NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

FAULT DETECTION AND ISOLATION FOR THE BLUEBIRD TEST BED AIRCRAFT

by

Mario J.L. Lévesque

December, 1993

Thesis Advisor: Isaac I. Kaminer
Thesis Co-Advisor: Harold A. Titus

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release: distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*if applicable*) EC	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*if applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
Fault Detection and Isolation for the Bluebird Test Bed Aircraft

12. PERSONAL AUTHOR(S)
Lévesque, Mario J.L.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Master's Thesis	FROM 07/91 TO 12/93	December 1993	117

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Fault Detection and Isolation, Failure detection, FDI, sensor failure, actuator failure, Multiple Model algorithm, Kalman filter.

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

A Fault Detection and isolation (FDI) algorithm design is presented using the Multiple Model algorithm technique for the Bluebird aircraft being developed at the Naval Postgraduate School. The requirement to maintain high performance in the dynamic system of the aircraft necessitates the use of FDI techniques to detect and isolate malfunctions in the sensors and actuators of the aircraft without using hardware redundancy. The solution presented makes use of analytical redundancy in a bank of Kalman filters. Statistical tests using Bayesian theory are applied on the filter's innovations to perform the task of detection and isolation. The algorithm was developed using MATLAB software from The Math Works, Inc. The work presented in this thesis is related only to the task of FDI. The remaining task of the monitoring system, reconfiguration and continued operation by the observed plant after a failure detection, will not be addressed.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Isaac I. Kaminer	22b. TELEPHONE (*Include Area Code*) (408) 656-2972	22c. OFFICE SYMBOL AA/KA

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

Approved for public release; distribution is unlimited

Fault Detection and Isolation
for the Bluebird Test Bed Aircraft

by

Mario J.L. Lévesque
Captain, Canadian Air Force
B.Eng., Royal Military College, 1988

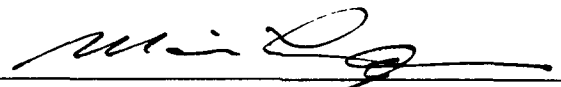
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

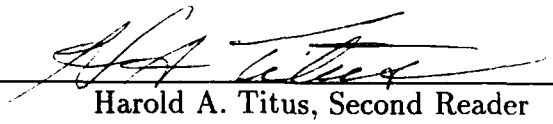
December , 1993

Author: 

Mario J.L. Lévesque

Approved by: 

Isaac I. Kaminer, Thesis Advisor



Harold A. Titus, Second Reader

Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

A Fault Detection and isolation (FDI) algorithm design is presented using the Multiple Model algorithm technique for the Bluebird aircraft being developed at the Naval Postgraduate School. The requirement to maintain high performance in the dynamic system of the aircraft necessitates the use of FDI techniques to detect and isolate malfunctions in the sensors and actuators of the aircraft without using hardware redundancy. The solution presented makes use of analytical redundancy in a bank of Kalman filters. Statistical tests using Bayesian theory are applied on the filter's innovations to perform the task of detection and isolation. The algorithm was developed using MATLAB software from The Math Works, Inc. The work presented in this thesis is related only to the task of FDI. The remaining task of the monitoring system, reconfiguration and continued operation by the observed plant after a failure detection, will not be addressed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express my appreciation to Professor Isaac Kaminer for his advice and professional counsel without which I could not have completed this work. I would also like to thank Professor Hal Titus and Professor Roberto Cristi for their help and discussion during the Kalman filter design phase of this thesis. Thanks also to Carl Marquis for his assistance with LaTeX and Unix. Special thanks go to Michèle Girard for her unfailing support and many words of encouragement over the past several months. I dedicate this thesis to you.

I. INTRODUCTION TO FAULT DETECTION AND ISOLATION

The field of Fault Detection and Isolation (FDI) has evolved as a result of efforts to produce better control algorithms for dynamic systems. Continually higher standards of performance, reliability and survivability dictate the development of new control techniques to achieve certain hardware goals. Hardware performance goals can be related to lowering noise, eliminating biases and meeting bandwidth specifications for sensors actuators and other mechanical components. Reliability specifications are evolving and form the basis for determining the minimum hardware and software complement to deal with performance requirements. An example of a reliability specification is that which applies to aircraft and dictates the probability of catastrophic failure to be at 10^{-9} or better for civilian aircraft or at 10^{-4} to 10^{-5} for military applications [Ref. 1, p.1-1]. The survivability criterion applies more to military applications where the continuation of a mission is important after the system has been partly damaged. FDI techniques can be applied to the design of Flight Guidance and Control Systems to increase their reliability, lower the cost associated with hardware redundancy, and improve aircraft survivability by allowing more dispersion of components [Ref. 1]. The main task of failure detection and compensation is to modify the normal mode configuration in order to include the capability of detecting abnormal changes and compensating for them by activating back-up systems, adjusting feedback gains, or taking other correction measures. The primary function of the FDI algorithm itself is to register an alarm when an abnormal condition develops in a monitored system and to identify the component at the source of the problem. The failure detection and isolation system should provide the capability

to detect the occurrence of failures in a given system and isolate the faulty sensors or hardware, all in the presence of noise and errors in the model. The basis for the decision in the event of a fault is the fault signature, in other words, a signal that is obtained from some kind of faulty system model defining the effects associated with a fault. The difficulty associated with the design of FDI systems lies in maintaining their ability to detect small failures, while avoiding false alarms in the presence of noise and erroneous models which cause effects similar to the failures signatures.

A. SUBDIVISIONS OF FDI

FDI systems are concerned with determining the details of a particular failure such as its type, its source, its location and its size.

1. Type and Size of Failure

The type of a failure can be divided into two parts:

- Abrupt or hard faults.
- Soft or incipient faults.

An abrupt fault can also be described as hard or catastrophic, and results in drastic changes in the model. On the other hand, an incipient or soft failure is characterized by a slowly developing and time varying perturbation in the system.

2. Sources of Failure

The sources of failures in Flight Control Systems are usually described by actuator or sensor failures. A failed sensor can be detected by hardware redundancy. This method is implemented by using several sensors to measure the same signal. The outputs can be compared in a voting scheme in order to detect significant differences between the repeated sensors and isolate a faulty sensor. This FDI method can become very costly for aircraft using hundreds of sensors, and this hardware multiplication can also significantly increase the overall weight of the aircraft. The additional

space required to accommodate the equipment also becomes a major concern in the already limited and crowded environment inside an aircraft. The use of hardware redundancy to detect erroneous actuators or system failures is usually not practical because the replication of components other than sensors is not feasible [Ref. 2]. The problems associated with hardware redundancy motivated the introduction of the concept of analytical redundancy in FDI. In regards to present aircraft, the onboard use of highly efficient computers capable of mathematically intensive calculations, now renders possible the use of algorithms to perform the task of FDI. The analytical redundancy approach uses the system model instead of hardware redundancy to detect and isolate failures in the observed system.

3. Location of Failures

The term location, identification or isolation are often used interchangeably in FDI when determining which sensor or actuator is at fault. The level of coupling in the system is a criteria to consider when classifying the isolation problem. For example, jet engine and flight control systems involve plants with strong coupling. Their subsystems are also strongly coupled, and because of this, usually not all the state variables are being measured. To achieve the isolation process for these systems use of analytical redundancy is necessary [Ref. 2].

B. FDI VERSUS BUILT IN TEST TECHNIQUE

FDI techniques differ from other error detection schemes implemented by means of Built-in Test Equipments (usually called BITE's). BITE are usually designed to test for hardware malfunction using hardware or software methods such as:

- CPU tests to check the instruction repertory.
- Parity control on data and address buses.
- Wrap around circuitries used to test two way interfaces and mainly useful at system level for integration purposes.

- Power sensing circuitries to detect when the applied power goes out of the expected range.
- RAM tests through write-then-read cycles performed on all the memory locations and with particular patterns.
- ROM tests performed by checking the run time computed checksums against prestored ones.
- Wrap around tests that require data management.
- Special-to-project logic checks that control the global development of the system versus time. [Ref. 3, 4]

These tests are implemented in self-monitored systems and run in external self-contained units. BITE's often make use of the concept of graceful degradation when the loss of redundancy would result in a penalization that would be considered too high. For some failures, specified degraded modes of operation are better suited. This approach is generally dealt with by specifying for each important function a nominal operation and a reduced level of operation. The BITE approach is more limited than the FDI using analytical redundancy in that it can only monitor the systems or signals that are physically measurable. An FDI algorithm on the other hand can monitor a wider array of parameters. Even though the sensors in a given system are dissimilar they are all driven by the same dynamic states and are therefore functionally related. This is what is referred to as analytical redundancy, inherent redundancy, or functional redundancy, as opposed to hardware redundancy. The work presented in this thesis will be oriented towards the development of an FDI algorithm using analytical redundancy.

C. FDI ALGORITHM PERFORMANCE EVALUATION

Many criteria are used to evaluate the performance of an FDI algorithm but the most prominent are:

- Rapidity of detection.

- Sensitivity to slowly developing faults.
- False alarm rate.
- Missed failure detection.
- Incorrect fault isolation. [Ref. 5, p. 7]

1. Failure Simulation

The standard method for testing the performance of an FDI scheme is to simulate a fault, maintain it, and look at the reaction of the algorithm. The faults of a dynamic system can be simulated in various ways. For example, a sensor fault on a recorded signal can be reproduced by adding noise or multiplying the signal by a constant, and the behavior of the fault detection device can then be observed. This method can be repeated to test and improve the FDI algorithm. More details are presented in a later chapter to show how the failures were simulated for the FDI algorithm presented in this thesis. It should be mentioned that the technique of emulating faults on one signal creates little or no effect on the other signals. This situation could differ significantly from what could happen in a feedback process subjected to some failure. In this case one fault would often affect several other signals.

2. Rapidity of Detection

The output of the detection system should either indicate that a particular system has failed or give no response if the system is still serviceable. The rapidity of detection plays a role depending on the purpose of the FDI implementation. In aircraft applications, the rapidity of detection is instrumental in ensuring the success of the mission, especially on the vital control sensor of jet aircraft equipped with stability augmentation systems. However, if the FDI algorithm is to be employed in a system for preventive maintenance purpose, the algorithm should be tuned to

detect slowly developing faults in the form of biases or noise at the expense of speed of detection.

3. False Alarm

False alarm occurs when an error is detected while the system is still operating properly. False alarms are indicative of a poor FDI algorithm. Effort should be made during the design of the algorithm to minimize the occurence of false alarms as they lead to lack of confidence in the system. Unfortunately the minimization of the false alarm rate is often done at the expense of detecting small errors. A trade–off between these two criteria must be made. To address this issue, the question of how important an undetected failure would affect the system must be answered. Sometimes the seriousness of the fault is such that it is better to respond to a false alarm by changing the suspected component and retesting it later than leaving it in operation. This discussion leads to the choice of thresholds in the algorithm. Clark and Walker present two interesting approaches to solve the threshold determination problem by presenting adaptive threshold selection methods in chapters 2 and 14 of [Ref. 5].

4. Incorrect Fault Isolation

False identification of the failure source is another effect to be minimized in the design of the algorithm. An erroneous identification of a correctly detected fault causes the reconfiguration system to compensate for the wrong sensor or actuator and can lead to multiple other failures.

D. ROBUSTNESS OF FDI ALGORITHM

The performance criterion of the FDI algorithm leads to the concept of robustness. Robustness of the algorithm is a measure of how the performance can remain unaffected by variation in parameters unaccounted for in the design or by conditions in the operating system that vary differently than what was assumed during the

modeling.

The primary sources of divergence from modeled deterministic systems are noise and uncertainties in the physical parameters of the operating plant. Most of the signal processing techniques used in the field of FDI treat the problem of noise by assuming that contributions have random fluctuations that follow stationary Gaussian process. If the actual system has disturbances and noise that are non-stationary andor non-Gaussian than the FDI algorithm performance will be degraded. Nonlinear models can also contribute to the inaccuracy of the model. In particular, when the parameters of the model are derived from a linearization process around some nominal conditions. The majority of the FDI techniques are based on state estimation methods common to linear systems theory. If the parameters of the model are known with precision, the state estimates representing the modeled plant will be accurate, the FDI scheme will be sensitive to incipient faults and the false alarm rate will be kept to a minimum [Ref. 5, p. 10].

Robustness with respect to the type of fault should also be considered in the design, since any given components can malfunction in many different ways. Sensors can experience biases, change of scale factor, wear and tear, friction, variation with hot or cold temperature, etc. All these symptoms can cause any given component to fail in a different pattern. If the FDI algorithm is set to detect and identify only a few of the many types of faults for any sensor this will limit its performance. Therefore, the FDI scheme should be built on taking the widest repertoire of fault signatures and should also be capable of accommodating any new types of failures as they develop.

E. PROGRESS IN FDI

Perhaps one of the most cited reference in FDI is the paper published by Willsky in 1976 [Ref. 6]. In his paper Willsky provides an overview of a number of the

basic concepts in failure detection. In particular, he concentrates on linear systems and compares the design of voting schemes, specific failure-sensitive filters, multiple hypothesis filter detectors, statistical tests on filter innovations and development of jump process formulations. Since 1976, several other survey papers have been published in the field of FDI [Ref. 7, 8, 9]. New approaches have been presented using geometrical interpretation of the concept of analytical redundancy. These approaches were developed for determining robust parity relations for failure detection in dynamic systems. The differences in the recently introduced methods are that the problems of model uncertainties are now addressed in detail. [Ref. 10, 2]. To respond to the same problem of model uncertainties, many authors have worked with and developed the theory of Unknown Input Observers(UIO) for use in linear uncertain dynamical systems [Ref. 11, 5]. The last two methods mentioned are part of what is referred to as the model-based approach to FDI. Some progress has also been reported using neural networks or the knowledge based approach. Recent research directions include use of techniques such as fuzzy logic, adaptive threshold selector and $\mathcal{H}_\infty$ observers [Ref. 12].

F. FDI APPLICATION

The FDI algorithm developed in this thesis uses the six degree of freedom model derived for the Bluebird test aircraft by Capt Kuechenmeister USMC [Ref. 13]. The non-linear model can be linearized around typical flight conditions. The Bluebird aircraft is used at the Naval Postgraduate School to test guidance, navigation and control systems in horizontal flight. The physical characteristics of the Bluebird are given in table 2.3 in [Ref. 13, p. 8]. In an aircraft flight control system the actuators are used as servomechanisms which drive the control surfaces and the engine. The actuators receive their input signals from an onboard Flight Management Computer.

The instrumentation of the Bluebird aircraft includes many sensors or transducers attached to the airframe, which provide signals proportional to the motion of the airplane, such as airspeed, altitude, heading, accelerations, attitude and rates of change of attitude, control surface deflections, engine thrust etc.

1. FDI on the Bluebird Aircraft

The four different actuators under test in the Bluebird model correspond to the following control surfaces and engine:

- Elevator.
- Aileron.
- Rudder.
- Thrust.

The sensors include the following parameters in the Bluebird model:

- u for the forward velocity.
- v for the lateral velocity.
- w for the vertical velocity.
- p for the roll rate.
- q for the pitch rate.
- r for the yaw rate.
- Φ (phi) for the Euler roll angle.
- Θ (theta) for the Euler angle of attack.
- Ψ (psi) for the Euler heading angle.

The FDI algorithm implemented uses the Bluebird aircraft model and the multiple model FDI technique described in [Ref. 7] and [Ref. 14]. The equations and explanation on how this algorithm works are presented in Chapter II.

The work presented in this thesis is related to the Fault Detection or Fault Isolation task (FDI). Reconfiguration or the remaining task of the monitoring system to reconfigure and permit continued operation by the observed plant after a fault has been detected and isolated is not addressed.

II. IMPLEMENTATION OF MULTIPLE MODEL TECHNIQUE FOR FDI

A. DESCRIPTION ON THE MULTIPLE MODEL TECHNIQUE

The multiple model method addresses the problem of FDI by observing a sequence of inputs and outputs from a system and then choosing one out of a given set of possible models that is felt most likely to have responded in an observed fashion. This broad definition for multiple model technique does not only apply to FDI problem but also to areas of system identification and adaptive control which prompted the early development of this method [Ref. 15].

The approach to FDI using multiple model technique is achieved as follows: the fault isolation is carried out on the basis of different fault hypothesis and a separate estimator(observer or Kalman filter) is assigned for a finite number of fault hypotheses. These hypotheses are then tested in terms of likelihood functions, using Bayesian decision theory, to detect which fault is present. The algorithm presented in this thesis makes use of a bank of 14 Kalman filters which represent each of the different error hypotheses tested. The first hypothesis is in the case of no failure. The remaining filters are tuned for hard failure in the four actuators and nine sensors. Figure 2.1 represent the flow diagram for the multiple model FDI algorithm.

B. STATE EQUATIONS OF THE AIRCRAFT

The states of the observed plant can be represented by a set of linearized stochastic equations in the time domain. In the following equation the input is represented by $\mathbf{u}$, the output vector by $\mathbf{y}$ and the states of the system by the vector $\mathbf{x}$. All other effects that obscure the fault detection are introduced in the vectors $\mathbf{w}$ and $\mathbf{v}$ and

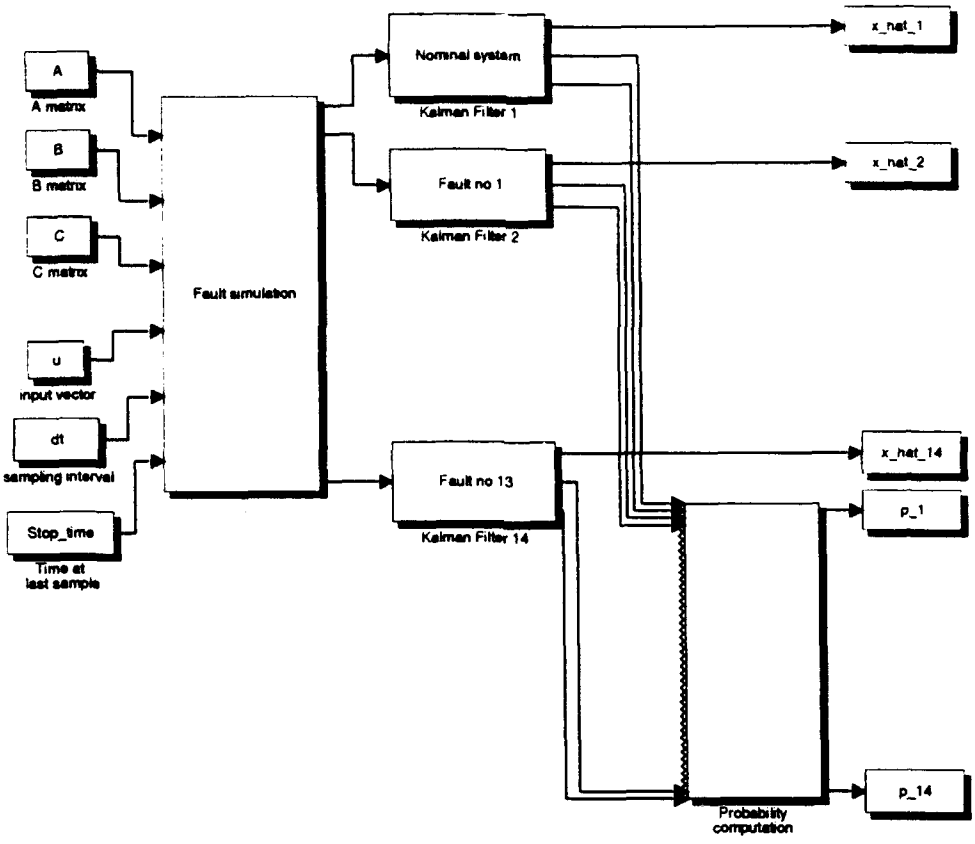


Figure 2.1: FDI Flow Diagram using Multiple Model Algorithm

include parameters such as noise or unknown inputs:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) + \mathbf{W}w(t) \tag{2.1}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{V}v(t) \tag{2.2}$$

In the Bluebird application, $\mathbf{x} \in \mathcal{R}^9$, $u \in \mathcal{R}^4$ and $\mathbf{y} \in \mathcal{R}^9$. The matrices **A, B, C** are known and of compatible dimensions.

1. State-Space Representation of Sampled and Discrete Systems

The solution to equation 2.1 has the following form:

$$\mathbf{x}(t) = \mathbf{\Phi}(t, t_o)\,\mathbf{x}(t_o) + \int_{t_o}^{t} \mathbf{\Phi}(t, \tau)\,\mathbf{B}\,u(\tau)\,d\tau + \int_{t_o}^{t} \mathbf{\Phi}(t, \tau)\,\mathbf{W}\,w(\tau)\,d\tau \tag{2.3}$$

If the system is sampled with sampling interval T, then we are interested in the values of the state $\mathbf{x}$ at discrete time intervals. Using equation 2.3, the value of the state's vector $\mathbf{x}$, at time $t = (k+1)T$ given its value at $t_o = kT$ can be obtained as follows:

$$\mathbf{x}((k+1)T) = \mathbf{\Phi}[(k+1)T, kT]\,\mathbf{x}(kT) + \int_{kT}^{(k+1)T} \mathbf{\Phi}((k+1)T, \tau)\,\mathbf{B}\,u(\tau)\,d\tau + \int_{kT}^{(k+1)T} \mathbf{\Phi}((k+1)T, \tau)\,\mathbf{W}\,w(\tau)\,d\tau \tag{2.4}$$

If $u(\cdot)$ and $w(\cdot)$ are piecewise constant between sampling intervals, that is,

$$u(t) = u(kT) \qquad \text{for } kT < t < (k+1)T$$

and

$$w(t) = w(kT) \qquad \text{for } kT < t < (k+1)T$$

then u and w can be taken out of the integral in equation 2.4 to obtain the sampled or discrete state-space representation

$$\mathbf{x}((k+1)T) = \mathbf{\Phi}[(k+1)T, kT]\,\mathbf{x}(kT) + \underline{\mathrm{f}}\Delta[(k+1)T, kT]\,u(kT) + \mathbf{\Gamma}[(k+1)T, kT]\,w(kT), \tag{2.5}$$

where

$$\mathbf{\Phi}[(k+1)T, kT] = \int_{kT}^{(k+1)T} exp\{\mathbf{A}[(k+1)T, \tau]\}\,d\tau \tag{2.6}$$

is the state transition matrix in the discrete form and

$$\mathbf{\Delta}[(k+1)T, kT] = \int_{kT}^{(k+1)T} \mathbf{\Phi}[(k+1)T, \tau]\,\mathbf{B}\,d\tau \tag{2.7}$$

$$\mathbf{\Gamma}[(k+1)T, kT] = \int_{kT}^{(k+1)T} \mathbf{\Phi}[(k+1)T, \tau]\,\mathbf{W}\,d\tau \tag{2.8}$$

A discrete measurement $\mathbf{y}$ is given by

$$\mathbf{y}(kT) = \mathbf{C}(kT)\,\mathbf{x}(kT) + \mathbf{V}(kT)\,\mathbf{v}(kT) \tag{2.9}$$

If the system is not a sampled continuous system, then T can be ignored and set to one [Ref. 16, p. 215]. The solution to the difference equation 2.5 can be obtained by letting k take on values $k = 0, 1, 2, 3, \ldots, N$ and collecting terms to get the solution of the discrete-time equation over kT samples:

$$\mathbf{x}(kT) = \mathbf{\Phi}(T)\,\mathbf{x}(0) + \sum_{j=0}^{k-1} \mathbf{\Phi}^{k-j-1}(T)\,\mathbf{\Delta}(T)\,u(jT) + \sum_{j=0}^{k-1} \mathbf{\Phi}^{k-j-1}(T)\,\mathbf{\Gamma}(T)\,w(jT) \quad (2.10)$$

The format used in the rest of the document as well as in the code of the FDI algorithm will utilize the nomenclature employed in [Ref. 17] which is a simplification of the equations derived above. The following equations represent the state and measurement equations of a linear plant using this simplified format. Notice that the vectors are represented by using bold characters. In the Bluebird application, the C matrix will be set to be identity since all measurements are equal to the state of the plant:

$$\mathbf{x}(k+1) = \mathbf{\Phi}(k+1,k)\,\mathbf{x}(k) + \mathbf{\Delta}(k+1,k)\,u(k) + \mathbf{\Gamma}(k+1,k)\,w(k) \quad (2.11)$$

$$\mathbf{y}(k) = \mathbf{C}(k)\,\mathbf{x}(k) + \mathbf{V}(k)\,v(k) \quad (2.12)$$

C. KALMAN FILTER EQUATIONS

The following reasons prompted the use of Kalman filters to detect failures. First the algorithm of the Kalman filter allows the calculation of the gain matrix in order to minimize the variance of the estimation error. This feature of optimization will be beneficial if the same estimates obtained from the FDI algorithm are to be used later on in the aircraft autopilot. Another advantage of using an estimation algorithm is that in the absence of hardware redundancy it is still possible to use a degraded instrument through the information provided by the estimator in the reconfiguration mode. However, the main reason for using Kalman filters is that the covariance of the residual or innovation process calculated in order to determine the Kalman gains

are also utilized in the detection calculation using Bayesian theory. This feature of the detection algorithm will be covered in more detail in section D.

The equations developed in the following sections are implemented in the MATLAB code listed in APPENDIX C.

