Spring 1988

A modified extended kalman filter as a parameter estimator for linear discrete-time systems

Bruno Johannes Schnekenburger
New Jersey Institute of Technology

A Modified Extended Kalman Filter As A Parameter Estimator For Linear Discrete-Time Systems

by

Bruno Johannes Schnekenburger

Thesis submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering

1988

APPROVAL SHEET

Title of Thesis:	A Modified Extended Kalman Filter As A Parameter Estimator For Linear Discrete-Time Systems
Name of Candidate:	Bruno Johannes Schnekenburger Master of Science in Electrical Engineering, 1988
Thesis and Abstract Approved:	[illegible] Date 6/1/88 Dr. Andrew U. Meyer Professor Electrical Engineering
Signature of other members of the thesis committee.	[illegible] Date 6/1/1988 Dr. B. Tarık Oranç

VITA

Name: Bruno Johannes Schnekenburger.

Permanent address:

Degree and date to be conferred: M.Sc., 1988.

Date of birth: .

Place of birth:

Secondary education: Technische Oberschule, Freiburg i. Br., West Germany, 1981

Collegiate institutions attended:	Date	Degree	Date of Degree
New Jersey Institute of Technology	9/86-5/88	M.Sc	Oct. 1988
Fachhochschule Offenburg	9/83-2/86	Dipl.-Ing.(FH)	Feb. 1986

Major: Electrical Engineering.

Publications: Schnekenburger, Bruno J. and Dahlmann, Horst "*Ringbus-System zur flexiblen Meßdatenerfassung*, Elektronik, vol. 24, December 1986, pp. 89–94.

Positions held: Teaching Assistant, New Jersey Inst. of Tech., Newark, NJ, 9/87-5/88
Research Engineer, Technologie Transfer Zentrum, Offenburg, West Germany, 4/86-6/86
Trainee, Hewlett-Packard R&D, Waldbronn, West Germany, 8/84-2/85.
Skilled Worker, Fernmeldeamt Offenburg, West Germany, 9/77-9/79.

ABSTRACT

Title of Thesis:	A Modified Extended Kalman Filter As A Parameter Estimator For Linear Discrete-Time Systems
Bruno J. Schnekenburger	Master of Science, 1988
Thesis directed by:	Prof. Dr. Andrew U. Meyer
	Asst. Prof. Dr. B. Tarık Oranç

This thesis presents the derivation and implementation of a modified Extended Kalman Filter used for joint state and parameter estimation of linear discrete-time systems operating in a stochastic Gaussian environment. A novel derivation for the discrete-time Extended Kalman Filter is also presented. In order to eliminate the main deficiencies of the Extended Kalman Filter, which are divergence and biasedness of its estimates, the filter algorithm has been modified. The primary modifications are due to Ljung, who stated global convergence properties for the modified Extended Kalman Filter, when used as a parameter estimator for linear systems.

Implementation of this filter is further complicated by the need to initialize the parameter estimate error covariance inappropriately small, to assure filter stability. In effect, due to this inadequate initialization process the parameter estimates fail to converge. Several heuristic methods have been developed to remove the effects of the inadequate initial parameter estimate covariance matrix on the filter's convergence properties.

Performance of the improved modified Extended Kalman Filter is compared with the Recursive Extended Least Squares parameter estimation scheme.

Für meine Eltern

To my Parents

Acknowledgement

The author would like to express his appreciation to Dr. Andrew U. Meyer of New Jersey Institute of Technology and to Dr. Halil Ö. Gülçür of Bosphorus University, Turkey, for their support, knowledge and insight, without this work would not have been possible. He also would like to extend a special thanks to Dr. B. Tarık Oranç of New Jersey Institute of Technology for his selfless dedication and generous support, far in excess of reasonable expectations.

Contents

List of Tables vi

List of Figures vii

1 Introduction **1**
- 1.1 Motivation and Objectives 1
- 1.2 Synopsis 2
- 1.3 Notation 3
- 1.4 Acronyms and Abbreviations 5

2 Theoretical Developments **6**
- 2.1 Introduction 6
- 2.2 The Discrete-Time Kalman Filter 7
  - 2.2.1 Problem Statement 9
  - 2.2.2 Kalman Filter Derivation 12
  - 2.2.3 KF for Systems with Deterministic Inputs 23
  - 2.2.4 Summary 25
  - 2.2.5 Conclusion 28
- 2.3 The Discrete-Time Extended Kalman Filter 29
  - 2.3.1 Problem Statement 30
  - 2.3.2 EKF Derivation 31
  - 2.3.3 Conclusion 41
- 2.4 The Extended Kalman Filter as a Parameter Estimator 42
- 2.5 A Modified Extended Kalman Filter 50
  - 2.5.1 Conclusions 53

3 Implementation of the MEKF **54**
- 3.1 Introduction 54
- 3.2 The Algorithm 55
- 3.3 Filter Initialization 64
- 3.4 Noise-Sequences Used for Testing the Filter 66
- 3.5 An Example of the Present Method 68
- 3.6 Conclusions 72

4 Investigation and Development of Various Methods to Improve Stability and Rate of Convergence of the MEKF Based on Single Parameter Case **73**
- 4.1 Introduction 73
- 4.2 Decelerated Convergence of P_3 74
- 4.3 Addition of Noise Term to the Parameter Vector 79
- 4.4 Enlargement of the Kalman Gain Matrix 85
- 4.5 Rejection of Spurious Data Points 88

4.6 Keep $\hat{\theta}$ in $\mathcal{D}_s$ 92
4.7 An Improved MEKF — Results 93
4.8 Conclusions 96

5 Comparison of an improved MEKF with a RELS Filter 98
5.1 Introduction 98
5.2 The RELS Algorithm 98
5.3 Implementation of the RELS Method 100
5.4 Results 101

6 Conclusions 105
6.1 Summary and Conclusions 105
6.2 Recommendations for Future Work 107

A Program Listing MEKF 108
A.1 Main Program 108
A.2 Subroutines 112

B Program Listing RELS 146

C Program Listing Noise Generator 149

D Properties of the Expected Value Operator E 151

E Proof of the Matrix Inversion Lemma 152

Bibliography 154

List of Tables

2.1 Summary of the 2 phase Kalman filter algorithm 25
2.2 Summary of the single phase Kalman filter algorithm 27
2.3 Summary of the 2 phase Extended Kalman Filter algorithm 35
2.4 Summary of the single phase Extended Kalman Filter algorithm . . 36
2.5 Summary of the partitioned single phase Extended Kalman Filter algorithm as a parameter estimator 49

3.1 Autocorrelation function associated with noise sequence N_SEQ13 . 67

List of Figures

3.1 Main Program MEKF Flowchart Part 1 58
3.2 Main Program MEKF Flowchart Part 2 59
3.3 Uniform Convergence of κ^i to $\partial K/\partial\theta^i$ 61
3.4 Oscillatory Convergence of κ^i to $\partial K/\partial\theta^i$ 61
3.5 Oscillatory Divergence of κ^i 61
3.6 Subprogram P31COMPKAPPA Flowchart 63
3.7 Parameter Estimates Test A 69
3.8 Variances of Parameter Estimates Test A 70
3.9 Parameter Estimates Test B 71

4.1 Parameter Estimates Test C, D and E 76
4.2 Variances of Parameter Estimates Test C, D and E 76
4.3 Comparison of Parameter Estimates from Test E and F 78
4.4 Comparison of Variances of Parameter Estimates from Test E and F 78
4.5 Parameter Estimates for Different Parameter Noise Covariances Test G and Test H 82
4.6 Parameter Estimates for Different Parameter Noise Covariances Test H and Test I 83
4.7 Variances of Parameter Estimates Test G and Test H 83
4.8 Variances of Parameter Estimates Test H and Test I 84
4.9 Parameter Estimates Test K (const. *gf* and Test L (variable *gf*) . . . 87
4.10 Variances of Parameter Estimates Test K and Test L 87
4.11 Parameter Estimates Test A and Test J 91
4.12 Variances of Parameter Estimates Test A and Test J 91
4.13 Parameter Estimates Test M 94
4.14 Variances of Parameter Estimates Test M 95

5.1 Parameter Estimates for 1st Order System 102
5.2 Parameter Estimates for 2nd Order System 103
5.3 Variances of Parameter Estimates (1st Order System) 103
5.4 Variances of Parameter Estimates (2nd Order System) 104

Chapter 1

Introduction

1.1 Motivation and Objectives

The application of modern control techniques to the control of today's industrial processes is gaining increasing importance. In order for the process control to be safe and economic, the process needs to be fully known. Mathematical process models allow the estimation of unmeasurable variables and process parameters. Many of these modern techniques are computationally costly. However, recent advances in hardware and software have made impressive computing power available at low cost, and thus opened up new fields of potential applications.

A wide spread and well known method to estimate and monitor the parameter process parameters, is the Extended Kalman Filter. It simultaneously estimates the state and the parameters of the system it is applied to. The Extended Kalman Filter, related to the well-known Kalman Filter, is an approximate filter, based on local linearization of the state equations. Though easy to implement, the Extended Kalman Filter tends to diverge, or gives biased estimates. Lennart Ljung disclosed in [33] the causes of biasedness and divergence, using his own method to analyse the asymptotic properties of recursive estimation algorithms. Ljung suggests a modification to the Extended Kalman Filter that converts the algorithm to a globally convergent filter [33].

In this thesis an improved modified Extended Kalman Filter is implemented, based on Ljung's recommendations. Numerous *Monte Carlo* simulations show the performance of this filter and the algorithms as developed by the author. To provide the reader with sufficient theoretical background, detailed derivations of both, the discrete-time Kalman Filter and the discrete-time Extended Kalman Filter are given.

1.2 Synopsis

Chapter 2 provides the necessary theoretical background, to provide a basis for the material of the later chapters. This includes derivations of the discrete-time Kalman Filter and of the discrete-time Extended Kalman Filter. Chapter 2 closes with the presentation of a modified Extended Kalman Filter.

Chapter 3 describes the different aspects associated with the implementation of the Modified Extended Kalman Filter, which are mainly the filter software and the filter initialization. Detailed information about the noise sequences used for testing the filters are also given, because properties of the noise is a very crucial part in system simulations.

In Chapter 4 several different methods intended to improve the rate of convergence in the Modified Extended Kalman Filter are introduced. Most presentations of these techniques are supplemented with reports on test outcomes of *Monte Carlo* simulations, to show their effectiveness.

How the Modified Extended Kalman Filter performs relative to another popular parameter estimator, namely the Recursive Extended Least Squares method, is examined in Chapter 5.

A summary of the thesis, along with concluding remarks and recommendations for future work on this subject, are given in Chapter 6.

The program listings and material furnishing mathematical background, are contained in the Appendices A–E.

1.3 Notation

x_k	State vector at time k
x_0	State vector at time k_0
$\hat{x}_{k\|j}$	Estimate for state vector at time k based on the set of measurements Z_j
$\tilde{x}_{k\|j}$	State estimation error vector $\tilde{x}_{k\|j} = x_k - \hat{x}_{k\|j}$
A_k	System matrix at time k
C_k	Measurement matrix at time k
e_k	Measurement noise vector
v_k	Process noise vector
$x \sim N(m, P)$	Indicates that x is normally distributed with mean m and covariance P
$\{x_k\}$	Denotes the set $\{(x_k, k) \mid k \geq k_0\}$
$P_{k\|j}$	Error covariance matrix at time k based on the set of measurement Z_j
$\hat{z}_{k+1}$	Estimate for z_{k+1} where the estimate is conditioned on the knowledge of Z_k
$z^{\star}_{k+1}$	Estimate for z_{k+1} where the estimate is conditioned on the knowledge of Z_k and x_{k+1}
$\tilde{z}_{k\|j}$	Residual at time k where the state estimate is based on the set of measurements Z_j
V_{k+1}	Error covariance matrix associated with the estimate $\hat{z}_{k+1}$
$V^{\star}_{k+1}$	Error covariance matrix associated with the estimate $z^{\star}_{k+1}$
K^F_k	Kalman gain matrix for updating the state vector (two phase algorithm)
K_k	Kalman gain matrix for updating the state vector (single phase algorithm)
L_k	Kalman gain matrix for updating the parameter vector
$f(\cdot)$	Nonlinear system matrix
$h(\cdot)$	Nonlinear measurement matrix

$F(\hat{x})$	Linear term in Taylor series expansion of $f(\cdot)$ at $\hat{x}$
$H(\hat{x})$	Linear term in Taylor series expansion of $h(\cdot)$ at $\hat{x}$
$p_{x_{k\|k}}$	Unconditional probability density function of $x_{k\|k}$
$p_{x_{k\|k}\|Z_k}$	Conditional probability density function of $x_{k\|k}$ knowing that Z_k has occurred
δ_{ij}	Kronecker delta δ_{ij} is 1 for $i = j$ and is zero otherwise
Z_k	Denotes the set of measurement vectors z_i starting from time $k_0 + 1$ up till time k; $Z = \{z_{k_0+1}, z_{k_0+2}, \cdots, z_k\}$
x_k^A	Augmented state vector at time k
η	Minimum variance estimate of x
S_k	Short form for $C_k P_{k\|k-1} C_k^T + Q_k^e$ (Kalman Filter case) or for $H(\hat{x}_{k\|k_1})\, P_{k\|k-1} \left(H(\hat{x}_{k\|k_1})\right)^T + Q_k^e$ (EKF case)
θ	Parameter vector
$\mathcal{R}^n$	Linear vector space of dimension n
$E\{\cdot\}$	Expected value
$EXP[\cdot]$	$e^{[\cdot]}$
M^T	Transpose of matrix M
M^{-1}	Inverse of matrix M
$\|M\|$	Determinant of matrix M
$trace[M]$	Trace of matrix M
$\|\|x\|\|$	L2-Norm (length) of vector x
x^i	Denotes i-th element of vector x
M^i	Denotes i-th column of matrix M
I	Identity matrix
$\underset{\sim}{0}$	Zero vector
$\equiv$	Identical (true for all $k \geq k_0$)

1.4 Acronyms and Abbreviations

ARMAX	Auto Regressive Moving Average eXogenous
EKF	Extended Kalman Filter
HOT	Higher order terms (in Taylor series expansion)
KF	Kalman Filter
MEKF	Modified Extended Kalman Filter
MIMO	Multiple Input Multiple Output
MV	Minimum variance
pd	positive definite
pdf	probability density function
RELS	Recursive Extended Least Square
RPE	Recursive Prediction Error

Chapter 2

Theoretical Developments

2.1 Introduction

As mentioned in the introductory chapter, the objective of this thesis is the implementation of a modified Extended Kalman Filter used for parameter estimation. This chapter is aimed to provide the reader with the theoretical background on modern filter theory necessary to fully grasp the material contained in the subsequent chapters. It is assumed however, that the reader is already familiar with such topics as matrix theory, state space techniques, probability theory and stochastic processes.

The discrete-time[1] Kalman Filter (KF) is derived first, because the two other filters, the Extended Kalman Filter (EKF) and a modified Extended Kalman Filter (MEKF) introduced subsequently are mere variations of the KF. In Section 2.3 it is shown, how Kalman Filter theory can, through linearization, be applied to nonlinear filtering problems and how this leads to the EKF algorithm. In Section 2.4 it is discussed that parameter estimation is, even for linear systems, a nonlinear filtering problem. How the EKF can be utilized to attack this problem, is also shown in Section 2.4. In practical applications the EKF tends to diverge or gives biased estimates. A modified EKF algorithm (modification due to Ljung [33]) with general

[1] Only filters, systems and models that are of discrete-time nature are treated here. The motivation for exclusion of continuous-time cases comes from the fact that in practical situations digital computers are used to observe and control systems.

convergence properties is introduced in Section 2.5.

2.2 The Discrete-Time Kalman Filter

In 1960, R. E. Kalman [27] derived a linear, optimal estimator for the estimation of state variables of linear, time-varying systems, operating in a Gaussian stochastic environment. *Optimal estimator* here is referred to a computational algorithm that processes measurements to deduce a minimum error covariance of the state of a system combining all the information available, i.e.:

- knowledge of system and measurement dynamics
- assumed statistics of system noise and measurement errors
- initial condition information

Kalman Filter theory includes — contrary to the classical techniques — non-stationary cases. A Kalman Filter (KF) is, under the Gaussian assumption, optimal, i.e. better than any other filter, and is for all non-Gaussian cases the best linear estimator. Kalman Filters have simple recursive[2] structures that can be implemented easily using digital computers.

There are many different ways to derive the KF algorithm. Kalman's original derivation of the discrete-time KF[3] [27] is based on the orthogonal projection method[4]. Another way to deduce the KF algorithm is to first assume the estimator to be linear and then to optimize it, by minimizing the length of the estimation error vector. These and other derivations of the KF can e.g. be found in [2], [25], [23], [7] or [39]. The basis for the derivation given in this work are Bayes' techniques,

[2]Recursive filters do not require the storage of past measurements, yet the present estimates are based on all data up to the present time.

[3]Kalman and Bucy derived in 1960/61 the continuous-time countertype to the KF[28], in the literature known as *Kalman-Bucy-Filter* or just *KF*.

[4]See [1] and [25] for further informations about *orthogonal projection*.

as described in [39][5]. Bayes' techniques will be used to propagate the conditional probability density function (pdf) of the state from one time instant to another.

Before going into the mathematical details of the derivation some thought will be given to what the Kalman Filter is about. In order to monitor a process, apply proper control signal or detect errors, the state of a system has to be known. An estimator is needed, wherever the states of a system are not directly accessible, but merely some noise corrupted measurements are available. It will be shown later that the state vector is Gaussian distributed at all discrete time instances, provided that system noise, measurement noise and intial state vector are Gaussian. Kalman Filter theory combines all the information available, to deduce the optimal estimate. The term *optimal* is used, because there is no other filter algorithm that results, on an average, in a smaller estimation error. The Kalman filter also computes an error covariance matrix for each estimate. This is as important as producing the estimate itself, because the state estimate is of little value if it is not known, how certain one can be about it.

To get a notion of how the KF operates, think of a simple example with only one state variable. Suppose, an estimate of this state variable and its associated variance are given at a certain time instant, say k. From this and the information about the system dynamics, the KF predicts (estimates) what the state will be at time $k+1$. This estimate is less certain (the covariance is larger) than the previous one, because of the process noise present, that drives the system state randomly. At each step in time, noise corrupted measurements become available. These measurements are utilized by the KF, to improve the quality of the state estimate, i.e. decrease the variance. If the measurement noise is large, the KF gives little weight to the measurement data, and modifies the estimate only a little; if the measurement noise is small, the new estimate is determined largely by the measurement data. In

[5]The derivation given in [39] served as one of the prime sources for the one presented here.

any case, whether the measurement noise is small or large, the measurement data contain some new information, that leads to a smaller variance, i.e. a state estimate, with greater certainty.

2.2.1 Problem Statement

Assume that a physical system[6] that generates the set of measurement vectors $Z_i = \{z_{k_0+1}, z_{k_0+2}, \ldots, z_{k_0+i}\}$, is adequately described by the following state space equations:

$$\begin{aligned} x_{k+1} &= A_k x_k + v_k & (2.1) \\ z_k &= y_k + e_k \\ &= C_k x_k + e_k & (2.2) \end{aligned}$$

where:

k	$>$	k_0	k_0 is initial time instant
x_k	$\in$	$\mathcal{R}^{nx}$	is system state vector
z_k	$\in$	$\mathcal{R}^{ny}$	is measurement vector
v_k	$\in$	$\mathcal{R}^{nx}$	is process noise vector
e_k	$\in$	$\mathcal{R}^{ny}$	is measurement noise vector
A_k	$\in$	$\mathcal{R}^{nx \times nx}$	is system matrix (in general time varying)
C_k	$\in$	$\mathcal{R}^{ny \times nx}$	is measurement matrix (in general time varying)

A_k and C_k are known matrices. x_k, z_k, y_k, e_k and v_k are random vectors of which only z_k is known. Scalars shall be included as special cases, so notation-wise no distinction will be made between vectors and scalars. The symbol $\{x_k\}$ stands for the set $\{(x_k, k) \mid k \geq k_0\}$; $\{x_k\}$ takes on the value x_k at time instant k. The meanings of $\{e_k\}$, $\{v_k\}$, $\{y_k\}$ and $\{z_k\}$ are analogous. The reader might have noticed that the model (2.1), (2.2) is without a deterministic input. This does not make the derivation less general, because a known input shifts the mean of the state vector, but has no effect on the shape of its distribution. The derivation is carried

[6]The underlying physical system can be of continuous-time type. A proper discretization as described in [25] or [2] will produce an equivalent discrete-time model.

out with this simpler system, solely to keep the equations involved more compact and easier to survey. Following the Kalman Filter derivation for systems without known inputs, it will be shown that the developed estimation algorithm can easily be modified to admit systems with deterministic inputs.

The filtering problem associated with the system described by equations (2.1) and (2.2) is to generate an *optimal*[7] estimate $\hat{x}_k$ for the state vector x_k at all time instants k utilizing the set of measurements Z_k. Without making certain assumptions about the initial state x_0 and the noise processes involved, very little can be done to produce this *optimal* estimate.

The following assumptions are made:

The initial state x_0 is assumed to be a Gaussian random vector. It thus suffices to know its mean $\hat{x}_0$ and its covariance matrix P_0:

$$E\{x_0\} = \hat{x}_0 \tag{2.3}$$

$$\tilde{x}_0 = x_0 - \hat{x}_0 \tag{2.4}$$

$$E\left\{\tilde{x}_0\tilde{x}_0^T\right\} = P_0 \tag{2.5}$$

where:

$\hat{x}_0$	$\in$	$\mathcal{R}^{nx}$	is mean of initial state vector
$\tilde{x}_0$	$\in$	$\mathcal{R}^{nx}$	is initial error vector
P_0	$\in$	$\mathcal{R}^{nx \times nx}$	is error covariance matrix of $\hat{x}_0$

Remark: P_0 is merely required to be positive semidefinite instead of being positive definite (pd). This permits the case where some of the initial states are known precisely. Note also that the trace of P is related to the length of the error vector $\tilde{x}$, as:

$$\begin{aligned} trace\,[P] &= E\left\{(\tilde{x}^1)^2 + (\tilde{x}^2)^2 + \cdots + (\tilde{x}^{nx})^2\right\} \\ &= E\left\{\|\tilde{x}\|^2\right\} \end{aligned} \tag{2.6}$$

[7]What exactly is meant with *optimal estimate* will be explained at a later point in this section.

where $\tilde{x}^i$ denotes the ith element of $\tilde{x}$ and $\|\tilde{x}\|$ stands for the length of vector $\tilde{x}$.

Further assumptions are that the noise processes $\{v_k\}$, $\{e_k\}$ are white, Gaussian, zero mean sequences with statistics:

$$E\{v_k\} = \underset{\sim}{0} \tag{2.7}$$

$$E\{e_k\} = \underset{\sim}{0} \tag{2.8}$$

$$E\left\{v_k v_j^T\right\} = Q_k^v \, \delta_{kj} \tag{2.9}$$

$$E\left\{e_k e_j^T\right\} = Q_k^e \, \delta_{kj} \tag{2.10}$$

where:

Q_k^v	is a symmetric positive semidefinite matrix
Q_k^e	is a symmetric positive definite (pd) matrix
δ_{kj}	is 1 for $k = j$ and is 0 otherwise

The initial state and the two noise sequences are uncorrelated:

$$E\left\{x_0 v_k^T\right\} = \underset{\sim}{0} \tag{2.11}$$

$$E\left\{x_0 e_k^T\right\} = \underset{\sim}{0} \tag{2.12}$$

$$E\left\{v_j e_k^T\right\} = \underset{\sim}{0} \tag{2.13}$$

Some of the assumptions given here, can be relaxed, and an optimal linear estimator can still be derived to solve the estimation problem. However, the above problem formulation is considered is general enough for the scope of this thesis.

The filtering problem is to find the optimal estimate $\hat{x}_{k|k}$ for the state vector x_k, taking into account all the available measurements Z. The estimate $\hat{x}_{k|k}$[8] is defined as:

$$\begin{aligned} \hat{x}_{k|k} &= E\{\, x_k \mid Z_k \} \\ &= \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} x_k \, p_{x_k|Z_k} \, dx_k^1 \cdots dx_k^{nx} \end{aligned} \tag{2.14}$$

where:

[8]This type of estimate is known as maximum a posteriori (MAP) estimate

$Z_k = \{z_1, z_2, \ldots, z_k\}$	is the set of all measurements up to time k
$p_{x_k \mid Z_k}$[9]	is the probability density function of x_k conditioned on the knowledge of Z_k
$\hat{x}_{k\mid k}$	is the estimate of x_k where the latest available measurement has already been incorporated. $\hat{x}_{k\mid k}$ is called the measurement-updated estimate.
$\hat{x}_{k\mid k-1}$	is the estimate of x_k at time k where the measurement vector z_k has not yet been considered. $\hat{x}_{k\mid k-1}$ is known as the time-updated estimate.

2.2.2 Kalman Filter Derivation

For the system (2.1), (2.2) is assumed that at time k ($k \geq k_0$) the latest measurement z_k has been processed, i.e. the measurement-updated state estimate $\hat{x}_{k\mid k}$ is available. Assume further that $p_{x_k \mid Z_k}$ is Gaussian with conditional mean $\hat{x}_{k\mid k}$ and conditional error covariance $P_{k\mid k}$.

$$p_{x_k \mid Z_k} = \frac{1}{(2\pi)^{\frac{n_x}{2}} \sqrt{\left|P_{k\mid k}\right|}} EXP\left[-\frac{1}{2}\left(x_k - \hat{x}_{k\mid k}\right)^T P_{k\mid k}^{-1}\left(x_k - \hat{x}_{k\mid k}\right)\right] \quad (2.15)$$

Thus, at time k, the statistic of the state vector x_k is completely known. But what is the value of this information at future time instances i ($i > k$) ? It should be intuitively clear, that for ($i - k$) large, the density $p_{x_k \mid Z_k}$ is of little value for estimation of the current state x_i. Therefore, the conditional probability density function p needs to be propagated, to produce the statistics of the state x at any time $i > k$. The problem to propagate $p_{x_k \mid Z_k}$ to the next time instant $k+1$, can be broken up into two distinct phases:

Phase 1: Time-update $p_{x_k \mid Z_k}$ to obtain the density function $p_{x_{k+1} \mid Z_k}$, the time-updated state estimate $\hat{x}_{k+1\mid k}$ and the time-updated error covariance matrix

[9] In order to make the notation simple, dummy variables are omitted here; e.g. for $p_{x\mid Z}(\alpha \mid Z)$ simply $p_{x\mid Z}$ is written.

$P_{k+1|k}$

Phase 2: Measurement-update $p_{x_{k+1}|Z_k}$, i.e. incorporate z_{k+1} in the conditional density to generate the conditional probability density function $p_{x_{k+1}|Z_{k+1}}$, the measurement-updated state estimate $\hat{x}_{k+1|k+1}$ and covariance matrix $P_{k+1|k+1}$ associated with $\hat{x}_{k+1|k+1}$.

Phase 1: From equation (2.1) one gets the state vector x_{k+1}. The important question that arises here is, whether the density $p_{x_{k+1}|Z_k}$ is still Gaussian. Since x_{k+1} is a linear combination of x_k and and the system noise vector v_k, the conditional density $p_{x_{k+1}|Z_k}$ will be Gaussian, if the conditional density $p_{x_k,v_k|Z_k}$ is Gaussian[10]. By Bayes' rule:

$$\begin{aligned} p_{x_k,v_k|Z_k} &= \frac{p_{x_k,v_k,Z_k}}{p_{Z_k}} \\ &= \frac{p_{x_k,Z_k}\, p_{v_k|x_k,Z_k}}{p_{Z_k}} \end{aligned} \tag{2.16}$$

In the problem statement it is assumed that v_k is independent of x_k and Z_k. Therefore, the conditional probability density function (pdf) $p_{v_k|x_k,Z_k}$ equals the unconditional pdf , p_{v_k}. This allows to rewrite the numerator of equation (2.16):

$$p_{x_k,v_k|Z_k} = \frac{p_{x_k,Z_k}\, p_{v_k}}{p_{Z_k}} \tag{2.17}$$

One further application of Bayes' rule gives:

$$p_{x_k,v_k|Z_k} = p_{x_k|Z_k}\, p_{v_k} \tag{2.18}$$

Both, $p_{x_k|Z_k}$ and p_{v_k} are Gaussian, by assumption. Probability theory states[11] that the product of two Gaussian densities is also Gaussian, provided that the associated random processes are independent. This answers the question stated above: $p_{x_{k+1}|Z_k}$ is a Gaussian density. The mean $\hat{x}_{k+1|k}$ (optimal time-updated estimate) of $p_{x_{k+1}|Z_k}$

[10]See e.g. [43] for proof of this statement

[11]See e.g. [12] or [43] for an introduction in probability theory and stochastic processes

is:

$$\begin{aligned}\hat{x}_{k+1|k} &= E\{x_{k+1} \mid Z_k\} \\ &= E\{(A_k x_k + v_k) \mid Z\} \\ &= A_k\, E\{x_k \mid Z_k\} + E\{v_k \mid Z_k\} \qquad (2.19)\end{aligned}$$

In equation (2.19) $E\{v_k \mid Z_k\}$ is equal to zero because v_k, Z_k are independent and $\{v_k\}$ is a zero mean sequence. Hence, the time-propagated state estimate $\hat{x}_{k+1|k}$ is given by:

$$\hat{x}_{k+1|k} = A_k\, \hat{x}_{k|k} \qquad (2.20)$$

The conditional covariance matrix $P_{k+1|k}$ associated with $\hat{x}_{k+1|k}$ is defined as:

$$P_{k+1|k} = E\left\{\left(x_{k+1} - \hat{x}_{k+1|k}\right)\left(x_{k+1} - \hat{x}_{k+1|k}\right)^T \mid Z_k\right\} \qquad (2.21)$$

If x_{k+1} is replaced by $A_k\, x_k + v_k$ (eqn. 2.1) and $A_k\, \hat{x}_{k|k}$ is used for $\hat{x}_{k+1|k}$ (eqn. 2.20), $P_{k+1|k}$ becomes:

$$\begin{aligned}P_{k+1|k} &= E\left\{\left(A_k x_k + v_k - A_k \hat{x}_{k|k}\right)\left(A_k x_k + v_k - A_k \hat{x}_{k|k}\right)^T \mid Z_k\right\} \\ &= E\left\{\left(A_k\left[x_k - \hat{x}_{k|k}\right] + v_k\right)\left(A_k\left[x_k - \hat{x}_{k|k}\right] + v_k\right)^T \mid Z_k\right\} \\ &= E\left\{\left(A_k\left[x_k - \hat{x}_{k|k}\right] + v_k\right)\left(\left[x_k - \hat{x}_{k|k}\right]^T A_k^T + v_k^T\right) \mid Z_k\right\} \qquad (2.22)\end{aligned}$$

Now, let $\tilde{x}_{k|k}$ be the estimation error vector at time k, based on the set of measurements Z_k. Also, define — as in equation (2.5) — $P_{k|k}$ as the conditional error covariance matrix associated with the time-updated state estimate $\tilde{x}_{k|k}$.

$$\begin{aligned}\tilde{x}_{k|k} &= x_k - \hat{x}_{k|k} \qquad (2.23) \\ P_{k|k} &= E\left\{\tilde{x}_{k|k}\, \tilde{x}_{k|k}^T \mid Z_k\right\} \qquad (2.24)\end{aligned}$$

The two equations above (2.23) and (2.24) allow to rewrite (2.22), as:

$$\begin{aligned} P_{k+1|k} &= E\left\{\left(A_k\,\tilde{x}_{k|k} + v_k\right)\left(\tilde{x}_{k|k}^T\,A_k^T + v_k^T\right)\,\Big|\,Z_k\right\} \\ &= A_k\,E\left\{\tilde{x}_{k|k}\,v_k^T \mid Z_k\right\} + E\left\{v_k\,\tilde{x}_{k|k}^T \mid Z_k\right\}\,A_k^T \\ &\quad + A_k\,E\left\{\tilde{x}_{k|k}\,\tilde{x}_{k|k}^T \mid Z_k\right\}\,A_k^T + E\left\{v_k v_k^T \mid Z_k\right\} \end{aligned} \tag{2.25}$$

The first two terms on the right hand side of equation (2.25) are zero, because v_k is independent of $\tilde{x}_{k|k}$ and Z_k, i.e. its conditional mean equals its unconditional mean, which is zero by assumption (eqn. 2.7). The last term equals Q_k^v (eqn. 2.9) for the same reason. The final expression for $P_{k+1|k}$ is:

$$P_{k+1|k} = A_k\,P_{k|k}\,A_k^T + Q_k^v \tag{2.26}$$

Now that the conditional mean $\hat{x}_{k+1|k}$ and the conditional covariance[12] $P_{k+1|k}$ have been evaluated and that $p_{x_{k+1}|Z_k}$ has been proven to be Gaussian, the conditional density $p_{x_{k+1}|Z_k}$ is completely known and can be expressed by:

$$\begin{aligned} p_{x_{k+1}|Z_k} &= \frac{1}{(2\pi)^{\frac{n_x}{2}}\sqrt{\left|P_{k+1|k}\right|}}\,EXP[\cdot] \\ EXP[\cdot] &= \left[-\frac{1}{2}\left(x_{k+1} - \hat{x}_{k+1|k}\right)^T\,P_{k+1|k}^{-1}\left(x_{k+1} - \hat{x}_{k+1|k}\right)\right] \end{aligned} \tag{2.27}$$

This completes phase 1 of the derivation; the measurement-update problem is treated next.

Phase 2: The next task is to incorporate z_{k+1} in the conditional density $p_{x_{k+1}|Z_k}$ to generate the measurement-updated pdf $p_{x_{k+1}|Z_{k+1}}$. With Bayes' rule as a tool,

[12]From the defining equation for $P_{k+1|k}$ (2.21) it follows that $P_{k+1|k}$ is also the conditional error covariance matrix associated with the error vector $\tilde{x}_{k+1|k} = x_{k+1} - \hat{x}_{k+1|k}$

$p_{x_{k+1}|Z_{k+1}}$ will be expressed in terms of three *easier-to-evaluate* probability density functions.

$$p_{x_{k+1}|Z_{k+1}} = \frac{p_{x_{k+1},Z_{k+1}}}{p_{Z_{k+1}}} \tag{2.28}$$

$$\begin{aligned} p_{x_{k+1},Z_{k+1}} &= p_{x_{k+1},Z_k,z_{k+1}} \\ &= p_{z_{k+1}|x_{k+1},Z_k}\, p_{x_{k+1},Z_k} \\ &= p_{z_{k+1}|x_{k+1},Z_k}\, p_{x_{k+1}|Z_k}\, p_{Z_k} \end{aligned} \tag{2.29}$$

$$\begin{aligned} p_{Z_{k+1}} &= p_{z_{k+1},Z_k} \\ &= p_{z_{k+1}|Z_k}\, p_{Z_k} \end{aligned} \tag{2.30}$$

Substituting (2.29) and (2.30) into (2.28) gives:

$$p_{x_{k+1}|Z_{k+1}} = \frac{p_{z_{k+1}|x_{k+1},Z_k}\, p_{x_{k+1}|Z_k}}{p_{z_{k+1}|Z_k}} \tag{2.31}$$

where $p_{x_{k+1}|Z_k}$ is already known from phase 1 and $p_{z_{k+1}|x_{k+1},Z_k}$ is the pdf of the latest measurement vector z_{k+1} conditioned on knowledge of Z_k and x_{k+1} (!). The fact that the state vector x_{k+1} is assumed to be known, might puzzle the reader and give rise to the question on how x_{k+1} can be assumed to be known, whereas it was stated earlier, that the system states are not directly accessible. The answer to this question is, that the state vector x_{k+1} is assumed to be known only for some intermediate steps , and that in the final filter algorithm x_{k+1} does not appear.

