CODEN:LUTEDX/(TEIE-5227)/1-36/(2006)

Industrial Electrical Engineering and Automation

Sensorless control of induction motors

Simulating the application of an extended Kalman filter together with a quadratic filter

Mari Lord

Dept. of Industrial Electrical Engineering and Automation

Lund University

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Pontifícia Universidade Católica do Rio de Janeiro
Department of Electrical Engineering

Lund Institute of Technology
Industrial Electrical Engineering

Sensorless control of induction motors

-

Simulating the application of an extended Kalman filter together with a quadratic filter

Mari Lord
Rio de Janeiro, 2006

Abstract

Title	Sensorless control of induction motors – Simulating the application of an extended Kalman filter together with a quadratic filter.
Author	Mari Lord, electrical engineering, Lund's Institute of Technology, Sweden.
Supervisors	Marcos Azevedo da Silveira, Department of Electrical Engineering, PUC-Rio, Rio de Janeiro, Brazil. Mats Alaküla, Department of Industrial Electrical Engineering and Automation, Lund's Institute of Technology, Sweden.
Report	Master thesis at the department of Industrial Electrical Engineering and Automation, Lund's Institute of Technology. Performed at university PUC-Rio, Rio de Janeiro, Brazil, from December 2005 to April 2006.
Purpose	The purpose is to create an algorithm that will make it possible to control induction motors without sensors. The idea is based on former research that consists of a method where the author of a doctoral thesis uses estimation instead of measuring to find out the speed of the rotor. Certain delimitations will be used in order to save memory and simplify the Matlab code.
Method	After interpretation of the doctoral thesis a program for the algorithm is written in the Matlab language. This algorithm is later to be implemented in Simulink via a so called S-function to be able to simulate and evaluate the results.
Conclusion	The filters turn out to show unstable results with a rotor speed that keeps on growing instead of stabilize when reaching its expected value. Since the theory shows that this filter set-up should be stable, it is interesting to keep on working on the algorithm in order to improve the performance of the filters.
Keywords	Induction motor, sensorless control, extended Kalman filter, rotor speed

Contents

1 Introduction

1.1 Background

Induction motors are electromechanical systems suitable for a large spectrum of industrial applications. It is necessary to be able to control the speed of these motor drives and the most common way of doing this is by using vector control. This method requires a speed sensor which is usually placed on the rotor shaft of the machine. The speed sensor has some disadvantages though, since it - besides from being costly - also reduces the robustness and reliability of the induction motor.

Consequently this has opened a new interesting area for research and during the last few years a variety of different solutions has reached the market and sensorless control has become industrial standard for medium and low performance applications. Artificial intelligence and neural networks are two examples of this new technology but they both show weak performance under speed changes and they also need offline calculations to work correctly. Since the induction motor is represented by a fifth order, nonlinear model with unknown state variables and external inputs, sensorless control is a challenging theoretical problem[1].

One of the results of all the research that has been made within this area is a doctoral thesis called "Motor speed estimation with sensorless vector control, employing an extended Kalman filter with estimation of the covariance of the noises", written by Jaime Antonio Gonzalez Castellanos[2]. In this thesis the author presents a solution of control of an induction motor without sensors where he uses an extended Kalman filter together with a quadratic filter in order to estimate the noise covariance matrices. These matrices are necessary for the calculations in the Kalman filter.

1.2 Purpose

Good results were obtained in the doctoral thesis and the aim of this master thesis is to take that work a little bit further. The main change is that the rotor speed will no longer be considered constant, which is a pretty rough approximation, but will instead be estimated in a completely closed system.

1.3 Method

After spending time on understanding the theory of the doctoral thesis, the work basically consists in creating an algorithm based on this theory but with modifications to suit the new idea. The algorithm is to be written in Matlab language and then implemented, with help of an S-function, in Simulink for simulations.

[1] http://www-lar.deis.unibo.it/woda/spider/af74.htm
[2] Gonzalez Castellanos J. A., (2004)

1.4 Delimitations

When writing the program in Matlab it is necessary to make some modifications in order not to occupy too much memory. Otherwise the memory would keep on growing and in the end require very much space, which partly would be very costly but at the same time it would also take a lot of time to execute the program.

1.5 Results

The filters do not manage to give desired results. It turns out to be difficult to control the system and the speed keeps on growing instead of stabilize in one expected value. In other words the filters act like they were unstable and apparently there is something in the algorithm that does not work satisfactory.

1.6 Thesis contents

The report starts with a theoretic part, explaining the background and former research. Then follows a part of algorithm creation, simulations and evaluations and finally a discussion with sum up, conclusions and proposals on further research.

Chapter 2: *Controlling the induction machine* gives a presentation of the induction machine and sensorless control in general and in Chapter 3: *Specification of the actual problem* a short explanation of the specific problem being handled in this thesis is presented. Chapter 4: *Filtering* treats filter theory, especially talking about the Kalman filter and its extended version, followed by an explanation of the secondary filter in Chapter 5: *The quadratic filter*. Further on, in Chapter 6: *Building the algorithm*, the actual algorithm is shown before reaching the simulations in Chapter 7: *Simulations and results* where some simulation issues are discussed before presenting the results. Finally in Chapter 8: *Conclusions* the thesis is wounded up with a discussion around the results.

In the end an appendix containing the algorithm written in Matlab code is to be find.

2 Controlling the induction machine

2.1 The induction machine

As mentioned in the introduction the induction machine, also known as asynchronous machine, is the most common type of electrical machine seen today in practice. It is used in high power as well as low power applications and obviously being so popular because of its cheap and simple construction[1].

2.1.1 The motor model

To understand the basic theory of this work it is necessary to have some knowledge of the induction motor itself. The dynamic model of the induction motor can be described in stator coordinates by the following equations:

$$\begin{bmatrix} i_{sd} \\ i_{sq} \\ \lambda_{rd} \\ \lambda_{rq} \\ \omega_r \end{bmatrix} = \begin{bmatrix} 1\ aD & 0 & bD & c\omega_r^{k-1}D & 0 \\ 0 & 1-aD & -c\omega_r^{k-1}D & bD & 0 \\ dD & 0 & 1-eD & -\omega_r^{k-1}D & 0 \\ 0 & dD & \omega_r^{k-1}D & 1-eD & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_{sd}^{k-1} \\ i_{sq}^{k-1} \\ \lambda_{rd}^{k-1} \\ \lambda_{rq}^{k-1} \\ \omega_r^{k-1} \end{bmatrix} + \begin{bmatrix} fD & 0 \\ 0 & fD \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{sd}^{k-1} \\ u_{sq}^{k-1} \end{bmatrix}$$

where the first vector is the state x(k), the last vector is the input u(k) and the output y(k) is given by C·x(k), where $C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$. The output consists in other words of the currents i_{sd} and i_{sq}. In this description the original motor model has been discretized and is to be sampled with an interval denoted by D. There has also been added a new state, ω_r, which is the state wished to be estimated. As a consequence of adding this new state, the motor model is no longer linear. Further on:

$$a = \frac{R_s}{\sigma \cdot L_s} + \frac{1-\sigma}{\sigma \cdot T_r}, \qquad b = \frac{M}{T_r \cdot \sigma \cdot L_s \cdot L_r}, \qquad c = \frac{M}{\sigma \cdot L_s \cdot L_r},$$

$$d = \frac{M}{T_r}, \qquad e = \frac{1}{T_r}, \qquad f = \frac{1}{\sigma \cdot L_s}, \qquad \sigma = 1 - \frac{M^2}{L_s \cdot L_r}, \qquad T_r = \frac{L_r}{R_r}$$

[1] Alaküla M., (2001)

$$\text{where}\left[\begin{array}{l} R_s = \text{the rotor resistance} \\ L_s = \text{the rotor inductance} \\ M = \text{the mutual inductance} \\ T_r = \text{the rotor time constant} \\ R_s = \text{the stator resistance} \\ \sigma = \text{the total linkage factor} \end{array}\right].$$

2.2 Sensorless control

The aim is to apply sensorless control to this machine and to do so by using two filters. An extended Kalman filter will work together with a quadratic filter in order to first linearize the non-linear system of this motor drive, then use the Kalman filter itself and eventually calculate the noise covariance matrices which are needed for the Kalman filter calculations.

