MODELING, IDENTIFICATION AND CONTROL, 1988, VOL. 9, NO. 4, 179–189
doi:10.4173/mic.1988.4.2

A data and program structure for a modular extended Kalman filter

INGAR SOLBERG†

Keywords: *extended Kalman filter, nonlinear filtering, state space methods, computer programming*

The paper presents a data and program structure that makes it easier to implement a nonlinear process or measurement model when using the extended Kalman filter. This is achieved by a composite data type containing both the estimated value and covariance information. The basic operators (+, −, ·, /) and common functions are implemented for this data type. These enable a model for the extended Kalman filter to be implemented as easily as a discrete-time simulation model.

1. Introduction

The Kalman filter is an optimal filter for the state estimation of linear stochastic processes. For a discrete process description the transition matrix is specified and used both for state prediction and covariance update:

$$\mathbf{x}(k+1) = \Phi(k)\mathbf{x}(k) + v(k)$$

$$X(k+1) = \Phi(k)X(k)\Phi(k)^{\mathrm{T}} + V(k)$$

For nonlinear processes, a modified version called the extended Kalman filter may be used. In this case, the state space model is used for the state prediction, while its Jacobian is used for the covariance update.

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{v}(k), k)$$

$$X(k+1) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} X(k) \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \frac{\partial \mathbf{f}}{\partial \mathbf{v}} V(k) \left(\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right)^{\mathrm{T}}$$

The conventional way to do this is to specify both the state space model and its Jacobian by separate program statements. This means that the Jacobian must be found by differentiation and then coded. Errors in this process will lead to an expression for the Jacobian which is not consistent with the state space model. These problems are particularly extensive in the design phase of an extended Kalman filter, as frequent model changes normally occur.

One way to solve this problem is to have a program that makes code for the calculation of the Jacobian once the state space model has been described. Programs for symbolic differentiation (e.g. MACSYMA), or other programs (Rall (1981)) can be used for this purpose.

The Jacobian may also be calculated by numerical differentiation using the state space model.

Received 18 August 1988.

† Division of Engineering Cybernetics, The Norwegian Institute of Technology, N-7034 Trondheim, Norway.

This paper presents another method, where the estimate and the covariance information is updated simultaneously when the code for the process model is executed. This is achieved by a set of functions for a composite data type that carries both the estimated value and the associated covariance information.

The paper is organized as follows: First the data structure is presented. Then it is demonstrated how functions for this type of data are made. The way this applies to the extended Kalman filter is shown, and some details of a possible implementation (using Ada) are presented. These are followed by a brief example showing how a process model can be implemented. Some practical experience with the method is presented, and finally, a possible modification/extension to the functions is discussed.

2. A composite data structure

An estimate can be described by its value and covariance. Normally this is done using a vector for the estimated values and a matrix for the covariances. Here, another representation will be used: An estimate is represented as a composite variable where one part is the value, and the other part is a vector carrying covariance information. Such a variable will be denoted with a tilde, while the vector part will be denoted with an asterisk.

$$\tilde{a} = \{a, \overset{*}{\mathbf{a}}^{\mathrm{T}}\}$$

A set of such variables can be stacked to form a vector with composite components.

When estimating a vector $\mathbf{x} = (x_1\ x_2\ \dots\ x_n)^{\mathrm{T}}$, a vector of estimates $\tilde{\hat{\mathbf{x}}} = (\tilde{\hat{x}}_1\ \tilde{\hat{x}}_2\ \dots\ \tilde{\hat{x}}_n)^{\mathrm{T}}$ can be defined with components

$$\tilde{\hat{x}}_i = \{\hat{x}_i, \overset{*}{\hat{\mathbf{x}}}_i^{\mathrm{T}}\}$$