From equation (2.2) it is known that the measurement vector z_{k+1} is given by:

$$z_{k+1} = C_{k+1}\, x_{k+1} + e_{k+1} \tag{2.32}$$

The measurement matrix C_{k+1} and the state vector x_{k+1} are assumed to be known. So, the product $C_{k+1}\, x_{k+1}$ is a fixed vector and therefore z_{k+1} has the same[13] type

[13]The addition of a constant to a random variable does not change the shape of its density function, but merely shifts its mean by that constant.

of distribution as e_{k+1}. The only difference is that the pdf of e_{k+1} is zero mean (by assumption), whereas the mean of z_{k+1} denoted by $z^{\star}_{k+1}$ is $C_{k+1}\, x_{k+1}$. The results of this discussion put in equation form yields:

$$\begin{aligned} z^{\star}_{k+1} &= E\left\{z_{k+1} \mid x_{k+1}, Z_k\right\} \\ &= E\left\{C_{k+1}\, x_{k+1} + e_{k+1} \mid x_{k+1}, Z_k\right\} \\ &= E\left\{C_{k+1}\, x_{k+1} \mid x_{k+1}, Z_k\right\} + E\left\{e_{k+1} \mid x_{k+1}, Z_k\right\} \\ &= C_{k+1}\, x_{k+1} + \underset{\cdot}{0} \end{aligned} \tag{2.33}$$

where the $\underset{\cdot}{0}$ follows from the fact that e_{k+1} is independent of x_{k+1}, Z_k and that $\{e_k\}$ is zero mean. The error covariance matrix $V^{\star}_{k+1}$ associated with $z^{\star}_{k+1}$ is:

$$\begin{aligned} V^{\star}_{k+1} &= E\left\{\left(z_{k+1} - z^{\star}_{k+1}\right)\left(z_{k+1} - z^{\star}_{k+1}\right)^T \mid x_{k+1}, Z_k\right\} \\ &= E\left\{\left(C_{k+1}x_{k+1} + e_{k+1} - C_{k+1}x_{k+1}\right)\left(C_{k+1}x_{k+1} + e_{k+1} - C_{k+1}x_{k+1}\right)^T \mid x_{k+1}, Z_k\right\} \\ &= E\left\{e_{k+1}e^T_{k+1} \mid x_{k+1}, Z_k\right\} \\ &= E\left\{e_{k+1}e^T_{k+1}\right\} \\ &= Q^e_{k+1} \end{aligned} \tag{2.34}$$

As a result of the discussion above $p_{z_{k+1}|x_{k+1},Z_k}$ is known to be Gaussian, so it is exactly described by:

$$\begin{aligned} p_{z_{k+1}|x_{k+1},Z_k} &= \frac{1}{(2\pi)^{\frac{n_y}{2}} \sqrt{\left|Q^e_{k+1}\right|}} EXP\left[\cdot\right] \\ EXP\left[\cdot\right] &= \left[-\frac{1}{2}\left(z_{k+1} - z^{\star}_{k+1}\right)^T \left(Q^e_{k+1}\right)^{-1} \left(z_{k+1} - z^{\star}_{k+1}\right)\right] \end{aligned} \tag{2.35}$$

The last pdf to be evaluated on the right hand side of (eqn. 2.31) is $p_{z_{k+1}|Z_k}$. The measurement vector z_{k+1} is obviously a linear combination of e_{k+1} and x_{k+1} (cf. eqn. 2.32), so $p_{z_{k+1}|Z_k}$ will be Gaussian if the conditional density $p_{e_{k+1},x_{k+1}|Z_k}$ is

Gaussian[14]. By Bayes' rule:

$$\begin{aligned} p_{e_{k+1},x_{k+1}|Z_k} &= \frac{p_{e_{k+1},x_{k+1},Z_k}}{p_{Z_k}} \\ &= \frac{p_{e_{k+1}|x_{k+1},Z_k}\, p_{x_{k+1},Z_k}}{p_{Z_k}} \\ &= p_{e_{k+1}|x_{k+1},Z_k}\, p_{x_{k+1}|Z_k} \end{aligned} \tag{2.36}$$

As stated above, e_{k+1} is independent of x_{k+1} and Z_k. Therefore, its conditional pdf equals its unconditional (Gaussian) pdf. It was proven earlier, that $p_{x_{k+1}|Z_k}$ is also Gaussian. In other words, e_{k+1} and x_{k+1} are — conditioned on the set of measurements Z_k — independent, normally distributed random vectors. Hence, the product of their conditional densities $p_{e_{k+1}|x_{k+1},Z_k}$ and $p_{x_{k+1}|Z_k}$ is a Gaussian density. Let $\hat{z}_{k+1}$ be the mean and V_{k+1} be the variance associated with the conditional density function $p_{z_{k+1}|Z_k}$.

$$\begin{aligned} \hat{z}_{k+1} &= E\left\{z_{k+1} \mid Z_k\right\} \\ &= E\left\{C_{k+1}\, x_{k+1} + e_{k+1} \mid Z_k\right\} \\ &= E\left\{C_{k+1}\, x_{k+1} \mid Z_k\right\} + E\left\{e_{k+1} \mid Z_k\right\} \\ &= C_{k+1}\, E\left\{x_{k+1} \mid Z_k\right\} + E\left\{e_{k+1}\right\} \\ &= C_{k+1}\, \hat{x}_{k+1} \end{aligned} \tag{2.37}$$

$$\begin{aligned} V_{k+1} &= E\left\{\left(z_{k+1} - \hat{z}_{k+1}\right)\left(z_{k+1} - \hat{z}_{k+1}\right)^T \mid Z_k\right\} \\ &= E\left\{\left(C_{k+1}\left[x_{k+1} - \hat{x}_{k+1|k}\right] + e_{k+1}\right)\left(\left[x_{k+1} - \hat{x}_{k+1|k}\right]^T C_{k+1}^T + e_{k+1}^T\right) \mid Z_k\right\} \\ &= C_{k+1}\, E\left\{\left(x_{k+1} - \hat{x}_{k+1|k}\right)\left(\left[x_{k+1} - \hat{x}_{k+1|k}\right)^T \mid Z_k\right\} C_{k+1}^T \\ &\quad + C_{k+1} E\left\{\left(x_{k+1} - \hat{x}_{k+1|k}\right) e_{k+1}^T \mid Z_k\right\} \\ &\quad + E\left\{e_{k+1}\left(x_{k+1} - \hat{x}_{k+1|k}\right)^T \mid Z_k\right\} C_{k+1}^T + E\left\{e_{k+1}\, e_{k+1}^T \mid Z_k\right\} \end{aligned} \tag{2.38}$$

$$\begin{aligned} V_{k+1} &= C_{k+1}\, E\left\{\left(x_{k+1} - \hat{x}_{k+1|k}\right)\left(x_{k+1} - \hat{x}_{k+1|k}\right)^T \mid Z_k\right\} C_{k+1}^T \\ &\quad + C_{k+1}\, E\left\{x_{k+1} e_{k+1}^T \mid Z_k\right\} + E\left\{e_{k+1} x_{k+1}^T \mid Z_k\right\} C_{k+1}^T \\ &\quad - C_{k+1}\, \hat{x}_{k+1|k}\, E\left\{e_{k+1}^T \mid Z_k\right\} - E\left\{e_{k+1}^T \mid Z_k\right\} \hat{x}_{k+1|k}^T\, C_{k+1}^T \end{aligned}$$

[14] Notice the similarity to the evaluations made in section **phase 1** on page 13.

$$+E\left\{e_{k+1}e_{k+1}^T \mid Z_k\right\} \tag{2.39}$$

The first term on the right hand side of equation (2.39) above is recognized[15] to be $C_{k+1}P_{k+1|k}C_{k+1}^T$. As mentioned earlier, e_{k+1} is independent of Z_k, x_{k+1} and $\{e_k\}$ is zero mean. Therefore, the terms 2, 3, 4 and 5 are all zero. The last term in (2.39) equals Q_{k+1}^e. The measurement covariance matrix V_{k+1} can now be written as:

$$V_{k+1} = C_{k+1}P_{k+1|k}C_{k+1}^T + Q_{k+1}^e \tag{2.40}$$

The Gaussian density is explicitly given by:

$$\begin{aligned} p_{z_{k+1}|Z_k} &= \frac{1}{(2\pi)^{\frac{ny}{2}}\sqrt{\left|C_{k+1}P_{k+1|k}C_{k+1}^T + Q_{k+1}^e\right|}} EXP[\cdot] \\ EXP[\cdot] &= \left[-\frac{1}{2}\left(z_{k+1} - \hat{z}_{k+1}\right)^T \left(C_{k+1}P_{k+1|k}C_{k+1}^T + Q_{k+1}^e\right)^{-1}\left(z_{k+1} - \hat{z}_{k+1}\right)\right] \end{aligned} \tag{2.41}$$

At this point, all densities on the right hand side of equation (2.31) are evaluated and the density $p_{x_{k+1}|Z_{k+1}}$ can now be expressed by:

$$\begin{aligned} p_{x_{k+1}|Z_{k+1}} &= \frac{\sqrt{\left|P_{k+1|k}^{-1}\left(C_{k+1}P_{k+1|k}C_{k+1}^T + Q_{k+1}^e\right)\left(Q_{k+1}^e\right)^{-1}\right|}}{(2\pi)^{\frac{ny}{2}}} EXP[\cdot] \\ EXP[\cdot] &= \left[-\frac{1}{2}\left(x_{k+1} - \hat{x}_{k+1|k}\right)^T P_{k+1|k}^{-1}\left(x_{k+1} - \hat{x}_{k+1|k}\right)\right. \\ &\quad + \left(z_{k+1} - C_{k+1}x_{k+1}\right)^T \left(Q_{k+1}^e\right)^{-1}\left(z_{k+1} - C_{k+1}x_{k+1}\right) \\ &\quad \left. - \left(z_{k+1} - \hat{z}_{k+1}\right)^T \left(C_{k+1}P_{k+1|k}C_{k+1}^T + Q_{k+1}^e\right)^{-1}\left(z_{k+1} - \hat{z}_{k+1}\right)\right] \end{aligned} \tag{2.42}$$

It is possible to transform equation (2.42) into a quadratic standard form for Gaussian densities, from which it can be concluded that $p_{x_{k+1}|Z_{k+1}}$ indeed is Gaussian. Because this step is tedious and requires intensive use of algebra, it is left out and just the resulting equation is given here. The interested reader is referred to

[15] Refer to eqn. (2.21)

Maybeck ([39] page 213ff).

$$p_{x_{k+1}|Z_{k+1}} = \frac{1}{(2\pi)^{\frac{nx}{2}}\sqrt{\left|P_{k+1|k+1}\right|}} EXP[\cdot] \tag{2.43}$$

$$EXP[\cdot] = \left[-\frac{1}{2}\left(x_{k+1}-\hat{x}_{k+1|k+1}\right)^T P_{k+1|k+1}^{-1}\left(x_{k+1}-\hat{x}_{k+1|k+1}\right)\right]$$

where $P_{k+1|k+1}$ is the measurement updated covariance matrix, which is calculated as:

$$\begin{aligned} P_{k+1|k+1} &= E\left\{\left(x_{k+1}-\hat{x}_{k+1|k+1}\right)\left(x_{k+1}-\hat{x}_{k+1|k+1}\right)^T \mid Z_{k+1}\right\} \\ &= \left[C_{k+1}^T\left(Q_{k+1}^e\right)^{-1} C_{k+1} + P_{k+1|k}^{-1}\right]^{-1} \end{aligned} \tag{2.44}$$

and the mean (measurement updated state estimate) $\hat{x}_{k+1|k+1}$ is given by:

$$\begin{aligned} \hat{x}_{k+1|k+1} &= E\left\{x_{k+1} \mid Z_{k+1}\right\} \\ &= \left[C_{k+1}^T\left(Q_{k+1}^e\right)^{-1} C_{k+1} + P_{k+1|k}^{-1}\right]^{-1}\left[C_{k+1}^T\left(Q_{k+1}^e\right)^{-1} z_{k+1} + P_{k+1|k}^{-1}\hat{x}_{k+1|k}\right] \end{aligned} \tag{2.45}$$

The equations derived so fare solve the estimation problem as given in section *Problem Statement* and one could stop here. However, the KF algorithm in this form is not the most efficient one. The reason for this is, that the equations above require inversions of nx-by-nx[16] matrices, whereas it is possible to transform the equations (2.44) and (2.45) into equivalent representations that require only inversions of ny-by-ny[17] matrices. This yields an algorithm that is computationally less costly, because for most systems is the number of states larger than the number of outputs. The *Matrix Inversion Lemma*[18] (eqn. 2.46) shall be used to convert (2.44) and (2.45) into the desired forms:

$$\left(C^TQ^{-1}C + P^{-1}\right)^{-1} = P - PC^T\left(CPC^T + Q\right)^{-1}CP \tag{2.46}$$

[16] nx is the number of states

[17] ny is the number of outputs

[18] See Appendix F for proof of this lemma

Substituting (2.44) into (2.46) immediately yields the proper expression for the time-updated error covariance matrix $P_{k+1|k+1}$:

$$P_{k+1|k+1} = P_{k+1|k} - P_{k+1|k} C^T_{k+1} \left(C_{k+1} P_{k+1|k} C^T_{k+1} + Q^e_{k+1}\right)^{-1} C_{k+1} P_{k+1|k} \tag{2.47}$$

Next, (2.46) is substituted into (2.45), which gives:[19]

$$\begin{aligned} \hat{x}_{k+1|k+1} &= \left[P - PC^T\left(CPC^T + Q_e\right)^{-1} CP\right]\left[P^{-1}\hat{x} + C^T Q_e^{-1} z\right] \qquad (2.48)\\ &= \hat{x} - PC^T\left(CPC^T + Q_e\right)^{-1} C\hat{x} \\ &\quad + PC^T\left[Q_e^{-1} - \left(CPC^T + Q_e\right)^{-1} CPC^T Q_e^{-1}\right] z \\ &= \hat{x} - PC^T\left(CPC^T + Q_e\right)^{-1} C\hat{x} \\ &\quad + PC^T\left(CPC^T + Q_e\right)^{-1}\left[\left(CPC^T + Q_e\right) Q_e^{-1} - CPC^T Q_e^{-1}\right] z \\ &= \hat{x} - PC^T\left(CPC^T + Q_e\right)^{-1} C\hat{x} + PC^T\left(CPC^T + Q_e\right)^{-1} [I]\, z \end{aligned}$$

where I denotes the identity matrix. Factoring out $PC^T\left(CPC^T + Q_e\right)^{-1}$ and reintroducing the time indices yields:

$$\begin{aligned} \hat{x}_{k+1|k+1} &= \hat{x}_{k+1|k} + P_{k+1|k} C_{k+1}\left(C_{k+1} P_{k+1|k} C^T_{k+1} + Q^e_{k+1}\right)^{-1} \qquad (2.49)\\ &\quad \times \left(z_{k+1} - C_{k+1} x_{k+1|k}\right) \end{aligned}$$

Exploiting common terms in (2.47) and (2.49) to further simplify the algorithm gives:

$$K^F_{k+1} = P_{k+1|k} C^T_{k+1}\left(C_{k+1} P_{k+1|k} C^T_{k+1} + Q^e_{k+1}\right)^{-1} \tag{2.50}$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K^F_{k+1}\left(z_{k+1} - C_{k+1}\hat{x}_{k+1|k}\right) \tag{2.51}$$

$$P_{k+1|k+1} = \left(I - K^F_{k+1} C_{k+1}\right) P_{k+1|k} \tag{2.52}$$

[19] In order to keep the algebra tractable, time indices on the right hand side will be dropped for this step. It is understood, that C means C_{k+1}, P means $P_{k+1|k}$, z means z_{k+1}, $\hat{x}$ means $\hat{x}_{k+1|k}$ and Q_e means Q^e_{k+1}

where K^F_{k+1} is the Kalman Filter gain matrix at time $k+1$.

What has been shown so far, is how to propagate $p_{x_k|Z_k}$ from time instant k to time instant $k+1$ and how to incorporate the new measurements, that become available at time $k+1$, into the density to generate $p_{x_{k+1}|Z_{k+1}}$. It has been proven that if $p_{x_k|Z_k}$ is Gaussian, so is $p_{x_{k+1}|Z_{k+1}}$. This is true for any k $(k \geq k_0)$, in particular for $k = k_0$. Because $p_{x_0|Z_0}$ is assumed to be Gaussian, all $p_{x_i|Z_i}$ $(i > k_0)$ will be Gaussian, which completes the proof.

The state estimate, generated by the derived algorithm, is optimal in many ways. One criteria of optimallity, that is of special interest for the developments on the EKF, is the minimum variance (MV) criteria. It shall be shown that $\hat{x}$ is also a minimum variance estimate.

Recall that the error covariance matrix is defined as:

$$\begin{aligned} P &= E\left\{\tilde{x}\tilde{x}^T\right\} \\ &= \int_{-\infty}^{+\infty}\cdots\int_{-\infty}^{+\infty} (x-\eta)(x-\eta)^T p_{x|Z}\, dx_k^1 \cdots dx_k^{nx} \end{aligned} \tag{2.53}$$

where η is the minimum variance state estimate. As expressed by equation (2.6), the square root of $trace[P]$ equals the expected length of the error vector $(x - \eta)$. Thus, minimizing the trace of P leads to an estimate, that is best in a mean square sense. The problem is to identify this η that minimizes $trace[P]$. Let $J = trace[P]$ and consider J as a function of η. As usual, the minimum of J is found by setting the derivative of J, with respect to η, to zero (necessary condition).

$$\begin{aligned} \frac{\partial J}{\partial \eta} &= \frac{\partial}{\partial \eta}\left[trace\left[P\right]\right] \\ &= \frac{\partial}{\partial \eta}\left[trace\left[\int_{-\infty}^{+\infty}\cdots\int_{-\infty}^{+\infty} (x-\eta)(x-\eta)^T p_{x|Z}\, dx_k^1 \cdots dx_k^{nx}\right]\right] \\ &= \frac{\partial}{\partial \eta}\left[\int_{-\infty}^{+\infty}\cdots\int_{-\infty}^{+\infty} (x-\eta)^T (x-\eta)\, p_{x|Z}\, dx_k^1 \cdots dx_k^{nx}\right] \end{aligned} \tag{2.54}$$

Exchanging the order of integration and differentiation in equation (2.54) yields:

$$\frac{\partial J}{\partial \eta} = \int_{-\infty}^{+\infty}\cdots\int_{-\infty}^{+\infty} \frac{\partial}{\partial \eta}\left[(x-\eta)^T (x-\eta)\right] p_{x|Z}\, dx_k^1 \cdots dx_k^{nx}$$

$$
\begin{aligned}
&= \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} \frac{\partial}{\partial \eta} \left[x^T x - \eta^T x - x^T \eta + \eta^T \eta \right] p_{x|Z} \, dx_k^1 \cdots dx_k^{nx} \\
&= \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} [0 - x - x + 2\eta] \, p_{x|Z} \, dx_k^1 \cdots dx_k^{nx} \\
&= 2\eta - 2 \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} x \, p_{x|Z} \, dx_k^1 \cdots dx_k^{nx} \qquad (2.55)
\end{aligned}
$$

The last line of (2.55) clearly shows that the gradient $\frac{\partial J}{\partial \eta}$ is only zero if η equals the conditional mean, i.e. equals $\hat{x}$. Hence, the KF state estimate is also the minimum variance estimate[20]. Note that for the MV estimate, no assumption about the nature of the conditional probability density function $p_{x|Z}$ were made; it is sufficient to know its mean.

It should be pointed out, that the covariance matrix update equations (2.26) and (2.47) do not depend on the actual measurements z_i. Therefore, covariance matrix P and Kalman gain matrix K can be computed prior to the actual application time. The necessary on-line computations are, for finite-time processes, reducible to just updating the state estimate.

2.2.3 KF for Systems with Deterministic Inputs

The KF algorithm derived above does not account for known inputs. Now it will be shown that only little changes in the algorithm are necessary to make the KF applicable for the broad class of systems with known inputs.

The system that generates the set of measurement vectors Z_k is adequately described by the following state space equations:

$$
\begin{aligned}
x_{k+1} &= A_k x_k + B_k u_k + v_k \qquad (2.56) \\
z_k &= y_k + e_k \\
&= C_k x_k + e_k \qquad (2.57)
\end{aligned}
$$

where B_k is a nx-by-nu input distribution matrix and u_k is the known input vector

[20]To complete the proof one would also have to show that the sufficient condition is also satisfied, i.e. the Hessian of J is positive semidefinite. It can readily be shown that this holds for $\frac{\partial^2 J}{\partial \eta^2}$.

of dimension nu. All assumptions about initial state, noise sequences etc. are as before (see equations 2.3-2.13).

The difference between (2.1) and (2.56) is that in the latter equation a known vector $B_k u_k$ is added to x_{k+1}. It is stated earlier (see footnote 13 on page 16) that this changes only the mean of the state vector x_{k+1} but not the shape of its density function. So the covariance matrices associated with x_{k+1} are not affected. Therefore, it is sufficient to consider only the estimate update equations here and derive how they have to be changed to account for known inputs. The optimal time-updated state estimate $\hat{x}_{k+1|k}$ for the system as described by (2.56) and (2.57), is calculated as:

$$\begin{aligned}
\hat{x}_{k+1|k} &= E\left\{x_{k+1} \mid Z_k\right\} \\
&= E\left\{(A_k x_k + B_k u_k + v_k) \mid Z\right\} \\
&= A_k\, E\left\{x_k \mid Z_k\right\} + B_k u_k + E\left\{v_k \mid Z_k\right\} \\
&= A_k\, \hat{x}_{k|k} + B_k\, u_k \qquad (2.58)
\end{aligned}$$

where x_{k+1} has been substituted by (2.56). The next question one has to ask is, whether the measurement-update equation (2.51) needs to be changed.

$$\hat{x}_{k+1|k+1} - \hat{x}_{k+1|k} = K^F_{k+1}\left(z_{k+1} - C_{k+1}\hat{x}_{k+1|k}\right) \qquad (2.59)$$

where the right hand side is the updating vector by which $\hat{x}$ is corrected, when the new measurements z_{k+1} become available. Substituting (2.56), (2.57) and (2.58) into equation (2.59) yields:

$$\begin{aligned}
x_{k+1|k+1} - x_{k+1|k} &= K^F_{k+1}\,\big(C_{k+1}A_k x_k + C_{k+1}B_k u_k + C_{k+1}v_k + e_{k+1} \\
&\qquad - C_{k+1}A_k\hat{x}_{k|k} + C_{k+1}B_k u_k\big) \\
&= K^F_{k+1}\left(C_{k+1}\left(A_k x_k - A_k\hat{x}_{k|k}\right) + e_{k+1}\right) \qquad (2.60)
\end{aligned}$$

So clearly, a deterministic input has no effect on the measurement update. The

Table 2.1: Summary of the 2 phase Kalman filter algorithm

2 Phase^{21}KF Algorithm

Phase 1: time-update equations

$$\hat{x}_{k+1|k} = A_k \hat{x}_{k|k} + B_k u_k \tag{2.61}$$
$$P_{k+1|k} = A_k P_{k|k} A_k^T + Q_k^v \tag{2.62}$$

Phase 2: measurement-update equations

$$K_{k+1}^F = P_{k+1|k} C_{k+1}^T \left(C_{k+1} P_{k+1|k} C_{k+1}^T + Q_{k+1}^e\right)^{-1} \tag{2.63}$$
$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}^F \left(z_{k+1} - C_{k+1} \hat{x}_{k+1|k}\right) \tag{2.64}$$
$$P_{k+1|k+1} = \left(I - K_{k+1}^F C_{k+1}\right) P_{k+1|k} \tag{2.65}$$

KF algorithm is suited for system with deterministic input, if equation (2.20) is replaced by equation (2.58).

2.2.4 Summary

To show the order, in which the individual equations of the derived KF algorithm has to be computed, a summary of the algorithm is given in Table 2.1.

Phase 1 and phase 2 have to be executed recursively; with phase 1 the filter "jumps" in time from time instant k to time instant $k+1$, where phase 2 is performed, as soon as z_{k+1} becomes available and so on.

There are situations, where it suffices to singly compute either the time-updated state estimate $\hat{x}_{k|k-1}$, or the measurement-updated state estimate $\hat{x}_{k|k}$. For these cases, it is desirable to convert the 2 phase KF algorithm into a single phase algorithm, because the latter one is computationally more efficient. Ljung choose for his paper [33] on the asymptotic behavior of the EKF, a filter algorithm that generates only the time-updated estimates. So, in preparation of the evaluations about [33], it is shown next how the derived KF algorithm can be converted into a single phase algorithm, which computes the time-updated state estimate $\hat{x}_{k|k-1}$ and the with this estimate associated covariance matrix $P_{k|k-1}$. To start this transformation, substitute equation (2.64) for $\hat{x}_{k|k}$ into (2.61):

$$\begin{aligned} \hat{x}_{k+1|k} &= A_k \left(\hat{x}_{k|k-1} + K_k^F \left(z_k - C_k\hat{x}_{k|k-1}\right)\right) + B_k u_k \\ &= A_k\hat{x}_{k|k-1} + A_k K_k^F \left(z_k - C_k\hat{x}_{k|k-1}\right) + B_k u_k \end{aligned} \tag{2.66}$$

Let:

$$S_k{}^{22} = C_k P_{k|k-1} C_k^T + Q_k^e \tag{2.67}$$

and redefine the Kalman gain matrix K_k to be:

$$\begin{aligned} K_k &= A_k P_{k|k-1}\, C_k^T \left(C_k P_{k|k-1} C_k^T + Q_{k+1}^e\right)^{-1} \\ &= A_k P_{k|k-1}\, C_k^T S_k^{-1} \end{aligned} \tag{2.68}$$

With (2.67) and (2.68) equation (2.66) can be rewritten:

$$\hat{x}_{k+1|k} = A_k\hat{x}_{k|k-1} + B_k u_k + K_k \left(z_k - C_k\hat{x}_{k|k-1}\right) \tag{2.69}$$

To get a similar expression for the covariance matrix, substitute (2.65) and (2.63) into equation (2.62) which yields:

$$\begin{aligned} P_{k+1|k} &= A_k P_{k|k} \left(I - K_k^F C_k\right) P_{k|k-1}\, A_k^T + Q_k^v \\ &= A_k P_{k|k}\, A_k^T - K_k C_k P_{k|k-1}\, A_k^T + Q_k^v \\ &= A_k P_{k|k}\, A_k^T - K_k S_k S_k^{-1} C_k P_{k|k-1}\, A_k^T + Q_k^v \\ &= A_k P_{k|k}\, A_k^T - K_k S_k \left(A_k P_{k|k-1}\, C_k^T S_k^{-1}\right)^T + Q_k^v \\ &= A_k P_{k|k}\, A_k^T - K_k S_k K_k^T + Q_k^v \end{aligned} \tag{2.70}$$

[21] *2 Phase* because the propagation of conditional mean $\hat{x}$ and covariance P is broken up into two phases, time-update and measurement-update

[22] S_k is pd, so its inverse exists

For further reference, the complete single phase KF algorithm is summarized in Table 2.2

Table 2.2: Summary of the single phase Kalman filter algorithm

Single Phase KF Algorithm	
Filter equations:	
$S_k = C_k P_{k\|k-1} C_k^T + Q_k^e$	(2.71)
$K_k = A_k P_{k\|k-1} C_k^T S_k^{-1}$	(2.72)
$\hat{x}_{k+1\|k} = A_k \hat{x}_{k\|k-1} + B_k u_k + K_k \left(z_k - C_k \hat{x}_{k\|k-1} \right)$	(2.73)
$P_{k+1\|k} = A_k P_{k\|k-1} A_k^T - K_k S_k K_k^T + Q_k^v$	(2.74)

One of the requirements for successful application of the KF as a state estimator is that the system has to be known completely. In practical cases however, the system is often not fully known from the very beginning of operation, or it is time-varying in an unpredictable way, caused e.g. through wear of some parts. These unknowns about the system can be included in the model as parameters, which can be regarded as random variables with known a priori statistics. Assume there are np such parameters combined into a vector θ. In general all system matrices are dependent on θ. If A, B and C are otherwise time-invariant, one gets the following model description:

$$x_{k+1} = A(\theta)x_k + B(\theta)u_k + v_k \tag{2.75}$$

$$z_k = C(\theta)x_k + e_k \tag{2.76}$$

The states of the system and these parameters are both not directly accessible. Hence, an obvious thing to do, is to extend the state vector x by the parameter vector θ to form an augmented state vector, denoted x^A.

$$x_k^A = \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} \tag{2.77}$$

As before, the filtering problem is to estimate the (augmented) state vector. The question that arises here is: Can one still use the KF to attack this filtering problem? The state equation for the augmented system as given below, yields the answer to this question.

$$x_{k+1}^A = f\left(x_k^A, u_k\right) + \begin{bmatrix} v_k \\ 0 \end{bmatrix} \tag{2.78}$$

$$z_k = h\left(x_k^A\right) + e_k \tag{2.79}$$

where:

$$f\left(x_k^A, u_k\right) = \begin{bmatrix} A(\theta_k)\, x_k + B(\theta_k)\, u_k \\ \theta \end{bmatrix} \tag{2.80}$$

$$h\left(x_k^A\right) = C(\theta_k)\, x_k \tag{2.81}$$

From the equation (2.80) it is obvious that the system is not linear in x^A (x^A can not be factored out). Hence, the KF is not the proper tool to be applied for parameter estimation, because it does not account for nonlinearities.

2.2.5 Conclusion

Two Kalman filter algorithms suitable for estimating the state of linear discrete-time multiple-input/multiple-output systems have been derived.

The filters are not the most general ones possible, but sufficient for the needs in this work. If necessary, the Kalman filter algorithms can be modified to include cases where measurement and process noise are correlated, where the noise sequences are non-zero mean (biased) or where some of the measurements are noise free. Kalman filtering yields an state estimate, that is — based on the assumptions made — satisfying many optimality criteria. In real applications the initial state and the initial error covariance matrix are generally not known. However, the algorithm will still converge to its 'best' for some large $(k - k_0)$.

It has been shown that the KF is not suited for parameter estimation, as this is inherently a nonlinear filtering problem. An extension of the KF, known as the Extended Kalman Filter (EKF) can be used instead. This filter is described in the next section.

2.3 The Discrete-Time Extended Kalman Filter

The Kalman filter, introduced in the previous section is an optimal state estimator for linear systems that are completely known. Real systems however, are often nonlinear and/or there are some uncertainties about it. In case the system under consideration belongs to this group, there are two ways to proceed:

- using nonlinear filter theory (which provides solutions only for some special cases)
- linearize (approximate) the problem and then apply linear filter theory.

Extended Kalman filtering is based on the latter method. At each step, the state equation has to be linearized, which is done by expanding it into a Taylor series, evaluated about an state estimate and truncated after the linear term. Hence the extended Kalman filter is an approximate filter, based on first order linearization.

Better approximations are achieved by including higher order terms in the expansion. Filters, based on this method, where in the Taylor series expansions the quadratic terms are included, are referred to as *2nd Order Extended Kalman Filter*. Another method[23] to get better state estimates $\hat{x}_k$ is to repeatedly calculate $\hat{x}_k$, K_k and P_k each time about the most recent estimate.

Good descriptions of the extended Kalman filter and the other methods mentioned here are e.g. given in [14],[25] and [40].

[23]This method is known as Iterated Extended Kalman Filter.

2.3.1 Problem Statement

Assume that the nonlinear time-invariant[24] system, from which the measured output data are obtained, is adequately described by the following state space equations:

$$x_{k+1} = f(x_k, u_k) + v_k \tag{2.82}$$

$$z_k = h(x_k) + e_k \tag{2.83}$$

where:

k	$\geq$	k_0	k_0 is the initial time instant
x_k	$\in$	$\mathcal{R}^{nx}$	is the system state vector
u_k	$\in$	$\mathcal{R}^{nx}$	is the (known) input vector
z_k	$\in$	$\mathcal{R}^{ny}$	is the measurement vector
v_k	$\in$	$\mathcal{R}^{nx}$	is the process noise vector
e_k	$\in$	$\mathcal{R}^{ny}$	is the measurement noise vector
$f(\cdot)$	$\in$	$\mathcal{R}^{nx \times nx}$	is the in x nonlinear system matrix
$h(\cdot)$	$\in$	$\mathcal{R}^{ny \times nx}$	is the in x nonlinear measurement matrix

$f(\cdot)$ and $h(\cdot)$ are fully known. x_k, z_k, u_k, e_k and v_k are random vectors of which only z_k and u_k are known. For this derivation, it is assumed that there is a deterministic input signal present.[25] As in the case for the KF, some assumptions about initial state and the noise sequences involved have to be made, such that extended Kalman filtering becomes applicable.

A 1: Mean and covariance of the initial state vector x_0 are known

A 2: The two noise sequences $\{v_k\}$ and $\{e_k\}$ are white Gaussian, zero mean sequences with statistics:

$$E\{v_k\} = 0 \tag{2.84}$$

$$E\{e_k\} = 0 \tag{2.85}$$

$$E\left\{v_k v_j^T\right\} = Q_k^v \, \delta_{kj} \tag{2.86}$$

$$E\left\{e_k e_j^T\right\} = Q_k^e \, \delta_{kj} \tag{2.87}$$

[24]This will be relaxed later on to the case of systems with are slowly time-varying

[25]Any accessible input signals belong to this group, e.g. known disturbances, control inputs etc.

where:

Q_k^v	is a symmetric positive semidefinite matrix
Q_k^e	is a symmetric positive definite (pd) matrix
δ_{kj}	is 1 for $k = j$ and is 0 otherwise

A 3: Initial state and noise sequences are uncorrelated:

$$E\left\{x_0 v_k^T\right\} \equiv 0 \tag{2.88}$$

$$E\left\{x_0 e_k^T\right\} = \underline{0} \tag{2.89}$$

$$E\left\{v_j e_k^T\right\} \equiv \underline{0} \tag{2.90}$$

The filtering problem is to deduce estimates for the states of the system, utilizing all the information available, which includes state space equations, noise statistics and initial conditions. Because extended Kalman filtering is, as noted earlier, an approximate method, the resulting state estimates are no longer *optimal*, but the *best* one can get applying a linear estimator.

2.3.2 EKF Derivation

There are many different ways to derive the EKF algorithm. Here, the author outlines two different approaches: the first one is less rigorous and primarily shows how through linearization, the problem can be converted in an approximately linear filtering problem. This linear problem can then be solved using the KF. In the second approach it is shown how the three constraints:

1.) that the estimate has to be unbiased
2.) that the estimation variance is minimal
3.) that the estimator is linear

lead to an algorithm, that is similar to the KF algorithm.

First Approach:

Assume the system under consideration is at time k ($k \geq k_0$) and that the latest measurement z_k has already been processed, so that estimate $\hat{x}_{k|k}$ and conditional error covariance matrix $P_{k|k}$ are available. Based on this, the first step is to generate the time-updated state estimate $\hat{x}_{k+1|k}$ and the covariance matrix associated with this estimate. Recall from the Kalman filter derivation, that the optimal time-updated estimate for x_{k+1} is the mean of $p_{x_{k+1}|Z_k}$. Hence, $\hat{x}_{k+1|k}$ is given by:

$$\begin{aligned}\hat{x}_{k+1|k} &= E\{x_{k+1} \mid Z_k\} \\ &= E\{f(x_k, u_k) + v_k \mid Z_k\} \\ &= E\{f(x_k, u_k) \mid Z_k\} + E\{v_k \mid Z_k\} \end{aligned} \tag{2.91}$$

where $E\{v_k \mid Z_k\}$ is zero, because $\{v_k\}$ is a zero mean sequence and v_k is independent of Z_k. So, $\hat{x}_{k+1|k}$ can be rewritten as:

$$\hat{x}_{k+1|k} = E\{f(x_k, u_k) \mid Z_k\} \tag{2.92}$$

To solve equation (2.92) precisely, one would have to know $p_{x_{k+1}|Z_k}$, but there is little justification to assume $p_{x_{k+1}|Z_k}$ to be Gaussian; even if x_i would be normally distributed, x_{i+1} would not, because $f(\cdot)$ is nonlinear in x. A practical thing to do, is to expand $f(x_k, u_k)$ in a Taylor series about a vector, say $\bar{x}$, truncated after the linear[26] term. The quality of this approximate method depends to a great extend on how close, in a mean square sense, $\bar{x}$ is to the state vector x. But how can such a vector $\bar{x}$ be generated? All the information available about x is represented by $\hat{x}$, which is the *best guess* for x. So naturally, the Taylor series expansion of $f(x_k, u_k)$ is done about $\hat{x}_{k|k}$, which is (hopefully) so close to x_k, that the truncation error is small with respect to remaining terms. Therefore, $\hat{x}_{k+1|k}$ can be computed as:

$$\hat{x}_{k+1|k} = E\left\{ f(\hat{x}_{k|k}, u_k) + \left.\frac{\partial f}{\partial x_k}\right|_{x_k = \hat{x}_{k|k}} (x_k - \hat{x}_{k|k}) + HOT \,\middle|\, Z_k \right\} \tag{2.93}$$

$$= f(\hat{x}_{k|k}, u_k) + E\left\{F(\hat{x}_{k|k}, u_k)\left(x_k - \hat{x}_{k|k}\right) + HOT \mid Z_k\right\} \tag{2.94}$$

where $F(\hat{x}_{k|k}, u_k)$ is the nx-by-nx matrix of partial derivatives, evaluated along $\hat{x}_{k|k}$. Neglecting the higher order terms (HOT) and assuming that the state estimate is

[26]This is motivated by the idea to apply linear estimation theory to this class of filtering problem.

unbiased[27] ($E\{x - \hat{x}\} = \underline{0}$), equation (2.94) simply becomes:

$$\hat{x}_{k+1|k} \cong f(\hat{x}_{k|k}, u_k) \tag{2.95}$$

Let $\tilde{x}_{k+1|k}$ be the time-updated error vector associated with $\hat{x}_{k+1|k}$, and $\tilde{x}_{k|k}$ be the measurement-updated error vector:

$$\begin{aligned} \tilde{x}_{k+1|k} &= x_{k+1} - \hat{x}_{k+1|k} && (2.96) \\ \tilde{x}_{k|k} &= x_k - \hat{x}_{k|k} && (2.97) \end{aligned}$$

Substituting (2.97), (2.95) and (2.82) into equation (2.96) yields:

$$\begin{aligned} \tilde{x}_{k+1|k} &= \Big(f(\hat{x}_{k|k}, u_k) + F(\hat{x}_{k|k}, u_k)(x_k - \hat{x}_{k|k}) + v_k\Big) - f(\hat{x}_{k|k}, u_k) \\ &= F(\hat{x}_{k|k}, u_k)\, \tilde{x}_{k|k} + v_k && (2.98) \end{aligned}$$

where $f(x_{k|k}, u_k)$ is replaced by the truncated Taylor series as given in (2.93). It is interesting to note, that equation (2.98) is linear in $\tilde{x}$. The vector $\tilde{x}$ is random and can — under the given assumptions[28] — still assumed to be (approximately) Gaussian distributed. Substituting (2.96) and (2.97) into equation (2.98) yields another interesting relation:

$$\begin{aligned} x_{k+1} - \hat{x}_{k+1|k} &= F(\hat{x}_{k|k}, u_k)\,(x_k - \hat{x}_{k|k}) + v_k \\ x_{k+1} &= \hat{x}_{k+1|k} + F(\hat{x}_{k|k}, u_k)\,(x_k - \hat{x}_{k|k}) + v_k \\ &= + F(\hat{x}_{k|k}, u_k)\,\hat{x}_{k|k} + F(\hat{x}_{k|k}, u_k)\,(x_k - \hat{x}_{k|k}) + v_k \\ &= F(\hat{x}_{k|k}, u_k)\, x_k + v_k && (2.99) \end{aligned}$$

which is identical with equation (2.1), except that A_k is replaced by $F(\hat{x}_k, u_k)$. The question that arises here is, whether applying the linearization method to the measurement equation (2.83) will result in an equation that is of the same type as (2.1), because then the problem could be solved with the KF. To get a linearized

[27]This assumption will be verified later on.

[28]Particularly that $\hat{x}$ is a good estimate of x, which is only true if all $\mid v^i \mid$ are small.

measurement equation, expand $h(x_k)$ into a Taylor series evaluated along the time-updated state estimate $\hat{x}_k$.

$$h(x_k) = h(\hat{x}_{k|k-1}) + \left.\frac{\partial f}{\partial x_k}\right|_{x_k=\hat{x}_{k|k-1}} \left(x_k - \hat{x}_{k|k-1}\right) + HOT \tag{2.100}$$

The higher order terms in (2.100) can be neglected, without causing a significant error, provided that $x_k - \hat{x}_{k|k-1}$ is small in a mean square sense. The linearized measurement equation is computed as:

$$z_k = h(\hat{x}_{k|k-1}) + H(\hat{x}_{k|k-1})\,(x_k - \hat{x}_{k|k-1}) + e_k \tag{2.101}$$

where $H(\hat{x}_{k|k-1})$ is the ny-by-nx matrix of partial derivatives evaluated along $\hat{x}_{k|k-1}$. Taking the expectations on both sides of (2.101) yields:

$$\hat{z}_k = h(\hat{x}_{k|k-1}) \tag{2.102}$$

because $E\{x_k - \hat{x}_{k|k-1}\} = \underline{0}$ and $\{e_k\}$ is a zero mean sequence.

Let:

$$\tilde{z}_k = z_k - \hat{z}_k \tag{2.103}$$

and substitute equations (2.101) and (2.102) into (2.103):

$$\begin{aligned}\tilde{z}_k &= h(\hat{x}_{k|k-1}) + H(\hat{x}_{k|k-1})\,(x_k - \hat{x}_{k|k-1}) + e_k - h(\hat{x}_{k|k-1})\\ &= H(\hat{x}_{k|k-1})\,\tilde{x}_{k|k-1} + e_k\end{aligned} \tag{2.104}$$

Recall, that in the linear case, the residual $\tilde{z}_k$ is given by:

$$\begin{aligned}\tilde{z}_k &= z_k - \hat{z}_k\\ &= C_k\,x_k + e_k - C_k\,\hat{x}_{k|k-1})\\ &= C_k\,\tilde{x}_{k|k-1} + e_k\end{aligned} \tag{2.105}$$

which is identical with (2.104) except for the fact that C_k is replaced by $H(\hat{x}_{k|k-1})$. What can be concluded from this is that, the KF is applicable to the linearized system. The formal differences between the KF and the EKF are, that for the latter filter algorithm, A and C are replaced by F and H, respectively. The algorithm for the 2 phase Extended Kalman Filter is summarized in Table 2.3.

Table 2.3: Summary of the 2 phase Extended Kalman Filter algorithm

2 Phase EKF Algorithm

Phase 1: time-update equations

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k}, u_k) \tag{2.106}$$

$$P_{k+1|k} = F(\hat{x}_{k|k}, u_k)\, P_{k|k} \left(F(\hat{x}_{k|k}, u_k)\right)^T + Q_k^v \tag{2.107}$$

Phase 2: measurement-update equations

$$K_{k+1}^F = P_{k+1|k}\, H^T(\hat{x}_{k+1|k}) \times \left[H(\hat{x}_{k+1|k}) P_{k+1|k} H(\hat{x}_{k+1|k}) + Q_{k+1}^e\right]^{-1} \tag{2.108}$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}^F \left[z_{k+1} - h(\hat{x}_{k+1|k})\right] \tag{2.109}$$

$$P_{k+1|k+1} = \left[I - K_{k+1}^F H(\hat{x}_{k+1|k})\right] P_{k+1|k} \tag{2.110}$$

Definitions:

$$F(\hat{x}_{k|k}, u_k) = \left.\frac{\partial f(x_k, u_k)}{\partial x_k}\right|_{x_k = \hat{x}_{k|k}} \tag{2.111}$$

$$H(\hat{x}_{k+1|k}) = \left.\frac{\partial h(x_{k+1})}{\partial x_{k+1}}\right|_{x_{k+1} = \hat{x}_{k+1|k}} \tag{2.112}$$

It is possible to convert the 2 phase EKF algorithm into a single phase EKF algorithm. For the KF case, this transformation led to an ***equivalent algorithm***, i.e. the time-updated[29] covariances and state estimates of single phase and 2 phase algorithm, are identical. This is not the case for the Extended Kalman Filter algorithm. As the derivation above has shown, the nonlinear system matrix needs

[29]Only those are available in the single phase algorithm

to be expanded in a Taylor series, in order to make a linear estimator applicable. The expansion is carried out around the most recent estimate, the measurement-updated state estimate $\hat{x}_{k|k}$. This estimate is not available in the single phase algorithm, so the time-updated estimate $\hat{x}_{k|k-1}$ is used instead. It is expected, that $\hat{x}_{k|k}$ is the better estimate, i.e. is closer to x_k in a mean square sense, because it is based on all the information $\hat{x}_{k|k-1}$ is based on, plus the information about the latest measurement z_k. What can be concluded from this is that, the 2 phase EKF algorithm should work better than the single phase algorithm, because the errors made by neglecting the higher order terms in the Taylor series expansion of $f(\cdot)$ are smaller. Because the procedure to convert the 2 phase EKF algorithm into a single phase algorithm is so similar to the one applied in the KF case, it is omitted here. A summary of the single phase EKF algorithm is given in Table 2.4.