1. Assumptions and Initial Values

a. Noise Assumptions

With the plant equations given by 2.11 and 2.12, the following assumptions are made: The measurement noise has zero mean,

$$E[\mathbf{v}(k)] = 0, \qquad \text{for } k = 0, 1, 2, \ldots \tag{2.13}$$

is uncorrelated,

$$E[\mathbf{v}(k)\mathbf{v}^T(j)] = E[\mathbf{v}(j)\mathbf{v}^T(k)] = 0, \qquad \text{for } j \neq k \tag{2.14}$$

and has covariance

$$E[\mathbf{v}(k)\,\mathbf{v}^T(k)] = \mathbf{R}(k), \qquad \text{for } k = 0, 1, 2, \ldots \tag{2.15}$$

Equations 2.14 and 2.15 can also be expressed with the Kroneker delta as

$$E[\mathbf{v}(k)\mathbf{v}^T(j)] = \mathbf{R}(k)\,\delta_{kj} \tag{2.16}$$

where

$$\delta_{kj} = \begin{array}{ll} 0, & k \neq j \\ 1, & k = j \end{array} \tag{2.17}$$

The random process noise has zero mean

$$E[\mathbf{w}(k)] = 0, \qquad \text{for } k = 0, 1, 2, \ldots \tag{2.18}$$

is uncorrelated, and has covariance

$$E[\mathbf{w}(k)\mathbf{w}^T(j)] = E[\mathbf{w}(j)\mathbf{w}^T(k)] = \mathbf{Q}(k)\,\delta_{kj} \tag{2.19}$$

The random process noise and the measurement noise are uncorrelated

$$E[\mathbf{w}(k)\mathbf{v}^T(j)] = E[\mathbf{v}(j)\mathbf{w}^T(k)] = 0, \qquad \text{for } k, j = 0, 1, 2, \ldots \tag{2.20}$$

b. Initial States Assumptions

The input $\mathbf{u}(k)$ for $k = 0, 1, 2, \ldots$ is known and deterministic.

The initial state is a random variable with known mean

$$E[\mathbf{x}(0)] = \overline{\mathbf{x_0}} \tag{2.21}$$

and covariance

$$E\{[\mathbf{x}(0) - \overline{\mathbf{x_0}}][\mathbf{x}(0) - \overline{\mathbf{x_0}}]^T\} = \mathbf{M} \tag{2.22}$$

the measurement noise and initial states are uncorrelated:

$$E[\mathbf{x}(0)\,\mathbf{v}^T(k)] = E[\mathbf{v}(k)\,\mathbf{x}^T(0)] = 0, \qquad \text{for } k = 0, 1, 2, \ldots \tag{2.23}$$

The random process noise and initial state values are uncorrelated as well:

$$E[\mathbf{w}(k)\mathbf{x}^T(0)] = E[\mathbf{x}(0)\mathbf{w}^T(k)] = 0, \qquad \text{for } k = 0, 1, 2, \ldots \tag{2.24}$$

2. Discrete Kalman Filter Equations

The observer equation is given by

$$\hat{\mathbf{x}}(k/k) = \hat{\mathbf{x}}(k/k-1) + \mathbf{G}(k)[\mathbf{y}(k) - \hat{\mathbf{y}}(k)] \tag{2.25}$$

where

$$\hat{\mathbf{y}}(k) = \mathbf{C}\,\hat{\mathbf{x}}(k/k-1) \tag{2.26}$$

The nomenclature $\hat{x}(k/k-1)$ means estimate of x at time or observation k given measurements at time up to and including $(k-1)$. The goal of the optimal estimator is to minimize the estimation error defined as

$$\mathbf{e}(k/k) \stackrel{\text{def}}{=} \hat{\mathbf{x}}(k/k) - \mathbf{x}(k) \tag{2.27}$$

A zero mean estimation error is sought for all k to obtain an unbiased estimator. The complete development of the gain, covariance, and update matrices in order to achieve the minimization of the variance of the measurement errors can be found in [Ref. 17, p.4-32 to 4-41]. The following equations summarize the different parameters involved in the calculations inside the Kalman filter. The computational steps involve calculations before use of the Kalman filter as well as generation of estimates during the filter operation. The gain schedule can be evaluated prior to using the estimator since the gains do not depend on the measurement data sequence [Ref. 17, p4-42].

The following calculations are used to compute the gains and are implemented in the MATLAB file `kalmn\_gain.m` found in APPENDIX C.

First, let $k = 0$ and set the initial value for the covariance matrix $P(k/k-1) = M$ and then calculate the gain schedule with

$$\mathbf{G}(k) = \mathbf{P}(k/k-1)\,\mathbf{C}^T(k)\,[\mathbf{C}(k)\,\mathbf{P}(k/k-1)\,\mathbf{C}^T(k) + \mathbf{R}(k)]^{-1} \tag{2.28}$$

The term in the bracket of equation 2.28 represents the hypothesized filter's internally computed residual covariance or the innovation covariance. This term will become important in section D. and is denoted as

$$\mathbf{a_i}(k) = \mathbf{C}(k)\,\mathbf{P}(k/k-1)\,\mathbf{C}^T(k) + \mathbf{R}(k) \tag{2.29}$$

Next, compute the covariance of the estimation error matrix

$$\mathbf{P}(k/k) = [\mathbf{I} - \mathbf{G}(k)\,\mathbf{C}(k)]\,\mathbf{P}(k/k-1) \tag{2.30}$$

Now, compute the predicted value for $P(k+1/k)$ using the discrete Riccati equation:

$$\mathbf{P}(k+1/k) = \mathbf{\Phi}(k+1,k)\,\mathbf{P}(k/k)\,\mathbf{\Phi}^T(k+1,k) + \mathbf{\Gamma}(k+1,k)\,\mathbf{Q}(k)\,\mathbf{\Gamma}(k+1,k) \tag{2.31}$$

Repeat equation 2.28, 2.30 and 2.31 recursively until the final observation value for k is reached. Hopefully, k should be large enough so that the gains will have

reached steady-state values. Notice that the last value calculated for $P(k+1/k)$ becomes $P(k/k-1)$ when the values are brought up again at the beginning of this computation loop.

After the gain schedule has been computed, the values are stored in memory and the observer can be implemented. The next equations are required to build the Kalman filter. Once again these equations can be evaluated on a digital computer. The MATLAB code for the Bluebird application can be found at APPENDIX C in file kalman_i.m. First, k is set to zero and $\hat{\mathbf{x}}(0/-1)$ is initialized to $\overline{\mathbf{x}_0}$. After this, $\hat{\mathbf{x}}(k/k)$ can be evaluated when the first observation $\mathbf{y}(k)$ becomes available with

$$\hat{\mathbf{x}}(k/k) = \hat{\mathbf{x}}(k/k-1) + \mathbf{G}(k)\,[\mathbf{y}(k) - \mathbf{C}(k)\,\hat{\mathbf{x}}(k/k-1)] \tag{2.32}$$

The term in the bracket of equation 2.32 is referred to as the innovation process or the residual and is represented by

$$r_i(k) = \mathbf{y}(k) - \hat{\mathbf{y}}(k/k-1) \tag{2.33}$$

The predicted value for $\hat{\mathbf{x}}(k+1/k)$ is then evaluated with

$$\hat{\mathbf{x}}(k+1/k) = \mathbf{\Phi}(k+1,k)\,\hat{\mathbf{x}}(k/k) + \mathbf{\Delta}(k+1,k)\,\mathbf{u}(k) \tag{2.34}$$

Equations 2.32 and 2.34 are also repeated recursively to generate the state estimates. As in the previous procedure, the last value calculated for $\hat{\mathbf{x}}(k+1/k)$ becomes $\hat{\mathbf{x}}(k/k-1)$ when the loop is reentered.

The updated measurement estimates are also calculated in the above steps in equation 2.32 since

$$\hat{\mathbf{y}}(k) = \mathbf{C}(k)\,\hat{\mathbf{x}}(k/k-1) \tag{2.35}$$

D. PROBABILITY OF FAILURE USING BAYESIAN THEORY

As explained earlier, the Multiple Model algorithm makes use of Bayesian theory to perform the task of detection and isolation. The Multiple Model technique involves

the design of a finite set of linear stochastic systems indexed by $i = 1, 2, \ldots, N$ with the i^{th} model being the one that corresponds to the actual model being observed. This leads to the standard multiple hypothesis testing problem. Suppose that H_i denotes the hypothesis that the real system corresponds to the i^{th} model, and $p_i(0)$ the a-priori probability that H_i is true, then similarly $p_i(k)$ can denote the probability that H_i is true for all the measurements up to and including the k^{th} observation. The previous measurements can be represented by the set $M_k = \{u(0), u(1), \ldots, u(k-1), y(1), y(2), \ldots, y(k-1)\}$. The Bayes' rule gives the following formula to calculate $p_i(k)$,

$$p_i(k) = \frac{p(\, y(k)|H_i, M_k\,)\, p_i(k-1)}{\sum_{j=i}^{N} p(\, y(k)|H_j, M_k\,)\, p_j(k-1)} \tag{2.36}$$

where $p(\cdot)$ is the conditional probability density function. The density functions must be calculated at each observation. This density function is conditioned on H_i and is fortunately the same one step prediction density function produced by the i^{th} Kalman filter. This term is represented by a_i in 2.29. Under hypothesis H_i, the residual $r_i(k)$ should have a mean of zero, a covariance of $a_i(k)$ and be normally distributed. Moreover, $y(k)$ conditioned on H_i and M_k should be Gaussian with mean $C_i(k)\,\hat{\mathbf{x}}_\mathbf{i}(k/k-1)$ and with covariance $a_i(k)$. [Ref. 7, p. 2-2] These assumptions lead to the following important formula for the probability density function

$$p(\, y(k)|H_i, M_k\,) = \frac{1}{(2\pi)^{\frac{m}{2}}[\det\, a_i(k)]^{\frac{1}{2}}} exp\{-\tfrac{1}{2} r_i^T(k)\, a_i^{-1}(k)\, r_i(k)\} \tag{2.37}$$

where m is the dimension of y(k).

Equations 2.32, 2.33, 2.34, 2.36 and 2.37 represent the Multiple Model algorithm.

III. ADJUSTMENTS TO THE MULTIPLE MODEL ALGORITHM

A. MODEL SELECTION FOR EACH FILTER

As mentioned in Chapter II. , section A., the Multiple Model algorithm is implemented using a bank of Kalman filters. Each Kalman filter is based or "tuned" for one possible error scenario. The inputs to the filters are the measurements of the different aircraft sensors and the outputs are the innovation sequences $r_i(k)$, which are a measure of how close the estimates are to the true measurements. These residuals should be white sequences for the filter tracking the correct model. If the filter's model is not correct, then the residuals will not be white and will include errors due to the fact that the estimates are based on incorrect models. The probability calculations equation 2.36 give a measure of which model, based on a specific fault, is most likely to be correct compared to other models.

1. Fault Modeling for the Filters and for the Simulation Runs

Each Kalman filter is based on a particular failure; several methods can be used to change the nominal model, i.e., the model representing no failure, to one that is modified to isolate a specific fault.

a. Actuator Failure

Typically, actuator malfunction can be modeled by setting one of the columns of the B matrix to zero for the respective actuator to be at fault. Another way is to increase the covariance of $w(k)$ over the normal range of operation. A third way, equivalent to the previous two, would be to add a driving term $g(k)$ in equation 2.34, [Ref. 8, p.461]. The algorithm developed in this thesis will utilize

the first two schemes to simulate faults and only the zeroing scheme to set the model used by each Kalman filter. This is done in order to keep the number of Kalman filter to a minimum. Work by Menke and Maybeck has also indicated that soft failure can be detected by Kalman filters tuned with hard failure models [Ref. 14, p. 3136]. Therefore, the method employed in this thesis will use the first modeling scenario where the columns of the B matrix are set to zero to tune the Kalman filters. However, during the simulation, the appropriate column of the system's B matrix fed into all the different Kalman filters will only be set to zero for simulation of hard actuator failure. Incipient actuator failure simulation will be done by changing the covariance of the process noise for the system's model. In other words, Kalman filters based on the models with B matrice's column set to zero will be expected to detect both hard and soft failure for the one specific actuator.

b. Sensor Failure

Sensor failures can also be modeled in three ways. The methods are the same as for the three schemes explained above for the actuators except that the zeros are included in the respective sensor's column of the C matrices and the measurement noise $v(k)$ is varied instead of the process noise.

B. TUNING OF THE KALMAN FILTERS

After selecting the models for the design of each filter, it is important to set the parameters of every Kalman filter so that the process of detection will be optimized to prevent the occurrence of false alarms. The Kalman filter is said to be tuned if it provides an optimal or minimal error covariance on the estimates of the states.

The performance of the filter can be evaluated and adjusted by a series of statistical tests. The operation of the estimator in equation 2.32 can be simplified

by the following representation

$$\hat{\mathbf{x}}_{new} = \hat{\mathbf{x}}_{old} + \mathbf{G}[\text{residuals}_{new}] \tag{3.1}$$

From equation 3.1, it is easy to see that the new estimates are formed partly using the old estimates, which are a function of the plant model, and also from the new measurements residuals. As one can see, the value of the new estimates is also a function of the Kalman gains. For small G's, the new estimates use the model and for large gains, the new estimates will be influenced by the measurements. The choice and derivation of the gains is therefore instrumental in the operation of the Kalman filters.

1. Influence of the Kalman Filter's Gains

As can be seen in equation 2.28, the gains are influenced by two variables, $P(k/k-1)$ and $R(k)$. Thus, there are four ways to influence the estimates. First $P(k/k-1)$ are small and $R(k)$ are fixed which gives small gains and is in agreement with the model behavior since small $P(k/k-1)$ means that the model is adequate (small variances on the errors). Second, $R(k)$ are large and $P(k/k-1)$ are fixed, which leads to small gains and also means that the model should be believed since the measurements are noisy. Now for the case where the gains are large. If the $P(k/k-1)$ is large and $R(k)$ is fixed then the model is inadequate and the new measurements should be used. If $R(k)$ is small and $P(k/k-1)$ is fixed, then it means that the measurements are good.

A Kalman filter does not operate correctly if the gains becomes small before they should, meaning that the filter thinks that the new measurements do not carry valuable information when this is actually not the case. The filter is said to diverge when this occurs. To avoid filter divergence a series of tests must be performed.

2. Tests on the Kalman Filter

As a first check for the operation of the Kalman filter, the innovations should form zero-mean and white sequences (see proof in [Ref. 18]). Formulae can be found in [Ref. 16, p.95–97] to evaluate the mean with test statistics and to perform whiteness tests on the covariance of the residuals. These tests can also be performed in MATLAB using the mean(·) and cov(·) functions.

3. The Filter Knobs

The different parameters available in the Kalman filter for tuning will be discussed in this section. The normal approach taken for state estimation in aircraft application is to use a model developed from the aircraft's equations of motion, test it to estimate the noise statistics then construct the filters and adjust them using the data collected in the testing. Once these steps have been done the filters can be evaluated to test their performance. If the designer is not satisfied with the results, adjustments are done until satisfactory performance is obtained. This constitutes the phase called "tuning of the Kalman filter".

Various parameters are changed during the tuning process. These parameters, sometimes called "filter knobs", are the process noise covariance $Q(k)$, the measurement noise covariance $R(k)$, and the initial condition on $P(0/0)$. From equation 2.31, it is apparent that for large $Q(k)$, $P(k+1/k)$ will also become large indicating high uncertainties and an unreliable model. Conversely, if the values for $Q(k)$ are small, $P(k+1/k)$ will be small and the model will become adequate.

Another way to look at varying the knobs is to observe their effects on the filter's transient response and the noise in the state estimates. As $Q(k)$ increase so will the gains and the system transient performance will be faster. However in this case the noise in the state estimates will be larger. The same effect would occur if $R(k)$ would be smaller, but in many cases, the noise on the sensors is fixed. (See

Table 4.2). The second possibility is to lower values for $Q(k)$. This results in lower gains, and therefore in a slower filter transient performance. However, this would also filter more noise from the estimates.

The initial values for $P(0/0)$ also have an effect on the algorithm. As $P(0/0)$ is set to large values, the initial gains will also become large and the initial measurement will be heavily weighted compared to the model which will be considered inadequate. It should be noted however that as more observations become available the effects of the initial values of $P(0/0)$ disappear. This effect is favorable as the initial values for $P(0/0)$ are seldom known. These results will occur only if the filter is stable and the system is observable and controllable. [Ref. 16]

C. PROBLEMS WITH THE PROBABILITY CALCULATIONS

A minimum value problem can arise when using equation 2.36. If the value calculated for $p_i(k)$ is small the next value $p_i(k+1)$ will only grow back slowly if it has to. If $p_i(k)$ becomes zero, then the next values for $p_i(n)$ for $n > k$ will also be set to zero.

To avoid the problem of slow variation in the $p_i(k)$ calculation and the generation of null values, the $p_i(k)$ are set to a minimum value if the calculations fall below a certain level. The bound has been set to 10^{-4} for the application studied in this thesis. This number can be changed easily in the MATLAB file pk.m under the variable name "min" for any other value. However, this one seemed to work fairly well for the Bluebird application and lies in the range recommended by Willsky [Ref. 7, p.2–5].

D. PERFORMANCE OF THE MULTIPLE MODEL ALGORITHM IN DIFFERENT ENVIRONMENTS

1. Non-Gaussian Noise

The Gaussian noise assumption is made in the formulation of equation 2.36 more specifically in the density function of equation 2.37. According to Willsky, [Ref. 7], even in the presence of non-Gaussian residuals, the performance of the algorithm should not be influenced. That is, the algorithm is designed to measure how well each of the Kalman filters is tracking by observing the residuals that are produced. The probability calculations of equation 2.36 only measure how well each filter is estimating data relative to each other and how well they are expected to be tracking. The important term in equation 2.37 is

$$r_i^T(k)\, a_i^{-1}(k)\, r_i(k). \tag{3.2}$$

In fact previous efforts have noted the leading coefficient before the exponential in equation 2.37 provides little information in the identification of a failure [Ref. 14]. If the likelihood quotient that equation 3.2 represents is large, the i_{th} model will be non valid and if it is small, then the i_{th} model is tracking well. This is reflected in the calculation for the $p_i(k)$ and the Multiple Model algorithm will still produce the expected results in the presence of non-Gaussian statistics.

2. Non-Linearities

The usual approach to non-linearities is to linearize the system around a number of operating points that reflect the flight condition of the aircraft for each of the models in the algorithm. These linearized models span the flight envelope and can then be used to design extended Kalman filters which can replace the standard Kalman filters in the Multiple Model algorithm. This has been done in this thesis but around one operating point only. The A and B matrices were generated by Lt D.

TABLE 3.1: NOMINAL CONDITIONS FOR THE BLUEBIRD

Turn rate	0.1 (rad/sec)
Ground speed	80 (ft/sec)
Rate of descent	300 (ft/min)

Hallberg USN in his present thesis work to design a controller for a nonlinear system, such as the flight control system of the Bluebird . The plant was linearized around the conditions in Table 3.1 and the numbers in the A and B matrices were generated using the MATLAB function "linmod". The values are shown in APPENDIX C in file ABCU_load.m. The C matrix is set as the identity.

The problem associated with the linearized approach is to determine whether the tracking errors from the extended Kalman filters corresponding to the linearized model are significantly smaller than the errors from filters based on more distant models. In the nonlinear case, the inaccuracies of the Kalman filters increase the values of the internally generated residual covariance $a_i(k)$ in equation 2.29 and 2.28. This has the effect of reducing the value of the probabilities $p_i(k)$ obtained through equation 2.37. As the values for $a_i(k)$ increase it becomes more difficult to detect errors.

IV. SIMULATION AND RESULTS FOR THE MULTIPLE MODEL FDI ALGORITHM

The procedure to simulate faults and test the Multiple Model FDI algorithm is explained in APPENDIX A.

The simulation included trials to simulate hard failures in the actuators, hard failures in the sensors, soft failures in the actuators and soft failures in the sensors. The results are shown in APPENDIX B. An input function was introduced in the system in order to generate fluctuations in the sensors and drive the states to nonzero values (see file `kalman\_i.m` in APPENDIX C). If a state or sensor is at or close to zero it is very difficult or impossible to detect a failure that is based on the hypothesis that the sensor or actuator will go to zero when it fails.

The following paragraphs will introduce some of the data used to initialize the algorithm prior to testing as well some discussion of the results.

A. SIMULATION PARAMETERS

As discussed in chapter III. section B. 3., the value chosen for the process noise will influence the behavior of the filters. The numbers employed in the simulation are shown in Table 4.1 and can be changed in file ABCU_load.m. These values were selected to be as low as possible in order to have each of the separate filters tuned to follow their respective model closely and to put little weight on the new measurements. This is done to isolate the effect of a hard failures in each specific filter.

It was found that in certain runs where the process noise values were too low, the numbers generated by the Kalman filters in MATLAB would become unstable or out

TABLE 4.1: PROCESS NOISE VALUES

State	Value
u	1.50000 (ft/sec)
v	0.75000 (ft/sec)
w	0.75000 (ft/sec)
p	0.08550 (rad/sec)
q	0.08550 (rad/sec)
r	0.60000 (rad/sec)
Φ	0.00785 (rad)
Θ	0.00785 (rad)
Ψ	0.02618 (rad)

of range and the output of the probability computation were assigned to NaN (Not a Number). NaN is produced in MATLAB when a division of either zero by zero or infinity by infinity occurs. To fix this problem, numbers assigned for the process noise were increased until the algorithm could function properly. Andrews and Grewal in [Ref. 19, p.192–260] provide several interesting implementation methods for Kalman filters where ill-conditioned problems arise. None of these special methods have been tested in this thesis but they should be investigated if more accuracy is to be required in future usage of this algorithm.

The values used for the measurement noise are those provided by the manufacturer of the sensor. These numbers are given in Table 4.2 and were obtained from [Ref. 13, p. 60,61,71]. Samples of white noise were generated in MATLAB to be represented by w and v in equations 2.11 and 2.12 and are shown in figure B.1 and B.2.

B. KALMAN FILTER TESTS

The Kalman filter developed for the algorithm was first tested using the plant described in [Ref. 16, p. 127] and [Ref. 19, p. 143] and the error covariance matrices

TABLE 4.2: MEASUREMENT NOISE VALUES

State	Value
u	1.00000 (ft/sec)
v	0.50000 (ft/sec)
w	0.50000 (ft/sec)
p	0.57000 (rad/sec)
q	0.57000 (rad/sec)
r	0.57000 (rad/sec)
Φ	0.00873 (rad)
Θ	0.00873 (rad)
Ψ	0.05236 (rad)

matched those in the referenced books. The covariance matrices for the simulation runs are shown in Figures B.3 - B.11. All converge to small values as expected.

The gains for the bank of Kalman filters are shown in Figures B.12 - B.20. The values for the gains converges to zero or close to it. This was expected since the process noise numbers were chosen to be small in order to get small gains. This way the filters would trust the model derived for their appropriate failure hypothesis.

The measurement state estimates or filtered measurements were plotted against the noisy measurements in Figures B.21 - B.29 for the case of no failure. On all the figures, the estimates from the Kalman filters follow the general curve or orientation of the "true" measurements which was expected from the theory. The "notch" seen on some of the figures at the 100^{th} observation is caused by the step input function that comes down at this point. The residuals or innovation processes are also shown in Figures B.30 - B.38 and are used in the calculation of the probability of detecting error in equation 2.36.

C. DETECTION AND ISOLATION RESULTS

Menke and Maybeck in [Ref. 14, p. 3136] decided to set the detection threshold for the different failure scenarios to a repetitive probability of 0.9 or higher for ten consecutive observations in their application of FDI. This definition sets a minimum that was well exceeded when testing this algorithm for the hard failure scheme. A set of 13 hard failures were tested and some of the results are plotted in Figures B.39 - B.51. All of the data was not shown due to the large volume generated. Every run for each failure generates 13 graphs of probability computation and only one of those should show values close to one. The algorithm was quite good at detecting abrupt failures in the system since all the probability plots show values reaching one fairly soon and staying at that level until the end of all the observations. The plots shown in Figures B.39 - B.51 represent only the particular output from the filter of interest for the particular failure that they were supposed to be tuned to detect. The other plots from the filters tuned for other failures gave values of probability in the vicinity of zero.

The test for no failure was also successful and the filter tuned for no failure resulted in probability computation of one fairly early in the simulation as seen in Figure B.52.

The results obtained during the soft or incipient failure runs were incoclusive and the algorithm used in this detection scheme should not be considered yet as operating satisfactorily. The probability computations were not consistent from one run to another and several false alarms were generated, i.e., probabilities of 0.9 were reached on filter channels not tuned to detect the error being tested. The plots in Figures B.53 - B.55 were generated for the soft failure simulation on the elevator, Figure B.53 shows that the failure was not detected. Figure B.54 and Figure B.55 show false alarm occurrences on the no failure filter and on the pitch rate filter.

The same types of results were obtained when soft failure was tested for the airspeed velocity sensor as seen in Figures B.56 - B.58.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The results obtained were very encouraging for the detection of hard failures. The filters were well tuned to detect failures under those particular situations. The problematic results obtained when simulating soft failures were anticipated. These difficulties have also been encountered by other researchers working in the field. More work will be required to render this feature operational for the present FDI algorithm.

B. RECOMMENDATIONS

The Kalman gains should be precalculated and stored prior to being used since they do not vary with measured data.

The work done in this thesis did not address the task of reconfiguration. If the FDI algorithm is to be implemented on the Bluebird this part will need to be addressed.

The whiteness tests described in chapter III. should also be conducted once the algorithm is installed on the test bed since the noise experienced in the working model might be quite different from that which what was generated numerically in the MATLAB code.

A dual failure detection scheme could be implemented in future development in order to detect several failures occurring at the same time.

The implementation methods from [Ref. 19, p.192–260] should be investigated in order to improve the robustness of the algorithm when small numbers are generated by the filter, thus eliminating ill-conditioned situations inside the algorithm.

The algorithm implemented was tested on a model derived from the linearization of a nonlinear system around one nominal flight condition. More tests should be conducted using models linearized around several flight conditions.

