easiest to implement, (2) the Fast Transversal Filter is the preferred method on single processor machines and (3) the Lattice Filter may be more suitable in a multiprocessor environment.

Table of Contents

Chapter 1

Introduction	1
1.1 Literature Survey	1
1.2 Research Objectives	3
1.3 Thesis Outline	4

Chapter 2

Background Theory	5
2.1 System Identification Procedure	5
2.3 Recursive System Identification	17
2.3.1 Introduction to the Classical Recursive Least-Squares Filter	17
2.3.2 Recursive Least-Squares Algorithm	22
2.3.3 Fast Transversal Filter Algorithm	23
2.3.4 Least Squares Lattice Filter	27

Chapter 3

Analytical Evaluation	32
3.1 Single-Input/Single-Output Example	32
3.2 Complexity	45
3.3 Computational Cost	48

Chapter 4

Experiment Setup	52
4.1 Testbed	52
4.2 Computer Control System (IBM RISC System/6000)	55
4.3 Instrumentation	55
4.4 Real-Time Control System Hardware	57
4.4.1 Science Instrument Simulator (SIS)	57

4.4.2 Optical Scoring System.....	58
4.5 Experiment Description	60
Chapter 5	
Experimental Results.....	63
Chapter 6	
Conclusions.....	74
6.1 Concluding Remarks.....	74
6.2 Future Studies.....	76
Chapter 7	
References.....	78
Chapter 8	
Appendix	81
8.1 C Languages Files.....	81
8.1.1 Recursive Least Squares Function	81
8.1.2 Fast Transversal Filter Functions	84
8.1.3 Least Squares Lattice Filter	97
8.2 Other Functions and header files needed for the C programs	104
8.3 Matlab Programs.....	117
8.3.1 Recursive Least Squares	117
8.3.2 Fast Transversal Filter.....	118
8.3.3 Least Squares Lattice Filter	122

Acknowledgments

First I would like to thank Dr. Lucas G. Horta, my NASA mentor, for patiently answering all my questions and for reviewing my thesis. I'd like to thank also my advisor Dr. Robert H. Tolson for helping me through these two years and letting me take the project at my own speed. I'd like to thank Dr. Jer-Nan Juang who help me understand the algorithms.

Special thanks is for Roberto Ugoletti who help me every time I had a problem with the programs and also to Dr. Chris Sandridge and Jeff Sulla who help me understand their real-time adaptive control program which is the program that calls my routines.

I would also like to thank all the people at the Spacecraft Dynamics Branch for all their support and for making me feel part of the branch since the first day.

Last, but not least I wish to thank my family for all their love and support.

List of Acronyms

3I/3O	Three-Input / Three-Output
BLS	Batch Least Squares
CAMAC	Computer Automated Measurement And Control
CEM3	CSI Evolutionary Model Phase-3
CSI	Controls-Structure Interaction
EOS	Earth Observation Satellite
FTF	Fast Transversal Filter
HGA	High Gain Antenna
IBE	Initialization Backward-Time Estimation
IFE	Initialization Forward-Time Estimation
LF	Least Squares Lattice Filter
LS	Least Squares
MIMO	Multi-Input / Multi-Output
OMP	Observer Markov Parameters
OSS	Optical Scoring System
POWER	Performance Optimized With Enhanced RISC
RISC	Reduced Instruction Set Computing
RLS	Recursive Least Squares
RSID	Recursive System Identification
SA	Solar Array
SID	System Identification
SISO	Single-Input / Single-Output
SITO	Single-Input / Two-Output
SMP	System Markov Parameters
TTTO	Two-Input / Two-Output

List of Symbols

\bar{Y}	Observer Markov Parameters (OMP)
$\hat{\bar{Y}}_p(k)$	estimated OMP
$P_p(k)$	covariance matrix
$\hat{Y}(k)$	estimated output
$e(k)$	output estimation error
$G_p(k)$	gain vector
$\hat{Y}^-(k)$	a priori output estimation
$\hat{Y}^+(k)$	a posteriori output
$e^-(k)$	a priori estimation error
$e^+(k)$	a posteriori estimation error
$E_p(k)$	equation error
$E_p(k)$	squared estimation error
$\hat{\bar{Y}}_p(k)$	forward-time OMP
$\hat{\bar{Y}}_p(k)$	backward-time OMP
$\bar{E}_p(k)$	forward-time squared estimation error
$\bar{E}_p(k)$	backward-time squared estimation error
$\gamma_p(k)$	conversion factor
$\bar{e}^-(k)$	forward-time a priori estimation error
$\bar{e}^+(k)$	forward-time a posteriori estimation error
$\bar{e}^-(k)$	backward-time a priori estimation error
$\bar{e}^+(k)$	backward-time a posteriori estimation error
$\Delta_p(k)$	time-update coefficient
$\bar{\Gamma}_p(k)$	forward-time reflection coefficient
$\bar{\Gamma}_p(k)$	backward-time reflection coefficient

List of Figures

Figure 2.1.1 Diagram of a Dynamic System.....	5
Figure 2.1.2 Flowchart of the system identification procedure.....	7
Figure 2.2.1 Block Diagram of a continuous-time state space model.....	9
Figure 3.1.1 A simple mass-spring-dashpot system.....	32
Figure 3.1.2 Convergence of the OMP.....	34
Figure 3.1.3 Residuals for the SISO system.....	34
Figure 3.1.4 RLS estimates and 95% confidence interval.....	38
Figure 3.1.5 FTF estimates and 95% confidence interval.....	39
Figure 3.1.6 LF estimates and 95% confidence interval.....	40
Figure 3.1.7 Residuals obtained using the RLS, FTF, and LF respectively.....	42
Figure 3.1.8 Estimated and measure output.....	43
Figure 3.1.9 Estimated output obtained using the three methods.....	43
Figure 3.1.10 Loss Function vs Order of the filter.....	44
Figure 3.2.1 Data flow diagram of a FTF recursion	46
Figure 3.2.2 Data flow diagram of two stages of the LF.....	47
Figure 3.3.1 Comparison of processing time for a SISO	49
Figure 3.3.2 Comparison of processing time for a SITO	50
Figure 3.3.3 Comparison of processing time for a TITO	50
Figure 3.3.4 Comparison of processing time for a 3I/3O	51
Figure 4.1.1 EOS AM-1 spacecraft.....	53
Figure 4.1.2 CEM3 Testbed	54
Figure 4.2.1 CAMAC module.....	56
Figure 4.2.2 Full-size CAMAC Crate	56
Figure 4.4.1 Gimbal Control System.....	58
Figure 4.4.2 Optical Scoring System Connection.....	59

Figure 4.4.3 Real-Time Control System	60
Figure 4.5.1 Gimbal 2 in testbed.....	61
Figure 4.5.2 Flowchart of the Experiment	62
Figure 5.1 Convergence of the Observer Markov Parameters using RLS, FTF and LF.....	65
Figure 5.2 Comparison of estimated and measured elevation angle using RLS	66
Figure 5.3 Comparison of measured and estimated azimuth angle using RLS	67
Figure 5.4 Comparison of measured and estimated elevation angle using FTF	68
Figure 5.5 Comparison of measured and estimated azimuth angle using FTF.....	69
Figure 5.6 Comparison of measured and estimated elevation angles using LF.....	70
Figure 5.7 Comparison of estimated and measured azimuth angle using LF	71
Figure 5.8 Tracking error for all three algorithms.....	72
Figure 5.9 Estimated elevation/azimuth angles for all three algorithms.....	73
Figure 5.10 Loss Function vs. Filter Order.....	74

List of Tables

- 3.1.1 Comparison of the OMP obtained by the three methods
when the system noise is described as $N(0,1\% Y_{\max})$ 35
- 3.1.2 Comparison of the OMP obtained by the three methods
when the system noise is described as $N(0,6\% Y_{\max})$ 36
- 3.3.1 Computational Cost of the Algorithms 48
- 5.1.0 Maximum Filter Order (60 Hz) 64

Chapter 1

Introduction

1.1 Literature Survey

Analytical models for structures are constructed for a number of purposes, to aid in the design process, to enable cost effective sensitivity studies, for load analyses, control design, and many others . In spite of laborious modeling efforts by experienced engineers, the resulting models have only limited capabilities. To complement these analytical models, measurement models are estimated using data from experiments. System Identification deals with the problem of building mathematical models of dynamical systems based on observed data. The types of models required could be quite different depending on the intended use. For modern control design, state space models are frequently used. One approach to obtain state space models is to identify first a model which has a linear relationship between parameters and data, such as a difference equation, and then transform this difference equation to state space form [1],[2],[19],[21].

The identification of mathematical models from input/output data can be thought of as a filter design problem. There are three types of filter design problems: filtering, smoothing, and prediction. The fundamental difference is whether the filter affects or predicts past, present, or future data. Adaptive filters have been proposed to provide the flexibility and adaptability for analysis of various types of data. Furthermore, adaptive filters are based on recursive algorithms which makes them self-tuning and amenable for on-line implementation. Their recursive formulation allows one to perform a parameter estimate $\hat{Y}_p(k+1)$ at time $k+1$ based on the solution $\hat{Y}_p(k)$ using data up to time k . This process is referred to as Recursive System Identification (RSID).

Some of the applications where RSID methods have been used are:

1) For control or signal processing applications where the action to be taken depends on the identified system model.

2) To detect sudden changes in systems for fault detection and health monitoring.

Recursive identification approaches can use time or frequency domain data for parameter estimation. This study concentrates on time domain approaches using a deterministic form of an Auto-Regressive-Moving-Average (ARMA) model structure. The ARMA model is used because the identified model is used for control applications. The Least Squares (LS) method seeks to minimize the squared error difference between measured and estimated output. It can be formulated to use all available data at once (batch) or to accept data as it becomes available from the experiment (recursive). Two recursive approaches are studied in this work, the Recursive Least Squares (RLS) and the Least Squares Lattice Filter. RLS uses a transversal fixed order filter model as the model structure. In the transversal filter algorithm a fixed order solution for the filter model is generated recursively in time. The formulation of the RLS relies on the inversion of the covariance matrix using the matrix inversion lemma [2],[5],[17]. One limitation of the RLS is its computational cost, which increases quadratically with the assumed filter order. Cioffi, et al.[13] presented a fast version of the RLS called the Fast Transversal Filter (FTF). The FTF exploits the shifting property of serialized data and the relationship of the forward-time estimation and the backward-time estimation which share a common data matrix [1],[2],[13]. For the FTF, the number of operations increases linearly with the order of the filter. Lee, et al.[25] presented a fast RLS with a ladder structure called the Least Squares Lattice Filter (LF). The LF (growing order) solves the least-squares problem by using a multistage lattice predictor [1],[2],[5],[25] as the structural basis for implementing the lattice filter. The LF is recursive in time and order and numerically efficient because the number of operations increases linearly with the order of the filter. The LF exhibits excellent convergence behavior and has very fast parameter tracking

capability. All these algorithms have been proven effective with simulated data but experimental validation is rare.

A number of applications have been reported where on-line identification is required. One example is in health monitoring and fault detection [27] of space structures. Damage scenarios such as radiation degradation of load carrying members and loosening of joints can degrade the performance or functionality of the spacecraft. From the identified parameter, frequencies and mode shapes can be obtained and compared to a baseline set of parameters to determine configuration changes. A second application is in adaptive controls of in-flight simulators [26], where an aircraft is trying to match the flying characteristics of another aircraft. In this case, the identified aircraft parameters are used to update control laws which are changing with the parameters during flight. Another application of on-line estimation and adaptive controls is in the identification of two types of models: the forward and the inverse models. The forward model predicts the system response given the inputs, while the inverse model predicts the inputs that caused the observed outputs. In control applications, the desired output is often known (tracking the inputs or maintaining a nominal position) and the problem is to determine the control inputs needed. Horta and Sandridge [3] used the identified inverse model to generate inputs that would track a desired response. In their study, the inverse and forward models were identified on-line and the inverse model was used to compute control commands to track a desired trajectory. The recursive procedure used, was a fast gain recursive algorithm presented in Ref. [3].

1.2 Research Objectives

The thesis objectives are multiple:

- 1- To compare analytically three recursive algorithms for on-line parameter estimation. The three algorithms implemented are the RLS, the FTF and the LF. This study will

examine noise effects, model order selection, initialization problems, and tracking performance of resulting models.

2- To evaluate experimentally all approaches using an IBM RS/6000 computer working in conjunction with a CAMAC (Computer Automated Measurement and Control) interface. Results from on-line identification of a two-axis gimbal instrument mounted on a laboratory model of a spacecraft are shown and compared for all three algorithm. Studies comparing processing time versus assumed system order are presented to rank the different approaches. Processing time is an important factor because the purpose of the algorithms is to be used in a real time environment and both, the update of the parameters and the control should be done in the sampling rate period.

3- To provide the real time computer system at NASA LaRC with C language software that efficiently implements all three parameter estimation algorithms on-line.

4- To study the algorithm structure and propose potential changes to the algorithms and/or computer to improve their performance.

1.3 Thesis Outline

The thesis is divided in three main parts. The first part, shown in Chapter 2, has some background identification theory and the recursive algorithms. Chapter 3, which constitutes the second part, discusses the analytical evaluation of the algorithms. This chapter covers computational cost, algorithm complexity, noise effects, order determination, and a numerical example based on a Single-Input Single-Output (SISO) system. The third part, discussed in Chapter 4 and Chapter 5, deals with the laboratory facility, instrumentation, computer system, and experimental results. Finally, some concluding remarks and suggested topics for future work are discussed in Chapter 6.

Chapter 2

Background Theory

In the following, the computational steps of the different algorithms are presented. The first section gives an introduction to the system identification problem. Section 2.2 explains the relationship between the state space and the difference equation representation. This relationship is known as the System Markov Parameters (SMP) which are related to the system matrices of the state space representation. This section also introduces the observer concept and shows how the difference equation model used in the recursive system identification methods is obtained. Section 2.3 presents the three recursive system identification methods used in this study. The last section discusses some statistical properties of the least squares estimate.

2.1. System Identification Procedure

System identification is the process of identifying mathematical models from experimental data. A dynamic system is affected by exogenous inputs and noise as shown in Figure 2.1.1.

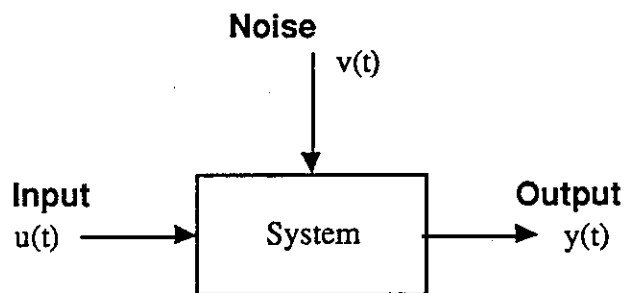


Figure 2.1.1 Diagram of a Dynamic System

Causality of the system implies that inputs at time t_0 will influence the outputs at time $t \geq t_0$. Model construction can proceed in a number of different ways. However, there are two types of models which are most often used; analysis based and/or experimental based. Analytical approaches use the laws of physics to model the dynamic behavior of a system. Models can be described by partial differential equations, ordinary differential equations, and/or difference equations.

Models can also be experimentally derived from input/output data. In the experimental approach the parameters of a model are obtained in order to fit the recorded data. Sometimes hybrid models can be obtained by using analytically derived models with parameters updated based on experimental data. Experimentally derived models have limited capabilities. For example, models obtained using system identification have the following restrictions:

- Limited dynamic range (certain types of inputs, certain working point).
- Provide little physical insight (have no physical meaning)
- Developed about nominal equilibrium state.

When identifying models from experimental data, one must be concern with a number of issues must be considered (see Figure 2.1.2):

1) Design of experiment: The performance of some models will depend on the operating conditions. The best experimental conditions are those close to the conditions the system will experience. Selection of the input signal and location is important and should be one that resembles a real input to the system. Selection of the output types, location of the sensors and signal conditioning are all important.

2) Model structure selection: Model structures are divided into parametric and nonparametric models. Nonparametric model structure uses a transfer function or frequency response as a model description. This type of model structure requires a significant number of frequency response coefficients. To reduce or compress the number of parameters needed in this system representation, parametric models are used. Parametric

models make assumptions about the mathematical structure of the model, for example, difference equation, partial differential equations, canonical representation, etc. In this form the number of coefficients used in this representation is fixed to a relatively small number, as compared to the nonparametric approach.

3) Parameter estimation process: System identification can be performed off-line and/or on-line. In the off-line case, all the data is available prior to processing the data to obtain a model. The on-line case uses data as it becomes available to update the model parameters.

4) Model validation: Two ways that can be use for model validation is the use of a performance metric and the comparison of the model prediction with the experiment data. A performance metric is a number that shows how good the model fits the data.

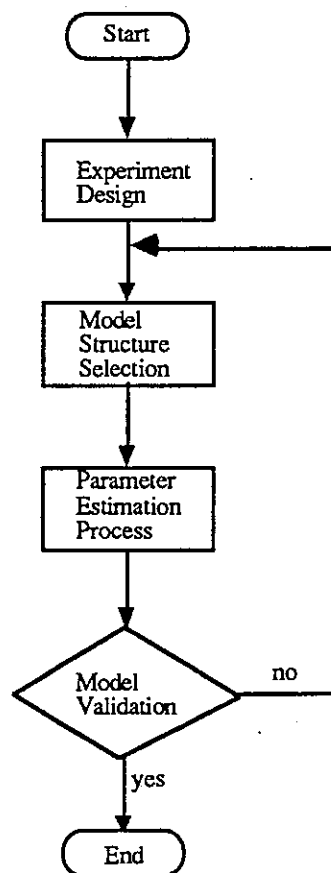


Figure 2.1.2 Flowchart of the system identification procedure

2.2. Representation of Dynamic Systems

There are various forms in which a dynamic system can be represented. The two considered here are the state space representation and the difference equation. From the state space representation a difference equation representation can be obtained and viceversa. Difference equations in the form of an ARMA model is the primary model structure used in this development.

Consider a continuous time multi-input/multi-output (MIMO) system given in state variable as:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{2.2.1}$$

where A is the state matrix, B the input matrix, C the output matrix, and D the direct transmission matrix. Assuming the system has r inputs, m outputs and n states the dimensions of these matrices are:

$$\begin{aligned} A & \quad n \times n \\ B & \quad n \times r \\ C & \quad m \times n \\ D & \quad m \times r \end{aligned}$$

A block diagram representation of Eq. (2.2.1) is given in Figure 2.2.1.

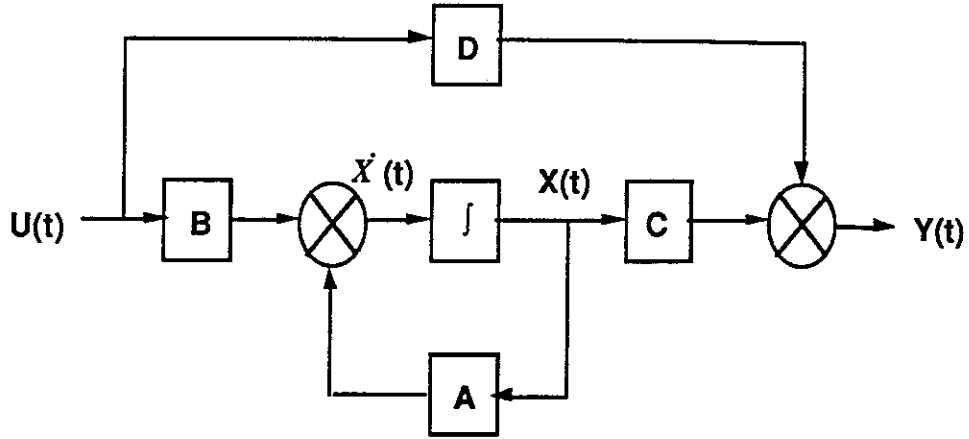


Figure 2.2.1 Block Diagram of a continuous-time state space model

A solution to Eq. (2.2.1) can be written as

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau \quad (2.2.2)$$

If the initial time is taken to be t_0 instead of 0 then Eq. (2.2.2) is modified to

$$x(t) = e^{A(t-t_0)}x(t_0) + \int_{t_0}^t e^{A(t-\tau)}Bu(\tau)d\tau \quad (2.2.3)$$

Eq. (2.2.3) describes the state change in x and its initial value $x(t_0)$ as a function of the input $u(t)$. A discrete time representation of Eq. (2.2.3) can be obtained after substituting $t_0 = k\Delta t$ and $t = (k+1)\Delta t$ as

$$x((k+1)\Delta t) = e^{A\Delta t}x(k\Delta t) + \int_{k\Delta t}^{(k+1)\Delta t} e^{A((k+1)\Delta t-\tau)}Bu(\tau)d\tau \quad (2.2.4)$$

where Δt is the sample time and k is an arbitrary non-negative integer.

Assuming the input $u(t)$ remains constant between sample times and introducing the change of variables $\tau' = (k + 1)\Delta t - \tau$, Eq. (2.2.4) becomes

$$x((k + 1)\Delta t) = e^{A\Delta t}x(k\Delta t) + \left[\int_0^{\Delta t} e^{A\tau'} d\tau' B \right] u(k\Delta t) \quad (2.2.5)$$

Defining

$$\begin{aligned} \tilde{A} &= e^{A\Delta t} \\ \tilde{B} &= \int_0^{\Delta t} e^{A\tau'} d\tau' B \\ x(k + 1) &= x[(k + 1)\Delta t] \\ u(k) &= u(k\Delta t) \end{aligned} \quad (2.2.6)$$

the discrete time version of Eq. (2.2.1) is obtained as

$$\begin{aligned} x(k + 1) &= \tilde{A}x(k) + \tilde{B}u(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \quad (2.2.7)$$

In the following, parameters for the identification problem are defined.

System Markov Parameters (SMP)

Since experimental data is obtained at discrete times, the set of equations defined in Eq. (2.2.7) forms the basis for the identification approach used here. Starting with a discrete time model and assuming an initial condition of zero, the system output $y(k)$ can be propagated as follows:

$$\begin{aligned}x(k+1) &= \tilde{A}x(k) + \tilde{B}u(k) \\y(k) &= Cx(k) + Du(k)\end{aligned}\tag{2.2.8}$$

$$\begin{aligned}x(0) &= 0 \\y(0) &= Du(0)\end{aligned}$$

$$\begin{aligned}x(1) &= \tilde{B}u(0) \\y(1) &= Cx(1) + Du(1) \\&= C\tilde{B}u(0) + Du(1)\end{aligned}$$

$$\begin{aligned}x(2) &= \tilde{A}x(1) + \tilde{B}u(1) \\&= \tilde{A}\tilde{B}u(0) + \tilde{B}u(1) \\y(2) &= Cx(2) + Du(2) \\&= C[\tilde{A}\tilde{B}u(0) + \tilde{B}u(1)] + Du(2) \\&= C\tilde{A}\tilde{B}u(0) + C\tilde{B}u(1) + Du(2)\end{aligned}$$

$$\vdots \quad \quad \quad \vdots$$

$$\begin{aligned}x(k) &= \sum_{i=1}^k \tilde{A}^{i-1} \tilde{B}u(k-i) \\y(k) &= \sum_{i=1}^k C\tilde{A}^{i-1} \tilde{B}u(k-i) + Du(k)\end{aligned}\tag{2.2.9}$$

In this form, the output equation has a series of constant matrices of dimension $m \times r$ as follows:

$$Y_0 = D \quad Y_1 = C\tilde{B} \quad Y_2 = C\tilde{A}\tilde{B}, \dots, Y_k = C\tilde{A}^{k-1}\tilde{B}\tag{2.2.10}$$

These matrices are known as Markov Parameters [1]-[2],[21] or pulse response of the system. Markov Parameters completely characterizes the dynamics of the system because they are the response of the system when it is subjected to a pulse input. The sequence of Eq. (2.2.10) will converge to a value for asymptotically stable systems. A system is said to be asymptotically stable if the real part of the eigenvalues of the state matrix A are negative.

Lightly damped systems converge very slowly and the number of nonzero terms in the sequence of Markov parameters can be excessively large.

From Eq. (2.2.10) the output $y(k)$ from Eq. (2.2.9) can be written in short form as

$$y(k) = \sum_{i=0}^k Y_i u(k-i) \quad (2.2.11)$$

For asymptotically stable systems the summation in Eq. (2.2.11) will converge to a value in a finite number of terms. A system is said to be asymptotically stable if the real part of the eigenvalues of the state matrix A are negative. Lightly damped systems converge very slowly and the number of nonzero terms in the sequence of Markov parameters can be excessively large.

In the following observer concepts are used to reduce the number of nonzero markov parameters needed for identification.

Observer Markov Parameters (OMP)

Since the state variables in general are not accessible for direct measurement [1],[11],[21] a state estimator could be used to get state estimates from input and output measurements. This state estimator is also known as an observer. To formulate the observer problem, add and subtract the term $Gy(k)$ to Eq. (2.2.7), where G is an arbitrary $n \times m$ matrix, and using the output equation yields

$$x(k+1) = (\tilde{A} + GC)x(k) + (\tilde{B} + GD)u(k) - Gy(k)$$

Defining

$$\begin{aligned}\bar{A} &= \bar{A} + GC \\ \bar{B} &= [\bar{B} + GD \quad -G] \\ v(k) &= \begin{bmatrix} u(k) \\ y(k) \end{bmatrix}\end{aligned}\tag{2.2.12}$$

a set of equations identical to Eq. (2.2.8) is obtained as

$$x(k+1) = \bar{A}x(k) + \bar{B}v(k)\tag{2.2.13}$$

$$y(k) = Cx(k) + Du(k)\tag{2.2.14}$$

The eigenvalues of \bar{A} can be placed by using the freedom in GC if the system is observable. Since matrix G can be chosen arbitrarily [2] the matrix \bar{A} can be made asymptotically stable. The matrix G can be interpreted as an observer gain and the observer form of the state equation and output equation are

$$\hat{x}(k+1) = \bar{A}\hat{x}(k) + \bar{B}u(k) - G[y(k) - \hat{y}(k)]\tag{2.2.15}$$

$$\hat{y}(k) = C\hat{x}(k) + Du(k)\tag{2.2.16}$$

where the (^) means estimate. Substituting Eq. (2.2.16) into Eq. (2.2.15) yields

$$\hat{x}(k+1) = (\bar{A} + GC)\hat{x}(k) + (\bar{B} + GD)u(k) - Gy(k)\tag{2.2.17}$$

$$= \bar{A}\hat{x}(k) + \bar{B}v(k)\tag{2.2.18}$$

Defining the state estimation error as

$$e(k) = x(k) - \hat{x}(k)$$

$$\hat{y}(k) + \sum_{i=1}^k \bar{Y}_i^{(2)} y(k-i) = \sum_{i=1}^k \bar{Y}_i^{(1)} u(k-i) + Du(k) \quad (2.2.22)$$

or

$$\hat{y}(k) + \sum_{i=1}^p \bar{Y}_i^{(2)} y(k-i) = \sum_{i=1}^p \bar{Y}_i^{(1)} u(k-i) + Du(k) \quad (2.2.23)$$

provided that \bar{A} is deadbeat of order p , i.e. ($\bar{A}^p = 0$). Using Eq. (2.2.21) , Eq. (2.2.23) can be written as

$$y(k) = \sum_{i=1}^p \bar{Y}_i v(k-i) + Du(k) \quad (2.2.24)$$

Equation (2.2.24) is a linear difference equation in the form of an Auto-Regresive Moving Average model. When comparing Eq. (2.2.11) with Eq. (2.2.24) note that the number of parameters is limited in this case to p .

The philosophy of this approach is to identify the OMP directly, and then recover the SMP from the observer. A solution to this procedure is discussed next.

Solution of the System Markov Parameters from the Observer Markov Parameters

The system identification algorithms to be presented in the next section solve for the Observer Markov Parameters of the system. From Eq. (2.2.20) the OMP are defined as

$$\bar{Y} = [D \quad C\bar{B} \quad C\bar{A}\bar{B} \quad \dots \quad C\bar{A}^{k-1}\bar{B}] \quad (2.2.25)$$

where

$$\begin{aligned}\bar{A} &= \tilde{A} + GC \\ \bar{B} &= \begin{bmatrix} -G & \tilde{B} + GD \end{bmatrix}\end{aligned}$$

Partitioning Eq. (2.2.25), the OMP are,

$$\begin{aligned}\bar{Y} &= D \\ \bar{Y}_k &= C\bar{A}^{k-1}\bar{B} = \begin{bmatrix} -\bar{Y}_k^{(2)} & \bar{Y}_k^{(1)} \end{bmatrix}\end{aligned}\tag{2.2.26}$$

The first SMP, $C\tilde{B}$ can be obtained by substituting the definition of \bar{B} and \bar{A} into Eq. (2.2.26) for $k=1$:

$$\begin{aligned}Y_1 &= C\tilde{B} = C(\tilde{B} + GD) - (CG)D \\ &= \bar{Y}_1^{(1)} - \bar{Y}_1^{(2)}D\end{aligned}$$

Performing the same operation for all the other parameters, a general expression between the OMP and the SMP [2] is

$$\begin{aligned}Y_0 &= \bar{Y}_0 = D \\ Y_k &= \bar{Y}_k^{(1)} - \sum_{i=1}^k \bar{Y}_i^{(2)} Y_{k-i} \quad k = 1 \dots p\end{aligned}\tag{2.2.27}$$

This section discussed a model structure in terms of OMP. It was shown how this model structure reduced the number of parameters needed for identification. Having formulated the model structure, the next step is to formulate the system identification problem.