Table 2.4: Summary of the single phase Extended Kalman Filter algorithm

Single Phase EKF Algorithm

Filter equations:

$$S_k = H(\hat{x}_{k|k-1})\, P_{k|k-1}\, H^T(\hat{x}_{k|k-1}) + Q_k^e \qquad (2.113)$$

$$K_k = F(\hat{x}_{k|k-1}, u_k)\, P_{k|k-1}\, H^T(\hat{x}_{k|k-1})\, S_k^T \qquad (2.114)$$

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k-1}, u_k) + K_k \left[z_k - h(\hat{x}_{k|k-1})\right] \qquad (2.115)$$

$$\begin{aligned} P_{k+1|k} = {} & F(\hat{x}_{k|k-1}, u_k)\, P_{k|k-1} \left(F(\hat{x}_{k|k-1}, u_k)\right)^T \\ & - K_k S_k K_k^T + Q_k^v \end{aligned} \qquad (2.116)$$

Definitions:

$$F(\hat{x}_{k|k-1}, u_k) = \left.\frac{\partial f(x_k, u_k)}{\partial x_k}\right|_{x_k = \hat{x}_{k|k-1}} \qquad (2.117)$$

$$H(\hat{x}_{k|k-1}) = \left.\frac{\partial h(x_k)}{\partial x_k}\right|_{x_k = \hat{x}_{k|k-1}} \qquad (2.118)$$

In this first approach, it is rather loosely shown, how the nonlinear system equations can be linearized and converted into a form, that allows to formally apply the Kalman filter structure. The EKF is more rigorously derived using the MV optimality criteria and other constraints. The derivation given below follows this methods.

Second Approach:

To gain better insight and deeper understanding of the EKF algorithm, another, more rigorous filter derivation is given here. This derivation is achieved by forcing the filter to have the same linear structure as the KF. In order to keep the evaluations compact, references to the first approach will be made, where possible.

The first step is to find the time-update equations for the error covariance matrix. The state estimate time-update equation is derived in the previous section (cf. eqn. 2.95). The time-updated estimation covariance is by definition:

$$P_{k+1|k} = E\left\{\tilde{x}_{k+1|k}\,\tilde{x}^T_{k+1|k} \mid Z_k\right\} \tag{2.119}$$

where the time-updated error vector $\tilde{x}_{k+1|k}$ is given by equation (2.98). Therefore, $P_{k+1|k}$ can be expressed as:

$$\begin{aligned} P_{k+1|k} &= E\left\{\left(F(\hat{x}_{k|k},u_k)\,\tilde{x}_{k|k} + v_k\right)\left(F(\hat{x}_{k|k},u_k)\,\tilde{x}_{k|k} + v_k\right)^T \mid Z_k\right\} \\ &= F(\hat{x}_{k|k},u_k)\,E\left\{\tilde{x}_{k|k}\,\tilde{x}^T_{k|k} \,\middle|\, Z_k\right\} F^T(\hat{x}_{k|k}) + E\left\{v_k v_k^T \mid Z_k\right\} \\ &\quad + E\left\{v_k\,\tilde{x}^T_{k|k} \mid Z_k\right\} F^T(\hat{x}_{k|k},u_k) + F(\hat{x}_{k|k},u_k)\,E\left\{\tilde{x}_{k|k}\,v_k^T \mid Z_k\right\} \end{aligned} \tag{2.120}$$

The last two terms in the equation above are zero, because the system noise vector v_k is independent of error vector $\tilde{x}_{k|k}$ and measurement set Z_k and because $\{v_k\}$ is a zero mean sequence. The second term in (2.120) is the system noise covariance matrix Q^v_k (cf. Chapter 2.3.1). Finally, substituting the defining equation for the measurement updated covariance matrix $P_{k|k}$

$$P_{k|k} = E\left\{\tilde{x}_{k|k}\,\tilde{x}^T_{k|k} \mid Z_k\right\} \tag{2.121}$$

into equation (2.120) yields:

$$P_{k+1|k} = F(\hat{x}_{k|k}, u_k)\, P_{k|k}\, F(\hat{x}_{k|k}, u_k)^T + Q_k^v \qquad (2.122)$$

The time-updating of the state estimate is now completed. The next step is to take into account the new measurements, i.e. to obtain the measurement-update equations for the state estimate $\hat{x}$ and the with $\hat{x}$ associated covariance matrix. Led by the desire to retain a linear filter structure, the measurement updated estimate $\hat{x}_{k+1|k+1}$ is forced to be a linear function of the measurement vector z_{k+1}. Therefore, let $\hat{x}_{k+1|k+1}$ be given by:

$$\hat{x}_{k+1|k+1} = a_{k+1} + K_{k+1}^F z_{k+1} \qquad (2.123)$$

where the vector a_{k+1} and the matrix K_k are to be determined. Substituting (2.123) and (2.83) into equation (2.97) yields:

$$\tilde{x}_{k+1|k+1} = a_{k+1} + K_{k+1}^F \left(h(x_{k+1}) + e_{k+1}\right) + \tilde{x}_{k+1|k} - \hat{x}_{k+1|k} \qquad (2.124)$$

Taking the expectations on both sides, yields:

$$\begin{aligned} \underline{0} &= a_{k+1} + K_{k+1}^F\, E\left\{h(x_{k+1}\right\} - \hat{x}_{k+1|k} \\ a_{k+1} &= \hat{x}_{k+1|k} - K_{k+1}^F\, E\left\{h(x_{k+1})\right\} \end{aligned} \qquad (2.125)$$

Substituting equation (2.125) back into (2.119) gives:

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}^F \left(z_{k+1} - E\left\{h(x_{k+1})\right\}\right) \qquad (2.126)$$

It is impractical to compute the conditional mean of $h(x_{k+1})$, as given in (2.125), because this requires the knowledge of the conditional probability density function $p_{h(\cdot)|Z_k}$. An expansion of $h(x_{k+1})$ into a Taylor series, evaluated about the most recent state estimate, does circumvent this obstacle. The series is truncated after the linear term in order to preserve a linear filter structure. Omission of the higher order terms is justified by the assumption that, the state estimate $\hat{x}$ is close (in a mean square sense) to the system state x at any time instant k.

$$\begin{aligned} h(x_{k+1}) &= h(\hat{x}_{k+1|k}) + \left.\frac{\partial h}{\partial x}\right|_{x=\hat{x}_{k+1|k}} (x_{k+1} - \hat{x}_{k+1|k}) \\ &= h(\hat{x}_{k+1|k}) + H(\hat{x}_{k+1|k})\,(x_{k+1} - \hat{x}_{k+1|k}) \end{aligned} \tag{2.127}$$

Applying the expected value operator on both sides generates:

$$E\{h(x_{k+1})\} = h(\hat{x}_{k+1|k}) \tag{2.128}$$

Substituting (2.127) into (2.83) produces:

$$z_{k+1} = h(\hat{x}_{k+1|k}) + H(\hat{x}_{k+1|k})\,(x_{k+1} - \hat{x}_{k+1|k}) + e_{k+1} - h(\hat{x}_{k|k-1}) \tag{2.129}$$

With (2.128) and (2.129) equation (2.126) can now be rewritten:

$$\begin{aligned} \hat{x}_{k+1|k+1} &= \hat{x}_{k+1|k} + K^F_{k+1}\left(H(x_{k+1|k})(x_{k+1} - \hat{x}_{k+1|k}) + e_{k+1}\right) \\ &= \hat{x}_{k+1|k} + K^F_{k+1}\,\tilde{z}_{k+1|k} \end{aligned} \tag{2.130}$$

where for the last step, equation (2.104) is used.

$$\tilde{x}_{k+1|k+1} = \tilde{x}_{k+1|k} - K^F_{k+1}\,\tilde{z}_{k+1|k} \tag{2.131}$$

The gain matrix K^F_{k+1} in (2.130) has yet to be determined. To do so, one further constraint on $x_{k+1|k+1}$ is imposed. The measurement-updated state estimate $x_{k+1|k+1}$ is desired to be a minimum variance estimate. Recall from the previous section, that a MV estimate is equal to the conditional mean no matter what the underlying distribution is. The measurement-updated error covariance matrix $P_{k+1|k+1}$ is defined as:

$$P_{k+1|k+1} = E\left\{\tilde{x}_{k+1|k+1}\,\tilde{x}^T_{k+1|k+1}\right\} \tag{2.132}$$

Let J_{k+1} denote the trace of $P_{k+1|k+1}$, that is:

$$\begin{aligned} J_{k+1} &= trace\left[P_{k+1|k+1}\right] \\ &= trace\left[E\left\{\tilde{x}_{k+1|k+1}\,\tilde{x}^T_{k+1|k+1}\right\}\right] \\ &= trace\left[E\left\{\left(\tilde{x}_{k+1|k} - K^F_{k+1}\tilde{z}_{k+1|k}\right)\left(\tilde{x}_{k+1|k} - K^F_{k+1}\tilde{z}_{k+1|k}\right)^T\right\}\right] \end{aligned} \tag{2.133}$$

From the evaluations about minimum variance estimates (cf. Chapter 2.2.2) it is known, that minimizing J_{k+1} yields the desired MV estimate. The minimum is found, by setting the partial derivative of J_{k+1} with respect to K^F_{k+1}, to zero.

$$\frac{\partial J_{k+1}}{\partial K^F_{k+1}} = \frac{\partial}{\partial K^F}\left[trace\,E\left\{\tilde{x}\tilde{x}^T - \tilde{x}\tilde{z}^T(K^F)^T - K^F\tilde{z}\tilde{x}^T + K^F\tilde{z}\tilde{z}^T(K^F)^T\right\}\right] \tag{2.134}$$

Exchanging the order of integration and differentiation in equation (2.134) yields:

$$\begin{aligned}\frac{\partial J_{k+1}}{\partial K^F_{k+1}} &= trace\,E\left\{\frac{\partial}{\partial K^F}\left[\tilde{x}\tilde{x}^T - \tilde{x}\tilde{z}^T(K^F)^T - K^F\tilde{z}\tilde{x}^T + K^F\tilde{z}\tilde{z}^T(K^F)^T\right]\right\}\\ &= trace\,E\left\{\left[\underline{0} - \tilde{x}\tilde{z}^T - \tilde{x}\tilde{z}^T + 2K^F\tilde{z}\tilde{z}^T\right]\right\}\\ &= 2\times trace\left[K^F E\left\{\tilde{z}\tilde{z}^T\right\} - E\left\{\tilde{x}\tilde{z}^T\right\}\right]\end{aligned} \tag{2.135}$$

The right hand side of (2.135) is in general zero, if the expression within the brackets is zero. Therefore:

$$K^F\,E\left\{\tilde{z}\tilde{z}^T\right\} = E\left\{\tilde{x}\tilde{z}^T\right\} \tag{2.136}$$

$$K^F = E\left\{\tilde{x}\tilde{z}^T\right\}\left(E\left\{\tilde{z}\tilde{z}^T\right\}\right)^{-1} \tag{2.137}$$

Using the equations for the time-updated covariance $P_{k+1|k}$ (2.119) and for the output error $\tilde{z}_{k+1|k}$ (2.104), the expectation operations in the equation above are readily carried out. Recall also, that e_{k+1} is not correlated to $\tilde{x}_{k+1|k}$ and $\hat{x}_{k+1|k}$.

$$\begin{aligned}E\left\{\tilde{x}_{k+1|k}\tilde{z}^T_{k+1|k}\right\} &= E\left\{\tilde{x}_{k+1|k}\left(\tilde{x}^T_{k+1|k}\left(H(\hat{x}_{k+1|k})\right)^T + e_{k+1}\right)\right\}\\ &= E\left\{\tilde{x}_{k+1|k}\tilde{x}^T_{k+1|k}\left(H(\hat{x}_{k+1|k})\right)^T\right\} + E\left\{\tilde{x}_{k+1|k}\,e_{k+1}\right\}\\ &= P_{k+1|k}\left(H(\hat{x}_{k+1|k})\right)^T + 0\end{aligned} \tag{2.138}$$

$$\begin{aligned}E\left\{\tilde{z}_{k+1|k}\tilde{z}^T_{k+1|k}\right\} &= E\left\{\left(H(\hat{x}_{k+1|k})\tilde{x}_{k+1|k} + e_{k+1}\right)\left(H(\hat{x}_{k+1|k})\,\tilde{x}_{k+1|k} + e_{k+1}\right)^T\right\}\\ &= H(\hat{x}_{k+1|k})E\left\{\tilde{x}_{k+1|k}\tilde{x}^T_{k+1|k}\right\}\left(H(\hat{x}_{k+1|k})\right)^T + E\left\{e_{k+1}e^T_{k+1}\right\}\\ &\quad + E\left\{e_{k+1}\tilde{x}^T_{k+1|k}\right\}\left(H(\hat{x}_{k+1|k})\right)^T + H(\hat{x}_{k+1|k})E\left\{\tilde{x}_{k+1|k}\,e^T_{k+1}\right\}\\ &= H(\hat{x}_{k+1|k})P_{k+1|k}\left(H(\hat{x}_{k+1|k})\right)^T + Q^e_{k+1}\end{aligned} \tag{2.139}$$

So, the gain matrix that minimizes the variance of the state estimate $\hat{x}_{k+1|k}$ is given by:

$$K_{k+1} = P_{k+1|k} \left(H(\hat{x}_{k+1|k})\right)^T \left(H(\hat{x}_{k+1|k}) P_{k+1|k} \left(H(\hat{x}_{k+1|k})\right)^T + Q^e_{k+1}\right)^{-1} \tag{2.140}$$

The measurement-update equation for the covariance matrix P is:

$$\begin{aligned} P_{k+1|k+1} &= E\left\{\tilde{x}_{k+1|k}\,\tilde{x}^T_{k+1|k} - \tilde{x}_{k+1|k}\,\tilde{z}^T_{k+1|k}\left(K^F_{k+1}\right)^T - K^F_{k+1}\tilde{z}_{k+1|k}\,\tilde{x}^T_{k+1|k} \right. \\ &\quad \left. + K^F_{k+1}\tilde{z}_{k+1|k}\,\tilde{z}^T_{k+1|k}\left(K^F_{k+1}\right)^T\right\} \\ &= P_{k+1|k} - K^F_{k+1}E\left\{\tilde{z}_{k+1|k}\,\tilde{x}^T_{k+1|k}\right\} - E\left\{\tilde{x}_{k+1|k}\tilde{z}^T_{k+1|k}\right\}\left(K^F_{k+1}\right)^T \\ &\quad + K^F_{k+1}E\left\{\tilde{z}_{k+1|k}\,\tilde{z}^T_{k+1|k}\right\}\left(K^F_{k+1}\right)^T \end{aligned} \tag{2.141}$$

Recalling that K^F_{k+1} is such that equation (2.136) holds, the last two terms in (2.141) cancel out. The second term can also be expressed by:

$$\begin{aligned} K^F_{k+1}E\left\{\tilde{z}_{k+1|k}\,\tilde{x}^T_{k+1|k}\right\} &= K^F_{k+1}E\left\{H(\hat{x}_{k+1|k})\tilde{x}_{k+1|k}\,\tilde{x}^T_{k+1|k} + e_{k+1}\tilde{x}^T_{k+1|k}\right\} \\ &= K^F_{k+1}H(\hat{x}_{k+1|k})P_{k+1|k} \end{aligned} \tag{2.142}$$

Substituting these results back into (2.141) yields the covariance measurement-update equation.

$$\begin{aligned} P_{k+1|k+1} &= P_{k+1|k} - K^F_{k+1}H(\hat{x}_{k+1|k})\,P_{k+1|k} \\ &= \left[I - K^F_{k+1}H(\hat{x}_{k+1|k})\right]P_{k+1|k} \end{aligned} \tag{2.143}$$

The second Extended Kalman Filter derivation is now complete. The algorithm obtained via this approach, is identical with the one derived before. Summary of the EKF algorithms for the 2 phase filter and for the single phase filter are given in Table 2.3 and Table 2.4, respectively.

2.3.3 Conclusion

When comparing the EKF algorithm with the KF algorithm, one gets the impression, that the differences between them are insignificant. (Just replace A and B in the KF algorithm, by F and H to get to the EKF algorithm.) There are however, substantial differences. The matrices F and H are random, because the Taylor series is evaluated around the most recent state estimate, which itself depends on the observed output data. Therefore, error covariance matrix P and Kalman gain matrix K can non longer -- contrary to the KF -- be precomputed, since they are dependent on the actual measurements taken. Also, the data might be such, that covariance P becomes singular, and appropriate heuristic methods might become necessary, in order to keep P positive definite.

Note, that the EKF algorithm works only, if the errors made in truncating the Taylor series, are indeed negligible. This will be the case, when good initial state estimates are available, i.e. P_0 is small, and when the process noise is not too large, such that all state estimates $\hat{x}$ are close to the system state vector x. Any large $\|v_k\|$ can bring the algorithm out of the linear region[30] and probably cause the algorithm to diverge. In any case, Monte Carlo simulations are essential to assure satisfactory performance of the EKF in a particular application.

At the end of Chapter 2.2.4 it is discussed whether parameter estimation problems could be attacked using the KF. It is shown there, that parameter estimation is a nonlinear filtering problem, so the Kalman filter is not the right choice. The EKF, on the other hand, is suitable to treat such problems. In the next section it is shown, how uncertain parameters of linear systems can be estimated by the application of Extended Kalman Filter theory.

2.4 The Extended Kalman Filter as a Parameter Estimator

Up till now, it was assumed, that state space equations that describe the observed input/output data adequately, are available. In practical cases however, one often

[30]With *linear region* is meant the nx-dimensional region where the higher order terms in the series expansions are much smaller than the linear term

can only — by exploiting the physical laws governing the process — determine the structure of the system, but not all its coefficients. Another problem one frequently encounters in practice is, that the system under consideration is time-varying in an unpredictable way, e.g. caused by aging, wear or changes in the environment. In any case, where it is desired to apply optimal control inputs, to estimate the system state or to detect errors in the system, a complete description is essential. It is natural to model these unknown coefficients as parameters in the state space equations, and then estimate the system states and the parameters simultaneously. It was shown is Chapter 2.2.4 that this is a nonlinear filtering problem, for which the KF in not applicable. The EKF, derived in the previous section, is suited for attacking this problem. How this can be done, is shown next. Note, that the presentation of the EKF algorithm given here, is taken from [33]; the only differences are that more intermediate steps are shown here, and that the algorithm is corrected by a few misprints that appeared in [33].

Consider a linear, time-invariant, discrete-time system, adequately described by:

$$x_{k+1} = A_o x_k + B_o u_k + v_k \tag{2.144}$$

$$z_k = C_o x_k + e_k \tag{2.145}$$

where:

k	$\geq$	k_0	k_0 is initial time instant
x_k	$\in$	$\mathcal{R}^{nx}$	is system state vector
z_k	$\in$	$\mathcal{R}^{ny}$	is measurement vector
v_k	$\in$	$\mathcal{R}^{nx}$	is process noise vector
e_k	$\in$	$\mathcal{R}^{ny}$	is measurement noise vector
A_o	$\in$	$\mathcal{R}^{nx \times nx}$	is system matrix
B_o	$\in$	$\mathcal{R}^{nx \times nu}$	is input distribution matrix
C_o	$\in$	$\mathcal{R}^{ny \times nx}$	is measurement matrix

The time-invariant matrices A, B and C is given the subscript "o" to indicate, that they describe the input/output behavior of the system optimally, i.e. there is no other set (A, B, C) that describes the measured data better.

As in the case for the KF, the intial state is assumed to be Gaussian, zero-mean random vector with covariance Π_0. Further assumptions are that the noise processes $\{v_k\}$, $\{e_k\}$ are white, Gaussian, zero mean sequences with statistics:

$$E\{v_k\} = 0 \tag{2.146}$$

$$E\{e_k\} = 0 \tag{2.147}$$

$$E\left\{v_k v_j^T\right\} = Q_k^v \delta_{kj} \tag{2.148}$$

$$E\left\{e_k e_j^T\right\} = Q_k^e \delta_{kj} \tag{2.149}$$

where:

Q_k^v	is a symmetric positive semidefinite matrix
Q_k^e	is a symmetric positive definite (pd) matrix
δ_{kj}	is 1 for $k = j$ and is 0 otherwise

Initial state and noise sequences are uncorrelated, that is:

$$E\left\{x_0 v_k^T\right\} = \underline{0} \tag{2.150}$$

$$E\left\{x_0 e_k^T\right\} = \underline{0} \tag{2.151}$$

$$E\left\{v_j e_k^T\right\} = \underline{0} \tag{2.152}$$

Suppose, that the structure of the system is known, but the information about the matrices A, B, C is incomplete, i.e. not all of their entries are completely known. These uncertainties can be included in the model as parameters and — just like the states — be regarded as random variables. Assume, there are np parameters, combined into a parameter vector, say $\boldsymbol{\theta}$. In general it has to be assumed, that all of the system matrices are dependent on this parameter vector $\boldsymbol{\theta}$. So, the system (2.144), (2.145) is adequately modelled by:

$$x_{k+1} = A(\boldsymbol{\theta})x_k + B(\boldsymbol{\theta})u_k + v_k \tag{2.153}$$

$$z_k = C(\boldsymbol{\theta})x_k + e_k \tag{2.154}$$

As stated earlier, an obvious thing to do, is to extend the state vector x by the parameter vector θ to form an augmented state vector x^A.

$$x_k^A = \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} \tag{2.155}$$

where:

$$\begin{array}{lcll} x_k & \in & \mathcal{R}^{nx} & \text{is system state vector} \\ \theta_k & \in & \mathcal{R}^{np} & \text{is parameter vector} \\ x_k^A & \in & \mathcal{R}^{nA} & \text{is augmented state vector} \end{array}$$

Estimation of x^A is a joint parameter and state estimation problem. The system is assumed to be time-invariant. Hence, the parameter vector is best modeled by:

$$\theta_{k+1} = \theta_k \tag{2.156}$$

Combining equations (2.153), (2.154), (2.155) and (2.156) allows to rewrite the model state space equations in terms of x^A:

$$x_{k+1}^A = \begin{bmatrix} A(\theta_k)x_k + B(\theta_k)u_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} v_k \\ 0 \end{bmatrix} \tag{2.157}$$

$$z_k = C(\theta_k)x_k + e_k \tag{2.158}$$

Let:

$$f(x_k^A, u_k) = \begin{bmatrix} A(\theta_k)x_k + B(\theta_k)u_k \\ \theta \end{bmatrix} \tag{2.159}$$

$$h(x_k^A) = C(\theta_k)x_k \tag{2.160}$$

With (2.159) and (2.160) the state equations for the augmented system (2.157), (2.158) become:

$$x_{k+1}^A = f(x_k^A, u_k) + \begin{bmatrix} v_k \\ 0 \end{bmatrix} \tag{2.161}$$

$$z_k = h(x_k^A) + e_k \tag{2.162}$$

From the above equations it is observed that estimation of the unknown parameter vector θ is a nonlinear filtering problem. A comparison of the above two equations, along with the nonlinear state space equations (2.82) and (2.83) reveals, that the

EKF algorithm as given in Table 2.4, is directly applicable to this problem. The single phase EKF algorithm for this problem is:

$$S_k = H(\hat{x}^A_{k|k-1})\, P_{k|k-1}\, H^T(\hat{x}^A_{k|k-1}) + Q^e_k \tag{2.163}$$

$$N_k = F(\hat{x}^A_{k|k-1}, u_k)\, P_{k|k-1}\, H^T(\hat{x}^A_{k|k-1})\, S^T_k \tag{2.164}$$

$$\hat{x}^A_{k+1|k} = f(\hat{x}^A_{k|k-1}, u_k) + N_k \left[z_k - h(\hat{x}^A_{k|k-1})\right] \tag{2.165}$$

$$P_{k+1|k} = F(\hat{x}^A_{k|k-1}, u_k)\, P_{k|k-1} \left(F(\hat{x}^A_{k|k-1}, u_k)\right)^T - N_k S_k N^T_k + \overline{Q}^v_k \tag{2.166}$$

where:

$$\begin{aligned}
F(\hat{x}^A_{k|k-1}, u_k) &= \left.\frac{\partial f(x^A_k, u_k)}{\partial x^A_k}\right|_{x^A_k = \hat{x}^A_{k|k-1}} \\
&= \left.\frac{\partial}{\partial x^A_k}\begin{bmatrix} A(\theta_k)x_k + B(\theta_k)u_k \\ \theta_k \end{bmatrix}\right|_{x^A_k = \hat{x}^A_{k|k-1}} \\
&= \left.\begin{bmatrix} \frac{\partial}{\partial x_k}\left[A(\theta_k)x_k + B(\theta_k)u_k\right] & \frac{\partial}{\partial \theta_k}\left[A(\theta_k)x_k + B(\theta_k)u_k\right] \\ \frac{\partial}{\partial x_k}\theta_k & \frac{\partial}{\partial \theta_k}\theta_k \end{bmatrix}\right|_{x^A_k = \hat{x}^A_{k|k-1}} \\
&= \begin{bmatrix} A(\hat{\theta}_{k|k-1}) & M(\hat{\theta}_{k|k-1}, \hat{x}_{k|k-1}, u_k) \\ \underline{0} & I \end{bmatrix}
\end{aligned} \tag{2.167}$$

$$\begin{aligned}
H(\hat{x}^A_{k|k-1}) &= \left.\frac{\partial h(x^A_k)}{\partial x^A_k}\right|_{x^A_k = \hat{x}^A_{k|k-1}} \\
&= \left.\begin{bmatrix} \frac{\partial}{\partial x_k}\left[C(\theta_k)x_k\right] & \frac{\partial}{\partial \theta_k}\left[C(\theta_k)x_k\right] \end{bmatrix}\right|_{\theta_k = \hat{\theta}_{k|k-1}} \\
&= \begin{bmatrix} C(\hat{\theta}_{k|k-1}) & D(\hat{\theta}_{k|k-1}, \hat{x}_{k|k-1}) \end{bmatrix}
\end{aligned} \tag{2.168}$$

$$\overline{Q}^v_k = \begin{bmatrix} Q^v_k & \underline{0} \\ \underline{0} & \underline{0} \end{bmatrix} \tag{2.169}$$

The structure of the matrices F and H, as revealed by equations (2.167), (2.168) suggests to rewrite the EKF algorithm in a partitioned, computationally more efficient, form. To keep this step tractable and compact, a short form notation is introduced next.

$$\begin{aligned}
P_k &= P_{k|k-1} \\
A_k &= A(\hat{\theta}_{k|k-1}) \\
B_k &= B(\hat{\theta}_{k|k-1})
\end{aligned}$$

$$
\begin{aligned}
C_k &= C(\hat{\theta}_{k|k-1}) \\
M_k &= M(\hat{\theta}_{k|k-1}, \hat{x}_{k|k-1}, u_k) \\
D_k &= D(\hat{\theta}_{k|k-1}, \hat{x}_{k|k-1})
\end{aligned}
$$

The gain matrix N_k is naturally subdivided into:

$$
N_k = \begin{bmatrix} K_k \\ L_k \end{bmatrix} \tag{2.170}
$$

where:

K_k	$\in$	$\mathcal{R}^{nx \times ny}$	is the state update Kalman gain matrix
L_k	$\in$	$\mathcal{R}^{np \times ny}$	is the parameter update Kalman gain matrix

The covariance matrix P_k is partitioned into:

$$
P_k = \begin{bmatrix} P1_k & P2_k \\ P2_k^T & P3_k \end{bmatrix} \tag{2.171}
$$

where:

$P1_k$	$\in$	$\mathcal{R}^{nx \times nx}$	is the state estimate covariance matrix
$P2_k$	$\in$	$\mathcal{R}^{nx \times np}$	is the parameter/state estimate cross-covariance matrix
$P3_k$	$\in$	$\mathcal{R}^{np \times np}$	is the parameter estimate covariance matrix

Substitution of (2.171) into equation (2.163) yields:

$$
\begin{aligned}
S_k &= \begin{bmatrix} C_k & D_k \end{bmatrix} \begin{bmatrix} P1_k & P2_k \\ P2_k^T & P3_k \end{bmatrix} \begin{bmatrix} C_k^T \\ D_k^T \end{bmatrix} + Q_k^e \\
&= \begin{bmatrix} C_k & D_k \end{bmatrix} \begin{bmatrix} P1_k C_k^T + P2_k D_k^T \\ P2_k^T C_k^T + P3_k D_k^T \end{bmatrix} + Q_k^e \\
&= C_k P1_k C_k^T + C_k P2_k D_k^T + D_k P2_k^T C_k^T + D_k P3_k D_k^T + Q_k^e
\end{aligned} \tag{2.172}
$$

The Kalman gain matrices are given by:

$$
\begin{aligned}
\begin{bmatrix} K_k \\ L_k \end{bmatrix} &= \begin{bmatrix} A_k & M_k \\ \underline{0} & I \end{bmatrix} \begin{bmatrix} P1_k & P2_k \\ P2_k^T & P3_k \end{bmatrix} \begin{bmatrix} C_k^T \\ D_k^T \end{bmatrix} \times S_K^{-1} \\
&= \begin{bmatrix} A_k & M_k \\ \underline{0} & I \end{bmatrix} \begin{bmatrix} P1_k C_k^T + P2_k D_k^T \\ P2_k^T C_k^T + P3_k D_k^T \end{bmatrix} \times S_k^{-1} \\
&= \begin{bmatrix} A_k P1_k C_k^T + A_k P2_k D_k^T + M_k P2_k^T C_k^T + M_k P3_k D_k^T \\ P2_k^T C_k^T + P3_k D_k^T \end{bmatrix} \times S_k^{-1}
\end{aligned}
$$

$$
K_k = \left(A_k P1_k C_k^T + A_k P2_k D_k^T + M_k P1_k C_k^T + M_k P2_k D_k^T\right) \times S_k^{-1} \tag{2.173}
$$

$$
L_k = \left(P2_k^T C_k^T + P3_k D_k^T\right) \times S_k^{-1} \tag{2.174}
$$

With (2.159), (2.160) and (2.170) the estimate update equation (2.165) becomes:

$$\begin{aligned}
\begin{bmatrix} x_{k+1|k} \\ \theta_{k+1|k} \end{bmatrix} &= \begin{bmatrix} A_k x_{k|k-1} + B_k u_k \\ \theta_{k|k-1} \end{bmatrix} + \begin{bmatrix} K_k \\ L_k \end{bmatrix} \times \left[z_k - C_k x_{k|k-1} \right] \\
x_{k+1|k} &= A_k x_{k|k-1} + B_k u_k + K_k \left[z_k - C_k x_{k|k-1} \right] && (2.175) \\
\theta_{k+1|k} &= \theta_{k|k-1} + L_k \left[z_k - C_k x_{k|k-1} \right] && (2.176)
\end{aligned}$$

The last equation of the EKF algorithm to be partitioned, is the covariance matrix update equation (2.166):

$$\begin{aligned}
\begin{bmatrix} P1_{k+1} & P2_{k+1} \\ P2_{k+1}^T & P3_{k+1} \end{bmatrix} &= \begin{bmatrix} A_k & M_k \\ \underline{0} & I \end{bmatrix} \begin{bmatrix} P1_k & P2_k \\ P2_k^T & P3_k \end{bmatrix} \begin{bmatrix} A_k & M_k \\ \underline{0} & I \end{bmatrix}^T - \\
&\quad \begin{bmatrix} K_k \\ L_k \end{bmatrix} S_k \begin{bmatrix} K_k \\ L_k \end{bmatrix}^T + \begin{bmatrix} Q_k^v & \underline{0} \\ \underline{0} & \underline{0} \end{bmatrix} \\
&= \begin{bmatrix} A_k & M_k \\ \underline{0} & I \end{bmatrix} \begin{bmatrix} P1_k A_k^T + P2_k M_k^T & P2_k \\ P2_k^T A_k^T + P3_k M_k^T & P3_k \end{bmatrix} - \\
&\quad \begin{bmatrix} K_k S_k K_k^T - Q_k^v & K_k S_k L_k^T \\ L_k S_k K_k^T & L_k S_k L_k^T \end{bmatrix} \\
P1_{k+1} &= A_k P1_k A_k^T + A_k P2_k M_k^T + M_k P2_k^T A_k^T + M_k P3_k M_k^T \\
&\quad - K_k S_k K_k^T + Q_k^v && (2.177) \\
P2_{k+1} &= A_k P2_k + M_k P3_k - K_k S_k L_k^T && (2.178) \\
P3_{k+1} &= P3_k - L_k S_k L_k^T && (2.179)
\end{aligned}$$

A summary of this partitioned EKF algorithm, is given in Table 2.5. The order in which the computation of the individual equations needs to be done, corresponds to their locations in Table 2.5. To start the algorithm, estimates for the state vector and for the parameter vector as well as the with these estimates associated covariances have to be available.

Table 2.5: Summary of the partitioned single phase Extended Kalman Filter algorithm as a parameter estimator

Partitioned Single Phase EKF Algorithm

The system:

$$\begin{aligned} x_{k+1} &= A_o x_k + B_o u_k + v_k && (2.180) \\ z_k &= C_o x_k + e_k && (2.181) \end{aligned}$$

Filter equations:

$$\begin{aligned} S_k &= C_k P1_k C_k^T + C_k P2_k D_k^T + D_k P2_k^T C_k^T + \\ &\quad D_k P3_k D_k^T + Q_k^e && (2.182) \\ K_k &= \left(A_k P1_k C_k^T + A_k P2_k D_k^T + M_k P1_k C_k^T + \right. \\ &\quad \left. M_k P2_k D_k^T + Q_k^v\right) \times S_k^{-1} && (2.183) \\ L_k &= \left(P2_k^T C_k^T + P3_k D_k^T\right) \times S_k^{-1} && (2.184) \\ \theta_{k+1|k} &= \theta_{k|k-1} + L_k \left[z_k - C_k \hat{x}_{k|k-1}\right] && (2.185) \\ \hat{x}_{k+1|k} &= A_k \hat{x}_{k|k-1} + B_k u_k + K_k \left[z_k - C_k \hat{x}_{k|k-1}\right] && (2.186) \\ P1_{k+1} &= A_k P1_k A_k^T + A_k P2_k M_k^T + M_k P2_k^T A_k^T + \\ &\quad M_k P3_k M_k^T - K_k S_k K_k^T + Q_k^v && (2.187) \\ P2_{k+1} &= A_k P2_k + M_k P3_k - K_k S_k L_k^T && (2.188) \\ P3_{k+1} &= P3_k - L_k S_k L_k^T && (2.189) \end{aligned}$$

2.5 A Modified Extended Kalman Filter

As stated in Chapter 2.3, the EKF algorithm (cf. Table 2.5) is likely to give biased or divergent estimates. Lennart Ljung [31], [32] developed a general method to analyse the asymptotic properties of recursive identification algorithms. The basis for this method is to find a differential equation associated with the identification algorithm, whose stability properties are related to the convergence properties of the identification algorithm. In [33] Ljung applies this method specifically to the EKF, which is being used as a parameter estimator. The article shows the causes for divergence and biasedness of the EKF. Ljung also suggests in [33] a modification to the EKF algorithm that improves the convergence behavior of parameter estimator considerably. Because the theory behind Ljung's convergence analysis is rather demanding and mathematically involved, it is not covered in this thesis. The interested reader is referred to [31], [32], [33], [34] or [35]. This section merely tries to give the reader some intuition on the suggested modification in the EKF algorithm and shows how the algorithm needs to be changed.

Ljung's approach to analyze the EKF parameter estimation algorithm is to keep the model parameter vector constant at, say $\bar{\theta}$, and then examine the process produced by (2.182-2.189). To hold $\hat{\theta}$ constant, replace equation (2.185) by $\hat{\theta}_{k+1|k} = \bar{\theta}$. In [33] it is shown, that for $k - k_0$ large, the matrices $P3$ and $P2$ then tend to zero, whereas the matrices $P1$, S and K approach there steady state values $\overline{P1}$, $\bar{S}$ and $\bar{K}$ given by the solutions of:

$$\overline{P1}(\bar{\theta}) = A(\bar{\theta})\overline{P1}(\bar{\theta})A^T(\bar{\theta}) + Q^v - \bar{K}(\bar{\theta})\bar{S}(\bar{\theta})\bar{K}^T(\bar{\theta}) \tag{2.190}$$

$$\bar{S}(\bar{\theta}) = C(\bar{\theta})\overline{P1}(\bar{\theta})C^T(\bar{\theta}) + Q^e \tag{2.191}$$

$$\bar{K}(\bar{\theta}) = \left[A(\bar{\theta})\overline{P1}(\bar{\theta})C^T(\bar{\theta})\right]\bar{S}^{-1}(\bar{\theta}) \tag{2.192}$$

Define $\bar{\hat{x}}$ as the estimates obtained for this constant parameter vector $\bar{\theta}$.

$$\bar{\hat{x}}_{k+1|k} = A(\bar{\theta})\bar{\hat{x}}_{k|k-1} + B(\bar{\theta})u_k + \bar{K}(\bar{\theta})\bar{\epsilon}_k \tag{2.193}$$

with:

$$\bar{\epsilon}_k = z_k - C(\bar{\theta})\bar{\hat{x}}_{k|k-1} \tag{2.194}$$

Ljung interprets in [33] the EKF as an

> *...attempt to minimize the expected value of the squared residual associated with model θ.*

He seeks to minimize:

$$V(\bar{\theta}) = E\left\{\bar{\epsilon}_k\bar{\epsilon}_k^T\right\} \tag{2.195}$$

It is reasonable that in order to achieve minimization of (2.195), the parameter vector should be asymptotically adjusted in a negative gradient direction of $V(\bar{\theta})$. The negative gradient of $V(\bar{\boldsymbol{\theta}})$ is given by:

$$-\frac{\partial}{\partial\bar{\theta}}V(\bar{\theta}) = -2 \times E\left\{\left(\frac{\partial\epsilon_k^T}{\partial\bar{\theta}}\right)\bar{\epsilon}_k\right\} \tag{2.196}$$

Carrying out the operations on the right hand side of equation (2.196) leads to an updating scheme that is almost identical to the one of the EKF algorithm, as given in Table 2.5. The only differences are that the S^{-1} term in (2.184) is replaced by an ny-dimensional identity matrix, and that the $\left[\partial\bar{K}(\bar{\theta})/\partial\bar{\theta}\right]\bar{\epsilon}_k$ term is added to the M-matrix as given by (2.167). One might expect to obtain an EKF with improved convergence properties, when the modifications just mentioned, are included in the EKF algorithm. An approximation of the term $\left[\partial\bar{K}(\theta)/\partial\theta\right]\epsilon_k$ can be obtained from the equations (2.190 2.192). Define:

$$\kappa_k^{(i)} = \left.\frac{\partial\bar{K}(\theta)}{\partial\bar{\theta}^i}\right|_{\bar{\theta}^i = \hat{\theta}^i_{k|k-1}} \tag{2.197}$$

$$\sigma_k^{(i)} = \left.\frac{\partial\bar{S}(\theta)}{\partial\bar{\theta}^i}\right|_{\bar{\theta}^i = \hat{\theta}^i_{k|k-1}} \tag{2.198}$$

$$\Pi_k^{(i)} = \left.\frac{\partial\bar{P1}(\theta)}{\partial\bar{\theta}^i}\right|_{\bar{\theta}^i = \hat{\theta}^i_{k|k-1}} \tag{2.199}$$

The (i)'s in the equations above denote that the derivatives are taken with respect to the i-th entry in the parameter vector $\hat{\theta}$. Note, that $\partial\bar{K}(\boldsymbol{\theta})/\partial\bar{\theta}$ is an array of dimension $nx \times ny \times np$ and that the term $\left[\partial \overline{K}(\bar{\theta})/\partial \bar{\theta}\right] \bar{\epsilon}_k$ is a $nx \times np$ matrix, compatible with M.