APPENDIX A: USER'S MANUAL

This appendix describes in detail the steps required to open and run the software which generates the various probability charts to verify if failures have been detected.

The user must be familiar with the basic operation of the MATLAB software package. Additionally, it is assumed that the user is already logged on to a MATLAB capable Unix work station. Before entering the MATLAB environment, one must change the working directory to the one which contains all the ".m" files, in this case, "marioth". The command is:

```
cd marioth
```

If the user is remotely logged onto a work station, he must set the DISPLAY environment variable appropriately in order to display graphics. The command which sets this variable to intrepid, a Sparc 2 work station in the Avionics Lab is:

```
setenv DISPLAY intrepid.aa.nps.navy.mil:0
```

Now it is time to begin the MATLAB session by typing:

```
matlab
```

The next command to run the main program is:

```
fdi
```

A series of windows will appear for the user to select the type of failure to be simulated. The user is required to position the arrow of the mouse on the pad of his choice and click the left button of the mouse to select the item. The first window to appear is for the selection of the type of failure to be simulated. The choices are:

- no failure
- Hard or abrupt failure
- Soft or incipient failure

The second window to appear, if a selection other than "no failure" has been chosen from the previous window, is for the selection of a failure source. The choices are:

- Actuator Failure
- Sensor Failure

The last window to appear will be to either select an actuator type or a sensor type, depending on the choice made from the previous window. If the selection was for an actuator failure, the choices will be:

- Elevator
- Aileron
- Rudder
- Thrust Actuator

If the selection was for a sensor failure, the choices will be:

- u
- v
- w
- p
- q
- r
- phi
- theta
- psi

Before running the program, the user might want to change some parameters inside the program in order to save time or to look at more graphics.

The structure of the FDI program will now be presented in order to give the user insight into how the algorithm works prior to changing the values inside the different files. The structure of the algorithm is made of several MATLAB files and functions.

The main file is called `fdi.m` and the majority of the calls to the different functions are initiated within this file.

The first file to be accessed is `ABCU_load.m`. The values for the plant matrices A,B,C,u are stored in this file along with the initial value for $\hat{\mathbf{x}}(0/-1)$, $P(k/k-1)$ and the values for the standard deviation of the process noise and measurement noise. If many runs were to be conducted for different linearized models around operating points, the plant matrices would be changed in this file. The values for the time increment `dt` as well as the `Stop time` can be changed appropriately to vary the total number of observations.

The second file accessed by `fdi.m` is the function `faultchoice.m`. This function brings up the various pop-up menus and allows the user to simulate a certain fault in the plant by zeroing the column or changing the noise on certain matrices. No changes are required from the user to this file unless the structure of the program needs to be changed.

The third step combines the calls to the function `kalmn_gain.m` in order to calculate the gain for each of the Kalman filters associated with a particular fault. Since the gains are independent of the measurement data, once the program has been run once and the gains are stored in memory, the flag "`need_to_calculate_gains`" can be set to "**0**" and some time will be saved for the future runs since the gains will not be recalculated every time.

The fourth step comprises the calls to the function **kalman_i.m**. This function returns values for the residuals and the state estimates. Modifications to this file can be made under the heading "measurements", where a step input of one second is forced into the plant. The size of the step input is set to five degrees for the control surfaces and to 50% for the thrust actuator. A call to the function **kalman_plots** is located at the bottom of the file and can be enabled by removing the "%" at the beginning of the line. The **kalman_plots** function generates plots for the process noise, measurement noise, Kalman gains, error covariance matrices P(k/k), estimates vs. noisy measurements, and plots for the residuals.

The "last call" of the main file is to the function **pk.m**. This function calculates the hypothesis conditional probability pk as the probability that a measured parameter assumes the value at k conditioned on the observed measurement history up to time k. It also generates the plots to show the probabilities of failure associated with each sensor and actuator. No changes are required from the user to this file unless the structure of the program needs to be changed.