The equation above can be written as $x(k) = A(k\text{-}1, \omega_r^{k\text{-}1}) \cdot x(k\text{-}1) + B \cdot u(k\text{-}1)$ with *k* representing a new iteration, the time. Then, as we wish to apply Kalman filter theory, the noises are simply added to the equations as follows:

$$x(k) = A(k\text{-}1, \omega_r^{k\text{-}1}) \cdot x(k\text{-}1) + B \cdot u(k\text{-}1) + G \cdot v(k\text{-}1)$$
$$y(k) = C \cdot x(k) + w(k)$$

Knowing that G is a weighting matrix for the noise of the system and can be chosen easily, these equations give us two unknowns: v(k-1) and w(k). They are noise sequences which can be represented by their covariance matrices Q and R and consequently need to be calculated or estimated. In this case a secondary filter, the quadratic filter, will be used to estimate the optimal values of the noise covariance matrices. Observing the order of performance, the initial values of the unknown covariance matrices are obviously important[1].

The purpose of the sensorless control is to estimate the motor speed, ω_r, instead of measuring it. Thus, an extended Kalman filter will after each iteration, for every new *k*, give a new value of ω_r – the rotor speed. The value of the rotor speed is needed in order to perform vector control. Vector control is the name of a group of methods that are based on the motor model. The methods consist in controlling the torque without being dependent on the currents that produce the flux and the torque. A Simulink model gives a better picture of how the induction machine works without sensorless control:

[1] Gonzalez Castellanos J. A., (2004)

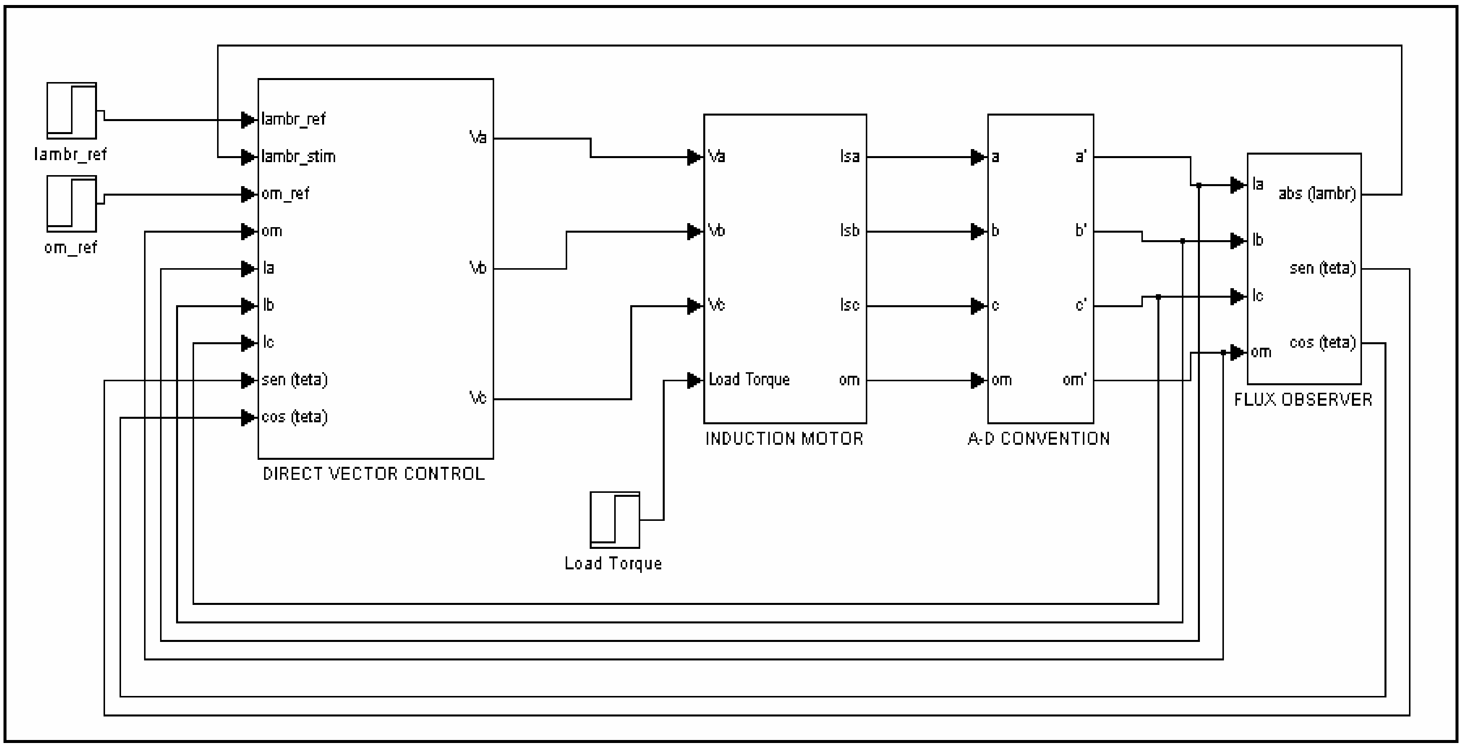

Figure 1. Simulink model of an induction motor with vector control.

To sum up the ideas of this general problem, the goal is to estimate the rotor speed with the help of two filters – an extended Kalman filter and a quadratic filter. The quadratic filter, also referred to as the secondary filter, is used in order to estimate optimal values of the noise covariance matrices Q and R which are needed for the performance of the Kalman filter calculations.

3 Specification of the actual problem

The ideas presented in the previous chapter underlie the doctoral thesis, "Motor speed estimation with sensorless vector control, employing an extended Kalman filter with estimation of the covariance of the noises", by Jaime Antonio Gonzalez Castellanos, mentioned in the introduction. In this work the author managed to obtain very good estimates of the rotor speed.