Substitute the equations (2.190–2.192) into (2.197–2.199), respectively, and let $P1_k$, K_k, S_k, A_k and C_k be defined as in Table 2.5. Then:

$$\begin{aligned}
\sigma_k^{(i)} &= \frac{\partial}{\partial \bar{\theta}^i}\left[C(\bar{\theta})\overline{P1}(\bar{\theta})C^T(\bar{\theta}) + Q^e\right]\Bigg|_{\bar{\theta}^i = \hat{\theta}^i_{k|k-1}} \\
&= \left[\frac{\partial}{\partial \bar{\theta}^i} C(\bar{\theta}) P1_k C_k^T + C_k \Pi^{(i)} C_k^T + C_k P1_k \frac{\partial}{\partial \bar{\theta}^i} C^T(\bar{\theta})\right]\Bigg|_{\bar{\theta}^i = \hat{\theta}^i_{k|k-1}} &(2.200)\\
\kappa_k^{(i)} &= \frac{\partial}{\partial \bar{\theta}^i}\left[\left(A(\bar{\theta})\overline{P1}(\bar{\theta})C^T(\bar{\theta})\right)\bar{S}^{-1}(\bar{\theta})\right]\Bigg|_{\bar{\theta}^i = \hat{\theta}^i_{k|k-1}} \\
&\left[\frac{\partial}{\partial \bar{\theta}^i} A(\bar{\theta}) P1_k C_k^T + A_k \Pi^{(i)} C_k^T + A_k P1_k \frac{\partial}{\partial \bar{\theta}^i} C^T(\bar{\theta})\right]\Bigg|_{\bar{\theta} = \hat{\theta}_{k|k-1}} \times S_k^{-1} \\
&- K_k \sigma_k^{(i)} S_k^{-1} &(2.201)\\
\Pi_k^{(i)} &= \frac{\partial}{\partial \bar{\theta}^i}\left[A(\bar{\theta})\overline{P1}(\bar{\theta})A^T(\bar{\theta}) + Q^v - \overline{K}(\bar{\theta})\bar{S}(\bar{\theta})\overline{K}^T(\bar{\theta})\right]\Bigg|_{\bar{\theta}^i = \hat{\theta}^i_{k|k-1}} \\
&= \left[\frac{\partial}{\partial \bar{\theta}^i} A(\bar{\theta}) P1_k A_k^T + A_k \Pi^{(i)} A_k^T + A_k P1_k \frac{\partial}{\partial \bar{\theta}^i} A^T(\bar{\theta})\right]\Bigg|_{\bar{\theta} = \hat{\theta}_{k|k-1}} - \\
&\kappa_k^{(i)} S_k K_k^T - K_k \sigma_k^{(i)} K_k^T - K_k S_k \kappa_k^{(i)T} &(2.202)
\end{aligned}$$

Equations (2.200), (2.201) and (2.202) are coupled matrix Riccati equations. They have to be executed recursively at each step in time, till the sequence $\kappa_k^{(i)}$ converges,[31] to yield a good approximation for the term $\partial K/\partial \hat{\theta}^i$. Ljung has proven in [33] that if:

1. the matrix M_k is replaced by M_k^*, whose i-th column is given by:

$$M_k^{*(i)} = M_k^{(i)} + \kappa_k^{(i)}\left(z_k - C_k \hat{x}_{k|k-1}\right) \qquad (2.203)$$

2. the matrix S_k^{-1} in equation (2.184) is replaced by an identity matrix

[31] One iteration might be enough, if the difference between the old parameter estimate and new parameter estimate is sufficiently small.

the estimates $\hat{\theta}$ converge with probability 1 to a minimum of (2.195). This assertion holds only, if "*... the algorithm is complemented with a projection facility to keep $\hat{\theta}$ in a compact subset of*

$$D_S = \{\ \theta \mid (A(\theta), C(\theta)) \textit{ detectable and } (A(\theta), Q^v) \textit{ stabilizable}\}"$$

2.5.1 Conclusions

Ljung's convergence analysis [33] shows the possible causes of divergence and bias in the EKF algorithm. The reason for divergence can be traced down to the fact that there is no coupling term between the Kalman gain and the parameter vector θ. With other words, a change in the parameter vector θ has no direct effect on the Kalman gain K, but the *optimal* Kalman gain does in general[32] depend on θ. To overcome this problem, Ljung suggests a modification in the algorithm. The modification assures that the parameter estimates will converge to a local minimum of $V(\theta)$ (see eqn. 2.195), provided that the algorithm is complemented with a projection facility that guarantees $\hat{\theta}$ to stay within D_S. A disadvantage of the MEKF is the high computationally load imposed by the requirement to solve three coupled matrix Riccati equations (2.200–2.202) at each step in time. Further problems associated with the MEKF are the questions, on how the algorithm can be complemented with the required projection facility to keep $\hat{\theta}$ in D_S, and how the set D_S can be determined for any kind of state space model.

What can be concluded from the discussion above is, that the suggested modification is quite promising, but also that the MEKF, as given in [33], is not a *ready-to-apply* filter algorithm and that some work is needed to implement this modified Extended Kalman Filter. The author has made the step to implement the MEKF. How this has been done, the difficulties and problems encountered, are described in the next chapter.

[32]Applications have shown that the EKF algorithm converges to the true parameter vector when K is independent of θ.

Chapter 3

Implementation of the MEKF

3.1 Introduction

The use of the EKF as a on-line parameter estimator for linear, discrete-time systems is well known and widely spread. Yet, the EKF algorithms has a major disadvantage: the filter is likely to diverge or give biased estimates. Lennart Ljung suggests in [33] a modification in the EKF algorithm, that promises to improve the convergence behavior of the filter considerably. Motivated by Ljung's results, the author implemented this encouraging filter algorithm, in order to gain experience with the modified EKF and to explore for applicability of this parameter estimator.

The algorithm is implemented in a FORTRAN software package, described in the next section. Before the identification process can be started at time k_0, several matrices and vectors need to be initialized. How this initial values should be chosen, for the filter to perform satisfactory is described in Chapter 3.3. The input/output data for the parameter estimator is generated by a simulated linear, discrete-time system, randomly excited by process noise. The noise corrupted output signal of the simulated system is fed into the MEKF. The developments in [33] are based on correct noise assumptions. Therefore, special care has been taken, to assure that the noise sequences used for testing the filter were ideally white, normally distributed sequences. Chapter 3.4 is devoted to this subject. Finally, in Chapter 3.5 of results of performance tests of the MEKF on simple systems are presented.

3.2 The Algorithm

This section describes the computer programs for parameter estimation of linear, discrete-time systems based on the MEKF algorithm (see Chapter 2.5). Before going into the details of the more important routines, the main features of the developed software shall be listed:

- All routines are strictly modular, i.e. they consist of small subroutines, to enhance flexibility and simplify debugging.
- The routines are preceded by headers, that describe the function of the particular program.
- In order to improve readability, all subroutines are given self explanatory names.
- The programs are written in FORTRAN 77.
- The software package is device independent, i.e. it does not use any other packages, libraries[1] etc.
- A broad class of systems that range from scalar SISO systems, up to MIMO systems with ten inputs and ten outputs, can be treated.
- To enhance numerical stability, all operations are executed in *double precision.*

For the performance tests reported in this thesis, the estimation programs were installed an a VAX9900 computer. Most routines of the software developed do not need any further description; they are simple and self explanatory. There are however three exceptions to this, that deserve special consideration. The first one is the main program MEKF, and the second one is the subroutine P32MMODIFY

[1]The main reasons for not using other software packages were availability and the desire to maintain device independency.

which modifies the M^* matrix, as suggested by Ljung. To change the M^* matrix, $\kappa_k^{(i)}$ (cf. eqn. 2.203), the solution of the three coupled matrix Riccati equations (2.200)-(2.202) needs to be computed. This computationally most costly problem is solved by P31COMPKAPPA; so this routine shall find special consideration here.

The main program MEKF

The program MEKF consists of two main parts. In the first part, the system state vector, the estimated state vector, the estimated parameter vector, as well as the covariances P_1, P_2 and P_3 are initialized. This is done by the subroutine U02INITI. Under normal operation, this part is executed only once, at the startup of the estimation process. All other data, required to run the parameter estimation routines, are contained in "data" lines in the preamble of the main program. To set the program up for a different system, these "data" lines need to be changed accordingly. The following information is needed:

- number of system states
- number of inputs
- number of outputs
- number of parameters
- entries of system matrices A, B and C
- entries of model matrices $A(\theta)$, $B(\theta)$ and $A(\theta)$
- entries of system noise covariance matrix Q^v
- entries of measurement noise covariance matrix Q^e

The second part of the main program produces the output data of the simulated system, and simultaneously generates estimates for state vector and parameter vector. This part is executed recursively, till the program is terminated by a stopping rule, which consists in the simplest case of a *loop-counter* to *maximum-number-of-iterations* comparison.

Before the program MEKF can be applied to parameter estimation problem, the user has to decide on a parameterization, which constitutes the interrelation between the parameter dependent matrices A, B, C and the parameter vector θ.

The following subroutines are dependent on the type of parameterization chosen and therefore need to be changed, accordingly.

U04DDERI	computes the matrix D, where D is defined be equation (2.168)
U05MDERI	computes the matrix M (not $M^\star$), where M is defined be equation (2.168)
U06CDERI	computes the derivative of matrix C with respect to the ith parameter
U07ADERI	computes the derivative of matrix A with respect to the ith parameter
U08UPDATE ABC	updates $A(\theta)$, $B(\theta)$, $C(\theta)$ so that they correspond to the new parameter estimate

After the utility routines above are recoded, to account for the parameterization chosen, the program is prepared for joint state vector and parameter vector estimation. The flow charts of the main program (see Figure 3.1 and Figure 3.2) show that, due to modular programming, a simple structure is maintained.

The subroutine P32MMODIFY

The subroutine P32MMODIFY modifies the M matrix, as suggested by Ljung [33]. The "new" M-matrix $M^\star$ is given by:

$$M_k^{\star(i)} = M_k^{(i)} + \kappa^i \left(z_k - C(\hat{\theta}_{k|k-1})\hat{x}_{k|k-1}\right) \tag{3.1}$$

where: $M_k^{\star(i)}$	denotes the ith column of the modified M matrix
$M_k^{(i)}$	denotes the ith column of the original M matrix, as computed by the subroutine U05MDERI
κ^i	is the approximate solution of the three coupled matrix Riccati equations (2.200) (2.202), where all derivatives have been taken with respect to the ith parameter, evaluated at the current estimate
$\left(z_k - C(\hat{\theta}_{k|k-1})\hat{x}_{k|k-1}\right)$	is the residual, or *one-step-ahead-prediction* error vector at time k

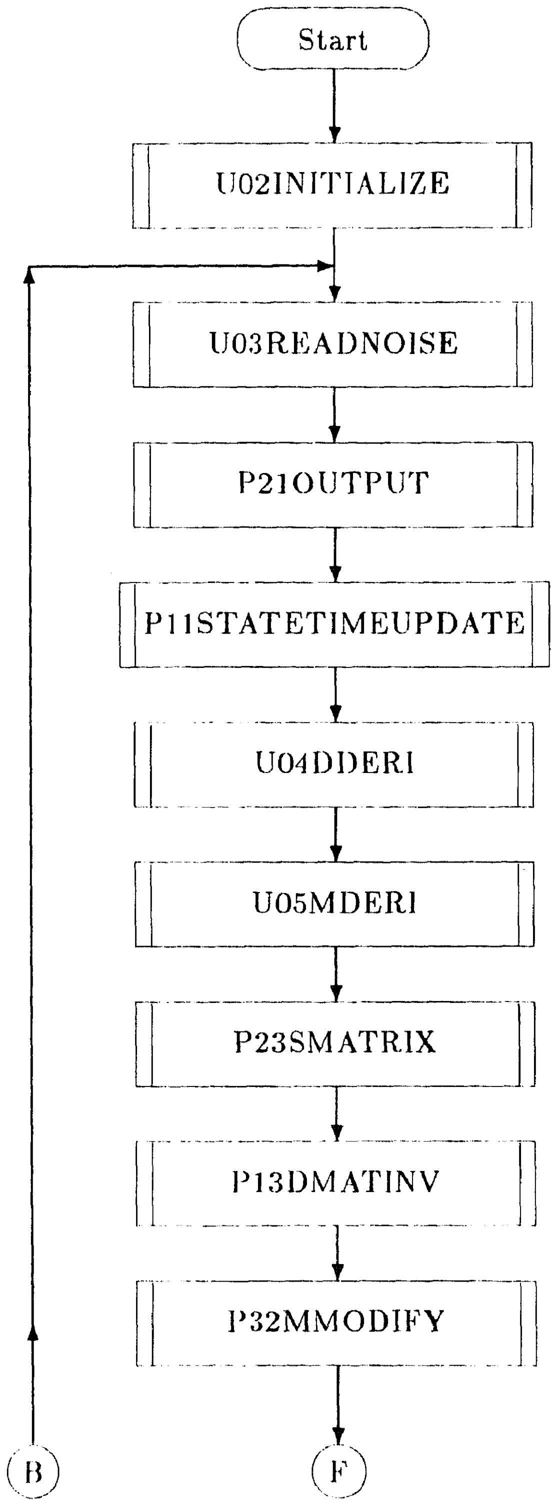

Figure 3.1: Main Program MEKF Flowchart Part 1

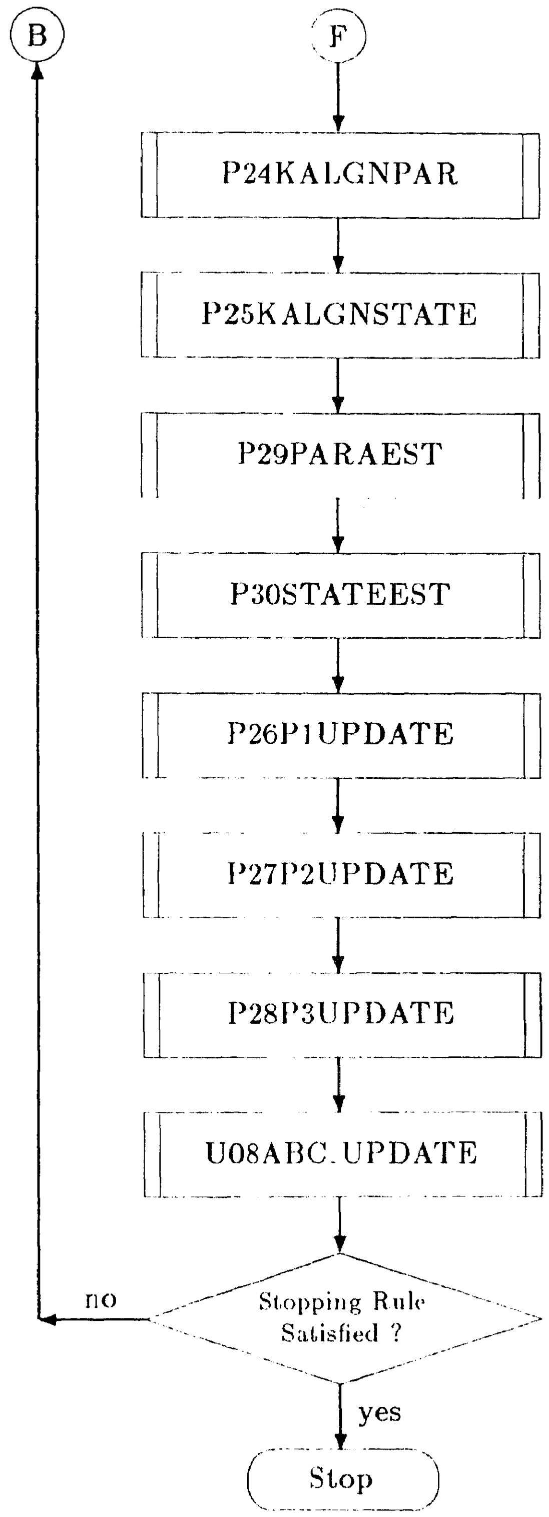

Figure 3.2: Main Program MEKF Flowchart Part 2

P32MMODIFY computes first the residual $\left(z_k - C(\hat{\theta}_{k|k-1})\hat{x}_{k|k-1}\right)$ and then calls the subroutine P31COMPKAPPA, which calculates κ^i. This second step has to be executed separately for each entry in the parameter vector; P32MMODIFY sets the parameter pointer appropriately. The ith column of the M matrix is changed, as soon as a new κ^i becomes available. The subprogram P32MMODIFY is terminated after all columns $M^{*(i)}$ $(i = 1, \ldots, np)$ have been computed.

The subroutine P31COMPKAPPA

The subroutine P31COMKAPPA generates the solutions of the three equations (2.200)–(2.202). In the tests conducted by the author it was observed that it is in general not sufficient to compute the three coupled matrix equations, as suggested by Ljung, only once per step in time. If the difference between the previous and the current parameter estimates is large,[2] the algorithm (2.200)–(2.202) takes approximately ten[3] iterations to converge. Although the matrix κ^i is sufficiently close to the term $[\partial K/\partial\theta^i]$ after just one iteration, if the changes in the parameter estimates are minute, the number of iterations per step in time is kept constant at eleven.

If the covariance matrix *P1* is ill conditioned, i.e. the relations between the eigenvalues of *P3* are large, the algorithm (2.200)–(2.202) shows either very slow convergence or even divergence. In these cases, the computed κ^i is of no value and one wants to discard it, that is leave the corresponding ith column of the M matrix unchanged. This is readily achieved by setting κ^i to zero, whenever divergence has been detected. The question that arises here is, how divergence *can* be detected. To discuss this, consider for simplicity that κ^i is scalar valued. When κ^i converges uniformly to the optimum value $\partial K/\partial\theta^i$, one simply has to check, whether the increments (or decrements) of the κ^i sequence are descendent. Unfortunately, however, the solutions of the matrix equations (2.200)–(2.202) do often not converge

[2]This will usually be the case at the startup of the estimation process

[3]This could be observed for the small scale estimation problems reported in this thesis, but this rule of thumb might not hold for other problem classes.

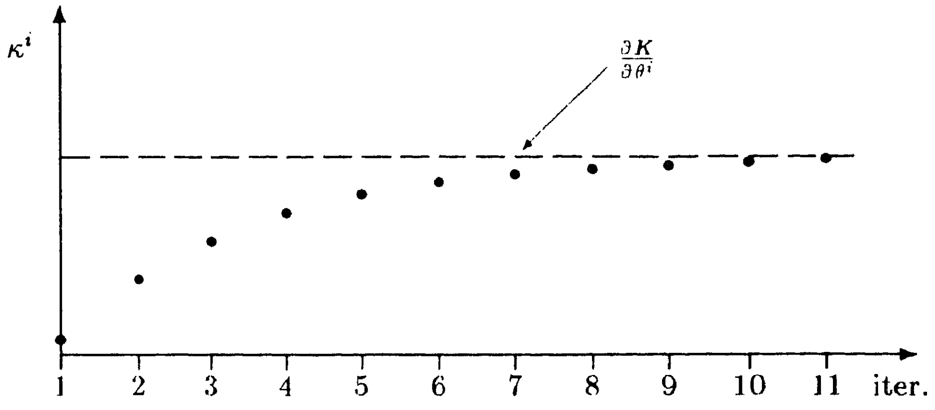

Figure 3.3: Uniform Convergence of κ^i to $\partial K/\partial\theta^i$

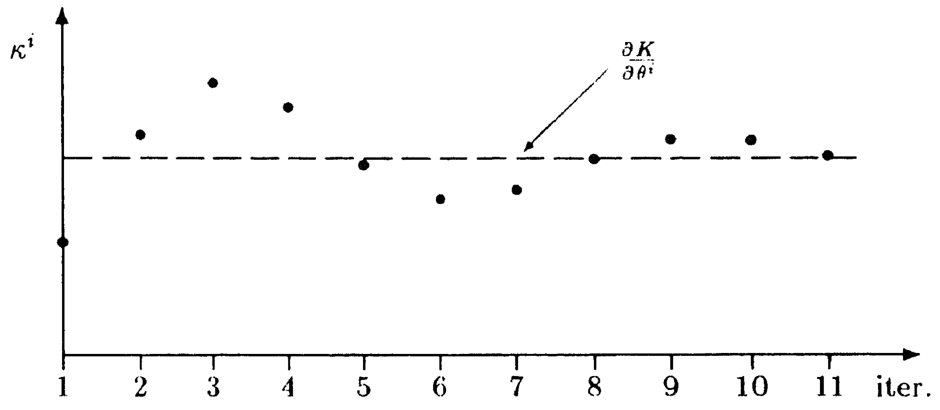

Figure 3.4: Oscillatory Convergence of κ^i to $\partial K/\partial\theta^i$

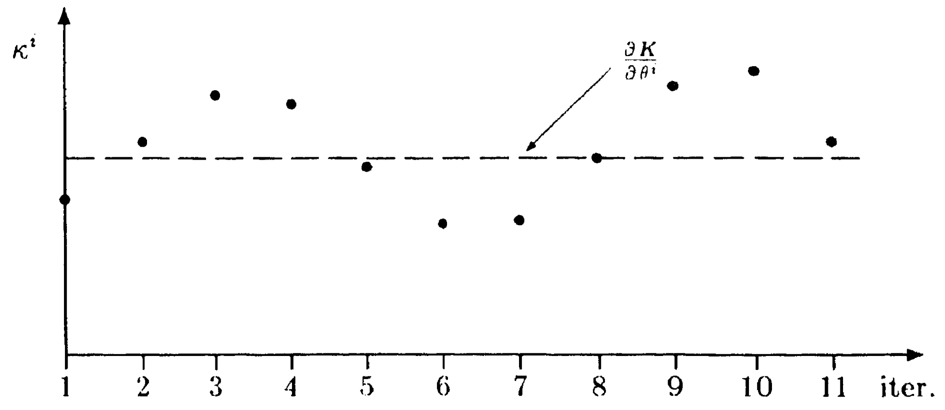

Figure 3.5: Oscillatory Divergence of κ^i

uniformly (if they converge at all), but in form of a damped oscillation, as shown in Figure 3.4. To differentiate between oscillatory convergence and divergence[4] a more sophisticated convergence check is needed.

The program P31COMPKAPPA detects divergence by comparing sum of the increments for the first six iterations HALFSUM, with the respective sum for all iterations. If the latter sum is larger than two times HALFSUM, P31COMPKAPPA considers this as an occurrence of divergence and consequently sets κ^i to zero, so that the respective column in M remains unchanged. To avoid numerical *under/overflow*, the algorithm is complemented with a boundary check. The subprogram P31COMKAPPA discards κ^i and terminates, whenever one of the entries in κ^i, σ^i or Π^i is beyond certain limits. There is no guarantee that the implemented convergence check leads to correct decision, in all cases. However, it worked sufficiently well in the tests undertaken. For additional information about this subprogram, a flow chart of P31COMKAPPA is given with Figure 3.6.

[4]Note, that in both cases the increments, or decrements, are increasing for the first few iterations.

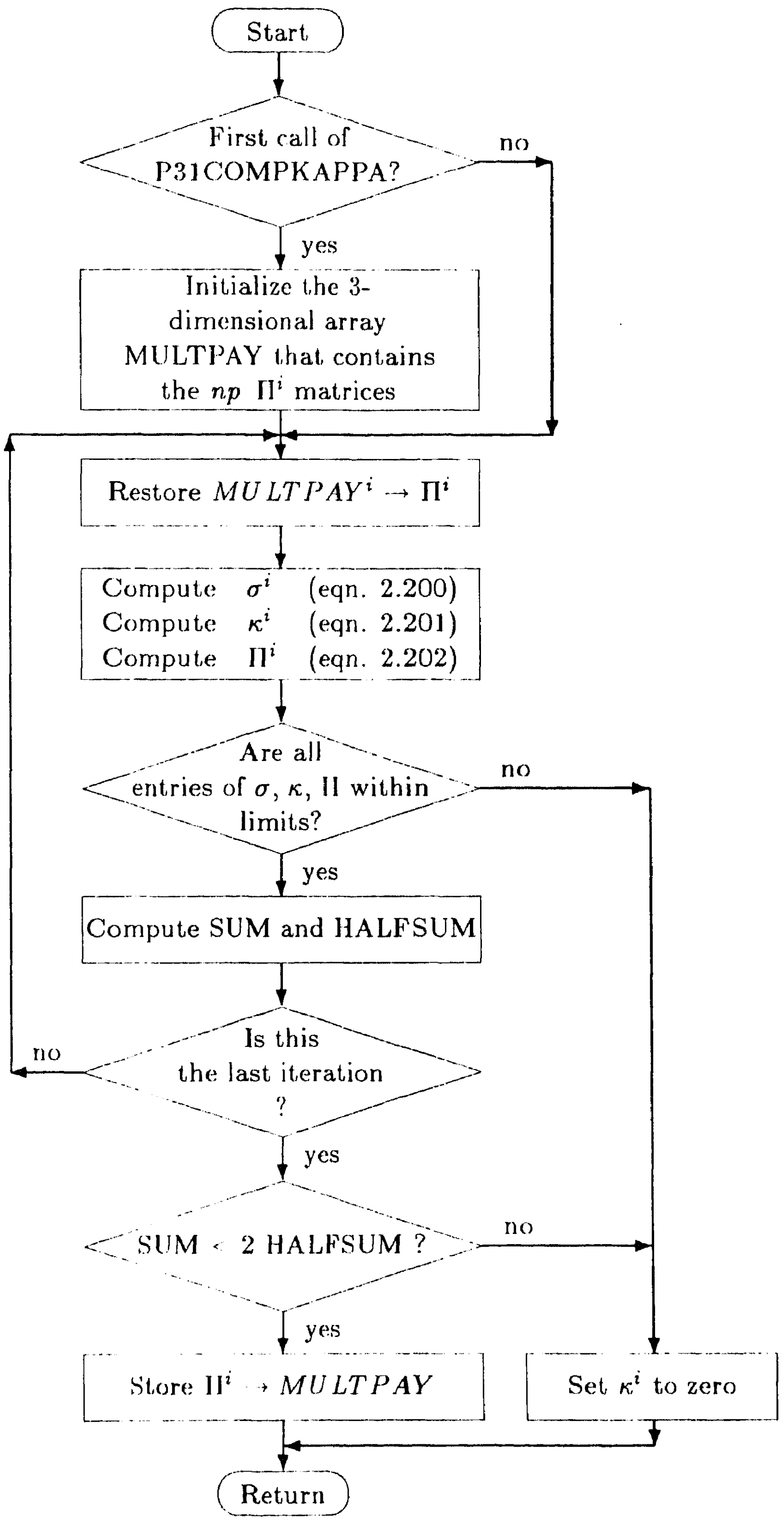
Start
First call of P31COMPKAPPA?
no
yes
Initialize the 3-dimensional array MULTPAY that contains the np Π^i matrices
Restore $MULTPAY^i \rightarrow \Pi^i$
Compute σ^i (eqn. 2.200)
Compute κ^i (eqn. 2.201)
Compute Π^i (eqn. 2.202)
Are all entries of σ, κ, Π within limits?
no
yes
Compute SUM and HALFSUM
Is this the last iteration ?
no
yes
SUM < 2 HALFSUM ?
no
yes
Store $\Pi^i \rightarrow MULTPAY$
Set κ^i to zero
Return

Figure 3.6: Subprogram P31COMPKAPPA Flowchart

3.3 Filter Initialization

In the introduction to this chapter, it is stated that prior to the startup of the identification process, several matrices and vectors need to be initialized. These are specifically:

P1	state estimate covariance matrix
P2	state estimate/parameter estimate crosscovariance matrix
P3	parameter estimate covariance matrix
$\hat{\theta}$	parameter vector estimate
$\hat{x}$	system state vector estimate

The question of how to initialize the filter, is quite different for practical cases, then it is for testing the algorithm in a laboratory environment. If the filter is to be applied to physical systems, one often knows a good deal about the initial system state and parameter vector, e.g. such as the range a particular parameter can lay in. So, if information of this type is available, or can be obtained, it should be used and the covariances and the estimation vectors initialized, accordingly.

Ljung suggests in [33] that for cases where no a priori information is available, to choose the initial parameter estimate $\hat{\theta}_0$ to be zero and the associated error covariance matrix $P3_0$ to be $100 \cdot$ (variance of z). The author does not quite agree to this rule of thumb, because whether a zero initial parameter estimate is a good choice or not, depends on the parameterization chosen. It might be that $\hat{\theta}_0 = \underline{0}$ is not an element of the compact subset $\mathcal{D}_S$ (cf. Chapter 2.5), i.e. violates one of the constraints of the MEKF. A better choice should be, to initialize the parameter estimate $\hat{\theta}$ such that it lies close to the "center" of $\mathcal{D}_S$. One method to find such an interior point of $\mathcal{D}_S$, is to randomly select three $\boldsymbol{\theta} \in \mathcal{D}_S$ to form an triangle, and to set $\hat{\theta}_0$ to their center of mass. This method produces an initial parameter estimate which lies inside the subset $\mathcal{D}_S$, provided that the topology of $\mathcal{D}_S$ is "sufficiently" moderate.

Regarding Ljung's other suggestion, to initialize $P3$ with $100 \cdot$ (variance of z), the author does not know what the reasoning behind this rule is and why it should be a "... *good choice*". However, for the performance tests of the MEKF conducted by the author, it turned out to be an inadequate choice. The initial covariance $P3_0$ for which the filter worked best, was in magnitude about three powers of ten smaller than the one advised.

The problematic nature of the filter initialization is quite different for applications of the MEKF to simulated systems — as used for this thesis — because one actually has complete knowledge about system states, parameters and covariances and it is thus possible to initialize the filter ideally. This, however, would not reflect practical cases. So, the question is how much or little one should pretend to know. The author did not assume any information about the system states. Hence, a good choice for the initial state estimate is $\hat{x}_0 = \underline{0}$. The filter proved to be rather insensitive towards the choice of the covariance matrix associated with initial state estimate. Values in the range from 1 to 100 for the diagonal[5] entries of $P1_0$ worked best.

The choice of the initial parameter estimates is, as discussed above, dependent on the selected parameterization. For the tests reported in this thesis, the parameterization is in general such that $\hat{\theta}_0 = \underline{0}$ is permissible. So, unless stated otherwise, the initial parameter estimate is set to zero.

Because both the parameter estimate and the state estimate are chosen randomly, there is no justification to assume the corresponding error vectors other than uncorrelated. Therefore, the crosscovariance matrix $P2$ is initially set to zero.

The last matrix to be initialized, is the parameter estimation covariance matrix $P3$. The diagonal entries have to be chosen undesirable small ($0.0001 \rightarrow 0.025$), because for larger values covariance matrix $P3$ became singular, after a few iterations.

[5]All off-diagonal entries are set to zero, as initially the individual states are assumed to be uncorrelated.

It is not obvious from the MEKF algorithm why this can happen. The author observed in tests, that a "large" covariance $P3$ produces via equation (2.188) a large $P2$, that in turn leads to a large gain matrix L (cf. eqn. 2.184 in Table 2.5). From equation (2.189) it is observed that the updated covariance $P3_{k+1}$ is given by $P3_k$ minus the term $L_k S_k L_k^T$, which can become larger than $P3_k$ for some k, if the initial covariance $P3_0$ was large. On the other hand, to choose the covariance $P3_0$ very small, is not a remedy to this problem, because this "tells" the filter that $\hat{\theta}_0$ is a good estimate for the parameter vector θ that needs only minor changes. This results in a slow convergence of the algorithm to the true parameter vector, that is in general not acceptable. Hence, the only critical part of the filter initialization is the choice of covariance $P3_0$, which requires fine tuning to find values that result in both, fast convergence and a stable filter algorithm. Unfortunately, for many cases it is not possible to find such a initial covariance matrix $P3_0$. In Chapter 4 the author suggests some modifications to the filter algorithm, to overcome this problem.

3.4 Noise-Sequences Used for Testing the Filter

Ljung stated in [33] that the parameter estimates the MEKF produces will only converge to the system parameters if the noise assumptions are correct. Also, the performance of the MEKF as a parameter estimator can not, due to the randomness of the estimation process, be judged from the results of a single test run. Valid conclusions about the filter performance can be drawn from the outcome of *Monte Carlo* simulations, which are performed by letting the identification process rerun n-times[6] with different noise sequences, but under otherwise identical conditions. This clearly states, that to test the filter algorithm, a set of independent, zero-mean, white noise sequences is needed.

[6] n should be at least 15–25 for the simulation to produce valid results

The author experimented with different random number generators. The best results were achieved with the noise generator RANDUGEN, the program listing of which is given in the Appendix C. The random number generator RANDUGEN, initialized with $SEED = 824064364$ and $SIG = 1.0$, was used to generate 30000 samples, randomly split up into fifty files to 600 samples each. These noise sequence files are named N SEQ10,...,N SEG59.

The computation of the means and the variances of these noise sequences revealed that they were not exactly zero-mean and their variances were slightly off, from one. Subtracting the mean of a particular sequence from each number of that file, results in ideally zero-mean sequences. In addition, the variance of each number sequence was normalized to one by multiplying each number with the factor $\sqrt{1/\varsigma^2}$, where ς^2 is the variance of the particular sequence, prior to the normalization process. This yields zero-mean sequences with variance one. To determine the whiteness of the number sequences estimates for the autocorrelation sequences were also computed utilizing the following formula:

$$c_{nn}(m) = \frac{1}{600 - |m|} \sum_{j=0}^{600-|m|-1} n_{(j)} n_{(j+m)} \tag{3.2}$$

where the n's stand for the numbers of a particular noise sequence. Representative for all other number sequences, the first six values of the autocorrelation function associated with N_SEG13 are given in Table 3.1.

Table 3.1: Autocorrelation function associated with noise sequence N SEQ13

m	$c_{nn}(m)$
0	+1.003
1	+0.013
2	−0.028
3	+0.001
4	+0.009
5	−0.030

Ideally, one would expect the autocorrelation function to be one for $m = 0$, and to be zero for $m \neq 0$. Similar results were achieved for the crosscorrelation functions, where the ideal value is zero, for all m.

The data files modified in this way N_SEQ10,... ,N_SEQ59 constitute fifty noise sequences that are a good match to the noise assumptions made in the derivation of the MEKF.

3.5 An Example of the Present Method

Now that the filter algorithm is implemented (Chapter 3.2), the question on how to initialize the filter is answered (Chapter 3.3) and (almost) ideal noise sequences are generated (Chapter 3.4), the MEKF is ready for *Monte Carlo* simulations. Naturally, one starts with a simple estimation problem.

TEST A:

The simulated system is given by:

$$x_{k+1} = 0.5\,x_k + u_k + v_k \tag{3.3}$$

$$z_k = 2.0\,x_k + e_k \tag{3.4}$$

with noise statistics:

$$\begin{aligned} v_k &\sim N(0, 0.5) \\ e_k &\sim N(0, 0.5) \end{aligned}$$

Assume everything, but the A matrix (here a scalar) is known about the system. The uncertainty about A is modeled as:

$$x_{k+1} = \theta\; x_k + u_k + v_k \tag{3.5}$$

$$z_k = 2.0\,x_k + e_k \tag{3.6}$$

where θ is a parameter to be estimated, using the MEKF. The filter is initialized as follow:

$$\begin{array}{lllllllll} P1_0 & = & [2.0] & \hat{x}_0 & = & [0.0] & Q^v & = & [0.5] \\ P2_0 & = & [0.0] & \hat{\theta}_0 & = & [0.0] & Q^e & = & [0.5] \\ P3_0 & = & [0.015] & & & & & & \end{array}$$

where $P3_0$ has to be chosen undesirably small (recall discussion in Chapter 3.3 concerning this problem), because for larger initial values, covariance $P3$ becomes singular, at some time instant k. The choices for $\hat{\theta}_0$ and $\hat{x}_0$ following directly from the reasoning in Chapter 3.3. As explained in Chapter 3.4, the filter performance can, due to the randomness of the estimation process, not be judged from the results of a single run. Hence, the parameter estimation process was restarted 25 times, using the following noise sequences:

Runs	System Noise	Measurement Noise
Run 1	N_SEQ10	N_SEQ11
Run 2	N_SEQ12	N_SEQ13
Run 3	N_SEQ14	N_SEQ15
⋮	⋮	⋮
Run 25	N_SEQ58	N_SEQ59

Figure 3.7 shows the averaged parameter estimates of these 25 runs.

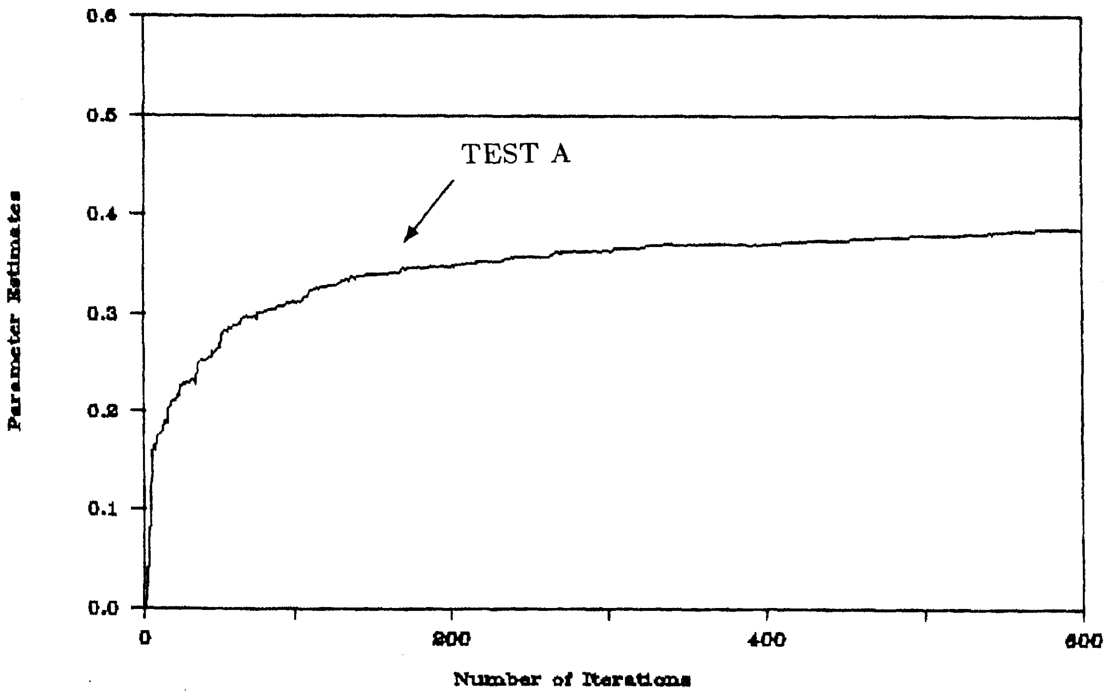

Figure 3.7: Parameter Estimates Test A (Average of 25 Runs)

In order to examine how sensitive the algorithm is towards the system and measurement noise sequences, the variances for all parameter estimates from *Run 1* up to *Run 25* were computed; the results of which are given with Figure 3.8. A

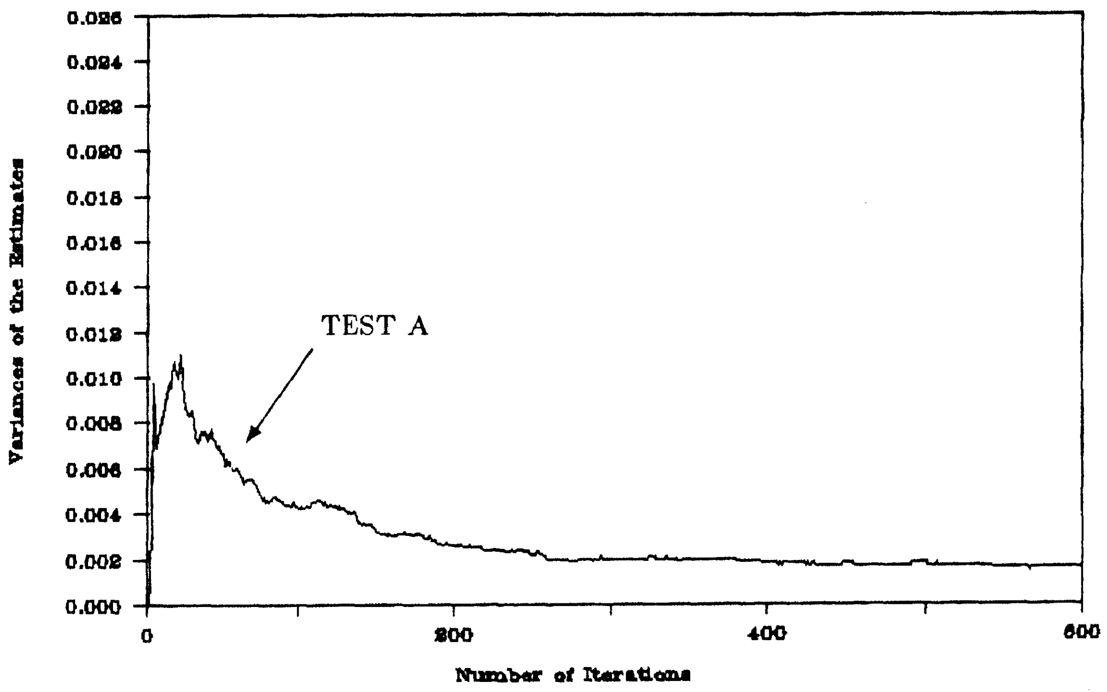

Figure 3.8: Variances of Parameter Estimates Test **A**

large variance at a certain time instant k indicates, that the corresponding estimate (cf. Figure 3.7) is inconsistent.