APPENDIX B: GRAPHICAL RESULTS

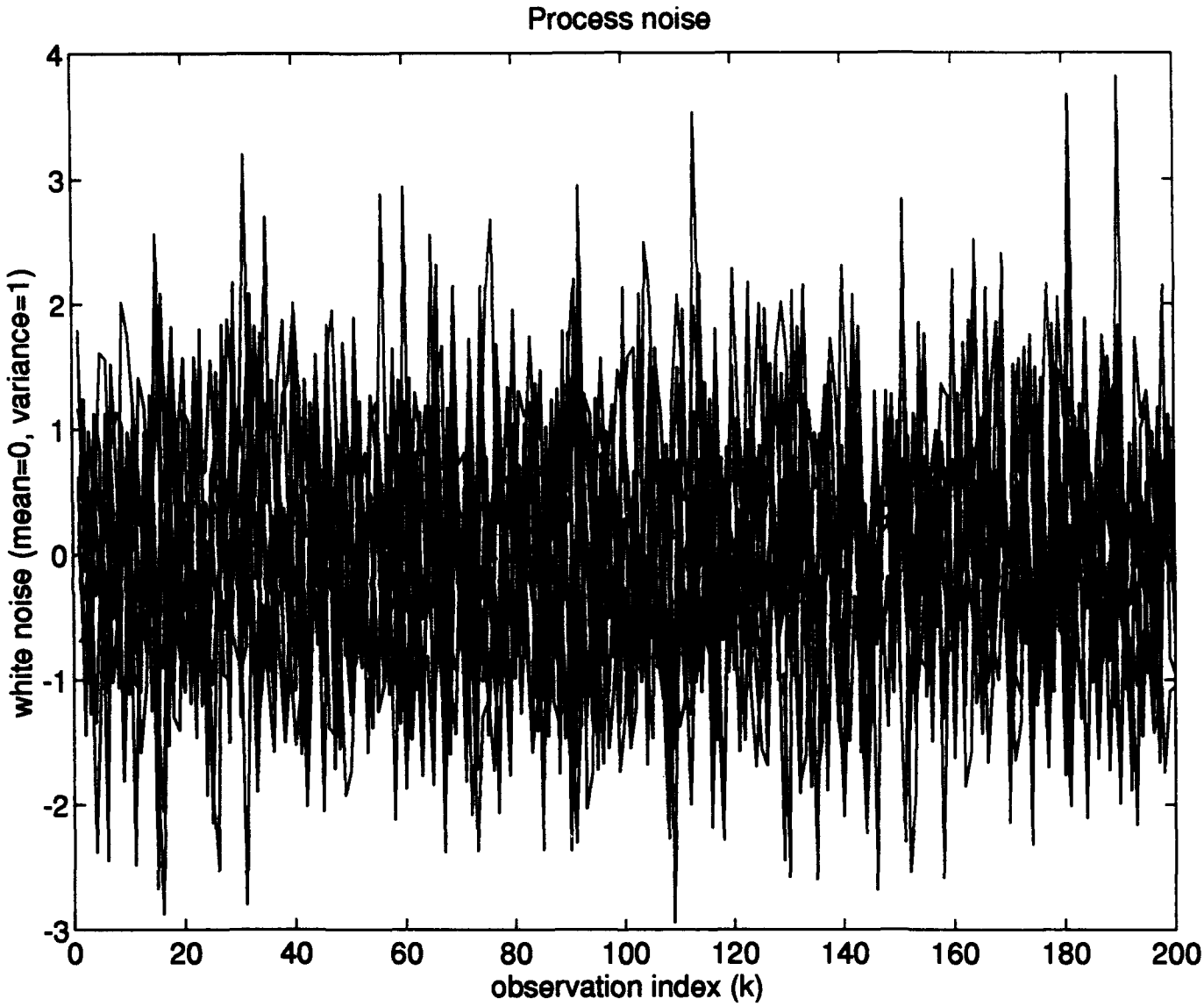


Figure B.1: Process Noise w

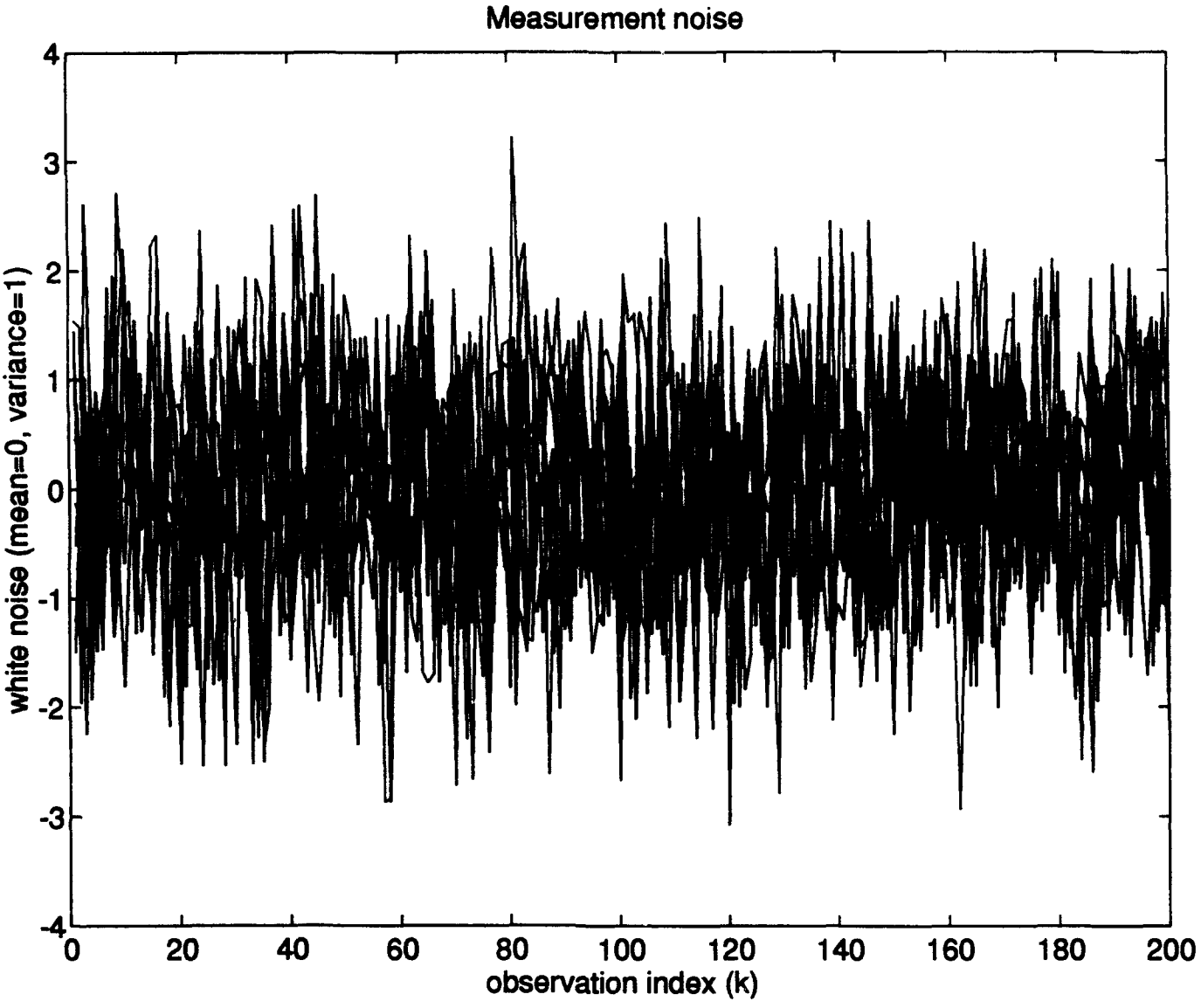


Figure B.2: Measurement Noise v

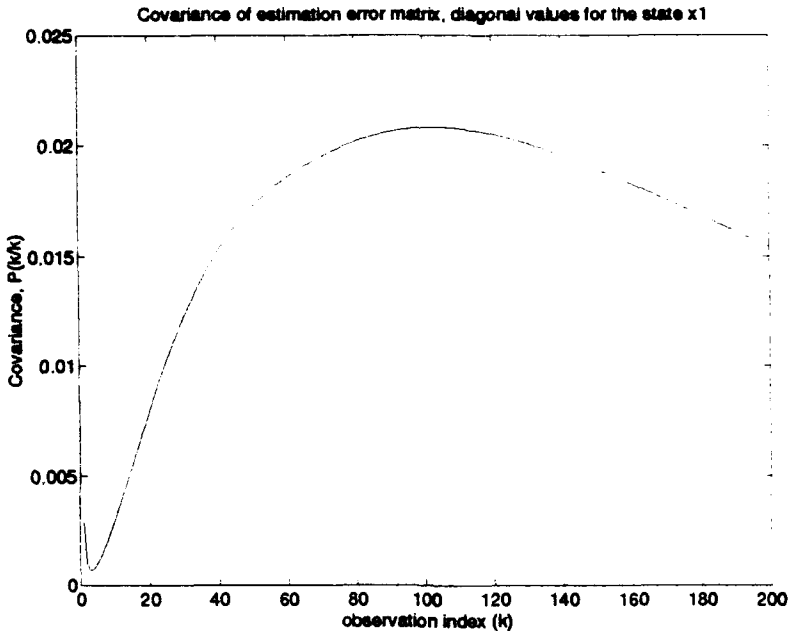


Figure B.3: Transpose Values of the Covariance Matrix P for State u

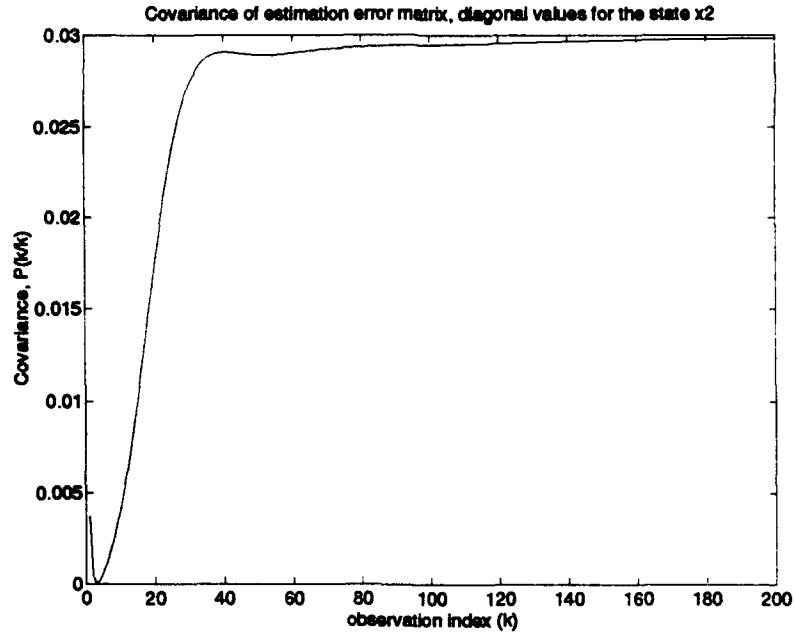


Figure B.4: Transpose Values of the Covariance Matrix P for State v

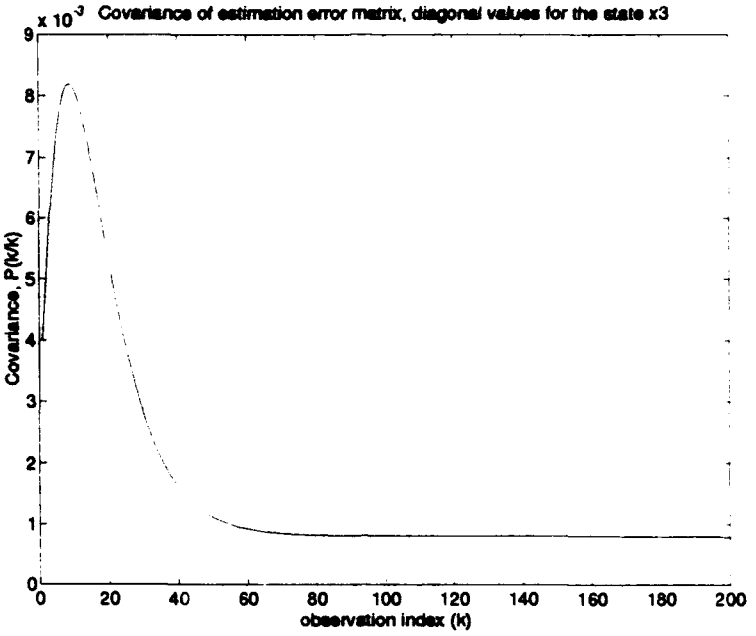


Figure B.5: Transpose Values of the Covariance Matrix P for State w

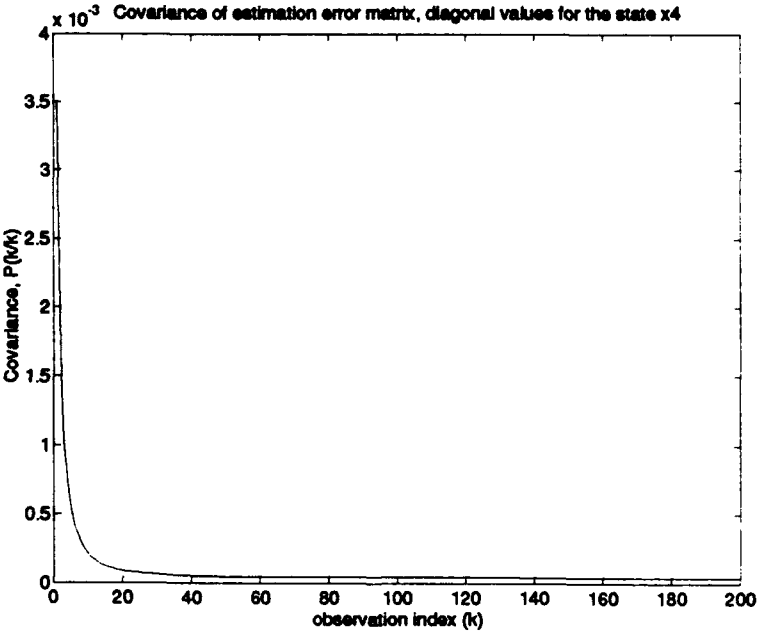


Figure B.6: Transpose Values of the Covariance Matrix P for State p

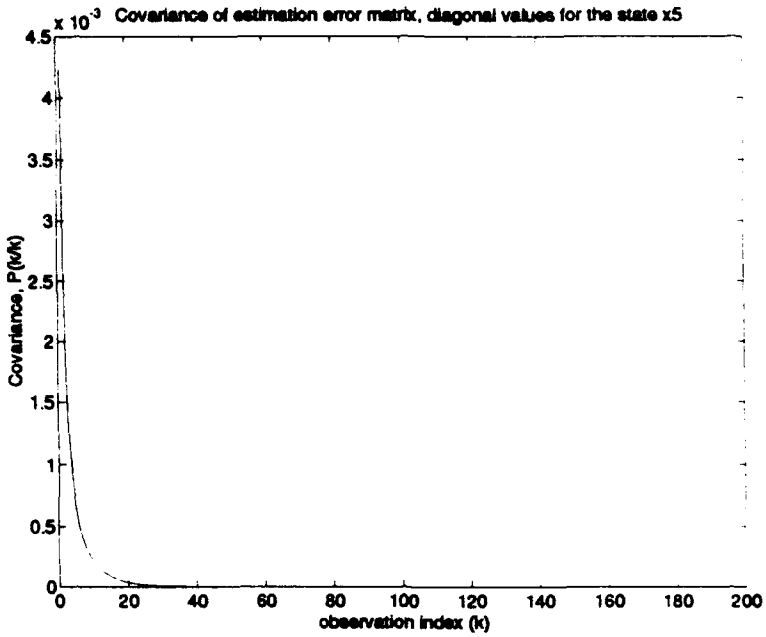


Figure B.7: Transpose Values of the Covariance Matrix P for State q

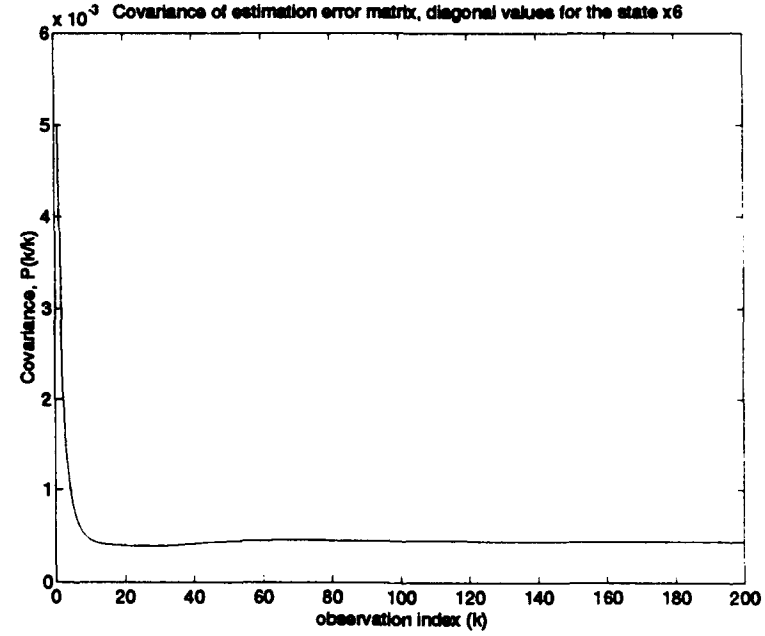


Figure B.8: Transpose Values of the Covariance Matrix P for State r

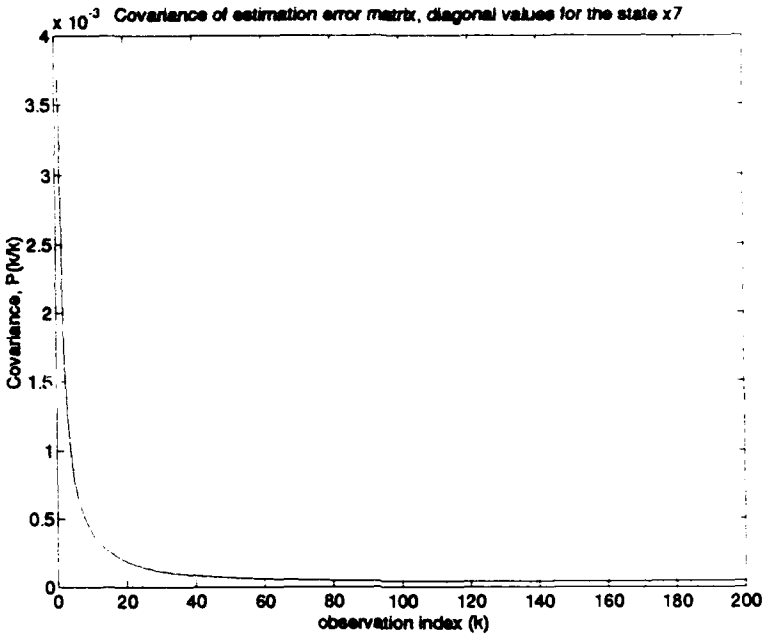


Figure B.9: Transpose Values of the Covariance Matrix P for State phi

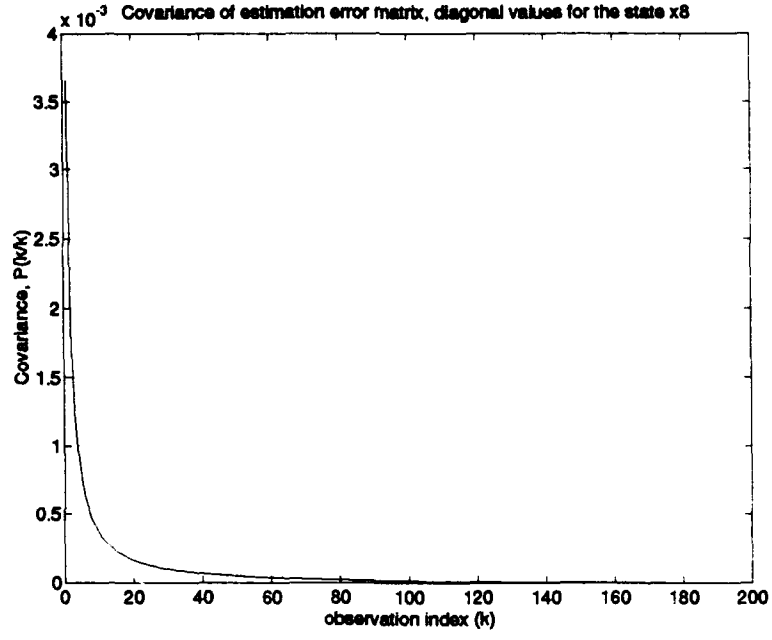


Figure B.10: Transpose Values of the Covariance Matrix P for State theta

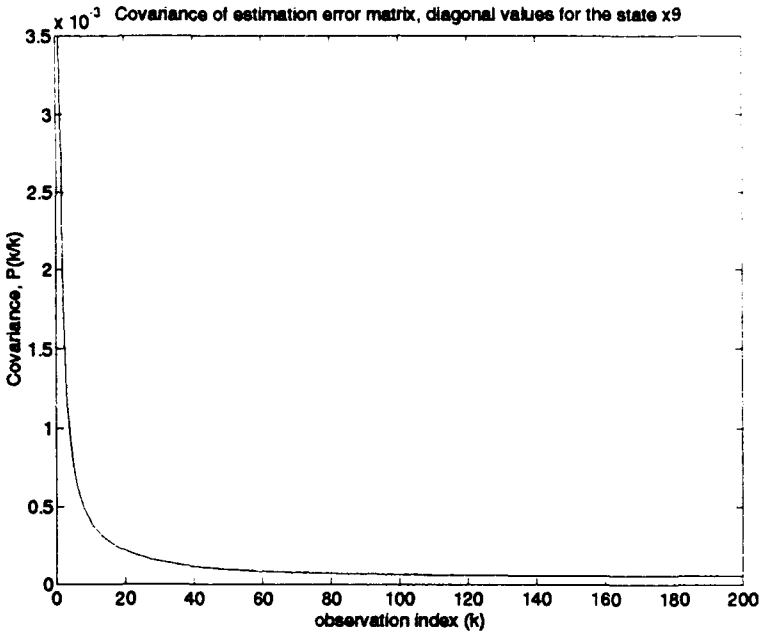


Figure B.11: Transpose Values of the Covariance Matrix P for State psi

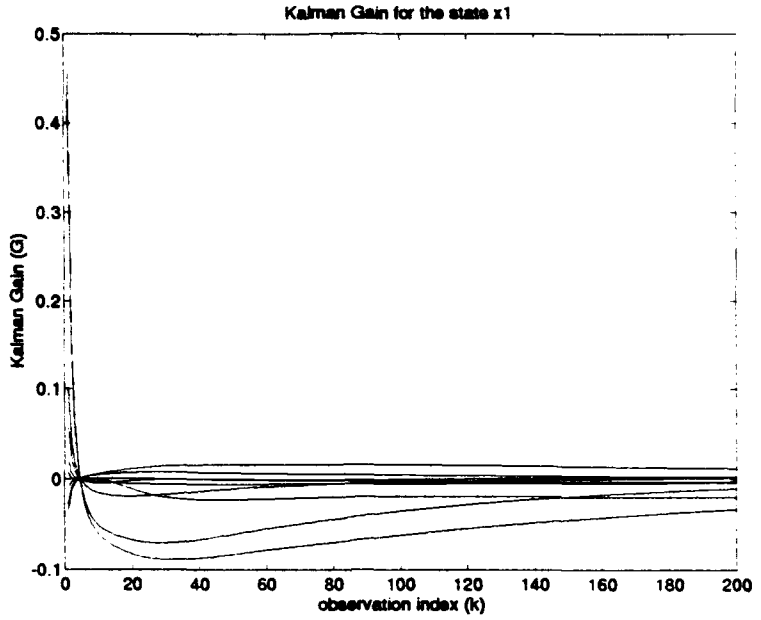


Figure B.12: Kalman Gains for the State u

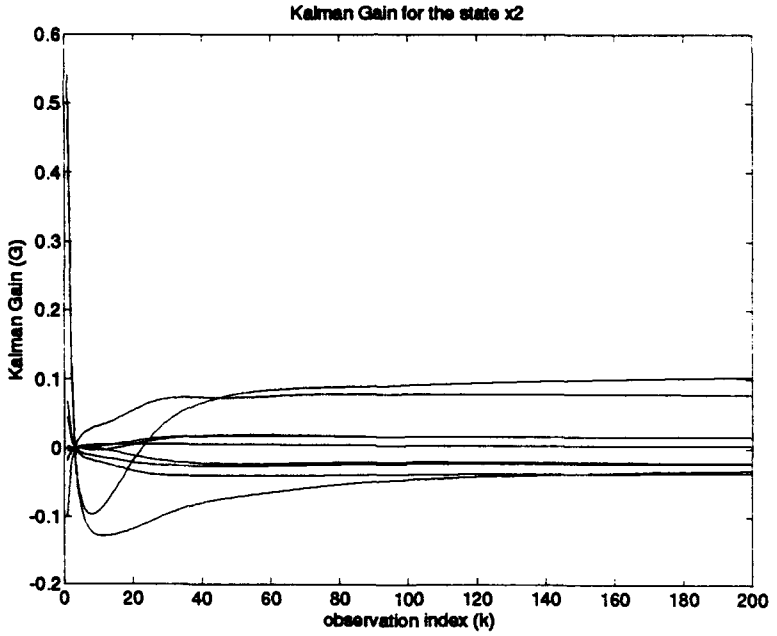


Figure B.13: Kalman Gains for the State v

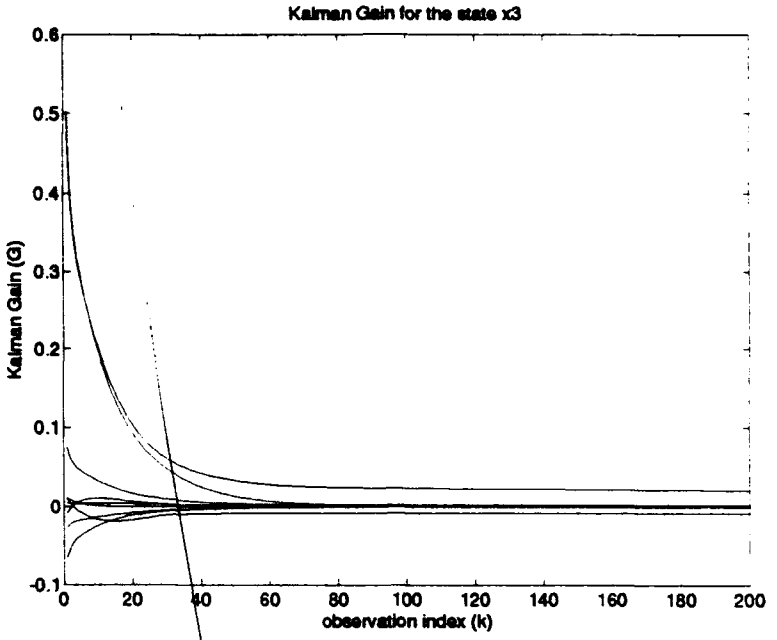


Figure B.14: Kalman Gains for the State w

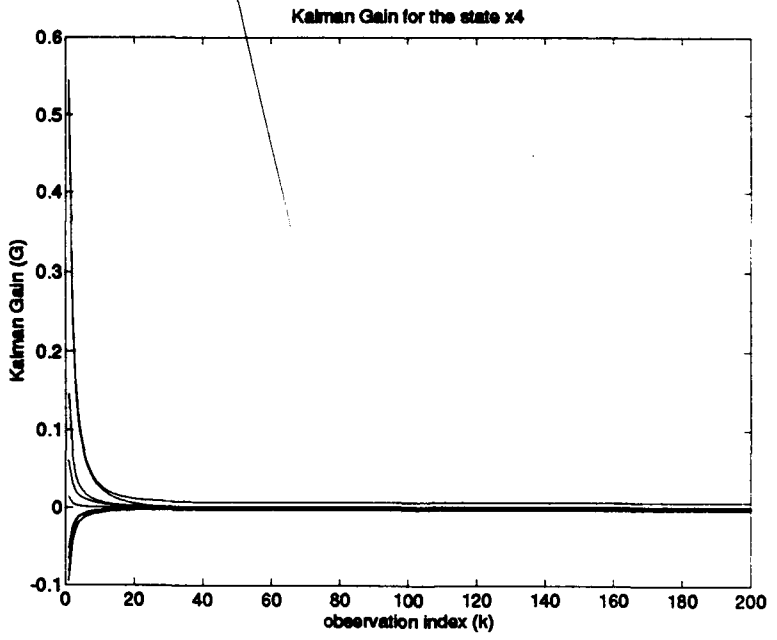


Figure B.15: Kalman Gains for the State p

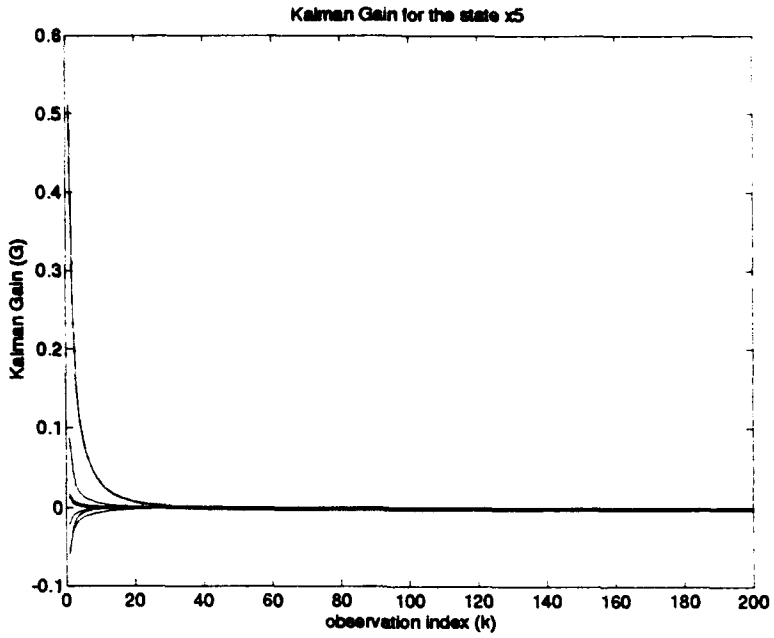


Figure B.16: Kalman Gains for the State q

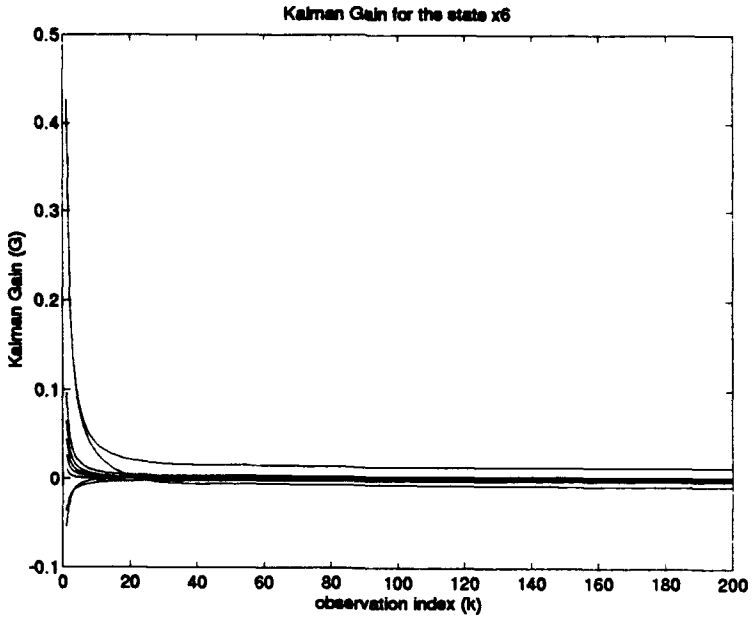


Figure B.17: Kalman Gains for the State r

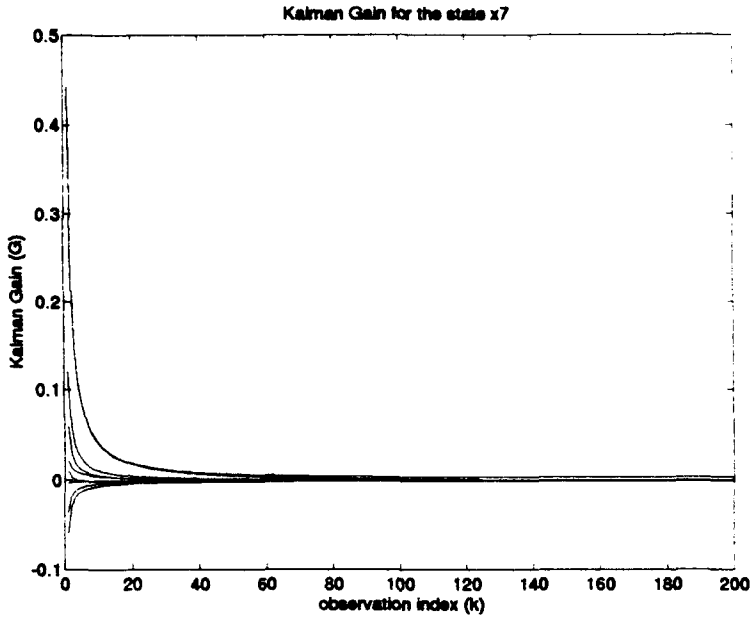


Figure B.18: Kalman Gains for the State phi

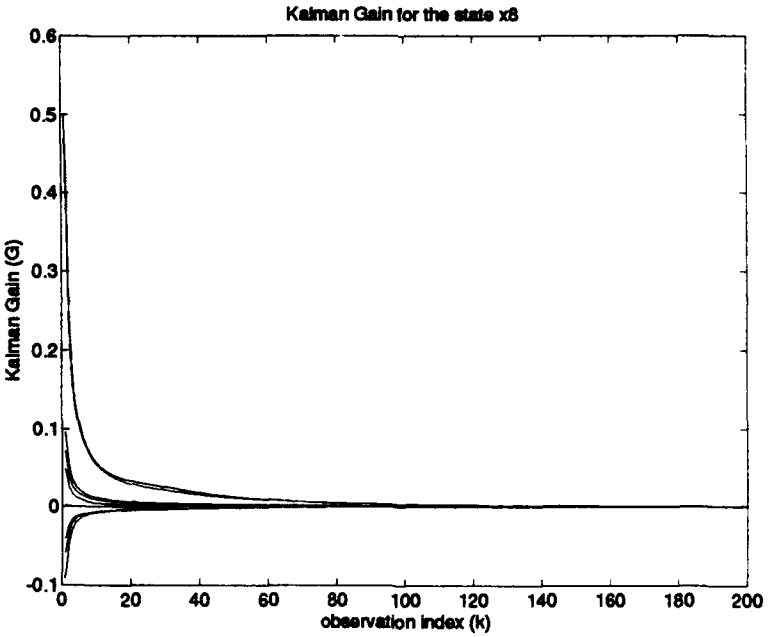


Figure B.19: Kalman Gains for the State theta

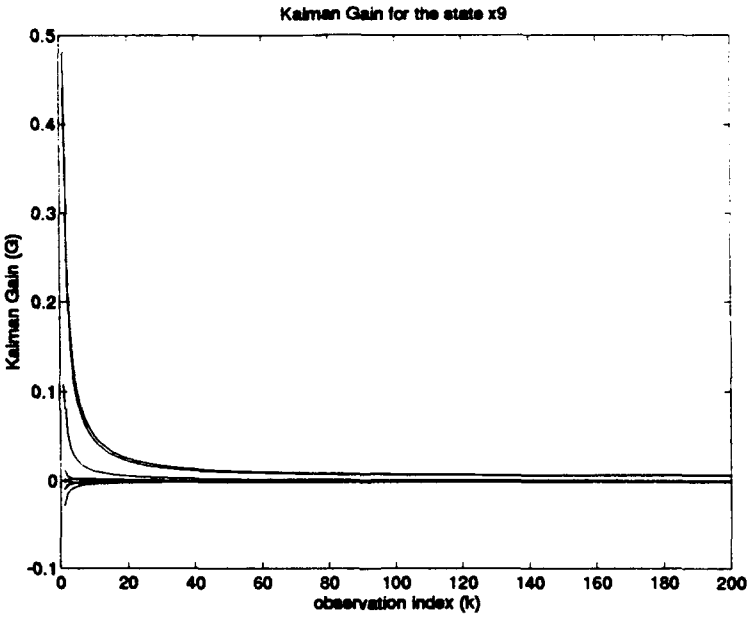


Figure B.20: Kalman Gains for the State psi

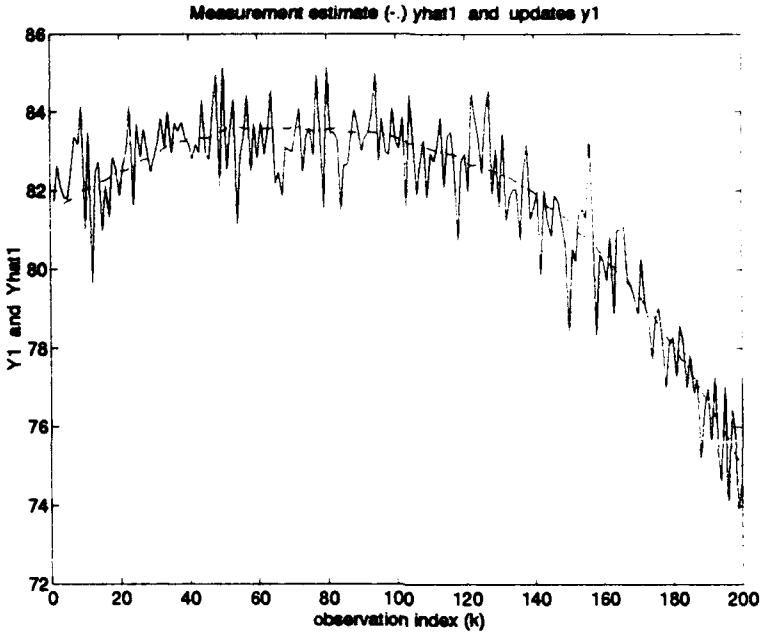


Figure B.21: Measurement and State Estimate u

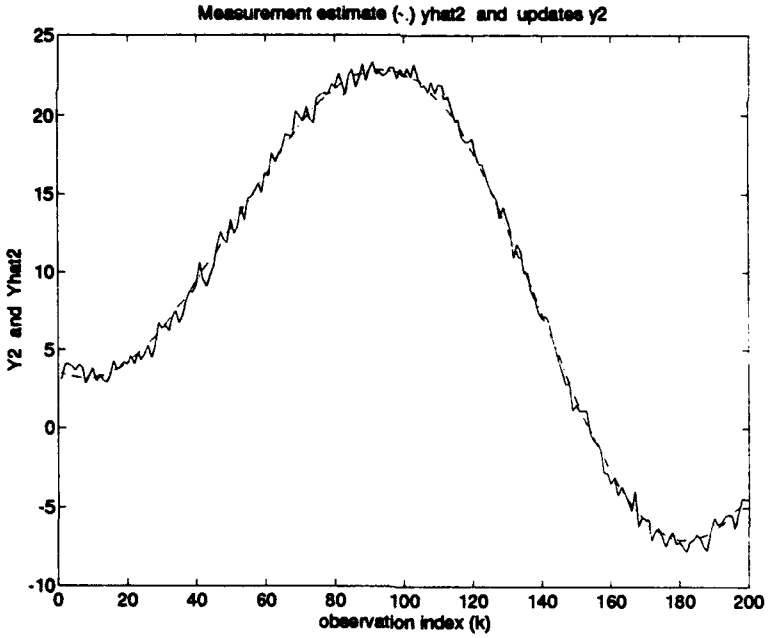


Figure B.22: Measurement and State Estimate v

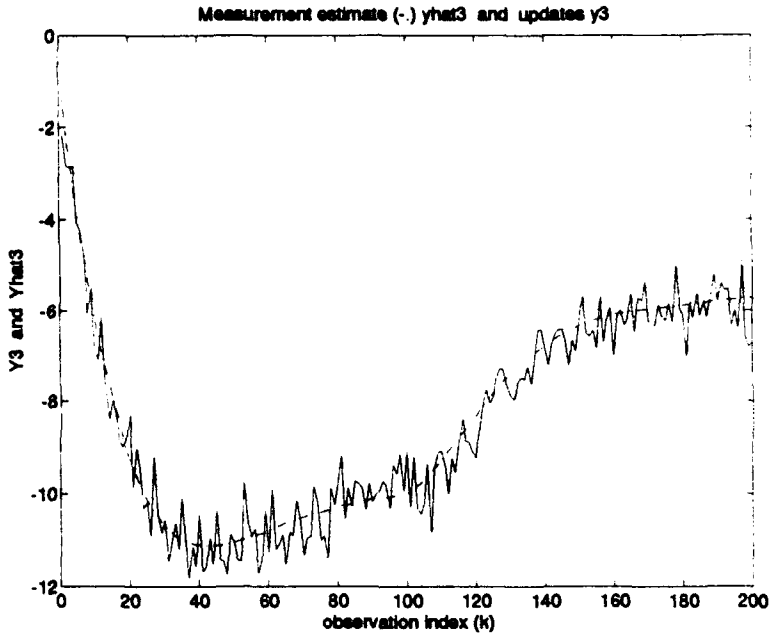


Figure B.23: Measurement and State Estimate w

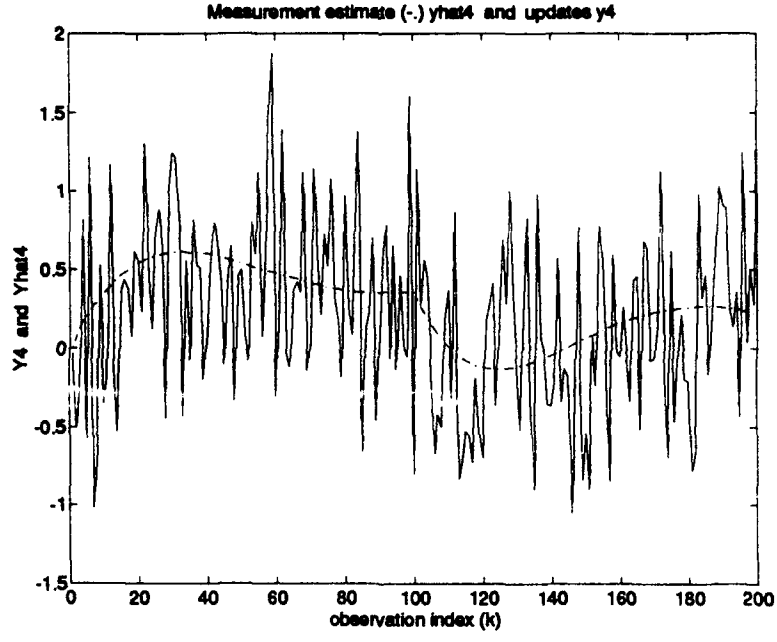


Figure B.24: Measurement and State Estimate p

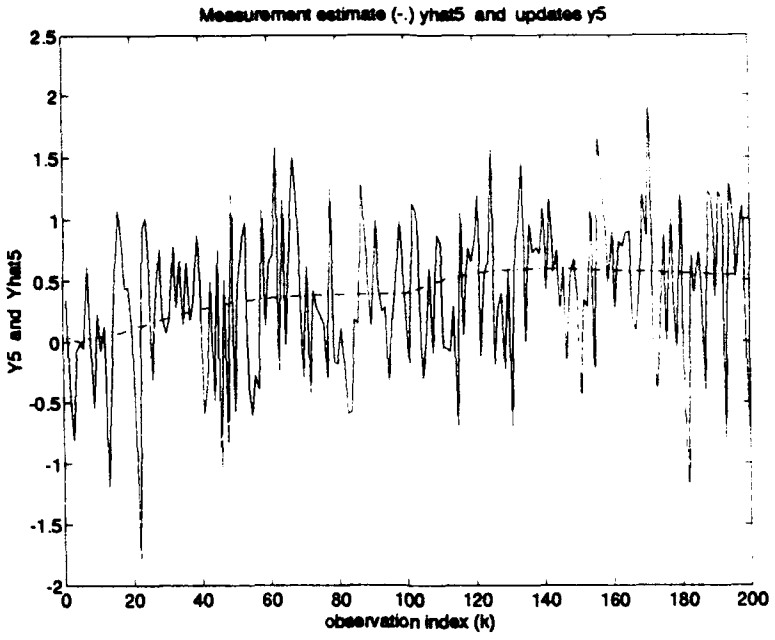


Figure B.25: Measurement and State Estimate q

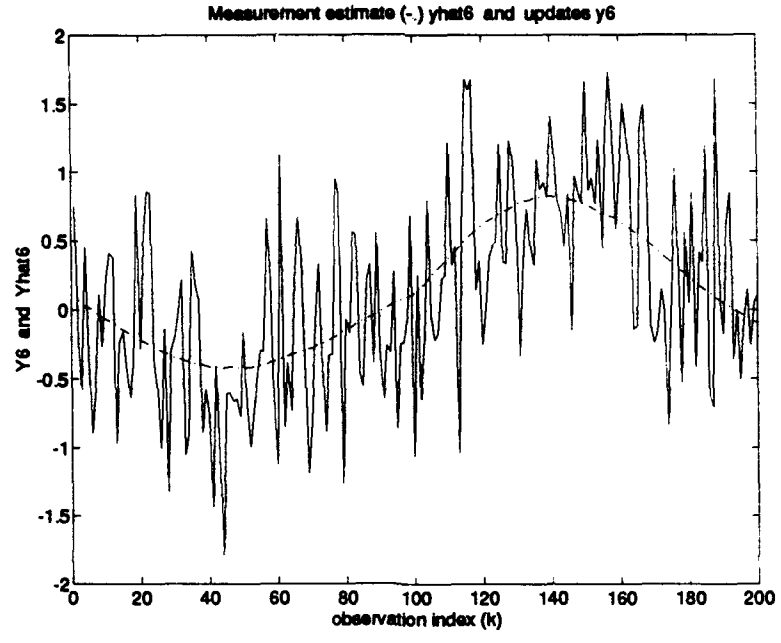


Figure B.26: Measurement and State Estimate r

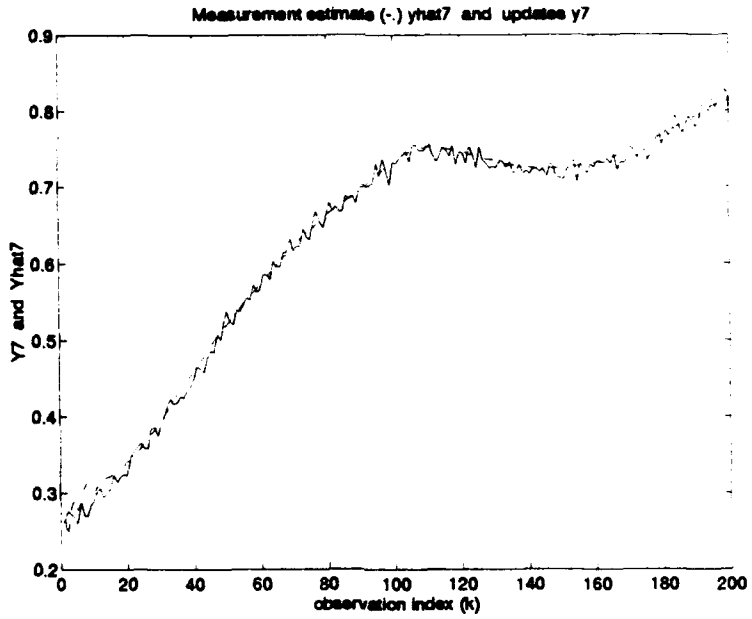


Figure B.27: Measurement and State Estimate phi

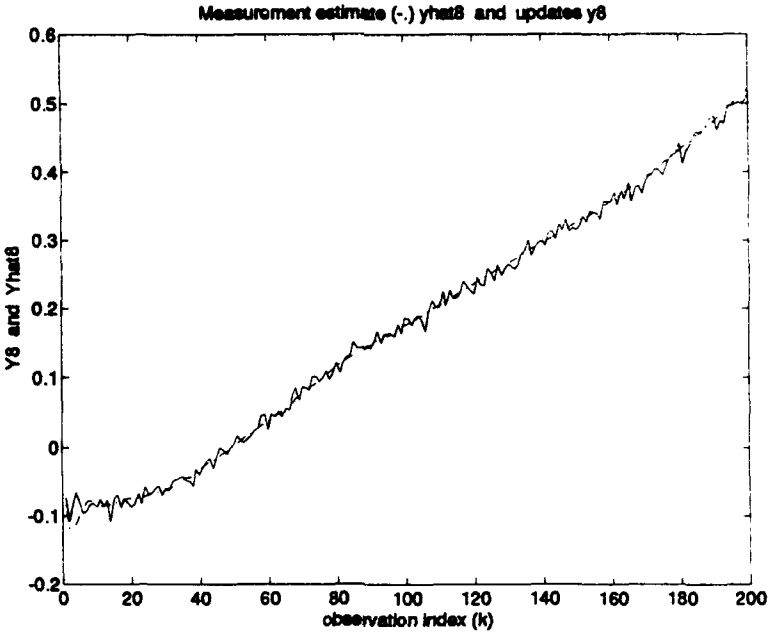


Figure B.28: Measurement and State Estimate theta

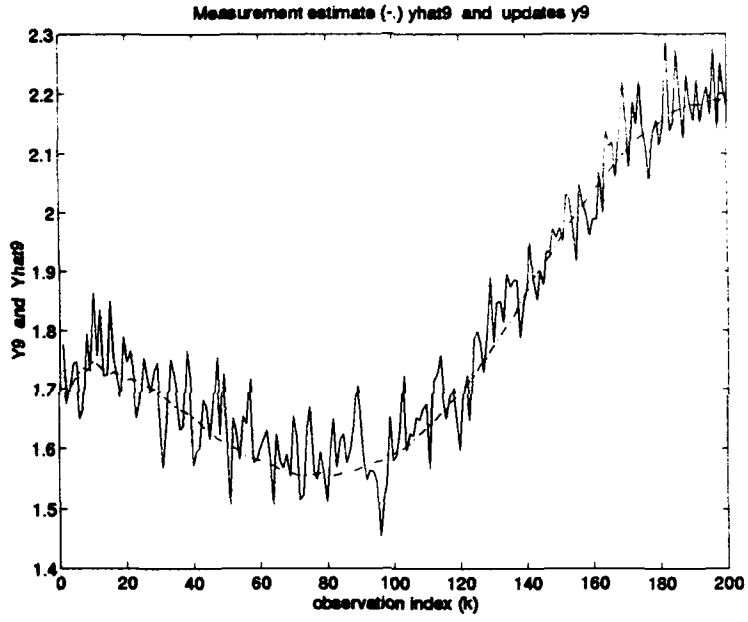


Figure B.29: Measurement and State Estimate psi

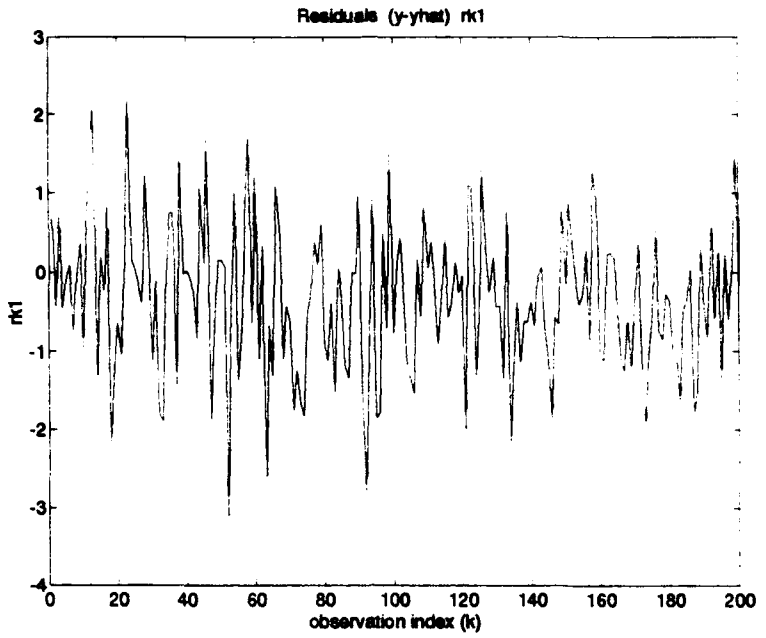


Figure B.30: Innovation Process for the State u

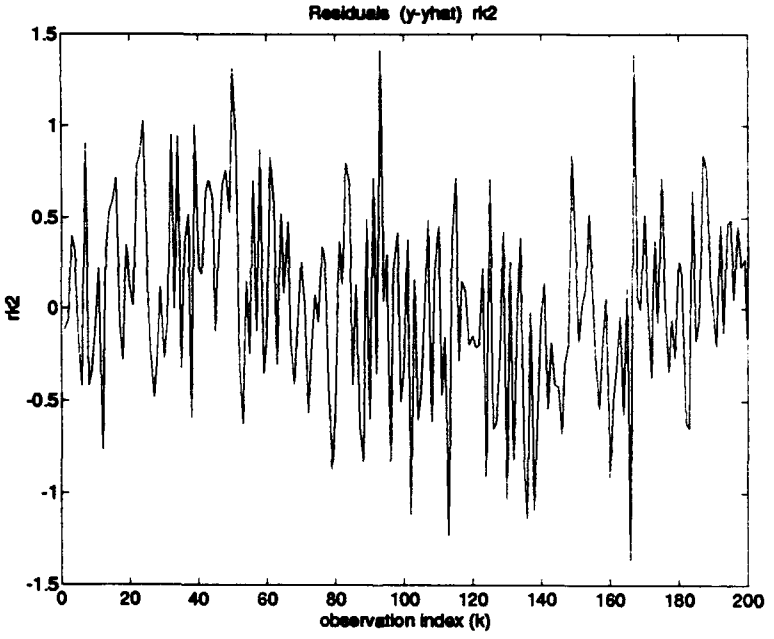


Figure B.31: Innovation Process for the State v

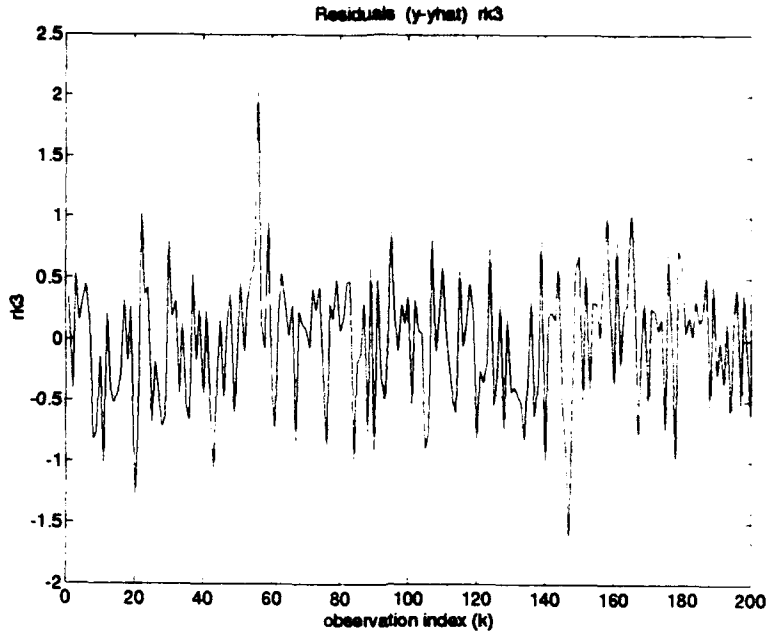


Figure B.32: Innovation Process for the State w

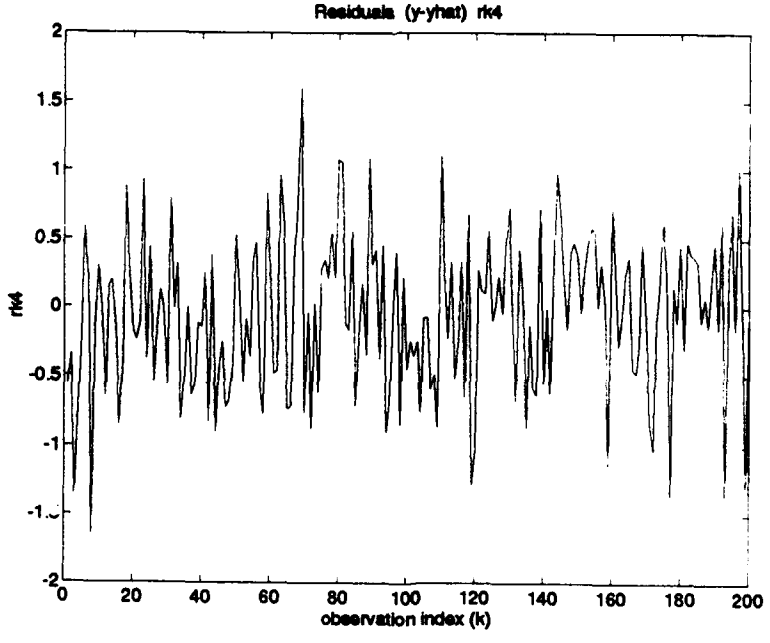


Figure B.33: Innovation Process for the State p

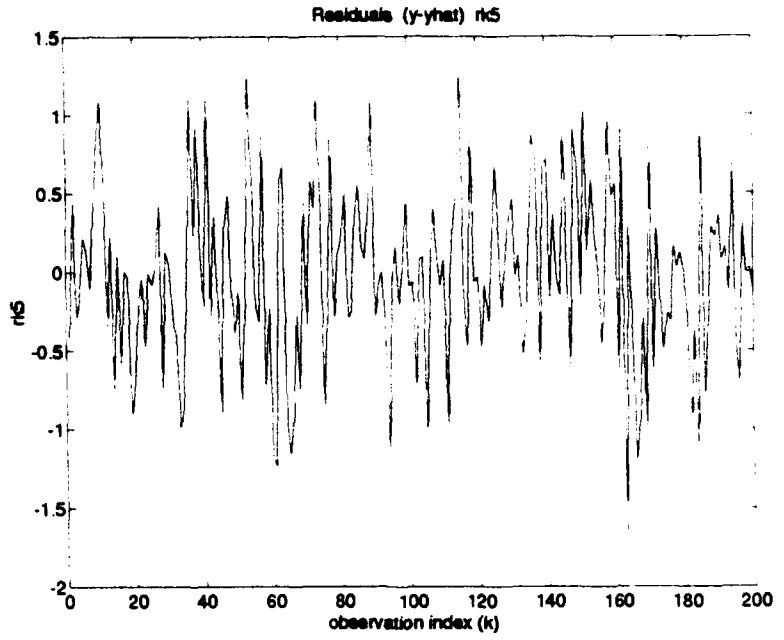


Figure B.34: Innovation Process for the State q

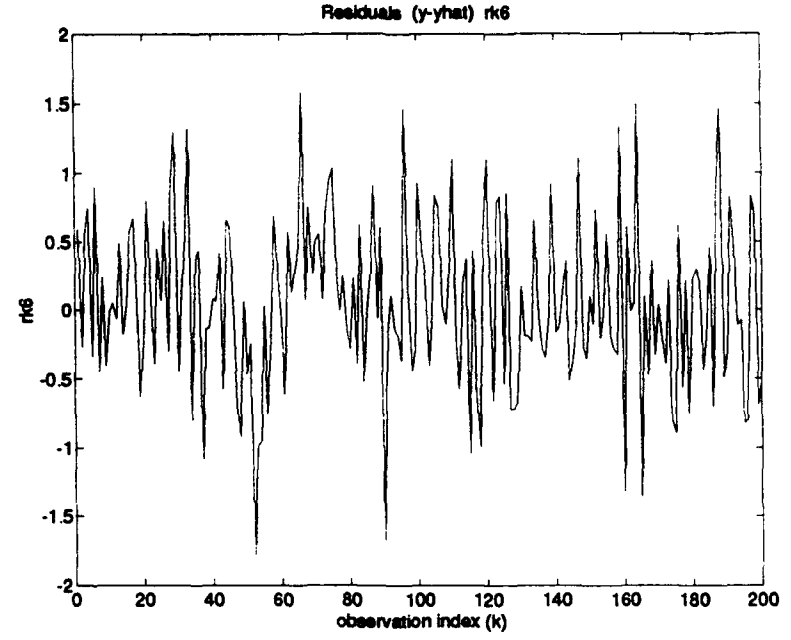


Figure B.35: Innovation Process for the State r

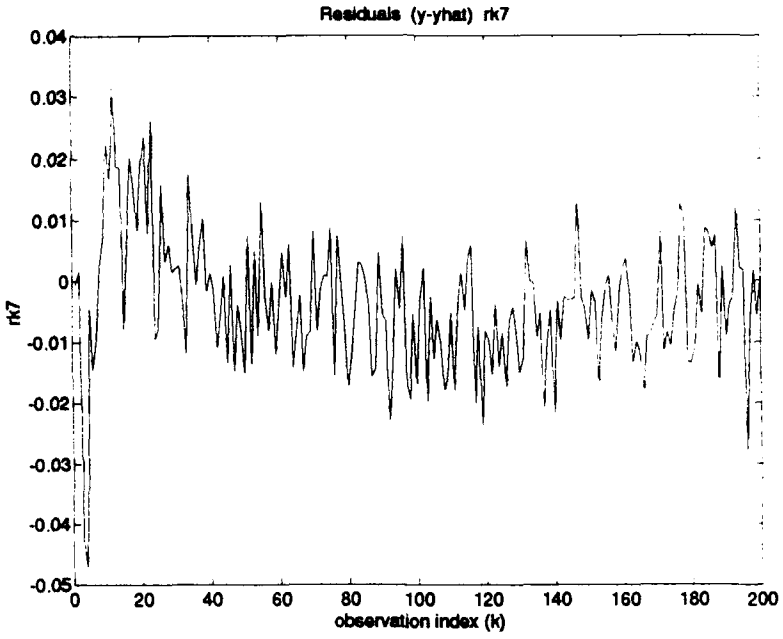


Figure B.36: Innovation Process for the State phi

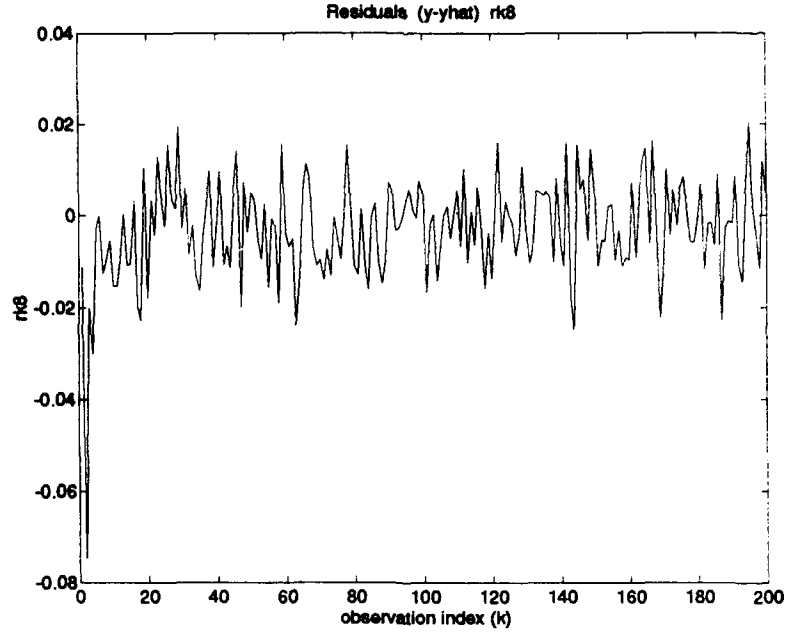


Figure B.37: Innovation Process for the State theta

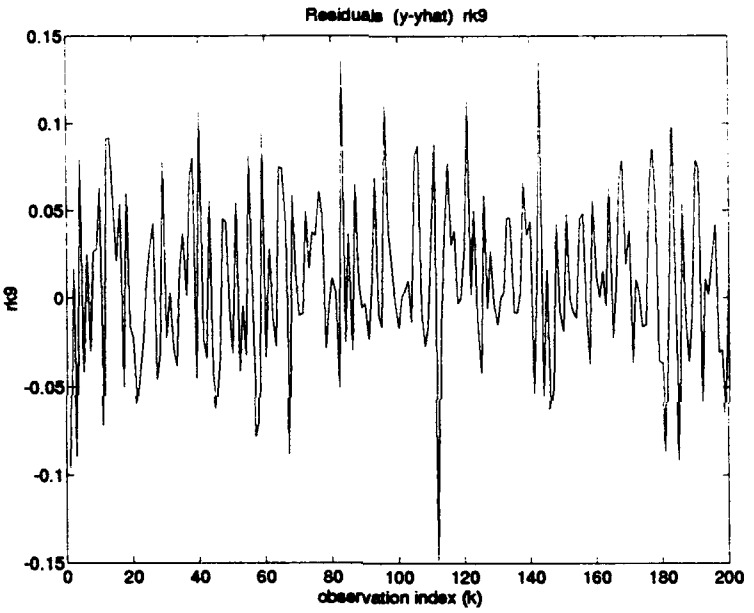


Figure B.38: Innovation Process for the State psi

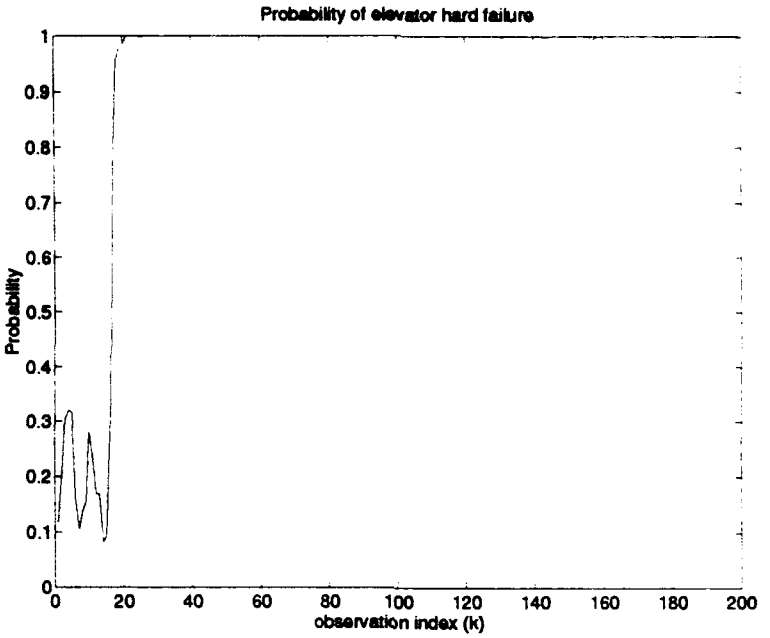


Figure B.39: Probability of Elevator Hard Failure

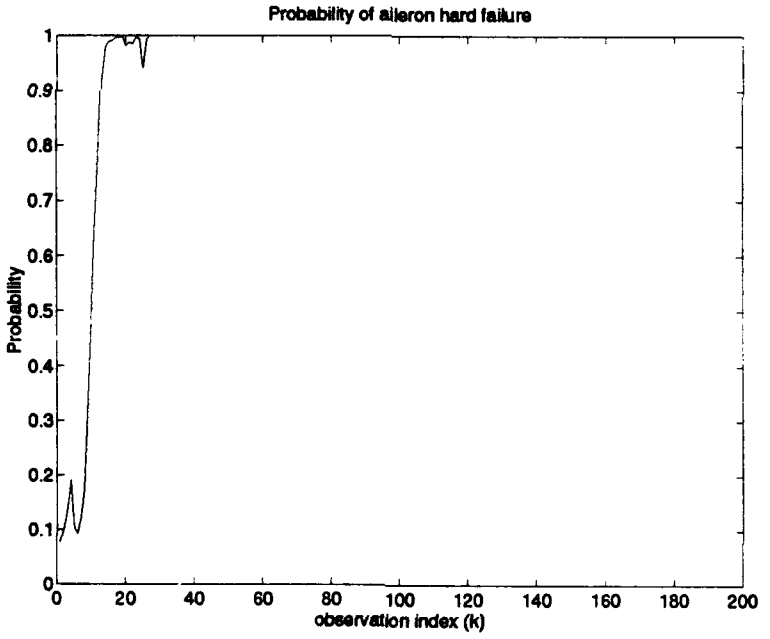


Figure B.40: Probability of Aileron Hard Failure

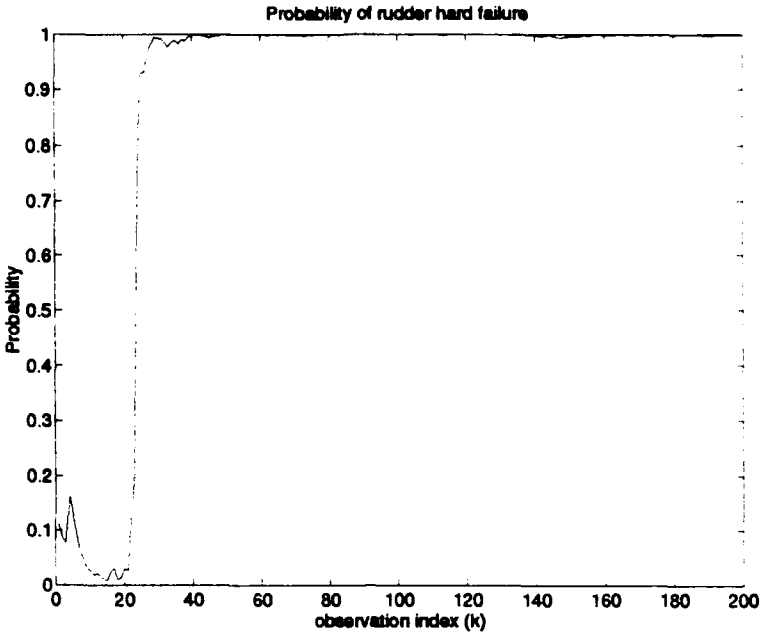


Figure B.41: Probability of Rudder Hard Failure

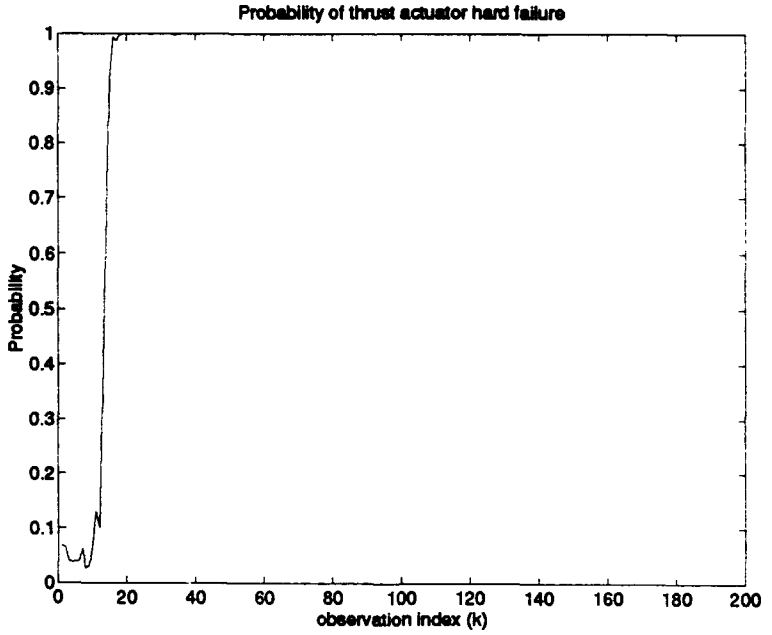


Figure B.42: Probability of Thrust Actuator Hard Failure

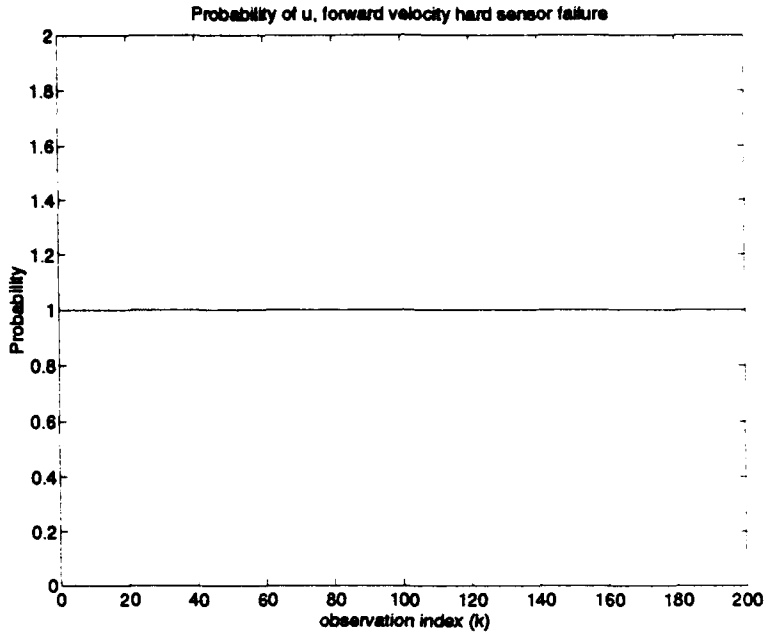


Figure B.43: Probability of u Sensor Hard Failure

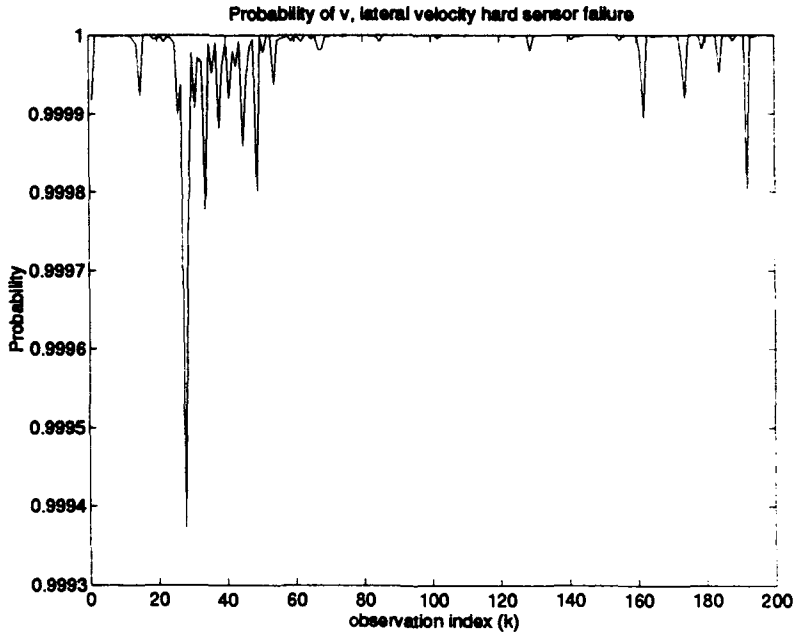


Figure B.44: Probability of v Sensor Hard Failure

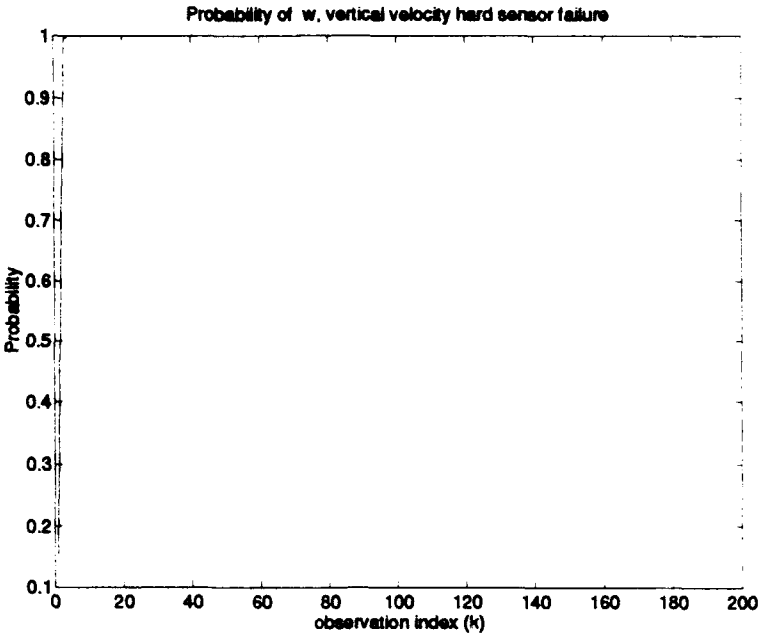


Figure B.45: Probability of w Sensor Hard Failure

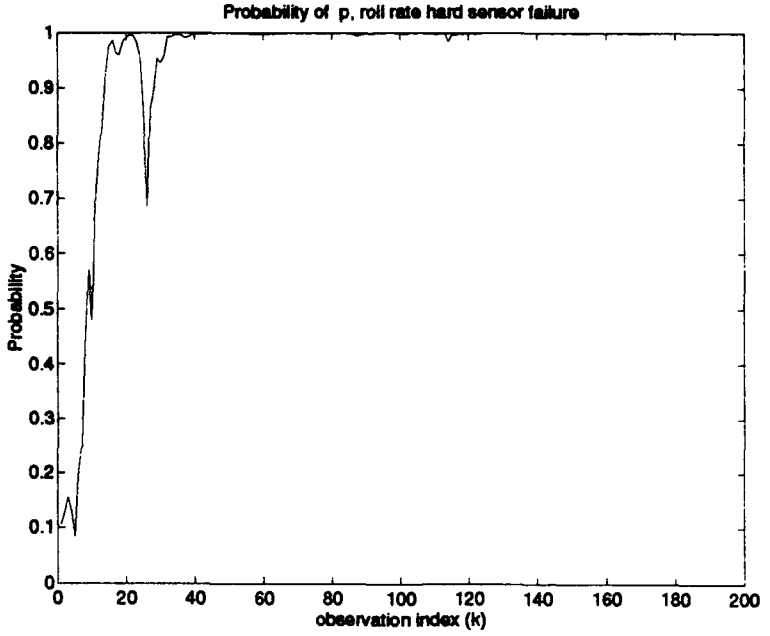


Figure B.46: Probability of Roll Rate Sensor Hard Failure

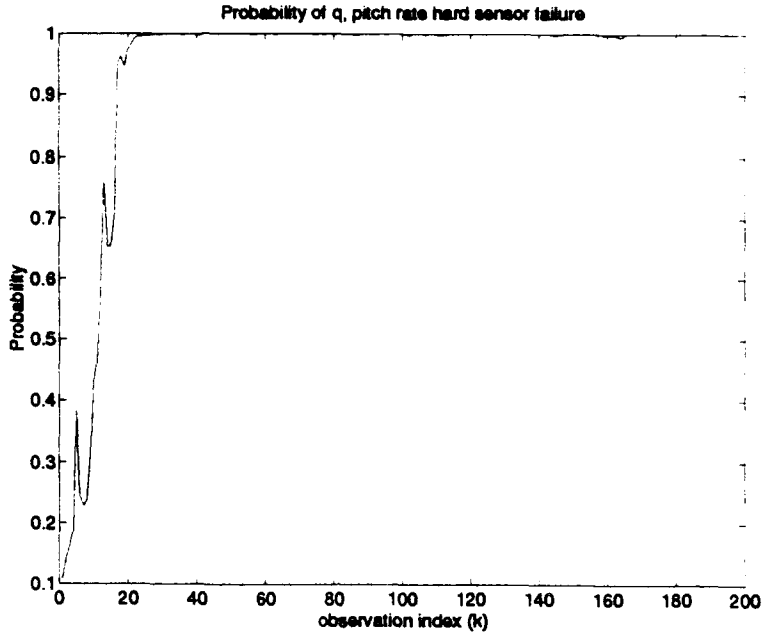


Figure B.47: Probability of Pitch Rate Sensor Hard Failure

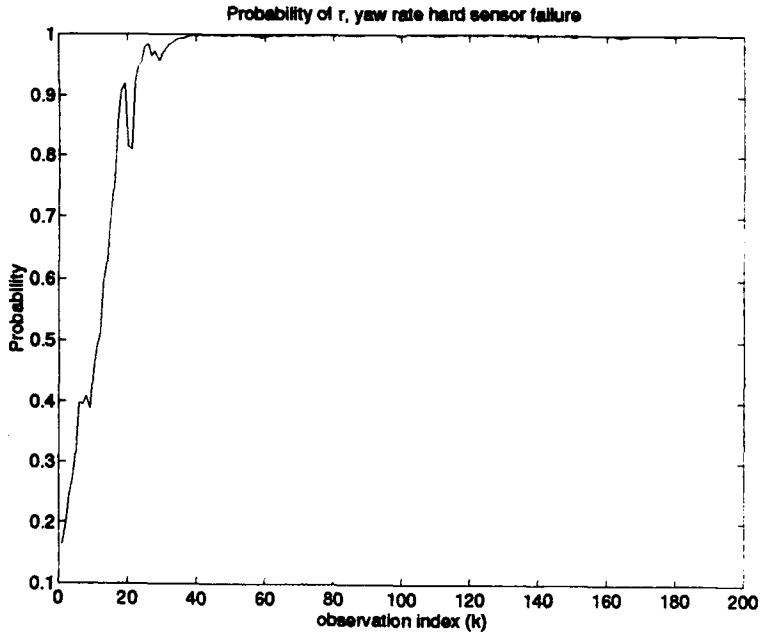


Figure B.48: Probability of Yaw Rate Sensor Hard Failure

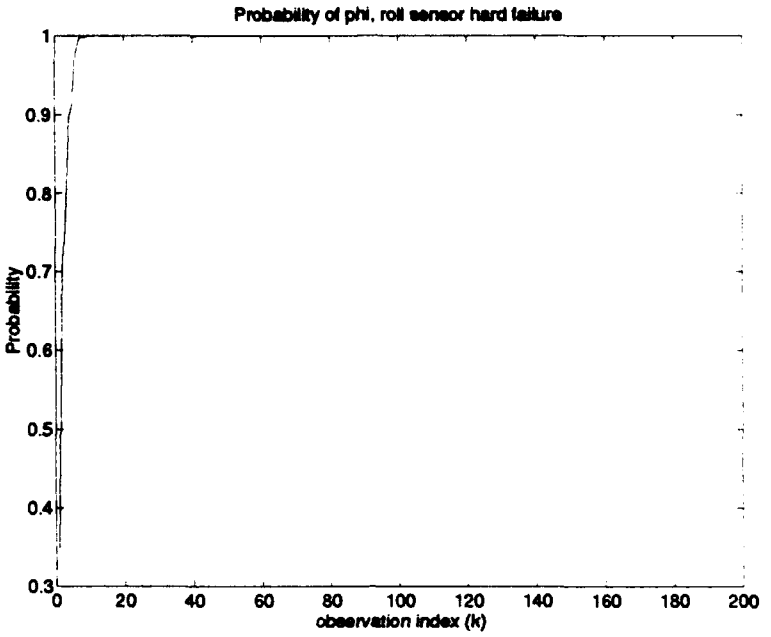


Figure B.49: Probability of Roll Angle Sensor Hard Failure

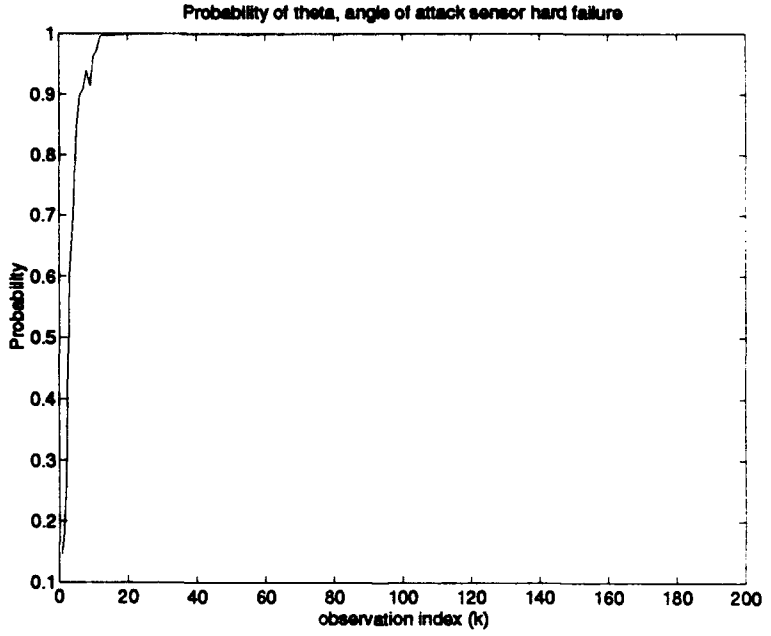


Figure B.50: Probability of Angle of Attack Sensor Hard Failure

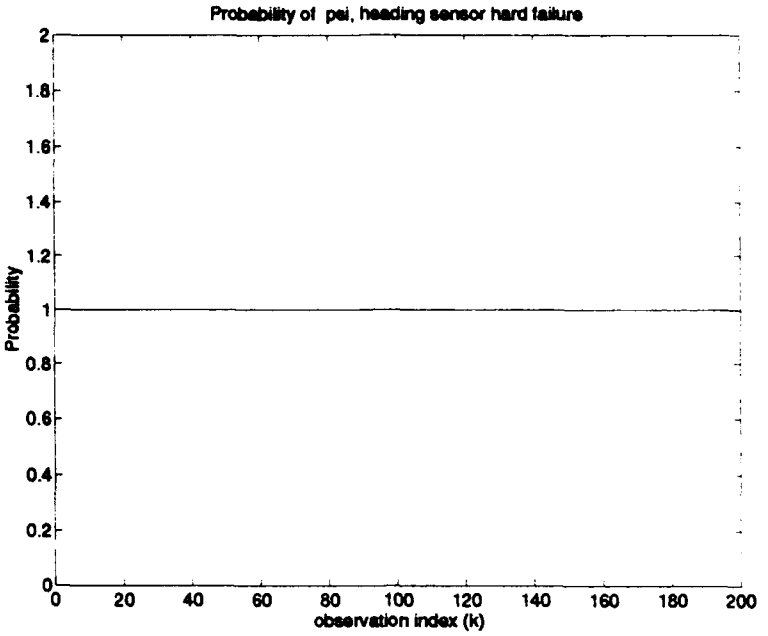


Figure B.51: Probability of Heading Angle Hard Failure

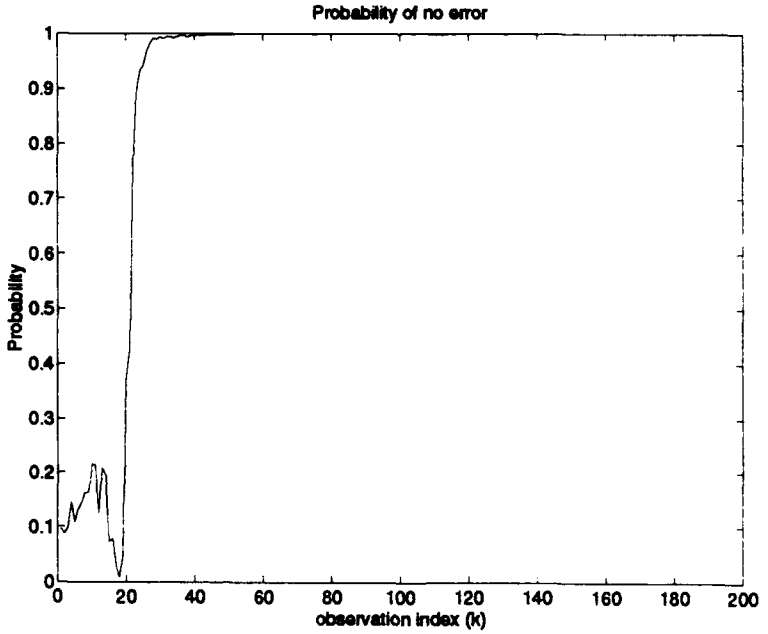


Figure B.52: Probability of No Failure

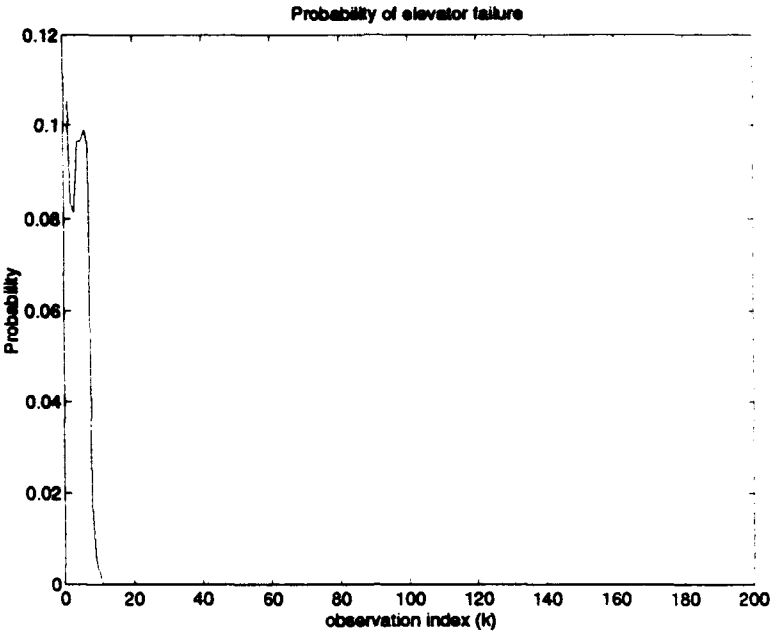


Figure B.53: Probability of Soft Failure on the Elevator Actuator

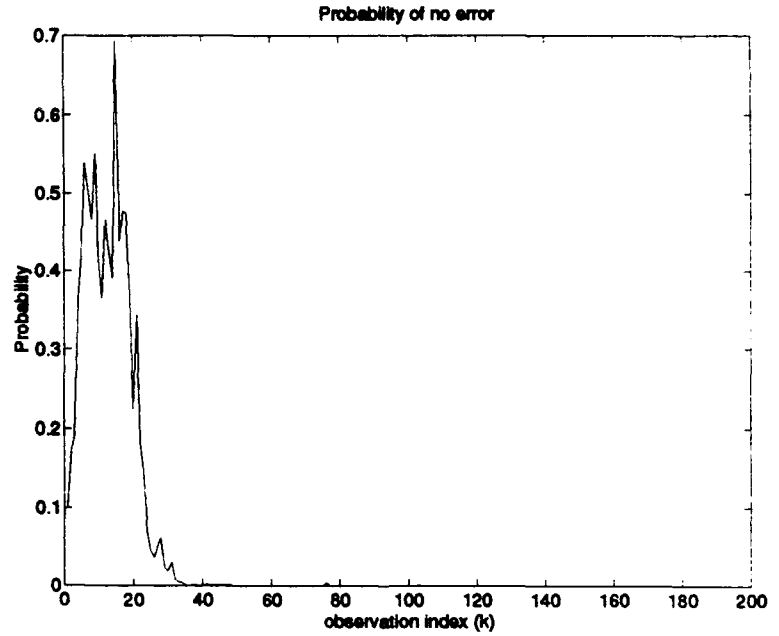


Figure B.54: Probability of Soft Failure on the Elevator Actuator

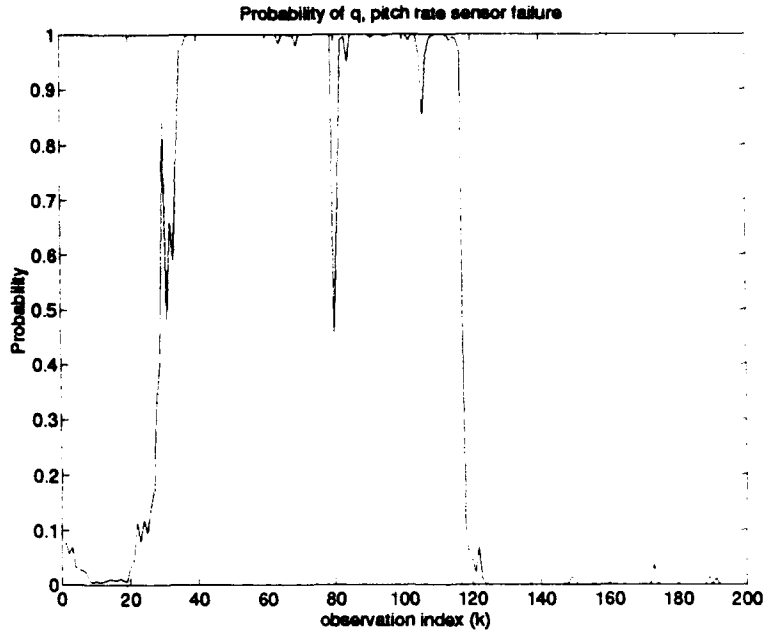


Figure B.55: Probability of Soft Failure on the Elevator Actuator

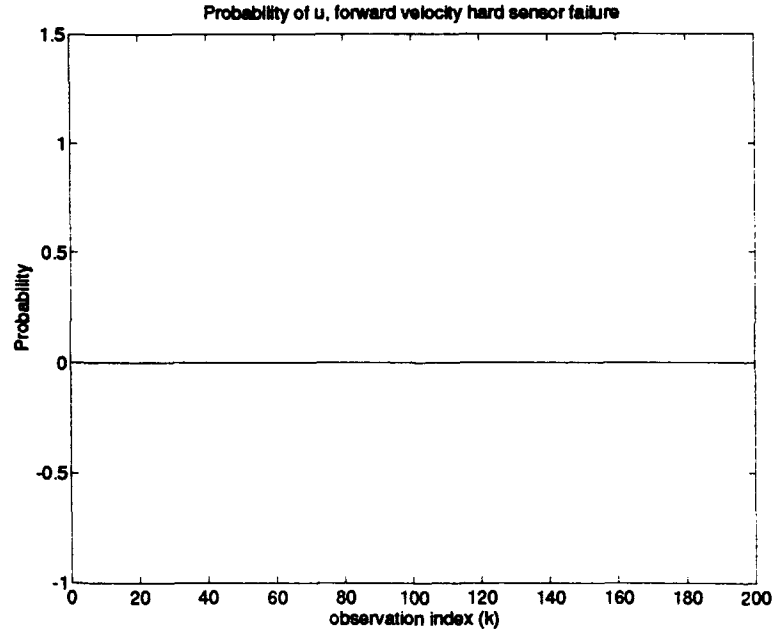


Figure B.56: Probability of Soft Failure on u sensor

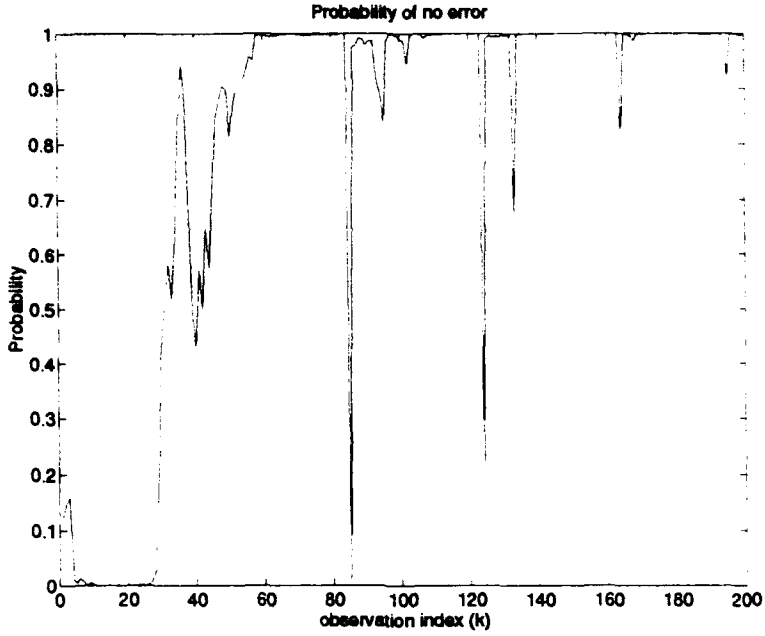


Figure B.57: Probability of Soft Failure on u sensor

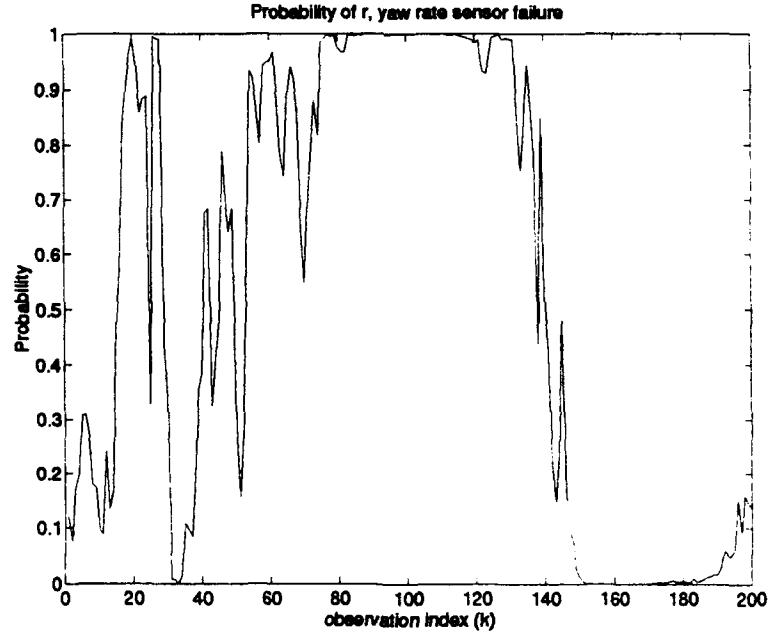


Figure B.58: Probability of Soft Failure on u sensor

APPENDIX C: FDI PROGRAM LISTING

fdi.m

```
% Fault detection and isolation for the Bluebird test bed aircraft
%
% By: Capt M. Levesque
%     Canadian Air Force
%
% Main file in thesis for the partial fulfillement of
% the requirement for the degree of MSEE
%
%   This is the main file that call all the subroutines
%required for the fdi algorithm
%
clear
clg
clf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ABCU_load       %loads nominal matrix values for aircraft, A,B,C and u

kmax=Stop_time/dt;

% Plant matrix
[rC,cC]=size(C);
[rA,cA]=size(A);

%initialisation of input matrices from selected failure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[Bm,Cm,Wm,Vm]=faultchoice(B,C,Wik,Vik,incipient_fault_noise_factor);

%noise generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Qm=zeros(rA,(rA*(kmax)));       %covariance of process noise
Rm=zeros(rC,(rC*(kmax)));       %covariance of measurment noise

  wm=randn(rA,kmax);  %randn(M,N) is an M by N matrix with random
  vm=randn(rC,kmax);  %entries normally distributed, mean 0.0 var 1.

  %cov(X) is covariance matrix of X when each row is an observation and
  %each column a variable