$\hat{x}_i$: estimated value of component i

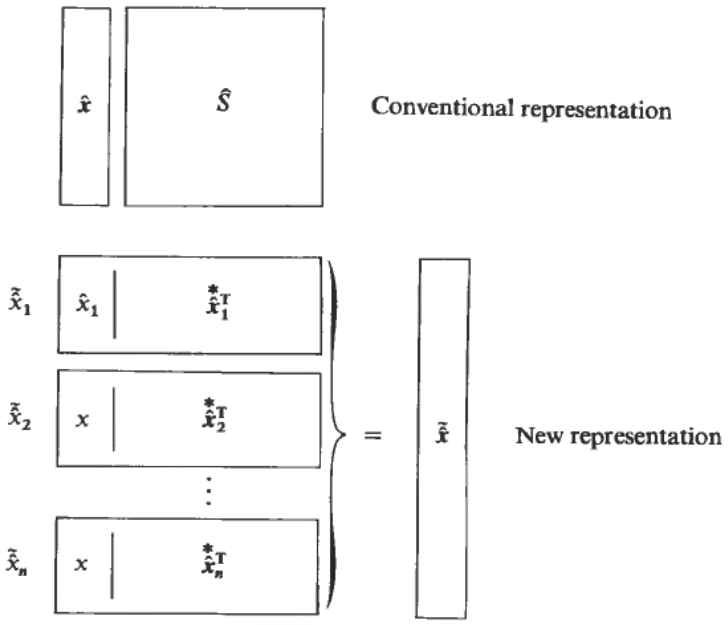

Figure 1. Representation of estimates and covariances.

with the vectors $\overset{*}{\hat{\mathbf{x}}}_i^T$ carrying covariance information satisfying

$$\overset{*}{\hat{\mathbf{x}}}_i^T \overset{*}{\hat{\mathbf{x}}}_j = E[(\hat{x}_i - x_i)(\hat{x}_j - x_j)]$$

This corresponds to a square root representation of the covariances

$$E[(\hat{\mathbf{x}} - \mathbf{x})(\hat{\mathbf{x}} - \mathbf{x})^T] = \hat{S}\hat{S}^T$$

where

$$\hat{S}^T = [\overset{*}{\hat{\mathbf{x}}}_1 \ \overset{*}{\hat{\mathbf{x}}}_2 \ \dots \ \overset{*}{\hat{\mathbf{x}}}_n]$$

The square root matrix $\hat{S}$ is not unique, but it is often preferred to keep it square and triangular. The conventional and new representation are shown in Fig. 1. Both representations contain the same information.

3. Functions for composite variables

A convenient means of using variables of the composite type is having a set of operators and functions for such variables.

From an original function $f(a, b, c \dots)$ the composite function $\tilde{f}$ is defined:

$$\tilde{f}(\tilde{a}, \tilde{b}, \tilde{c} \dots) = \left\{ f(a, b, c \dots), \frac{\partial f}{\partial a} \overset{*}{\mathbf{a}}^T + \frac{\partial f}{\partial b} \overset{*}{\mathbf{b}}^T + \frac{\partial f}{\partial c} \overset{*}{\mathbf{c}}^T + \dots \right\} \tag{1}$$

$$\tilde{a} = \{a, \overset{*}{\mathbf{a}}^T\} \quad \tilde{b} = \{b, \overset{*}{\mathbf{b}}^T\} \quad \tilde{c} = \{c, \overset{*}{\mathbf{c}}^T\} \quad \dots$$

Some examples:

$$\tilde{a} + \tilde{b} = \{a, \overset{*}{\mathbf{a}}^T\} + \{b, \overset{*}{\mathbf{b}}^T\} = \{a + b, \overset{*}{\mathbf{a}}^T + \overset{*}{\mathbf{b}}^T\}$$

$$\lambda\tilde{a} = \lambda\{a, \overset{*}{\mathbf{a}}^T\} = \{\lambda a, \lambda\overset{*}{\mathbf{a}}^T\}$$

$$\tilde{a} \cdot \tilde{b} = \{a, \overset{*}{\mathbf{a}}^T\} \cdot \{b, \overset{*}{\mathbf{b}}^T\} = \{a \cdot b, b \cdot \overset{*}{\mathbf{a}}^T + a \cdot \overset{*}{\mathbf{b}}^T\}$$

$$\tilde{\sin}(\tilde{a}) = \tilde{\sin}(\{a, \overset{*}{\mathbf{a}}^T\}) = \{\sin(a), \cos(a) \cdot \overset{*}{\mathbf{a}}^T\}$$

The two first examples can be combined to show that the composite data type has the properties of a vector field.

There is no problem with functions of functions

$$h(a) = f(g(a))$$

$$\tilde{h}(\tilde{a}) = \tilde{f}(\tilde{g}(\tilde{a})) = \tilde{f}\left(\left\{g(a), \frac{\partial g}{\partial a} \overset{*}{\mathbf{a}}^T\right\}\right)$$

$$= \left\{f(g(a)), \frac{\partial f}{\partial g}\frac{\partial g}{\partial a} \overset{*}{\mathbf{a}}^T\right\} = \left\{h(a), \frac{\partial h}{\partial a} \overset{*}{\mathbf{a}}^T\right\}$$

The chain rule is automatically applied.