It is observed from Figure 3.7 that the filter performance is far from being satisfactory. The parameter estimates are even after 600 iterations, not close to the true parameter value of 0.5. This filter behaviour is mainly due to the choice of initial covariance $P3_0$, that "tells" the filter, that $\hat{\theta}_0 = 0$ is a good estimate for θ, that does not need much change. Consequently, the Kalman gain factor L, for updating the parameter estimates, becomes after just a few iterations so small, that the parameter estimates practically "freeze" at a certain value. The parameter estimate error covariance for the given problem can not be initialized with a value larger than 0.015, because otherwise $P3_k$ becomes singular for some k, which results in a numerical *blow-up* of the estimation algorithm. The author suspected that yet another reason could cause the parameter estimates to being off the true value: the MEKF inherently produces biased estimates. To find out if this is one of the reasons for the MEKF to perform poorly, the author conducted another test (TEST B), very similar to the previous one. The only difference is that this time $\hat{\theta}_0$

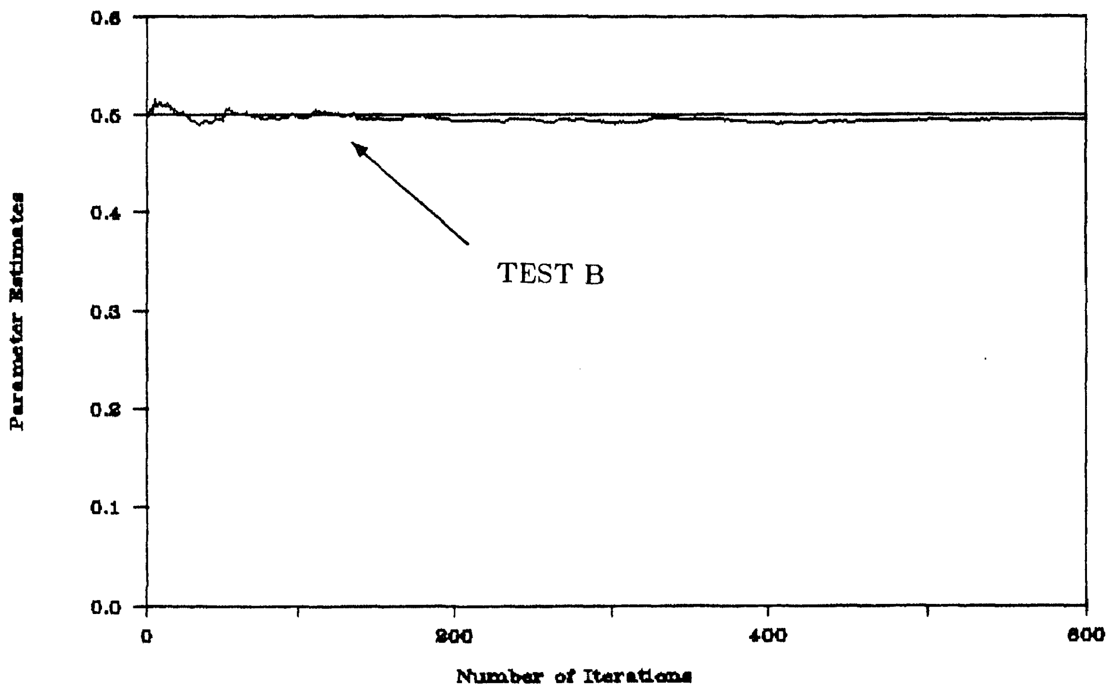

Figure 3.9: Parameter Estimates Test B (Average of 25 Runs)

is initialized with the true parameter value of 0.5. If the filter inherently produces biased estimates, one would expect the estimates of TEST B to be biased. From Figure 3.9 the reader can observe, that this is not the case.

3.6 Conclusions

In this chapter, the actual implementation of the modified Extended Kalman Filter has been discussed. General facts about the developed software, as well as detailed information about some of the more important routines, are given in Chapter 3.2. The reader has learned about the filter initialization and the preparation of the noise sequences used in this thesis, in Chapter 3.3 and Chapter 3.4, respectively. Finally, convergence results of a *Monte Carlo* simulation for a single parameter case are presented in Chapter 3.5.

The results of the simulations conducted were rather unsatisfactory. The filter, though stable, showed very slow convergence and the parameter estimates stayed biased. From the results of a further test, it could be concluded, that the MEKF does not inherently produce biased estimates, but that the slow convergence is attributed to the small initial parameter covariance $P3_0$, necessary for the filter algorithm to be stable. Because a parameter estimator with properties as shown by the MEKF is useless, the author experimented with several heuristic method, to improve the filter's behavior. The next chapter is devoted to this topic.

Chapter 4

Investigation and Development of Various Methods to Improve Stability and Rate of Convergence of the MEKF Based on Single Parameter Case

4.1 Introduction

The application of the MEKF to a simple parameter estimation problem, has shown that the "plain" filter algorithm, as presented in [33], does not meet the requirements of a parameter estimator. What is desired, is an estimator that produces unbiased estimates, converges fast to the true parameter values and shows little variation in its estimates with respect to system and measurement noise. To upgrade the filter performance, several heuristic methods were applied to the MEKF algorithm. Most of the methods presented in this chapter, were developed by the author; some were adopted from the literature and tailored to meet the specific requirements of the MEKF. None of this techniques is based on any theoretical developments. They are all based on heuristic discussions.

The methods to improve the MEKF properties are applied to the single parameter case from Chapter 3 (TEST A). The reason for doing so, are threefold: first, it makes the results comparable, secondly, it simplifies the implementation of the developed techniques, and third it enhances the clearness of the methods applied. The fact, that these methods are only applied to first order systems, does not mean that their usefulness is limited to single parameter cases. It is expected that they prove to be equally effective when applied to higher order problems.

4.2 Decelerated Convergence of $\mathbf{P}_3$

In Chapter 3 it is discussed that the poor convergence properties of the MEKF might mainly be attributed to the small initial parameter estimates covariance matrix $P3_0$. Such a small $P3_0$ "tells" the filter that the initial guess $\hat{\theta}_0$ for the system parameter is of high quality, i.e. already so close to the true value θ_0, that no large changes in the estimate are needed nor desired. As a result of this, the filter "freezes" the parameter estimates, before they converged to the value of the system parameter. One way to circumvent this problem, is to slow down the convergence of the covariance matrix $P3$, because it is this matrix that reflects the quality of the parameter estimates. By doing so, the parameter updating process is kept active for more iterations, in compensation for the inadequate initialization of covariance $P3$. Recall, that the updating equation for $P3$ is given by:

$$P3_{k+1|k} = P3_{k|k-1} - L_k\, S_k\, L_k^T \tag{4.1}$$

So, it is the term $L_k S_k L_k^T$ that diminishes the covariance $P3$.[1] To slow this process down, the parameter update gain matrix L should be rescaled with a reduction factor rf ($|rf| > 1.0$), at least for the first few iterations. However, the parameter update equation is still given by:

$$\hat{\theta}_{k+1|k} = \hat{\theta}_{k|k-1} - L_k \left(z_k - C(\hat{\theta}_{k|k-1})\hat{x}_{k|k-1}\right) \tag{4.2}$$

As a result of the modification in the update equation for covariance $P3$, one also expects, that $P3_0$ can be chosen larger, than in the case of the original update

[1] Note that LSL^T has only positive entries for all L, because the matrix S is p.d.

equation. In order to examine, whether this heuristically developed method works in practice, *Monte Carlo* simulations with different reduction factors (rf) were performed. The initial covariance $P3_0$ was, as in the previous tests, set to the highest value possible, for the filter algorithm to be stable. The simulated system is given by equations (3.3) and (3.4). Noise statistic and noise sequences used for the individual runs, are as before. The system is modeled by equations (3.5) and (3.6), where the parameter θ is to be estimated using the MEKF algorithm of Chapter 2.5, but where the $P3$ covariance update equation is given by:

$$P3_{k+1|k} = P3_{k|k-1} - \frac{L_k}{rf_k} S_k \frac{L_k^T}{rf_k} \tag{4.3}$$

TEST C, TEST D and TEST E

The reduction factor for test C was arbitrarily chosen to be:

$$rf_k = 1.2 \qquad \forall\, k, \quad k = 1, \ldots, 600$$

The filter is initialized as follow:

$$\begin{array}{lll} P1_0 = |2.0| & \hat{x}_0 = |0.0| & Q^v = |0.5| \\ P2_0 = |0.0| & \hat{\theta}_0 = |0.0| & Q^e = |0.5| \\ P3_0 = |0.02| & & \end{array}$$

which is almost identical with the initialization for TEST A. The only difference is that covariance $P3$ could be initialized with a slightly larger value. From Figure 4.1 it is observed that, the method to decelerate the convergence of $P3$ does have the desired effect on the filter's convergence properties.

Motivated by this result, the author conducted several additional simulations operating with different reduction factors. The results for two of these simulations (TEST D, with a constant rf of 1.7, and TEST E, with a constant rf of 2.2) are also given in Figure 4.1. As one can observe from Figure 4.1, a larger reduction factor causes the filter to converge faster, but also causes the variance in the estimation process to increase (see Figure 4.2), which is undesirable. Recall, that the reduction

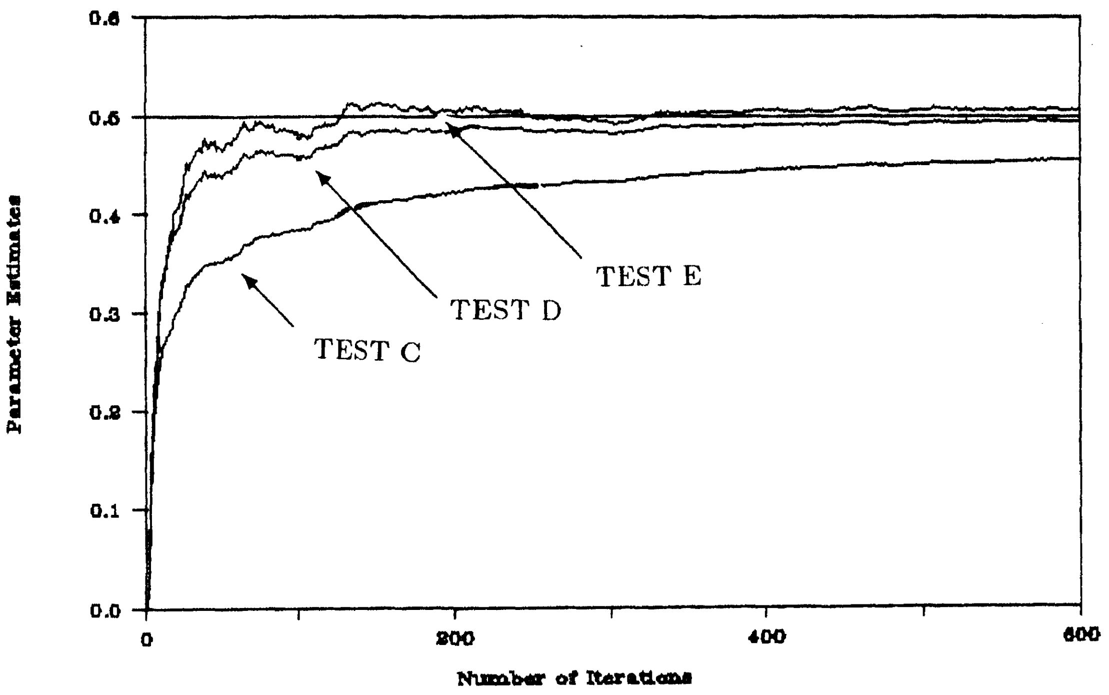

Figure 4.1: Parameter Estimates Test C, D and E

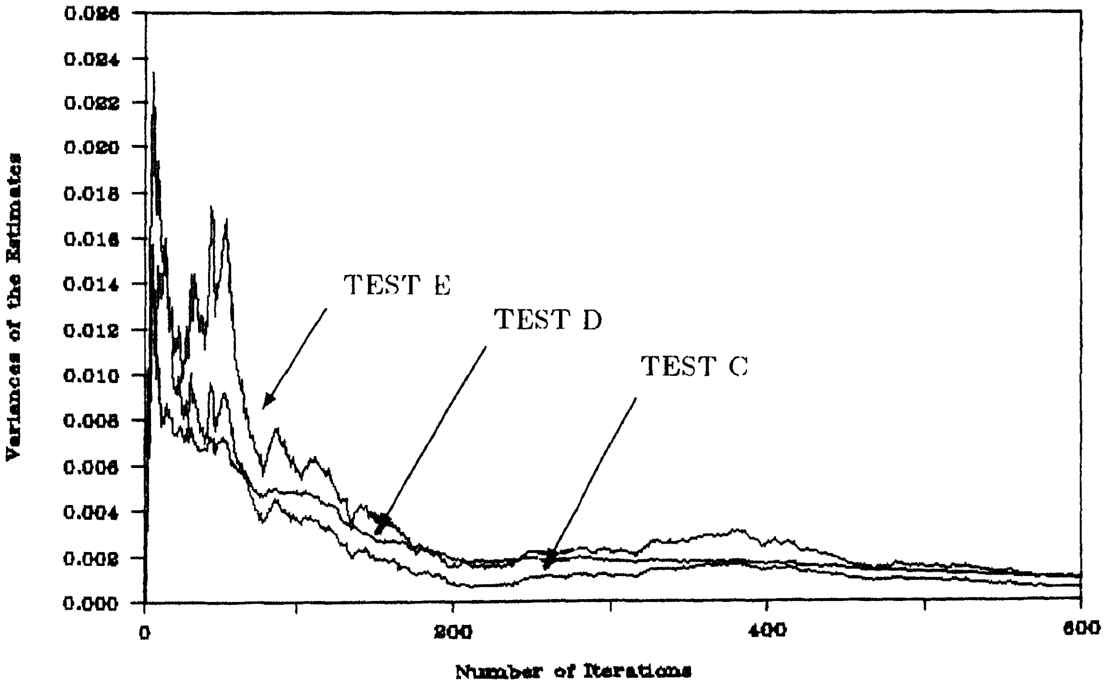

Figure 4.2: Variances of Parameter Estimates Test C, D and E

factor was introduced, to compensate for the initial parameter estimate covariance $P3$, that had to be chosen smaller than desired in order to prevent $P3_k$ from becoming singular. Hence, it should be sufficient to operate with a large reduction factor only for the first few iterations, and to decrease rf for later iterations, where the covariance $P3$ is small, independent of the initial value. So, in order to achieve fast convergence of the parameter estimates without trading this property in for a high sensitivity towards the noise sequences, the reduction factor should be large at the beginning of the estimation process and decrease during time to a value slightly larger than one. To verify this argument a for this method a final test (TEST F) was carried out.

TEST F

System, model, noise sequences and filter initialization are as in the previous tests. The reduction factor rf is chosen to be:

$$rf_k = \begin{cases} 2.2 & for\ k < 9 \\ 2.2 - 0.02\,k & for\ 10 \leq k \leq 60 \\ 1.0 & for\ k > 60 \end{cases}$$

Comparing the resulting parameter estimates and associated variances given in Figure 4.3 and Figure 4.4, respectively, with the outcomes of TEST E, where rf was kept constant at 2.2, reveals, that the reasoning above was correct, although the variances are almost the same.

Conclusions

A comparison of the results obtained by applying the "plain" MEKF algorithm (see Chapter 3.5), with the ones achieved using the described method, clearly shows the

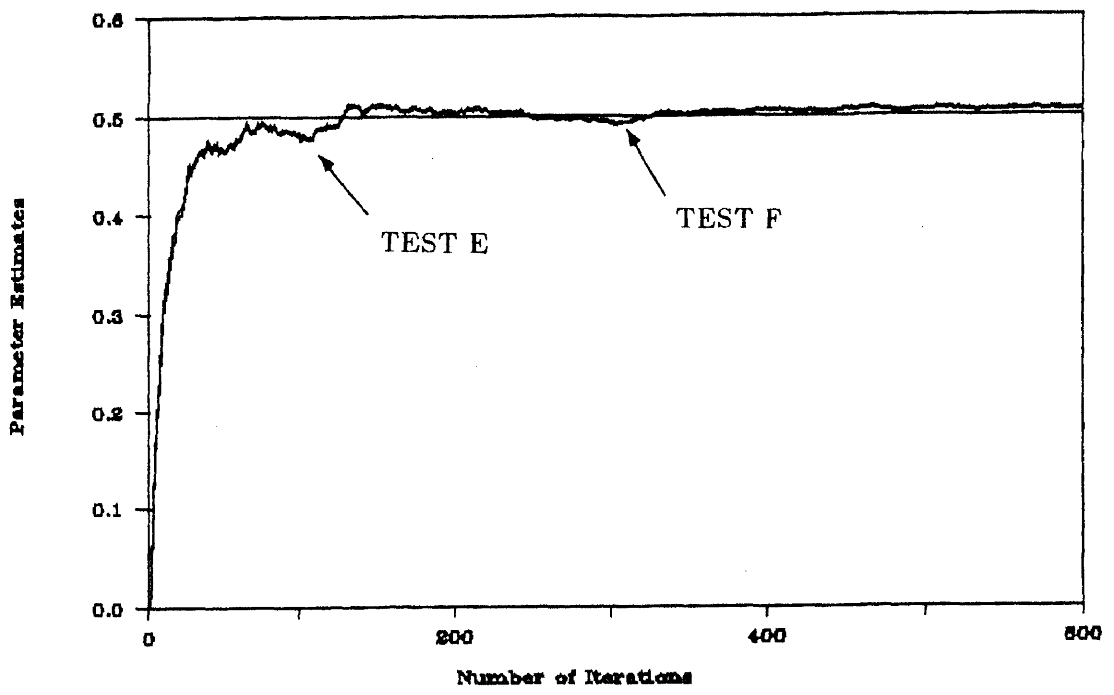

Figure 4.3: Comparison of Parameter Estimates Gained for const. *rf* (TEST E) and for time-varying *rf* (TEST F)

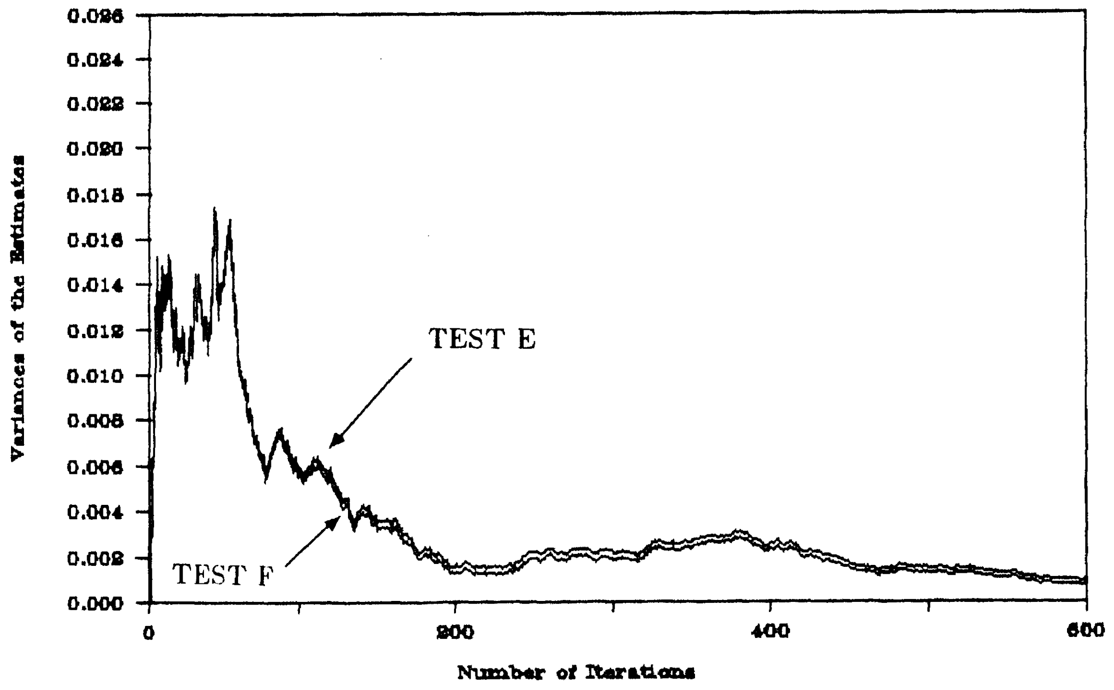

Figure 4.4: Comparison of Variances of Parameter Estimates from Test E and F

usefulness, that lies in a decelerated convergence of covariance $P3$. The author does not claim to have found the optimal method with the few simulations conducted. Further tests, supplemented with theoretical developments will be necessary, before this method can be applied to field problems. It should also be mentioned, that for large reduction factors the estimation algorithm became unstable. The reason for this instability is attributed to the fact, that the parameter estimates were such that the $\left(A(\hat{\theta}), Q^v\right)$ was no longer stabilizable, i.e. the parameter estimates were no longer elements of $\mathcal{D}_s$ (see Chapter 2.5). So, to fully exploit the fast convergence this method can provide, the algorithm needs to be complemented with a facility to keep the parameter estimates inside of $\mathcal{D}_s$.

Rather than elaborating on a single method in detail, the author emphasized on development and test of several different techniques to improve the MEKF. One further method to keep the estimation process active is introduced in the next section.

4.3 Addition of Noise Term to the Parameter Vector

In the introduction to the current chapter, it is discussed that the insufficient performance of the MEKF as a parameter estimator is attributed to the initialization of covariance $P3$. The parameter estimate covariance $P3$ has to be chosen small, because otherwise the matrix becomes singular for some k. It is this inadequate initialization, that causes the parameter estimates $\hat{\theta}$ to "freeze", before they can converge to the system parameter value. A method which effectively prevents the "freezing" of the estimates, is presented in the section above. The method is based on a technique that changes the rate of convergence of the parameter estimate covariance $P3$. Here, the author introduces a different technique to improve the MEKF algorithm, which also directly effects the covariance $P3$, and is thus some-what similar to the previous method.

Recall from Chapter 2.4, where it is shown how the EKF can be utilized for parameter estimation problems, that the system parameter vector is modeled by:

$$\theta_{k+1} = \theta_k \tag{4.4}$$

which is appropriate, because a time-invariant system is assumed. That the parameter estimates of the MEKF can "freeze", is due to the fact that there is no dynamics in the parameter system (4.4). Suppose, one does model a time-invariant system as being time-varying, although it is not. This would keep the update process of the parameter estimates active for all k, because of the assumed dynamic in the parameter system. This approach has two advantages: first, the parameter estimates produced by the MEKF can no longer "freeze", and secondly, it broadens the field of possible applications to systems, which are slowly time-varying. The MEKF can be used to keep tracking of parameters, that vary in an unpredictable fashion. A time-varying system parameter is readily modeled by:

$$\theta_{k+1} = \theta_k + w_k \tag{4.5}$$

where w_k is a Gaussian noise vector with statistics:

$$E\{w_k\} = \underline{0} \tag{4.6}$$

$$E\left\{w_k w_j^T\right\} = Q_k^w \delta_{kj} \tag{4.7}$$

From the equations for the augmented system (2.155–2.166) it follows, that $\overline{Q}_k^v$ (2.169) is now given by:

$$\overline{Q}_k^v = \begin{bmatrix} Q_k^v & \underline{0} \\ \underline{0} & Q_k^w \end{bmatrix} \tag{4.8}$$

Substituting this $\overline{Q}_k^v$ back into equation (2.169), and partioning the augmented error covariance matrix P_{k+1}, yields for the parameter covariance update equation:

$$P3_{k+1} = P3_k - L_k S_k L_k^T + Q_k^w \tag{4.9}$$

As in the previous method, where one had to choose a proper reduction factor, the question is how to select a parameter noise covariance matrix Q^w, that optimizes the filter's performance in a particular case. A practical thing to do is to start with very small Q^w ($|P3_0| > 100\,|Q^w|$) and gradually increase Q^w, for different runs, while recording the filter's performance, till an optimum is reached. Possibly, a time-varying parameter noise covariance matrix Q^w_k should be selected.

TEST G, TEST H and TEST I

The MEKF algorithm, complemented with the described method, is applied to the single parameter case from the previous chapter. System, model, noise sequences and filter initialization, are precisely as in Chapter 3.5. The parameter error covariance $P3$ is updated via equation 4.9, where the noise covariance is chosen to be:

$$\text{TEST G:}\quad Q^w_k = \left\{ 2.0 \times 10^{-5} \quad \text{for all } k \right.$$

$$\text{TEST H:}\quad Q^w_k = \left\{ 2.0 \times 10^{-4} \quad \text{for all } k \right.$$

$$\text{TEST I:}\quad Q^w_k = \begin{cases} 2.0 \times 10^{-4} & \text{for } k \leq 50 \\ 2.0 \times 10^{-6}(150 - k) & \text{for } 50 < k < 140 \\ 2.0 \times 10^{-5} & \text{for } k \geq 140 \end{cases}$$

Discussion of the Results

Modeling the time-invariant system as being slowly time-varying considerably increases the rate of convergence of the parameter estimates to the system parameters. The larger the parameter noise covariance matrix Q^w, the faster do the estimates converge. However, there are limits to this. It can be observed from Figure 4.7 that for a Q^w of 0.0002 the variances in the estimation process are already so large that the parameter estimates become inconsistent. As discussed earlier, it is sufficient to enlarge the covariance $P3$ only for the first few iterations, to account for the improper initialization. This idea was pursued with TEST I, where the covariance Q^w is decreased to one tenth of its initial value, during the iterations 50–140. Figure 4.6–4.8 show that, although the rate of convergence is still the same, the variance in the estimation process is reduced substantially, due to the time-varying Q^w.

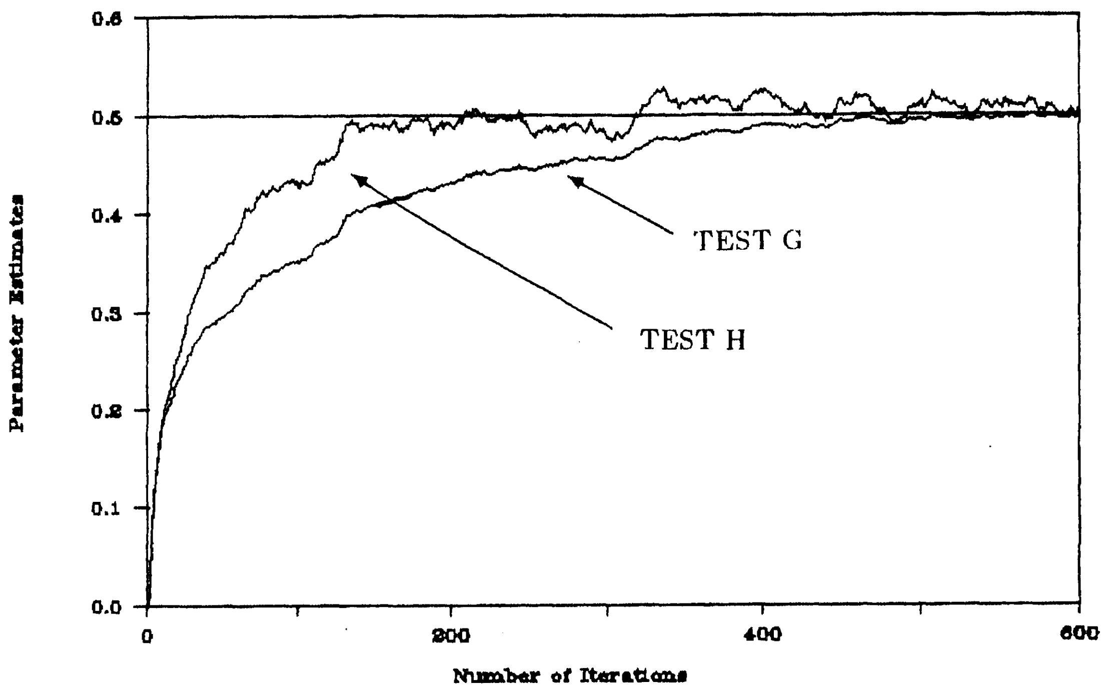

Figure 4.5: Parameter Estimates for Different Parameter Noise Covariances Test G and Test H

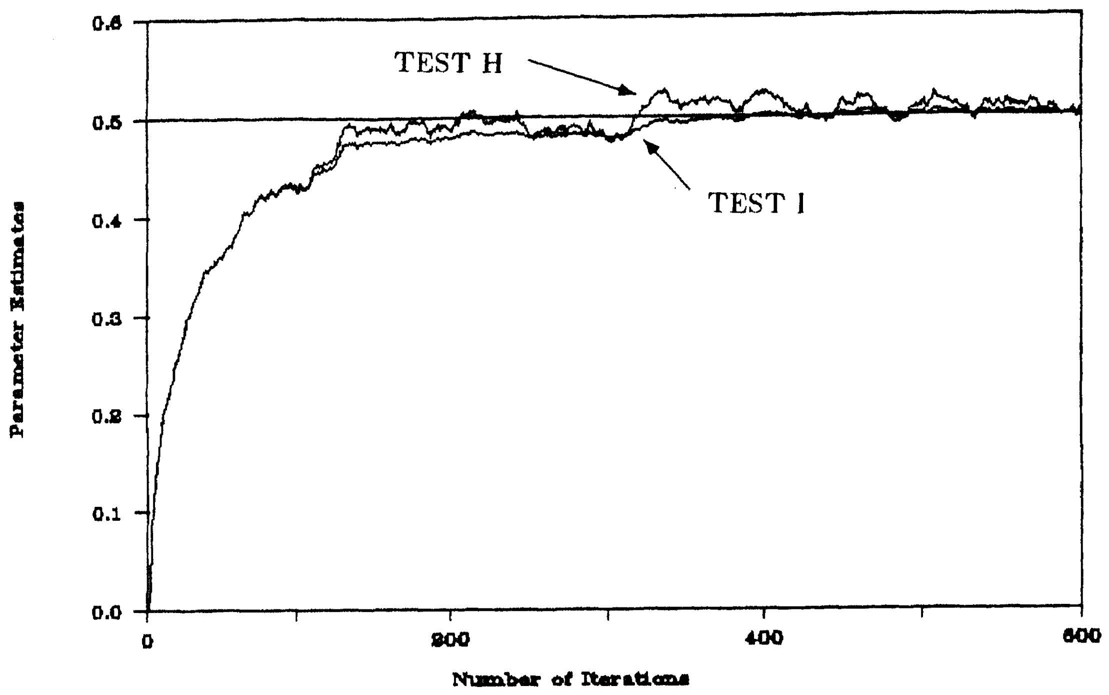

Figure 4.6: Parameter Estimates for Different Parameter Noise Covariances Test **H** and Test I

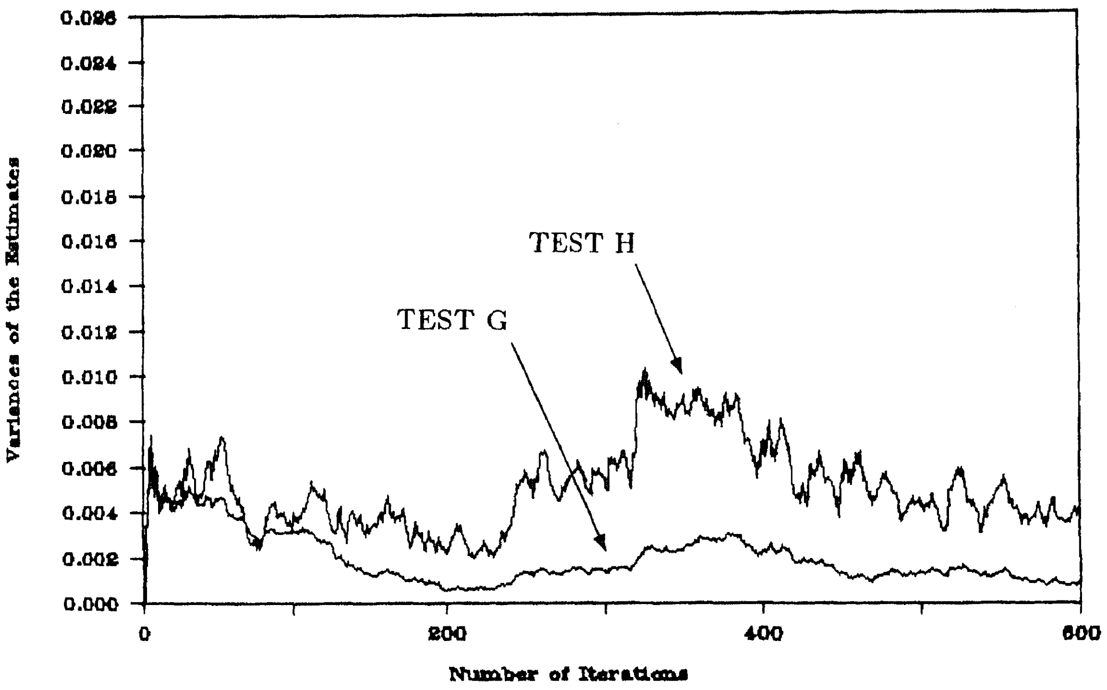

Figure 4.7: Variances of Parameter Estimates Test G and Test H

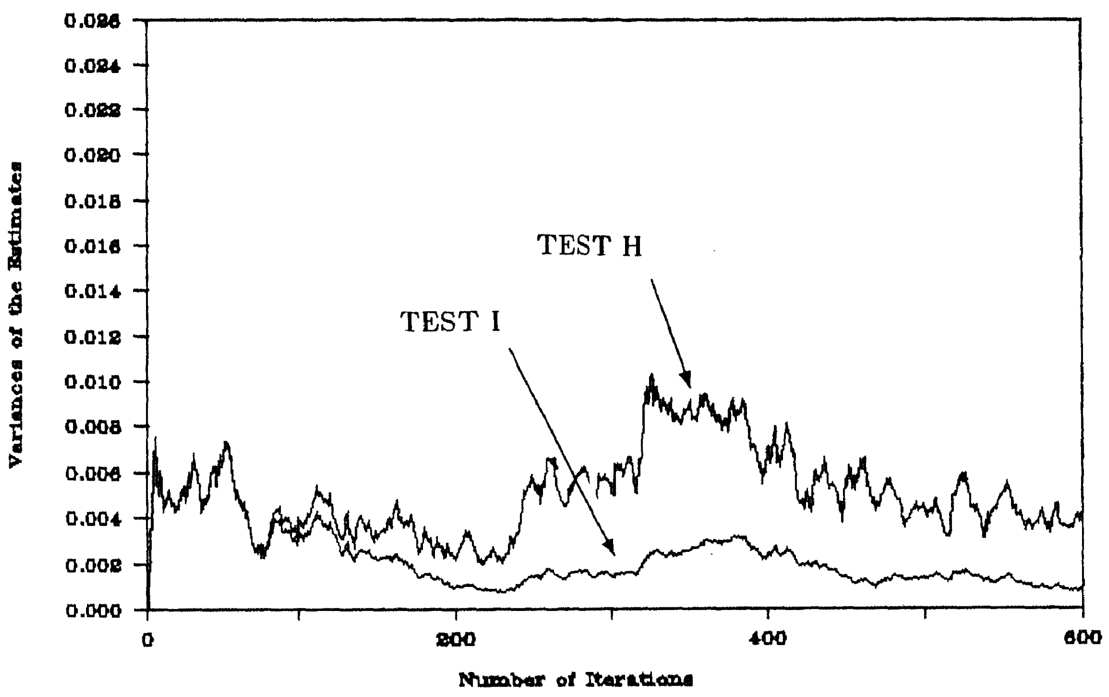

Figure 4.8: Variances of Parameter Estimates Test H and Test I

4.4 Enlargement of the Kalman Gain Matrix

So far, two different methods to improve the convergence properties of the MEKF have been introduced. Both methods prevent the parameter error covariance matrix $P3$ from decreasing at its normal pace, in order to compensate for the inadequate initialization of $P3$. A different approach to neutralize the improper covariance $P3_0$, is to enlarge the parameter Kalman gain matrix L by a certain factor, say gf. Because one merely wants to change the step size in the parameter updating process but not affect the updating direction, each entry of L has to be multiplied by gf (cf. equation 2.185). The update equations for covariances $P2$ (2.188) and $P2$ (2.189) remain unchanged. To examine the effectiveness of the method described, two simulation results are given next.

The two simulations are executed under identical conditions to that of TEST **A**, i.e. system, model, noise sequences and filter initialization are precisely as in TEST A. However, the parameter update equation of the plain MEKF is modified as:

$$\hat{\theta}_{k+1|k} = \hat{\theta}_{k|k-1} + gf_k \times L_k \left(z_k - C(\hat{\theta}_{k|k-1})\hat{x}_{k|k-1}\right) \tag{4.10}$$

TEST K

For this test a constant gain factor gf is selected.

$$gf = 2.2 \quad \forall k \quad k = 1,\ldots,600$$

TEST L

Following the reasoning of the previous two sections, a time-varying gain factor gf was chosen.

$$gf_k = \begin{cases} 2.5\,(0.985)^{k-1} & \mathit{for}\ k \leq 50 \\ 1.2 & \mathit{for}\ k > 50 \end{cases}$$

The results of these two tests are given with Figure 4.9 for the averaged parameter estimates and with Figure 4.10, which shows the respective variances, associated with the estimates.

The smoothness of the graphs of Figure 4.9, which show the averaged parameter estimates, is striking. As revealed by Figure 4.10, the variance in the estimation process is better than in any of the other tests considered so far. This phenomenon is explained by the fact, that this technique — contrary to the methods of the previous two sections — does not have a strong effect on the time history of covariance $P3$. The parameter covariance matrix does still decrease rapidly to very small values, so that there is very little variation in the parameter estimates for later iterations, which is comparable with the behavior of the plain MEKF algorithm. On the other hand, due to the larger parameter gain, the estimates do approach the system parameter value, in relatively few iterations. It is observed from Figure 4.9, that the larger gain factor results in slightly inferior convergence properties. This is due to the large parameter updating gain which puts the parameter estimates to high values during the first few iterations, from which they can not return, because of the fast decreasing covariance $P3$. The author experimented with even larger gain factors. The results of these experiments was that the filter algorithm became unstable, because some of the parameter estimates were larger than one (see also Section 4.6).