  Qm=cov(wm'); %unweighted covariance matrix of the process/plant noise
  Rm=cov(vm'); %unweighted covariance matrix of the measurement noise
 [phidummy,GAMMAwm]=c2d(A,Wm,dt);      %gives phi(k+1,k) and Gammaw(k)

%calls to calculate gains covariances of error matrices
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
need_to_calculate_gains=1
if need_to_calculate_gains==1

%Kalman filter tuned for no failure;
   B1kf=B;
   C1kf=C;
[G1,Pkk1,a1]=kalmn_gain(P_init,Vik,Wik,A,B1kf,C1kf,dt,Stop_time);

%hard actuator failures
%%%%%%%%%%%%%%%%%%%%%%%%
%Kalman filter tuned for the system with elevator hard failure;
   B2kf=B*diag([0 1 1 1]);
   C2kf=C;
[G2,Pkk2,a2]=kalmn_gain(P_init,Vik,Wik,A,B2kf,C2kf,dt,Stop_time);

%Kalman filter tuned for the system with aileron hard failure;
   B3kf=B*diag([1 0 1 1]);
   C3kf=C;
[G3,Pkk3,a3]=kalmn_gain(P_init,Vik,Wik,A,B3kf,C3kf,dt,Stop_time);

%Kalman filter tuned for the system with rudder hard failure;
   B4kf=B*diag([1 1 0 1]);
   C4kf=C;
[G4,Pkk4,a4]=kalmn_gain(P_init,Vik,Wik,A,B4kf,C4kf,dt,Stop_time);

%Kalman filter tuned for the system with thrust actuator hard failure;
   B5kf=B*diag([1 1 1 0]);
   C5kf=C;
[G5,Pkk5,a5]=kalmn_gain(P_init,Vik,Wik,A,B5kf,C5kf,dt,Stop_time);

%hard sensor failures
%%%%%%%%%%%%%%%%%%%%%%%%
%Kalman filter tuned for the system with u, forward velocity
 hard failure;
   B6kf=B;
   C6kf=C*diag([0 1 1 1 1 1 1 1 1]);
[G6,Pkk6,a6]=kalmn_gain(P_init,Vik,Wik,A,B6kf,C6kf,dt,Stop_time);

%Kalman filter tuned for the system with v, lateral velocity
hard failure;
   B7kf=B;
   C7kf=C*diag([1 0 1 1 1 1 1 1 1]);
[G7,Pkk7,a7]=kalmn_gain(P_init,Vik,Wik,A,B7kf,C7kf,dt,Stop_time);

%Kalman filter tuned for the system with w, vertical velocity
 hard failure;
   B8kf=B;
   C8kf=C*diag([1 1 0 1 1 1 1 1 1]);
[G8,Pkk8,a8]=kalmn_gain(P_init,Vik,Wik,A,B8kf,C8kf,dt,Stop_time);
```

```
%Kalman filter tuned for the system with p, roll rate hard failure;
   B9kf=B;
   C9kf=C*diag([1 1 1 0 1 1 1 1 1]);
[G9,Pkk9,a9]=kalmn_gain(P_init,Vik,Wik,A,B9kf,C9kf,dt,Stop_time);

%Kalman filter tuned for the system with q, pitch rate hard failure;
   B10kf=B;
   C10kf=C*diag([1 1 1 1 0 1 1 1 1]);
[G10,Pkk10,a10]=kalmn_gain(P_init,Vik,Wik,A,B10kf,C10kf,dt,Stop_time);

%Kalman filter tuned for the system with r, yaw rate hard failure;
   B11kf=B;
   C11kf=C*diag([1 1 1 1 1 0 1 1 1]);
[G11,Pkk11,a11]=kalmn_gain(P_init,Vik,Wik,A,B11kf,C11kf,dt,Stop_time);

%Kalman filter tuned for the system with phi, roll sensor hard failure;
   B12kf=B;
   C12kf=C*diag([1 1 1 1 1 1 0 1 1]);
[G12,Pkk12,a12]=kalmn_gain(P_init,Vik,Wik,A,B12kf,C12kf,dt,Stop_time);

%Kalman filter tuned for the system with theta, angle of attack sensor
 hard failure;
   B13kf=B;
   C13kf=C*diag([1 1 1 1 1 1 1 0 1]);
[G13,Pkk13,a13]=kalmn_gain(P_init,Vik,Wik,A,B13kf,C13kf,dt,Stop_time);

%Kalman filter tuned for the system with psi, heading sensor
hard failure;
   B14kf=B;
   C14kf=C*diag([1 1 1 1 1 1 1 1 0]);
[G14,Pkk14,a14]=kalmn_gain(P_init,Vik,Wik,A,B14kf,C14kf,dt,Stop_time);


end;  %for if  need_to_calculate_gains

%Calls to kalman filter function to calculate residue and state
estimates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%the nominal system
%%%%%%%%%%%%%%%%%%%%%%%
%Kalman filter tuned for no failure;
[res1,xhat1]=kalman_i(G1,Pkk1,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B1kf,
C1kf,dt,Stop_time);

%hard actuator failures
%%%%%%%%%%%%%%%%%%%%%%%
%Kalman filter tuned for the system with elevator hard failure;
[res2,xhat2]=kalman_i(G2,Pkk2,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B2kf,
C2kf,dt,Stop_time);
```

```
%Kalman filter tuned for the system with aileron hard failure;
[res3,xhat3]=kalman_i(G3,Pkk3,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B3kf,
C3kf,dt,Stop_time);

%Kalman filter tuned for the system with rudder hard failure;
[res4,xhat4]=kalman_i(G4,Pkk4,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B4kf,
C4kf,dt,Stop_time);

%Kalman filter tuned for the system with thrust actuator hard failure;
[res5,xhat5]=kalman_i(G5,Pkk5,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B5kf,
C5kf,dt,Stop_time);

%hard sensor failures
%%%%%%%%%%%%%%%%%%%%%%%%
%Kalman filter tuned for the system with u, forward velocity hard
 failure;
[res6,xhat6]=kalman_i(G6,Pkk6,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B6kf,
C6kf,dt,Stop_time);

%Kalman filter tuned for the system with v, lateral velocity hard
 failure;
[res7,xhat7]=kalman_i(G7,Pkk7,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B7kf,
C7kf,dt,Stop_time);

%Kalman filter tuned for the system with w, vertical velocity hard
 failure;
[res8,xhat8]=kalman_i(G8,Pkk8,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B8kf,
C8kf,dt,Stop_time);

%Kalman filter tuned for the system with p, roll rate hard failure;
[res9,xhat9]=kalman_i(G9,Pkk9,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,B9kf,
C9kf,dt,Stop_time);

%Kalman filter tuned for the system with q, pitch rate hard failure;
[res10,xhat10]=kalman_i(G10,Pkk10,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,
B10kf,C10kf,dt,Stop_time);

%Kalman filter tuned for the system with r, yaw rate hard failure;
[res11,xhat11]=kalman_i(G11,Pkk11,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,
B11kf,C11kf,dt,Stop_time);

%Kalman filter tuned for the system with phi, roll sensor hard failure;
[res12,xhat12]=kalman_i(G12,Pkk12,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,
B12kf,C12kf,dt,Stop_time);

%Kalman filter tuned for the system with theta, angle of attack
 sensor hard failure;
[res13,xhat13]=kalman_i(G13,Pkk13,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,
B13kf,C13kf,dt,Stop_time);

%Kalman filter tuned for the system with psi, heading sensor hard
```

```
 failure;
[res14,xhat14]=kalman_i(G14,Pkk14,xk_init,GAMMAwm,Vm,wm,vm,A,Bm,Cm,u,
B14kf,C14kf,dt,Stop_time);

%Soft or incipient failure
%%%%%%%%%%%%%%%%%%%%%%%%%%

%Calls to probability function to calculate the probabilities on
 each filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pkx=pk(res1,a1,res2,a2,res3,a3,res4,a4,res5,a5,res6,a6,res7,a7,res8,a8,
res9,a9,res10,a10,res11,a11,res12,a12,res13,a13,res14,a14,dt,Stop_time);
```

ABCU_load.m

```
%this m file is executed at the beginning of the fdi main file to load
%the memory with the following initial conditions

Stop_time=2;
dt=0.01;

%verification with model in the book by Candy p.127
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A=[0.999 0.0
   0    1];
B=[1
   0];
u=0;
C=[29.8 -0.623
    0     24.9];
xk_init=[2.5
         2.5];
P_init=1e-6;
Wik=[sqrt(10) sqrt(10)]';
Vik=[sqrt(5.06e4) sqrt(1.4e5)]';

%verification with model in the book by Andrews and Grewal p.143
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A=[0 1