The extension to vectors and vector functions is (for proof, see appendix):

Theorem 1:

From a function

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}) \dots f_m(\mathbf{x}))^T$$

a new function is defined

$$\tilde{\mathbf{f}}(\tilde{\mathbf{x}}) = (\tilde{f}_1(\tilde{\mathbf{x}}), \tilde{f}_2(\tilde{\mathbf{x}}) \dots \tilde{f}_m(\tilde{\mathbf{x}}))^{\mathrm{T}}$$

with components

$$\tilde{f}_i(\tilde{\mathbf{x}}) = \left\{ f_i(\mathbf{x}), \frac{\partial f_i}{\partial \mathbf{x}} [\overset{*}{\mathbf{x}}_1, \overset{*}{\mathbf{x}}_2 \dots \overset{*}{\mathbf{x}}_n]^{\mathrm{T}} \right\}$$

where each $\tilde{x}_i = \{x_i, \overset{*}{\mathbf{x}}_i\}$.

Then $\tilde{\mathbf{f}}(\tilde{\mathbf{x}})$ contains both $\mathbf{f}(\mathbf{x})$ and $\dfrac{\partial \mathbf{f}}{\partial \mathbf{x}} S$, where

$$[\overset{*}{\mathbf{x}}_1, \overset{*}{\mathbf{x}}_2 \dots \overset{*}{\mathbf{x}}_n]^{\mathrm{T}} = S$$

4. The extended Kalman filter

The state vector $\mathbf{x}$, the process noise $\mathbf{v}$, the measurements $\mathbf{y}$ and the measurement noise $\mathbf{w}$ are defined:

$$\mathbf{x} = (x_1, x_2 \dots x_n)^{\mathrm{T}} \qquad \mathbf{v} = (v_1, v_2 \dots v_r)^{\mathrm{T}}$$
$$\mathbf{y} = (y_1, y_2 \dots y_m)^{\mathrm{T}} \qquad \mathbf{w} = (w_1, w_2 \dots w_s)^{\mathrm{T}}$$

4.1. Time propagation

The state and covariance updates of the extended Kalman filter for time propagation is

$$\bar{\mathbf{x}}(k+1) = \mathbf{f}(\hat{\mathbf{x}}(k), \bar{\mathbf{v}}(k))$$
$$\bar{X}(k+1) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \hat{X}(k) \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \frac{\partial \mathbf{f}}{\partial \mathbf{v}} V(k) \left(\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right)^{\mathrm{T}}$$

with the partial derivatives taken at $\mathbf{x} = \hat{\mathbf{x}}(k)$ and $\mathbf{v} = \bar{\mathbf{v}}(k)$. $\bar{\mathbf{x}}$ and $\bar{X}$ are the *a priori* estimate and covariance, while $\hat{\mathbf{x}}$ and $\hat{X}$ are the *a posteriori* estimate and covariance. $\bar{\mathbf{v}}$ and V are the expected value and covariance of the process noise.

$\hat{\mathbf{x}}$ and $\bar{\mathbf{v}}$ can be stacked together into a vector $\mathbf{z}$ and their covariances into a matrix Z.

$$\mathbf{z} = \begin{bmatrix} \hat{\mathbf{x}} \\ \bar{\mathbf{v}} \end{bmatrix} \qquad Z = \begin{bmatrix} \hat{X} & 0 \\ 0 & V \end{bmatrix}$$

A square root matrix for Z can be formed

$$Z = S_z S_z^{\mathrm{T}} \qquad S_z = \begin{bmatrix} \hat{S} & 0 \\ 0 & S_v \end{bmatrix}$$

Then the equations can be rewritten

$$\bar{\mathbf{x}}(k+1) = \mathbf{f}(\mathbf{z}(k))$$
$$\bar{X}(k+1) = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} Z \left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right)^{\mathrm{T}} = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} S_z S_z^{\mathrm{T}} \left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right)^{\mathrm{T}} = \left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}} S_z\right)\left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}} S_z\right)^{\mathrm{T}}$$