2.3 Recursive System Identification

2.3.1 Introduction to the Classical Recursive Least-Squares Filter

The output from Eq. (2.2.24) can be written in the following form :

$$y(k) = \bar{Y}v_p(k-1) \quad (2.3.1)$$

where

$$v_p^T(k-1) = [u^T(k) \quad v^T(k-1) \quad \dots \quad v^T(k-p)]; \langle 1 \times (r+m)p+r \rangle \quad (2.3.2)$$

$$\bar{Y} = [D \quad \bar{Y}_1 \quad \dots \quad \bar{Y}_p]; \langle 1 \times (r+m)p+r \rangle \quad (2.3.3)$$

In general, systems with r inputs and m outputs will be considered. The time index $(k-1)$ references the last or newest output information, and the assumed system order is p . Writing Eq. (2.3.1) for different k , the following matrix equation can be written as:

$$Y(k) = \bar{Y}V_p(k-1) \quad (2.3.4)$$

where

$$V_p(k-1) = \begin{bmatrix} u(0) & u(1) & u(2) & \dots & u(p) & \dots & u(k) \\ 0 & v(0) & v(1) & \dots & v(p-1) & \dots & v(k-1) \\ 0 & 0 & v(0) & \dots & v(p-2) & \dots & v(k-2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & v(0) & \dots & v(k-p) \end{bmatrix} \quad (2.3.5)$$

$; \langle (r+m)p+r \times (k+1) \rangle$

and

$$\mathbb{Y}(k) = [y(0) \ \cdots \ y(k)]; (m \times (k+1)) \quad (2.3.6)$$

Matrix V_p is the input/output data matrix and $\mathbb{Y}(k)$ is the output matrix.

In order to obtain the observer markov parameters, Eq. (2.3.4) has to be solved. Post multiplying both sides of Eq. (2.3.4) by $V_p^T(k-1)$ produces

$$\mathbb{Y}(k)V_p^T(k-1) = \bar{\mathbb{Y}}[V_p(k-1)V_p^T(k-1)] \quad (2.3.7)$$

assuming k is greater than $(r+m)p+r$, inverting the covariance matrix, $[V_p(k-1)V_p^T(k-1)]$, and post multiplying it on both sides of Eq. (2.3.7), the observer markov parameters are obtained. The matrix $[V_p(k-1)V_p^T(k-1)]$ is a positive definite matrix if the input/output signals are sufficiently rich, so its rank is $(r+m)p+r$. A solution for the observer parameters is

$$\hat{\bar{\mathbb{Y}}}_p(k) = \mathbb{Y}(k)V_p^T(k-1)[V_p(k-1)V_p^T(k-1)]^{-1} \quad (2.3.8)$$

A solution to Eq. (2.3.8) can only be obtained after all data have been collected; this solution is known as the batch least-squares (BLS). A problem with the BLS is that every time new data becomes available the covariance matrix has to be inverted. This matrix can be large depending on the system order and the number of inputs and outputs. When a new set of input and output data is obtained the parameter matrix must satisfy the following equation

$$\hat{\bar{\mathbb{Y}}}_p(k+1) = \mathbb{Y}(k+1)V_p^T(k)[V_p(k)V_p^T(k)]^{-1} \quad (2.3.9)$$

It can be shown that

$$Y(k+1) = [Y(k) \quad y(k+1)] \quad (2.3.10)$$

and

$$V_p(k) = [V_p(k-1) \quad v_p(k)] \quad (2.3.11)$$

Defining the covariance matrix at time step $k+1$ as

$$P_p(k) = [V_p(k)V_p^T(k)]^{-1} \quad (2.3.12)$$

and substituting Eq. (2.3.11) in Eq. (2.3.12) a recursive definition of the covariance matrix is obtained

$$\begin{aligned} P_p(k) &= \left[[V_p(k-1) \quad v_p(k)] \begin{bmatrix} V_p^T(k-1) \\ v_p^T(k) \end{bmatrix} \right]^{-1} \\ &= [V_p(k-1)V_p^T(k-1) + v_p(k)v_p^T(k)]^{-1} \\ &= [P_p^{-1}(k-1) + v_p(k)v_p^T(k)]^{-1} \end{aligned} \quad (2.3.13)$$

Using the matrix inversion lemma [2],[17],[24], Eq. (2.3.13) can be simplified and a recursive solution for $P_p(k)$ is obtained

$$\begin{aligned} P_p(k) &= P_p(k-1) - P_p(k-1)v_p(k)[1 + v_p^T(k)P_p(k-1)v_p(k)]^{-1}v_p^T(k)P_p(k-1) \\ &= P_p(k-1) \left[I - \frac{v_p(k)v_p^T(k)P_p(k-1)}{S} \right] \end{aligned} \quad (2.3.14)$$

where S is a scalar quantity equal to

$$S = [1 + v_p^T(k)P_p(k-1)v_p(k)]$$

Substituting Eqs. (2.3.11) and (2.3.14) into Eq. (2.3.9) gives

$$\hat{\bar{Y}}_p(k+1) = \left[\bar{Y}(k)V_p^T(k-1) + y(k+1)v_p^T(k) \right] P_p(k-1) \left[I - \frac{v_p(k)v_p^T(k)P_p(k-1)}{S} \right]$$

Simplifying this equation noting that

$$\hat{\bar{Y}}_p(k) = \bar{Y}(k)V_p^T(k-1)P_p(k-1)$$

yields the recursive solution for observer markov parameters

$$\hat{\bar{Y}}_p(k+1) = \hat{\bar{Y}}_p(k) + \left[y(k+1) - \hat{\bar{Y}}_p(k)v_p(k) \right] \frac{v_p^T(k)P_p(k-1)}{S} \quad (2.3.15)$$

Using the following definitions

$$G_p(k) = \frac{v_p^T(k)P_p(k-1)}{S} \quad (2.3.16)$$

$$\hat{y}(k+1) = \hat{\bar{Y}}_p(k)v_p(k) \quad (2.3.17a)$$

$$e(k+1) = y(k+1) - \hat{y}(k+1) \quad (2.3.17b)$$

the covariance and observer markov parameters matrices are given by

$$P_p(k) = P_p(k-1)[I - v_p(k)G_p(k)] \quad (2.3.18)$$

$$\hat{\bar{Y}}_p(k+1) = \hat{\bar{Y}}_p(k) + e(k+1)G_p(k) \quad (2.3.19)$$

The output residual $e(k+1)$ is the difference between the measured output at time $k+1$ and the estimated output obtained from Eq. (2.3.17a). The gain vector $G_p(k)$ is a weighting factor that controls the contribution of the output residual to parameter update.

$P_p(0)$ and $\hat{Y}_p(1)$ can be assumed or estimated by performing a BLS with the initial data. In the numerical implementation, $P_p(0)$ and $\hat{Y}_p(1)$ are initialized as follows:

$$P_p(0) = d \times I_{(r+m)p+r}$$

$$\hat{Y}_p(1) = 0_{m \times (r+m)p+r}$$

To obtain the estimated output $y(k+1)$ from Eq. (2.3.17) the measurement at time $k+1$ is not used because $\hat{Y}_p(k)$ takes into account only the output at time k . This means that $\hat{y}(k+1)$ from Eq. (2.3.17a) is an a priori estimate of the output $\hat{y}^-(k+1)$. The output residual from Eq. (2.3.17b) is the a priori estimation error [2],[5] and will be denoted as $e^-(k+1)$. Using $\hat{Y}_p(k+1)$ to obtain $\hat{y}(k+1)$ results in the a posteriori estimation of the output $\hat{y}^+(k+1)$, and the difference of the measured output $y(k+1)$ and $\hat{y}^+(k+1)$ is the a posteriori estimation error denoted by $e^+(k+1)$.

$$\hat{y}^-(k+1) = \hat{Y}_p(k)v_p(k) \quad (2.3.20)$$

$$\hat{y}^+(k+1) = \hat{Y}_p(k+1)v_p(k) \quad (2.3.21)$$

$$e^-(k+1) = y(k+1) - \hat{y}^-(k+1) \quad (2.3.22)$$

$$e^+(k+1) = y(k+1) - \hat{y}^+(k+1) \quad (2.3.23)$$

From these definitions, Eq. (2.3.19) can be written as:

$$\hat{Y}_p(k+1) = \hat{Y}_p(k) + e^-(k+1)G_p(k) \quad (2.3.24)$$

Substituting Eq (2.3.21) into Eq. (2.3.23) and using Eq. (2.3.24), the a posteriori estimation error is obtained in terms of the a priori estimation error as

$$\begin{aligned}
e_p^+(k+1) &= y(k+1) - \hat{Y}_p(k+1)v_p(k) \\
&= [1 - G_p(k)v_p(k)]e_p^-(k+1) \\
&= \frac{e_p^-(k+1)}{S}
\end{aligned} \tag{2.3.25}$$

It can be shown [2] that the scalar, $S > 1$, makes the a posteriori error smaller than the a priori error.

The equation error is analogous to Eq. (2.3.17b), but in matrix form including all the outputs up to time step $k+1$. The equation error is defined as

$$E_p(k+1) = Y(k+1) - \bar{Y}_p(k+1)V_p(k) \tag{2.3.26}$$

Using Eqs. (2.3.10), (2.3.11), (2.3.23), (2.3.24) $E_p(k+1)$ becomes

$$E_p(k+1) = \begin{bmatrix} E_p(k) - e_p^-(k+1)G_p(k)V_p(k-1) & e_p^+(k+1) \end{bmatrix}_{m \times k+2} \tag{2.3.27}$$

The squared estimation error is defined as:

$$E_p(k+1) = E_p(k+1)E_p^T(k+1) \tag{2.3.28}$$

Substituting Eq. (2.3.27) into Eq. (2.3.28) and simplifying the recursive equation for the squared error yields

$$E_p(k+1) = E_p(k) + e_p^+(k+1)[e_p^-(k+1)]^T \tag{2.3.29}$$

In the following, all computational steps for the recursive parameter estimation are presented.

2.3.2 Recursive Least-Squares Algorithm

Computational Steps

Initialization:

$$P_p(0) = d \times I_{(r+m)p+r}$$

$$\hat{Y}_p(1) = 0_{m \times (r+m)p+r}$$

where d is a large number.

Recursion Steps:

$$1) \quad v_p^T(k) = [u^T(k+1) \quad v^T(k) \quad \dots \quad v^T(k+1-p)]$$

$$2) \quad S = [1 + v_p^T(k)P_p(k-1)v_p(k)]$$

$$3) \quad G_p(k) = \frac{v_p^T(k)P_p(k-1)}{S}$$

$$4) \quad \hat{y}(k+1) = \hat{Y}_p(k)v_p(k)$$

$$5) \quad e(k+1) = y(k+1) - \hat{y}(k+1)$$

$$6) \quad P_p(k) = P_p(k-1)[I - v_p(k)G_p(k)]$$

$$7) \quad \hat{Y}_p(k+1) = \hat{Y}_p(k) + e(k+1)G_p(k)$$

Increment the time index k and start again in Step 1.

2.3.3 Fast Transversal Filter Algorithm

The Fast Transversal Filter (FTF) is a least squares filter that uses the shifting property of serialized data [2],[5],[13] to reduce computational complexity. The FTF uses the relationship between forward-time estimation and backward-time estimation which share a common data matrix. In the following the steps of the FTF are given for completeness.

Computational Steps

Initialization: $k > (r + m)p + r$

$$1) \quad v_p^T(k-1) = [u^T(k) \quad v^T(k-1) \quad \cdots \quad v^T(k-p)]$$

$$2) \quad V_p(k-2) = \begin{bmatrix} u(0) & u(1) & u(2) & \cdots & u(p) & \cdots & u(k-1) \\ 0 & v(0) & v(1) & \cdots & v(p-1) & \cdots & v(k-2) \\ 0 & 0 & v(0) & \cdots & v(p-2) & \cdots & v(k-3) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & v(0) & \cdots & v(k-p-1) \end{bmatrix}$$

$$3) \quad V_p(k-1) = [V_p(k-2) \quad v_p(k-1)]$$

$$4) \quad \mathbb{Y}_u(k-1) = \begin{bmatrix} u(1) & u(2) & u(3) & \cdots & u(p+1) & \cdots & u(k) \\ y(0) & y(1) & y(2) & \cdots & y(p) & \cdots & y(k-1) \end{bmatrix}$$

$$5) \quad \mathbb{Y}_u(-1) = \begin{bmatrix} u(0) \\ 0_{m \times 1} \end{bmatrix}$$

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

Initialization of the Forward-Time Estimation

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

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

$$9) \quad \hat{Y}_p(k-1) = Y_{\bar{u}}(k-1)V_p^T(k-2)P_p(k-2)$$

$$10) \quad \bar{E}_p(k-1) = Y_{\bar{u}}(-1)Y_{\bar{u}}^T(-1) + \left[Y_{\bar{u}}(k-1) - \hat{Y}_p(k-1)V_p(k-2) \right] \times \\ \left[Y_{\bar{u}}(k-1) - \hat{Y}_p(k-1)V_p(k-2) \right]^T$$

$$11) \quad \gamma_p(k-1) = 1 - G_p(k-1)v_p(k-1)$$

Initialization of the Backward-Time Estimation

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

$$13) \quad \hat{Y}_p(k-1) = Y_{\bar{u}}(k-1-p)V_p^T(k-1)P_p(k-1)$$

$$14) \quad \bar{E}_p(k-1) = \left[Y_{\bar{u}}(k-1-p) - \hat{Y}_p(k-1)V_p(k-1) \right] \times \\ \left[Y_{\bar{u}}(k-1-p) - \hat{Y}_p(k-1)V_p(k-1) \right]^T$$

Recursion Steps

Forward-Time Update Recursion

$$15) \quad v_p^T(k) = [u^T(k+1) \quad v^T(k) \quad \dots \quad v^T(k-p+1)]$$

$$16) \quad Y_{\bar{u}}(k) = \begin{bmatrix} u(k+1) \\ y(k) \end{bmatrix}$$

$$17) \quad \bar{e}_p^-(k) = Y_{\bar{u}}(k) - \hat{Y}_p(k-1)v_p(k-1)$$

$$18) \quad \bar{e}_p^+(k) = \gamma_p(k-1)\bar{e}_p^-(k)$$

$$19) \quad \hat{Y}_p(k) = \hat{Y}_p(k-1) + \bar{e}_p^+(k)G_p(k-1)$$

$$20) \quad \bar{E}_p(k) = \bar{E}_p(k-1) + \bar{e}_p^+(k)[\bar{e}_p^-(k)]^T$$

$$21) \quad G_{p+1}(k) = \begin{bmatrix} \bar{e}_p^+(k)^T \bar{E}_p^{-1}(k) & G_p(k-1) - \bar{e}_p^+(k)^T \bar{E}_p^{-1}(k) \hat{Y}_p(k) \end{bmatrix}$$

$$22) \quad \gamma_{p+1}(k) = \gamma_p(k-1) - [\bar{e}_p^+(k)]^T \bar{E}_p^{-1}(k) \bar{e}_p^+(k)$$

Partition the gain vector into two vectors of dimensions:

$$G_{p+1}^{(r)}(k) \quad 1 \times [(r+m)p+r]$$

$$G_{p+1}^{(a)}(k) \quad 1 \times (r+m)$$

$$23) \quad G_{p+1}(k) = \begin{bmatrix} G_{p+1}^{(r)}(k) & G_{p+1}^{(a)}(k) \end{bmatrix}$$

Backward-Time Update Recursion

$$24) \quad v_p^T(k) = [u^T(k+1) \quad v^T(k) \quad \cdots \quad v^T(k-p+1)]$$

$$25) \quad Y_u(k-p) = \begin{bmatrix} y(k-p) \\ u(k-p) \end{bmatrix}$$

$$26) \quad \bar{e}_p^-(k) = Y_u(k-p) - \hat{Y}_p(k-1)v_p(k)$$

$$27) \quad G_p(k) = \frac{G_{p+1}^{(r)}(k) + G_{p+1}^{(a)}(k)\hat{Y}_p(k-1)}{1 - G_{p+1}^{(a)}(k)\bar{e}_p^-(k)}$$

$$28) \quad \hat{Y}(k) = \hat{Y}(k-1) + \bar{e}_p^-(k)G_p(k)$$

$$29) \quad \gamma_p(k) = \frac{\gamma_{p+1}(k)}{1 - G_{p+1}^{(a)}(k)\bar{e}_p^-(k)}$$

$$30) \quad \bar{e}_p^+(k) = \gamma_p(k)\bar{e}_p^-(k)$$

$$31) \quad \bar{E}_p(k) = \bar{E}_p(k-1) + \bar{e}_p^+(k)[\bar{e}_p^-(k)]^T$$

Increment the time index k and start again in the Forward-Time Update Recursion.

2.3.4 Least Squares Lattice Filter Algorithm

The Least Squares Lattice Filter (LF) [1],[2],[4],[5] is another type of least-squares filter. It follows the same approach as the FTF where there is a forward-time and backward-time estimation. The difference is that the LF has a time and order update. Also,

the LF lends itself for running in a parallel machine. The LF has basically three steps: time update, forward-time order update and backward-time order update.

The computational steps of the LF algorithm are :

Computational Steps

Initialization at k=0 Steps

$$1) \quad P_0(0) = d \times I_r \quad \text{where } d \text{ is a large positive number}$$

$$2) \quad G_0(0) = 0_{1 \times r}$$

$$3) \quad \gamma_0(0) = 1$$

$$4) \quad \hat{Y}_0(0) = 0_{(r+m) \times r}$$

$$5) \quad \hat{Y}_p(0) = 0_{(r+m) \times \{r+(r+m)p\}}$$

$$6) \quad \bar{E}_0(0) = \delta \times I_{(r+m)} \quad \text{where } \delta \text{ is a small positive number}$$

$$7) \quad \bar{E}_p(0) = \delta \times I_{(r+m)}$$

$$8) \quad \Delta_p(0) = 0_{(r+m)}$$

$$9) \quad \bar{e}_p^+(0) = 0_{(r+m) \times 1}$$

Initialization at $k \geq 1$ and $p=0$ Steps

Forward-Time Estimation Initialization

$$10) \quad v_0(k-1) = u(k)$$

$$11) \quad Y_{\bar{z}}(k) = \begin{bmatrix} u(k+1) \\ y(k) \end{bmatrix}$$

$$12) \quad \bar{e}_0^-(k) = Y_{\bar{z}}(k) - \hat{Y}_0(k-1)v_0(k-1)$$

$$13) \quad \bar{e}_0^+(k) = \gamma_0(k-1)\bar{e}_0^-(k)$$

$$14) \quad \bar{E}_0(k) = \bar{E}_0(k-1) + \bar{e}_0^+(k)[\bar{e}_0^-(k)]^T$$

$$15) \quad \hat{Y}_0(k) = \hat{Y}_0(k-1) + \bar{e}_0^-(k)G_0(k)$$

Backward-Time Estimation Initialization

$$16) \quad v_0(k) = u(k+1)$$

$$17) \quad Y_u(k) = \begin{bmatrix} y(k) \\ u(k) \end{bmatrix}$$

$$18) \quad \bar{e}_0^-(k) = Y_u(k) - \hat{Y}_0(k-1)v_0(k)$$

$$19) \quad G_0(k) = \frac{v_0^T(k)P_0(k-1)}{1 + v_0^T(k)P_0(k-1)v_0(k)}$$

$$20) \quad P_0(k) = P_0(k-1)[I_r - v_0(k)G_0(k)]$$

$$21) \quad \hat{\bar{Y}}_0(k) = \hat{\bar{Y}}_0(k-1) + \bar{e}_0^-(k)G_0(k)$$

$$22) \quad \gamma_0(k) = 1 - G_0(k)v_0(k)$$

$$23) \quad \bar{e}_0^+(k) = \gamma_0(k-1)\bar{e}_0^-(k)$$

$$24) \quad \bar{E}_0(k) = \bar{E}_0(k-1) + \bar{e}_0^+(k)[\bar{e}_0^-(k)]^T$$

Time-Update Coefficient Recursion: $p \geq 0$

$$25) \quad \Delta_p(k) = \Delta_p(k-1) + \frac{\bar{e}_p^+(k)[\bar{e}_p^+(k-1)]^T}{\gamma_p(k-1)}$$

Forward-Time Order-Update Recursion: $p \geq 0$

$$26) \quad \bar{\Gamma}_{p+1}(k) = -\Delta_p(k)\bar{E}_p^{-1}(k-1)$$

$$27) \quad \hat{\bar{Y}}_{p+1}(k) = \begin{bmatrix} \hat{\bar{Y}}_p(k) + \bar{\Gamma}_{p+1}(k)\hat{\bar{Y}}_p(k-1) & -\bar{\Gamma}_{p+1}(k) \end{bmatrix}$$

$$28) \quad \bar{e}_{p+1}^+(k) = \bar{e}_p^+(k) + \bar{\Gamma}_{p+1}(k)\bar{e}_p^+(k-1)$$

$$29) \quad \bar{E}_{p+1}(k) = \bar{E}_p(k) - \Delta_p(k)\bar{E}_p^{-1}(k-1)\Delta_p^T(k)$$

Backward-Time Order-Update Recursion: $p \geq 0$

$$30) \quad \bar{\Gamma}_{p+1}(k) = -\Delta_p(k) \bar{E}_p^{-1}(k)$$

$$31) \quad \hat{Y}_{p+1}(k) = \left[-\bar{\Gamma}_{p+1}(k) \quad \hat{Y}_p(k-1) + \bar{\Gamma}_{p+1}(k) \hat{Y}_p(k) \right]$$

$$32) \quad \bar{e}_{p+1}^+(k) = \bar{e}_p^+(k-1) + \bar{\Gamma}_{p+1}(k) \bar{e}_p^+(k)$$

$$33) \quad \bar{E}_{p+1}(k) = \bar{E}_p(k-1) - \Delta_p^T(k) \bar{E}_p^{-1}(k) \Delta_p(k)$$

$$34) \quad \gamma_{p+1}(k-1) = \gamma_p(k-1) - [\bar{e}_p^+(k-1)]^T \bar{E}_p^{-1}(k-1) \bar{e}_p^+(k-1)$$

Increment time index k and go back to Step 1 of the Forward-Time Estimation Initialization.

Chapter 3

Analytical Evaluation

This chapter presents the analytical evaluation of the three algorithms based on computational cost, complexity and rate of convergence. Section 3.1 compares the results obtained from the three algorithms using simulated data of a single-input single-output (SISO) system. Complexity in terms of algorithm structure and number of steps per recursion is discussed in section 3.2. Section 3.3 compares the computational cost of the three algorithms in terms of processing time and number of operations.

3.1. Single-Input/Single-Output (SISO) Example

The SISO system shown in Figure 3.1.1, is used to generate data for processing with all three algorithms.

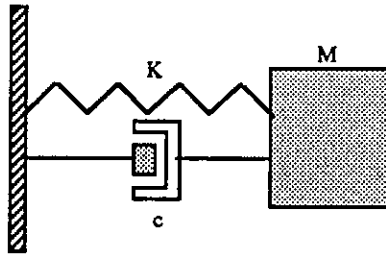


Figure 3.1.1 A simple mass-spring-dashpot system.

The simulated system parameters are chosen as $M=1$ Kg, $K=1$ N/m, and $c=.01$ Kg/s. Equations of motion for the continuous-time state space model are:

$$\begin{aligned}\dot{X} &= AX + BU \\ Y &= CX + DU\end{aligned}$$

$$A = \begin{bmatrix} 0 & 1 \\ -1 & -.01 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = [-1 \quad -.01], \quad D = [1]$$

The mass acceleration is used as the system output. The input is a vector of normally distributed numbers with zero mean and variance of one. The system is discretized using a $dt=0.2$ and the time responses are generated using Matlab. For plotting purposes, the time scale is normalized using the fundamental period of the system. The input and output data generated from MATLAB was scaled in terms of the maximum value of the inputs and the maximum value of the outputs to prevent ill conditioning of the problem. In this way the magnitude of the inputs and the outputs are between 0 and 1.

The identification for this system was done first with noise-free data, to evaluate convergence rates for the different parameters and also to determine the amount of data cycles needed to achieve converged results. Two levels of noise were added to the output of the system to study noise effects on the parameter convergence. This noise added to the output represents measurement errors, random disturbances and fitting errors. All noise sequences were generated using a zero mean normally distributed sequence with variance equal to a percent of the maximum value of the output. The two noise levels are given as $N(0, 1\% Y_{\max})$ and $N(0, 6\% Y_{\max})$.

Results for the simulated SISO system without noise are shown in Figure 3.1.2. It takes about 5 cycles for the RLS and the LF to converge to the true values of the OMP. The FTF uses a BLS for the initialization, and since the data is without noise the OMP obtained from the BLS are converged OMP.

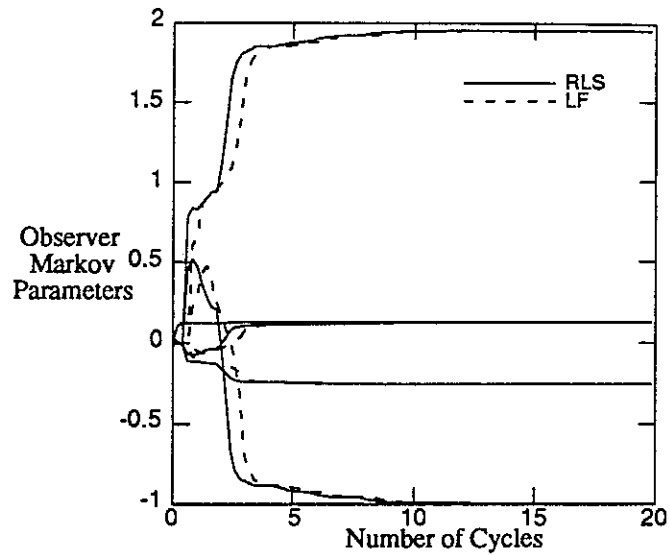


Figure 3.1.2 Convergence of the OMP

Another metric of model quality is fitting error, which is the difference between the measure and estimated output. For noise-free data the residuals go to zero in about 5 cycles as seen in Figure 3.1.3 for the RLS and the LF.

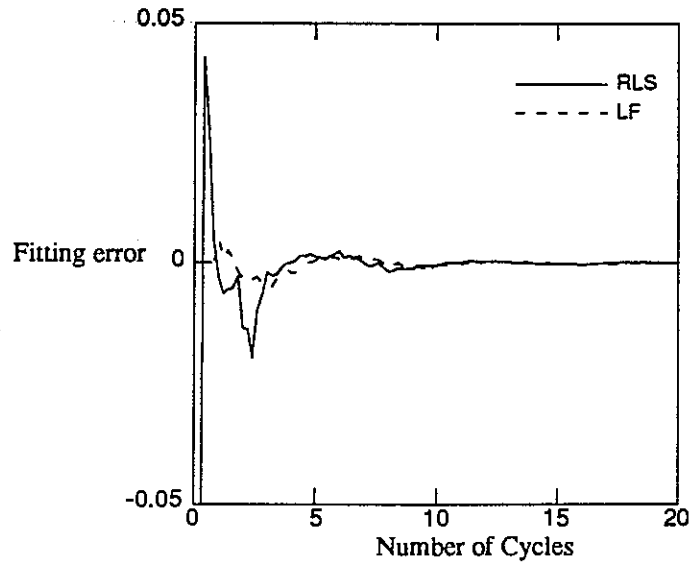


Figure 3.1.3 Residuals for the SISO system

Table 3.1.1 shows a comparison of the exact parameters versus results obtained using all three methods when the added noise is $N(0, 1\% Y_{\max})$ and Table 3.1.2 is for the case when the noise level is $N(0, 6\% Y_{\max})$. These tables show the estimation after the last data pair (in this case $n=5000$) is processed.

Table 3.1.1 Comparison of the OMP obtained by the three methods with exact values when the noise level is $N(0, 1\% Y_{\max})$.

Exact Values						
\bar{Y}		.1280	1.9582	-.2534	-.9980	.1254
Recursive Least Squares (RLS)						
$\bar{\hat{Y}}$	mean	.1267	1.8736	-.2404	-.9139	.1131
\hat{Y}_{last}	last	.1268	1.8750	-.2405	-.9154	.1135
σ	std. dev.	.0002	.0252	.0030	.0252	.0032
Fast Transversal Filter (FTF)						
$\bar{\hat{Y}}$	mean	.1267	1.8736	-.2405	-.9140	.1131
\hat{Y}_{last}	last	.1268	1.8751	-.2406	-.9155	.1136
σ	std. dev.	.0002	.0252	.0030	.0251	.0032
Lattice Filter (LF)						
$\bar{\hat{Y}}$	mean	.1267	1.8736	-.2404	-.9139	.1131
\hat{Y}_{last}	last	.1268	1.8549	-.2405	-.9153	.1135
σ	std. dev.	.0002	.0253	.0030	.0252	.0032

Table 3.1.2 Comparison of the OMP obtained by the three methods with exact values when the noise level is $N(0, 6\% Y_{max})$.

Exact Values						
\bar{Y}		.1280	1.9582	-.2534	-.9980	.1254
Recursive Least Squares (RLS)						
$\hat{\bar{Y}}$	mean	.1258	.9309	-.1193	.0073	-.0065
\hat{Y}_{last}	last	.1267	.9214	-.1171	.0169	-.0067
σ	std. dev.	.0013	.0901	.0103	.0834	.0104
Fast Transversal Filter (FTF)						
$\hat{\bar{Y}}$	mean	.1257	.9306	-.1192	.0075	-.0066
\hat{Y}_{last}	last	.1267	.9214	-.1170	.0169	-.0067
σ	std. dev.	.0013	.0903	.0103	.0835	.0104
Lattice Filter (LF)						
$\hat{\bar{Y}}$	mean	.1257	.9307	-.1192	.0074	-.0066
\hat{Y}_{last}	last	.1267	.9214	-.1170	.0170	-.0067
σ	std. dev.	.0013	.0902	.0103	.0835	.0104

All of the OMP obtained from the three algorithms and for each noise level are nearly identical. The small differences observed are caused by numerical round-off of the different algorithms. Also shown in the table is the mean value of the OMP and their standard deviation. The OMP mean and standard deviation values for the RLS and the LF were computed using the last 4800 points to compare those results with the ones from the FTF, because the FTF uses the first 200 data pairs to run the BLS for initialization. When noise is present in the data the three algorithms produce the same standard deviation and mean of the OMP.