   -25 -2*.2*5];
B=[0
   12];
u=1;
C=[1 0];
xk_init=[0
         0];
P_init=2;
Wik=[0 sqrt(4.47)]';
Vik=[sqrt(0.01)]';
incipient_fault_noise_factor=5;

%model simulation using numbers from Dave Kuechenmeister's thesis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A=[-0.0635       0  0.3277       0 -1.4922        0       0 -32.1740  0
         0 -0.3911       0  1.6086       0 -72.6109 32.1740        0  0
   -0.7572       0 -4.7741       0 67.9934        0 -0.0002  -0.0002  0
         0 -0.1471       0 -5.4414       0   1.5183       0        0  0
    0.0151       0 -0.1933       0 -3.1672        0       0        0  0
         0  0.1440       0 -1.0578       0  -0.8114       0        0  0
         0       0       0       1       0        0       0        0  0
         0       0       0       0       1        0       0        0  0
         0       0       0       0       0        1       0        0  0];
%
B=[ -4.5835       0       0 8.7745
```

```
         0  5.8282       0       0
  -38.8681       0       0       0
         0  0.6252 47.1717       0
  -22.0417       0       0       0
         0 -7.3151 -6.4345       0
         0       0       0       0
         0       0       0       0
         0       0       0       0];

%deterministic control input
u=[ -0.0181
         0
         0
    0.2336];
C=eye(length(A));

xk_init=[73.3000
         0.0
         0.0
         0.0
         0.0
         0.0
         0.0
         0.0
         0.0];

P_init=1;

Wik=[sqrt(0.1) sqrt(0.01) sqrt(0.01),...
        sqrt(0.001) sqrt(0.001) sqrt(0.001),...
        sqrt(0.001) sqrt(0.001) sqrt(0.001)]'; %Ww=[sigma ....]

Vik=[sqrt(10) sqrt(1) sqrt(1) sqrt(0.057) sqrt(0.057) sqrt(0.057),...
        sqrt(0.2*pi/180) sqrt(0.2*pi/180) sqrt(3*pi/180)]';

incipient_fault_noise_factor=5;

%model simulation using numbers from Eric Hallberg's thesis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A=[-0.0825  0.1193  0.0467  0.0011  1.3303   3.4789  0.0027 -32.0613 0
   -0.1020 -0.4281 -0.0059 -1.4341       0 -80.7651 30.8814   0.7255 0
   -0.8561 -0.0427 -5.2818 -3.4481 75.6252   0.0006 -8.4582   2.5534 0
    0.0048 -0.1583  0.0149 -6.0885  0.3198   1.6817 -0.0074   0.0022 0
    0.0069 -0.0039 -0.2155 -0.1305 -3.5224   0.0441  0.1064  -0.0321 0
   -0.0061  0.1680 -0.0002 -0.8975 -0.0086  -0.9863  0.0002        0 0
         0       0       0  1.0000 -0.0226  -0.0811 -0.0001   0.0971 0
         0       0       0       0  0.9632  -0.2687 -0.0965        0 0
         0       0       0       0  0.2696   0.9666  0.0016  -0.0082 0];

%
```

```
B=[-7.5739         0 -0.3370 8.7745
   -0.2898         0  7.2176      0
  -47.9100         0 -1.4904      0
    1.8944  58.1206  0.4870      0
  -27.2996   1.5561  0.0352      0
   -0.0418 -10.8643 -9.0965      0
         0         0       0      0
         0         0       0      0
         0         0       0      0];

%deterministic control input
u=[ 0
    0
    0
    .75];

C=eye(length(A));

xk_init=[81.5317
          3.5119
         -1.4341
          0.0042
          0.0275
          0.0925
          0.2720
         -0.0840
          1.7317];

P_init=1;
wf=1.5;
Wik=[1*wf 0.5*wf 0.5*wf 0.057*wf 0.057*wf 0.4*wf 0.3*pi/180*wf
 0.3*pi/180*wf 1*pi/180*wf]';

Vik=[1 .5 .5 0.57 0.57 0.57 0.5*pi/180 0.5*pi/180 3*pi/180]';

incipient_fault_noise_factor=2;
```

faultchoice.m

```
function [Bm,Cm,Wm,Vm]=faultchoice(B,C,Wik,Vik,
incipient_fault_noise_factor)
%
% This function modifies the values of the matrices A, B, C, or u
% according to the type of failure selected from the user.
%
%
%

%choice of failure to be simulated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Failure_type=menu('Choose a fault scenario to test the FDI algorithm',
                   ...
                 'no failure', ...
                 'Hard or abrupt failure', ...
                 'Soft or incipient failure');

if Failure_type ~= 1
   Failure_source=menu('Choose a failure source', ...
                       'Actuator Failure', ...
                       'Sensor Failure');
   if Failure_source == 1
       Actuator_type=menu('Choose a  control surface related to your
 actuator failure', ...
                          'Elevator', ...
                          'Aileron', ...
                          'Rudder', ...
                          'thrust actuator');
   elseif  Failure_source == 2
       Sensor_type=menu('Choose a parameter related to your
                         sensor  failure', ...
                        'u', ...
                        'v', ...
                        'w', ...
                        'p', ...
                        'q', ...
                        'r', ...
                        'phi', ...
                        'theta', ...
                        'psi');
       end;
   end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%implementation of selected failure to input matrices
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%no failures
%%%%%%%%%%%%%%%%%%%%
```

```
if Failure_type == 1
     Bm=B;
     Cm=C;
     Wm=Wik;
     Vm=Vik;
%Hard or abrupt failures
%%%%%%%%%%%%%%%%%%%%%%%%
elseif Failure_type == 2
  if Failure_source == 1          %actuator hard failure
     Wm=Wik;
     Vm=Vik;
     Cm=C;
     if  Actuator_type == 1              %elevator hard failure
        Bm=B*diag([0 1 1 1]);
     elseif  Actuator_type == 2          %aileron hard failure
        Bm=B*diag([1 0 1 1]);
     elseif  Actuator_type == 3          %rudder hard failure
        Bm=B*diag([1 1 0 1]);
     elseif  Actuator_type == 4          %Thrust actuator hard failure
        Bm=B*diag([1 1 1 0]);
     end;

  elseif Failure_source == 2     %sensor hard failure
     Wm=Wik;
     Vm=Vik;
     Bm=B;
     if  Sensor_type == 1              %u, forward velocity hard failure
        Cm=C*diag([0 1 1 1 1 1 1 1 1]);
     elseif  Sensor_type == 2          %v, lateral velocity hard failure
        Cm=C*diag([1 0 1 1 1 1 1 1 1]);
     elseif  Sensor_type == 3         %w, vertical velocity hard failure
        Cm=C*diag([1 1 0 1 1 1 1 1 1]);
     elseif  Sensor_type == 4                %p, roll rate hard failure
        Cm=C*diag([1 1 1 0 1 1 1 1 1]);
     elseif  Sensor_type == 5               %q, pitch rate hard failure
        Cm=C*diag([1 1 1 1 0 1 1 1 1]);
     elseif  Sensor_type == 6                %r, yaw rate hard failure
        Cm=C*diag([1 1 1 1 1 0 1 1 1]);
     elseif  Sensor_type == 7            %phi, roll sensor hard failure
        Cm=C*diag([1 1 1 1 1 1 0 1 1]);
     elseif  Sensor_type == 8%theta, angle of attack sensor hard failure
        Cm=C*diag([1 1 1 1 1 1 1 0 1]);
     elseif  Sensor_type == 9         %psi, heading  sensor hard failure
        Cm=C*diag([1 1 1 1 1 1 1 1 0]);
     end;
  end;


%Soft or incipient failure