First of all it will therefore be necessary to create a program (which will be written in Matlab) for the algorithm that will give these estimates. Thus, the estimation of the rotor speed will be made in a way similar to the method used in the doctoral thesis. Then, in order to proceed with the research made in this doctoral thesis, the aim is to make the whole system work as a completely closed system. This idea is explained in *Figure 2*:

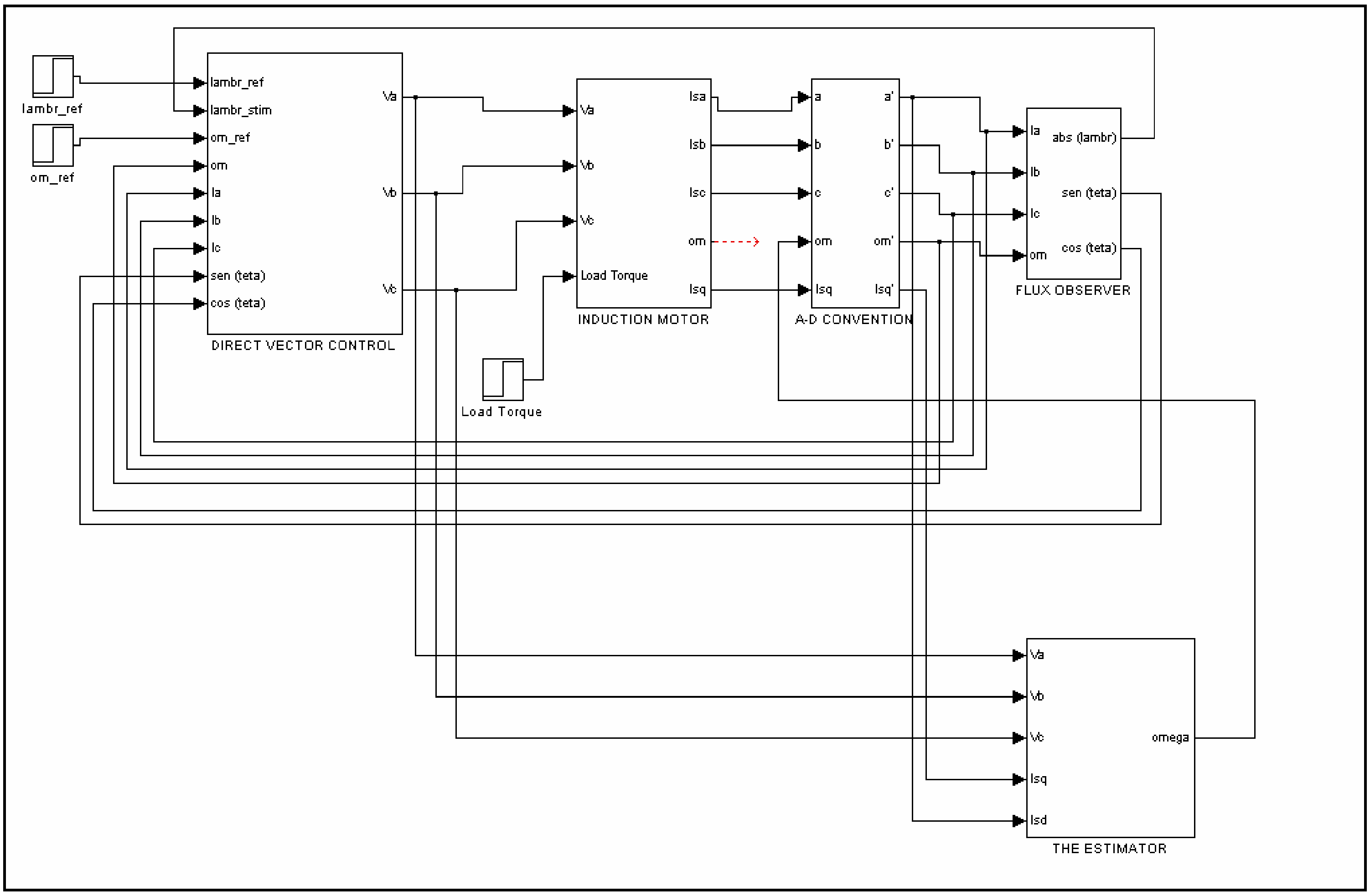

Figure 2. Simulink model of an induction motor with vector control and sensorless control.

Observe the important difference in the "INDUCTION MOTOR"-box and the output "om" that refers to omega, ω_r, in other words the rotor speed. This parameter is no longer the input to the omega in the "AC-DC CONVENTION"-box where it has been replaced with the ω_r estimated in "THE ESTIMATOR", the box that performs the sensorless control.

4 Filtering

There exist many methods for speed estimation in induction machines. One category that shows good performance is the kind of methods based on vector control where the motor model makes it possible to estimate the speed. Among these methods the extended Kalman filter is one of the more successful. To be able to understand the implementation of the methods used in this project will in this chapter be given a more detailed filter theory description.

4.1 The Kalman filter

The Kalman filter and its extended version are efficient and robust speed estimators for linear and non-linear systems respectively. The filters use knowledge of the dynamic system, its statistic characteristics and noise sources in order to produce an optimal estimate of the state and at the same time minimize the covariance error.

The filter is formed in terms of the state variables of the system and its solution is recursively computed. This means that every time an estimated state is updated, the only information that is needed in order to compute this new state is the previously estimated values and the new information given from the system at the current time. It is therefore necessary to store only the information of the value estimated most recently because at every new instant the new estimation is a projection on the former estimations[1].

But the Kalman filter needs certain information to be able to work in this way. First of all the filter has to have some knowledge of the basic parameters of the system. Then it is also necessary to know the values of the noise covariance matrices – of the system as well as of the observations. There are special methods to obtain the optimal values of these matrices. If there is lacking information about one or more of these matrices, the filter will be called sub-optimal.

The purpose of the Kalman filter is to produce an algorithm that makes it possible to compute an optimal estimate and the error of the covariance (which in this report will be referred to as P). The non-linear system on which this is applied can be described by the following equation:

$$x(k) = A(k\text{-}1, \omega_r^{k-1}) \cdot x(k\text{-}1) + B \cdot u(k\text{-}1) + G \cdot v(k\text{-}1)$$

where:
x(k) = the state vector of length *n* at time *k*,
A = a non-singular matrix for state transition of size *n x n*, depending on ω_r^{k-1} and therefore non-linear.
u(k-1) = the input vector at time *k-1*,
G = a weighting matrix for the noise of the system of size *n x n*,
v = a vector to describe the noise sequence of length *r*.

[1] Luenberger D. G., (1969)

The noise is of so called white Gaussian type, or normal distributed in other words, which means that its values are random, Gaussian (normal distributed) variables, uncorrelated and with zero mean when time goes towards infinity. When this is the case the noise can be totally represented by its covariance. Consequently the expected value

$$E\{v_k v_m\} = \begin{cases} Q & \text{for } m = k \\ 0 & \text{for } m \neq k \end{cases}$$

The observation system, or the output, is represented by:

$y(k) = C \cdot x(k) + w(k)$

and its noise characteristics given by $E\{w_k w_m\} = \begin{cases} R & \text{for } m = k \\ 0 & \text{for } m \neq k \end{cases}$

4.1.1 The Kalman filter algorithm

As described above the Kalman filter uses a recursive way to solve the problem. This can be seen clearly in the algorithm which is performed in the following steps:

· Prediction of the state

$$x^f(k) = A(k-1, \omega_r^{k-1}) \cdot x^a(k-1) + B \cdot u(k-1)$$

· Estimation of the matrix of the covariance error

$$P^f(k) = \Psi(k-1) \cdot P^a(k-1) \cdot \Psi^T(k-1) + G \cdot Q(k-1) \cdot G^T$$

· Calculation of the gain of the Kalman filter

$$K^{KB}(k) = P^f(k) \cdot C^T [C \cdot P^{f} \cdot (k) \cdot C^T + R(k-1)]^{-1}$$

· Estimation of the state

$$x^a(k) = x^f(k) + K^{KB}(k) [y(k) - C \cdot x^f(k)]$$

· Updating the matrix of the estimation covariance error

$$P^a(k) = [I - K^{KB}(k) \cdot C] P^f(k)$$

where $\Psi(k-1)$ is the derivative of the matrix $A(k-1, \omega_r^{k-1})$ with respect to $x(k-1)$.