An alternate way of looking at noise effects on the parameters estimates, is to plot the exact value of the OMP and the value obtained when noise is present. As stated before, the noise added to the system accounts for process, measurement, and fitting errors. At the same time, plots of the confidence interval for the mean of the parameters is used to see if the parameters converge to a value when $k \rightarrow \infty$. The confidence interval used in this study is a 95% confidence that the expected value of the parameter is inside the interval. For a 95% confidence, the interval is defined [15] as:

$$\bar{z} - 1.96 \frac{\sigma}{\sqrt{n}} < \mathfrak{S} < \bar{z} + 1.96 \frac{\sigma}{\sqrt{n}} \quad (3.1.1)$$

where \bar{z} is mean of the OMP, σ is the standard deviation of the OMP, n is the number of data points used for the computations and \mathfrak{S} is the limiting mean of the OMP or the value the OMP approach when $k \rightarrow \infty$. The interval was computed using a sliding window composed of 50 data points. Figure 3.1.4 shows RLS results for the first four OMP of the SISO system when the noise in the system is $N(0, 1\% Y_{\max})$. True OMP values are also plotted to show the bias in the solution. The bias in the solution is because the methods are least square based and the selection of the model must take into account explicitly the noise contribution entering the problem.

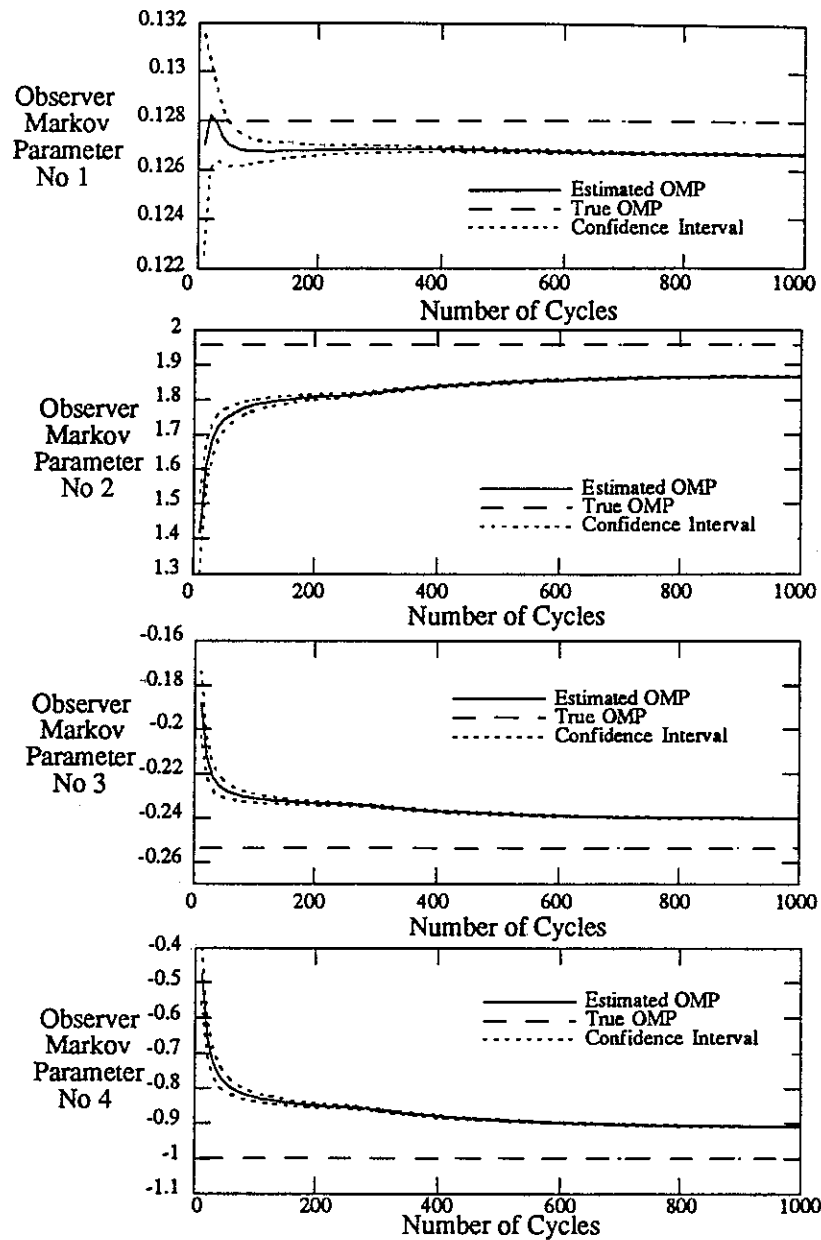


Figure 3.1.4 RLS estimates and 95% confidence interval

Figure 3.1.4 shows the 95% confidence interval for the expected value of the first four OMP for the SISO system. It can be seen that the expected value of the mean of all the OMP converge to a value, even though is not the correct value. The exact OMP value is also plotted to show bias in the estimated OMP compared to the exact values. Figures

3.1.5 and 3.1.6 shows the same plots as Figure 3.1.4 , but these were obtained using the FTF and the LF respectively.

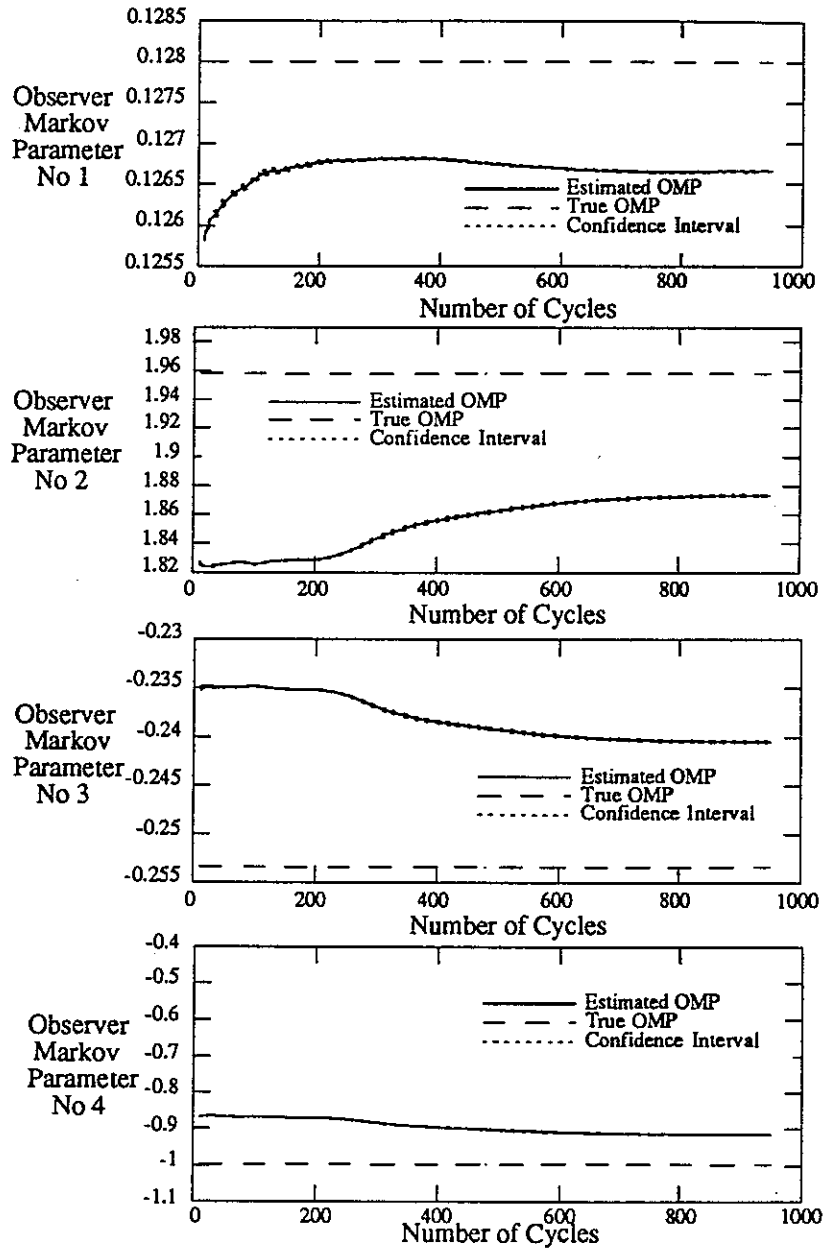


Figure 3.1.5 FTF estimates and 95% confidence interval

Figures 3.1.4 and 3.1.6 are identical as is expected. The difference in Figure 3.1.5, which is the one obtained using the FTF, is due to the initialization of the FTF. Since a BLS is run before, the initial estimates of the OMP are close to the value it will converge and

therefore the confidence interval is smaller even in the initial cycles of the on-line estimation.

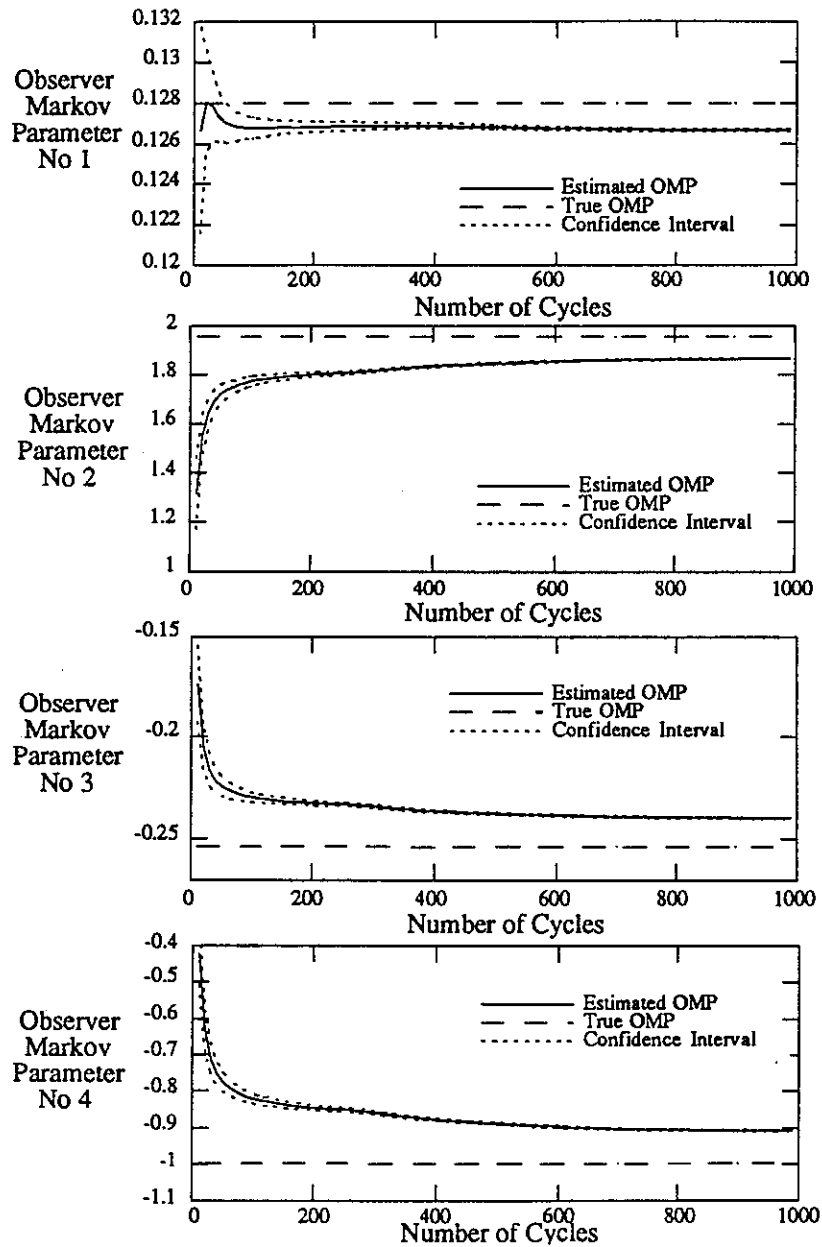


Figure 3.1.6 LF estimates and 95% confidence interval

Even though the estimates of the OMP are biased, the fit using the estimated model is good as seen in the residuals. The residuals provide information regarding assumptions about error terms and the appropriateness of the model. Figure 3.1.7 shows the residuals for the SISO with noise level is $N(0, 1\% Y_{\max})$ using the three methods. In the example, the residuals are centered around zero with a standard deviation less than 0.05. The standard deviation of the residuals is around 0.05 as opposed to 0.01 because the model used for the identification used $p=2$.

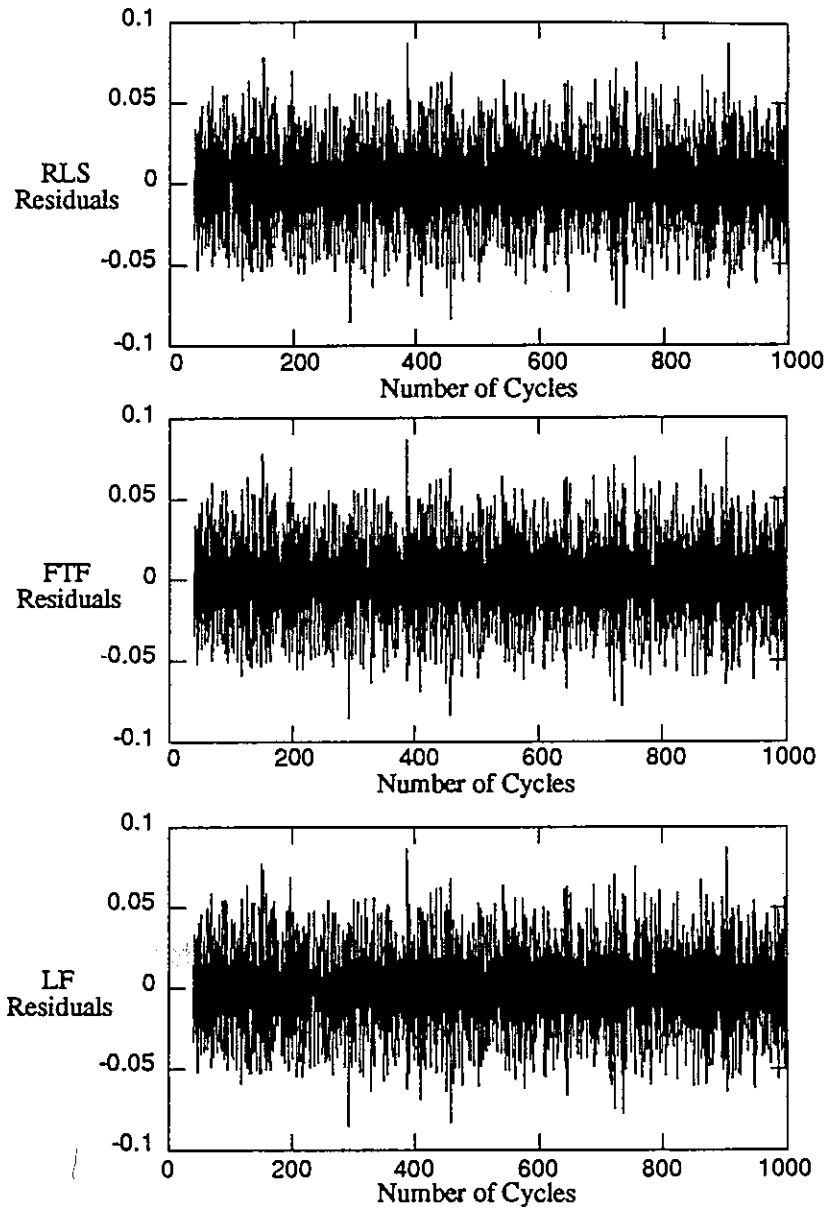


Figure 3.1.7 Residuals obtained using the RLS, FTF, and LF respectively

Figure 3.1.8 shows a plot of the measure output and the estimated output using the RLS.

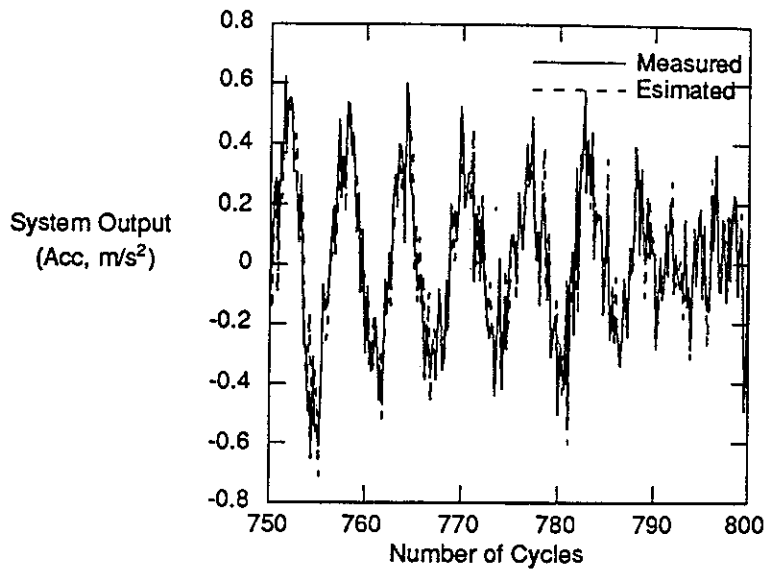


Figure 3.1.8 Estimated and measure output.

To show that the three methods converge to the same answer, Figure 3.1.9 shows the estimated output obtained from all three methods.

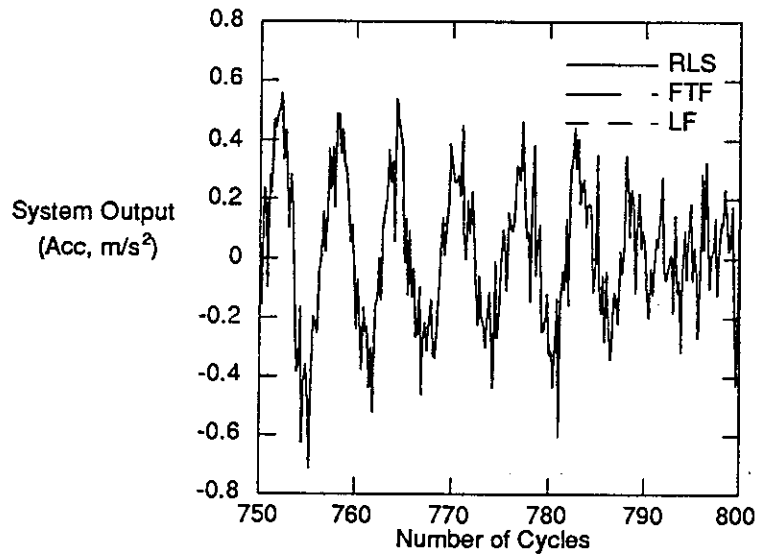


Figure 3.1.9 Estimated output obtained using the three methods.

System order is a parameter that may not be known a priori, but estimated using some form of goodness criterion. The value of a criterion or loss function is used to

determined at what point the order of the system should be increased to improve the model fit. There is going to be an order after which the loss function will not get smaller. At this point the Principle of Parsimony is applied. The principle states that the order of the system selected is the one in which the minimization of the loss function does not improve in a significant way for the upcoming orders. The loss function used in this study is defined as

$$\text{trace}[E_p(k)E_p^T(k)] = \text{trace}\left[\left(\Upsilon(k) - \bar{Y}V_p(k-1)\right)\left(\Upsilon(k) - \bar{Y}V_p(k-1)\right)^T\right] \quad (3.1.2)$$

which is the trace of the squared error. Figure 3.1.10 shows the plot of the loss function versus the system order for SISO systems when the noise is $N(0, 6\% Y_{\max})$.

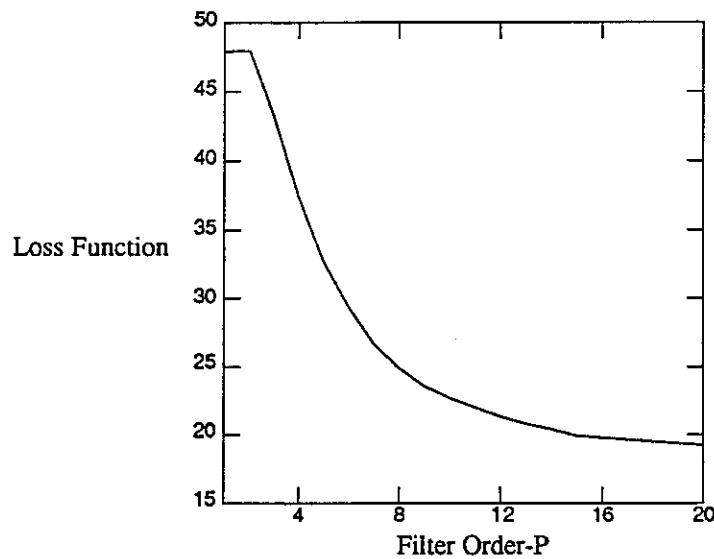


Figure 3.1.10 Loss Function vs Order of the filter

At this point the analyst must select a system order. Even with the aid of a loss function this could be a difficult task.

3.2. Complexity

The RLS is the simplest, in terms of number of operations, of the three methods to understand. To initialize the RLS one needs values for the OMP and the covariance matrix. These values can be assigned arbitrarily or estimated by a BLS. In the RLS no matrix inversion is involved, but updating the covariance matrix $P_p(k)$ and the gain vector $G_p(k)$ is time consuming. The RLS allows one to omit/skip data if the number of computation between sample time take longer than the sampling time. This property makes it attractive for on line applications, but the RLS is very slow for large p and omitting data may cause it to converge very slowly.

The FTF is the most complicated because it has the largest number of steps per recursion. To initialize the FTF it is necessary to run a BLS that requires the inversion of the covariance matrix. Figure 3.2.1 shows a data flow diagram for one recursion of the FTF.

The LF is suitable to run in a parallel machine. The LF consists of a number of stages connected in cascade with each stage in the form of a lattice. The number of stages in the LF equals the assumed model order, p . Thus, for a model order p there are p stages plus the initial stage. Since the LF uses forward-time and backward-time estimation in a single stage, the initial stage (when the model order $p=0$) is known as the Initialization of the Forward-Time Estimation (IFE) and the Initialization of the Backward-Time Estimation (IBE). Time-update takes a short time and the order-update is done from a lower to a higher order. As soon as the information is pass from a lower order to a higher order the algorithm is ready for the next time-update or a new data set. This makes the algorithm very attractive for on-line system identification. The LF structure can be seen in Figure 3.2.2.

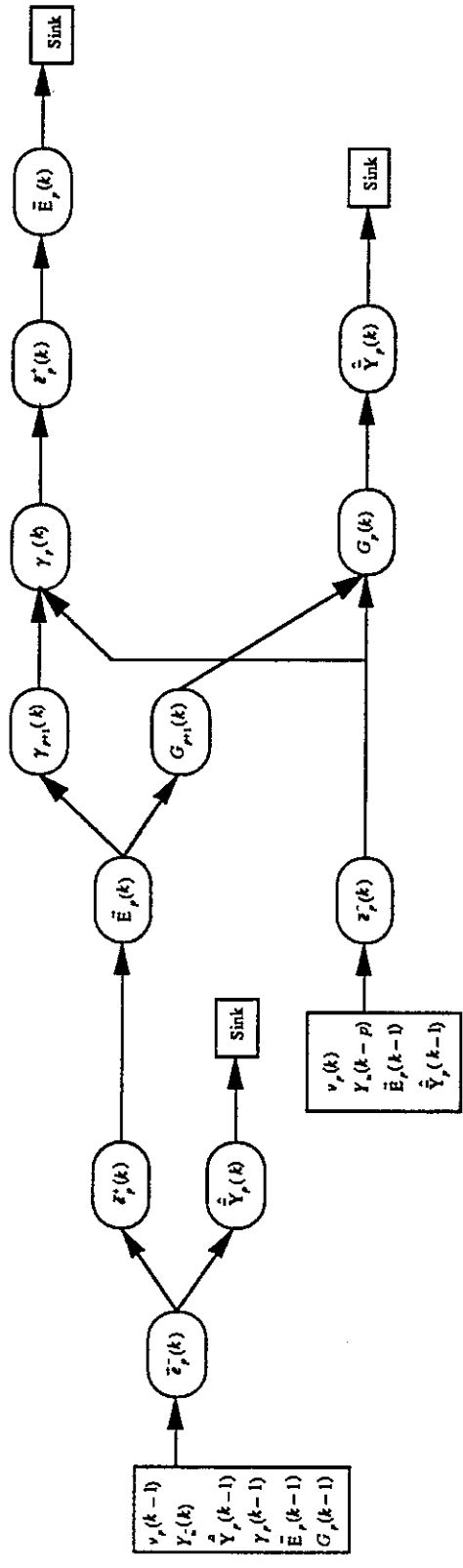


Figure 3.2.1 Data flow diagram of a FTF recursion

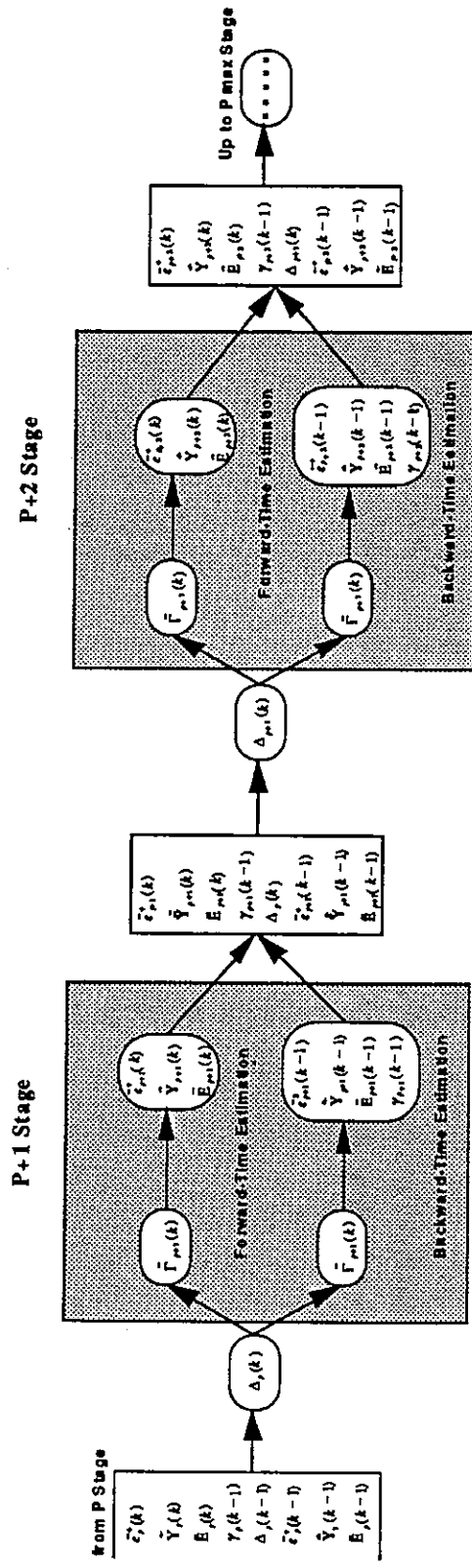


Figure 3.2.2 Data flow diagram of two stages of the LF

3.3. Computational Cost

To examine the computer requirements for each of the algorithms, Table 3.3.1 shows the order of the number of operations per recursion.

Table 3.3.1 Computational Cost of the Algorithms

Algorithm	Number of operations
Recursive least squares RLS	$O[(r+m)^2 p^2]$
Fast Transversal Filter FTF	$O[(r+m)^3]$
Lattice Filter LF	$O[(r+m)^3 p]$

From the table it is seen that the number of operations for the RLS increases quadratically with the assumed system order p . The number of operations for the Lattice Filter increases linearly with p and the order of the number of operations for the FTF does not depend on p . For systems with large number of inputs and outputs the LF and the FTF may not have an advantage over the RLS because the number of operation increases cubically with the number of inputs plus the number of outputs.

Another way to look at computational cost is to compare the time it takes each algorithm to process all the data in a simulation. Simulation results are computed using 5000 data points. The data used for the MIMO systems was obtained in a similar manner to the case of the SISO. For the MIMO systems more masses, dampers and springs were added as needed. All algorithms were coded in C language and ran on an IBM RS/6000. Timing studies were performed using the clock function of the system and running the program under super user mode. In this mode, processing time calculations are reliable because the system is dedicated completely to one user. Processing times are computer and software dependent but nevertheless provide a good reference for comparison.

Figure 3.3.1 shows results for a single-input single-output (SISO) system. Initially, when $p < 3$ the RLS is the fastest algorithm. However, for p values ranging from 4 to 15 the FTF is the fastest. The LF is slower than the RLS for orders up to order 10, but since the number of operations increase linearly in contrast with the RLS which increases quadratically, for orders higher than 10 the LF is the fastest.