Thus by defining

$$\tilde{\mathbf{z}} = (\tilde{z}_1, \tilde{z}_2 \dots \tilde{z}_{n+r})^{\mathrm{T}} = (\tilde{\bar{\mathbf{x}}}^{\mathrm{T}}, \tilde{\bar{\mathbf{v}}}^{\mathrm{T}})^{\mathrm{T}}$$

$$[\overset{*}{\tilde{z}}_1, \overset{*}{\tilde{z}}_2 \dots \overset{*}{\tilde{z}}_{n+r}]^{\mathrm{T}} = S_z$$

$$\tilde{z}_i = \{z_i, \mathbf{z}_i^{\mathrm{T}}\}$$

$\tilde{\mathbf{f}}(\tilde{\mathbf{z}})$ will contain $\mathbf{f}(\mathbf{z})$ and $\partial\mathbf{f}/\partial\mathbf{z}\ S_z$ from Theorem 1.

Since $\partial\mathbf{f}/\partial\mathbf{z}\ S_z$ is rectangular, modified Gram–Schmidt orthogonalization or Householder transformations are applied to obtain

$$\bar{X}(k+1) = \bar{S}\bar{S}^{\mathrm{T}}$$

where $\bar{S}$ is square and triangular.

The matrices $\bar{X}$, $\hat{X}$, V and Z are not used by the algorithm, they are only included here to show that the square root formulation is equivalent to the original one. The square root formulation possesses superior numerical properties.

4.2. Measurement update

The calculation of the predicted measurements can be done in a similar fashion: The equation is:

$$\bar{\mathbf{y}} = \mathbf{g}(\bar{\mathbf{x}}, \bar{\mathbf{w}})$$

$\bar{\mathbf{x}}$ and $\bar{\mathbf{w}}$ are stacked into a vector $\mathbf{q}$ and the covariances $\bar{X}$ and W into a block diagonal matrix Q (similar to $\mathbf{z}$ and Z in the previous section). Then a square root matrix S_q is formed with the two blocks S and S_w.

$$\mathbf{q} = \begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{w}} \end{bmatrix} \qquad Q = \begin{bmatrix} \bar{X} & 0 \\ 0 & W \end{bmatrix} \qquad S_q = \begin{bmatrix} \bar{S} & 0 \\ 0 & S_w \end{bmatrix}$$

Next the composite vector $\tilde{\mathbf{q}}$ is defined

$$\tilde{\mathbf{q}} = (\tilde{q}_1, \tilde{q}_2 \dots \tilde{q}_{n+s})^{\mathrm{T}} = (\tilde{\bar{\mathbf{x}}}^{\mathrm{T}}, \tilde{\bar{\mathbf{w}}}^{\mathrm{T}})^{\mathrm{T}}$$

$$[\overset{*}{\tilde{q}}_1, \overset{*}{\tilde{q}}_2 \dots \overset{*}{\tilde{q}}_{n+s}]^{\mathrm{T}} = S_q$$

$$\tilde{q}_i = \{q_i, \overset{*}{\tilde{q}}_i\}$$

The Kalman gain is

$$\begin{aligned} K &= \bar{X}\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\right)^{\mathrm{T}}\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\,\bar{X}\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\right)^{\mathrm{T}} + \frac{\partial\mathbf{g}}{\partial\mathbf{w}}\,W\left(\frac{\partial\mathbf{g}}{\partial\mathbf{w}}\right)^{\mathrm{T}}\right)^{-1} \\ &= \bar{S}\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\,\bar{S}\right)^{\mathrm{T}}\left(\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\,\bar{S}\right)\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\,\bar{S}\right)^{\mathrm{T}} + \left(\frac{\partial\mathbf{g}}{\partial\mathbf{w}}\,S_w\right)\left(\frac{\partial\mathbf{g}}{\partial\mathbf{w}}\,S_w\right)^{\mathrm{T}}\right)^{-1} \\ &= \bar{S}\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\,\bar{S}\right)^{\mathrm{T}}\left(\left(\frac{\partial\mathbf{g}}{\partial\mathbf{q}}\,S_q\right)\left(\frac{\partial\mathbf{g}}{\partial\mathbf{q}}\,S_q\right)^{\mathrm{T}}\right)^{-1} \end{aligned}$$

The update of the estimate is

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + K(\mathbf{y} - \bar{\mathbf{y}})$$

and the covariance update is

$$\hat{X} = \bar{X} - K\,\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\,\bar{X} = \bar{S}\bar{S}^{\mathrm{T}} - K\,\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\,\bar{S}\bar{S}^{\mathrm{T}}$$