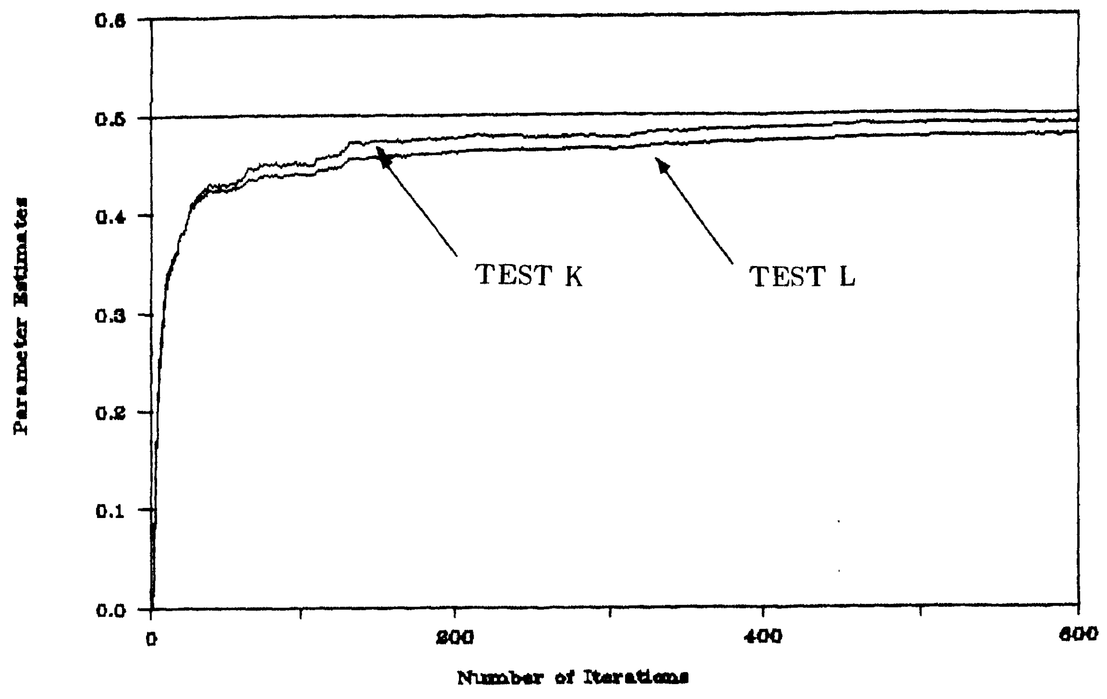

Figure 4.9: Parameter Estimates Test K (const. *gf* and Test L (variable *gf*)

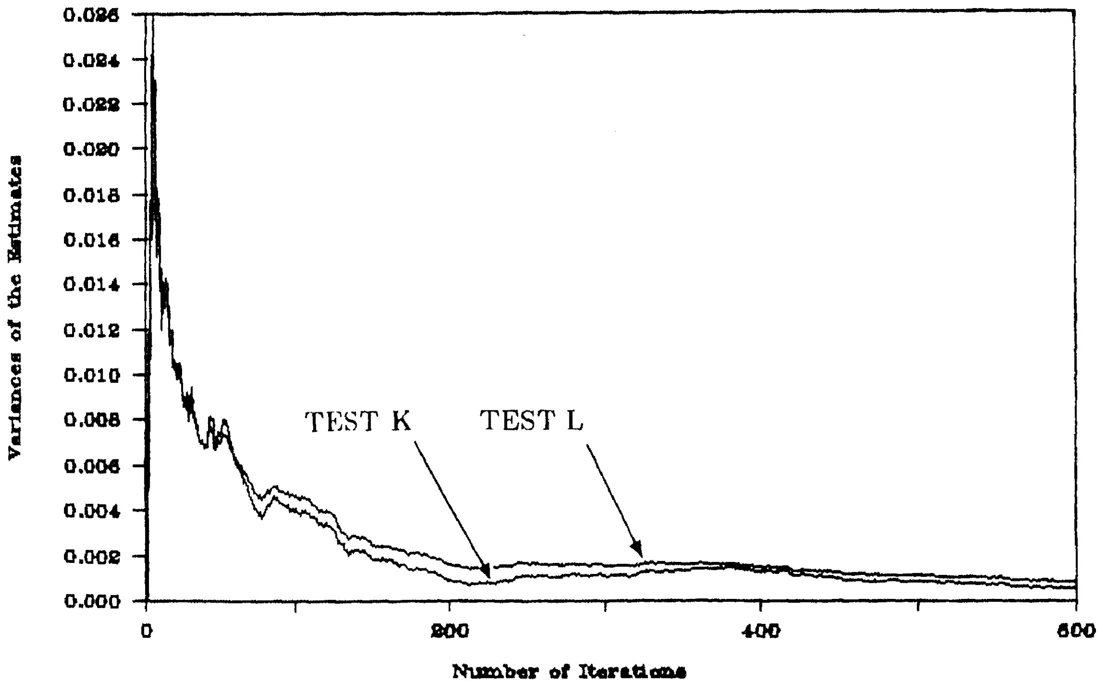

Figure 4.10: Variances of Parameter Estimates Test K and Test L

4.5 Rejection of Spurious Data Points

It has been mentioned earlier, that covariance $P3$ should be initialized with the largest value possible, for the filter algorithm to be stable. Initialization of the filter with a slightly larger $P3$ does not cause the algorithm to diverge for all runs; it usually results in one or two "bad" runs, out of 25. But what is it, that makes the algorithm work in some cases, and causes the algorithm to fail, in others? To answer this question, the author examined the noise sequences involved, at and prior to the point, where the filter started to diverge. It turned out, that those numbers of the noise sequences, where the divergence started out, were particularly large in magnitude. As pointed out in the derivation of the EKF (Section 2.3), any large noise term can bring the algorithm out of the linear region[2] and cause the algorithm to diverge. This explains the observed interrelation between the large noise terms and the divergence of the algorithm, following the processing of these samples. In order to maintain a stable algorithm, one rather suppresses spurious data points, than endanger the stability of the estimation process, by processing them.[3] But how can spurious data points be detected, and where should one draw the borderline between "good" and "bad" measurements? To answer the first question, recall that the residual $(z - C(\hat{\theta})\hat{x})$ is the difference between expected measurement $\hat{z}$ and actual measurement z, where a large difference indicates a spurious data point. These differences are readily qualified, using the prediction error covariance V, which is defined as:

$$V_{k|k-1} = E\left\{(z_k - C(\theta)\hat{x}_{k|k-1})(z_k - C(\theta)\hat{x}_{k|k-1})^T\right\} \tag{4.11}$$

It is shown in Chapter 2, that the above equation can be rewritten as:

$$V_{k|k-1} = C(\theta)\, P_{k|k-1}\, C(\theta) + Q_k^e \tag{4.12}$$

[2]See footnote 30 on page 42

[3]Suppression of spurious data points is also suggested by Maybeck [39] for a different application.

Because the measurement matrix C is in general assumed to be dependent on the parameter vector θ, which is not precisely known to the filter, an approximation of (4.12), given with equation (4.13), has to be used, instead.

$$V_{k|k-1} = C(\hat{\theta}_{k|k-1})\, P_{k|k-1}\, C(\hat{\theta}_{k|k-1}) + Q_k^e \tag{4.13}$$

Maybeck suggests in [39] to reject measurements, whenever the square of the residual exceeds three times the corresponding autocovariance factor in V.

It turned out, that the suggested limit is too large for the MEKF to improve the filters stability at the startup of the estimation process. On the other hand, the limits can not be chosen extremely small, because this results in rejecting too many data points, and thus slow down the estimation process. It is reasonable, to choose the rejection limit small for the first few iterations, where the MEKF is highly sensitive towards "bad" data. For later iterations, the limit should be gradually increased, to make use of all the information contained in the measurements. Once the parameter estimates have converged to the system parameter and are "frozen" there, the rejection limit can be dropped completely, without risking the filter to diverge, because the MEKF has transformed itself into a KF, with all its properties.

The idea pursued here, is to reject spurious measurement, and thus being able to initialize covariance *P3* with a larger value, that should increase the rate of convergence in the parameter estimation process. An appropriate rejection limit, that is neither not too large (no measurements would be rejected and thus make the method useless), nor too small (this would cause the loss of valuable information needed, for the algorithm to converge). The author experimented with numerous, constant and time-varying rejection limits. Unfortunately, none of them would improve the filter's performance substantially. In most cases, the performance of the filter was worse than for the plain MEKF algorithm. With a high rejection limit, the method showed no effect at all, and with a low rejection limit, the rate of convergence was slowed down. The increase in the initial value for covariance *P3*, made possible by suppression of spurious measurements, was more than outweighed by the loss of information due to the "throw-away" of output data. For completeness, an example of the described method is given below.

TEST J

System, model, noise sequences and filter initialization are as in TEST A, with the exception of $P3_0$, which could be set to 0.017, a value slightly larger than in TEST A. An approximation for the residual covariance is for the particular problem given by:

$$\begin{aligned} V_{k|k-1} &= C(\hat{\theta}_{k|k-1})\, P_{k|k-1}\, C(\hat{\theta}_{k|k-1}) + Q^e_k \\ &= 4\, P3_{k|k-1} + 0.5 \end{aligned}$$

Measurements were rejected, i.e. not processed by the filter, whenever the residual RES was in magnitude larger than:

$$\begin{array}{lll} |RES| > 1.0\sqrt{V} & & \text{for } k \leq 20 \\ |RES| > (0.1k - 1)\sqrt{V} & & \text{for } 20 < k < 35 \\ |RES| > 2.5\sqrt{V} & & \text{for } k \geq 35 \end{array}$$

Discussion of the Results

As it can be observed from Figure 4.11, the method to reject spurious data points, failed to improve the MEKF algorithm. The benefits gained by rejecting some measurements, were too small, to compensate for the loss of information. Yet, this method maybe beneficial if the limits are chosen, such that only those measurements that could disturb the filter severely are suppressed.

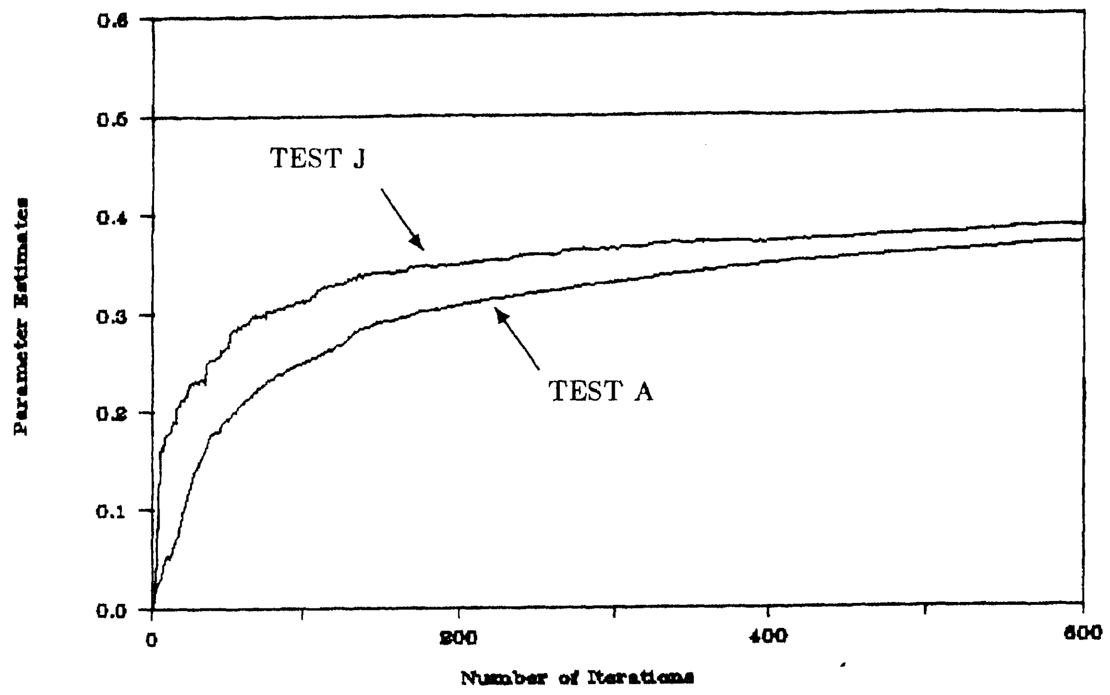

Figure 4.11: Parameter Estimates Test A and Test J

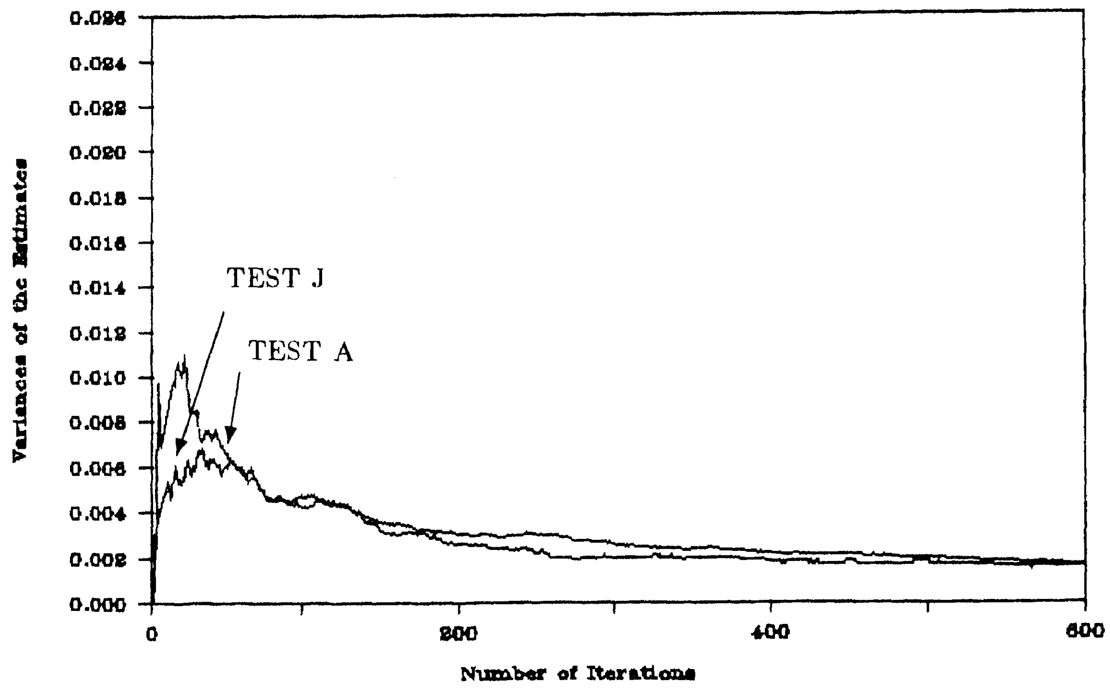

Figure 4.12: Variances of Parameter Estimates Test A and Test J

4.6 Keep $\hat{\theta}$ in $\mathcal{D}_s$

Ljung's proof [33] of the convergence properties of the MEKF is based on the condition, that the filter algorithm is complemented with a projection facility, which assures that all parameter estimates are element of the compact subset $\mathcal{D}_S$, where $\mathcal{D}_S$ is defined by:

$$\mathcal{D}_S = \{\, \theta \mid (A(\theta), B(\theta)) \text{ detectable and } (A(\theta), Q^v) \text{ stabilizable} \,\}$$

No elegant technique to determine this set for the general state space model, is known to the author. A prude for higher order systems computationally costly method, is to check for any $\hat{\theta}$, whether it is an element of $\mathcal{D}_S$, i.e., following this approach, one has to test at each time step if the pair $(A(\theta), B(\theta))$ is detectable, and if $(A(\theta), Q^v)$ is stabilizable. A simple projection facility, to keep the parameter estimates inside of $\mathcal{D}_S$, is then given by the rule:

- if $\hat{\theta}_{k+1|k} \in \mathcal{D}_S$, update the parameter vector, i.e. make $\hat{\theta}_{k+1|k}$ to the current parameter estimate
- if $\hat{\theta}_{k+1|k} \notin \mathcal{D}_S$, discard $\hat{\theta}_{k+1|k}$, i.e. do not update the parameter vector

Provided that $\hat{\theta}_0$ is element of $\mathcal{D}_S$, this rule ensures, that all $\hat{\theta}_{k+1|k}$ lie inside of the compact subset $\mathcal{D}_S$.

In case the system under consideration is a physical system, information about the parameter vector might be available, to further restrict a for the parameter estimates permissible region. The projection facility should be altered for those cases to account for the additional information.

For the first order system of Section 3.5, the set $\mathcal{D}_S$ is readily derived to be:

$$\mathcal{D}_S = \{ \text{ interior of unit circle } \}$$

The MEKF algorithm applied to the single parameter case considered in this chapter, should be complemented with the described projection facility, to keep all parameter estimates inside the unit circle. No simulation results are given here, because convergence properties of the plain MEKF algorithm are such that all the parameter estimates (cf. TEST A), stay well inside the unit circle. This is also attributed to the fact, that there is a substantial margin between the unit circle and the system parameter. Yet, the discussed projection facility is incorporated in a improved MEKF, where the dynamic of the parameter estimation process is such that, parameter estimates outside of $\mathcal{D}_S$ are more likely.

4.7 An Improved MEKF — Results

The simulation results for the techniques described in this chapter, are in general positive. The question is, whether it is possible, through combining all the methods that work, to achieve even better results. The goal is to design a filter, that unites all the positive features of the different techniques, so that the parameter estimates converge faster, than in any of the other filters shown, so far.

The method to enlarge the Kalman gain matrix, produces estimates that vary least for the different runs, once the algorithm has converged. On the other hand, the improvement techniques of Section 4.2 and Section 4.3 do result in comparatively smaller variances for the first few iterations, but larger ones for later time instances. Combining these three methods hopefully results in a filter, that produces fast convergent parameter estimates, and yet is insensitive towards system and measurement noise.

TEST M

System, model, noise sequences and filter initialization are precisely as in TEST A. To combine all the techniques introduces in this chapter in one filter, the following settings are selected:

$$\text{reduction factor:} \quad rf_k = \begin{cases} 1.6 & for\ k < 9 \\ 1.7 - 0.01\,k & for\ 10 \leq k \leq 70 \\ 1.0 & for\ k > 60 \end{cases}$$

$$\text{gain factor:} \quad gf_k = \left\{ 1.5 \quad for\ 1 \leq k \leq 600 \right.$$

$$\text{noise term:} \quad Q^w_k = \left\{ 1.0 \times 10^{-5} \quad \text{for } 1 \leq k \leq 600 \right.$$

This example shows (cf. Figure 4.13) that it is possible to further improve the filter's performance by combining the individual techniques, described in this chapter, into one filter.

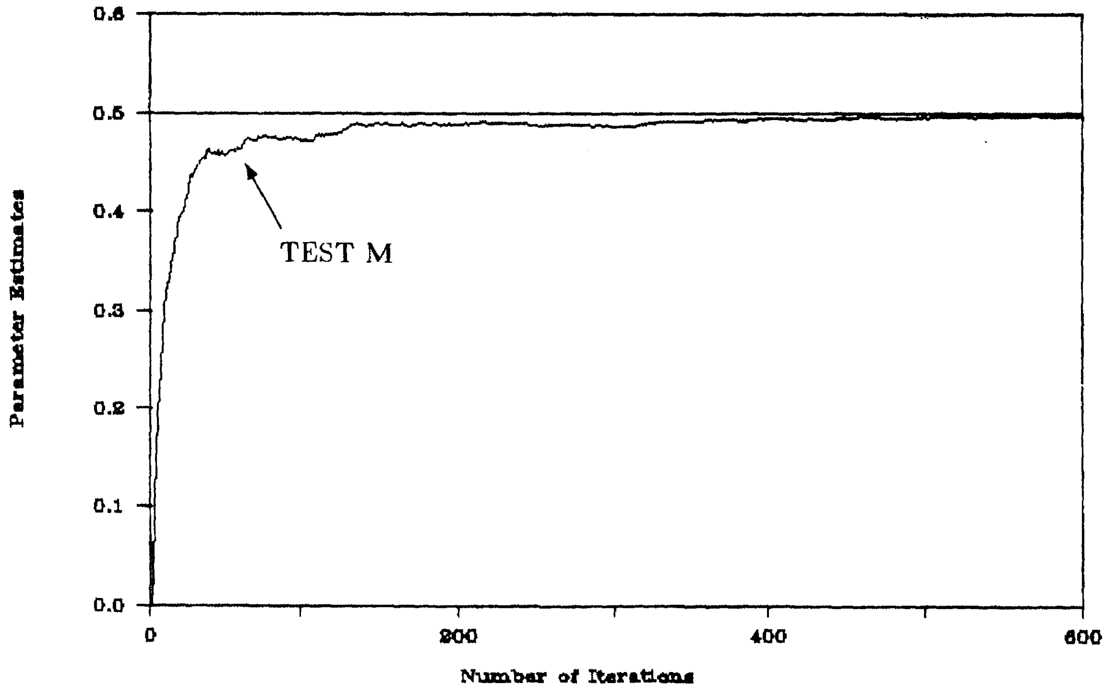

Figure 4.13: Parameter Estimates Test M

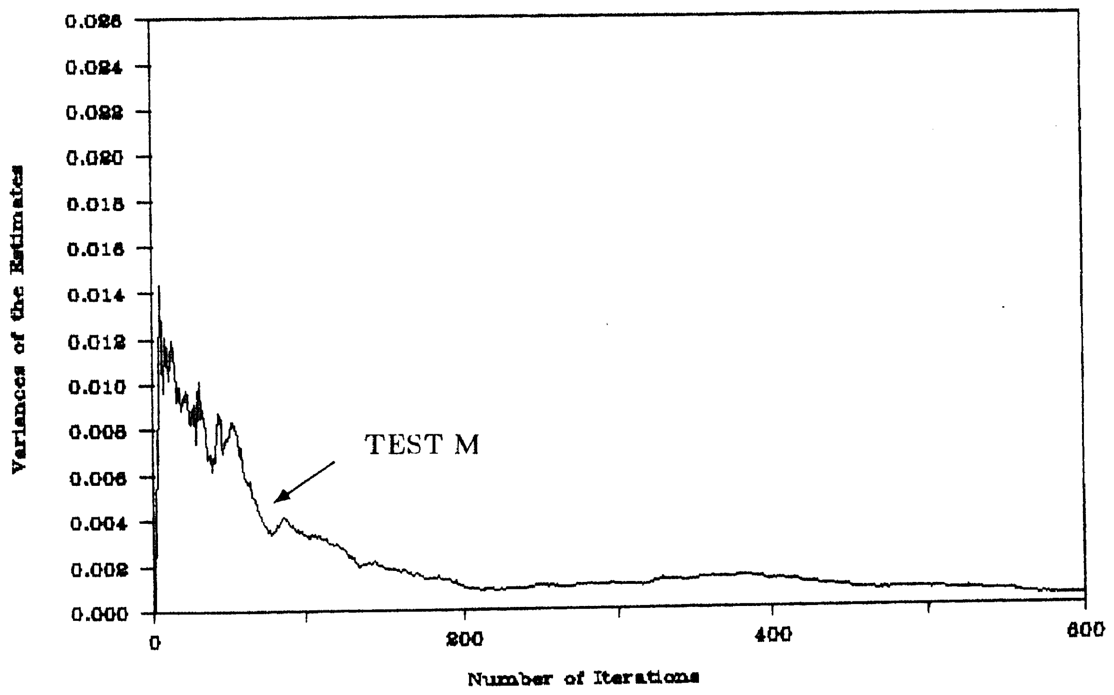

Figure 4.14: Variances of Parameter Estimates Test M

4.8 Conclusions

In this chapter, five different methods, intended to improve the inacceptable convergence properties of the plain MEKF, have been introduced. In order to keep results comparable, they have all been applied to the same single parameter case of Chapter 3. Due to the stochastic nature of the estimation process, *Monte Carlo* simulations were necessary to study the effects of the different techniques on the filter's performance.

The method to decelerate the "shrinking" of the parameter estimate error covariance $P3$, not only increases the rate of convergence, but also leads to a more robust filter algorithm. Slightly inferior results were achieved by modeling the time-invariant system as being slowly time-varying. This second method has the advantage, that it can identify the parameters, and then keep track of them throughout the process time. This feature is of particular interest in detecting failures in the process, e.g. caused through wear of parts, before this is indicated by measurable output signals. Good results in both the variations on the parameter estimates for different runs as well as the rate of convergence, were accomplished by increasing the step size in the parameter update process. In case the dynamics in the estimation process such that some of the parameter estimates fall outside of $\mathcal{D}_S$, a projection facility is needed, which discards these estimates and thus assures that all $\hat{\theta}$ are element of $\mathcal{D}_S$. The primary problem associated with this technique is, that there is no general method to derive this set $\mathcal{D}_S$. A computationally costly check of each $\hat{\theta}$ whether it is an element of the set $\mathcal{D}_S$, or not, seems to be the only feasible solution. Section 4.5 informs the reader about the authors attempt to increase the rate of convergence in the plain MEKF, by rejecting, i.e. non-processing, of spurious data points. Although this method does have its merits for other applications, it failed to increase the rate of convergence in the MEKF. In Section 4.6, finally, it is reported that the filtering results achieved by application of just one of the techniques described, are not the best ones possible, and combinations of it should be used, to further enhance the MEKF.

Though only applied to a single parameter case, the introduced filter upgrade techniques are designed for the general state space model. However, the question remains how well they will work, when applied to higher order cases. The dependency between given system, noise statistic and appropriate gain factor, reduction factor, covariance Q^w etc. is subject to further research.

Chapter 5

Comparison of an improved MEKF with a RELS Filter

5.1 Introduction

In the previous chapter, several techniques have been developed to increase the rate of convergence in the MEKF. Although the MEKF, when complemented with these methods, performs substantially better than the plain filter algorithm, the question remains, how "good" or "bad" this filter is with repect to other parameter estimators.

The MEKF shall be compared with a Recursive Extended Least Squares (RELS) parameter estimator. The RELS, a particularly easy to implement and robust parameter estimator, is for this reason often the first method to be tried out.

A brief summary of the RELS algorithm is given next. For comprehensive treatments of the Recursive Extended Least Square method the reader is referred to [38], [23] or [15].

5.2 The RELS Algorithm

The RELS algorithm is based on an Auto Regressive Moving Average eXogenous (ARMAX) representation of the system given by:

$$z_k = -\sum_{i=1}^{n} a_i \, z_{k-i} + \sum_{i=1}^{m} b_i \, u_{k-i} + \sum_{i=0}^{n} c_i \, e_{k-i} \tag{5.1}$$

where $\{e_k\}$ is a zero mean white noise sequence. The order of the system (n, m) is assumed to be known, but the coefficients in (5.1) are unknown. The unknown coefficients a_i, b_i and c_i forme, just as in the case of the EKF, a parameter vector θ, to be estimated.

In the RELS algorithm these parameters are estimated by comparing the system output z with the output of an implemented model of the system z^m.

Define the output errors as:

$$\hat{e}_k = z_k - z_k^m \tag{5.2}$$

where z^m is given by:

$$z_k^m = -\sum_{i=1}^{n} \hat{a}_i z_{k-i} + \sum_{i=1}^{m} \hat{b}_i u_{k-i} + \sum_{i=1}^{n} \hat{c}_i e_{k-i} \tag{5.3}$$

Defining a data vector $\hat{\psi}_k$:

$$\hat{\psi}_k^T = [-y_{k-1}, \ldots, -y_{k-n}, u_{k-1}, \ldots, u_{k-m}, \hat{e}_{k-1}, \ldots, \hat{e}_{k-n}] \tag{5.4}$$

and a parameter estimate vector $\hat{\theta}_k$:

$$\hat{\theta}_k^T = [\hat{a}_1, \ldots, \hat{a}_n, \hat{b}_1, \ldots, \hat{b}_m, \hat{c}_1, \ldots, \hat{c}_n]^T \tag{5.5}$$

allows to rewrite equation (5.3) in a compact form:

$$z_k^m = \hat{\psi}_k^T \hat{\theta}_k \tag{5.6}$$

The sequence of output errors contains the information to drive the parameter estimate vector $\hat{\theta}$ to the parameter vector θ. Once $\hat{\theta}_k$ has converged to the parameter vector θ_k the output errors $\hat{e}$ coincide with the noise samples e, i.e., become a white noise sequences, that contains no futher information.

The parameter estimates are generated by the algorithm, given with equations (5.7-5.10):

$$\gamma_k = \frac{P3_k \hat{\psi}_{k+1}}{1 + \hat{\psi}_{k+1}^T P3_k \hat{\psi}_k} \tag{5.7}$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \gamma_k [z_k - \hat{\psi}_{k+1}^T \hat{\theta}_k] \tag{5.8}$$

$$\beta_{k+1} = [I - \gamma_k \psi_{k+1}^T] \beta_k \tag{5.9}$$

$$\hat{e}_{k+1} = z_k - \hat{\psi}_{k+1}^T \hat{\theta}_{k+1} \tag{5.10}$$

5.3 Implementation of the RELS Method

Prior to the start up of the parameter estimation process, covariance matrix, $P3$, and the parameter estimate vector $\hat{\theta}$ need to be initialized. For the simulations reported in this thesis $P3$ was set to

$$P3_0 = 1000 \times I$$

But any other diagonal matrix ranging from $10I$–$5000I$ produced similar results. Assuming no a priori knowledge about the system state, the parameter estimates were initially set to zero. The input/output data is still assumed to be generated by a system[1] in state space representation adequately described by the equations (2.144) and (2.145). Note that in these state space equations two independent noise sources, namely system noise and measurement noise, are assumed to be present. This can not be modelled adequately with an equation of the type of (5.1). Since noise structure and noise statistics are not need to be known for the RELS method to be applicable, one shall not worry about this. That is, when converting the state space model into an ARMAX model, all the information about the noise is neglected. The noise sources will simply be modelled by the term $\sum_{i=0}^{n} e_{k-i}$, no matter what the actual noise statistics look like. Thus, it is important to note for the interpretation of the simulation results to follow, that the RELS algorithm is supplied with substantially less information about the system, as compared to the MEKF. It might be possible to incorporate the information about the noise in the RELS algorithm, but this is subject to further research.

[1] Here restricted to a single-input, single-output model

5.4 Results

To achieve a practical measure of the performance of the MEKF, its convergence results are compared with the ones obtained by application of the RELS method. This is done for two systems; the first order system of Chapter 3 and a second order system which is described by:

$$
\begin{aligned}
x_{k+1} &= \begin{bmatrix} 0.0 & 1.0 \\ +0.4 & -0.2 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k + v_k && (5.11) \\
z_k &= \begin{bmatrix} 2.0 & 1.0 \end{bmatrix} x_k + e_k && (5.12)
\end{aligned}
$$

with noise statistics $v_n \sim N\left(\underline{0}, \left(\begin{smallmatrix} 0.5 & 0.0 \\ 0.0 & 0.5 \end{smallmatrix}\right)\right)$ and $e_n \sim N(0.0, 0.5)$ input/output data for both filters is produced by these systems. For the MEKF, only the second order system needs to be considered, as data for the scalar case is already available (see Section 4.6). Assume that in case of the second order system the entries a_{21} and a_{22} of the system matrix are unknown. This can be modelled by:

$$
\begin{aligned}
x_{k+1} &= \begin{bmatrix} 0.0 & 1.0 \\ \theta_1 & \theta_2 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k + v_k && (5.13) \\
z_k &= \begin{bmatrix} 2.0 & 1.0 \end{bmatrix} x_k + e_k && (5.14)
\end{aligned}
$$

Hence, the filtering problems is to identify these two parameters θ_1 and θ_2. An upgraded version of the MEKF as described in Section 4.6 is applied here.

The filter is initialized as:

$$
\begin{aligned}
P1 &= \begin{bmatrix} 0 & 0 \\ 0 & 10 \end{bmatrix} & \hat{x}_0 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} & Q^v &= \begin{bmatrix} 0.5 & 0.0 \\ 0.0 & 0.5 \end{bmatrix} \\
P2 &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \hat{\theta}_0 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} & Q^e &= \begin{bmatrix} 0.5 \end{bmatrix} \\
P3 &= \begin{bmatrix} 0.015 & 0 \\ 0 & 0.015 \end{bmatrix}
\end{aligned}
$$

To make the RELS method applicable to these estimation problems, the system equations need to be converted in an ARMAX model representation, which is done by completely neglecting noise structure and noise statistics. The first order system is therefore given by:

$$z_k = 0.5\, z_{k-1} + 2\, u_{k-1} + c_1\, e_{k-1} \tag{5.15}$$

For the second order system one can write:

$$z_k = -0.2\, z_{k-1} + 0.4\, z_{k-2} + 2\, u_{k-1} + u_{k-2} + c_1\, e_{k-1} + c_2\, e_{k-2} \tag{5.16}$$

The RELS algorithm for both systems is initialized with $P3_0 = 1000\, I$ and $\hat{\theta}_0 = \underline{0}$.

Figure 5.1 and Figure 5.2 show the parameter estimates produced by the RELS and the MEKF for the two systems described. Their sensitivity towards the noise sequences can be observed from Figure 5.3 and Figure 5.4.

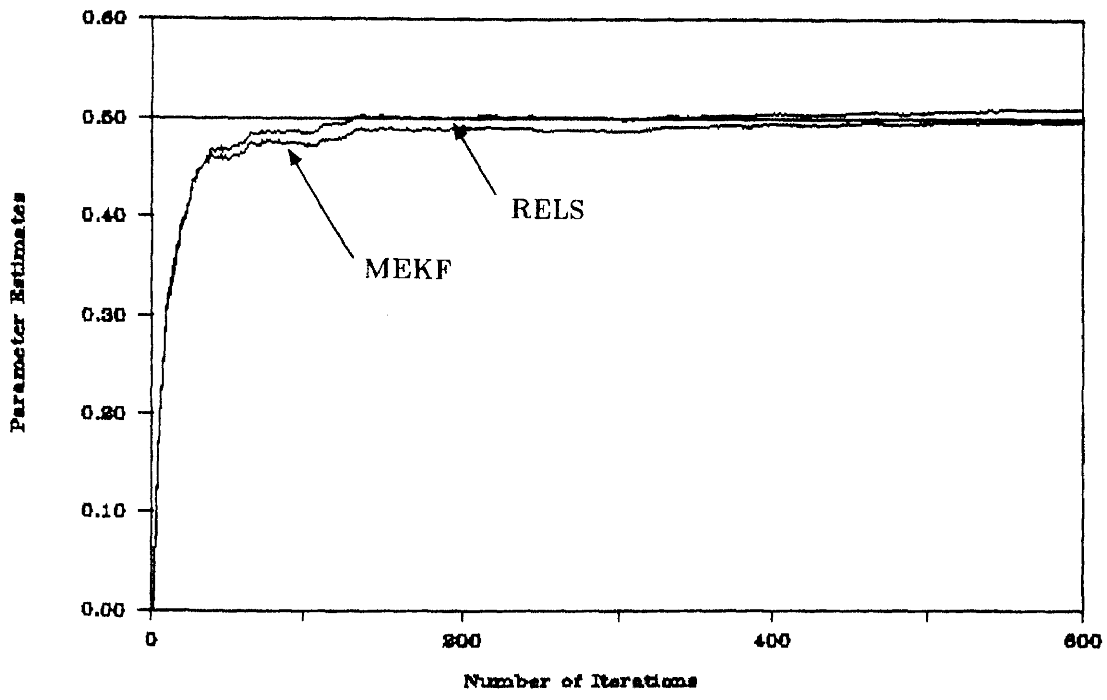

Figure 5.1: Parameter Estimates for 1st Order System

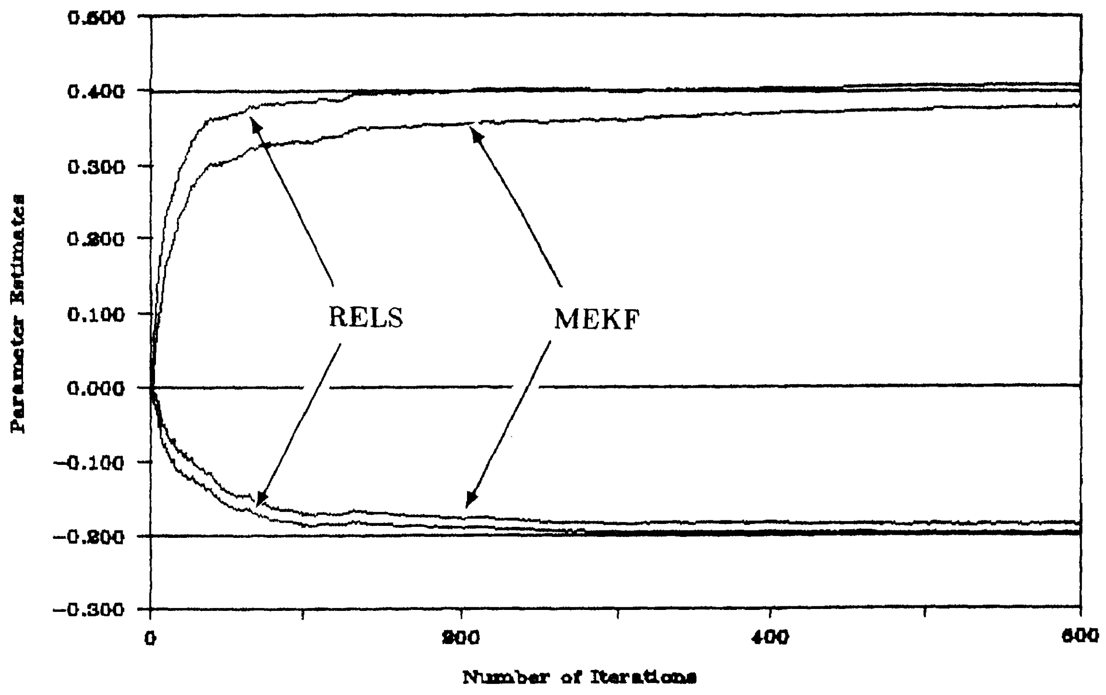

Figure 5.2: Parameter Estimates for 2nd Order System

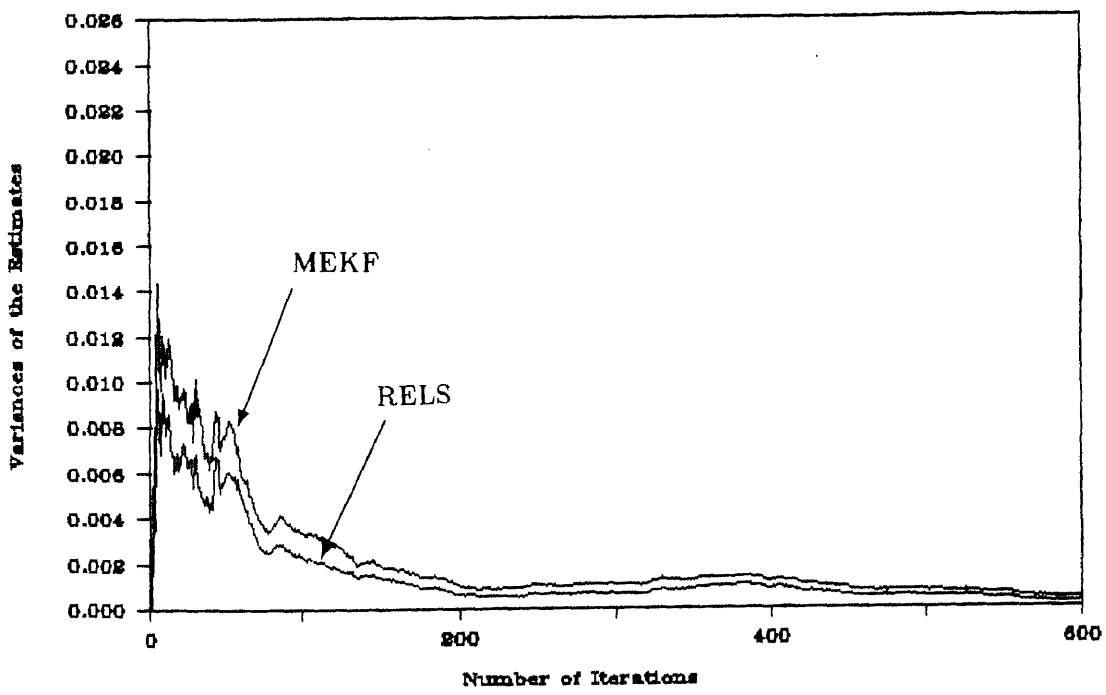

Figure 5.3: Variances of Parameter Estimates (1st Order System)

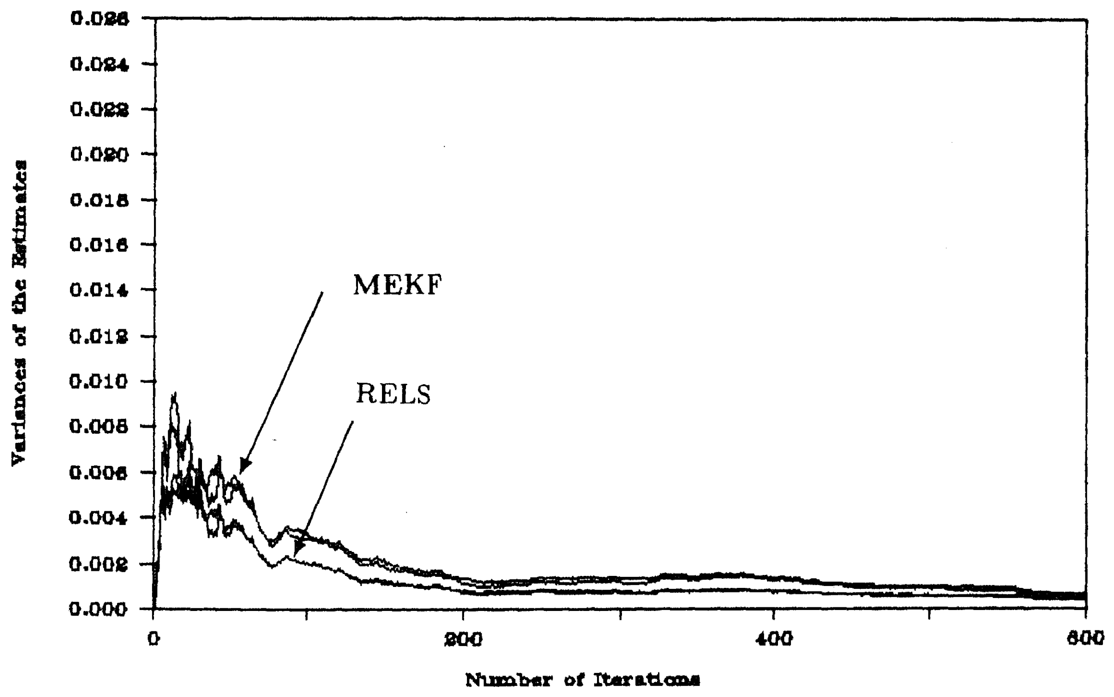

Figure 5.4: Variances of Parameter Estimates (2nd Order System)

Chapter 6

Conclusions

6.1 Summary and Conclusions

In this thesis parameter estimation of linear, discrete-time systems using a modified Extended Kalman Filter by Ljung has been studied. The prime motivation in using the MEKF to attack the parameter estimation problem was the excellent global convergence properties stated by Ljung.