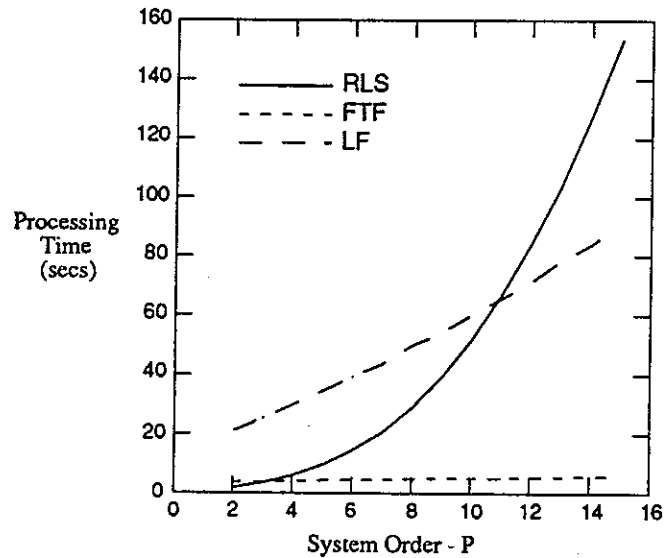


Figure 3.3.1 Comparison of processing time for a SISO (5000 data pairs)

Similar processing time, shown in Figure 3.3.2-3.3.3, are obtained for systems with one input / two outputs (SITO) and two inputs / two outputs (TITO). The FTF is the fastest of the three for all orders. SITO systems show that the RLS is faster than the LF for orders up to 8 but TITO systems showed the RLS is faster than the LF for $p < 7$.

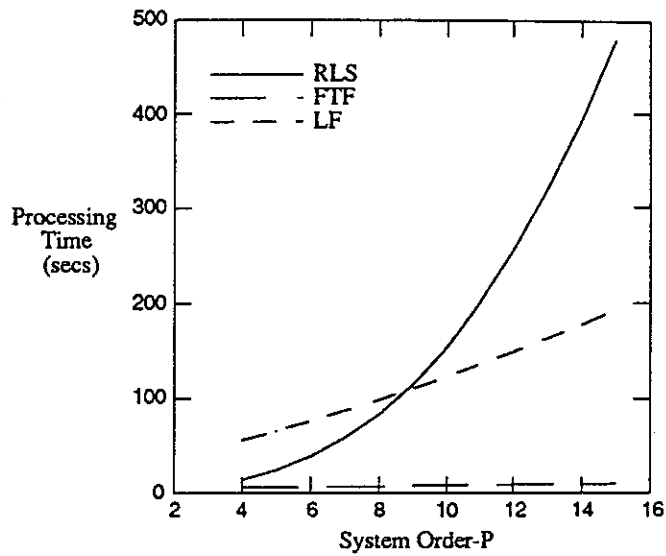


Figure 3.3.2 Comparison of processing time for a SITO (5000 data pairs)

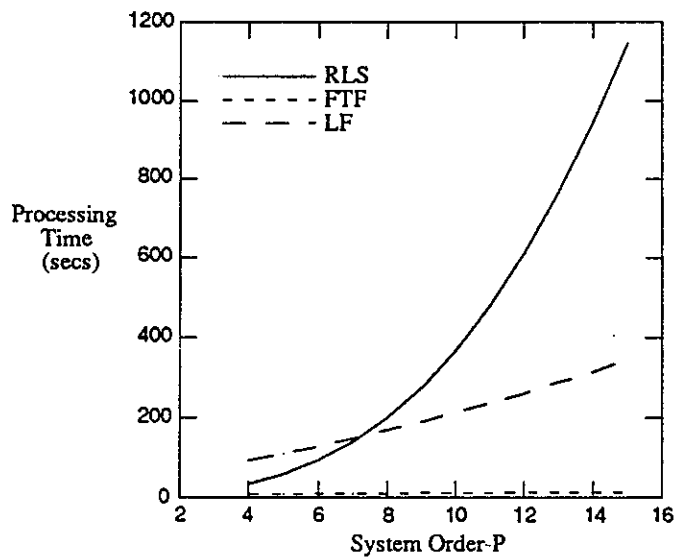


Figure 3.3.3 Comparison of processing time for a TITO (5000 data pairs)

Cases with three inputs / three outputs (3I/3O), depicted in Figure 3.3.4, showed the FTF as the fastest of the three algorithms for all orders. The RLS is only faster than the LF for orders of 6 or less.

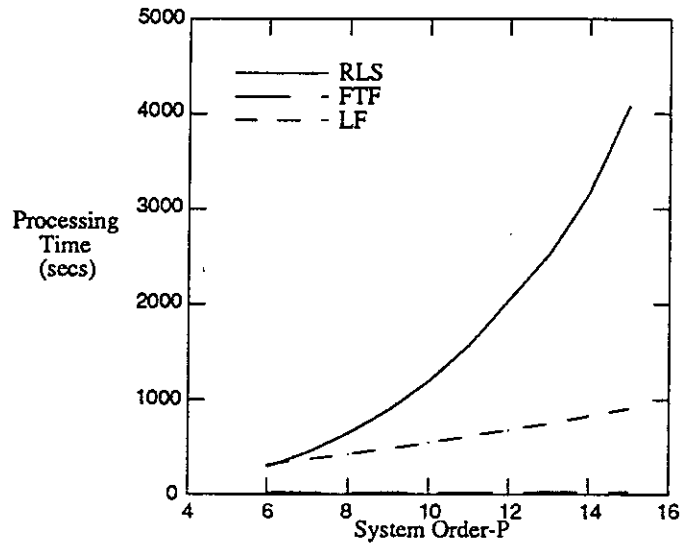


Figure 3.3.4 Comparison of processing time for a 3I/3O (5000 data pairs)

In summary, for cases with large filter order the RLS will be the slowest of all the approaches.

So far the numerical evaluation of the algorithms used simulated data to compare the approaches. Chapter 4 will present a description of the real-time experiment performed to evaluate the algorithms in a real application.

Chapter 4

Experiment Setup

This chapter will present an overview of the experiment setup. The first section presents a description of the Controls-Structure Interaction (CSI) Phase 3 Testbed. Section 4.2 gives some general information about the computer system used in the experiment. The Computer Automated Measurement and Control (CAMAC) instrumentation system is described in Section 4.3. Section 4.4 presents how these elements form the Real-time Control System Hardware. The last section presents a description of the experiment.

4.1. Testbed

The testbed, used for the experiments, is the CSI Evolutionary Model Phase-3 testbed (CEM3) which exhibits dynamics representative of the Earth Observation Satellite (EOS) AM-1 spacecraft. The CEM3 model consist of a spacecraft bus structure, flexible appendages, science instrument simulators and dummy masses to simulate both science payloads and spacecraft subsystems. The baseline EOS AM-1 spacecraft in its deployed configuration is shown in Figure 4.1.1. The spacecraft is comprised of a bus assembly and two flexible appendages which are the Solar Array (SA), the High Gain Antenna (HGA) and a number of scientific instruments sharing the same structural bus. The overall dimensions of the truss primary structure without the payloads or appendages are approximately 232 inches in length, 68 inches in width and 28 inches in height. The deployed SA is 351 inches in length with a blanket width of 196 inches, while the HGA is a slender cantilevered beam with a length of 100 inches [31].

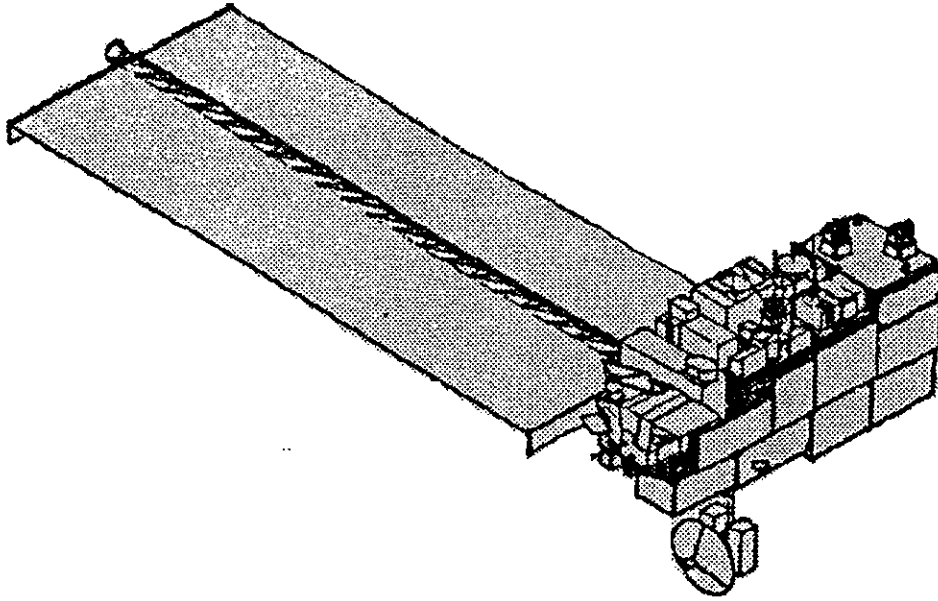


Figure 4.1.1. EOS AM-1 spacecraft

The CEM3 testbed, shown in the photograph Figure 4.1.2, is comprised of a truss structure, flexible appendages, payload mass simulators, three science instrument simulator, and control electronics. The science instrument simulator uses lasers, mounted on two axis gimbals to point to an advanced Optical Scoring Systems (OSS) (located on the floor of the laboratory) which provide a direct measurement of the incident angle of the laser beam.

CEM3 has two flexible appendages. The first one is a deployable articulated mast which is used to simulate the low-frequency dynamics of the EOS AM-1 SA. The mast is approximate one-half the length of the scaled EOS SA (180" Vs 351"). Even though the mast overall geometry does not closely match EOS AM-1, when a 40 lb. weight is put at the tip of the mast, it has approximately the same weight and center of gravity location as the EOS AM-1 SA [31].

The second flexible appendage is a HGA simulator which simulates the low-frequency dynamics of the EOS AM-1 HGA. All the important payloads and subsystems

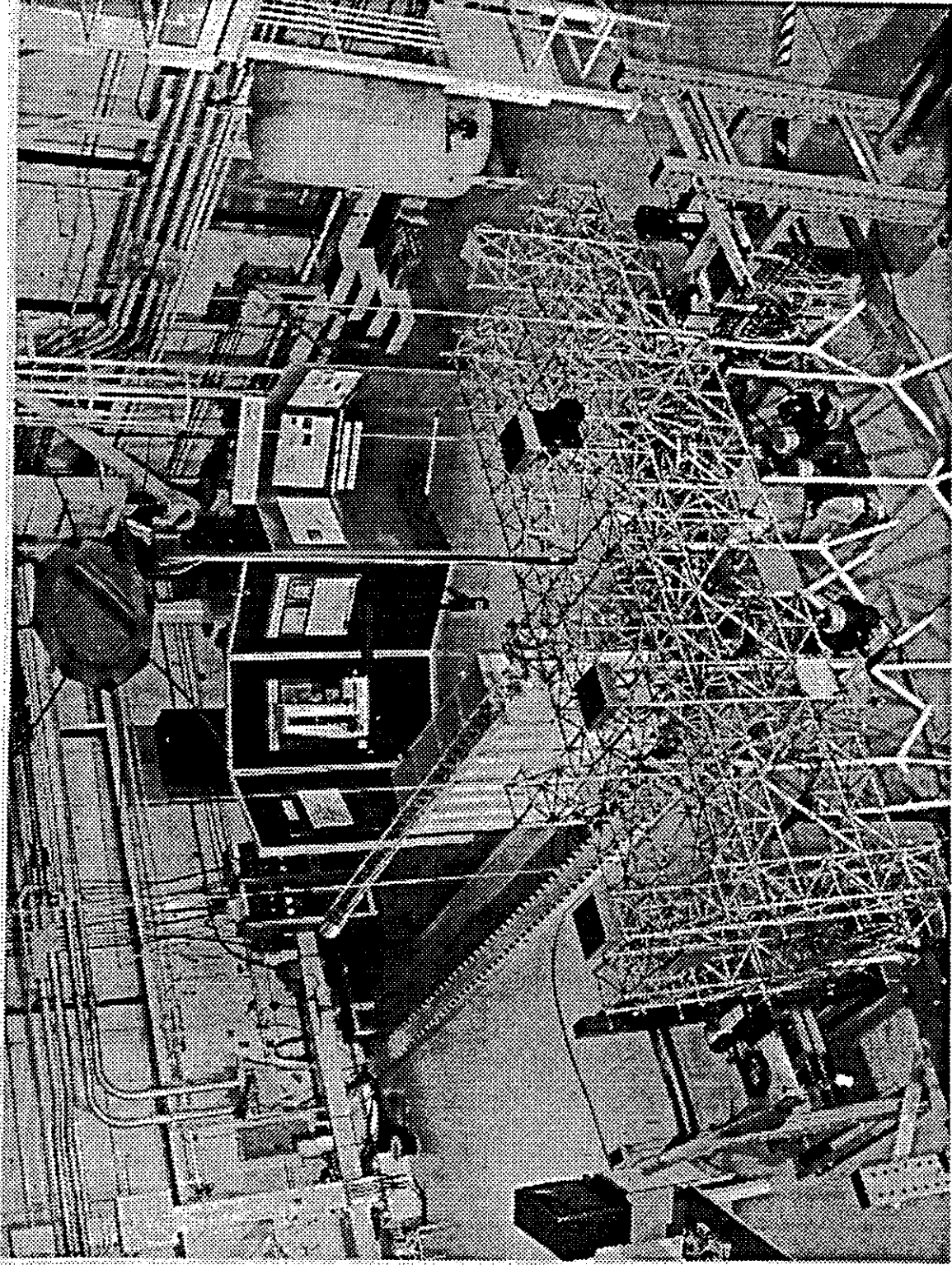


Figure 4.1.2 CEM3 Testbed

on the EOS AM-1 spacecraft were modeled on the CEM3 testbed using either dummy masses or a 2-axis gimbal with a mass payload.

The goals of the CEM3 testbed were to approximate the overall size, shape, inertia properties, first mode frequency, appendage bending mode dynamic interaction, and weight of a scaled EOS AM-1 spacecraft.

4.2. Computer Control System (IBM RISC System/6000)

The experiment control and data acquisition computer is an IBM RISC 6000. The RS/6000 family is based on a second generation RISC architecture called Performance Optimized With Enhanced RISC (POWER) architecture. The actual RS/6000 model used in the experiment is the POWER server 530H. The processor for this workstation uses the POWER architecture with a 33 MHz clock cycle. The performance of the POWER server 530H based on benchmark testing is 96.7 SPECfp and 29.2 SPECint [30]. This model executes up to two double-precision floating-point operations per clock cycle.

4.3. Instrumentation

The control computer is interface to the experiment via a Computer Automated Measurement and Control (CAMAC -IEEE-583), which is a worldwide standard for real-time interfacing. This standard provides for both stand-alone and distributed data acquisition and control systems. CAMAC [29] is an open-architecture concept that provides high-speed data throughput, a wide variety of I/O interface modules, computer independence and large variety of I/O modules.

The basic building block for any CAMAC system is the module, which is generally an interface between the external process and the CAMAC internal bus or Dataway. There is a wide variety of modules available, including digital and analog I/O, counters, queues,

stacks, communication interfaces, microprocessors, etc. Figure 4.2.1 shows a CAMAC module.

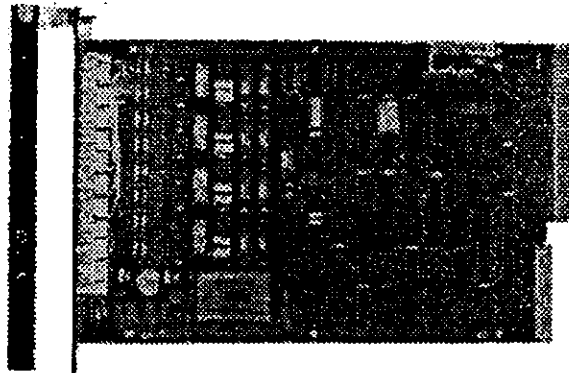


Figure 4.2.1 CAMAC module

The CAMAC crate provides the physical mounting, power, and cooling for the modules. The Dataway is at the rear of the crate. Full-size crates (see Figure 4.2.2) contains 25 modules slots.

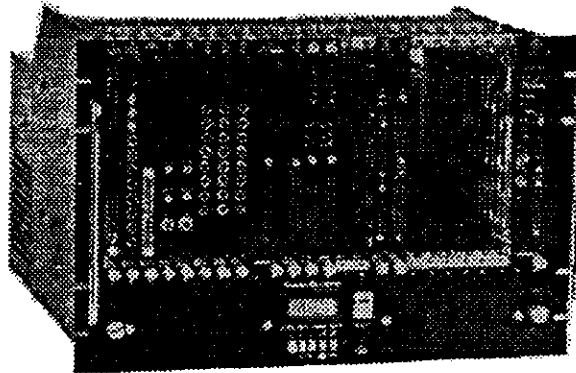


Figure 4.2.2 Full-size CAMAC Crate.

The crate controller resides in the right-hand slot of the crate and provides the hardware link between the I/O modules and the computer. The crate controller must address the modules with the appropriate CAMAC commands, transfer data, accept status information, and monitor the interrupt lines. A CAMAC crate controller can interface

directly to a computer bus, the standard CAMAC Parallel or Serial Highway or to a communication protocol such as RS-232 or GPIB.

4.4. Real-time Control System Hardware

4.4.1 Science Instrument Simulators (SIS)

The SIS are two axis gimbals driven by DC torque motors, a dummy weight as a load simulator, and an electronics control unit. The gimbal direct drive torque motors have optical encoders to measure gimbal position. Gimbal electronics using a 386SX processor controls the gimbal angular position to within arc-sec resolution. Communication with the host computer is via ARCNET at 1 KHz. The mounting and payload weight is 51.4 lbs and the gimbal and electronics is 46 lbs.

Figure 4.4.1 shows the gimbal connection to the computer system [28]. Each gimbal in the CEM3 testbed is connected to a CAMAC crate through a digital ARCNET, and the CAMAC crate is connected to the RS/6000 through a digital parallel k-bus.

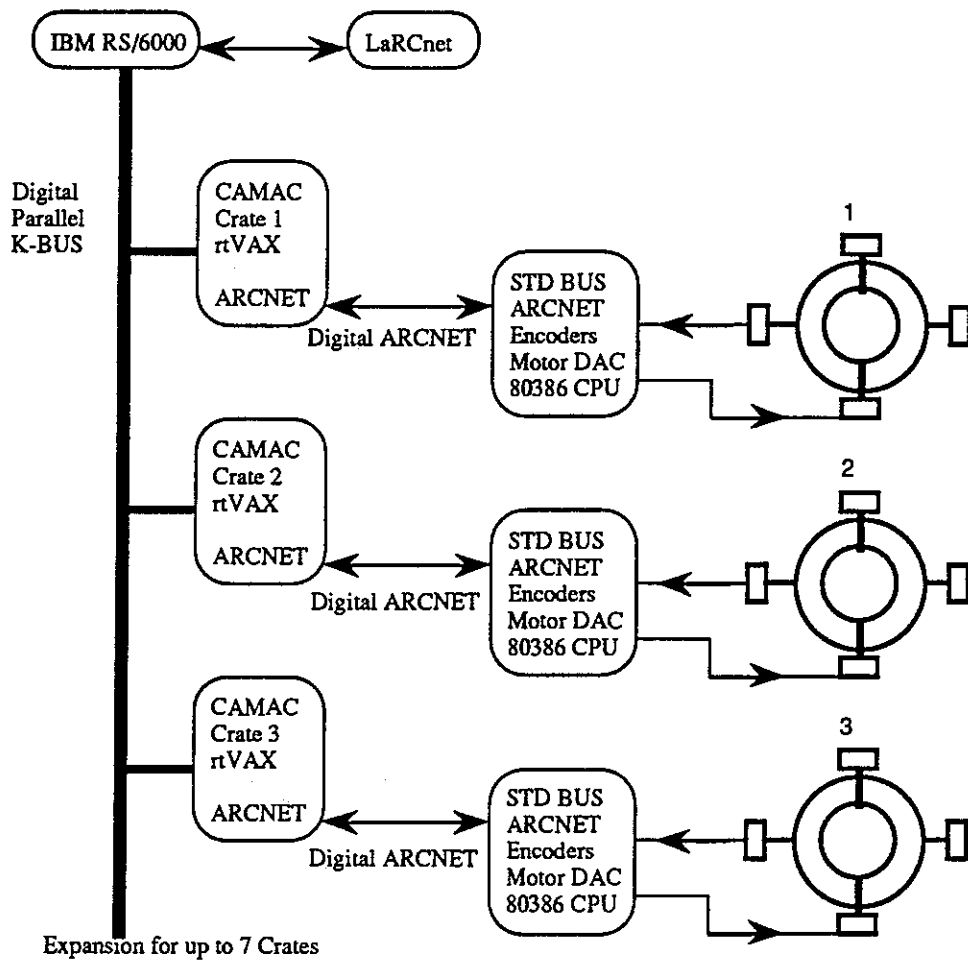


Figure 4.4.1 Gimbal Control System

4.4.2 Optical Scoring System

The OSS is part of the SIS whose function is to measure the incident angle of a laser beam as it enters the OSS optics. Laser beam translations are canceled by the OSS optics.

Figure 4.4.2 shows the OSS connection to the computer system [28]. There are five OSS in total and they are connected to the CAMAC crate through an ARCNET. The nominal range of the OSS is +/- 1000 arc-seconds.

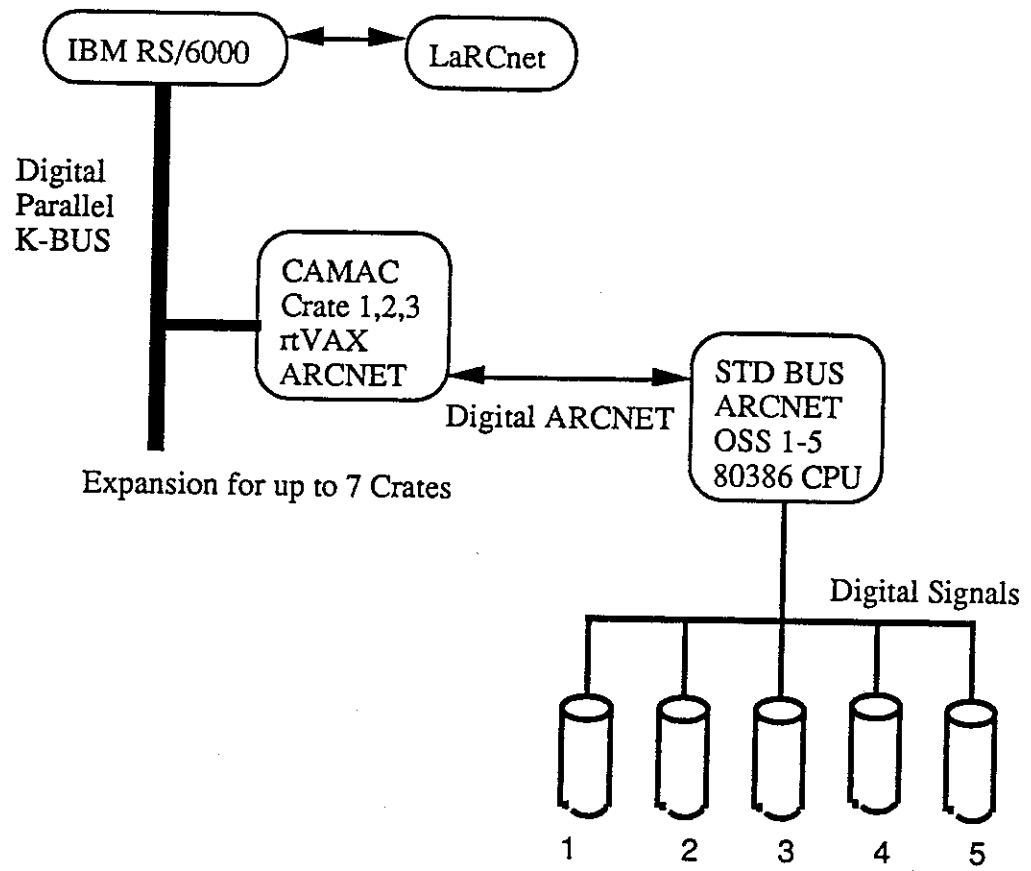


Figure 4.4.2 Optical Scoring System Connection

Combining all the subsystems into one makes the Real-Time Control System Hardware. Figure 4.4.3 shows a diagram of the Control system [28]. All the CEM3 subsystems are connected to the computer system through CAMAC crates.

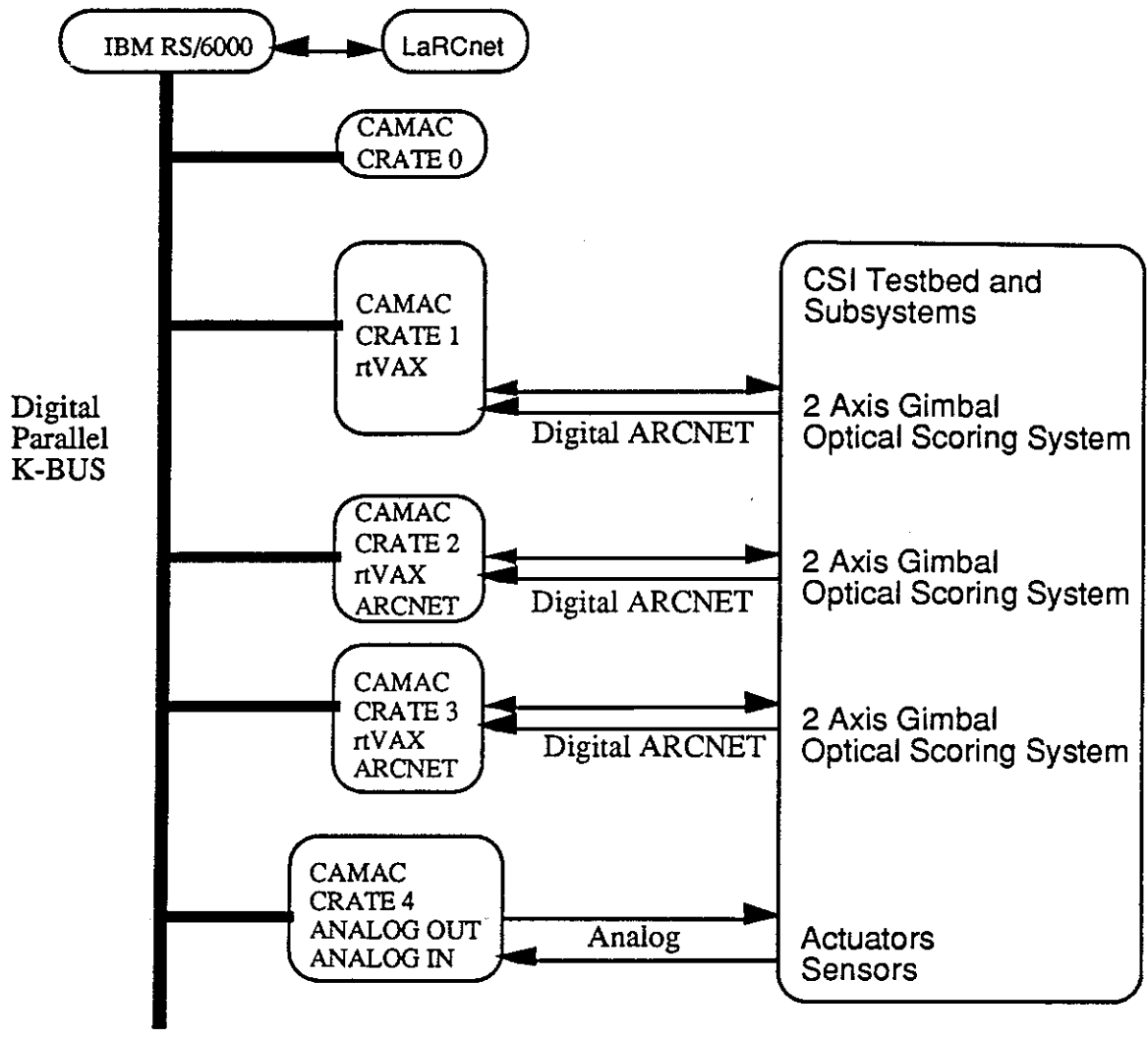
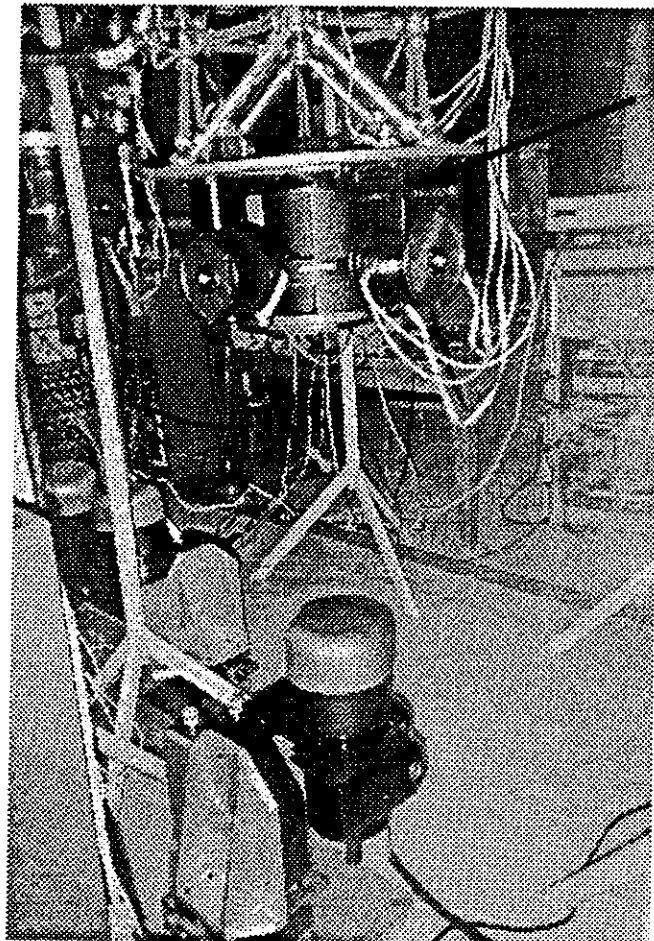


Figure 4.4.3 Real-Time Control System

4.5. Experiment Description

To study the on-line identification problem, one of the science instrument simulators was used. Figure 4.5.1 shows a photograph of the gimbal used in this study. Gimbal motor commands were used as inputs and the two OSS angular positions were used as outputs. Figure 4.5.2 shows a flowchart of the experiment.