Observe that when calculating $x^a(k)$ (the new state value) the Kalman filter gain is multiplied with the error of the output – the innovation. The innovation process has an important part of the solution in this work and the innovations are defined by:

$\eta(k) = y(k) - C\cdot x^f(k)$

where $x^f(k)$ equals the predicted x at time k, knowing the value of x(k-1).

The innovation process consists in creating a quadratic output, using estimators for the second order innovation moments $E\{\eta(k)\cdot\eta^T(k-m)\}$, where $0 \leq m \leq k$. In other words the new innovation, at time *k*, is multiplied with the old innovations, at time *k – m*. One important quality of the innovations is that when m = 0, they are orthogonal one to another, from which follows that $E\{\eta(k)\cdot\eta^T(k)\} = 0$[1].

4.2 The extended Kalman filter

Since we are working with a system that is non-linear, the ordinary Kalman filter is not a filter that will solve our problems. But it is possible, through a process of linearization, to extend the Kalman filter in order to use it on a non-linear system. This extended Kalman filter first linearizes the non-linear state from time *k-1* and then in its next step, at time *k* it uses this linearized state in the normal Kalman filter. Here, it is this extended Kalman filter that is called the primary filter and is represented by the formulas given above in the algorithm.

As mentioned in the beginning of this chapter it is necessary to know some of the basic parameters and to have knowledge of the noise covariance matrices. In this case the noise covariance matrices of the system and of the observation, Q and R respectively, are unknown and the filter will in other words be sub-optimal.

4.3 The sub-optimal filter

When one or more matrices are not fully known they will be replaced with a corresponding matrix and the filter will be called sub-optimal. In this case the S-matrices will represent the optimal P-matrices from the extended Kalman filter, principally in order to create the observation matrix F. The S-matrices are defined by:

$$S_i^f(k) = A(k-1)\cdot S_i^a(k-1)\cdot A^T(k-1) + G\cdot Q_i\cdot G^T$$
$$S_i^a(k) = [I - K^{KB}(k)\cdot C]\cdot S_i^f(k)\cdot[I-K^{KB}(k)\cdot C]^T + K^{KB}(k)\cdot R_i\cdot K^{KB\,T}(k)$$

and

$$S^a(k) = \sum_{i=1}^{N}\alpha_i(k)S_i^a\ ,\ S^f(k) = \sum_{i=1}^{N}\alpha_i(k)S_i^f$$

where α is a vector of length N and will be treated more in detail in Chapter 5.

The F-matrices are then constructed in the following way:

[1] Dee D. P., (1983)

$F_i(k,0) = C \cdot S_i^f(k) \cdot C^T + R_i(k)$

$F_i(k,1) = C \cdot \Psi(k-1) \cdot [I - K^{KB}(k-1) \cdot C] \cdot S_i^f(k) \cdot C^T - K^{KB} \cdot R_i(k)$

$F_i(k,m) = C \cdot \Psi(k-1) \cdot [I - K^{KB}(k-1) \cdot C] \cdot S_i^f(k) \cdot C^T$ for $m>1$

Since the matrices used in this sub-optimal filter are not exact, the estimated state will differ slightly from the real state and the estimation will obviously not be optimal. It has been shown though that $P^f(k) = S^f(k) + \text{error}(k)$ and that the error exponentially converges to zero[1]. Therefore, with a *k* that is big enough, the filter will give optimal results after a sufficient time.

[1] Dee D. P., (1983)

5 The quadratic filter

The purpose of the quadratic filter is to obtain the optimal values of the noise covariance matrices Q and R. To estimate these values, an algorithm will be proposed that is based on the innovation process of the primary filter.

5.1 The filter algorithm

First the noise covariance matrices are described as linear combinations of already known matrices, Q_i and R_i, according to a method presented by Bélanger in 1974[1].

$$Q(k) = \sum_{i=1}^{N} \alpha_i(k) \cdot Q_i \quad \text{and} \quad R(k) = \sum_{i=1}^{N} \alpha_i(k) \cdot R_i$$

where α is a vector of length N. It is necessary to find an observation model for this vector and in order to do so the estimator of α will be formed as a secondary filter of the sub-optimal extended Kalman filter. The equation that describes the observation model can be written as:[2]

$$\eta(k) \cdot \eta^T(k) = \sum_{i=1}^{N} \alpha_i(k) \cdot F_i(k,m) + \xi(k,m)$$

where η(k), as mentioned in chapter 4.1 is the innovation $y(k) - C \cdot x^f(k)$.

ξ(k,m) is a double sequence of random variables. In the same way as in the case of the noise sequences, these variables are considered uncorrelated, Gaussian and of zero mean why in other words they can be totally represented by their covariance matrix W(k,m). Since ξ(k,m) is a double sequence, W(k,m) will be a matrix of forth order moments.

$F_i(k,m)$ is a matrix, recursively calculated from the sub-optimal matrices $S_i^a(k)$ and $S_i^f(k)$ and used as an observation matrix, thus corresponding to the C-matrix when comparing with the extended Kalman filter.

Further on the gain of the extended Kalman filter K^{KB} will be represented by K and the actual covariance error matrix (in the Kalman filter referred to as P) is in the secondary filter named θ.

Next step is to form the observations, consisting of the innovations $\eta(k) \cdot \eta^T(k-m)$ and the F_i-matrices, in a special way in order to be coherent with the vector α, (that is to be observed). The idea is to stack the columns of the matrices and in this way create a vector (vec). If the matrix is symmetric the repeated terms are ignored (Tvec):

[1] Bélanger P. and Carrew B., (1973)
[2] Dee D. P., (1983)

$$z = \begin{cases} \mathrm{Tvec}[\eta\mathrm{vec}\cdot \eta^{T}(k)] + \xi(k,0) & \text{if } m = 0 \\ \mathrm{vec}[\eta\mathrm{vec} \cdot \eta^{T}(k-m)] + \xi(k,m) & \text{if } m > 0 \end{cases}$$

The $F_i(k,m)$- and the $W(k,m)$-matrices are constructed in the same way as the observations, $\eta(k)\cdot\eta^T(k\text{-}m)$, in the sense that they are all multiplied with their former values at time *k-m*. The same technique of stacking the matrix columns is therefore used to form the F- and the W-matrices and the quadratic filter can then be described with the following formulas:

$K(p) = \theta(p-1)\cdot F^T(p)\cdot[F(p)\cdot\theta(p-1)\cdot F^T(p) + W(p)]^{-1}$
$\theta(p) = [I - K(p)\cdot F(p)]\cdot\theta(p-1)$
$\alpha(p) = \alpha(p-1) + K(p)\cdot[z(p) - F(p)\cdot\alpha(p-1)]$

A new index, *p*, is introduced here specially for the quadratic filter and each *p* simply corresponds to each couple (*k,m*) in the primary filter. Thus, at every time *k*, the quadratic filter will make *k* iterations, since *m* runs from 0 to *k*. The quadratic filter passes on the values given from the last of these iteration, that is to say when *m* = *k*.