A substantial part of this thesis is devoted to estimation theory. A complete and detailed derivation of the discrete-time Kalman Filter using a Bayesian approach is given in Chapter 2. For the discrete-time EKF, two different derivations are given; the first one is entirely heuristic, but possibly provides more insight to what the EKF is about, than the second which is mathematically more rigorous.

Chapter 2 also shows how the EKF, a state estimator for non-linear systems, can be utilized for parameter estimation of linear systems.

An EKF algorithm modified by Ljung and tailored for parameter estimation is presented, and rederived in parts to show some intermediate steps, that are left out in [33]. This modification adds a tremendous computational burden on the filter. The actual computing time to account for the modification, is insignificant, because steady advances in hardware will overcome of this problem.

However, the primary concern is numerical stability and the difficulties involved in solving three coupled matrix Ricatti equations. It has been shown that, even for the low order cases, which are treated in this thesis, quite sophisticated programming techniques are required to detect convergence and/or divergence of the algorithm.

Further problems encountered with the implementation of the MEKF were related to the filter initialization. Initializing the parameter estimation error covariance matrix $P3$ appropriately such that $P3$ reflects the uncertainty about the parameter vector, resulted in numerical overflow after just few iterations.

On the other hand, selecting a initial covariance $P3_0$, such that the algorithm is stable, resulted in biased estimates, because the filter "freezes" the parameter updating process before the system parameter estimators could converge.

Chapter 4 presents several techniques developed by the author, which yield a stable filter algorithm and at the same time provide sufficient dynamics in the parameter updating process. Some of these methods worked excellent for the systems they were applied to. These techniques might also be applicable to other parameter estimation methods. However, in the case of MEKF, the question that remains is whether these techniques are still useful when used with higher order systems. How should their parameter (gain factor, reduction factor e.t.c) be selected, for certain classes of systems? This is subject to further research.

In summary, the MEKF as a parameter estimator proved to be problematic to initialize and difficult to operate. This was especially revealing when the author implemented for comparison reason, an RELS estimator and was striked by the easiness this could be done and by the simplicity of the estimation algorithm. The MEKF does have its value, in cases where knowledge about noise statistics, noise and system structures are available, because it can incorporate all these information in its state estimates. However, in practice it will be difficult to obtain all this information. In any case, the user has to ask herself or himself, whether it is worth to go through these difficulties that are likely to be encountered when using the MEKF. Also, whether it is not more advisable to use a different estimation algorithm such as RELS and others, which are substantially simpler to implement.

6.2 Recommendations for Future Work

The author lists here some of the problems encountered, and considers to be worthwhile subjects for further research.

1. In this thesis, several techniques were developed to compensate for improper initialization of the parameter estimation error covariance. Whether these techniques do have any value in applications other than the MEKF, should be investigated.

2. The MEKF formulation is much more general then shown in this thesis. It is recommended to apply it to MIMO systems, where other estimation techniques are too limited.

3. Ljung proposed another MEKF with similar convergence properties that is based on an innovation model. In such a filter, the Kalman gain matrix is directly parametrized. So the additional gain term $\frac{\partial K(\theta)}{\partial \theta}$ is particularly easy to compute. The author suggests the implementation and comparison of this filter with the one for the general state space model.

4. In practical cases one usually does not arrive at precise knowledge of noise statistics and noise structures. The question to be answered is, how sensitive the MEKF is towards inadequate selection of noise covariance Q^e and Q^v.

Appendix A

Program Listing MEKF

A.1 Main Program

```
**************************************************************************
*                                                                 *
*            RECURSIVE PARAMETER IDENTIFICATION ROUTINE                  *
*            (recursive prediction error method                          *
*                                                                        *
*                                          Last Revision NOVEMBER 1, 1987 *
*                                                                        *
*------------------------------------------------------------------------*
*                                                                        *
*      System model           x     = A x  + B u  + v ,   v ..N(O,QM)    *
*                              k+1       k      k    k     k             *
*                                                                        *
*      Measurement model      z  = C x  + e ,             e ..N(O,QS)    *
*                              k      k    k               k             *
*                                                                        *
*                                  T                                     *
*      Assumptions            E{e v } = 0 for all j,k = 0,1,2,...        *
*                                k j                                     *
*                                                                        *
*------------------------------------------------------------------------*
*      Notes:    x  =  x(t ), v  = v(t ), e(t ) are the state vector,    *
*                 k        k    k      k      k                          *
*                system noise, and the measurement noise of dimensions   *
*                                                                        *
*                nx, nu and ny, respectively.   A, B  and C are the      *
*                                                                        *
*                (nxxnx) state transition matrix, the (nxxnu) input-     *
*                                                                        *
*                distribution matrix and the (nyxnx) measurement         *
*                                                                        *
*                matrix (or observation matrix), respectively.           *
* *
*                                                                           *
*****************************************************************************
*
       DOUBLE PRECISION A(10,10),B(10,10),C(10,10),AEST(10,10),
     & CEST(10,10),QS(10,10),QM(10,10),X(10),XEST(10),BEST(10,10),
     & P1(10,10),P2(10,10),P3(10,10),P1T(10,10),P2T(10,10),
     & P3T(10,10),GK(10,10),GL(10,10),M(10,10),D(10,10),S(10,10),
     & SINV(10,10),THETA(10),PARA(10),EPS(10),U(10),V(10),E(10),Y(10),
     & DET,ERR1,RES,RJT
       INTEGER LCOUNT
       OPEN (UNIT=11,FILE='[BJS8884.NOISE]N_SEQ12',STATUS='OLD')
       OPEN (UNIT=22,FILE='[BJS8884.NOISE]N_SEQ13',STATUS='OLD')
       OPEN (UNIT=33,FILE='[BJS8884.NOISE]N_SEQ58',STATUS='OLD')
       OPEN (UNIT=44,FILE='A1EST12',STATUS='NEW')
*
*
*      READ ALL DATA NECESSARY TO RUN THE PROGRAM
*
*
       DATA NX/1/NU/1/NY/1/NP/1/
       DATA A(1,1),A(1,2),A(2,1),A(2,2)/0.5,1.,.5,.0/
       DATA B(1,1),B(2,1),C(1,1),C(1,2)/1.,0.,1.4,0./
       DATA AEST(1,1),AEST(1,2),AEST(2,1),AEST(2,2)/0.,1.,0.,0./
       DATA BEST(1,1),BEST(2,1),CEST(1,1),CEST(1,2)/1.,0.,2.0,0./
       DATA QS(1,1),QS(1,2),QS(2,1),QS(2,2)/0.55,0.,0.,1./
       QM(1,1)=0.55
       DATA PARA(1)/.5/PARA(2)/1./
*
*      INITIALIZE REAL STATE VECTOR X, ESTIMATED STATE VECTOR XEST,
*      PARAMETER VECTOR THETA AND COVARIANCE MATRICES P1,P2, P3
*
       CALL U02INITIALIZE(NX,NU,NY,X,XEST,THETA,P1,P2,P3)
*
*
*      THIS IS THE BEGIN OF THE ACTUAL IDENTIFICATION LOOP
*
*
*
       LOOP=600
       DO 2000 LCOUNT=1,LOOP
       WRITE(*,*) LCOUNT
*
*      READ INPUT-VECTOR u, SYSTEMNOISE-VECTOR v AND MEASUREMENTNOISE-
*      VECTOR e
*
       CALL U03READNOISE(NX,NU,NY,V,E,U)
       V(1)=0.7*V(1)
       E(1)=0.7*E(1)
```

```
*
*     COMPUTE TRUE STATE VECTOR AND TRUE OUTPUT
*
      CALL P21OUTPUT(Y,C,X,E,NY,NX)
      CALL P11STATETIMEUPDATE(A,X,B,U,V,NX,NU)
*
*     GENERATE THE MATRICES D AND M
*
      CALL U04DDERI(D,CEST,THETA,XEST,NX,NY,NP)
      write(*,*) 'd ',(( d(i,j), j=1,np), i=1,ny)
      CALL U05MDERI(M,AEST,BEST,THETA,XEST,U,NU,NX,NY,NP)
      write(*,*) 'm ',(( m(i,j), j=1,np), i=1,nx)
*
*
*
      CALL P23SMATRIX(NP,NX,NY,S,CEST,D,P1,P2,P3,QM)
      write(*,*) 's ',(( s(i,j), j=1,ny), i=1,ny)
      CALL P13DMATINV(SINV,S,NY,DET)
      write(*,*) 'sinv ',(( sinv(i,j), j=1,ny), i=1,ny)
      CALL P32MMODIFY(NU,NX,NY,NP,AEST,P1,XEST,S,SINV,GK,Y,M,THETA,
     & CEST,EPS,LCOUNT)
      write(*,*) 'm ',(( m(i,j), j=1,np), i=1,nx)
      CALL P24KALGNPAR(GL,P2,CEST,P3,D,SINV,NP,NX,NY)
      write(*,*) 'gl ',(( gl(i,j), j=1,ny), i=1,np)
      CALL P25KALGNSTATE(GK,AEST,CEST,P1,P2,P3,D,M,SINV,NX,NY,NP)
      write(*,*) 'gK ',(( gK(i,j), j=1,ny), i=1,nx)
      CALL P29PARAEST(THETA,GL,Y,CEST,XEST,NU,NX,NY,NP)
      CALL P30STATEEST(AEST,BEST,XEST,U,GK,Y,CEST,NU,NX,NY,NP)
      CALL P26P1UPDATE(AEST,P1,P2,P3,M,S,GK,QS,NX,NY,NP)
      write(*,*) 'P1 ',(( P1(i,j), j=1,nX), i=1,nx)
      CALL P27P2UPDATE(AEST,P2,P3,M,S,GK,GL,NX,NY,NP)
      write(*,*) 'P2 ',(( P2(i,j), j=1,nP), i=1,nx)
      CALL P28P3UPDATE(P3,S,GL,NX,NY,NP,LCOUNT)
      write(*,*) 'P3 ',((P3(i,j), j=1,nP), i=1,NP)
*
*     NOW UPDATE MATRIX AEST
*
      CALL U08UPDATE_ABC(AEST,BEST,CEST,THETA,NU,NX,NY,NP)
      write(*,*) 'para', (para(i), i=1,np)
      write(*,51) 'theta    ', (theta(i), i=1,np)
      write(44,40) (theta(i), i=1,np)
 2000 CONTINUE
*
*
      CLOSE(44)
      CLOSE(33)
      CLOSE(22)
      CLOSE(12)
```

```
   CLOSE(11)
   STOP
10 FORMAT(/A)
20 FORMAT(//A,I4)
30 FORMAT(//A)
40 FORMAT(F8.3)
50 FORMAT(2F8.3)
51 FORMAT(a10,F8.3)
   END
```

A.2 Subroutines

Program Package MATRIX

```
************************************************************************
*                                                                      *
*      Subroutine P01MATMULNN for the matrix operation     A := A B    *
*                                                                      *
*      A and B are N dimensional square matricies                      *
*      column vector.                                                  *
************************************************************************
      SUBROUTINE P01MATMULNN(A,B,N)
      DIMENSION A(10,10),B(10,10), C(10)
      DOUBLE PRECISION A,B,C
*
*     C is a dummy storage vector.
*
        DO 1000 J = 1,N
*       Obtain the j-th column of the product and store it
*       temporarily in the column vector C.
          DO 1001 I = 1,N
             C(I) = 0.0
             DO 1002 K = 1,N
                IF ((A(I,K) .NE. 0.0) .AND. (B(K,J) .NE. 0.0))
     &              C(I) = C(I) + A(I,K)*B(K,J)
 1002        CONTINUE
 1001     CONTINUE
*
*       Now replace j-th column of matrix B with the j-th column
*       of the result (stored in vector C).
*
          DO 1003 I = 1, N
             A(I,J) = C(I)
 1003     CONTINUE
 1000   CONTINUE
      RETURN
      END
*
************************************************************************
*                                                                      *
*      Subroutine P02MATVEC   for the matrix operation     y:= A*x     *
*                                                                      *
*      A is an (nxm) matrix and x is an m-dimensional column vector.   *
*                               y is an n-dimensional column vector.   *
************************************************************************
      SUBROUTINE P02MATVEC(A,X,Y,N,M)
      DIMENSION A(10,10),X(10), C(10),Y(10)
      DOUBLE PRECISION A,X,C,Y
*
*     C is a dummy storage vector.
*
        DO 1000 J = 1,N
*       Obtain the j-th element of the product and store it
*       temporarily in the column vector C.
             C(J) = 0.0
             DO 1002 K = 1,M
                IF ((A(J,K) .NE. 0.0) .AND. (X(K) .NE. 0.0))
     &              C(J) = C(J) + A(J,K)*X(K)
 1002        CONTINUE
 1000     CONTINUE
*
*       Now replace C with Y
*
          DO 1003 J = 1, N
               Y(J) = C(J)
 1003     CONTINUE
      RETURN
      END
```

```
***********************************************************************
*                                                                     *
*                                                                   T *
*     Subroutine PO3MATMULTRANSNN for the matrix operation   A := A B *
*                                                                     *
*     A and B are N dimensional square matrices.                      *
*                                                                     *
***********************************************************************
      SUBROUTINE PO3MATMULTRANSNN(A,B,N)
      DIMENSION A(10,10),B(10,10), C(10)
      DOUBLE PRECISION A,B,C
*     C is a dummy storage vector
        DO 1000 I = 1,N
*       Obtain the i-th row of the product and store it
*       temporarily in vector C.
          DO 1001 J = 1,N
             C(J) = 0.0
             DO 1002 K = 1,N
                IF ((A(I,K) .NE. 0.0) .AND. (B(J,K) .NE. 0.0))
     &              C(J) = C(J) + A(I,K)*B(J,K)
 1002        CONTINUE
 1001     CONTINUE
*       Now replace i-th row of matrix A with the i-th row
*       of the result (stored in vector C).
          DO 1003 J = 1, N
             A(I,J) = C(J)
 1003     CONTINUE
 1000   CONTINUE
      RETURN
      END
***********************************************************************
*                                                                     *
*     Subroutine PO4MATADD   for the matrix operation   C := A+B      *
*                                                                     *
*     RA and CA denotes the number of rows and the number of the      *
*     columns of the matrices A, B and C.                             *
*                                                                     *
***********************************************************************
      SUBROUTINE PO4MATADD(C,A,B,RA,CA)
      INTEGER RA,CA
      DIMENSION A(10,10),B(10,10),C(10,10)
      DOUBLE PRECISION A,B,C
      DO 1001 I = 1,RA
          DO 1001 J = 1,CA
                C(I,J) = A(I,J) + B(I,J)
 1001 CONTINUE
      RETURN
      END
```

```
***********************************************************************
*                                                                     *
*     Subroutine PO4VECADD   for the vector operation  c := a+b       *
*                                                                     *
*     RA denotes the number of elements of the vectors a,b,c.         *
*                                                                     *
***********************************************************************
      SUBROUTINE PO4VECADD(C,A,B,RA)
      INTEGER RA
      DIMENSION A(10),B(10),C(10)
      DOUBLE PRECISION A,B,C
      DO 1001 I = 1,RA
                C(I) = A(I) + B(I)
 1001 CONTINUE
      RETURN
      END
***********************************************************************
*                                                                     *
*     Subroutine PO5MATMUL for the matrix multiplication  C = A B     *
*                                                                     *
*     A and B are RAxCA and RBxCB dimensional matrices, respectively. *
*                                                                     *
***********************************************************************
      SUBROUTINE PO5MATMUL(A,B,C,RA,CA,RB,CB,RC,CC)
      INTEGER RA,CA,RB,CB,RC,CC
      DOUBLE PRECISION A(10,10),B(10,10),C(10,10),DUMMY
      RC = RA
      CC = CB
      IF(CA .NE. RB) GOTO 2001
        DO 1000 I = 1,RA
          DO 1001 J = 1,CB
             DUMMY = 0.0
             DO 1002 K = 1,CA
                IF ((A(I,K) .NE. 0.0) .AND. (B(K,J) .NE. 0.0))
     &              DUMMY = DUMMY + A(I,K)*B(K,J)
 1002        CONTINUE
          C(I,J) = DUMMY
 1001     CONTINUE
 1000   CONTINUE
      GOTO 2002
 2001 WRITE(*,*) '  ERROR IN PO5MATMUL:  Dimension mismatch.'
 2002 CONTINUE
      RETURN
      END
```

```
************************************************************************
*                                                                      *
*     PO6MATMULTRANS                                                   *
*                                                          T           *
*     Subroutine for the matrix multiplication    C = A B .            *
*                                                                      *
*     A and B are RAxCA and RBxCB dimensional matrices, respectively   *
*                                                                      *
************************************************************************
      SUBROUTINE PO6MATMULTRANS(A,B,C,RA,CA,RB,CB,RC,CC)
      INTEGER RA,CA,RB,CB,RC,CC
      DOUBLE PRECISION A(10,10),B(10,10), C(10,10)
*
*
      RC = RA
      CC = CB
*
*
      IF(CA .NE. CB) GOTO 2001
        DO 1000 I = 1,RA
          DO 1001 J = 1,RB
             C(I,J)= 0.0
             DO 1002 K = 1,CA
                IF ((A(I,K) .NE. 0.0) .AND. (B(J,K) .NE. 0.0))
     &              C(I,J) = C(I,J) + A(I,K)*B(J,K)
 1002        CONTINUE
 1001     CONTINUE
 1000   CONTINUE
*
*
      GOTO 2002
 2001 WRITE(6,*) '  ERROR IN PO6MATMULTRANS:  Dimension mismatch.'
 2002 CONTINUE
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*     SUBROUTINE P07                           T                     *
*     for the matrix operation P := O P O  + Q                       *
*     O, P and Q are NxN matrices, each.  P and Q are symmetrical.   *
*                                                                    *
**********************************************************************
      SUBROUTINE P07(O,P,Q,N)
      DIMENSION P(10,10),O(10,10),Q(10,10),C(10)
      DOUBLE PRECISION P,O,Q,C
      CALL PO3MATMULTRANSNN(P,O,N)
*
*     C is a dummy storage vector.
*
        DO 1000 J = 1,N
*       Obtain the j-th column of the product and store it
*       temporarily in the column vector C.
          DO 1005 I = 1,J-1
 1005        C(I) = 0.0
          DO 1001 I = J,N
             C(I) = 0.0
             DO 1002 K = 1,N
                IF ((O(I,K) .NE. 0.0) .AND. (P(K,J) .NE. 0.0))
     &              C(I) = C(I) + O(I,K)*P(K,J)
 1002        CONTINUE
 1001     CONTINUE
*
*       Now replace j-th column of matrix P with the j-th column
*       of the result (stored in vector C).  Also add matrix Q.
*
          DO 1003 I = J, N
 1003        P(I,J) = C(I) + Q(I,J)
 1000   CONTINUE
        DO 1020 I = 1, N
           DO 1020 J = I, N
              IF( I .NE. J)
     &             P(I,J) = P(J,I)
 1020   CONTINUE
       RETURN
       END
```

```
**********************************************************************
*                                                                    *
*     Subroutine P08 for the matrix operation      A := H Y + R      *
*                                                                    *
*     H is a an n-dimensional row vector, Y is an n-dimensional      *
*     column vector.                                                 *
**********************************************************************
      SUBROUTINE P08(H,Y,R,A,N)
      DIMENSION H(10),Y(10)
      DOUBLE PRECISION H,Y,A
             A = 0.0
             DO 1000 K = 1,N
                IF ((H(K) .NE. 0.0) .AND. (Y(K) .NE. 0.0))
     &             A = A + H(K)*Y(K)
 1000        CONTINUE
      A=A+R
      RETURN
      END
```

```
************************************************************************
*                                                                      *
*     P09MATROWMUL                                                     *
*                                                                T     *
*     Subroutine for the matrix-vector multiplication    Y = P H .     *
*                                                                      *
*     P is an NxN matrix H is an N-dimensional row vector.             *
*                                                                      *
************************************************************************
      SUBROUTINE P09MATROWMUL(P,H,Y,N)
      DIMENSION P(10,10),H(10), Y(10)
      DOUBLE PRECISION P,H,Y
*
        DO 1000 I = 1,N
             Y(I)= 0.0
             DO 1002 J = 1,N
                IF ((P(I,J) .NE. 0.0) .AND. (H(J) .NE. 0.0))
     &             Y(I) = Y(I) + P(I,J)*H(J)
 1002        CONTINUE
 1000   CONTINUE
      RETURN
      END


************************************************************************
*                                                                      *
*     Subroutine P10SCALARRESIDUE                                      *
*     for the matrix operation        v := z - H x                     *
*                                                                      *
*     H is a an n-dimensional row vector, x is an n-dimensional        *
*     column vector.                                                   *
*                                                                      *
************************************************************************
      SUBROUTINE P10SCALARRESIDUE(H,X,Z,V,N)
      DIMENSION H(10),X(10)
      DOUBLE PRECISION H,X,V,Z
*
             V = 0.0
             DO 1000 K = 1,N
                IF ((H(K) .NE. 0.0) .AND. (X(K) .NE. 0.0))
     &             V = V + H(K)*X(K)
 1000        CONTINUE
      V = Z - V
      RETURN
      END
```

```
**************************************************************************
*                                                                        *
*     Subroutine P11STATETIMEUPDATE                                      *
*                                                                        *
*     for the matrix operation           x := A * x + B * u + v          *
*                                                                        *
*     A is a NxN matrix, B is a NxM matrix,  x,v are n-dimensional       *
*                                                                        *
*     column vectors and u is a m-dimensional column vector              *
*                                                                        *
**************************************************************************

      SUBROUTINE P11STATETIMEUPDATE(A,X,B,U,V,N,M)

      DIMENSION A(10,10),B(10,10),X(10),C(10),U(10),V(10)
      DOUBLE PRECISION A,B,X,C,U,V
*
*     C is a dummy storage vector.
*
        DO 1000 J = 1,N
*
*       Obtain the j-th element of the product and store it
*       temporarily in the column vector C.
*
             C(J) = 0.0
             DO 1002 K = 1,N
                IF ((A(J,K) .NE. 0.0) .AND. (X(K) .NE. 0.0))
     &              C(J) = C(J) + A(J,K)*X(K)
 1002        CONTINUE
             DO 1004 K=1,M
                IF ((B(J,K) .NE. 0.0) .AND. (U(K) .NE. 0.0))
     &              C(J) = C(J) + B(J,K)*U(K)
 1004     CONTINUE
 1000     CONTINUE
*
*       Now replace vector X with vector C and add vector V.
*
        DO 1003 J = 1, N
             X(J) = C(J)+V(J)
 1003   CONTINUE
       RETURN
       END
```

```
*************************************************************************
*                                                          -1           *
*     Subroutine P13DMATINV for the matrix operation    C := B          *
*                                                                       *
*     A is N dimensional square matrix                                  *
*     Operation in double precision                                     *
*                                                                       *
*************************************************************************
      SUBROUTINE P13DMATINV(C,B,N,D)
      DIMENSION C(10,10),B(10,10),A(100),L(10),M(10)
      DOUBLE PRECISION C,B,A,D,HOLD,BIGA
C
      DO 5 I=1,N
         IZ=N*(I-1)
            DO 5 J=1,N
               IJ=IZ+J
    5          A(IJ)=B(I,J)
      D=1.0
      NK=-N
      DO 80 K=1,N
         NK=NK+N
         L(K)=K
         M(K)=K
         KK=NK+K
         BIGA=A(KK)
            DO 20 J=K,N
               IZ=N*(J-1)
                  DO 20  I=K,N
                     IJ=IZ+I
                     IF(DABS(BIGA)-DABS(A(IJ))) 15,20,20
   15                BIGA=A(IJ)
                     L(K)=I
                     M(K)=J
   20       CONTINUE
C
C
C
            J=L(K)
            IF(J-K) 35,35,25
   25       KI=K-N
               DO 30 I=1,N
                  KI=KI+N
                  HOLD=-A(KI)
                  JI=KI-K+J
                  A(KI)=A(JI)
   30             A(JI)=HOLD
C
C
```

```
C
   35         I=M(K)
              IF(I-K) 45,45,38
   38         JP=N*(I-1)
                 DO 40 J=1,N
                    JK=NK+J
                    JI=JP+J
                    HOLD=-A(JK)
                    A(JK)=A(JI)
   40               A(JI)=HOLD
C
C
C
   45         IF(BIGA) 48,46,48
   46         D=0.0
              RETURN
   48         DO 55 I=1,N
                 IF(I-K) 50,55,50
   50            IK=NK+I
                 A(IK)=A(IK)/(-BIGA)
   55         CONTINUE
C
C
C
          DO 65 I=1,N
             IK=NK+I
             HOLD=A(IK)
             IJ=I-N
                DO 65 J=1,N
                   IJ=IJ+N
                   IF(I-K) 60,65,60
   60              IF(J-K) 62,65,62
   62              KJ=IJ-I+K
                   A(IJ)=HOLD*A(KJ)+A(IJ)
   65           CONTINUE
C
C
C
             KJ=K-N
                DO 75 J=1,N
                   KJ=KJ+N
                   IF(J-K) 70,75,70
   70              A(KJ)=A(KJ)/BIGA
   75           CONTINUE
C
C
C
            D=D*BIGA
```

```
C
C
C
             A(KK)=1.0/BIGA
   80    CONTINUE
C
C
C
         K=N
  100    K=(K-1)
         IF(K) 150,150,105
  105    I=L(K)
         IF(I-K) 120,120,108
  108    JQ=N*(K-1)
         JR=N*(I-1)
            DO 110 J=1,N
               JK=JQ+J
               HOLD=A(JK)
               JI=JR+J
               A(JK)=-A(JI)
  110          A(JI)=HOLD
  120    J=M(K)
         IF(J-K) 100,100,125
  125    KI=K-N
            DO 130 I=1,N
               KI=KI+N
               HOLD=A(KI)
               JI=KI-K+J
               A(KI)=-A(JI)
  130          A(JI)=HOLD
         GO TO 100
  150    DO 500 I=1,N
            IZ=N*(I-1)
               DO 500 J=1,N
                  IJ=IZ+J
  500             C(I,J)=A(IJ)
         RETURN
         END
```

```
*************************************************************************
*                                                            -1        *
*    Subroutine MATINV for the matrix operation        A := A          *
*                                                                       *
*    A is N dimensional square matrix                                  *
*                                                                       *
*************************************************************************
      SUBROUTINE MATINV(A,N)
C
      DIMENSION A(10,10)
C
      DO 10 K=1,N
         DO 20 I=1,N
            DO 30 J=1,N
               IF(I.EQ.K) GOTO 30
               IF(J.EQ.K) GOTO 30
               A(I,J)=A(I,J)-A(K,J)*A(I,K)/A(K,K)
   30       CONTINUE
   20    CONTINUE
C
C
         DO 40 I=1,N
            DO 50 J=1,N
               IF(I.EQ.J) GOTO 50
               IF(I.NE.K) GOTO 50
               A(I,J)=-A(I,J)/A(K,K)
   50       CONTINUE
   40    CONTINUE
C
          DO 60 I=1,N
            DO 70 J=1,N
               IF(J.NE.K) GOTO 70
               IF(I.EQ.J) GOTO 70
               A(I,J)=A(I,J)/A(K,K)
   70       CONTINUE
   60    CONTINUE
      A(K,K)=1.0/A(K,K)
   10 CONTINUE
      RETURN
      END
```

```
*********************************************************************
*                                                                   *
*                                                               T   *
*     Subroutine P15TRANSPOMATRIX for the matrix operation   B := A *
*                                                                   *
*     A is (n*m) dimensional matrix.                                *
*     B is (m*n) dimensional matrix.                                *
*                                                                   *
*********************************************************************
      SUBROUTINE P15TRANSPOMATRIX(A,B,N,M)
      DIMENSION A(10,10),B(10,10)
      DOUBLE PRECISION A,B,C
*
      DO 1 I=1,N
         DO 2 J=1,M
            B(J,I)=A(I,J)
    2    CONTINUE
    1 CONTINUE
      RETURN
      END


*********************************************************************
*                                                                   *
*     Subroutine P16MAKEIDENT                                       *
*                                                                   *
*     Generates a n-dimensional identity matrix A                   *
*                                                                   *
*********************************************************************
      SUBROUTINE P16MAKEIDENT(A,N)
      DOUBLE PRECISION A(10,10)
*
      DO 1 I=1,N
         DO 2 J=1,N
            IF(I.EQ.J) A(I,J)=1.0
            IF(I.NE.J) A(I,J)=0.0
    2    CONTINUE
    1 CONTINUE
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*     Subroutine P17MATTRACE for the matrix operation  TR := TRACE(A)  *
*                                                                    *
*     A is a general n-dimensional square matrix.                    *
*     TR is the sum of the diagonal entries.                         *
*                                                                    *
**********************************************************************
      SUBROUTINE P17MATTRACE(A,N,TR)
      DIMENSION A(10,10)
*
      DOUBLE PRECISION TR,A
      TR=0.0
        DO 1001 I = 1,N
                TR=TR+A(I,I)
 1001   CONTINUE
      RETURN
      END


**********************************************************************
*                                                                    *
*     Subroutine P18AUGMENT                                          *
*                                                                    *
*     X and THETA are N and M dimensional vectors, respectively.     *
*     Z is the augmented vector of dimension NM=N+M.                 *
*                                                                    *
**********************************************************************
      SUBROUTINE P18AUGMENT(X,THETA,N,M,NM)
      DIMENSION X(10),THETA(10),Z(20)
      DOUBLE PRECISION X,THETA,Z
*
      DO 1001 I = 1,N
              Z(I)=X(I)
 1001 CONTINUE
      DO 1002 I = 1,M
              Z(I+N)=THETA(I)
 1002 CONTINUE
      RETURN
      END
```

```
**************************************************************************
*                                                                        *
*                                                                        *
*     Subroutine P19MATCOPY for the matrix operation  B :=  A            *
*                                                                        *
*     A and B are RxC dimensional matrices.                              *
*                                                                        *
**************************************************************************
      SUBROUTINE P19MATCOPY(A,B,R,C)
      INTEGER R,C
      DIMENSION A(10,10),B(10,10)
      DOUBLE PRECISION A,B
*
      DO 1000 I = 1,R
          DO 1000 J = 1,C
 1000         B(I,J)=A(I,J)
      RETURN
      END


**************************************************************************
*                                                                        *
*     Subroutine P20VECTORRESIDUE                                        *
*     for the matrix operation       res := y - H x                      *
*                                                                        *
*     H is a an MxN dimensional matrix, x is an N dimensional            *
*     column vector and y,res are M dimensional row vectors.             *
*                                                                        *
**************************************************************************
      SUBROUTINE P20VECTORRESIDUE(X,H,Y,RES,N,M)
      DIMENSION H(10,10),X(10),Y(10),RES(10)
      DOUBLE PRECISION H,X,Y,RES
*
        DO 1000 I = 1,M
          RES(I) = 0.0
          DO 1001 J = 1,N
             RES(I) = RES(I)+H(I,J)*X(J)
 1001     CONTINUE
*
*       Now subtract this from y-vector.
*
          RES(I)=Y(I)-RES(I)
 1000   CONTINUE
      RETURN
      END
```

```
*****************************************************************************
*                                                                           *
*     Subroutine P21OUTPUT for the matrix operation    y:= C*x + e          *
*                                                                           *
*     C is an (mxn) matrix and x is an n-dimensional column vector          *
*     and e is a m-dimensional column vector.                               *
*                                                                           *
*****************************************************************************
      SUBROUTINE P21OUTPUT(Y,C,X,E,NY,NX)
      DIMENSION C(10,10),X(10),E(10),Y(10)
      DOUBLE PRECISION C,X,E,Y
*
*
        DO 1000 I = 1,NY
          Y(I) = 0.0
          DO 1001 J = 1,NX
             Y(I) = Y(I)+C(I,J)*X(J)
 1001     CONTINUE
*
*       Now add vector E to vector Y.
*
             Y(I) = Y(I)+E(I)
 1000   CONTINUE
      RETURN
      END


******************************************************************************
*                                                                            *
*     Subroutine P22MMATMUL for the matrix multiplication  D = A B C         *
*                                                                            *
*     A, B and C are RAxCA, RBxCB and RCxCC dimensional matrices,            *
*                                                                            *
*     respectively.                                                          *
*                                                                            *
******************************************************************************
      SUBROUTINE P22MMATMUL(D,A,B,C,RA,CA,RB,CB,RC,CC,RD,CD)
      INTEGER RA,CA,RB,CB,RC,CC,RD,CD,RT,CT
      DIMENSION A(10,10),B(10,10),C(10,10),D(10,10),TEMP(10,10)
      DOUBLE PRECISION A,B,C,D,TEMP
*
*
      CALL P05MATMUL(A,B,TEMP,RA,CA,RB,CB,RT,CT)
      CALL P05MATMUL(TEMP,C,D,RT,CT,RC,CC,RD,CD)
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*     Subroutine P23SMATRIX for the matrix operation                 *
*                     T        T      T T       T                    *
*          S := C P1 C + C P2 D + D P2 C + D P3 D + QM               *
*                                                                    *
*     S is a NYxNY dimensional matrix.                               *
*                                                                    *
**********************************************************************
      SUBROUTINE P23SMATRIX(NP,NX,NY,S,CEST,D,P1,P2,P3,QM)
      DOUBLE PRECISION S(10,10),CEST(10,10),P1(10,10),P2(10,10)
      DOUBLE PRECISION D(10,10),QM(10,10),TEMP(10,10),CT(10,10)
      DOUBLE PRECISION DT(10,10),P3(10,10),P2T(10,10)
*
*
      CALL P19MATCOPY(QM,S,NY,NY)
      CALL P15TRANSPOMATRIX(CEST,CT,NY,NX)
      CALL P15TRANSPOMATRIX(D,DT,NY,NP)
      CALL P15TRANSPOMATRIX(P2,P2T,NX,NP)
      CALL P22MMATMUL(TEMP,D,P3,DT,NY,NP,NP,NP,NP,NY,NY,NY)
      CALL P04MATADD(S,S,TEMP,NY,NY)
      CALL P22MMATMUL(TEMP,D,P2T,CT,NY,NP,NP,NX,NX,NY,NY,NY)
      CALL P04MATADD(S,S,TEMP,NY,NY)
      CALL P22MMATMUL(TEMP,CEST,P2,DT,NY,NX,NX,NP,NP,NY,NY,NY)
      CALL P04MATADD(S,S,TEMP,NY,NY)
      CALL P22MMATMUL(TEMP,CEST,P1,CT,NY,NX,NX,NX,NX,NY,NY,NY)
      CALL P04MATADD(S,S,TEMP,NY,NY)
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*     Subroutine P24KALGNPAR for the matrix operation                *
*                             T T      T                             *
*                  GL := ( P2 C + P3 D )/ S                          *
*                                                                    *
*     GL is a NPxNY dimensional matrix.                              *
*                                                                    *
**********************************************************************
      SUBROUTINE P24KALGNPAR(GL,P2,CEST,P3,D,SINV,NP,NX,NY)
      DIMENSION GL(10,10),SINV(10,10),CEST(10,10),P2(10,10),P3(10,10)
      DIMENSION D(10,10),TEMP(10,10),CT(10,10),P2T(10,10)
      DIMENSION DT(10,10),P3T(10,10)
      DOUBLE PRECISION GL,SINV,CEST,P2,P3,D,TEMP,CT,P2T,DT,P3T
*
*
      CALL P15TRANSPOMATRIX(CEST,CT,NY,NX)
      CALL P15TRANSPOMATRIX(D,DT,NY,NP)
      CALL P15TRANSPOMATRIX(P2,P2T,NX,NP)
      CALL P05MATMUL(P2T,CT,TEMP,NP,NX,NX,NY,NP,NY)
      CALL P19MATCOPY(TEMP,GL,NP,NY)
      CALL P05MATMUL(P3,DT,TEMP,NP,NP,NP,NY,NP,NY)
      CALL P04MATADD(TEMP,TEMP,GL,NP,NY)
*
*     don't multiply with SINV when MEKF is used
*     use the idendity matrix instead
*
C      CALL P05MATMUL(TEMP,SINV,GL,NP,NY,NY,NY,NP,NY)
      RETURN
      END
```

```
************************************************************************
*                                                                      *
*     Subroutine P25KALGNSTATE for the matrix operation                *
*                           T       T T        T        T              *
*          GK := ( A P1 C + M P2 C + A P2 D + M P3 D )/S               *
*                                                                      *
*     GK is a NXxNY dimensional matrix.                                *
*                                                                      *
************************************************************************
      SUBROUTINE P25KALGNSTATE(GK,AEST,CEST,P1,P2,P3,D,M,SINV,NX,NY,NP)
      DOUBLE PRECISION SINV(10,10),CEST(10,10),P1(10,10),P2(10,10)
      DOUBLE PRECISION D(10,10),AEST(10,10),TEMP(10,10),CT(10,10)
      DOUBLE PRECISION M(10,10),DT(10,10),P3(10,10),P2T(10,10)
      DOUBLE PRECISION GK(10,10)
*
*
      CALL P15TRANSPOMATRIX(CEST,CT,NY,NX)
      CALL P15TRANSPOMATRIX(D,DT,NY,NP)
      CALL P15TRANSPOMATRIX(P2,P2T,NX,NP)
      CALL P22MMATMUL(GK,M,P3,DT,NX,NP,NP,NP,NP,NY,NX,NY)
      CALL P22MMATMUL(TEMP,AEST,P2,DT,NX,NX,NX,NP,NP,NY,NX,NY)
      CALL P04MATADD(GK,GK,TEMP,NX,NY)
      CALL P22MMATMUL(TEMP,M,P2T,CT,NX,NP,NP,NX,NX,NY,NX,NY)
      CALL P04MATADD(GK,GK,TEMP,NX,NY)
      CALL P22MMATMUL(TEMP,AEST,P1,CT,NX,NX,NX,NX,NX,NY,NX,NY)
      CALL P04MATADD(GK,GK,TEMP,NX,NY)
      CALL P05MATMUL(GK,SINV,GK,NX,NY,NY,NY,NX,NY)
      RETURN
      END
```

```
*********************************************************************
*                                                                   *
*     Subroutine P26P1UPDATE for the matrix operation               *
*                   T        T      T T        T         T          *
*       P1 := A P1 A + A P2 M + M P2 A + M P3 M - GK D GK + QS      *
*                                                                   *
*     P1 is a NXxNX dimensional matrix.                             *
*                                                                   *
*********************************************************************
      SUBROUTINE P26P1UPDATE(AEST,P1,P2,P3,M,S,GK,QS,NX,NY,NP)
      DOUBLE PRECISION AEST(10,10),P1(10,10),P2(10,10),P3(10,10)
      DOUBLE PRECISION M(10,10),S(10,10),GK(10,10),QS(10,10),GKT(10,10)
      DOUBLE PRECISION TEMP(10,10),AT(10,10),MT(10,10),P2T(10,10)
*
*
      CALL P15TRANSPOMATRIX(AEST,AT,NX,NX)
      CALL P15TRANSPOMATRIX(M,MT,NX,NP)
      CALL P15TRANSPOMATRIX(GK,GKT,NX,NY)
      CALL P15TRANSPOMATRIX(P2,P2T,NX,NP)
      CALL P22MMATMUL(P1,AEST,P1,AT,NX,NX,NX,NX,NX,NX,NX,NX)
      CALL P22MMATMUL(TEMP,AEST,P2,MT,NX,NX,NX,NP,NP,NX,NX,NX)
      CALL P04MATADD(P1,P1,TEMP,NX,NX)
      CALL P22MMATMUL(TEMP,M,P2T,AT,NX,NP,NP,NX,NX,NX,NX,NX)
      CALL P04MATADD(P1,P1,TEMP,NX,NX)
      CALL P22MMATMUL(TEMP,M,P3,MT,NX,NP,NP,NP,NP,NX,NX,NX)
      CALL P04MATADD(P1,P1,TEMP,NX,NX)
      CALL P22MMATMUL(TEMP,GK,S,GKT,NX,NY,NY,NY,NY,NX,NX,NX)
      DO 1 I=1,NX
         DO 1 J=1,NX
    1 TEMP(I,J)=-TEMP(I,J)
      CALL P04MATADD(P1,P1,TEMP,NX,NX)
      CALL P04MATADD(P1,P1,QS,NX,NX)
      RETURN
      END
```

```
************************************************************************
*                                                                      *
*     Subroutine P27P2UPDATE for the matrix operation                  *
*                                                   T                  *
*                    P2 := A P2 + M P3 - GK D GL                       *
*                                                                      *
*     P2 is a NXxNP dimensional matrix.                                *
*                                                                      *
************************************************************************
      SUBROUTINE P27P2UPDATE(AEST,P2,P3,M,S,GK,GL,NX,NY,NP)
      DOUBLE PRECISION AEST(10,10),P2(10,10),P3(10,10)
      DOUBLE PRECISION M(10,10),S(10,10),GK(10,10),GL(10,10)
      DOUBLE PRECISION TEMP(10,10),GLT(10,10)
*
*
      CALL P15TRANSPOMATRIX(GL,GLT,NP,NY)
      CALL P05MATMUL(AEST,P2,P2,NX,NX,NX,NP,NX,NP)
      CALL P05MATMUL(M,P3,TEMP,NX,NP,NP,NP,NX,NP)
      CALL P04MATADD(P2,P2,TEMP,NX,NP)
      CALL P22MMATMUL(TEMP,GK,S,GLT,NX,NY,NY,NY,NY,NP,NX,NP)
      DO 1 I=1,NX
         DO 1 J=1,NP
    1 TEMP(I,J)=-TEMP(I,J)
      CALL P04MATADD(P2,P2,TEMP,NX,NP)
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*     Subroutine P28P3UPDATE for the matrix operation                *
*                                     T -1             -1            *
*               P3 := {[P3 - GL S GL ]  + delta * I }                *
*                                                                    *
*     P3 is a NPxNP dimensional matrix.                              *
*     delta has to be found                                          *
**********************************************************************
      SUBROUTINE P28P3UPDATE(P3,S,GL,NX,NY,NP,LCOUNT)
      DOUBLE PRECISION P3(10,10),S(10,10),GL(10,10),DELTA
      DOUBLE PRECISION TEMP(10,10),TEMP1(10,10),GLT(10,10),DET
*
*
      delta=1.0D-09
C      DO 10 I=1,NP
C   10    DELTA=DELTA+P3(I,I)
C      DELTA=.01/DELTA
*
*
      CALL P15TRANSPOMATRIX(GL,GLT,NP,NY)
      CALL P22MMATMUL(TEMP,GL,S,GLT,NP,NY,NY,NY,NY,NP,NP,NP)
      DO 1 I=1,NP
         DO 1 J=1,NP
    1 TEMP(I,J)=-TEMP(I,J)
      CALL P04MATADD(P3,P3,TEMP,NP,NP)
      CALL P13DMATINV(TEMP,P3,NP,DET)
      DO 2 I=1,NP
    2 TEMP(I,I)=TEMP(I,I)+DELTA
      CALL P13DMATINV(P3,TEMP,NP,DET)
      RETURN
      END
```

```
************************************************************************
*                                                                      *
*      Subroutine P29PARAEST for the matrix operation                  *
*                                                                      *
*               THETA := THETA + GL * ( Y - C X )                      *
*                                                                      *
************************************************************************
      SUBROUTINE P29PARAEST(THETA,GL,Y,CEST,XEST,NU,NX,NY,NP)
      DOUBLE PRECISION THETA(10),CEST(10,10),GL(10,10)
      DOUBLE PRECISION Y(10),XEST(10),RES(10)
*
*
      CALL P20VECTORRESIDUE(XEST,CEST,Y,RES,NX,NY)
      CALL P02MATVEC(GL,RES,RES,NP,NY)
      CALL P04VECADD(THETA,THETA,RES,NP)
      RETURN
      END


************************************************************************
*                                                                      *
*      Subroutine P30STATEEST for the matrix operation                 *
*                                                                      *
*               X := A * X + B * U + GK * ( Y - C X )                  *
*                                                                      *
************************************************************************
      SUBROUTINE P30STATEEST(AEST,BEST,XEST,U,GK,Y,CEST,NU,NX,NY,NP)
      DOUBLE PRECISION AEST(10,10),BEST(10,10),CEST(10,10),GK(10,10)
      DOUBLE PRECISION U(10),Y(10),XEST(10),RES(10)
*
*
      CALL P20VECTORRESIDUE(XEST,CEST,Y,RES,NX,NY)
      CALL P02MATVEC(GK,RES,RES,NX,NY)
      CALL P11STATETIMEUPDATE(AEST,XEST,BEST,U,RES,NX,NU)
      RETURN
      END
```

```
*******************************************************************************
*                                                                             *
*     Subroutine P31COMPKAPPA for the matrix operation                        *
*                                                                             *
*      PAY :=  (defintion see Chapter 2.5)                                    *
*      RHO :=  (defintion see Chapter 2.5)                                    *
*    KAPPA :=  (defintion see Chapter 2.5)                                    *
*                                                                             *
*                                                                             *
*******************************************************************************
      SUBROUTINE P31COMPKAPPA(NU,NX,NY,NP,AEST,P1,KAPPA,S,SINV,GK,DAEST,
     &DCEST,CEST,RHO,AT,CT,GKT,DCT,DAT,LCOUNT,K)
      DOUBLE PRECISION AEST(10,10),P1(10,10),PAY(10,10),KAPPA(10,10)
      DOUBLE PRECISION S(10,10),GK(10,10),RHO(10,10),GKT(10,10)
      DOUBLE PRECISION TEMP(10,10),AT(10,10),DAT(10,10),DAEST(10,10)
      DOUBLE PRECISION TEMP1(10,10),KAPT(10,10),CT(10,10),DCEST(10,10)
      DOUBLE PRECISION CEST(10,10),DCT(10,10),SINV(10,10)
      DOUBLE PRECISION PAYOLD(10,10),SUMSQ,HALFSUMSQ,PAYMULT(10,10,5)
*
*
      IF((LCOUNT.EQ.1).AND.(K.EQ.1)) THEN
         DO 101 I=1,NP
            DO 101 J=1,NX
               DO 101 J1=1,NX
                  PAYMULT(J,J1,I)=0.0
 101              IF(J.EQ.J1) PAYMULT(J,J,I)=1.0
      ELSE
         DO 102 I=1,NX
            DO 102 J=1,NX
 102           PAY(I,J)=PAYMULT(I,J,K)
      ENDIF
*
*
      I3=0
      DO 100 LOOP=1,11
*
*     RHO
*
      CALL P22MMATMUL(RHO,DCT,P1,CT,NY,NX,NX,NX,NX,NY,NY,NY)
      CALL P22MMATMUL(TEMP,CEST,PAY,CT,NY,NX,NX,NX,NX,NY,NY,NY)
      CALL PO4MATADD(RHO,TEMP,RHO,NY,NY)
      CALL P22MMATMUL(TEMP,CEST,P1,DCT,NY,NX,NX,NX,NX,NY,NY,NY)
      CALL PO4MATADD(RHO,TEMP,RHO,NY,NY)
*
*     KAPPA
*
      CALL P22MMATMUL(KAPPA,DAEST,P1,CT,NX,NX,NX,NX,NX,NY,NX,NY)
      CALL P22MMATMUL(TEMP,AEST,PAY,CT,NX,NX,NX,NX,NX,NY,NX,NY)
      CALL P04MATADD(KAPPA,TEMP,KAPPA,NX,NY)
      CALL P22MMATMUL(TEMP,AEST,P1,DCT,NX,NX,NX,NX,NX,NY,NX,NY)
      CALL P04MATADD(KAPPA,TEMP,KAPPA,NX,NY)
      CALL P05MATMUL(KAPPA,SINV,KAPPA,NX,NY,NY,NY,NX,NY)
      CALL P22MMATMUL(TEMP,GK,RHO,SINV,NX,NY,NY,NY,NY,NY,NX,NY)
      DO 2 I=1,NX
         DO 2 J=1,NY
    2 TEMP1(I,J)=-TEMP(I,J)
      CALL P04MATADD(KAPPA,TEMP1,KAPPA,NX,NY)
      CALL P15TRANSPOMATRIX(KAPPA,KAPT,NX,NY)
*
*
*     PAY
*
      CALL P22MMATMUL(TEMP1,KAPPA,S,GKT,NX,NY,NY,NY,NY,NX,NX,NX)
      CALL P22MMATMUL(TEMP,GK,RHO,GKT,NX,NY,NY,NY,NY,NX,NX,NX)
      CALL P04MATADD(TEMP1,TEMP,TEMP1,NX,NX)
      CALL P22MMATMUL(TEMP,GK,S,KAPT,NX,NY,NY,NY,NY,NX,NX,NX)
      CALL P04MATADD(TEMP1,TEMP,TEMP1,NX,NX)
      DO 1 I=1,NX
         DO 1 J=1,NX
    1 TEMP1(I,J)=-TEMP1(I,J)
      CALL P22MMATMUL(PAY,AEST,PAY,AT,NX,NX,NX,NX,NX,NX,NX,NX)
      CALL P04MATADD(PAY,TEMP1,PAY,NX,NX)
      CALL P22MMATMUL(TEMP1,DAEST,P1,AT,NX,NX,NX,NX,NX,NX,NX,NX)
      CALL P04MATADD(PAY,TEMP1,PAY,NX,NX)
      CALL P22MMATMUL(TEMP1,AEST,P1,DAT,NX,NX,NX,NX,NX,NX,NX,NX)
      CALL P04MATADD(PAY,TEMP1,PAY,NX,NX)
*****************************************************************
*     CHECK LIMITS                                              *
*****************************************************************
      DO 111 I1=1,NX
         DO 111 I2=1,NX
      IF (DABS(PAY(I1,I2)) .GT. 200.)THEN
      I3=1
      ELSE
      ENDIF
  111 CONTINUE
      DO 113 I1=1,NY
         DO 113 I2=1,NY
      IF (DABS(RHO(I1,I2)) .GT. 200.)THEN
      I3=1
      ELSE
      ENDIF
  113 CONTINUE
      DO 112 I1=1,NX
         DO 112 I2=1,NY
      IF(DABS(KAPPA(I1,I2)) .GT. 200.)THEN
      I3=1
      ELSE
      ENDIF
  112 CONTINUE
      IF(I3.EQ.1) THEN
      DO 1201 I=1,NX
         DO 1201 J=1,NY
 1201       KAPPA(I,J)=0.0
      GOTO 100
      ELSE
      ENDIF
******************************************************************
*     CHECK FOR DIVERGENCE                                       *
******************************************************************
      DO 104 I=1,NX
         DO 104 J=1,NY
  104       SUMSQ=SUMSQ+(PAYOLD(I,J)-PAY(I,J))**2
      IF(LOOP.EQ.6) HALFSUMSQ=SUMSQ
            DO 105 I=1,NX
         DO 105 J=1,NX
  105       PAYOLD(I,J)=PAY(I,J)
  100 CONTINUE
      IF(I3.EQ.1)THEN
      GOTO 1000
      ELSE
      ENDIF
*
*     IF SUMSQ IS TOO LARGE DO NOT UPDATE PAY AND SET KAPPA TO ZERO
*
      IF(SUMSQ.GT.HALFSUMSQ*2.) THEN
      DO 201 I=1,NX
         DO 201 J=1,NY
  201       KAPPA(I,J)=0.0
      ELSE
            DO 202 I=1,NX
         DO 202 J=1,NX
  202       PAYMULT(I,J,K)=PAY(I,J)
      ENDIF
*
 1000 RETURN
      END
```

```
***********************************************************************
*                                                                     *
*     Subroutine P32MMODIFY for the matrix operation                  *
*                                                                     *
*    i    i        i                                                  *
*   M := M + KAPPA * ( Y - C X )                                      *
*                                                                     *
*    i                         T -1     i   -1                -1    i *
* EPS := 0.5 * (( Y - C X ) S   * RHO * S  ( Y - C X ) + tr S  * RHO ) *
* (optional)                                                          *
***********************************************************************
      SUBROUTINE P32MMODIFY(NU,NX,NY,NP,AEST,P1,XEST,S,SINV,GK,Y,M,
     &THETA,CEST,EPS,LCOUNT)
      DOUBLE PRECISION AEST(10,10),AT(10,10),P1(10,10),KAPPA(10,10)
      DOUBLE PRECISION S(10,10),GK(10,10),GKT(10,10),RHO(10,10),EPS(10)
      DOUBLE PRECISION TEMP(10,10),TEMP1(10,10),DAEST(10,10),M(10,10)
      DOUBLE PRECISION DCEST(10,10),CEST(10,10),SINV(10,10),TEMP2(10)
      DOUBLE PRECISION Y(10),RES(10),XEST(10),THETA(10),DUMMY
      DOUBLE PRECISION CT(10,10),DAT(10,10),DCT(10,10)
*
      CALL P20VECTORRESIDUE(XEST,CEST,Y,RES,NX,NY)
      DO 100 K=1,NP
      DO 3 I=1,NX
         DO 3 J=1,NY
    3 KAPPA(I,J)=0.0D00
      CALL P15TRANSPOMATRIX(AEST,AT,NX,NX)
      CALL P15TRANSPOMATRIX(CEST,CT,NY,NX)
      CALL P15TRANSPOMATRIX(GK,GKT,NX,NY)
      CALL U06CDERI(CEST,DCEST,CT,DCT,THETA,K,NU,NX,NY,NP)
      CALL U07ADERI(AEST,DAEST,AT,DAT,THETA,K,NU,NX,NY,NP)
      CALL P31COMPKAPPA(NU,NX,NY,NP,AEST,P1,KAPPA,S,SINV,GK,DAEST,
     &DCEST,CEST,RHO,AT,CT,GKT,DCT,DAT,LCOUNT,K)
      CALL P02MATVEC(KAPPA,RES,TEMP2,NX,NY)
      DO 1 I=1,NX
    1 M(I,K)=M(I,K)+TEMP2(I)
*
*     COMPUTE EPS
*
      CALL P05MATMUL(SINV,RHO,TEMP,NY,NY,NY,NY,NY,NY)
      CALL P17MATTRACE(TEMP,NY,DUMMY)
      CALL P05MATMUL(SINV,TEMP,TEMP1,NY,NY,NY,NY,NY,NY)
      CALL P02MATVEC(TEMP1,RES,TEMP2,NY,NY)
      DO 2 I=1,NY
    2 DUMMY=DUMMY+RES(I)*TEMP2(I)
      EPS(K)=5.0D-01*DUMMY
  100 CONTINUE
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*     Subroutine P33COMP_XFIL for the matrix operation               *
*                                                                    *
*                                    *                               *
*                XFIL := XEST + GK [ Y - C XEST]                     *
*                                                                    *
*                    *          T      T                             *
*                  GK := ( P1 C + P2 D )/ S                          *
*        *                                                           *
*     GK is a NXxNY dimensional matrix.                              *
*                                                                    *
**********************************************************************
      SUBROUTINE P33COMP_XFIL(XEST,XFIL,Y,CEST,P1,P2,D,SINV,NP,NU,NX,NY)
      DOUBLE PRECISION GKSTAR(10,10),SINV(10,10),CEST(10,10),P2(10,10)
      DOUBLE PRECISION D(10,10),TEMP(10,10),CT(10,10)
      DOUBLE PRECISION DT(10,10),P1(10,10),XEST(10),XFIL(10),Y(10)
*
*
      CALL P15TRANSPOMATRIX(CEST,CT,NY,NX)
      CALL P15TRANSPOMATRIX(D,DT,NY,NP)
*              *
*     COMPUTE GK
*
      CALL P05MATMUL(P1,CT,TEMP,NX,NX,NX,NY,NX,NY)
      CALL P19MATCOPY(TEMP,GKSTAR,NX,NY)
      CALL P05MATMUL(P2,DT,TEMP,NX,NP,NP,NY,NX,NY)
      CALL P04MATADD(TEMP,TEMP,GKSTAR,NX,NY)
      CALL P05MATMUL(TEMP,SINV,GKSTAR,NX,NY,NY,NY,NX,NY)
*
*     COMPUTE FILTERED STATE ESTIMATE XFIL
*
      CALL P20VECTORRESIDUE(XEST,CEST,Y,RES,NX,NY)
      CALL P02MATVEC(GKSTAR,RES,RES,NX,NY)
      CALL P04VECADD(XFIL,XEST,RES,NP)

      RETURN
      END
```