Gimbal

Figure 4.5.1 Gimbal in CEM3 testbed

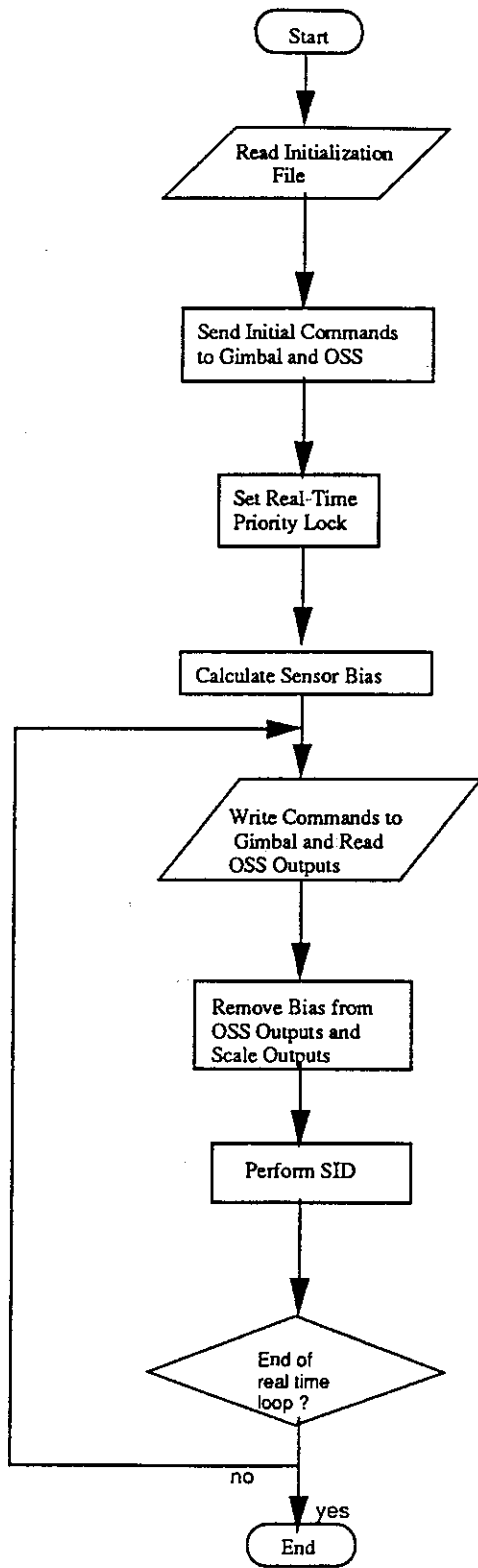


Figure 4.5.2 Flowchart of the Experiment

The initialization file contains information on the total time of the experiment, sampling rate, excitation on time, excitation off time, SID on time, order of the filter, etc. Other parameters defined in the initialization file are the crate used for data transfer and the ADC card in the crate from which to start receiving data. The experiment total time is 20 seconds, sampling at 60 Hz. The gimbal is being excited with a random input. The on-line SID is started 4 seconds after the experiment is started to allow the data from the OSS to be scaled. The on-line SID is continue until the end of the experiment. After the initialization file is read, the gimbals and OSS are initialize and the computer is set to work in Real-Time. The first task is to calculate sensor bias followed by the real-time loop which last for 16 seconds. During the real-time loop, commands are sent to the gimbals (random input or actual desired control input) and the OSS outputs are read to perform the RSID. The experiment was performed using the RLS, FTF and the LF. One objective is to implement all the algorithms in a real-time system to perform a realistic timing study. Parameters identified on-line can then be used for controls as discussed in Ref. [3]. The control part is beyond the scope of this thesis, but is the subject of future work. For on-line controls, if the algorithm computations take longer than one sampling time, clock overrun occurs and data is out of sync with the rest of the data. For control applications this could cause instabilities.

This chapter presented the experiment setup. A description of the testbed, computer system, and instrumentation system used in the experiment was presented. The next chapter will cover experimental evaluation and results obtained from the three methods will be shown.

Chapter 5

Experimental Results

This chapter compares the algorithms results obtained from real-time experiments. The algorithms are compared in term of convergence rate, computational speed and the estimated outputs of the three methods are compared to see if all converge to the same values.

Table 5.1 shows the maximum filter order that each algorithm can estimate without clock overruns using a sampling rate of 60 Hz.

Table 5.1 Maximum Filter Order (60 Hz)

Algorithm	Filter Order
RLS	5
FTF	60
LF	2

Consistent with simulated results the FTF is the fastest of the three algorithms. The LF is the slowest in this implementation because the parallel structure of the algorithm is not exploited. Timing computations for higher values of p are not possible with the current configurations. Hardware changes to incorporate digital signal processors (DSP) is being pursued, but it is not available at this time.

Since the tracking error for a filter order of 2 is small and in order to compare the three algorithms the experiments were performed using $p=2$.

Figure 5.1 shows the first five OMP identified using the RLS, FTF and LF. The OMP identified using the FTF converged immediately because the algorithm is initialize using a BLS. The OMP obtained using the RLS converges a little faster than the ones obtained with the LF.

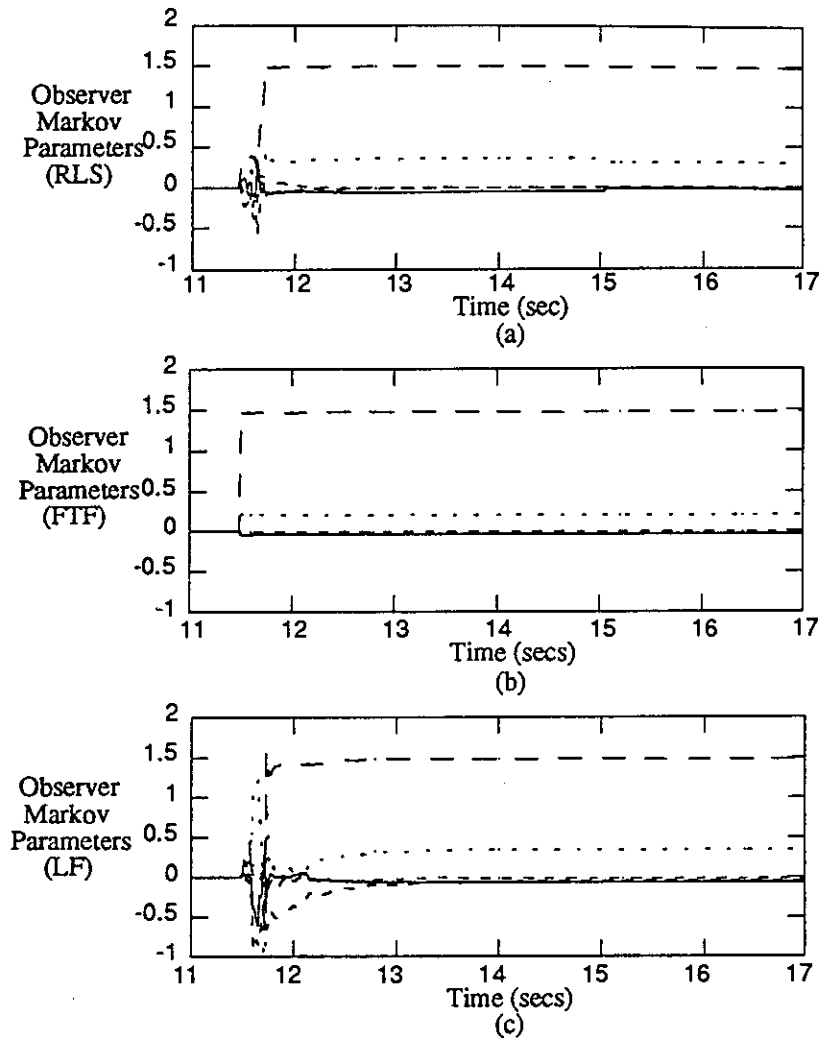


Figure 5.1 Convergence of the Observer Markov Parameters using RLS, FTF and LF.

The estimated outputs obtained using the identified parameters are compared to those measured using the OSS, these outputs are referred to as elevation and azimuth angles measured in units of arc-seconds. The RSID is started at 11.5 seconds after the excitation is turned-on. Figure 5.2 shows the estimated, using RLS, and measured elevation angles. Figure 5.2.a shows the first 1.5 seconds after the RSID is started. It took approximately 0.3 seconds for the solution to converge, which is about 18 function calls to the RLS. Figure 5.2.b shows the last 3 seconds of the experiment. For

demonstration purposes, the control procedure in Ref. [3] was implemented using the parameters identified with the RLS. At time $t=15$ seconds the control part of the program is started and that caused a clock overrun which resulted in a small difference between the measured and the estimated elevation.

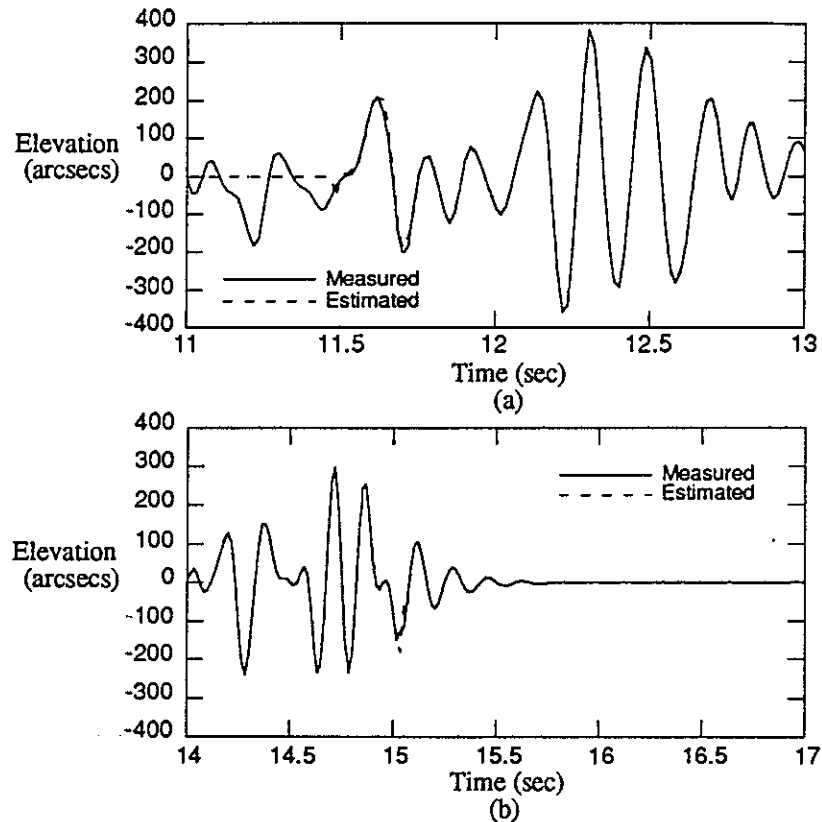


Figure 5.2 Comparison of estimated and measured elevation angle using RLS
a. Beginning of Experiment
b. End of Experiment

Figure 5.3 shows a comparison of the estimated and measured azimuth obtained using the RLS. In Figure 5.3.a the first 1.5 seconds are shown. It took also about 0.3 seconds for the solution to converge. Figure 5.3.b shows the last 3 seconds of the experiment. At that point the fit is almost perfect.

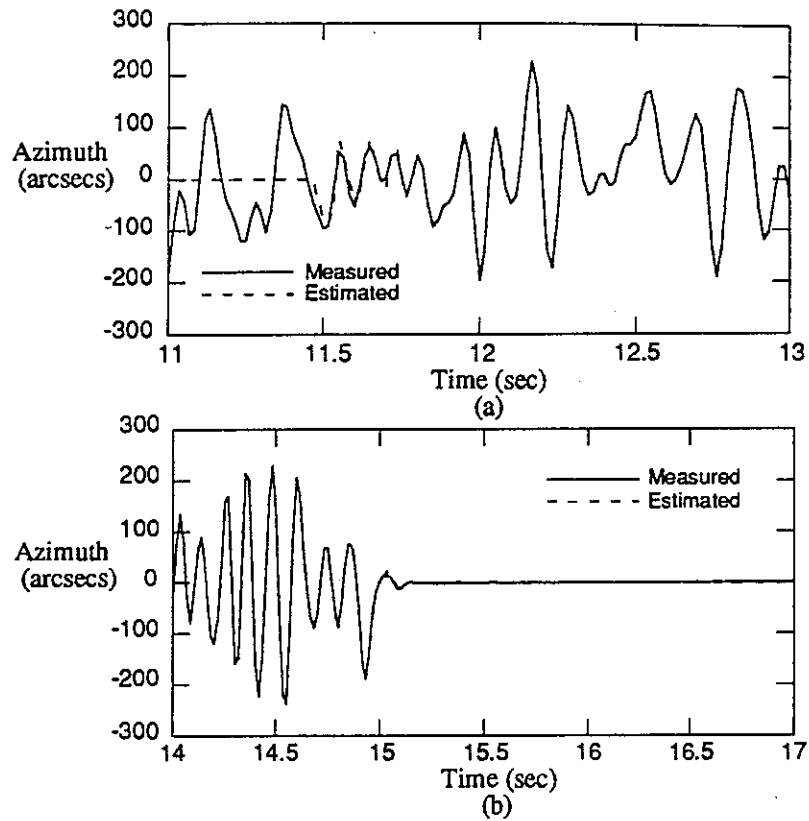


Figure 5.3 Comparison of measured and estimated azimuth angle using RLS
a. Beginning of Experiment
b. End of Experiment

When the FTF is used, BLS solution is used to initialize the parameters algorithm. Performing the BLS in the experiment is time consuming and causes a clock overrun during this period of time. When clock overruns occurs, data gaps are introduced making the data out of sync. Figure 5.4 shows the estimated and measured elevation obtained using the FTF. Figure 5.4.a shows the first 1.5 seconds after the RSID is started. Since the initial parameters used are close to the real ones it takes about 3 time steps or 0.05 seconds to converge. Figure 5.4.b shows the last 3 seconds of the experiment.

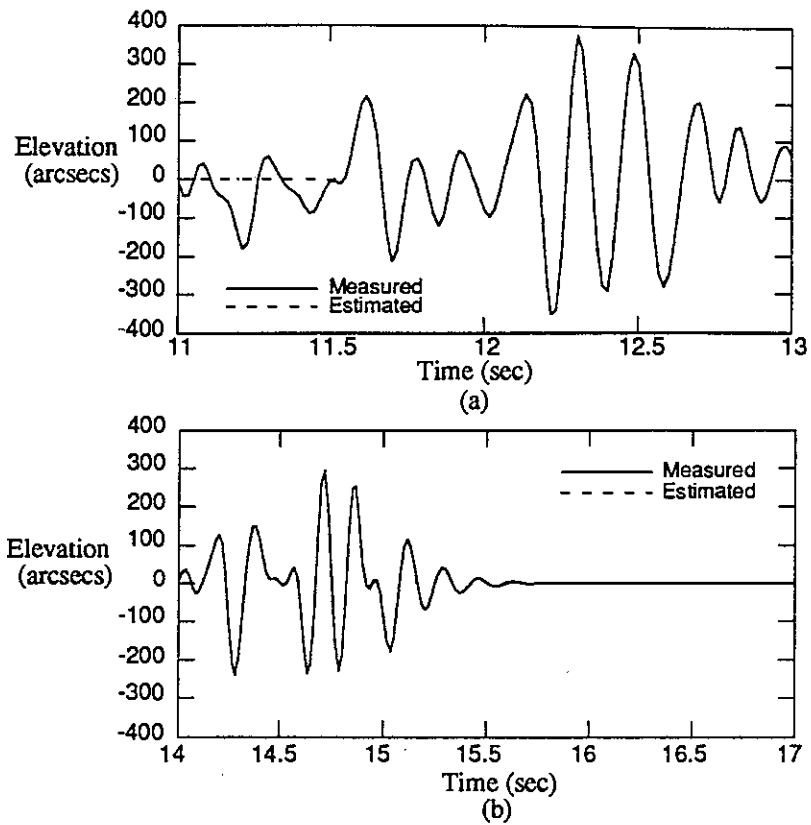


Figure 5.4 Comparison of measured and estimated elevation angle using FTF
 a. Beginning of Experiment
 b. End of Experiment

Figure 5.5 shows a comparison of the estimated and measured azimuth angles using the FTF. The first couple of seconds after the RSID started are shown in Figure 5.5.a. For the azimuth angle it took more time for the solution to converge than for the elevation angle, but the FTF still converges faster than for the RLS and the LF. Figure 5.5.b shows the last 3 seconds of the experiment.

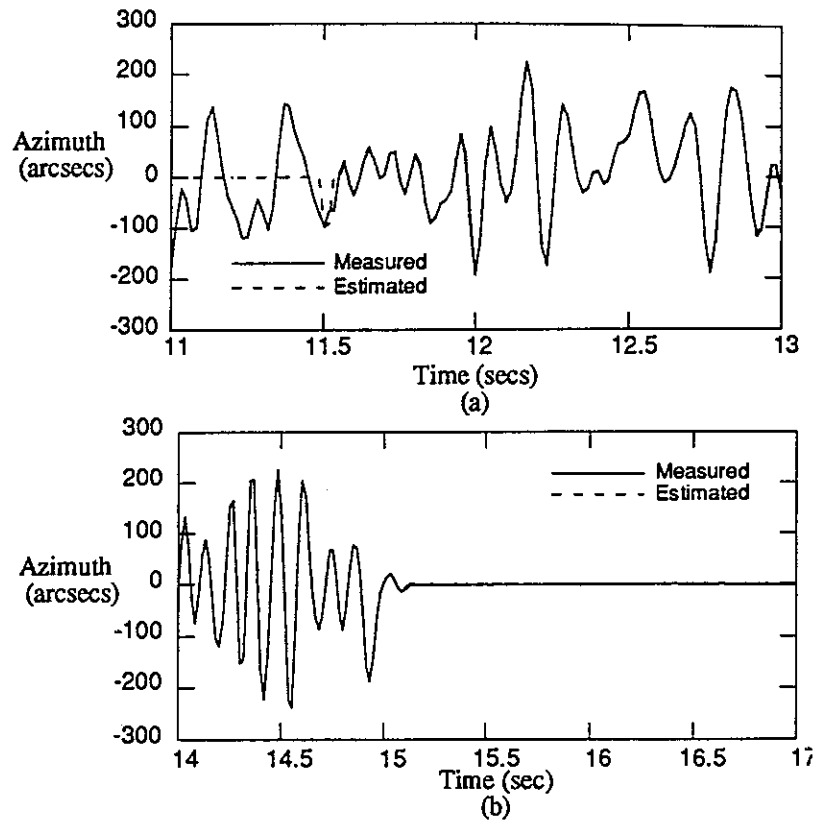


Figure 5.5 Comparison of measured and estimated azimuth angle using FTF
 a. Beginning of Experiment
 b. End of Experiment

The LF is the slowest one to converge, but it still showed a small tracking error. Figure 5.6 shows a comparison of the estimated and measured elevation angles using the LF. Note that it takes longer for the solution using LF to provide a good prediction. Figure 5.6.b shows the last 3 seconds of the experiment.

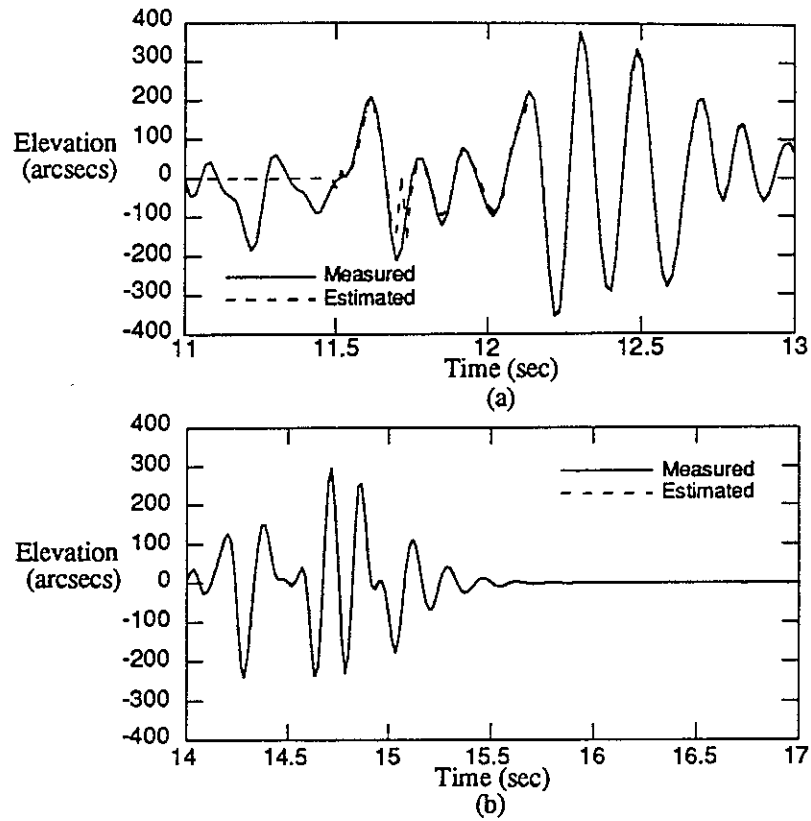


Figure 5.6 Comparison of measured and estimated elevation angles using LF
 a. Beginning of Experiment
 b. End of Experiment

Figure 5.7 shows a comparison of the estimated and measured azimuth angle using the LF.

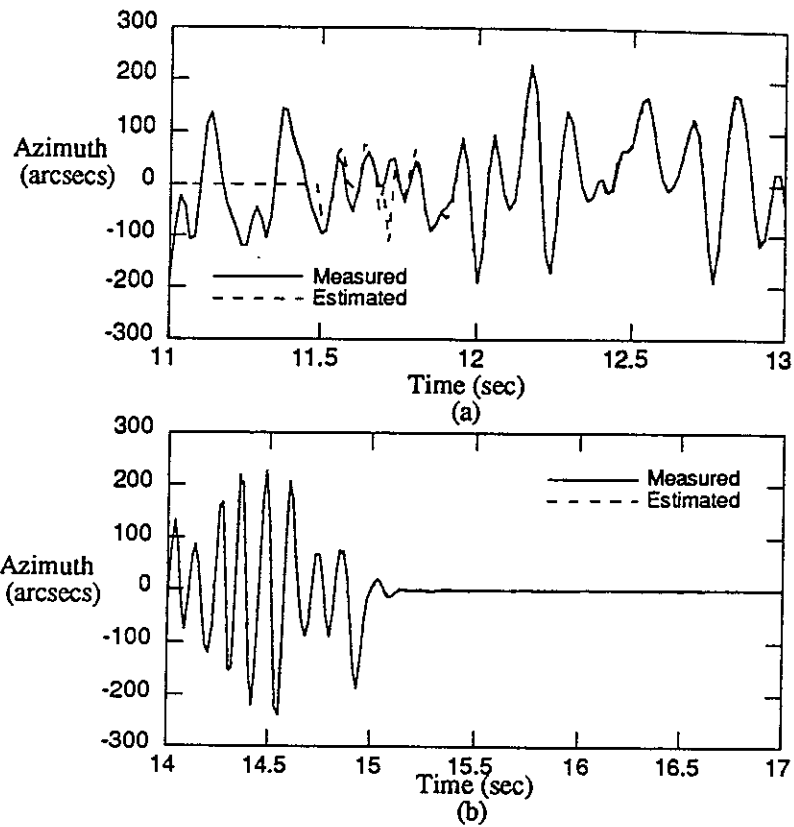


Figure 5.7 Comparison of estimated and measured azimuth angle using LF
 a. Beginning of Experiment
 b. End of Experiment

The residuals or tracking error obtained using all three algorithms are shown in Figure 5.8. Figure 5.8.a shows the residuals obtained when using the RLS. At time $t=15$ seconds a clock overrun occurs resulting in a large error spike at that time. After that, the algorithm converge again producing a small error.

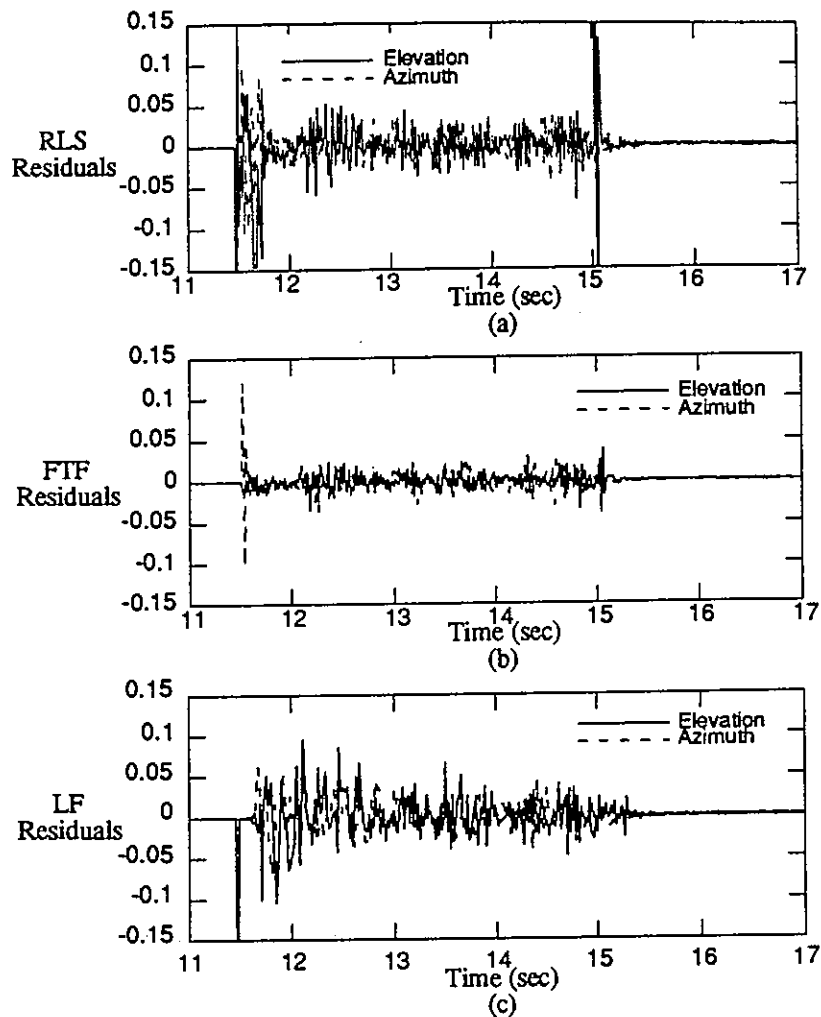


Figure 5.8 Tracking error for all three algorithms

In order to compare the estimated output obtained using the three algorithms an off-line simulation of the experiment is performed with data recorded from previous experiments. This way the three algorithms are using exactly the same input and output data. Figure 5.9 shows the estimated elevation and azimuth angles, respectively. No appreciable differences are observed from all three solutions.