6 Building the algorithm

After obtaining enough knowledge of the theory presented in the previous chapters it is possible to start working on the algorithm. This one is supposed to be written in Matlab language and created based on the following formulas:

The states

$x(k) = [i_{sd}\ i_{sq}\ \lambda_{rd}\ \lambda_{rq}\ \omega_r]^T$
$u(k) = [u_{sd}\ u_{sq}]^T$
$y(k) = C \cdot x(k)$ where $C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$

The primary filter

$x^f(k) = A(k-1) \cdot x^a(k-1) + B \cdot u(k-1)$
$P^f(k) = \Psi(k-1) \cdot P^a(k-1) \cdot \Psi^T(k-1) + G \cdot Q(k-1) \cdot G^T$
$K^{KB}(k) = P^f(k) \cdot C^T [C \cdot P^f \cdot (k) \cdot C^T + R(k-1)]^{-1}$
$x^a(k) = x^f(k) + K^{KB}(k) [y(k) - C \cdot x^f(k)]$
$P^a(k) = [I - K^{KB}(k) \cdot C] P^f(k)$

where Q and R are the noise covariance matrices which can be described as a linear combination of simpler matrices:

$$Q(k) = \sum_{i=1}^{N} \alpha_i(k) \cdot Q_i \text{ and } R(k) = \sum_{i=1}^{N} \alpha_i(k) \cdot R_i$$

The secondary filter

$K(p) = \theta(p-1) \cdot F^T(p) [F(p) \cdot \theta(p-1) \cdot F^T(p) + W(p)]^{-1}$
$\theta(p) = [I - K(p) \cdot F(p)] \theta(p-1)$
$\alpha(p) = \alpha(p-1) + K(p) [z(p) - F(p) \cdot \alpha(p-1)]$

where p is a new index representing all (*k*,*m*)-couples in each iteration *k*.

6.1 Initial values

In the beginning, for the very first rotation of the program in Matlab, there are various values that are unknown in the algorithm. Assuming for example that the iteration will start at time *k* = 1, the following parameters will have to be initialized with a trustworthy value:

$x^a(0)$, $P^a(0)$, $\alpha(0)$, $S^a(0)$ (which is used to calculate the F-matrix) and $\theta(0)$. The matrix G also has to be set but this is a constant matrix and its values are taken from the ones used in former research[1].

[1] Gonzalez Castellanos J. A., (2004)

$$G = \begin{bmatrix} 10^{-6} & 0 & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 & 0 \\ 0 & 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 0 & 10^{-6} & 0 \\ 0 & 0 & 0 & 0 & 10^{-3} \end{bmatrix}$$

In the other cases different values are tested and $x^a(0)$, $P^a(0)$, $\alpha(0)$, $\theta(0)$ work perfectly fine with simple initializations such as:

$x^a(0)$ = a vector of zeros,
$P^a(0)$ and $\theta(0)$ = the identity matrix, I,
$\alpha(0)$ = a vector of ones,
while $S^a(0)$ turns out to be more difficult though, since in a lot of literature $S^a(0)$ is said to be a matrix of zeros[1]. However, this did not work in a pleasant way in this algorithm and after trying out some different values, $S^a(0)$ is set to the identity matrix with two extra values in order two facilitate the calculations.

$$\mathrm{Sa}(0) = \begin{bmatrix} 1 & 0.0001 & 0 & 0 & 0 \\ 0.0001 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

6.2 The p-index

The p-index is introduced for the quadratic filter and serves in order to make this secondary filter perform *k* iterations for each iteration of the primary filter. Thus, for every step *k+1* the secondary filter has to make one more iteration than the time before and it is easy to understand that this amount of iterations after a certain time will occupy a lot of memory and a lot of execution time. It is therefore convenient to program the algorithm in a way to limit this growth.

The assumption can be made that it is sufficient to look at ten (*k*,*m*)-couples at each iteration *k* and this means that *p* in other words runs from 0 to 9. Since it is interesting to include only the ten most recent observations in the calculations, *p* corresponds in this case to *m* from which follows that $p = 0$ is equivalent to (k,0) and $p = 9$ to (k,9). Consequently when using this system for the algorithm, *p* can be replaced by *m*.

In the initialization of the program, it has to be taken into consideration that when $k<10$ the program does not work as in the general case, when more time has passed and $k>=10$.

[1] Dee D. P., (1983)

7 Simulations and results

When the program for the algorithm works satisfactory in Matlab, it needs to be implemented in Simulink in order to make simulations of the program. Only by simulating it is possible to see how the program for the algorithm works together with the actual induction motor. This is evident since the only inputs to the algorithm, u(k-1) and y(k), are the stator voltages ($u = [u_{sd}\ u_{sq}]$) and the stator currents ($y = [i_{sd}\ i_{sq}]$) of the motor. The implementation is made through a so called S-function which constitutes a part of Simulink that makes it possible to add your own blocks in a Simulink model. These blocks can be created for example in Matlab.

The S-function block is hidden inside the block called THE ESTIMATOR (see *figure 2*, chapter 3). This is, as mentioned in the text, the box that represents the algorithm that performs the sensorless control and inside it looks like this:

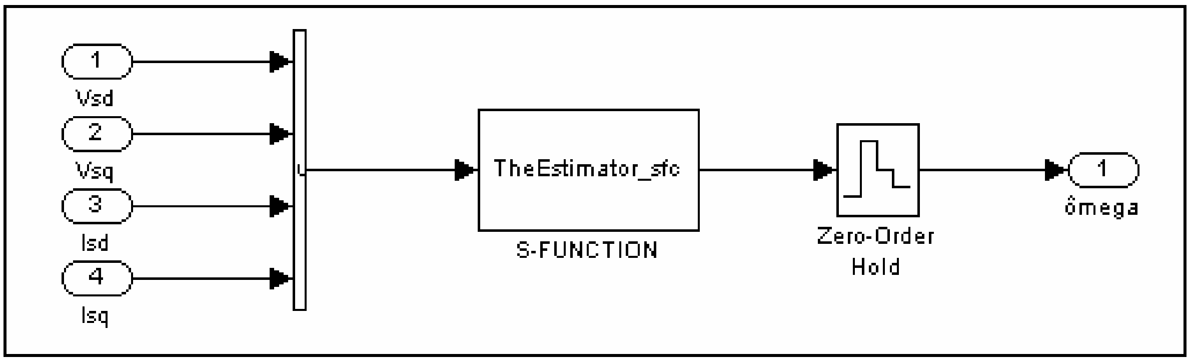

Figure 3. A Simulink model of the part that concludes the S-function.

When first executing the simulation it is necessary to look at the signal *ômega* coming from the estimation box and compare this signal with the one produced from the actual induction motor, *om*. In this way it is possible to see if the algorithm works satisfactory before trying to replace the real measured omega, *om*, with our new estimated omega, *ômega*.

One important thing to consider before starting the simulations is the mix of analog and digital signals appearing in the motor model and the estimator. Since the estimator is working in discreet time the inputs need to be digital signals and are therefore taken from appropriate places in the motor model (see *figure 2*, chapter 3). It is also important that the sample times in different blocks, as well as between the estimator and the motor model, are coherent.

7.1 Results

In the beginning it was difficult to make the program rotate and it stopped almost immediately with the message that the vector α had obtained values that were not considered values, (the error code is NaN in Matlab language which has the signification Not a Number). The reason of this turned out to be a numerical problem since the matrix that was to be inverted in the secondary filter became so small that the result of the inversion gave infinite numbers. Changing the order of the formulas and using new ideas for how to better create the matrices W and F, made the program advance slowly.