%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
elseif Failure_type == 3

nf=incipient_fault_noise_factor;

    if Failure_source == 1   %actuator soft failure
       Bm=B;
       Cm=C;
       Wm=Wik;
       Vm=Vik;
       if  Actuator_type == 1              %elevator hard failure
          Bm=B*diag([nf 1 1 1]);
       elseif  Actuator_type == 2          %aileron hard failure
          Bm=B*diag([1 nf 1 1]);
       elseif  Actuator_type == 3          %rudder hard failure
          Bm=B*diag([1 1 nf 1]);
       elseif  Actuator_type == 4      %Thrust actuator hard failure
          Bm=B*diag([1 1 1 nf]);
       end;

    elseif Failure_source == 2     %sensor soft failure
       Bm=B;
       Cm=C;
       Wm=Wik;
       if  Sensor_type == 1          %u, forward velocity hard failure
          Vm=diag([nf 1 1 1 1 1 1 1])*Vik;
       elseif  Sensor_type == 2      %v, lateral velocity hard failure
          Vm=diag([1 nf 1 1 1 1 1 1])*Vik;
       elseif  Sensor_type == 3      %w, vertical velocity hard failure
          Vm=diag([1 1 nf 1 1 1 1 1])*Vik;
       elseif  Sensor_type == 4      %p, roll rate hard failure
          Vm=diag([1 1 1 nf 1 1 1 1])*Vik;
       elseif  Sensor_type == 5      %q, pitch rate hard failure
          Vm=diag([1 1 1 1 nf 1 1 1])*Vik;
       elseif  Sensor_type == 6      %r, yaw rate hard failure
          Vm=diag([1 1 1 1 1 nf 1 1])*Vik;
       elseif  Sensor_type == 7      %phi, roll sensor hard failure
          Vm=diag([1 1 1 1 1 1 nf 1])*Vik;
       elseif  Sensor_type ==8%theta, angle of attack sensor hard failure
          Vm=diag([1 1 1 1 1 1 1 nf])*Vik;
       elseif  Sensor_type == 9      %psi, heading  sensor hard failure
          Vm=diag([1 1 1 1 1 1 1 1 nf])*Vik;
       end;

    end;
end;
```

kalmn_gain.m

```
function [G,Pkk,a]=kalmn_gain(P_init,Vik,Wik,A,Bikf,Cikf,dt,Stop_time)

kmax=Stop_time/dt;

% Plant matrix
[rC,cC]=size(Cikf);
[rA,cA]=size(A);

%matrix initialisation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pkkm1=zeros(rA,(rA*kmax+1));  %P(k/k-1)
Pkk=zeros(rA,(rA*(kmax)));    %P(k/k) covariance of estimation
                                             %error matrix
G=zeros(rA,(rC*(kmax)));      %Kalman Gains
a=zeros(rC,(rC*(kmax)));      %hypothesized filter's internally
                              %computed residual covariance

%noise generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Qik=zeros(rA,(rA*(kmax)));      %covariance of process noise
Rik=zeros(rC,(rC*(kmax)));      %covariance of measurment noise

  w=randn(rA,kmax);  %randn(M,N) is an M by N matrix with random
  v=randn(rC,kmax);  %entries normally distributed, mean 0.0 var 1.

  %cov(X) is covariance matrix of X when each row is an observation and
  %each column a variable
  Qik=cov(w'); %unweighted covariance matrix of the process/plant noise
  Rik=cov(v'); %unweighted covariance matrix of the measurement noise
 [phidummy,GAMMAik]=c2d(A,Wik,dt);       %gives phi(k+1,k) and Gammaw(k)


%                                  KALMAN FILTER CALCULATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[phi_ikf,del_ikf]=c2d(A,Bikf,dt);       %gives phi(k+1,k) and del(k)


% Offline calculations, generation of gain schedule
%   (The Gains do not depend on measurement data)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:kmax

  %set P(0/-1)=P_init, the covariance of initial state value
  if i==1,
     Pkkm1(:,(i*rA-(rA-1):i*rA)) = ...
```

```
        ones(rA,rA).*P_init;
end;

%hypothesized filter's internally computed residual covariance used
 in G
a(:,(i*rC-(rC-1):i*rC)) = ...
     ( Cikf * Pkkm1(:,(i*rA-(rA-1):i*rA)) * Cikf' + ...
     (Rik*diag(Vik)) );

%Kalman Gains
G(:,(i*rC-(rC-1):i*rC)) = ...
      Pkkm1(:,(i*rA-(rA-1):i*rA)) * Cikf' *...
      inv( a(:,(i*rC-(rC-1):i*rC)) );

%P(k/k), covariance of estimation error matrix
Pkk(:,(i*rA-(rA-1):i*rA)) = (eye(rA) - ...
    G(:,(i*rC-(rC-1):i*rC)) * Cikf) * ...
    Pkkm1(:,(i*rA-(rA-1):i*rA));

%discrete Lyapunov equation
%update the estimation error covariance matrix, new Pkkm1 or P(k+1/k)
Pkkm1(:,((i+1)*rA-(rA-1):(i+1)*rA)) = phi_ikf * ...
      Pkk(:,(i*rA-(rA-1):i*rA)) * phi_ikf' + ...
      (GAMMAik*GAMMAik'*Qik);


end;

% m=mean(a');
% tot_mean_a=mean(m)
% co=cov(a(1,:))
% size(a)
% a
%
```

kalman_i.m

```
function [residualk,xhatkk]=kalman_i(G,Pkk,xk_init,GAMMAwm,Vm,wm,vm,
                                     A,Bm,Cm,u,
Bikf,Cikf,dt,Stop_time)

%The function kalman_i
%
%This function calculates the residuals and the state estimates of
%a system using a kalman filter tuned according to the input values
%in above bracket.
%
%by CAPT M. Levesque
%   CAF
%   Nov 1993.

%                          INITIALISATION PART
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
!unalias rm
%!rm -r kalman.data
%diary kalman.data

kmax=Stop_time/dt;

% Plant matrix
[rC,cC]=size(Cm);
[rA,cA]=size(A);

%matrix initialisation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xk=zeros(rA,kmax+1);                %system states
yk=zeros(rC,kmax);                  %measurement updates
xhatkk=zeros(rA,kmax);              %xhat(k/k) state estimates
xhatkkm1=zeros(rA,kmax+1);          %xhat(k/k-1) state estimate a-priori
yhatk=zeros(rC,kmax);               %measurement estimates
residualk=zeros(rC,kmax);           %residual is y - yhat



%                                   KALMAN FILTER CALCULATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%              measurements
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[phi,del]=c2d(A,Bm,dt);      %gives phi(k+1,k) and del(k)
uinit=u;
for i=1:kmax,
      if i<1/dt %1 sec unit step input
      u=[3*pi/180
         5*pi/180
```

```
        3*pi/180
        1];
   else
   u=uinit;
   end

   if i==1,
      xk(:,i)=xk_init;
   end;

   xk(:,i+1)    = phi * xk(:,i) + del * u + diag(GAMMAwm) * wm(:,i);
   yk(:,i) = Cm * xk(:,i) + diag(Vm) * vm(:,i);

end;

%    Online calculations, generation of estimates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[phi_ikf,del_ikf]=c2d(A,Bikf,dt);      %gives phi(k+1,k) and del(k)


for i=1:kmax

  if i<1/dt  %1 sec unit step input
  u=[3*pi/180
     5*pi/180
     3*pi/180
     1];
  else
  u=uinit;
  end

  %initial states
  if i==1,
     xhatkkm1(:,i)=xk_init;
  end;

  %measurement estimate
  yhatk(:,i)       = Cikf*xhatkkm1(:,i);

  %innovation process, error in the measurement and estimate
  residualk(:,i) = yk(:,i)-yhatk(:,i);

  %state estimate
  xhatkk(:,i)     = xhatkkm1(:,i) + ...
                    G(:,(i*rC-(rC-1):i*rC)) * ...
                    (residualk(:,i));

  %State estimate update
  xhatkkm1(:,i+1)    = phi_ikf * xhatkk(:,i) + del_ikf * u;
 end;
```

```
%diary off

kalman_plots(A,Cm,G,Pkk,yhatk,yk,residualk,wm,vm,dt,Stop_time)
```