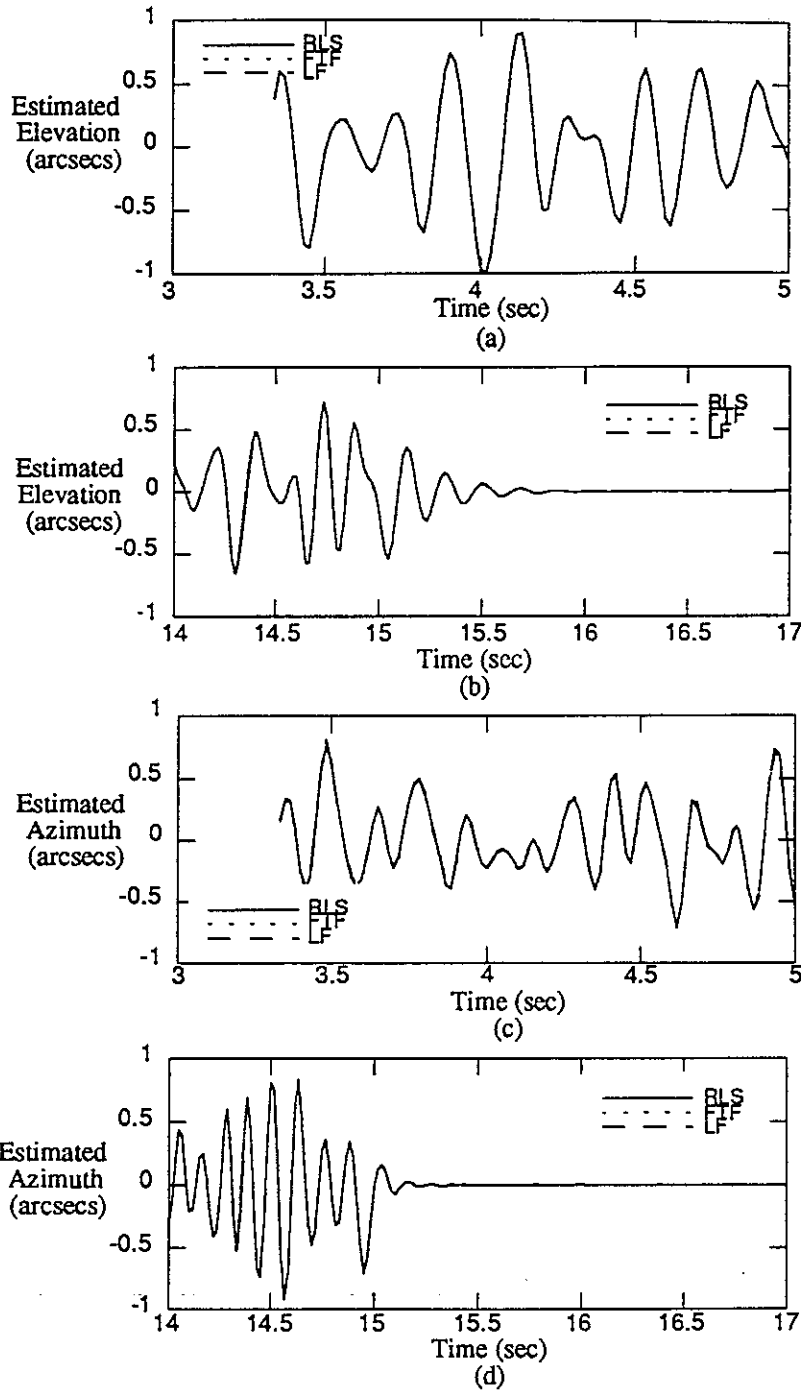


Figure 5.9 Estimated elevation/azimuth angles for all three algorithms
 a,c. Beginning of Experiment
 b,d. End of Experiment

The outputs shown in Figure 5.9 are scaled based on their maximum value.

To initiate the on-line estimation process, one must first determine/estimate the system order. Earlier, order determination was performed based on the Principle of Parsimony along with a loss function.

Figure 5.10 shows the loss function vs. filter order for one of the science instrument simulator. This model order determination is done off-line and is just to give an idea of the value of the order of the system.

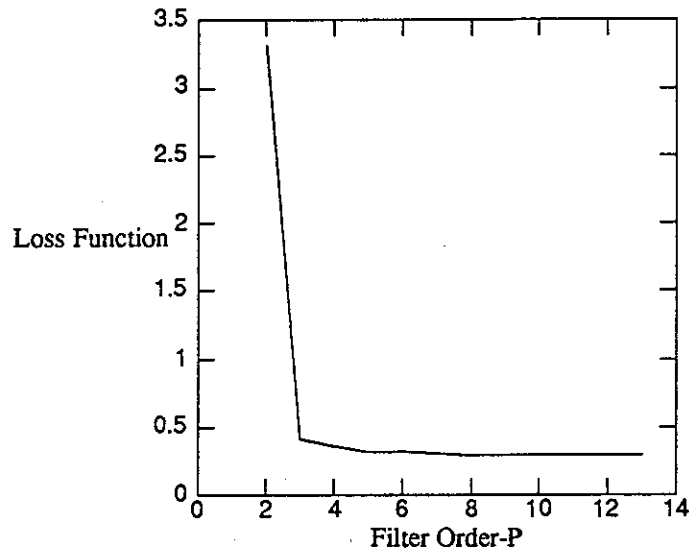


Figure 5.10 Loss Function vs. Filter Order

System orders equal to or greater than five will not further reduce the loss function. Varying the filter order, one can compute the time required for each algorithm to complete all calculations at each time step.

This chapter presented the results obtained from a real-time experiment. It was shown that the FTF is the fastest of the three algorithms. The convergence rate for the FTF is also faster than for the other two because of the initialization with the BLS. All of the algorithms converged to the same answer.

Chapter 6

Conclusions

6.1 Concluding Remarks

Most existing work in the literature dealing with on-line identification concentrated on the mathematical formulation of the algorithms, but applications to real systems are rarely considered. When on-line algorithms are programmed and experimentally implemented, issues regarding limitations and potential improvements are often discovered. A major contribution of this work is the experimental implementation of three identification algorithms to perform on-line least squares parameter estimation using computer equipment which is typical of systems used for on-line identification. The three algorithms discussed are Recursive Least Squares (RLS), Lattice Filters (LF), and Fast Transversal Filters (FTF). Issues such as input/output timing studies, processing time limitations, order determination, data gaps and some others have been discussed to a some extent. In addition, data from analytically simulated systems have been analyzed to study the numerical properties and noise effects of the different approaches.

Of the three algorithms studied the RLS is the simplest, in terms of number of operations. It was shown that when the assumed filter order is small, RLS is faster than LF. It is also faster than the FTF for SISO systems with filter orders less than 3. Updating the covariance matrix and the gain vector is the most time consuming computation of the RLS. The number of operations performed by RLS increases quadratically with the assumed filter order and number of inputs plus outputs. Data gaps are tolerated well by the RLS algorithm, which can be important when calculations are not completed in between samples, however, slower convergence rates are observed. For control purposes, data gaps can cause instabilities of the control system and should be avoided. For slowly

varying systems, data gaps may not be a problem; where slow is relative to the implementation sampling rate. Using an IBM 6000 and CAMAC interface, the RLS algorithm allowed a maximum assumed system order less than 5 at a sampling rate of 60 Hz while commanding two-inputs and sensing two-outputs.

FTF, which has the least number of operations of all three algorithms, is the fastest. In this formulation, the number of computer operations increases as a cubic function of the number of inputs plus the number of outputs, and the assumed system order has a secondary effect on the number of operations. In most practical applications, the number of inputs and outputs is fixed and small but the assumed filter order is usually unknown. Tradeoff studies are routinely performed varying the assumed filter order and looking at prediction error. If the computations are to be performed on-line, the FTF algorithm is the recommended approach. For demonstration purposes, estimated parameters using the FTF algorithm were used successfully with an adaptive control algorithm to control a two-axis gimbal system.

Although the LF algorithm did not perform well in the test cases considered, it should be pointed out that its structure is more suitable for multi-processor machines. Initially, a multi-processor computer was going to be available to evaluate this approach but unfortunately the equipment was not ready in time for the study. The LF algorithm consists of multiple stages connected in cascade with each stage in the form of a lattice. The number of operations involved in the LF algorithm increases linearly with the assumed filter order and cubically with the number of inputs plus the number of outputs of the system. The experimental implementation of the LF allowed a maximum assumed filter order of 2 at sampling rate of 60 Hz. However, the software developed will be used at a later time with a multi-processor machine.

All approaches showed significant bias in the parameter estimates when noise was in the data. Since all the approaches are least square based, selection of a model structure must take into account explicitly the noise contribution entering into the problem. This

aspect of the problem was not treated at this time because the models obtained from on-line parameter estimation are intended to be used with on-line control algorithms. For control applications, good tracking performance is critical and all the approaches studied performed well in this area.

All algorithms considered here have been programmed in C language and MATLAB. Listing of the files are included in the appendix and are currently part of the real time computer software at NASA Langley.

6.2 Future Studies

Implementation of all three algorithms in a multi-processor computer should be considered. As mentioned earlier, the LF algorithm is ideal for multi-processors machines. LF consists of a number of stages connected in cascade, with each stage in the form of a lattice. The LF uses the Forward-Time and Backward-Time estimation in a single stage; these two task can be performed in parallel. Each stage, in the LF, is comprised of all necessary computations to increase the model order by one. Therefore, the number of stages equals the assumed system order plus 1. When the information of a lower stage (order) is pass to the next stage (or next order) the previous stage is ready for new data. This process is known as pipelining. Conceptually each stage could be assigned to a separate processor but within each stage the task can be subdivided further in terms of Forward-Time Estimation and Backward-Time Estimation. With this approach one would take advantage of the parallel structure of a multi-processor system and as well as pipelining which should provide significant speed improvements. The number of operations still will be of order $O[(r+m)^3p]$ but it will seem as if the order is $O[(r+m)^3/2]$. Figure 3.2.2 shows a data flow diagram of two stages of the LF.

The FTF can also be implemented in a parallel machine because the algorithm has some computations that can be done in parallel. Implementing the FTF in this way will reduce the time it takes to do a recursion in a serial machine. Figure 3.2.1 shows a data flow diagram of a FTF recursion.

A parallel implementation of the RLS will not improve that much the timing of a recursion because the two computations (update of $G_p(k)$ and $P_p(k)$) which are time consuming are serial and can't be done in parallel.

Another aspect to study is the use of forgetting factors which are used to reduce the influence of old data in the parameter estimates.

Chapter 7

References

- [1] Chen, C.-W., Juang, J.-N., "Several Recursive Techniques for Observer/Kalman Filter System Identification from Data," *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Paper No.92-4386, Hilton Head, SC, August 1992.
- [2] Juang, J.-N., *Applied System Identification*, Prentice Hall, Inc, Englewood Cliffs, NJ, 1994.
- [3] Horta, L.G., Sandridge, C. A., "On-line Identification of Forward/Inverse Systems for Adaptive Control Applications," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Paper No. AIAA-92-4384 Hilton Head, SC August 1992.
- [4] Soderstrom, T., Stoica, P., *System Identification*, Prentice Hall, Inc, Englewood Cliffs, NJ, 1989.
- [5] Haykin, S., *Adaptive Filter Theory*, Prentice Hall, Inc., Englewood Cliffs, NJ 1986.
- [6] Graupe, D., *Identification of Systems*, Litton Educational Publishing, Inc., 1972
- [7] Ogata, K., *Modern Control Engineering*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1984.
- [8] Ogata, K., *Discrete-Time Control Systems*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1987.
- [9] Horta, L.G., Phan, M., Juang, J.-N., "Frequency Weighted System Identification and Linear Quadratic Controller Design," *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Paper No. AIAA-91-2733-CP, August 1991.
- [10] Phan, M., Horta, L.G., Juang, J.-N., "Linear System Identification Via an Asymptotically Stable Observer," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Paper No. AIAA-91-2734, New Orleans, LA, August 1991.
- [11] Juang, J.-N., Phan, M., Horta, L.G., "Identification of Observer/Kalman Filter Markov Parameters: Theory and Experiment," *Proceedings of the AIAA Guidance,*

Navigation, and Control Conference, Paper No. AIAA-91-2735, New Orleans, LA, August 1991.

[12] Chen, C.-W., Huang, J.-K., Phan, M., Juang, J.-N., "Integrated System Identification and Modal State Estimation for Control of Flexible Space Structures," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Paper No. 90-3740-CP, Portland, OR, August 1990.

[13] 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.

[14] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, P., *Numerical Recipes in C*, Cambridge University Press, 1992.

[15] Walpole, R.E., Myers, R.H., *Probability and Statistics for Engineers and Scientists*, Macmillan Publishing Company, 4th ed., 1989.

[16] Strang, G., *Linear Algebra and its Applications*, Harcourt Brace Jovanovich, Inc., 3rd ed., 1988.

[17] Hsia, T.C. *System Identification Least Squares Method*, D.C. Heath and Company, 1977.

[18] Goodwin, G.C., Payne, R.L., *Dynamic System Identification Experiment, Design and Data Analysis*, Academic Press, 1977.

[19] Goodwin, G.C., Sin, K.S., *Adaptive Filtering Prediction and Control*, Prentice Hall, Inc, Englewood Cliffs, NJ, 1984.

[20] Ljung, L., *System Identification: Theory for the User*, Prentice Hall, Inc., Englewood Cliffs, NJ 1987.

[21] Phan, M., Juang, J.-N., Longman, R.W., "On Markov Parameters in System Identification", NASA Technical Memorandum 104156, January 1991.

[22] Phan, M., Juang, J.-N., Longman, R.W., "Identification of Linear Multivariable Systems from a Single Set of Data by Identification of Observers with Assigned Real Eigenvalues," *AIAA 32nd Structures, Structural Dynamics & Materials Conference*, Paper No. 91-0949, Baltimore, MD., April 1991.

[23] Kuo, B.C., *Digital Control Systems*, Holt, Rinehart and Winston, Inc., 1980.

- [24] Kailath, T., *Linear Systems*, Prentice Hall, Inc, Englewood Cliffs, NJ, 1980.
- [25] Lee, D.T., Morf, M., and Friedlander, B., "Recursive Least-squares Ladder Estimation Algorithm", *IEEE Transaction on Circuits and System*, Vol. CAS-28, 1981.
- [26] Piñeiro, L.A., and Biezad, D.J., "Real-Time Parameter Identification Applied to Flight Simulation," *IEEE Transaction on Aerospace and Electronics Systems*, Vol 29, No.2 , April 1993.
- [27] Kashangaki, T.A., "On-Orbit Damage Detection and Health Monitoring of Large Space Trusses - Status and Critical Issues," *32nd Structures, Structural Dynamics and Material Conference*, Baltimore, MD, April 1991.
- [28] Ugoletti, R., and Sulla, J., "CSI Testbed Documentation IBM/RS6000 Realtime Software Libraries," Vol. 2. Reference in the Control Room of the Space Structure Lab. in NASA Langley Research Center, Hampton, VA.
- [29] Clearly, R.T., "CAMAC, an Ideal High-Performance Data Acquisition Standard", Kinetic Systems Corporation, Lockport, IL, Dec. 1989.
- [30] Hoskins, J., *IBM RISC SYSTEM/6000 A Business Perspective* , John Wiley & Sons Inc., 3rd ed. , 1993.
- [31] Gronet, M.J., Davis, D.A., and Tan, M.K., "Development of the CSI Phase-3 Evolutionary Model Testbed" Lockheed Missiles & Space Company, Inc. Prepared for Langley Research Center under Contract NAS1-19241.

Chapter 8

Appendix

8.1.1 Recursive Least Squares Function

```
/* rls.c */

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

void rls(float *u, float *y, int m, int r, int p, double **Yhat,
         double **Pmat, float **uold, float **yold, double *err,
         int dimVp, double *vp, double *yh, float lami[], float lamf)
```

```
/* This subroutine updates the markov parameters using
the classical recursive least squares algorithm.
```

Variables:

Input:

m :- # of outputs
r :- # of inputs
p :- system order
dimVp :- (r+m)*p + r
u :- input at time k <r x 1>
y :- output at time k <m x 1>
uold :- previous values of inputs <r x p>
yold :- previous values of outputs <m x p>
lamf :- final value of forgetting factor

Input/Output:

lami :- initial value of forgetting factor
Pmat :- covariance matrix <dimVp x dimVp>
Yhat :- estimated parameters <m x dimVp>

Output:

yh :- estimated output <m x 1>
vp :- input/output data vector <dimVp x 1>
err :- error in output using parameters <m x 1>

Note:

This program is based on the algorithm presented in the book Applied System Identification of Dr. Jer-Nan Juang. Based on the following difference equation:

$$y(k) + \{+ i=1..p\} H2(i) y(k-i) = \{+ i=1..p\} H1(i) u(k-i) + D u(k)$$

the parameter matrix Yhat is arranged as follows:

$$Y_{hat} = [D \ -H2(1) \ H1(1) \ -H2(2) \ H1(2) \ \dots \ -H2(p) \ H1(p)]$$

The memory allocation routines used are from the book:
 Numerical Recipes in C, The Art of Scientific Computing, 2nd ed.
 by W.H.Press, S.A.Teukolsky, W.T.Vetterling and B.P.Flannery.

In order to understand the programs, the comments of the programs
 have the equation number that appears in the thesis.

program by: date:
 Jose L. Maldonado-Salgado 8/5/1993

George Washington University
 Spacecraft Dynamics Branch NASA Langley
 */

```
{
double *gainvec,**cop,scalar;
double *tmp, *tmp2, **tmp3, tmp4;
int rm, i, j, z, r1, m1, p1, dimVp1;
rm=r+m; r1=r-1; m1=m-1; p1=p-1; dimVp1=dimVp-1; tmp4=0.0;

/* Memory allocation for temporary vectors and matrices */

gainvec=dvector(0,dimVp1);
tmp=dvector(0,dimVp1);
tmp2=dvector(0,m1);
tmp3=dmatrix(0,dimVp1,0,dimVp1);
cop=dmatrix(0,dimVp1,0,dimVp1);

/* Update of forgetting factor */

lami[0]=lamf*lami[0]+(1-lamf);
scalar=lami[0];

/* Form the Input/Output data vector */
/* Eq. # 1 */

for(i=0;i<=r1;i++){
    vp[i]=u[i];
}
for(j=0;j<=p1;j++){
    for(i=0;i<=r1;i++){
        vp[(rm*j)+rm+i]=uold[i][p1-j];
    }
}
for(j=0;j<=p1;j++){
    for(i=0;i<=m1;i++){
        vp[(rm*j)+r+i]=yold[i][p1-j];
    }
}
}
```

```

        /*      Eq. # 2      */
    for(j=0;j<=dimVp1;j++){
        for(i=0;i<=dimVp1;i++){
            tmp[j]=vp[i]*Pmat[i][j]+tmp[j];
        }
        scalar=scalar+vp[j]*tmp[j];
    }

/* Computing the gain vector */
    /*      Eq. # 3      */

    for(i=0;i<=dimVp1;i++){
        gainvec[i]=tmp[i]/scalar;
    }

/* Estimated output and residual */
    /* Eq. # 4 and #5      */

    for(i=0;i<=m1;i++){
        for(j=0;j<=dimVp1;j++){
            tmp2[i]=tmp2[i]+Yhat[i][j]*vp[j];
        }
        yh[i]=tmp2[i];
        err[i]=y[i]-tmp2[i];
    }
    for(j=0;j<=dimVp1;j++){
        for(i=0;i<=dimVp1;i++){
            tmp3[j][i]=vp[j]*gainvec[i]*(-1.);
        }
    }
    for(i=0;i<=dimVp1;i++){
        tmp3[i][i]=1.+tmp3[i][i];
    }
    tmp4=0;

/* Covariance Matrix */
/*      Eq. # 6      */

    for(i=0;i<=dimVp1;i++){
        for(j=0;j<=dimVp1;j++){
            for(z=0;z<=dimVp1;z++){
                tmp4=tmp4+Pmat[i][z]*tmp3[z][j];
            }
            cop[i][j]=tmp4;
            tmp4=0;
        }
    }
    for(i=0;i<=dimVp1;i++){
        for(j=0;j<=dimVp1;j++){
            Pmat[i][j]=cop[i][j];
        }
    }

```

```

/* Update of the Observer Markov Parameters */
/*   Eq # 7   */

for(i=0;i<=m1;i++){
  for(j=0;j<=dimVp1;j++){
    Yhat[i][j]=Yhat[i][j]+err[i]*gainvec[j];
  }
}
free_dvector(gainvec,0,dimVp1);
free_dvector(tmp,0,dimVp1);
free_dvector(tmp2,0,m1);
free_dmatrix(tmp3,0,dimVp1,0,dimVp1);
free_dmatrix(cop,0,dimVp1,0,dimVp1);

}

```

8.1.2. Fast Transversal Filter Functions

```

/* ftfini.c */

/* Batch Least Squares for Initializing the FTF */

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

#define nx 400

void ftfini(float **uld,float **yld,int dimVp,double **Yhatf,
            double **Yhatb,double **Ef,double **Eb,
            double *gamk1,double *gaink1,int r,int m,int p,int n,int flg[])

/* This subroutine initialize the matrices of the parameters,
the error squares, conversion factor and gain vector that
will be used for the fast transversal filter algorithm.

Variables:

Input:

    r :- number of inputs
    m :- number of outputs
    p :- system order
    n :- number of data points
    dimVp :- (r+m)*p + r
    uld :- input matrix <r x n>
    yld :- output matrix <m x n>

Output:
    Yhatf :- Forward time estimation <(r+m) x dimVp>

```



```

cop=dmatrix(1,dimVp,1,dimVp);
tmp5=dmatrix(0,dimVp1,0,dimVp1);
tmp2=dmatrix(0,nx,0,dimVp1);
tmp3=dmatrix(0,rm1,0,rm1);
tmp4=dmatrix(0,rm1,0,nx);
/*printf("n=%d n1=%d n2=%d\n",n,n1,n2);*/
for(i=0;i<=n;i++){
  in=i+p;
  for(j=0;j<=r1;j++){
    uw[j][in]=uld[j][i];
  }
  for(j=0;j<=m1;j++){
    yw[j][in]=yld[j][i];
  }
}
/*      Eq # 5      */
for(j=0;j<=r1;j++){
  yf1[j]=uld[j][0];
}
in=0;

/* Input/Output Data Matrix Construction */
/*      Equations      #1,      #2      #3      */

for(k=p;k<=n+p;k++){
  k1=k-1;
  for(i=0;i<=r1;i++){
    vp[i]=uw[i][k];
  }
  for(j=0;j<=p1;j++){
    for(i=0;i<=r1;i++){
      vp[(rm*j)+rm+i]=uw[i][k1-j];
    }
    for(i=0;i<=m1;i++){
      vp[(rm*j)+r+i]=yw[i][k1-j];
    }
  }
  if(k<=n1+p)
  {
    for(i=0;i<=dimVp1;i++){
      Vp2[i][in]=vp[i];
      Vp1[i][in]=vp[i];
    }
  }
  else
  {
    for(i=0;i<=dimVp1;i++){
      /*      printf("%f\n",vp[i]);*/
      Vp1[i][n]=vp[i];
    }
  }
  in=in+1;
}

```

```

for(i=0;i<=n1;i++){
    in=1+i;
/*      Eq. # 4      */

    for(j=0;j<=r1;j++){
        yfork1[j][i]=uld[j][in];
    }
    for(j=0;j<=m1;j++){
        yfork1[r+j][i]=yld[j][i];
    }
}
/*      Eq. # 6      */
for(i=0;i<=n-p-1;i++){
    in=p+1+i;
    for(j=0;j<=m1;j++){
        yuk1p[j][in]=yld[j][i];
    }
    for(j=0;j<=r1;j++){
        yuk1p[m+j][in]=uld[j][i];
    }
}
tmp1=0;

/* Covariance Matrix */
/*      Eq. # 7      */

for(i=0;i<=dimVp1;i++){
    row=i+1;
    for(j=0;j<=dimVp1;j++){
        col=j+1;
        for(z=0;z<=n1;z++){
            tmp1=tmp1+Vp2[i][z]*Vp2[j][z];
        }
        Ppk2[i][j]=tmp1;
        cop[row][col]=Ppk2[i][j];
        tmp1=0;
    }
}
svdcmp(cop,dimVp,dimVp,flg);
printf("flg=%d\n",flg[0]);
for(i=1;i<=dimVp;i++){
    row=i-1;
    for(j=1;j<=dimVp;j++){
        col=j-1;
        Ppk2[row][col]=cop[i][j];
    }
}
for(j=0;j<=dimVp1;j++){
    for(i=0;i<=dimVp1;i++){
        tmp[j]=vp[i]*Ppk2[i][j]+tmp[j];
    }
    scalar=scalar+vp[j]*tmp[j];
}

```

```

/* Gain vector computation */
/*   Eq # 8   */

for(i=0;i<=dimVp1;i++){
  gaink1[i]=tmp[i]/scalar;
}
tmp1=0;
for(i=0;i<=n1;i++){
  for(j=0;j<=dimVp1;j++){
    for(z=0;z<=dimVp1;z++){
      tmp1=tmp1+Vp2[z][i]*Ppk2[z][j];
    }
    tmp2[i][j]=tmp1;
    tmp1=0;
  }
}
tmp1=0;

/* Computation of the Forward-time Obaserver Markov Parameters */
/*   Eq # 9   */

for(i=0;i<=rm1;i++){
  for(j=0;j<=dimVp1;j++){
    for(z=0;z<=n1;z++){
      tmp1=tmp1+yfork1[i][z]*tmp2[z][j];
    }
    Yhatf[i][j]=tmp1;
    tmp1=0;
  }
}
tmp1=0;
for(i=0;i<=rm1;i++){
  for(j=0;j<=n1;j++){
    for(z=0;z<=dimVp1;z++){
      tmp1=tmp1+Yhatf[i][z]*Vp2[z][j];
    }
    tmp4[i][j]=yfork1[i][j]-tmp1;
    tmp1=0;
  }
}
tmp1=0;

/* Forward-time Error Squares */
/*   Eq. # 10   */

for(i=0;i<=rm1;i++){
  for(j=0;j<=rm1;j++){
    for(z=0;z<=n1;z++){
      tmp1=tmp1+tmp4[j][z]*tmp4[i][z];
    }
    Ef[i][j]=tmp1+yf1[j]*yf1[i];
    tmp1=0;
  }
}

```

```

gamk1[0]=1;
/* Eq. # 11 */

for(i=0;i<=dimVp1;i++){
    gamk1[0]=gamk1[0]-gank1[i]*vp[i];
}

/* Eq # 12 */

for(j=0;j<=dimVp1;j++){
    for(i=0;i<=dimVp1;i++){
        tmp5[j][i]=vp[j]*gank1[i]*(-1.);
    }
}

for(i=0;i<=dimVp1;i++){
    tmp5[i][i]=1.+tmp5[i][i];
}

tmp1=0;
for(i=0;i<=dimVp1;i++){
    row=1+i;
    for(j=0;j<=dimVp1;j++){
        for(z=0;z<=dimVp1;z++){
            col=z+1;
            tmp1=tmp1+cop[row][col]*tmp5[z][j];
        }
        Ppk2[i][j]=tmp1;
        tmp1=0;
    }
}

tmp1=0;
for(i=0;i<=n;i++){
    for(j=0;j<=dimVp1;j++){
        for(z=0;z<=dimVp1;z++){
            tmp1=tmp1+Vp1[z][i]*Ppk2[z][j];
        }
        tmp2[i][j]=tmp1;
        tmp1=0;
    }
}

tmp1=0;

/* Computation of the Backward-time Observer Markov Parameters */
/* Eq. # 13 */

for(i=0;i<=rml;i++){
    for(j=0;j<=dimVp1;j++){
        for(z=0;z<=n;z++){
            tmp1=tmp1+yuk1p[i][z]*tmp2[z][j];
        }
        Yhatb[i][j]=tmp1;
        tmp1=0;
    }
}

```

```

tmp1=0;
for(i=0;i<=rm1;i++){
  for(j=0;j<=n;j++){
    for(z=0;z<=dimVp1;z++){
      tmp1=tmp1+Yhatb[i][z]*Vp1[z][j];
    }
    tmp4[i][j]=yuk1p[i][j]-tmp1;
    tmp1=0;
  }
}
tmp1=0;

/* Backward-time Error Squares */
/*   Eq. # 14   */

for(i=0;i<=rm1;i++){
  for(j=0;j<=rm1;j++){
    for(z=0;z<=n;z++){
      tmp1=tmp1+tmp4[j][z]*tmp4[i][z];
    }
    Eb[i][j]=tmp1;
    tmp1=0;
  }
}
free_dvector(vp,0,dimVp1);free_dvector(tmp,0,dimVp1);free_dvector(yf1,0,rm1);
free_matrix(uw,0,r,0,n+1+p); free_matrix(yw,0,m,0,n+1+p);
free_dmatrix(Vp1,0,dimVp1,0,nx); free_dmatrix(Ppk2,0,dimVp1,0,dimVp1);
free_dmatrix(Vp2,0,dimVp1,0,nx); free_dmatrix(tmp5,0,dimVp1,0,dimVp1);
free_dmatrix(cop,1,dimVp,1,dimVp); free_dmatrix(tmp2,0,nx,0,dimVp1);
free_dmatrix(yfork1,0,rm1,0,nx); free_dmatrix(yuk1p,0,rm1,0,nx);
free_dmatrix(tmp3,0,rm1,0,rm1); free_dmatrix(tmp4,0,rm1,0,nx);
}

```

/* ftf.c */

/* Forward-Time Estimation */

```

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

```

```

void ftf(float *ufut,float *u,float *y,float **uold,float **yold,int m,
int r, int p, double **Yhatf, double **Yhatb,double *gainupa,
double **Ef,double *err,double **Eb,double *gamk1,double *gainupr,
double *gaink1,int dimVp,int flg[],double gamup[],double *yh)

```

/* This subroutine updates the markov parameters using the fast transversal algorithm.