Later occurred new problems when the F-matrices started to grow too fast and θ at the same time was going very fast towards 0. When observing the algorithm and the formulas in the secondary filter it is understood that with a θ that equals 0 the secondary filter is not using any new information when performing its calculations. A θ that equals 0 means that the covariance error is 0 and the filter would with other words be perfect and stop making corrections. Therefore a temporary solution was introduced and θ was given a new value, bigger than 0, in the end of every iteration.

With this correction the filter started to show better performance and the rotor speed, omega, started to grow with an acceptable velocity towards its desired value. Once reaching this value the rotor speed did not decline as expected but kept on growing. It is observed that the problem with fast growing F-matrices still remained which results in the gain of the filter, K, becoming very big. The gains of the two filters, K and K^{KB}, are supposed to grow a lot in the beginning and then, after some time, slowly decrease and become smaller. If this growth is very big, as will be the case in the secondary filter if the F-matrices become too big, it is likely that the numbers reach a size that is not within the Matlab limit. Then the problem is a numerical question and it would be necessary to use a calculation program that can handle numbers of this size.

8 Conclusions

The system turns out to be pretty difficult to control. The method being used when creating the S-function consists in saving all the conditions in one long vector and it is therefore difficult to get an overview of what is happening with each state at every iteration. A better method would probably be to create some kind of library where it is possible to get and store each one of the conditions and in this way get a better overview and more control of how the situation is developing.

Another, or an additional, suggestion is to rewrite the formulas, maybe in another order, but there is also the possibility of writing them in a different way in order to more likely avoid the numerical problems that we have seen here.

During the last month the solution was getting closer and closer for each week and with more time it would certainly be possible to solve this problem and obtain a lot better results. The time dedicated to this job still has its value since it will be passed on to another student to keep on working on the solution. There are a lot of things that are interesting to continue working on, not only with the program presented here in this thesis but also with alternative solutions in order to realize and develop this idea. Perhaps it is possible for example to use another set of filters that has a more simple construction and doesn't use forth order moments.

References

Alaküla M., (2001) *Power electronic control*, Lund's university, Lund

Alaküla M., Gertmar L., Samuelsson O.(2002) *Elenergiteknik*, KFS, Lund

Bélanger P. and Carrew B., (1973) *Identification of optimum filter steady–state gain for systems with unknown noise covariances*, IEEE Trans. on automatic control, Vol. AC18, N 6, page 582 – 588

Dee D. P., (1983) *Computational aspects of adaptive filtering and applications to numerical weather prediction*, Doctoral thesis CI-6-83, Courant institute of mathematics, New York university, New York

Gonzalez Castellanos J. A., (2004) *Estimação de velocidade do motor com controle vetorial sem sensor, utilizando filtro estendido de Kalman com estimação da covariância dos ruídos*, Doctoral thesis (5716), PUC-Rio, Rio de Janeiro

Hanselman D., Littlefield B., (2003) *MATLAB 6® – Curso completo*, Pearson Education Inc., São Paulo. (Brazilian version)

Luenberger D. G., (1969) *Optimization by vector space methods*, John Wiley and Sons Inc., Stanford university, California

Peterson B., (1996) Induction machine speed estimation, observation on observers, Department of Industrial Electrical Engineering and Automation, Lund's Institute of Technology, Lund

Internet sources

Montanari M. (2003) "Sensorless Control of Induction Motors: Nonlinear and Adaptive Techniques", found on http://www-lar.deis.unibo.it/woda/spider/af74.htm, 28 feb 2006