kalman_plots.m

```
function []=kalman_plots(A,C,G,Pkk,yhatk,yk,residualk,w,v,dt,Stop_time)

%The function kalman_plots
%
%This function is used to generate plots from the values calculated
%in the function kalman_i.
%
%by CAPT M. Levesque
%   CAF
%   Nov 1993.
%

%                                    PLOTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
[rC,cC]=size(C);
[rA,cA]=size(A);

kmax=Stop_time/dt;

k=1:kmax;

%plot of the noise
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
producenoiseplot=0;
if producenoiseplot==1

   plot(k,w)
   title('Process noise')
   xlabel('observation index (k)')
   ylabel('white noise (mean=0, variance=1)')
   pause;

   plot(k,v)
   title('Measurement noise')
   xlabel('observation index (k)')
   ylabel('white noise (mean=0, variance=1)')
   pause
end;
%plot of the kalman gains
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
producegainplot=0;
if producegainplot==1
k=1:kmax;
   for i=1:rA
      for r=1:rC
         plot(k,G(i,(k.*rC-(rC-r))))
         pause;
         hold on
```

```
      end;
      title(['Kalman Gain for the state x',num2str(i),])
      xlabel('observation index (k)')
      ylabel('Kalman Gain (G)')
      pause
      hold off


   end;
end;

%plot of the covariance matrix Pkk
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
producecovarianceplot=0;
if producecovarianceplot==1
k=1:kmax;
   for i=1:rA
     plot(k,Pkk(i,(k.*rA-(rA-i))))
     title(['Covariance of estimation error matrix, diagonal values for
            the state x',num2str(i),])
     xlabel('observation index (k)')
     ylabel('Covariance, P(k/k)')
     pause
    end;
end;

%plot of the measurements and its estimates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
producemeasurements=0;
if producemeasurements==1

k=1:(kmax);
for n=1:rC,
   plot(k,yhatk(n,:),'-.')
   hold on;
   plot(k,yk(n,:))
   title(['Measurement estimate (-.) yhat',num2str(n),'  and  updates
y',num2str(n),])
   xlabel('observation index (k)')
   ylabel(['Y',num2str(n),'  and  Yhat',num2str(n),])
   pause;
   hold off;
end;
end;
% plot of the residuals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
produceresidualplots=0;
if produceresidualplots==1

for n=1:rC,
   plot(k,residualk(n,:))
```

```
    title(['Residuals  (y-yhat)  rk',num2str(n),])
    xlabel('observation index (k)')
    ylabel(['rk',num2str(n),])
    pause;

end;
end;
```

pk.m

```
function [pk] = pk(res1,a1,res2,a2,res3,a3,res4,a4,res5,a5,res6,a6,
res7,a7,res8,a8,res9,a9,res10,a10,res11,a11,res12,a12,res13,a13,
res14,a14,dt,Stop_time)
%
%The function pk
%
%This function calculates the hypothesis conditional probability pk
%as the probability that a measured parameter assumes the value at k
%conditioned on the observed measurement history up to time k.
%
%


kmax=Stop_time/dt;
[ra,ca]=size(a1);

pk1_km1=1;  %pk(:,0)=1;
pk2_km1=1;
pk3_km1=1;
pk4_km1=1;
pk5_km1=1;
pk6_km1=1;
pk7_km1=1;
pk8_km1=1;
pk9_km1=1;
pk10_km1=1;
pk11_km1=1;
pk12_km1=1;
pk13_km1=1;
pk14_km1=1;

min= 0.0001;


% case for k=1 to have a value for p(t-1), index can not equal zero!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k=1;

        fzk1(k) = 1 / ((2*pi)^(ra*(1/2))*det(a1(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res1(:,k)'*inv(a1(:,(k*ra-(ra-1):k*ra))) * ...
res1(:,k) );

        fzk2(k) = 1 / ((2*pi)^(ra*(1/2))*det(a2(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res2(:,k)'*inv(a2(:,(k*ra-(ra-1):k*ra))) * ...
res2(:,k) );

        fzk3(k) = 1 / ((2*pi)^(ra*(1/2))*det(a3(:,(k*ra-(ra-1):k*ra)))
```

```
^0.5) * ...
exp(-0.5*res3(:,k)'*inv(a3(:,(k*ra-(ra-1):k*ra))) * ...
res3(:,k) );

        fzk4(k) = 1 / ((2*pi)^(ra*(1/2))*det(a4(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res4(:,k)'*inv(a4(:,(k*ra-(ra-1):k*ra))) * ...
res4(:,k) );

        fzk5(k) = 1 / ((2*pi)^(ra*(1/2))*det(a5(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res5(:,k)'*inv(a5(:,(k*ra-(ra-1):k*ra))) * ...
res5(:,k) );

        fzk6(k) = 1 / ((2*pi)^(ra*(1/2))*det(a6(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res6(:,k)'*inv(a6(:,(k*ra-(ra-1):k*ra))) * ...
res6(:,k) );

        fzk7(k) = 1 / ((2*pi)^(ra*(1/2))*det(a7(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res7(:,k)'*inv(a7(:,(k*ra-(ra-1):k*ra))) * ...
res7(:,k) );

        fzk8(k) = 1 / ((2*pi)^(ra*(1/2))*det(a8(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res8(:,k)'*inv(a8(:,(k*ra-(ra-1):k*ra))) * ...
res8(:,k) );

        fzk9(k) = 1 / ((2*pi)^(ra*(1/2))*det(a9(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res9(:,k)'*inv(a9(:,(k*ra-(ra-1):k*ra))) * ...
res9(:,k) );

       fzk10(k) = 1 / ((2*pi)^(ra*(1/2))*det(a10(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res10(:,k)'*inv(a10(:,(k*ra-(ra-1):k*ra))) * ...
res10(:,k) );

       fzk11(k) = 1 / ((2*pi)^(ra*(1/2))*det(a11(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res11(:,k)'*inv(a11(:,(k*ra-(ra-1):k*ra))) * ...
res11(:,k) );

       fzk12(k) = 1 / ((2*pi)^(ra*(1/2))*det(a12(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res12(:,k)'*inv(a12(:,(k*ra-(ra-1):k*ra))) * ...
res12(:,k) );

       fzk13(k) = 1 / ((2*pi)^(ra*(1/2))*det(a13(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
```

```
exp(-0.5*res13(:,k)'*inv(a13(:,(k*ra-(ra-1):k*ra))) * ...
res13(:,k) );

       fzk14(k) = 1 / ((2*pi)^(ra*(1/2))*det(a14(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res14(:,k)'*inv(a14(:,(k*ra-(ra-1):k*ra))) * ...
res14(:,k) );

pk1(k)=fzk1(k)*pk1_km1/...
(fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1 + fzk5(k)*pk5_km1
 + ...
 fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk1(k)*pk1_km1);
if pk1(k)<min, pk1(k)=min; end;

pk2(k)=fzk2(k)*pk2_km1/...
(fzk1(k)*pk1_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1 + fzk5(k)*pk5_km1
 + ...
 fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk2(k)*pk2_km1);
if pk2(k)<min, pk2(k)=min; end;

pk3(k)=fzk3(k)*pk3_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk4(k)*pk4_km1 + fzk5(k)*pk5_km1
 + ...
 fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk3(k)*pk3_km1);
if pk3(k)<min, pk3(k)=min; end;

pk4(k)=fzk4(k)*pk4_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk5(k)*pk5_km1
 + ...
 fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk4(k)*pk4_km1);
if pk4(k)<min, pk4(k)=min; end;

pk5(k)=fzk5(k)*pk5_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk5(k)*pk5_km1);
```

```
if pk5(k)<min, pk5(k)=min; end;

pk6(k)=fzk6(k)*pk6_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk6(k)*pk6_km1);
if pk6(k)<min, pk6(k)=min; end;

pk7(k)=fzk7(k)*pk7_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk8(k)*pk8_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk7(k)*pk7_km1);
if pk7(k)<min, pk7(k)=min; end;

pk8(k)=fzk8(k)*pk8_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk9(k)*pk9_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk8(k)*pk8_km1);
if pk8(k)<min, pk8(k)=min; end;

pk9(k)=fzk9(k)*pk9_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1
 + ...
 fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk9(k)*pk9_km1);
if pk9(k)<min, pk9(k)=min; end;

pk10(k)=fzk10(k)*pk10_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1
 + ...
 fzk9(k)*pk9_km1 + fzk11(k)*pk11_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk10(k)*pk10_km1);
if pk10(k)<min, pk10(k)=min; end;

pk11(k)=fzk11(k)*pk11_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1
```

```
 + ...
 fzk9(k)*pk9_km1 + fzk10(k)*pk10_km1 + fzk12(k)*pk12_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk11(k)*pk11_km1);
if pk11(k)<min, pk11(k)=min; end;

pk12(k)=fzk12(k)*pk12_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1
 + ...
 fzk9(k)*pk9_km1 + fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + ...
 fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1 + fzk12(k)*pk12_km1);
if pk12(k)<min, pk12(k)=min; end;

pk13(k)=fzk13(k)*pk13_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1
 + ...
 fzk9(k)*pk9_km1 + fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + ...
 fzk12(k)*pk12_km1 + fzk14(k)*pk14_km1 + fzk13(k)*pk13_km1);
if pk13(k)<min, pk13(k)=min; end;

pk14(k)=fzk14(k)*pk14_km1/...
(fzk1(k)*pk1_km1 + fzk2(k)*pk2_km1 + fzk3(k)*pk3_km1 + fzk4(k)*pk4_km1
 + ...
 fzk5(k)*pk5_km1 + fzk6(k)*pk6_km1 + fzk7(k)*pk7_km1 + fzk8(k)*pk8_km1
 + ...
 fzk9(k)*pk9_km1 + fzk10(k)*pk10_km1 + fzk11(k)*pk11_km1 + ...
 fzk12(k)*pk12_km1 + fzk13(k)*pk13_km1 + fzk14(k)*pk14_km1);
if pk14(k)<min, pk14(k)=min; end;


%calculation for the rest of the samples
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k = 2:kmax

        fzk1(k) = 1 / ((2*pi)^(ra*(1/2))*det(a1(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res1(:,k)'*inv(a1(:,(k*ra-(ra-1):k*ra))) * ...
res1(:,k) );

        fzk2(k) = 1 / ((2*pi)^(ra*(1/2))*det(a2(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res2(:,k)'*inv(a2(:,(k*ra-(ra-1):k*ra))) * ...
res2(:,k) );

        fzk3(k) = 1 / ((2*pi)^(ra*(1/2))*det(a3(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res3(:,k)'*inv(a3(:,(k*ra-(ra-1):k*ra))) * ...
res3(:,k) );
```

```
        fzk4(k) = 1 / ((2*pi)^(ra*(1/2))*det(a4(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res4(:,k)'*inv(a4(:,(k*ra-(ra-1):k*ra))) * ...
res4(:,k) );

        fzk5(k) = 1 / ((2*pi)^(ra*(1/2))*det(a5(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res5(:,k)'*inv(a5(:,(k*ra-(ra-1):k*ra))) * ...
res5(:,k) );

        fzk6(k) = 1 / ((2*pi)^(ra*(1/2))*det(a6(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res6(:,k)'*inv(a6(:,(k*ra-(ra-1):k*ra))) * ...
res6(:,k) );

        fzk7(k) = 1 / ((2*pi)^(ra*(1/2))*det(a7(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res7(:,k)'*inv(a7(:,(k*ra-(ra-1):k*ra))) * ...
res7(:,k) );

        fzk8(k) = 1 / ((2*pi)^(ra*(1/2))*det(a8(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res8(:,k)'*inv(a8(:,(k*ra-(ra-1):k*ra))) * ...
res8(:,k) );

        fzk9(k) = 1 / ((2*pi)^(ra*(1/2))*det(a9(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res9(:,k)'*inv(a9(:,(k*ra-(ra-1):k*ra))) * ...
res9(:,k) );

       fzk10(k) = 1 / ((2*pi)^(ra*(1/2))*det(a10(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res10(:,k)'*inv(a10(:,(k*ra-(ra-1):k*ra))) * ...
res10(:,k) );

       fzk11(k) = 1 / ((2*pi)^(ra*(1/2))*det(a11(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res11(:,k)'*inv(a11(:,(k*ra-(ra-1):k*ra))) * ...
res11(:,k) );

       fzk12(k) = 1 / ((2*pi)^(ra*(1/2))*det(a12(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res12(:,k)'*inv(a12(:,(k*ra-(ra-1):k*ra))) * ...
res12(:,k) );

       fzk13(k) = 1 / ((2*pi)^(ra*(1/2))*det(a13(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res13(:,k)'*inv(a13(:,(k*ra-(ra-1):k*ra))) * ...
res13(:,k) );
```

```
     fzk14(k) = 1 / ((2*pi)^(ra*(1/2))*det(a14(:,(k*ra-(ra-1):k*ra)))
^0.5) * ...
exp(-0.5*res14(:,k)'*inv(a14(:,(k*ra-(ra-1):k*ra))) * ...
res14(:,k) );


pk1(k)=fzk1(k)*pk1(k-1)/...
(fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*pk4(k-1) + fzk5(k)*
pk5(k-1) + ...
 fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*pk8(k-1) + fzk9(k)*
pk9(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk1(k)*pk1(k-1));
if pk1(k)<min, pk1(k)=min; end;


pk2(k)=fzk2(k)*pk2(k-1)/...
(fzk1(k)*pk1(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*pk4(k-1) + fzk5(k)*
pk5(k-1) + ...
 fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*pk8(k-1) + fzk9(k)*
pk9(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk2(k)*pk2(k-1));
if pk2(k)<min, pk2(k)=min; end;


pk3(k)=fzk3(k)*pk3(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk4(k)*pk4(k-1) + fzk5(k)*
pk5(k-1) + ...
 fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*pk8(k-1) + fzk9(k)*
pk9(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk3(k)*pk3(k-1));
if pk3(k)<min, pk3(k)=min; end;


pk4(k)=fzk4(k)*pk4(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk5(k)*
pk5(k-1) + ...
 fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*pk8(k-1) + fzk9(k)*
pk9(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk4(k)*pk4(k-1));
if pk4(k)<min, pk4(k)=min; end;


pk5(k)=fzk5(k)*pk5(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*pk8(k-1) + fzk9(k)*
pk9(k-1) + ...
```

```
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk5(k)*pk5(k-1));
if pk5(k)<min, pk5(k)=min; end;


pk6(k)=fzk6(k)*pk6(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*pk8(k-1) + fzk9(k)*
pk9(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk6(k)*pk6(k-1));
if pk6(k)<min, pk6(k)=min; end;


pk7(k)=fzk7(k)*pk7(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk8(k)*pk8(k-1) + fzk9(k)*
pk9(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk7(k)*pk7(k-1));
if pk7(k)<min, pk7(k)=min; end;


pk8(k)=fzk8(k)*pk8(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk9(k)*
pk9(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk8(k)*pk8(k-1));
if pk8(k)<min, pk8(k)=min; end;


pk9(k)=fzk9(k)*pk9(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*
pk8(k-1) + ...
 fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk9(k)*pk9(k-1));
if pk9(k)<min, pk9(k)=min; end;


pk10(k)=fzk10(k)*pk10(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*
pk8(k-1) + ...
 fzk9(k)*pk9(k-1) + fzk11(k)*pk11(k-1) + fzk12(k)*pk12(k-1) + ...
```

```
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk10(k)*pk10(k-1));
if pk10(k)<min, pk10(k)=min; end;


pk11(k)=fzk11(k)*pk11(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*
pk8(k-1) + ...
 fzk9(k)*pk9(k-1) + fzk10(k)*pk10(k-1) + fzk12(k)*pk12(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk11(k)*pk11(k-1));
if pk11(k)<min, pk11(k)=min; end;


pk12(k)=fzk12(k)*pk12(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*
pk8(k-1) + ...
 fzk9(k)*pk9(k-1) + fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + ...
 fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1) + fzk12(k)*pk12(k-1));
if pk12(k)<min, pk12(k)=min; end;


pk13(k)=fzk13(k)*pk13(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*
pk8(k-1) + ...
 fzk9(k)*pk9(k-1) + fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + ...
 fzk12(k)*pk12(k-1) + fzk14(k)*pk14(k-1) + fzk13(k)*pk13(k-1));
if pk13(k)<min, pk13(k)=min; end;


pk14(k)=fzk14(k)*pk14(k-1)/...
(fzk1(k)*pk1(k-1) + fzk2(k)*pk2(k-1) + fzk3(k)*pk3(k-1) + fzk4(k)*
pk4(k-1) + ...
 fzk5(k)*pk5(k-1) + fzk6(k)*pk6(k-1) + fzk7(k)*pk7(k-1) + fzk8(k)*
pk8(k-1) + ...
 fzk9(k)*pk9(k-1) + fzk10(k)*pk10(k-1) + fzk11(k)*pk11(k-1) + ...
 fzk12(k)*pk12(k-1) + fzk13(k)*pk13(k-1) + fzk14(k)*pk14(k-1));
if pk14(k)<min, pk14(k)=min; end;

end; % index k

plot(1:k,pk1)
title('Probability of no error')
xlabel('observation index (k)')
ylabel('Probability')
pause
```

```
plot(1:k,pk2)
title('Probability of elevator failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk3)
title('Probability of aileron failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk4)
title('Probability of rudder failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk5)
title('Probability of thrust actuator failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk6)
title('Probability of u, forward velocity sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause


plot(1:k,pk7)
title('Probability of v, lateral velocity sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk8)
title('Probability of  w, vertical velocity sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk9)
title('Probability of  p, roll rate sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk10)
```

```
title('Probability of q, pitch rate sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk11)
title('Probability of r, yaw rate sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk12)
title('Probability of phi, roll sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk13)
title('Probability of theta, angle of attack sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause

plot(1:k,pk14)
title('Probability of  psi, heading sensor failure')
xlabel('observation index (k)')
ylabel('Probability')
pause
```

LIST OF REFERENCES

1. Cunningham, T.B., "Introduction and Overview," AGARD-LS-109 Fault Tolerance Design and Redundancy Management Techniques, pp. 1-1 - 1-2, September 1980.

2. Horak, D.T., "System Failure Isolation in Dynamic Systems," *J. Guidance*, Vol 13, No. 6, pp. 1075-1082, Nov-Dec 1990.

3. Ranieri, R., Redaelli, R., "Automatic Error Detection and Recovery Techniques in Onboard Intelligent Units for Space and Avionic Application," AGARD-CP-361 Design for Tactical Avionics Maintainability, pp. 29-1 - 29-4, May 1984.

4. Drtil, H., Meyer, W., "Failure Self-Detection in Digital Flight Guidance Systems," AGARD-AG-224, Integrity in Electronic Flight Control Systems, pp. 11-1 - 11-7, September 1980.

5. Patton, R.J., Frank, P.M., and Clark, R.N., *Fault Diagnosis in Dynamic Systems. Theory and Application*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

6. Willsky, A.S., "A Survey of Design Methods for Failure Detection in Dynamic Systems," *Automatica*, Vol 12, pp. 601-611, 1976.

7. Willsky, A.S., "Failure Detection in Dynamic Systems," AGARD-AG-224, Integrity in Electronic Flight Control Systems, pp. 2-1 - 2-14, September 1980.

8. Frank, P.M., "Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge-based Redundancy - A Survey and Some New Results," *Automatica*, Vol 26, No 3, pp.459-474, 1990.

9. Isermann, R., "Process Fault Detection Based on Modeling and Estimation Methods - A Survey," *Automatica*, Vol 20, No 4, pp. 387-404, 1984.

10. Lou, X.C., Willsky, A.S., Verghese, G.C., "Optimally Robust Redundancy Relations for failure Detection in Uncertain Systems," *Automatica*, Vol 22, No 3, pp. 333-344, 1986.

11. Saif, M., Guan, Y., "A New Approach to Robust Fault Detection and Identification," *IEEE Transactions on Aerospace and Electronic Systems*, Vol 29, No 3, pp. 685-695, July 1993 .

12. Frank, P.M., "Lecture Notes", *Abstracts*, 12th Benelux Meeting on Systems and Control, Houffalize, Belgium, March 3-5, 1993,

13. Kuechenmeister, D.R., *A Non-linear Simulation for an Autonomous Unmanned Air Vehicle*, MSAE Thesis, Dept. of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA, September 1993.

14. Menke, T.E., Maybeck, P.S., "Sensor/Actuator Failure Detection in the VISTA F-16 by multiple Model Adaptive Estimation," *Proceedings of the ACC*, San-Francisco, CA, pp.3135-3140, June 1993.

15. Magill, D.T., "Optimal Adaptive Estimation of Sampled Stochastic Processes." *IEEE Transactions on Automatic Control*, Vol AC-10, pp. 434-439, October 1965.

16. Candy, J.V., *Signal Processing, The Model-Based Approach*, McGraw-Hill, 1986.

17. Kirk, D.E., "Optimal Estimation: An Introduction to the Theory and Applications," Class Notes, U.S. Naval Postgraduate School, Monterey, CA, 1975.

18. Anderson, B.D., Moore, J.B., *Optimal Filtering*, Englewood Cliffs, N.J., Prentice Hall, 1979.

19. Andrews, A.P., Grewal, M.S., *Kalman Filtering Theory and Practice*, Prentice Hall, 1993.

BIBLIOGRAPHY

Ackermann, J., "Robust Control System Design," AGARD-AG-289 Fault Tolerant Considerations and Methods for Guidance and Control Systems, July 1987.

Anderson, B.D.O., Moore, J.B., *Optimal Control Linear Quadratic Methods*, Prentice Hall, 1990.

Bueno, R.A., *Performance and Sensitivity Analysis of the Generalized Likelihood Ratio Method for Failure Detection*, MSAE Thesis, M.I.T., Dept. of Aeronautics and Astronautics, Cambridge, Mass., February 1977.

Chow, E.Y., *A Failure Detection System Design Methodology* Ph.D. Dissertation, M.I.T. Dept. of Elec. Eng. and Comp. Sci., Cambridge, Mass., July 1980.

Da, R., "Failure Detection in Hybrid Strapdown - INS/GPS," AIAA Guidance, Navigation and Control conference, Monterey CA, August 9-11 1993.

Deckert, J.C., Szalai, K.J., "Analytic Redundancy Management for Flight Control Sensors," AGARD-AG-272 Advances in Sensors and their Integration into Aircraft Guidance and Control, June 1983.

Farrell, J., Berger, T., Appleby, B., "Using Learning Techniques to Accommodate Unanticipated Faults," *IEEE Control Systems Magazine*, June 1993.

Gelb, A., *Applied Optimal Estimation*, The M.I.T. Press, 1974.

Hall, S.R., Walker, B.K., "Orthogonal Series Generalized Likelihood Ratio Test for Failure Detection and Isolation," *J.Guidance*, Vol 13, No. 6, Nov-Dec 1990.

Isermann, R., "Fault Diagnosis of Machines via Parameter Estimation and Knowledge Processing - Tutorial Paper," *Automatica*, Vol 29, No 4, 1993.

Kubba, W.J., "Failure Detection, Isolation and Indication in Highly Integrated Digital Guidance and Control System," AGARD-CP-272 Advances in Guidance and Control Systems Using Digital Technique, August 1979.

Labarrere, M., "Detection de Pannes de Capteurs d'Avion par Utilisation de la Redondance Analytique," AGARD-AG-224, Integrity in Electronic Flight Control Systems, September 1980.

Labarrere, M., Pelegrin, M., Pircher, M., "Automatic Recovery after Sensor Failure on Board," AGARD-CP-272 Advances in Guidance and Control Systems Using Digital Technique, August 1979.

Laprie, J.C., "Computing Systems Dependability and Fault Tolerance: Basic Concepts and Terminology," AGARD-AG-289 Fault Tolerant Considerations and Methods for Guidance and Control Systems, July 1987.

Mariton, M., "Detection Delays, False Alarm Rates and the Reconfiguration of Control Systems," *Int. J. Control*, Vol 49, No. 3, 1989.

Marquis, C.W., *Integration of Differential GPS and Inertial Navigation using a Complementary Kalman Filter*, MSAE Thesis, Dept. of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA, September 1993.

Min, P.S., Ribbens, W.B., "A Vector Space Solution to Incipient Sensor Failure Detection with Applications to Automotive Environments," *IEEE Transactions on Vehicular Technology*, Vol.38, No.3, August 1989.

Onken, R., "System Integrity by use of Selfdiagnosis Failure Detection," AGARD-AG-224, Integrity in Electronic Flight Control Systems, September 1980.

Park, J., Rizzoni, G., "A New Interpretation of the Fault Detection Filter: The Optimal Detection Filter," *Proceedings of the ACC*, San-Francisco, CA, June 1993.

Singh, M.G., *Systems & Control Encyclopedia Theory, Technology, Applications*, Vol. 1, A-Com, Pergamon Press, Manchester, UK, 1987.

Stukenberg, N., "An Observer System For Sensor Failure Detection and Isolation in Digital Flight Control Systems," AGARD-CP-272 Advances in Guidance and Control Systems Using Digital Technique, August 1979.

Stukenberg, N., "Optimal Detection of Sensor Failures in Flight Control Systems using Deterministic observers," AGARD-AG-289 Fault Tolerant Considerations and Methods for Guidance and Control Systems, July 1987.

Szalai, K.J., Larson, R.R. and Glover, R.D., "Flight Experience with Flight Control Redundancy Management," AGARD-AG-224, Integrity in Electronic Flight Control Systems, September 1980.

Tsui, C-C., "A General Failure Detection, Isolation and Accommodation System With Model Uncertainty and Measurement Noise," *Proceedings of the ACC*, San Francisco, CA, June 1993.

West-Vukovich, G., Zywiel, J., Scherzinger, B., "The Honeywell/DND Helicopter Integrated Navigation System (HINS)," *IEEE AES Magazine*, March 1989.

Xia, Q., Rao, M., Shen, S.X., and Gourishankar, V.-G. "Robust Failure Detection, Estimation and Compensation in Linear Systems," *Proceedings of the ACC*, San Francisco, CA, June 1993.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Dr. Isaac I. Kaminer, Code AA/Ka Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, CA 93943-5000	5
5.	Dr Harold A. Titus, Code EC/Ts Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
6.	Michele Girard 26030 Atherton Dr Carmel, CA 93923	1

No. of Copies

7. DPED 1 — 1
National Defence Headquarter (NDHQ)
MGen George R. Pearkes Bldg
Ottawa, Ont,
K1A 0K2
CANADA

8. Directorate of Avionics, Simulators and Photography (DASP 2) — 1
National Defence Headquarter (NDHQ)
MGen George R. Pearkes Bldg
Ottawa, Ont,
K1A 0K2
CANADA

9. Capt. Mario J.L. Levesque — 1
Directorate of Avionics, Simulators and Photography (DASP 2-3-6)
National Defence Headquarter (NDHQ)
MGen George R. Pearkes Bldg
Ottawa, Ont,
K1A 0K2
CANADA