Variables:


```

{
int row,col,i,j,k,z,m1,r1,dimVp1,p1,p2,rm,rm1;
double *vp, *yf, *efik, *efdk, *ga, *gr;
double *gainup, *yu;
double tmp1, tmp2;
double **iEf;
flg[0]=0;
rm=r+rm;rm1=rm-1;r1=r-1;m1=m-1;dimVp1=dimVp-1;p1=p-1;p2=p-2;

/* Memory allocation for temporary vectors and matrices */
vp=dvector(0,dimVp1);
yf=dvector(0,rm1);
yu=dvector(0,rm1);
efik=dvector(0,rm1);
efdk=dvector(0,rm1);
ga=dvector(0,rm1);
gr=dvector(0,dimVp1);
gainup=dvector(0,dimVp1+rm);
iEf=dmatrix(1,rm,1,rm);

/* Forward Time Update */

/* Eq. # 16 */
for(i=0;i<=r1;i++){
    vp[i]=u[i];
    yf[i]=ufut[i];
}
for(i=0;i<=m1;i++){
    yf[r+i]=y[i];
}

/* Form the Input/Output Data Matrix */
/* Eq. # 15 */
for(j=0;j<=p1;j++){
    for(i=0;i<=r1;i++){
        vp[(rm*j)+rm+i]=uold[i][p1-j];
    }
}
for(j=0;j<=p1;j++){
    for(i=0;i<=m1;i++){
        vp[(rm*j)+r+i]=yold[i][p1-j];
    }
}
tmp1=0;

/* Compute the estimated output, the a priori forward-time estimation error
and the a posteriori forward-time estimation error */
/* Eq. # 17 and Eq. # 18*/

for(i=0;i<=rm1;i++){
    for(z=0;z<=dimVp1;z++){
        yh[i]=tmp1=tmp1+Yhatf[i][z]*vp[z];
    }
    err[i]=efik[i]=yf[i]-tmp1;
}

```

```

    tmp1=0;
    efdk[i]=gamk1[0]*efik[i];
    ga[i]=0;
}

/* Forward-time error squares matrix */
/*   Eq. # 20   */

for(i=0;i<=rm1;i++){
    row=i+1;
    for(j=0;j<=rm1;j++){
        col=j+1;
        Ef[i][j]=Ef[i][j]+efdk[i]*efik[j];
        iEf[row][col]=Ef[i][j];
    }
}

svdcmp(iEf,rm,rm,flg);
if(flg[0]!=1){

/* Update the Forward-Time Observer Markov Parameters */
/*   Eq. # 19   */

for(i=0;i<=rm1;i++){
    for(j=0;j<=dimVp1;j++){
        Yhatf[i][j]=Yhatf[i][j]+efik[i]*gaink1[j];
    }
}
gamup[0]=gamk1[0];

/* Update and Augmentation of the gain matrix */
/*   Eq. # 21   and   Eq. # 22*/

for(i=0;i<=rm1;i++){
    row=i+1;
    for(j=0;j<=rm1;j++){
        col=j+1;
        ga[i]=efdk[j]*iEf[col][row]+ga[i];
    }
    gainup[i]=ga[i];
    gamup[0]=gamup[0]-ga[i]*efdk[i];
}
tmp1=0;

/* Partition of the gain matrix */
/*   Eq. # 23   */

for(i=0;i<=dimVp1;i++){
    for(j=0;j<=rm1;j++){
        tmp1=tmp1+ga[j]*Yhatf[j][i];
    }
    gr[i]=gaink1[i]-tmp1;
    gainup[rm+i]=gr[i];
    gainupr[i]=gainup[i];
}

```

```

    tmp1=0;
  }
  for(i=0;i<=rm1;i++){
    gainupa[i]=gainup[dimVp+i];
  }

/* call the Backward-Time update recursion */

ftfb(ufut,u,y,uold,yold,m1,r1,p1,p2,dimVp1,Yhatb,gainupa,gainupr,gaink1,gamk1,gamup,
p,Eb);
}
else
{

/* call the Backward-Time update recursion */

ftfb(ufut,u,y,uold,yold,m1,r1,p1,p2,dimVp1,Yhatb,gainupa,gainupr,gaink1,gamk1,gamup,
p,Eb);
}
free_dvector(vp,0,dimVp1); free_dvector(yf,0,rm1); free_dvector(yu,0,rm1);
free_dvector(efik,0,rm1);free_dvector(efdk,0,rm1);free_dvector(ga,0,rm1);
free_dvector(gr,0,dimVp1);free_dvector(gainup,0,dimVp1+rm);
free_dmatrix(iEf,1,rm,1,rm);
}

/* ffb.c */

/* Backward-Time Estimation */

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

void ffb(float *ufut,float *u,float *y,float **uold,float **yold,int m1,
int r1, int p1,int p2, int dimVp1, double **Yhatb,
double *gainupa,double *gainupr,double *gaink1,
double *gamk1,double gamup[],double **Eb)

/* This subroutine updates the backward markov parameters using
the fast transversal algorithm.

Variables:

Input:
r :- number of inputs
m :- number of outputs
p :- system order
dimVp :- (r+m)*p + r
u :- input time k <r x 1>
y :- output time k <m x 1>

```

```

    ufut  :- input time k+1 <r x 1>
    uold  :- input time k-1 : k-p <r x p>
    yold  :- output time k-1 : k-p <m x p>
Input/Output:
    Yhatb :- Backward time estimation <(r+m) x dimVp>
    Eb    :- Backward time estimation-error squares <(r+m) x (r+m)>
    gamk1 :- conversion factor
    gamup :- update of conversion factor
    gaink1 :- gain vector < 1 x dimVp>
    gainupr :- first <1 x dimVp> elements of the augmented gain vector
    gainupa :- last <1 x (r+m)> elements of the augmented gain vector

```

Note:

This program is based on the algorithm presented in the book
Applied System Identification of Dr. Jer-Nan Juang.
Based on the following difference equation:

$$y(k) + \{+ i=1..p\} H2(i) y(k-i) = \{+ i=1..p\} H1(i) u(k-i) + D u(k)$$

the parameter matrix Yhatf is arranged as follows:

$$Yhatf = [D \ -H2(1) \ H1(1) \ -H2(2) \ H1(2) \ \dots \ -H2(p) \ H1(p)]$$

The function "svdcmp" used to invert matrices is a modification
of the singular value decomposition presented in the book:
Numerical Recipes in C, The Art of Scientific Computing 2nd ed
by W.H.Press, S.A.Teukolsky, W.T.Vetterling and B.P.Flannery.
The routines for memory allocation are obtained from the same book.

To understand better the programs the equation number appearing in the
algorithms of the thesis are included also in the programs.

```

program by:          date:
Jose L. Maldonado-Salgado      8/23/1993

```

```

George Washington University
Spacecraft Dynamics Branch NASA Langley
*/

```

```

{
int i,j,rm1,m,rm,r,z;
double *vp, *yu,tmp1,tmp2,*ebi,*ebd;
rm1=r1+m1+1; m=m1+1; r=r1+1; rm=r+m;

/* Memory allocation for temporary vectors and matrices */

vp=dvector(0,dimVp1);
yu=dvector(0,rm1);
ebi=dvector(0,rm1);
ebd=dvector(0,rm1);

/* Backward Time Update */
/*           Eq. # 25           */

```

```

for(i=0;i<=r1;i++){
    v[i]=ufut[i];
    uold[i]=uold[i][0];

for(i=0;i<=m1;i++){
    yu[i]=yold[i][0];
}
for(j=0;j<=r1;j++){
    for(i=0;i<=p2;i++){
        uold[j][i]=uold[j][i+1];
    }
    uold[j][p1]=u[j];
}
for(j=0;j<=m1;j++){
    for(i=0;i<=p2;i++){
        yold[j][i]=yold[j][i+1];
    }
    yold[j][p1]=y[j];
}

/* Form the Input/Output Data Matrix */
/* Eq. # 24 */

for(j=0;j<=p1;j++){
    for(i=0;i<=r1;i++){
        vp[(rm*j)+rm+i]=uold[i][p1-j];
    }
}
for(j=0;j<=p1;j++){
    for(i=0;i<=m1;i++){
        vp[(rm*j)+r+i]=yold[i][p1-j];
    }
}
tmp1=0;
tmp2=1;

/* Compute the Backward-Time a priori and a posteriori estimation error */
/* Eq # 26 */

for(i=0;i<=rm1;i++){
    for(z=0;z<=dimVp1;z++){
        tmp1=tmp1+Yhatb[i][z]*vp[z];
    }
    ebi[i]=yu[i]-tmp1;
    tmp2=tmp2-gainupa[i]*ebi[i];
    tmp1=0;
}
tmp1=0;

/* Update the gain matrix */
/* Eq. # 27 */

for(i=0;i<=dimVp1;i++){

```

```

    for(j=0;j<=rm1;j++){
        tmp1=tmp1+gainupa[j]*Yhatb[j][i];
    }
    gaink1[i]=(gainupr[i]+tmp1)/tmp2;
    tmp1=0;
}

/* Update the Backward-Time Observer Markov Parameters */
/*   Eq. # 28   */

for(i=0;i<=rm1;i++){
    for(j=0;j<=dimVp1;j++){
        Yhatb[i][j]=Yhatb[i][j]+ebi[i]*gaink1[j];
    }
}
/*   Eq # 29   */

gamk1[0]=gamup[0]/tmp2;

/* Update the Backward-Time Error Squares */
/*   Eq. # 30   and   Eq. #31   */
for(i=0;i<=rm1;i++){
    ebd[i]=gamk1[0]*ebi[i];
    for(j=0;j<=rm1;j++){
        Eb[i][j]=Eb[i][j]+ebd[i]*ebi[j];
    }
}
free_dvector(vp,0,dimVp1);
free_dvector(yu,0,rm1);
free_dvector(ebi,0,rm1);
free_dvector(ebd,0,rm1);
}

```

8.1.3 Least Squares Lattice Filter

```

/* ls1f.c */

#include<stdlib.h>
#include<memory.h>
#include<stdio.h>
#include "nrutil.h"
#define MX 650

void ls1f(float **u,float *y,int r,int m,int p,double **Yhatf,double **Yhatfc,double
**Yhatb,double **P0,double **delta,double **Efc,double *err,double **Ebc,double
**iEbc,double **ebdc,double gam[],double *G0)

/* This subroutine uses the least squares lattice filter to
update the markov parameters.

```

Variables:


```

int a,b,d, o,jo, pmax, i, j, z, flag[1],dm, l, w, q, c, s,p1,base,h,in;
double **Yhf, **Yhatbc, **Yhb,
**dlt, **ec, *ef, *eb, *efi, *efd, *efdc, **Lf, **Lb,
**iE, **iEf, **iEfc, **Ef, **iEb, **Ebc2, tmp1,
*tmp, **tmp2, tmp3, scalar, *v, *yf, *ebi, *ebd, **tmp4,**P;
o=r+m; p1=p+1; pmax=p-1; dm=p*o+r;

/* memory allocation for vectors and matrices */

tmp=dvector(1,r);
v=dvector(1,r); yf=dvector(1,o);
Yhf=dmatrix(1,o,1,dm); ebi=dvector(1,o);
Yhatbc=dmatrix(1,o,1,MX); Yhb=dmatrix(1,o,1,dm);
ec=dmatrix(1,o,1,p1);
dlt=dmatrix(1,o,1,o*p1); ef=dvector(1,o);
eb=dvector(1,o); efi=dvector(1,o); efd=dvector(1,o); efdc=dvector(1,o);
Lf=dmatrix(1,o,1,o); Lb=dmatrix(1,o,1,o); iE=dmatrix(1,o,1,o);
iEf=dmatrix(1,o,1,o); iEfc=dmatrix(1,o,1,o);
Ef=dmatrix(1,o,1,o);
iEb=dmatrix(1,o,1,o*p1); Ebc2=dmatrix(1,o,1,o*p1); tmp2=dmatrix(1,o,1,o);
ebd=dvector(1,o); tmp4=dmatrix(1,o,1,o); P=dmatrix(1,r,1,r);

/* Initialization at k >= 1 & p = 0 */
/* Forward Time Estimation */
/* Eq. # 10 and Eq.# 11 */
for(i=1;i<=r;i++){
    v[i]=u[i][1];
    yf[i]=u[i][2];
}
for(i=1;i<=m;i++) yf[r+i]=y[i];
tmp1=0;
/* Eq. # 12 and Eq # 13 */
for(i=1;i<=o;i++){
    for(j=1;j<=r;j++) tmp1=tmp1+Yhatf[i][j]*v[j];
    efi[i]=yf[i]-tmp1;
    efdc[i]=efd[i]=gam[0]*efi[i];
    tmp1=0;
/* Eq. # 15 */
    for(j=1;j<=r;j++){ Yhatf[i][j]=Yhatf[i][j]+efi[i]*G0[j];
        Yhatfc[i][j]=Yhatf[i][j];}
}
/* Eq. # 14 */
for(i=1;i<=o;i++){
    for(j=1;j<=o;j++){
        Ef[i][j]=iEfc[i][j]=Efc[i][j]=Efc[i][j]+efd[i]*efi[j];}
}
svdcmp(iEfc,o,o,flag);

/* Backward Time Estimation */
/* Eq. #16 and Eq. # 17 */
for(i=1;i<=r;i++){
    v[i]=u[i][2];
    yf[m+i]=u[i][1];

```

```

    tmp[i]=0.;
  }
  for(i=1;i<=m;i++) yf[i]=y[i];
  tmp1=0;
  /*      Eq. # 18      */
  for(i=1;i<=o;i++){
    for(j=1;j<=r;j++){
      tmp1=tmp1+Yhatb[i][j]*v[j];
    }
    ebi[i]=yf[i]-tmp1;
    tmp1=0;
  }
  scalar=1.;
  for(j=1;j<=r;j++){
    for(z=1;z<=r;z++) tmp[j]=tmp[j]+P0[z][j]*v[z];
    scalar=scalar+v[j]*tmp[j];
  }
  /*      Eq. # 19      and      Eq # 22      */

  gam[1]=gam[0]; gam[0]=1.;
  for(i=1;i<=r;i++){
    G0[i]=tmp[i]/scalar;
    gam[0]=gam[0]-G0[i]*v[i];
  }
  for(i=1;i<=r;i++){
    for(j=1;j<=r;j++) tmp2[i][j]=v[i]*G0[j]*(-1);
    tmp2[i][i]=1.+tmp2[i][i];
  }
  tmp1=0;
  /*      Eq # 20      */

  for(i=1;i<=r;i++){
    for(j=1;j<=r;j++){
      for(z=1;z<=r;z++) tmp1=tmp1+P0[i][z]*tmp2[z][j];
      P[i][j]=tmp1;
      tmp1=0;
    }
  }
  for(i=1;i<=r;i++){
    for(j=1;j<=r;j++) P0[i][j]=P[i][j];
  }
  /*      Eq. # 21      and      Eq. # 23*/

  for(i=1;i<=o;i++){
    for(j=1;j<=r;j++) Yhatbc[i][j]=Yhatb[i][j]+ebi[i]*G0[j];
    ebd[i]=gam[0]*ebi[i];
    ec[i][1]=ebd[i];
  }
  /*      Eq # 24      */

  for(i=1;i<=o;i++){
    for(j=1;j<=o;j++){
      Ebc2[i][j]=Ebc[i][j]+ebd[i]*ebi[j];
      iE[i][j]=Ebc2[i][j];
      tmp4[i][j]=0;
    }
  }

```

```

    }
  }
  svdcmp(iE,o,o,flag);
  for(i=1;i<=o;i++){
    for(j=1;j<=o;j++) iEb[i][j]=iE[i][j];
  }

/* Time Update Coefficient Recursion */

j=1; in=1; i=1; l=0;
while(l<=pmax){
  jo=j*o;
/*   Eq. # 25   */

  for(a=1;a<=o;a++){
    for(b=1;b<=o;b++){
      tmp4[a][b]=tmp4[a][b]+efdc[a]*ebdc[b][l+1]/gam[1];
    }
  }
  for(q=1;q<=o;q++){
    a=1;
    for(z=i;z<=jo;z++){
      dlt[q][z]=delta[q][z]+tmp4[q][a];
      tmp4[q][a]=0;
      a++;
    }
  }

/* Forward Time Order Update Recursion p>=0 */

  tmp1=0;
  w=o*l+r;
/*   Eq. # 26   */

  for(q=1;q<=o;q++){
    s=1; ef[q]=efdc[q];
    for(z=i;z<=jo;z++){
      d=1;
      for(c=i;c<=jo;c++){
        tmp1=tmp1+dlt[q][c]*iEbc[d][z];
        d++;
      }
      Lf[q][s]=(-tmp1);
      tmp1=0;
      s++;
    }
    eb[q]=ebdc[q][l+1];
    for(c=1;c<=w;c++) Yhf[q][c]=Yhatfc[q][c];
    d=1;
    for(c=in;c<=in+w-1;c++){
      Yhb[q][d]=Yhatb[q][c];
      d++;
    }
  }
}

```

```

/*   Eq. # 27   and   Eq. # 28   */

for(q=1;q<=o;q++){
  for(c=1;c<=w;c++){
    for(s=1;s<=o;s++) Yhatfc[q][c]=Yhatfc[q][c]+Lf[q][s]*Yhb[s][c];
  }
  for(s=1;s<=o;s++){
    Yhatfc[q][w+s]=(-Lf[q][s]);
    err[q]=efdc[q]=efdc[q]+Lf[q][s]*eb[s];
  }
}
tmp1=0;
for(q=1;q<=o;q++){
  for(z=1;z<=o;z++){
    for(c=i;c<=jo;c++)tmp1=tmp1+iEbc[q][c]*dlt[z][c];
    tmp2[q][z]=tmp1;
    tmp1=0;
  }
}
/*   Eq # 29   */

tmp1=0;
for(q=1;q<=o;q++){
  for(s=1;s<=o;s++){
    z=1;
    for(c=i;c<=jo;c++){
      tmp1=tmp1+dlt[q][c]*tmp2[z][s];
      z++;
    }
    iEf[q][s]=Ef[q][s]=Ef[q][s]-tmp1;
    tmp1=0;
  }
}
svdcmp(iEf,o,o,flag);

/* Backward Time Order Update Recursion p >= 0 */

tmp1=0;
z=1;
/*   Eq. # 30   */

for(s=i;s<=jo;s++){
  for(c=1;c<=o;c++){
    for(q=1;q<=o;q++){
      tmp1=tmp1+dlt[q][s]*iEfc[q][c];
    }
    Lb[z][c]=(-tmp1);
    tmp1=0;
  }
  z++;
}
base=w+in;
for(c=1;c<=o;c++){
  z=1;

```

```

    for(q=base;q<=base+o-1;q++){ Yhatbc[c][q]=-Lb[c][z];
                                z++;}
}
/*      Eq. # 31      */
tmp1=0;
for(c=1;c<=o;c++){
  for(q=1;q<=w;q++){
    for(s=1;s<=o;s++) tmp1=tmp1+Lb[c][s]*Yhf[s][q];
    Yhatbc[c][q+base+o-1]=tmp1+Yhb[c][q];
    tmp1=0;
  }
}
z=1; tmp1=0; tmp3=0;
for(q=i;q<=jo;q++){
  for(s=1;s<=o;s++){
    tmp1=tmp1+ebdc[s][l+1]*iEbc[s][q];
    tmp3=tmp3+Lb[z][s]*ef[s];
  }
}
/*      Eq. # 32      and      Eq. # 34      */

gam[1]=gam[1]-tmp1*ebdc[z][l+1];
tmp1=0;
ec[z][l+2]=ebdc[z][l+1]+tmp3;
tmp3=0;
z++;
}
tmp1=0;
for(q=1;q<=o;q++){
  z=1;
  for(s=i;s<=jo;s++){
    for(c=1;c<=o;c++) tmp1=tmp1+iEfc[q][c]*dlt[c][s];
    tmp2[q][z]=tmp1;
    tmp1=0;
    z++;
  }
}
tmp1=0;
h=i;
for(q=1;q<=o;q++){
  d=1; a=i;
/*      Eq.# 33      */

  for(s=i+o;s<=jo+o;s++){
    for(c=1;c<=o;c++){
      tmp1=tmp1+dlt[c][h]*tmp2[c][d];
    }
    Ebc2[q][s]=Ebc[q][a]-tmp1;
    iE[q][d]=Ebc2[q][s];
    tmp1=0;
    d++; a++;
  }
  for(c=1;c<=o;c++) iEfc[q][c]=iEf[q][c];
  h++;
}

```

```

svdcmp(iE,o,o,flag);
for(q=1;q<=o;q++){
  z=1;
  for(s=i+o;s<=i+o+o-1;s++){
    iEb[q][s]=iE[q][z];
    z++;
  }
}
i=i+o; j=j+1; in=in+w; l=l+1;

}
for(a=1;a<=o;a++){
  for(b=1;b<=o*p1;b++){
    delta[a][b]=dlt[a][b]; iEbc[a][b]=iEb[a][b]; Ebc[a][b]=Ebc2[a][b];
    for(b=1;b<=p1;b++) ebdc[a][b]=ec[a][b];
    for(b=1;b<=MX;b++) Yhatb[a][b]=Yhatbc[a][b];
  }
  free_dvector(tmp,1,r); free_dvector(v,1,r); free_dvector(yf,1,o);
  free_dmatrix(Yhf,1,o,1,dm); free_dvector(ebi,1,o); free_dmatrix(Yhb,1,o,1,dm);
  free_dmatrix(Yhatbc,1,o,1,MX); free_dmatrix(ec,1,o,1,p1);
  free_dmatrix(dlt,1,o,1,o*p1); free_dvector(ef,1,o); free_dvector(efd,1,o);
  free_dvector(eb,1,o); free_dvector(efi,1,o); free_dmatrix(efdc,1,o);
  free_dmatrix(Lf,1,o,1,o); free_dmatrix(Lb,1,o,1,o); free_dmatrix(iE,1,o,1,o);
  free_dmatrix(iEf,1,o,1,o); free_dmatrix(iEfc,1,o,1,o); free_dmatrix(Ef,1,o,1,o);
  free_dmatrix(iEb,1,o,1,o*p1); free_dmatrix(Ebc2,1,o,1,o*p1);
  free_dmatrix(tmp2,1,o,1,o); free_dvector(ebd,1,o);
  free_dmatrix(tmp4,1,o,1,o); free_dmatrix(P,1,r,1,r);
}

```

8.2 Other Functions and header files needed for the C programs

```

/* nutil.h */
/*

```

This header file defines some functions and also declares the data types of the memory allocation routines used in the programs. For future used of any of the memory allocation routines or any of the functions in this file this file has to be included in the program or function that uses the functions defined here.

example of how to include this header file:

```
#include "nutil.h"
```

This header file was obtained from the book:

Numerical Recipes in C
by, W.H. Press, S.A. Teukolsky, W.T. Vetterling and P. Flannery

```

*/
#ifdef _NR_UTILS_H_
#define _NR_UTILS_H_

```

```

static double sqrarg;
#define SQR(a) ((sqrarg=(a))==0.0 ? 0.0 : sqrarg*sqrarg)

static double dsqarg;
#define DSQR(a) ((dsqarg=(a))==0.0 ? 0.0 : dsqarg*dsqarg)

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

static float 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)

void nerror(char error_text[]);
float *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);
float **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);
float **submatrix(float **a,long oldrl,long oldrh,long oldcl,long oldch,long
newrl,long newcl);
float **convert_matrix(float *a,long nrl,long nrh,long ncl,long nch);

```

```

float ***f3tensor(long nrl,long nrh,long ncl,long nch,long ndl,long ndh);
void free_vector(float *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(float **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(float **b,long nrl,long nrh,long ncl,long nch);
void free_convert_matrix(float **b,long nrl,long nrh,long ncl,long nch);
void free_f3tensor(float ***t,long nrl,long nrh,long ncl,long nch,long
ndl,long ndh);

```

```

#else
void nrerror();
float *vector();
#endif
#endif

```

```

/* nrutil.c */

```

```

/*
This file contains all the memory allocation routines
used in the programs. This functions is to dynamically
allocate the memory as is needed in the program.

```

```

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

```

```

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

void nrerror(char error_text[])
{
    fprintf(stderr,"run time error\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"now exiting to system\n");
    exit(1);
}

float *vector(long nl,long nh)
{
    float *v;
    v=(float *)calloc((size_t)((nh-nl+1+NR_END)),sizeof(float));
    if (!v) nrerror("allocation failure\n");
    return v-nl+NR_END;
}

```

```

void free_vector(float *v, long nl, long nh)
{
    free((FREE_ARG) (v+nl-NR_END));
}
float **matrix(long nrl, long nrh, long ncl, long nch)
{
    long i, nrow=nrh-nrl+1, ncol=nch-ncl+1;
    float **m;
    m=(float **)calloc((size_t)((nrow+NR_END)), sizeof(float*));
    if(!m) perror("allocation failure in matrix");
    m += NR_END;
    m -= nrl;
    m[nrl]=(float *)calloc((size_t)(nrow*ncol+NR_END), sizeof(float));
    if(!m[nrl]) perror("allocation failure");
    m[nrl] += NR_END;
    m[nrl] -= ncl;
    for(i=nrl+1; i<=nrh; i++) m[i]=m[i-1]+ncol;
    return m;
}
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
{
    free((FREE_ARG)(m[nrl]+ncl-NR_END));
    free((FREE_ARG)(m+nrl-NR_END));
}

double *dvector(long nl, long nh)
{
    double *v;
    v=(double *)calloc((size_t)(nh-nl+1+NR_END), sizeof(double));
    if (!v) perror("allocation failure\n");
    return v-nl+NR_END;
}

void free_dvector(double *v, long nl, long nh)
{
    free((FREE_ARG) (v+nl-NR_END));
}
double **dmatrix(long nrl, long nrh, long ncl, long nch)
{
    long i, nrow=nrh-nrl+1, ncol=nch-ncl+1;
    double **m;
    m=(double **)calloc((size_t)(nrow+NR_END), sizeof(double*));
    if(!m) perror("allocation failure in matrix");
    m += NR_END;
    m -= nrl;
    m[nrl]=(double *)calloc((size_t)(nrow*ncol+NR_END), sizeof(double));
    if(!m[nrl]) perror("allocation failure");
    m[nrl] += NR_END;
    m[nrl] -= ncl;
    for(i=nrl+1; i<=nrh; i++) m[i]=m[i-1]+ncol;
    return m;
}
void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch)

```

```

    {
        free((FREE_ARG)(m[nrl]+ncl-NR_END));
        free((FREE_ARG)(m+nrl-NR_END));
    }
int *ivector(long nl,long nh)
{
    int *v;
    v=(int *)calloc((size_t)(nh-nl+1+NR_END),sizeof(int));
    if (!v) nrerror("allocation failure\n");
    return v-nl+NR_END;
}

void free_ivector(int *v, long nl,long nh)
{
    free((FREE_ARG) (v+n1-NR_END));
}
int **imatrix(long nrl,long nrh,long ncl,long nch)
{
    long i,nrow=nrh-nrl+1,ncol=nch-ncl+1;
    int **m;
    m=(int **)calloc((size_t)(nrow+NR_END),sizeof(int*));
    if (!m) nrerror("allocation failure in matrix");
    m += NR_END;
    m -= nrl;
    m[nrl]=(int *)calloc((size_t)(nrow*ncol+NR_END),sizeof(int));
    if (!m[nrl]) nrerror("allocation failure");
    m[nrl] += NR_END;
    m[nrl] -= ncl;
    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;
    return m;
}
void free_imatrix(int **m,long nrl,long nrh,long ncl,long nch)
{
    free((FREE_ARG)(m[nrl]+ncl-NR_END));
    free((FREE_ARG)(m+nrl-NR_END));
}

```

/* pythag.c */

/* This function is used in the svdcmp routine.
it was obtained from the book:

Numerical Recipes in C

by: W.H. Press, S.A. Teukolky, W.T. Vetterling and P. Flannery.

*/

```

#include<math.h>
#include "nrutil.h"

```

```

double pythag(double a,double b)
{
    double absa, absb;
    absa=fabs(a);
    absb=fabs(b);
    if(absa>absb) return absa*sqrt(1.0+SQR(absb/absa));
}

```

```

    else return (absb== 0.0 ? 0.0 : absb*sqrt(1.0+SQR(absa/absb)));
}

```

```

/* svdcmp.c */

```

```

/*

```

This function returns the inverse or pseudoinverse of a matrix by using singular value decomposition.