Appendix

1. The algorithm written in Matlab:

```
function x = TheEstimator(t,x,Vsd,Vsq,Isd,Isq)

% The motor

Lr = 252.00e-3;                                   %rotor inductivity
Ls = 252.0e-3;                                    %stator inductivity
Rr = 1.87;                                        %rotor resistance
Rs = 3.88;                                        %stator resistance
M = 236.3e-3;                                     %main inductivity
Tr = Lr/Rr;
sig1 = 1-(M^2)/(Ls*Lr);

a = (Rs/(sig1*Ls)+(1-sig1)/(sig1*Tr));
b = M/(Tr*sig1*Ls*Lr);
c = M/(sig1*Ls*Lr);
d = M/Tr;
e = 1/Tr;
f = 1/(sig1*Ls);

delta = 0.0001;                                   % sampling interval


% The states

xa = [x(1) x(2) x(3) x(4) x(5)]';
Pa = [x(6) x(7) x(8) x(9) x(10); x(11) x(12) x(13) x(14) x(15); x(16) x(17) x(18) x(19) x(20); x(21) x(22)
x(23) x(24) x(25); x(26) x(27) x(28) x(29) x(30)];
Sa = [x(31) x(32) x(33) x(34) x(35); x(36) x(37) x(38) x(39) x(40); x(41) x(42) x(43) x(44) x(45); x(46)
x(47) x(48) x(49) x(50); x(51) x(52) x(53) x(54) x(55); x(56) x(57) x(58) x(59) x(60); x(61) x(62) x(63)
x(64) x(65); x(66) x(67) x(68) x(69) x(70); x(71) x(72) x(73) x(74) x(75); x(76) x(77) x(78) x(79) x(80);
x(81) x(82) x(83) x(84) x(85); x(86) x(87) x(88) x(89) x(90); x(91) x(92) x(93) x(94) x(95); x(96) x(97)
x(98) x(99) x(100); x(101) x(102) x(103) x(104) x(105); x(106) x(107) x(108) x(109) x(110); x(111)
x(112) x(113) x(114) x(115); x(116) x(117) x(118) x(119) x(120); x(121) x(122) x(123) x(124) x(125);
x(126) x(127) x(128) x(129) x(130); x(131) x(132) x(133) x(134) x(135); x(136) x(137) x(138) x(139)
x(140); x(141) x(142) x(143) x(144) x(145); x(146) x(147) x(148) x(149) x(150); x(151) x(152) x(153)
x(154) x(155); x(156) x(157) x(158) x(159) x(160); x(161) x(162) x(163) x(164) x(165); x(166) x(167)
x(168) x(169) x(170); x(171) x(172) x(173) x(174) x(175); x(176) x(177) x(178) x(179) x(180); x(181)
x(182) x(183) x(184) x(185); x(186) x(187) x(188) x(189) x(190); x(191) x(192) x(193) x(194) x(195);
x(196) x(197) x(198) x(199) x(200); x(201) x(202) x(203) x(204) x(205)];
Inov = [x(206) x(207) x(208) x(209) x(210) x(211) x(212) x(213) x(214) x(215); x(216) x(217) x(218)
x(219) x(220) x(221) x(222) x(223) x(224) x(225)];
Kkb = [x(226) x(227); x(228) x(229); x(230) x(231); x(232) x(233); x(234) x(235)];
U = [x(236) x(237)]';
Theta = [x(238) x(239) x(240) x(241) x(242) x(243) x(244); x(245) x(246) x(247) x(248) x(249) x(250)
x(251); x(252) x(253) x(254) x(255) x(256) x(257) x(258); x(259) x(260) x(261) x(262) x(263) x(264)
x(265); x(266) x(267) x(268) x(269) x(270) x(271) x(272); x(273) x(274) x(275) x(276) x(277) x(278)
x(279); x(280) x(281) x(282) x(283) x(284) x(285) x(286)];
alfa = [x(287) x(288) x(289) x(290) x(291) x(292) x(293)]';
M = [x(294) x(295) x(304) x(305) x(314) x(315) x(324) x(325)...;
        x(296) x(297) x(306) x(307) x(316) x(317) x(326) x(327)...;
        x(298) x(299) x(308) x(309) x(318) x(319) x(328) x(329)...;
        x(300) x(301) x(310) x(311) x(320) x(321) x(330) x(331)...;
```

```
     x(302) x(303) x(312) x(313) x(322) x(323) x(332) x(333)...];
V = [x(994) x(995) x(996) x(997) x(998) x(999) x(1000) x(1001) x(1002) x(1003) x(1004) x(1005)
x(1006)     x(1007) x(1008) x(1009) x(1010) x(1011) x(1012) x(1013); x(1014) x(1015) x(1016) x(1017)
x(1018) x(1019) x(1020) x(1021) x(1022) x(1023) x(1024) x(1025) x(1026) x(1027) x(1028) x(1029)
x(1030) x(1031) x(1032) x(1033)];
k = x(1034);


% Constant values and matrices

L = 10;                                   % Iterations in the secundary filter
B = [f*delta 0;0 f*delta;0 0;0 0;0 0];
C = [1 0 0 0 0;0 1 0 0 0];
G = [10e-6 0 0 0 0;0 10e-6 0 0 0;0 0 10e-6 0 0;0 0 0 10e-6 0;0 0 0 0 10e-3];
Pro5 = spdiags(ones(5,1),0,5,7);          % Creates a 5x7-matrix with 5 ones in the first "diagonal"
Pro2 = spdiags(ones(2,1),5,2,7);          % Creates a 2x7-matrix with 2 ones in the 6th "diagonal"
eps = zeros(5,5);
mu = zeros(2,2);
SF = zeros(5,5);
MA = zeros(5,16);


% Inputs

u = [Vsd,Vsq,Isd,Isq]';
U = [u(1) u(2)]';
y = [u(3) u(4)]';


% The primary filter:

omega = x(5);
A = [1-a*delta 0 b*delta c*delta*omega 0;0 1-a*delta -c*delta*omega b*delta 0;d*delta 0 1-e*delta -
delta*omega 0;0 d*delta delta*omega 1-e*delta 0;0 0 0 0 1];
Psi = [1-a*delta 0 b*delta c*delta*omega c*delta*xa(4);0 1-a*delta -c*delta*omega b*delta -
c*delta*xa(3);d*delta 0 1-e*delta -delta*omega -delta*xa(4);0 d*delta delta*omega 1-e*delta
delta*xa(3);0 0 0 0 1];

  xf = A*xa + B*U;
  Pf = ((Psi*Pa*Psi' + G*diag(Pro5*alfa)*G')+((Psi*Pa*Psi' + G*diag(Pro5*alfa)*G')'))/2;
  Kkb = Pf*C'*((((C*Pf*C' + diag(Pro2*alfa))^-1)+((C*Pf*C' + diag(Pro2*alfa))^-1)')/2);
  inov = y - C*xf;
  xa = xf + Kkb*inov;
  Pa = (((eye(5) - Kkb*C)*Pf)+((eye(5) - Kkb*C)*Pf)')/2;

  U = [Vsd Vsq]';


% Modification for the first ten iterations

  if k<10
    L=k;
  end


% Calculating the Fi-matrices

  for i=1:7
    j=8;
    r=1;
```

```
if i<6
   eps(i,i)=1;
else
   mu(i-5,i-5)=1;
end
Sai = Sa((1+(i-1)*5):(5+(i-1)*5),1:5);
Sf = ((Psi*Sai*Psi'+G*eps*G')+(Psi*Sai*Psi'+G*eps*G')')/2;
Sai = (((eye(5)-Kkb*C)*Sf*(eye(5)-Kkb*C)'+Kkb*mu*Kkb')+((eye(5)-Kkb*C)*Sf*(eye(5)-
Kkb*C)'+Kkb*mu*Kkb')')/2;
x(31+25*(i-1):35+25*(i-1)) = Sai(1,:); x(36+25*(i-1):40+25*(i-1)) = Sai(2,:);x(41+25*(i-
1):45+25*(i-1)) = Sai(3,:);x(46+25*(i-1):50+25*(i-1)) = Sai(4,:);x(51+25*(i-1):55+25*(i-1)) =
Sai(5,:);
SF = SF + Sf;
for m=1:L
   if m==1
      Ma1 = Sf*C';
      F = C*Ma1 + mu;
      F1 = F(1,1); F2 = F(2,1); F4 = F(2,2);
      vecF(1,i) = F1; vecF(2,i) = F2; vecF(3,i) = F4;
   elseif m==2
      KKB = [x(226) x(227); x(228) x(229); x(230) x(231); x(232) x(233); x(234) x(235)];
      Mi = M(1:5,(1+(i-1)*20):(2+(i-1)*20));
      Ma2 = Psi*(eye(5)-KKB*C)*Mi-KKB*mu;
      F = C*Ma2;
      F1 = F(1,1); F2 = F(1,2); F3 = F(2,1); F4 = F(2,2);
      vecF(4,i) = F1; vecF(5,i) = F3; vecF(6,i) = F2; vecF(7,i) = F4;
   else
      KKB = [x(226) x(227); x(228) x(229); x(230) x(231); x(232) x(233); x(234) x(235)];
      Mi = M(1:5,(3+(i-1)*20+(m-3)*2):(4+(i-1)*20+(m-3)*2));
      Ma = Psi*(eye(5)-KKB*C)*Mi;
      MA(1:5,r:r+1) = Ma;
      F = C*Ma;
      F1 = F(1,1); F2 = F(1,2); F3 = F(2,1); F4 = F(2,2);
      vecF(j,i) = F1; vecF(j+1,i) = F3; vecF(j+2,i) = F2; vecF(j+3,i) = F4;
      j = j+4;
      r=r+2;
   end
end
if k==1
     x(294+100*(i-1)) = Ma1(1,1); x(295+100*(i-1)) = Ma1(1,2);
     x(296+100*(i-1)) = Ma1(2,1); x(297+100*(i-1)) = Ma1(2,2);
     x(298+100*(i-1)) = Ma1(3,1); x(299+100*(i-1)) = Ma1(3,2);
     x(300+100*(i-1)) = Ma1(4,1); x(301+100*(i-1)) = Ma1(4,2);
     x(302+100*(i-1)) = Ma1(5,1); x(303+100*(i-1)) = Ma1(5,2);
elseif k==2
     x(294+100*(i-1)) = Ma1(1,1); x(295+100*(i-1)) = Ma1(1,2);
     x(296+100*(i-1)) = Ma1(2,1); x(297+100*(i-1)) = Ma1(2,2);
     x(298+100*(i-1)) = Ma1(3,1); x(299+100*(i-1)) = Ma1(3,2);
     x(300+100*(i-1)) = Ma1(4,1); x(301+100*(i-1)) = Ma1(4,2);
     x(302+100*(i-1)) = Ma1(5,1); x(303+100*(i-1)) = Ma1(5,2);
     x(304+100*(i-1)) = Ma2(1,1); x(305+100*(i-1)) = Ma2(1,2);
     x(306+100*(i-1)) = Ma2(2,1); x(307+100*(i-1)) = Ma2(2,2);
     x(308+100*(i-1)) = Ma2(3,1); x(309+100*(i-1)) = Ma2(3,2);
     x(310+100*(i-1)) = Ma2(4,1); x(311+100*(i-1)) = Ma2(4,2);
     x(312+100*(i-1)) = Ma2(5,1); x(313+100*(i-1)) = Ma2(5,2);
else
   x(294+100*(i-1)) = Ma1(1,1); x(295+100*(i-1)) = Ma1(1,2);
     x(296+100*(i-1)) = Ma1(2,1); x(297+100*(i-1)) = Ma1(2,2);
     x(298+100*(i-1)) = Ma1(3,1); x(299+100*(i-1)) = Ma1(3,2);
     x(300+100*(i-1)) = Ma1(4,1); x(301+100*(i-1)) = Ma1(4,2);
```

```
          x(302+100*(i-1)) = Ma1(5,1); x(303+100*(i-1)) = Ma1(5,2);
          x(304+100*(i-1)) = Ma2(1,1); x(305+100*(i-1)) = Ma2(1,2);
          x(306+100*(i-1)) = Ma2(2,1); x(307+100*(i-1)) = Ma2(2,2);
          x(308+100*(i-1)) = Ma2(3,1); x(309+100*(i-1)) = Ma2(3,2);
          x(310+100*(i-1)) = Ma2(4,1); x(311+100*(i-1)) = Ma2(4,2);
          x(312+100*(i-1)) = Ma2(5,1); x(313+100*(i-1)) = Ma2(5,2);
        o=0;
        for s=1:8
            x(314+100*(i-1)+(s-1)*10) = MA(1,1+o);
            x(315+100*(i-1)+(s-1)*10) = MA(1,2+o);
            x(316+100*(i-1)+(s-1)*10) = MA(2,1+o);
            x(317+100*(i-1)+(s-1)*10) = MA(2,2+o);
            x(318+100*(i-1)+(s-1)*10) = MA(3,1+o);
            x(319+100*(i-1)+(s-1)*10) = MA(3,2+o);
            x(320+100*(i-1)+(s-1)*10) = MA(4,1+o);
            x(321+100*(i-1)+(s-1)*10) = MA(4,2+o);
            x(322+100*(i-1)+(s-1)*10) = MA(5,1+o);
            x(323+100*(i-1)+(s-1)*10) = MA(5,2+o);
          o=o+2;
        end
    end
    if i<6
        eps(i,i)=0;
    else
        mu(i-5,i-5)=0;
    end
end
```