Program Package UTILITI

```
************************************************************************
*                                                                      *
*           Subroutine U02INITIALIZE                                   *
*           to initialize  the true state vector X, the estimated     *
*           vectors XEST (state) and THETA (parameter) and the         *
*           covariance matrices P1 (state), P2, P3 (parameter).        *
*                                                                      *
*                                   Last Revision SEPTEMBER 27, 1987 *
*                                                                      *
************************************************************************
*
      SUBROUTINE U02INITIALIZE(NX,NU,NY,X,XEST,THETA,P1,P2,P3)
      DIMENSION P1(10,10),P2(10,10),P3(10,10)
      DIMENSION X(10),XEST(10),THETA(10)
      DOUBLE PRECISION P1,P2,P3,X,XEST,THETA
*
*
*     INITIALIZE REAL STATE VECTOR X, ESTIMATED STATE VECTOR XEST,
*     PARAMETER VECTOR THETA AND COVARIANCE MATRICES P1, P2, P3
*
*
      DO 20 I=1,10
         DO 30 J=1,10
            P1(I,J)=0.0
            P2(I,J)=0.0
            P3(I,J)=0.0
            IF(I.EQ.J) THEN
               P1(I,J)=10.0
               P3(I,J)=0.08
            ELSE
            ENDIF
   30    CONTINUE
   20 CONTINUE
      DO 100 I=1,10
         X(I)=0.0
         XEST(I)=0.0
         THETA(I)=0.0
  100 CONTINUE
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*          Subroutine UO3READNOISE                                   *
*          to read in zero mean, Gaussian distributed noise vectors  *
*          v (process-n.), e (measurement-n.) and u (input).         *
*                                                                    *
*                                    Last Revision AUGUST 20, 1987 *
*                                                                    *
**********************************************************************
*
      SUBROUTINE UO3READNOISE(NX,NU,NY,V,E,U)
      DIMENSION V(NX),E(NY),U(NU)
      DOUBLE PRECISION V,E,U
*
*
*
      READ (11,*) (V(I),I=1,NX)
      READ (22,*) (E(I),I=1,NY)
      READ (33,*) (U(I),I=1,NU)
*
      RETURN
      END
```

```
***********************************************************************
*                                                                     *
*     Subroutine UO4DDERI                                             *
*                                                                     *
*     This subroutine generates the matrix D which is the             *
*     derivative of the product CEST*XEST with respect to the         *
*     parameter vector THETA at the current estimate.                 *
*                                                                     *
***********************************************************************

      SUBROUTINE UO4DDERI(D,CEST,THETA,XEST,NX,NY,NP)
      DIMENSION D(10,10),CEST(NY,NX),THETA(NP),XEST(NX)
      DOUBLE PRECISION D,CEST,THETA,XEST
*
*     WRITE HERE THE PROGRAM TO COMPUTE THE DERIVATIVE
*     OF THE VECTOR CEST*XEST WITH RESPECT TO THE
*     CURRENT PARAMETER ESTIMATE THETA.
*
      RETURN
      END


***********************************************************************
*                                                                     *
*     Subroutine UO5MDERI                                             *
*                                                                     *
*     This subroutine generates the matrix M which is the             *
*     derivative of AEST*XEST + BEST*U with respect to the            *
*     parameter vector THETA at the current estimate.                 *
*                                                                     *
***********************************************************************

      SUBROUTINE UO5MDERI(M,AEST,BEST,THETA,XEST,U,NU,NX,NY,NP)
      DIMENSION M(10,10),AEST(NX,NX),BEST(NX,NU),THETA(NP),XEST(NX)
      DIMENSION U(NU)
      DOUBLE PRECISION M,AEST,BEST,THETA,XEST,U
*
*     WRITE HERE THE PROGRAM TO COMPUTE THE DERIVATIVE
*     OF THE VECTOR (AEST*XEST + BEST*U) WITH RESPECT
*     TO THE CURRENT PARAMETER ESTIMATE THETA.
*
      RETURN
      END
```

```
**********************************************************************
*                                                                    *
*     Subroutine U06CDERI                                            *
*                                                                    *
*     This subroutine generates the matrix DCEST (DCT) which is the  *
*     derivative of CEST (CT)with respect to the parameter THETA(k)  *
*     at the current estimate.                                       *
*                                                                    *
**********************************************************************
      SUBROUTINE U06CDERI(CEST,DCEST,CT,DCT,THETA,K,NU,NX,NY,NP)
      DOUBLE PRECISION CEST(10,10),DCEST(10,10),THETA(10),CT(10,10)
      DOUBLE PRECISION DCT(10,10)
*
*     WRITE HERE THE PROGRAM TO COMPUTE THE DERIVATIVE
*     OF THE MATRIX CEST WITH RESPECT TO THE K-TH ENTRY
*     OF THE CURRENT PARAMETER ESTIMATE THETA, AS WELL AS
*     THE DERIVATIVE OF THE TRANSPOSE OF CEST WITH RESPECT
*     TO THE K-TH ENTRY OF THE CURRENT PARAMETER ESTIMATE
*     THETA, AS WELL AS
*
      RETURN
      END
**********************************************************************
*                                                                    *
*     Subroutine U07ADERI                                            *
*                                                                    *
*     This subroutine generates the matrix DAEST (DT) which is the   *
*     derivative of AEST (DAT) with respect to the parameter THETA(k) *
*     at the current estimate.                                       *
*                                                                    *
**********************************************************************
      SUBROUTINE U07ADERI(AEST,DAEST,AT,DAT,THETA,K,NU,NX,NY,NP)
      DOUBLE PRECISION AEST(10,10),DAEST(10,10),THETA(10)
      DOUBLE PRECISION AT(10,10),DAT(10,10)
*
*     WRITE HERE THE PROGRAM TO COMPUTE THE DERIVATIVE
*     OF THE MATRIX AEST WITH RESPECT TO THE K-TH ENTRY
*     OF THE CURRENT PARAMETER ESTIMATE THETA, AS WELL AS
*     THE DERIVATIVE OF THE TRANSPOSE OF AEST WITH RESPECT
*     TO THE K-TH ENTRY OF THE CURRENT PARAMETER ESTIMATE
*     THETA, AS WELL AS
*
      RETURN
      END
```

```
************************************************************************
*                                                                      *
*     Subroutine U08UPDATE_ABC                                         *
*                                                                      *
*     This subroutine updates the matrices A,B,C so that they match    *
*     the new parameter vector THETA.                                  *
*                                                                      *
************************************************************************
      SUBROUTINE U08UPDATE_ABC(AEST,BEST,CEST,THETA,NU,NX,NY,NP)
      DOUBLE PRECISION CEST(10,10),AEST(10,10),THETA(10),BEST(10,10)
*
*     WRITE PROGRAM, THAT TRANSFERS NEW PARAMETER ESTIMATES
*     INTO THE PARAMETER DEPENDENT MATRICES AEST, BEST AND
*     CEST.
*
      RETURN
      END
```

Appendix B

Program Listing RELS

```
************************************************************************
*                                                                      *
*              RECURSIVE EXTENDED LEAST-SQUARE ESTIMATOR               *
*                                                                      *
*                                   Last Revision FEBRUARY 24,1987 *
*                                                                      *
*  (algorithm from "Nichtlineare und adaptive Regelungssysteme",       *
*   J. Boecker, I. Hartmann, Ch. Zwanzig, Springer-Verlag Berlin,      *
*   1986, p.527)                                                       *
*                                                                      *
************************************************************************
*
      DOUBLE PRECISION A(10,10),B(10,10),C(10,10),LAMDA
      DOUBLE PRECISION X(10),P3(10,10),GL(10),THETA(10)
      DOUBLE PRECISION PARA(10),U(10),V(10),E(10),Y(10)
      DOUBLE PRECISION TEMPVEC(10), TEMPMAT(10,10),TEMP1,Y1
      DOUBLE PRECISION TEMPMAT1(10,10),PHI(10),RESIDUAL,V1
      INTEGER LCOUNT
*
*
*     OPEN CHANNELS
*
*
      OPEN (UNIT=11,FILE='[BJS8884.NOISE]N_SEQ12',STATUS='OLD')
      OPEN (UNIT=22,FILE='[BJS8884.NOISE]N_SEQ13',STATUS='OLD')
      OPEN (UNIT=33,FILE='[BJS8884.NOISE]CONSTO',STATUS='OLD')
      OPEN (UNIT=55,FILE='RELS10',STATUS='NEW')
*
*
*     READ ALL DATA NECESSARY TO RUN THE PROGRAM
*
*
      DATA A(1,1),A(1,2),A(2,1),A(2,2)/0.5,1.,-.9025,.95/
      DATA B(1,1),B(2,1),C(1,1),C(1,2)/0.,1.,1.,0./
      DATA NX,NU,NY,NP/1,1,1,4/
      DATA X(1),X(2)/0.,0./
      DATA THETA(1),THETA(2),THETA(3),THETA(4)/0.,0.,0.,0./
      DATA PHI/0.,0.,0.,0.,0.,0./
      DATA LAMDA,Y1/1.,0./
      DO 100 I=1,NP
         DO 100 J=1,NP
            IF(I.EQ.J) THEN
              P3(I,J)=1000.
            ELSE
              P3(I,J)=0.
            ENDIF
  100 CONTINUE
*
*
*     THIS IS THE BEGIN OF THE ACTUAL IDENTIFICATION LOOP
*
*
*
      LOOP=512
      DO 2000 LCOUNT=1,LOOP
C      WRITE(*,*) LCOUNT
************************************************************************
*     READ NOISE-SEQUENCES                                             *
************************************************************************
      READ(11,*) E(1)
      READ(22,*) V(1)
      READ(33,*) U(1)
C      V(1)=V(1)*1.
C      E(1)=E(1)*1.
************************************************************************
*     COMPUTE TRUE STATE VECTOR AND TRUE OUTPUT                        *
************************************************************************
      CALL P21OUTPUT(Y,C,X,E,NY,NX)
      CALL P11STATETIMEUPDATE(A,X,B,U,V,NX,NU)
************************************************************************
*     UPDATE VECTOR PHI                                                *
************************************************************************
      PHI(7)=PHI(6)
      PHI(6)=V1
      PHI(5)=PHI(4)
      PHI(4)=V1
      PHI(3)=U(1)
      PHI(2)=PHI(1)
      PHI(1)=-Y1
************************************************************************
*     COMPUTE GAIN GL                                                  *
************************************************************************
```

```
      CALL PO2MATVEC(P3,PHI,TEMPVEC,NP,NP)
      CALL P08(PHI,TEMPVEC,LAMDA,TEMP1,NP)
      DO 110 I=1,NP
  110    GL(I)=TEMPVEC(I)/TEMP1
***********************************************************************
*     COMPUTE UPDATED PARAMETER VECTOR THETA                          *
***********************************************************************
      CALL P10SCALARRESIDUE(PHI,THETA,Y(1),RESIDUAL,NP)
      DO 120 I=1,NP
  120    THETA(I)=THETA(I)+GL(I)*RESIDUAL
C      WRITE(44,40) RESIDUAL
      CALL P10SCALARRESIDUE(PHI,THETA,Y(1),V1,NP)
***********************************************************************
*     UPDATE P3                                                       *
***********************************************************************
      DO 130 I=1,NP
         DO 130 J=1,NP
  130       TEMPMAT(I,J)=PHI(I)*PHI(J)
      CALL P22MMATMUL(TEMPMAT1,P3,TEMPMAT,P3,NP,NP,NP,NP,NP,
     &                NP,NP,NP)
      DO 140 I=1,NP
         DO 140 J=1,NP
  140       P3(I,J)=(P3(I,J)-TEMPMAT1(I,J)/TEMP1)/LAMDA
*
      IF(LCOUNT.EQ.512) WRITE(*,51) (THETA(I), I=1,NP)
C      WRITE(55,50)(THETA(I), I=1,5)
      WRITE(*,50)(THETA(I), I=1,NP)
      Y1=Y(1)
*
 2000 CONTINUE
*
*
      CLOSE(55)
      CLOSE(45)
      CLOSE(44)
      CLOSE(33)
      CLOSE(22)
      CLOSE(11)
      STOP
   30 FORMAT(F9.5)
   50 FORMAT(5F6.2)
   51 FORMAT(7F6.2)
      END
```

Appendix C

Program Listing Noise Generator

```
C
C     RANDOM NUMBER GENERATOR
C
      INTEGER*4 SEED
      OPEN (UNIT=11,FILE='[BJS8884.NOISE]NOISE',STATUS='NEW')
      SEED = 824064364
      SIG  = 1.0
      WRITE(*,*) ' HOW MANY NUMBERS ?  '
      READ(*,*) N
      DO 100 I = 1,N
         RANDNUM = GAUSSN(SIG,SEED)
  100    WRITE(11,*)RANDNUM
      END

      REAL FUNCTION GAUSSN(SIG,SEED)
C
C     FOR GOOD RESULTS USE INITIL SEED = 824064364
C
      INTEGER*4 SEED
      GNOIZ=0.
      DO 10 I=1,12
         GNOIZ = GNOIZ + URAND(SEED)
   10 CONTINUE
      GAUSSN=SIG*(GNOIZ-6.0)
      RETURN
      END

      REAL FUNCTION URAND(SEED)
      INTEGER*4 B2E15,B2E16,MODLUS,HIGH15,HIGH31,LOW15,LOWPRD,
     &    MULT1,MULT2,OVFLOW,SEED
      DATA MULT1,MULT2/24112,26143/
      DATA B2E15,B2E16,MODLUS/32768,65536,2147483647/
      HIGH15 = SEED/B2E16
      LOWPRD = (SEED - HIGH15*B2E16)*MULT1
      LOW15  = LOWPRD/B2E16
      HIGH31 = HIGH15*MULT1 + LOW15
      OVFLOW = HIGH31/B2E15

      SEED = (((LOWPRD - LOW15*B2E16) - MODLUS) +
     &         (HIGH31 - OVFLOW*B2E15)*B2E16) + OVFLOW
      IF (SEED.LT.0) SEED = SEED + MODLUS

      HIGH15 = SEED/B2E16
      LOWPRD = (SEED - HIGH15*B2E16)*MULT2
      LOW15  = LOWPRD/B2E16
      HIGH31 = HIGH15*MULT2 + LOW15
      OVFLOW = HIGH31/B2E15

      SEED = (((LOWPRD - LOW15*B2E16) - MODLUS) +
     &         (HIGH31 - OVFLOW*B2E15)*B2E16) + OVFLOW
      IF (SEED.LT.0) SEED = SEED + MODLUS

      URAND = FLOAT(2*(SEED/256) + 1)/16777216.0
      RETURN
      END
```

Appendix D

Properties of the Expected Value Operator E

Let: $E\{\cdot\}$	Expected value operator
α	Random variable (unknown)
β	Random variable (unknown)
c	Deterministic variable (known)

The properties of $E\{\cdot\}$ are:

P 1:	$E\{c\alpha\}$	$=$	$c\,E\{\alpha\}$	
P 2:	$E\{\alpha+\beta\}$	$=$	$E\{\alpha\}+E\{\beta\}$	
P 3:	$E\{\alpha\beta\}$	$=$	$E\{\alpha\}\,E\{\beta\}$	if α,β are independent

Note: In the cases, where the argument of $E\{\cdot\}$ is a vector or a matrix, the rules given above apply to each entry of that vector or matrix.

Appendix E

Proof of the Matrix Inversion Lemma

Let:

$$\begin{array}{lcll} P_k & \in & \mathcal{R}^{nx \times nx} & \text{a positive definite matrix} \\ Q_k & \in & \mathcal{R}^{ny \times ny} & \text{a positive definite matrix} \\ M_k & \in & \mathcal{R}^{nx \times ny} & C^T Q^{-1} \\ C_k^T & \in & \mathcal{R}^{nx \times ny} & \\ D & \in & \mathcal{R}^{nx \times nx} & P^{-1} + MC \end{array}$$

The identity to be proven is:

$$\left(P^{-1} + MC\right)^{-1} = P - PM\left(I + CPM\right)^{-1} CP \tag{E.1}$$

Proof:

$$\begin{aligned} DD^{-1} &= I \\ &= D^{-1}P^{-1} + D^{-1}MC && \text{(E.2)} \\ P &= D^{-1} + D^{-1}MCP && \text{(E.3)} \\ P - D^{-1} &= D^{-1}MCP && \text{(E.4)} \end{aligned}$$

Post maltiplying equation (E.3) with M gives:

$$\begin{aligned} PM &= D^{-1}M + D^{-1}MCPM \\ &= D^{-1}M\left(I + CPM\right) && \text{(E.5)} \\ D^{-1}M &= PM\left(I + CPM\right)^{-1} && \text{(E.6)} \\ D^{-1}MCP &= PM\left(I + CPM\right)^{-1} CP && \text{(E.7)} \end{aligned}$$

Substituting (E.4) into (E.7) yields:

$$\begin{aligned} P - D^{-1} &= PM\left(I + CPM\right)^{-1} CP && \text{(E.8)} \\ D^{-1} &= P - PM\left(I + CPM\right)^{-1} CP && \text{(E.9)} \end{aligned}$$

Replacing D by $P^{-1} + MC$ and M by C^TQ^{-1} yields the desired result:

$$\begin{aligned}
\left(P^{-1} + C^TQ^{-1}C\right) &= P - PC^TQ^{-1}\left(I + CPC^TQ^{-1}\right)^{-1}CP \\
&= P - PC^T\left[\left(I + CPC^TQ^{-1}\right)Q\right]^{-1}CP \\
&= P - PC^T\left(Q + CPC^T\right)^{-1}CP \qquad \text{(E.10)}
\end{aligned}$$

Q.E.D.

Bibliography

[1] Alexander, S. T. "Fast Adaptive Filters: A Geometrical Approach", *IEEE Transaction on Acous., Speech and Sig. Processing*, vol. ASSP-34, October, 1986. pp. 18–28.

[2] Anderson, B. D. O. and Moore J.B. *Optimal Filtering*, Englewood Cliffs: Prentice-Hall., 1979

[3] Åström, Karl J. *Introduction to Stochastic Control Theory*, New York: Academic Press, 1970

[4] Åström, Karl J. and Eykhoff, P. "System Identification - a Survey", *Automatica*, vol. 7, 1971. pp. 123–167.

[5] Athans, Michael "Role and Use of the Stochastic Linear Quadratic Gaussian Problem in Control System Des‿gn", *IEEE Transactions on Automatic Control*, vol. AC-16, December, 1971. p . 529–552.

[6] Bierman, Gerald J. *Factorization Methods for Discrete Sequential Estimation*, New York: Academic Press, 1977

[7] Brammer, Karl and Siffling Gerhard *Kalman-Bucy-Filter*, München: R.Oldenburg Verlag., 1985

[8] Böcker, J., Hartmann, I. and Zwanzig, Ch. *Nichtlineare und adaptive Regelungssysteme*, Berlin: Springer Verlag., 1986

[9] Daum, Frederick E. "Exact Finite Dimensional Filters", *IEEE Transactions on Automatic Control*, vol. AC-31, July, 1986. pp. 616–622.

[10] ———. "New Exact Nonlinear Filters", submitted for publication in 1988

[11] Farison, James B., Graham, Richard E. and Shelton, Roy C. "Identification and Control of Linear Discrete Systems", *IEEE Transactions on Automatic Control*, vol. AC-14, August, 1969. pp. 438–442.

[12] O'Flynn, Michael *Probability, Random Variables and Random Processes*, New York: Harper & Row, 1982

[13] Frank, P.M. "Erhöhung der Robustheit und Zuverlässigkeit automatischer Prozesse", *Automatisierungstechnische Praxis*, 27.Jahrgang, Heft 2, 1985. pp. 64–71.

[14] Gelb, Arthur *Applied Optimal Estimation*, Massachusetts: M.I.T. Press, 1974

[15] Goodwin, Graham C. and Payne, Robert L. *Dynamic System Identification: Experiment Design and Data Analysis*, New York: Academic Press, 1977

[16] Maybeck, Peter S. *Stochastic Models, Estimation and Control Volume 3*, New York: Academic Press, 1982

[17] Maybeck, Peter S. *Stochastic Models, Estimation and Control Volume 3*, New York: Academic Press, 1982

[18] Maybeck, Peter S. *Stochastic Models, Estimation and Control Volume 3*, New York: Academic Press, 1982

[19] Grimble, M. J. and Johnson, M. A. "Recent Trends in Linear Optimal Quadratic Multivariable System Design", *IEE Proceedings Part D*, vol. 134, January, 1987. pp. 53-71.

[20] ————. "Implicit and Explicit LQG Self-tuning Controllers", *Automatica*, vol. 20, 1984. pp. 661-669.

[21] ————. "LQG Design of Discrete Systems Using a Dual Criterion", *IEE Proceedings Part D*, vol. 132, 1985. pp. 61–68.

[22] Hoel, Paul G. *Introduction to Mathematical Statistics*, New York: John Wiley & Sons, 1984

[23] Isermann, Rolf *Digital Control System*, Berlin: Springer Verlag, 1981

[24] ————. "Fehlerdiagnose mit Prozeßmodelen", *Technisches Messen*, 51. Jahrgang, Heft 10, 1984. pp. 345–355.

[25] Jazwins [illegible], A. H. *Stochastic Processes and Filtering Theory*, New York: Academic P[illegible]ss, 1971

[26] Johansson, Rolf "Parametric Models of Linear Multivariable Systems for Adaptive Control", *IEEE Transactions on Automatic Control*, vol. AC-32, April, 1987. pp. 303-313.

[27] Kalman R. E. "A New Approach to Linear Filtering and Prediction Problems", *Transactions ASME, Series D, Journal of Basic Engineering*, vol. 82, January, 1960. pp. 35–45.

[28] Kalman R. E. and Bucy R. S. "New Results in Linear Filtering and Prediction Theory", *Transactions ASME, Series D, Journal of Basic Engineering*, vol. 83, March, 1961. pp. 95-108.

[29] Królikowski, A. "Model Structure Selection in Linear System Identification", Department of Electrical Engineering, Eindhoven University of Technology, 1982, EUT Report 82-E-126

[30] Lewis, Frank L. *Optimal Estimation*, New York: John Wiley & Sons, 1986

[31] Ljung, Lennart "On Positive Real Transfer Function and the Convergence of some Recursive Schemes", *IEEE Transactions on Automatic Control*, vol. AC-22, August, 1977. pp. 539-551.

[32] ————. "Analysis of Recursive Stochastic Algorithms", *IEEE Transactions on Automatic Control*, vol. AC-22, August, 1977. pp. 551–575.

[33] ————. "Asymptotic Behavior of the Extended Kalman Filter as a Parameter Estimator for Linear Systems", *IEEE Transactions on Automatic Control*, vol. AC-24, February, 1979. pp. 36–50.

[34] ——————."Analysis of a General Recursive Prediction Error Identification Algorithm", *Automatica*, vol. 17, 1981. pp. 89–99.

[35] ——————.*Theory and Practice of Recursive Identification*, Cambridge: MIT Press, 1987

[36] ——————.*System Identification Theory For The User*, Englewood Cliffs: Prentice-Hall, 1987

[37] ——————.private communication

[38] Mendel, Jerry M. *Discrete Techniques of Parameter Estimation*, New York and Basel: Marcel Dekker Inc., 1973

[39] Maybeck, Peter S. *Stochastic Models, Estimation and Control Volume 1*, Orlando: Academic Press, 1979

[40] Maybeck, Peter S. *Stochastic Models, Estimation and Control Volume 2*, New York: Academic Press, 1982

[41] Maybeck, Peter S. *Stochastic Models, Estimation and Control Volume 3*, New York: Academic Press, 1982

[42] Nelson, Lawrence W. and Stear, Edwin "The Simultaneous On-Line Estimation of Parameters and States in Linear Systems", *IEEE Transactions on Automatic Control*, vol. AC-21, February, 1976. pp. 94–98.

[43] Papoulis, Athanasios *Probability, Random Variables and Stochastic Processes*, New York: McGraw Hill, 1984

[44] Rhodes, Ian B. "A Tutorial Introduction to Estimation and Filtering", *IEEE Transactions on Automatic Control*, vol. AC-16, December, 1971. pp. 688–706.

[45] Saridis, G. M. *Self-organizing Control of Stochastic Systems*, New York: Marcel Dekker Inc., 1977

[46] Special Issue on LQG Analysis and Design, *IEEE Transactions on Automatic Control*, vol. AC-16, December, 1971.

[47] Tse, Edison "On the Optimal Control of Stochastic Linear Systems", *IEEE Transactions on Automatic Control*, vol. AC-16, December, 1971. pp. 776–785.