Variables:

```

    a  :- input matrix and the inverse is return in it
    m  :- number of rows in matrix a
    n  :- number of columns in matrix a
    flg :- flag which says if the matrix is ill conditioned,
          the value return is 1 if it is ill conditioned

```

This function was obtained from the book :

Numerical Recipes in C

by: W.H. Press, S.A. Teukolsky, W.T. Vetterling and P. Flannery

The function was modified so the return value of matrix "a" is the inverse of it. The original version of the function only return the singular value decomposition of the matrix.

```

*/

```

```

#include<math.h>

```

```

#include "nrutil.h"

```

```

void svdcmp(double **a,int m,int n,int flg[])

```

```

{
    double pythag(double a,double b);
    int flag,i,its,j,jj,k,l,nm;
    double sum,**cop,anorm,c,f,g,h,s,**v,*w,tol,scale,x,y,z,*rvl,*norm;
    cop=dmatrix(1,n,1,n);
    v=dmatrix(1,n,1,n);
    rvl=dvector(1,n);
    norm=dvector(1,n);
    w=dvector(1,n);
    g=scale=anorm=0.0;
    flg[0]=0;
    for(i=1;i<=n;i++){
        l=i+1;
        rvl[i]=scale*g;
        g=s=scale=0.0;
        if(i<=m){
            for(k=i;k<=m;k++) scale += fabs(a[k][i]);
            if(scale){
                for(k=i;k<=m;k++){
                    a[k][i] /= scale;
                    s += a[k][i]*a[k][i];
                }
                f=a[i][i];
                g=-SIGN(sqrt(s),f);
                h=f*g-s;

```

```

    a[i][i]=f-g;
    for(j=1;j<=n;j++){
        for(s=0.0,k=i;k<=m;k++) s+=a[k][i]*a[k][j];
        f=s/h;
        for(k=i;k<=m;k++) a[k][j]+=f*a[k][i];
    }
    for(k=i;k<=m;k++) a[k][i]*=scale;
}
}
w[i]=scale*g;
g=s=scale=0.0;
if(i<=m && i!=n){
    for(k=1;k<=n;k++) scale+=fabs(a[i][k]);
    if(scale){
        for(k=1;k<=n;k++){
            a[i][k]/=scale;
            s+=a[i][k]*a[i][k];
        }
        f=a[i][1];
        g=-SIGN(sqrt(s),f);
        h=f*g-s;
        a[i][1]=f-g;
        for(k=1;k<=n;k++) rv1[k]=a[i][k]/h;
        for(j=1;j<=m;j++){
            for(s=0.0,k=1;k<=n;k++) s+=a[j][k]*a[i][k];
            for(k=1;k<=n;k++) a[j][k]+=s*rv1[k];
        }
        for(k=1;k<=n;k++) a[i][k]*=scale;
    }
}
anorm=DMAX(anorm,(fabs(w[i])+fabs(rv1[i])));
}
for(i=n;i>=1;i--){
    if(i<n){
        if(g){
            for(j=1;j<=n;j++)
                v[j][i]=(a[i][j]/a[i][1])/g;
            for(j=1;j<=n;j++){
                for(s=0.0,k=1;k<=n;k++) s+=a[i][k]*v[k][j];
                for(k=1;k<=n;k++) v[k][j] += s*v[k][i];
            }
        }
        for(j=1;j<=n;j++) v[i][j]=v[j][i]=0.0;
    }
    v[i][i]=1.0;
    g=rv1[i];
    l=i;
}
for(i=IMIN(m,n);i>=1;i--){
    l=i+1;
    g=w[i];
    for(j=1;j<=n;j++) a[i][j]=0.0;
    if(g){
        g=1.0/g;
    }
}

```

```

for(j=1;j<=n;j++){
  for(s=0.0,k=1;k<=m;k++) s+=a[k][i]*a[k][j];
  f=(s/a[i][i])*g;
  for(k=i;k<=m;k++) a[k][j]+=f*a[k][i];
}
for(j=i;j<=m;j++) a[j][i]*=g;
}else for(j=i;j<=m;j++) a[j][i]=0.0;
++a[i][i];
}
for(k=n;k>=1;k--){
  for(its=1;its<=30;its++){
    flag=1;
    for(l=k;l>=1;l--){
      nm=l-1;
      if((double)(fabs(rv1[l])+anorm)==anorm){
        flag=0;
        break;
      }
      if((double)(fabs(w[nm])+anorm)==anorm) break;
    }
    if(flag){
      c=0.0;
      s=1.0;
      for(i=1;i<=k;i++){
        f=s*rv1[i];
        rv1[i]=c*rv1[i];
        if((double) (fabs(f)+anorm)==anorm) break;
        g=w[i];
        h=pythag(f,g);
        w[i]=h;
        h=1.0/h;
        c=g*h;
        s=-f*h;
        for(j=1;j<=m;j++){
          y=a[j][nm];
          z=a[j][i];
          a[j][nm]=y*c+z*s;
          a[j][i]=z*c-y*s;
        }
      }
    }
    z=w[k];
    if(l==k){
      if(z<0.0){
        w[k]= -z;
        for(j=1;j<=n;j++) v[j][k]= -v[j][k];
      }
      break;
    }
  }
  if(its==30) nerror("no convergence in 30 svd iterations");
  x=w[l];
  nm=k-1;
  y=w[nm];
  g=rv1[nm];

```

```

h=rv1[k];
f=((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
g=pythag(f,1.0);
f=((x-z)*(x+z)+h*((y/(f+SIGN(g,f))-h))/x;
c=s=1.0;
for(j=1;j<=nm;j++){
    i=j+1;
    g=rv1[i];
    y=w[i];
    h=s*g;
    g=c*g;
    z=pythag(f,h);
    rv1[j]=z;
    c=f/z;
    s=h/z;
    f=x*c+g*s;
    g=g*c-x*s;
    h=y*s;
    y*=c;
    for(jj=1;jj<=n;jj++){
        x=v[jj][j];
        z=v[jj][i];
        v[jj][j]=x*c+z*s;
        v[jj][i]=z*c-x*s;
    }
    z=pythag(f,h);
    w[j]=z;
    if(z){
        z=1.0/z;
        c=f*z;
        s=h*z;
    }
    f=c*g+s*y;
    x=c*y-s*g;
    for(jj=1;jj<=m;jj++){
        y=a[jj][j];
        z=a[jj][i];
        a[jj][j]=y*c+z*s;
        a[jj][i]=z*c-y*s;
    }
}
rv1[1]=0.0;
rv1[k]=f;
w[k]=x;
}
}
sum=0;
/*for(i=1;i<=m;i++){
    sum=sum+w[i];
}
norm[1]=sqrt(abs(sum));
tol=m*norm[1]*2.2204e-16;*/
tol=m*w[1]*2.2204e-16;
/* printf("tol=%f\n",tol);*/

```

```

/* tol=.0001;*/
for(i=1;i<=m;i++){
/*   printf("%f\n",w[i]);*/
  if(w[i]<tol)
  {
    w[i]=0.;
    flg[0]=1;
  }
  else
  {
    w[i]=1./w[i];
  }
}
for(i=1;i<=m;i++){
  for(j=1;j<=m;j++){
    v[i][j]=v[i][j]*w[j];
  }
}
sum=0;
for(i=1;i<=m;i++){
  for(j=1;j<=m;j++){
    for(jj=1;jj<=m;jj++){
      sum=sum+v[i][jj]*a[j][jj];
    }
    cop[i][j]=sum;
    sum=0;
  }
}
for(i=1;i<=m;i++){
  for(j=1;j<=m;j++){
    a[i][j]=cop[i][j];
  }
}

free_dvector(rv,1,1,n);
free_dvector(w,1,n);
free_dmatrix(cop,1,n,1,n);
free_dmatrix(v,1,n,1,n);
}
/* Example C program that calls the FTF */

/* The program reads from a file the input /output data and
runs a BLS and then the FTF. This is to show the way variables should be declared
in the main program, and how to use the memory allocation functions. */

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

main()
{

```

```

FILE *fp,*fq;
int co,i,j,z,n,np,k,dimVp1,min,r1,m1,p1,rwflg,ie[1];
int in,min1,n1,m,r,p,dimVp,flg[1],rm,rm1,res,o;
long type[1], imagf[1], mrows[1], ncols[1], namlen, fl, fm;
float **uld,**yld,*u,*y,*ufut,**uold,**yold;
float **umin,**ymin,*time;
double **Yhatf,**Pmat,**Yhatb,**Ef,*err,**error,**Eb,*gamk1,*gank1;
double *gainupa, *gainupr, gamup[1]**par,*yh,**yhmat;
double **Y,**Ybar,*yh2,*par2,*er2,**par3;
char fname[12],ftime[40]="",fmp[40]="",node1[]="ftf.time",fomp[40]="";
char node2[]="ftf.mp",node3[]="ftf.omp",BUFFER[BUFSIZ];
char mov1[]="*.time *.mp *.omp",mov2[]="TEST/";
char name[20], fname2[20], letra[]="w";
clock_t start,end;
system("clear");
printf("Enter # of inputs=?\n"); scanf("%d",&r);
printf("Enter # of outputs=?\n"); scanf("%d",&m);
printf("Enter the order of the system=?\n"); scanf("%d",&p);
printf("Enter the # of data points=?\n"); scanf("%d",&n);
printf("Enter the # of points to be used in the BLS=?\n"); scanf("%d",&min);
printf("Enter the name of the data file=?\n"); scanf("%s",fname);
dimVp=(r+m)*p+r; rm=r+m-1,rm1=rm-1; o=rm+1;
n1=n-1; min1=min-1; dimVp1=dimVp-1; m1=m-1; r1=r-1; p1=p-1;
uld=matrix(0,r1,0,n); yld=matrix(0,m1,0,n1); u=vector(0,r1); y=vector(0,m1);
uold=matrix(0,r1,0,p1); yold=matrix(0,m1,0,p1); umin=matrix(0,r1,0,min1);
Yhatf=dmatrix(0,rm,0,dimVp1);ymin=matrix(0,m1,0,min1);par=dmatrix(0,n1,0,3*m);
Pmat=dmatrix(0,dimVp1,0,dimVp1);Yhatb=dmatrix(0,rm,0,dimVp1);
Ef=dmatrix(0,rm,0,rm); Eb=dmatrix(0,rm,0,rm);ufut=vector(0,r1);
uold=matrix(0,r1,0,p1); yold=matrix(0,m1,0,p1); gamk1=dvector(0,1);
gank1=dvector(0,dimVp1); gainupa=dvector(0,rm); gainupr=dvector(0,dimVp1);
err=dvector(0,rm); error=dmatrix(0,n-min1,0,m1); time=vector(0,n);
yhmat=dmatrix(0,n-min1,0,m1); yh=dvector(0,rm);yh2=dvector(0,n*m-1);
er2=dvector(0,n*m-1); par2=dvector(0,n*m*dimVp);Ybar=dmatrix(1,m,1,dimVp);
par3=dmatrix(0,n*m,0,dimVp);Y=dmatrix(1,m,1,2*dimVp);
if((fp=fopen(fname,"rt"))==0)
{
printf("cannot open the file\n");
exit(1);
}
for(i=0;i<=r1;i++){
for(j=0;j<=n1;j++) fscanf(fp,"%f",&uld[i][j]);
}
for(i=0;i<=m1;i++){
for(j=0;j<=n1;j++) fscanf(fp,"%f",&yld[i][j]);
}
fclose(fp);
for(i=0;i<=r1;i++) uld[i][n]=0;
for(i=0;i<=min1;i++){
for(j=0;j<=r1;j++){
umin[j][i]=uld[j][i];
}
for(j=0;j<=m1;j++){
ymin[j][i]=yld[j][i];
}
}

```

```

    }
in=0;
ftfni(umin,ymin,dimVp,Yhatf,Yhatb,Ef,Eb,gamk1,gaink1,r,m,p,min1,flg);
for(i=0;i<=rm;++){
    for(j=0;j<=dimVp1;j++){
        printf("%f\n",Yhatf[i][j]);
    }
    printf("input a # to cont.\n");
    scanf("%d",&res);
}
co=0;
k=min1-p; np=n1-p;
while(k<=np){
    for(i=0;i<=p1;i++){
        for(j=0;j<=r-1;j++){
            uold[j][i]=uld[j][k+i];
        }
    }
    for(j=0;j<=r-1;j++){
        u[j]=uld[j][k+p];
        ufut[j]=uld[j][k+1+p];
    }
    for(i=0;i<=p1;i++){
        for(j=0;j<=m-1;j++){
            yold[j][i]=yld[j][k+i];
        }
    }
    for(j=0;j<=m-1;j++){
        y[j]=yld[j][k+p];
    }
    start=clock();

ftf(ufut,u,y,uold,yold,m,r,p,Yhatf,Yhatb,gainupa,Ef,err,Eb,gamk1,gainupr,gaink1,dimV
p,flg,gamup,yh);
end=clock();
time[co]=(end-start)/1000000.0;
for(i=r;i<=rm;i++){
    for(j=0;j<=dimVp1;j++){par3[in][j]=Yhatf[i][j];
        in=in+1;}
    for(j=r;j<=rm;j++){
        error[co][j-r]=err[j];
        yhmat[co][j-r]=yh[j];
    }
    k=k+1;
    co=co+1;
}
printf("%d\n",co);
in=0;
for(i=0;i<=m1;i++){
    for(j=0;j<=n-min1-1;j++){
        er2[in]=error[j][i];
        yh2[in]=yhmat[j][i];
        in=in+1; } }
in=0;
for(i=0;i<=dimVp1;i++){

```

```

        for(j=0;j<=(n-min)*m-1;j++){
            par2[in]=par3[j][i];
            in=in+1; } }
    for(i=0;i<=rm;+i){
        for(j=0;j<=dimVp1;j++){
            printf("%f\n ",Yhatf[i][j]);
        }
        printf("input a # to cont.\n");
        scanf("%d",&res);
    }
    printf("enter matlab file name=\n");scanf("%s",fname2);
    fl=20; fm=2;
    mopen(fname2,letra,&rwflg,fl,fm);
    type[0]=1000;
    mrows[0]=m*(n-min);
    ncols[0]=dimVp;
    imagf[0]=0;
    namlen=5;
    sprintf(name,"par2");
    msave(type,name,mrows,ncols,imagf,par2,par2,namlen);
    type[0]=1000;
    mrows[0]=n-min;
    ncols[0]=m;
    imagf[0]=0;
    namlen=6;
    sprintf(name,"error");
    msave(type,name,mrows,ncols,imagf,er2,er2,namlen);
    type[0]=1000;
    mrows[0]=n-min;
    ncols[0]=m;
    imagf[0]=0;
    namlen=3;
    sprintf(name,"yh");
    msave(type,name,mrows,ncols,imagf,yh2,yh2,namlen);
    mclose();
    free_matrix(uld,0,r1,0,n); free_matrix(yld,0,m1,0,n1); free_vector(u,0,r1);
    free_matrix(uold,0,r1,0,p1); free_matrix(yold,0,m1,0,p1); free_vector(y,0,m1);
    free_matrix(umin,0,r1,0,min1); free_matrix(ymin,0,m1,0,min1);
    free_vector(ufut,0,r1);free_dmatrix(Yhatf,0,rm,0,dimVp1);
    free_dvector(err,0,rm);free_dmatrix(error,0,n-min1,0,m1);
    free_vector(time,0,n-min1); free_dvector(gainupr,0,dimVp1);
    free_dvector(gainupa,0,rm); free_dvector(gaink1,0,dimVp1);
    free_dvector(gamk1,0,1);free_dmatrix(Ef,0,rm,0,rm);free_dmatrix(Eb,0,rm,0,rm);
    free_dmatrix(Pmat,0,dimVp1,0,dimVp1);free_dmatrix(Yhatb,0,rm,0,dimVp1);
    free_dmatrix(par3,0,(n-min)*m,0,dimVp);free_dvector(par2,0,(n-min)*m*dimVp);
    free_dvector(yh2,0,(n-min)*m-1); free_dvector(er2,0,(n-min)*m-1);
}

```

```

/* Makefile Example */
/* This example program uses the C function from the MATLAB library
to save vectors and matrices from a FORTRAN program in MATLAB form.
The function is called "loadsav.c". */

```

```

#-----#
#                                     #
# Module : Makefile                   #
#                                     #
#                                     #
#-----#
CFLAGS = -g -Q -lm -o
OPT     = -O
MAPON   = -bloodmap:sysid.map
LIB     = /camac/lib/
XREF    = -qxref=full
all: mainftf

mainftf:
    cc mainftf.c nrutil.o svdcmp.o pythag.o ftfb.o \
        ftfini.o ftf.o loadsav.o \
        $(MAPON) $(XREF) $(CFLAGS) mainftf

```

8.3.1 MATLAB Version of the RLS

```

/* rls.m */

function [Yhat,Pmat,t,err]=rls(u,y,m,r,p,Yhatf,Ppk2,n,k,t,dimVp)

% Recursive Least Square Algorithm
%
% Variables:
%Input:
%    p  :- system order
%    m  :- number of outputs
%    r  :- number of inputs
%    u  :- inputs <r X n>
%    y  :- measured outputs <m X n>
%    n  :- total number of points
%    dimVp :- (r+m)*p+r
%    k  :- time index
%Input/Output:
%    Yhatf  :- assumed or initial parameters <m X dimVp>
%    Ppk2:- assumed or initial covariance matrix <dimVp X dimVp>
%    t  :- time history of the estimated parameters
%Output:
%    Yhat :- estimated parameters <m X dimVp>
%    Pmat :- covariance matrix <dimVp X dimVP>

```

```

%      err :- time hist. of error between measured and estimated outputs
%
%      To understand better the programs the equation number appearing in the
% algorithms of the thesis are also included in the programs.
% program by:
%      Jose L. Maldonado-Salgado
%
%      George Washington University
%      Spacecraft Dynamics Branch - NASA Langley
%
err=[];
k=k+1;
while k <= n,
    y1=y(:,k);
    block=[k-1:-1:k-p];
    ulast=u(:,k);
    ugrp=u(:,block);
    ygrp=y(:,block);
    mrg=[ygrp;ugrp];
%      Eq. # 1
    vp=[ulast(:);mrg(:)];
    tmp=vp'*Ppk2;
%      Eq. # 2
    scalar=1+tmp*vp;
%      Eq. # 3
    gainvec=tmp/scalar;
%      Eqs. # 4 & # 5
    dif=y1-Yhatf*vp;
%      Eq. # 6
    Pmat=Ppk2*(eye(dimVp)-vp*gainvec);
%      Eq. # 7
    Yhat=Yhatf+dif*gainvec;
    t=[t;Yhat];
    err=[err dif];
    Yhatf=Yhat;
    Ppk2=Pmat;
    k=k+1;
end

```

8.3.2 MATLAB Version of FTF

```

/* ftf.m */

% Forward-Time Estimation %

function [Yhatf,Yhatb,t,err]=ftf(u,y,m,r,p,n)

%
% Variables:
%      r :- number of inputs
%      m :- number of outputs
%      n :- total number of data pairs
%      p :- system order

```

```

% u :- inputs <r X n>
% y :- outputs <m X n>
% Yhatf :- forward time estimation <(r+m) X dimVp>
% Yhatb :- backward time estimation <(r+m) X dimVp>
% Ef :- forward time estimation error squares <(r+m) X (r+m)>
% Eb :- backward time estimation error squares <(r+m) X (r+m)>
% efik :- forward time apriori output error <(r+m) X 1>
% efdk :- forward time posteriori output error <(r+m) X 1>
% ebi :- backward time apriori output error <(r+m) X 1>
% ebd :- backward time posteriori output error <(r+m) X 1>
% gaink1:- gain vector <1 X dimVp>
% gainup:- update of the gain vector <1 X dimVp+r+m>
% gainupr :- first dimVp elements of the gain vector update <1 X dimVp>
% gainupa :- last r+m elements of the gain vector update <1 X (r+m)>
% gamk1 :- conversion factor
% gamup :- update of the conversion factor
% err :- time history of error between measured and estimated outputs
% t :- time history of the estimated parameters
%
% To understand better the programs the equation number appearing in the
% algorithms of the thesis are also included in the programs.
% program by:
% Jose L. Maldonad0-Salgado
%
% George Washington University
% Spacecraft Dynamics Branch-NASA Langley
%
% Initialization
o=r+m; m1=1; err=[]; t=[]; k1=p; g=2;
dimVp=p*o+r; k=dimVp*g+p; start=k;
vp=zeros(dimVp,1); u=[u zeros(r,1)];
while k1>0,
    ulast=u(:,k1);
    block=[k1-1:-1:m1];
    if isempty(block) == 0,
        ugrp=u(:,block);
        ygrp=y(:,block);
        mrg=[ygrp;ugrp];
        vp2=[ulast(:);mrg(:)];
    else
        vp2=[ulast(:)];
    end
    [w,z]=size(vp2);
    if w < dimVp,
        aug=zeros(dimVp-w,1);
        vp2=[vp2;aug];
    end
    Vp2=[vp2 Vp2];
    k1=k1-1;
end
for i=1:dimVp*g+1,
    block=[i+p-1:-1:i];

```

```

        ulast=u(:,i+p);
        ugrp=u(:,block);
        ygrp=y(:,block);
        mrg=[ygrp;ugrp];
        if i==dimVp*g+1,
%      Eq. # 1
            vp=[ulast(:);mrg(:)];
        else
            vp2=[ulast(:);mrg(:)];
%      Eq. # 2
            Vp2=[Vp2 vp2];
        end
    end
%      Eq. # 3
    Vp1=[Vp2 vp];
    block=[m1+1:k+1];
    ugrp=u(:,block);
    block=[m1:k];
    ygrp=y(:,block);
%      Eq. # 4
    yfork1=[ugrp;ygrp];
%      Eq. # 5
    yf1=[u(:,1);zeros(m,1)];
    block=[m1:dimVp*g];
    ugrp=u(:,block);
    ygrp=y(:,block);
    mrg=[ygrp;ugrp];
%      Eq. # 6
    yuk1p=[zeros(r+m,p+1) mrg];
%      Eq. # 7
    Ppk2=pinv(Vp2*Vp2');
    tmp=vp'*Ppk2;
    scalar=1+tmp*vp;
%      Eq. # 8
    gaink1=tmp/scalar;
%      Eq. # 9
    Yhatf=yfork1*Vp2'*Ppk2; t=[t; Yhatf];
%      Eq. # 10
    Ef=yf1*yf1'+(yfork1-Yhatf*Vp2)*(yfork1-Yhatf*Vp2)';
%      Eq. # 11
    gamk1=1-gaink1*vp;
%      Eq. # 12
    Ppk1=Ppk2*(eye(dimVp)-vp*gaink1);
%      Eq. # 13
    Yhatb=yuk1p*Vp1'*Ppk1;
%      Eq. # 14
    Eb=(yuk1p-Yhatb*Vp1)*(yuk1p-Yhatb*Vp1)';
    k=k+1;
% Forward Time Update
    co=1;
    for k=k:n;
        ulast=u(:,k);
        block=[k-1:-1:k-p];
        ugrp=u(:,block);

```

```

    ygrp=y(:,block);
    mrg=[ygrp;ugrp];
%   Eq. # 15
    vp=[ulast(:);mrg(:)];
%   Eq. # 16
    yf=[u(:,k+1);y(:,k)];
%   Eq. # 17
    efik=yf-Yhatf*vp;
%   Eq. # 18
    efdk=gamk1*efik; err=[err efdk];
%   Eq. # 20
    Ef=Ef+efdk*efik';
    mk=rank(Ef);
    if mk == r+m,
        iEf=pinv(Ef);
%   Eq. # 19
        Yhatf=Yhatf+efik*gaink1; t=[t; Yhatf];
        ga=efdk'*iEf;
        gr=gaink1-ga*Yhatf;
%   Eq. # 21
        gainup=[ga gr];
%   Eq. # 23
        gainupa=gainup(:,dimVp+1:dimVp+o);
        gainupr=gainup(:,1:dimVp);
%   Eq. # 22
        gamup=gamk1-ga*efdk;
        j=k+1;
        [gaink1,Yhatb,gamk1,Eb]=ftfb(u,y,p,j,k,Yhatb,gainupa,gainupr,gamup,Eb);
    else
        j=k+1;
        [gaink1,Yhatb,gamk1,Eb]=ftfb(u,y,p,j,k,Yhatb,gainupa,gainupr,gamup,Eb);
    end
    co=co+1;
end

```

/* ftfb.m */

% Backward-Time Estimation %

function[gaink1,Yhatb,gamk1,Eb]=ftfb(u,y,p,j,k,Yhatb,gainupa,gainupr,gamup,Eb)

```

%
% Variables:
%   r   :- number of inputs
%   m   :- number of outputs
%   n   :- total number of data pairs
%   p   :- system order
%   u   :- inputs <r X n>
%   y   :- outputs <m X n>

```

```

%      k  :- time index
%      j  :- k + 1
%      Yhatb :- backward time estimation <(r+m) X dimVp>
%      Eb   :- backward time estimation error squares <(r+m) X (r+m)>
%      ebi  :- backward time apriori output error <(r+m) X 1>
%      ebd  :- backward time posteriori output error <(r+m) X 1>
%      gainupr :- first dimVp elements of the gain vector update <1 X dimVp>
%      gaink1 :- gain vector update <1 X dimVp>
%      gainupa :- last r+m elements of the gain vector update <1 X (r+m)>
%      gamk1 :- conversion factor
%      gamup :- update of the conversion factor
%
%      To understand better the programs the equation number appearing in the
%      algorithms of the thesis are also included in the programs.
%
%      program by:
%      Jose L. Maldonad0-Salgado
%
%      George Washington University
%      Spacecraft Dynamics Branch-NASA Langley
%      Backward Time Update for the Fast Transversal Filter

      block=[j-1:-1:j-p];
      ulast=u(:,j);
      ugrp=u(:,block);
      ygrp=y(:,block);
      mrg=[ygrp;ugrp];
%      Eq. # 24
      vp=[ulast(:);mrg(:)];
      ugrp=u(:,k-p);
      ygrp=y(:,k-p);
%      Eq. # 25
      yu=[ygrp(:);ugrp(:)];
%      Eq. # 26
      ebi=yu-Yhatb*vp;
      tmp2=1-gainupa*ebi;
%      Eq. # 27
      gaink1=(gainupr+gainupa*Yhatb)/tmp2;

%      Eq. # 28
      Yhatb=Yhatb+ebi*gaink1;
%      Eq. # 29
      gaink1=gamup/tmp2;
%      Eq. # 30
      ebd=gamk1*ebi;
%      Eq. # 31
      Eb=Eb+ebd*ebi;

```

8.3.3 MATLAB Version of LF

```
/* lslf2.m */
```

```
function [Yhatfc, Yhatb, err, t]=lslf2(u,y,m,r,p,n)
```

```
%  
% r :- number of inputs  
% m :- number of outputs  
% n :- total number of points  
% p :- maximum assumed system order  
% u :- inputs  
% y :- outputs  
% err :- time history of error between the measured and estimated outputs  
% Yhatfc:- forward time estimation  
% Yhatb :- backward time estimation  
% t :- time history of the estimated parameters  
%  
% To understand better the programs the equation number appearing in  
% the thesis are also included in the programs.  
% program by:  
% Jose L. Maldonado-Salgado  
%  
% George Washington University  
% Spacecraft Dynamics Branch-NASA Langley  
%  
% Initialization at k=0  
s=.0001; d=10000; t=[]; o=r+m; pmax=p-1; gam0=1;  
u=[zeros(r,1) u zeros(r,1)]; err=[];  
y=[zeros(m,1) y];  
% Eq. # 1  
P0=d*eye(r);  
% Eq. # 2  
G0=zeros(1,r);  
% Eq. # 4  
Yhatf=zeros(o,r);  
% Eq. # 6  
Efc=s*eye(o);  
ic=pinv(Efc);  
ep=0*ones(o,1);  
delt=zeros(o);  
for i=1:pmax+1,  
iEbc=[iEbc ic];  
% Eq. # 7  
Ebc=[Ebc Efc];  
% Eq. # 9  
ebdc=[ebdc ep];  
% Eq. # 8  
delta=[delta delt];  
% Eq # 5
```

```

    Yhatb=[Yhatb zeros(o,(o*(i-1)+r))];
end
Ebc2=Ebc;
ec=ebdc;
Yhatbc=Yhatb;
% Initialization at k >= 1 and p=0
k=1;
while k<=n+1,

% Forward Time Estimation
p=0;
% Eq. # 10
v=u(:,k);
% Eq. # 11
yf=[u(:,k+1);y(:,k)];
% Eq # 12
efi=yf-Yhatf*v;
% Eq. # 13
efd=gam0*efi;
efdc=efd;
% Eq # 14
Efc=Efc+efd*efi';
% Eq. # 15
Yhatf=Yhatf+efi*G0;
Ef=Efc;
Yhatfc=Yhatf;
rnk=rank(Efc);
if rnk ~= r+m,
    flag=5;
    keyboard
else
    iEfc=pinv(Efc);
end

% Backward Time Estimation
% Eq. # 16
v=u(:,k+1);
% Eq. # 17
yf=[y(:,k);u(:,k)];
% Eq. # 18
ebi=yf-Yhatb(:,1:r)*v;
tmp=v'*P0;
% Eq. # 19
G0=tmp/(1+tmp*v);
% Eq. # 20
P0=P0*(eye(r)-v*G0);
% Eq. # 21
Yhatbc(:,1:r)=Yhatb(:,1:r)+ebi*G0;
gam1=gam0;
% Eq. # 22
gam0=1-G0*v;
% Eq. # 23
ebd=gam0*ebi;
% Eq. # 24

```

```

Ebc2(:,1:o)=Ebc(:,1:o)+ebd*ebi';
ec(:,1)=ebd;
rnk=rank(Ebc2(:,1:o));
if rnk ~= r+m,
    flag=6;
    keyboard
else
    iEb=pinv(Ebc2(:,1:o));
end
% Time Update Coefficient Recursion p>=0
p=0; j=1; in=1; i=1; l=0;
while p <= pmax,
%   Eq. # 25
    dlt(:,i:o*j)=delta(:,i:o*j)+efdc*ebdc(:,l+1)/gam1;

% Forward Time Order Update Recursion p>=0
%   Eq. # 26
    Lf=-dlt(:,i:o*j)*iEbc(:,i:o*j);
    Yhf=Yhatfc;
    Yhb=Yhatb(:,in:in+o*l+r-1);
    ef=efdc;
    eb=ebdc(:,l+1);
%   Eq. # 27
    Yhatfc=[Yhatfc+Lf*Yhb -Lf];
%   Eq. # 28
    efdc=efdc+Lf*eb;
%   Eq. # 29
    Ef=Ef-dlt(:,i:o*j)*iEbc(:,i:o*j)*dlt(:,i:o*j)';
    rnk=rank(Ef);
    if rnk ~= r+m,
        flag=7;
        keyboard
    else
        iEf=pinv(Ef);
    end
% Backward Time Order Update Recursion p>=0
%   Eq. # 30
    Lb=-dlt(:,i:o*j)*iEfc;
    base=r+l*o+in;
%   Eq. # 31
    Yhatbc(:,base:(o*(l+1)+r+base-1))=[-Lb Yhb+Lb*Yhf];
%   Eq. # 34
    gam1=gam1-(ebdc(:,l+1)*iEbc(:,i:j*o)*ebdc(:,l+1));
%   Eq. # 32
    ec(:,l+2)=ebdc(:,l+1)+Lb*ef;

%   Eq. # 33
    Ebc2(:,i+o:o*(j+1))=Ebc(:,i:o*j)-dlt(:,i:o*j)*iEfc*dlt(:,i:o*j);
    rnk=rank(Ebc2(:,i+o:o*(j+1)));
    if rnk ~= r+m,
        flag=8;
        keyboard
    else
        iE=pinv(Ebc2(:,i+o:o*(j+1)));

```

```

end
iEfc=iEf;
iEb=[iEb iE];
i=i+o; p=p+1; j=j+1; in=in+(r+m)*l+r; l=l+1;
end
t=[t; Yhatfc]; err=[err efdc];
k=k+1;
delta=dl;
iEbc=iEb;
iEb=[];
ebdc=ec;
Ebc=Ebc2;
Ebc2=[];
Yhatb=Yhatbc;
Yhatbc=[];
end

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100