% Creating the innovations

```
if k<10
    p=k;
    for m=2:L
        Inov(1:2,p)=Inov(1:2,p-1);
        p=p-1;
    end
    Inov(1:2,1)=inov
else
    p=10;
    for m=2:L
        Inov(1:2,p)=Inov(1:2,p-1);
        p=p-1;
    end
    Inov(1:2,1)=inov;
end
```

% Creating the V:s (the moments of the W-matrices)

```
v = C*SF*C'+diag(Pro2*alfa);

if k<10
    p=2*k-1;
    for m=2:L
        V(1:2,p:p+1)=V(1:2,p-2:p-1);
        p = p-2;
    end
    V(1:2,1:2) = v;
else
```

```
    p=19;
    for m=2:L
        V(1:2,p:p+1)=V(1:2,p-2:p-1);
        p = p-2;
    end
    V(1:2,1:2) = v;
end
a = V(1,1);
b = (V(1,2)+V(2,1))/2;
c = V(2,2);
if a<1e-3
    a = 1e-3;
elseif c<1e-3
    c = 1e-3;
end
```

% Creating the z:s and the F:s

```
s = 3;                          % index for W
j = 4;
for m=1:L
    z = inov*Inov(1:2,m)';
    z1 = z(1,1);
    z2 = z(1,2);
    z3 = z(2,1);
    z4 = z(2,2);
    if m==1
        z = [z1 z2 z4]';
        F = vecF(1:3,1:7);
    else
        z = [z1 z2 z3 z4]';
        F = vecF(j:j+3,1:7);
        j=j+4;
    end
```

% Creating the W:s

```
    if m==1
        W = [2*a*a 2*a*b 2*b*b;2*a*b b*b+a*c 2*b*c;2*b*b 2*b*c 2*c*c];
    else
    A = V(1,s);
    B = (V(1,s+1)+V(2,s))/2;
    C = V(2,s+1);
    if A<1e-3
        A = 1e-3;
    elseif C<1e-3
        C = 1e-3;
    end
    W = [a*A b*A a*B b*B;b*A c*A b*B c*B;a*B b*B a*C b*C;b*B c*B b*C c*C];
    s = s+2;
    end
```

% The secundary filter

```
    Mat = F*Theta*F' + W;
    MatI = ((Mat^-1)+(Mat^-1)')/2
```

```
      K = Theta*F'*MatI;
      %Theta = (((eye(7) - K*F)*Theta)+(((eye(7) - K*F)*Theta)'))/2; Theta
      alfa = alfa + K*(z - F*alfa); alfa
   end

   k=k+1;
```

% New values and matrices

```
   Theta = 100*(eye(7));
   x = [xa(1) xa(2) xa(3) xa(4) xa(5) Pa(1,1) Pa(1,2) Pa(1,3) Pa(1,4) Pa(1,5) Pa(2,1)….k];
```

2. The S-function written in Matlab

function [sys,x0,str,ts] = The_Estimator_sf(t,x,u,flag)

```
switch flag,
```

% Initialization

```
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes;
```

% Update

```
  case 2,
    sys = mdlUpdate(t,x,u);
```

% Output

```
  case 3,
    sys = mdlOutputs(t,x,u);
```

% Terminate

```
  case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
```

% Return the sizes, initial conditions, and sample times for the S-function.

function [sys,x0,str,ts]=mdlInitializeSizes

```
sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 1034;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

t = 100;
ny = 0.0001/7;
my = 1/7;
```

```
x0 =
[0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,my,ny,0,0,0,ny,my,0,0,0,0,0,my,0,0,0,0,0,my,0,0
,0,0,0,my,my,ny,0,0,0,ny,my,0,0,0,0,0,my,0,0,0,0,0,my,0,0,0,0,0,my,my,ny,0,0,0,ny,my,0,0,0,0,0,my,0,0,
0,0,0,my,0,0,0,0,0,my,my,ny,0,0,0,ny,my,0,0,0,0,0,my,0,0,0,0,0,my,0,0,0,0,0,my,my,ny,0,0,0,ny,my,0,0,0
,0,0,my,0,0,0,0,0,my,0,0,0,0,0,my,my,ny,0,0,0,ny,my,0,0,0,0,0,my,0,0,0,0,0,my,0,0,0,0,0,my,my,ny,0,0,0,
ny,my,0,0,0,0,0,my,0,0,0,0,0,my,0,0,0,0,0,my,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,t,0,0,0,0,0,0,0,t,0,0,0,0,0,0,0,t,0,0,0,0,0,0,0,t,0,0,0,0,0,0,0,t,0,0,0,0,0,0,0,t,0,0,0,0,0,0,0,t,1,1,1,1,1,1,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,1];

str = [];

ts  = [0.0001 0];               % Sample period of 0.0001 seconds (10 kHz)


% Handle discrete state updates, sample time hits, and major time step requirements.

function sys = mdlUpdate(t,x,u)

  Vsd = u(1);
  Vsq = u(2);
  Isd = u(3);
  Isq = u(4);

  sys = The_Estimator(t,x,Vsd,Vsq,Isd,Isq);


% Return the output vector for the S-function

function sys = mdlOutputs(t,x,u)
q = x(5)
sys = q;
```