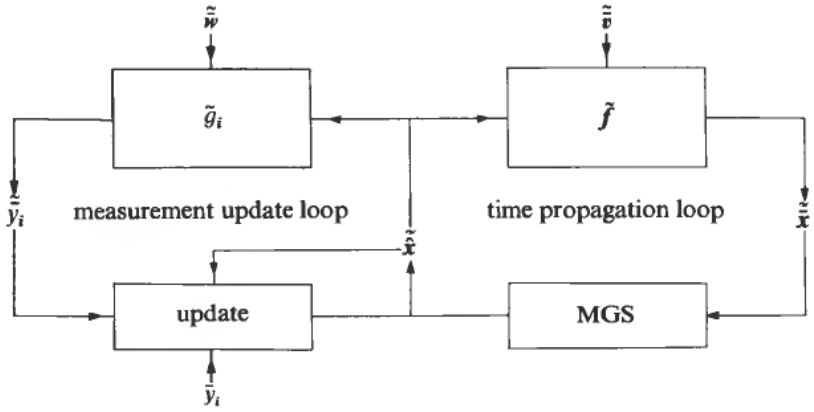

Figure 2. Information flow of the extended Kalman filter.

Evaluating

$$\tilde{\bar{\mathbf{y}}} = \tilde{\mathbf{g}}(\tilde{\mathbf{q}})$$

gives $\bar{\mathbf{y}}$ and $(\partial \mathbf{g}/\partial \mathbf{q})S_q$ from Theorem 1

$$\frac{\partial \mathbf{g}}{\partial \mathbf{q}} S_q = \left[\frac{\partial \mathbf{g}}{\partial \mathbf{x}} \bar{S}, \frac{\partial \mathbf{g}}{\partial \mathbf{w}} S_w\right]$$

Thus all the information needed for the update is available after this evaluation.

Quite often $(\partial \mathbf{g}/\partial \mathbf{w})W(\partial \mathbf{g}/\partial \mathbf{w})^{\mathrm{T}}$ is (assumed to be) diagonal. In this case it is possible to perform the update by processing one measurement after the other. The *a posteriori* estimate after the update using one measurement is the *a priori* estimate when the next measurement is used. Matrix inversion is then avoided, and it is possible to use a very robust algorithm for the update (Carlson 1973, Bierman 1977).

This algorithm requires $\bar{S}_i$, $(\partial g_i/\partial \mathbf{x})\bar{S}_i$ and $(\partial g_i/\partial \mathbf{w})S_w$ for the calculation of K_i and the *a posteriori* covariance $(\hat{S}_i)$. $\hat{\mathbf{x}}_i$ is calculated from $\bar{\mathbf{x}}_i$, $\bar{y}_i$, y_i and K_i. (The index i is for component i of $\mathbf{y}$.) Then $\tilde{\hat{\mathbf{x}}}_{i+1} = \tilde{\hat{\mathbf{x}}}_i$ until all measurements have been processed.

4.3. Summing up

A top view of the algorithm is shown in Fig. 2. The right loop is used for time propagation, while the left loop is passed once for each measurement y_i at the corresponding instant.

Here $\tilde{\bar{\mathbf{x}}}$ is used for the intermediate result in the time propagation loop, while $\tilde{\hat{\mathbf{x}}}$ is used for both *a priori* and *a posteriori* estimates in the measurement update loop.

The block 'update' represents Carlson and Bierman's algorithm together with the update of the estimate. The block 'MGS' contains the modified Gram–Schmidt orthogonalization making the covariance square root matrix triangular. The covariance information is contained within the composite variables $\hat{\mathbf{x}}$, $\tilde{\bar{\mathbf{x}}}$, and $\tilde{\bar{y}}_i$.

5. Implementation

It should be easy for a user to specify the process equations. Thus normal operators and functions should be used in the specification. To obtain this, a programming language that permits the overloading of operators and functions will be used.

Overloading means that many functions can have the same name as long as they can be distinguished from each other by the context in which they are used (e.g. the argument type). Since Ada permits overloading, it will be used here. (Another possibility is defining a special language and a precompiler producing code in a common programming language such as FORTRAN or Pascal.)

In the following a type VECTOR that can be used to store vectors is assumed to be available, together with operators for vector addition and multiplication with a scalar.

The composite data type ESTIMATE is then declared:

```
type ESTIMATE is
  record
    EST : FLOAT;        -- estimated value
    COV : VECTOR;       -- covariance information
  end record;
```

The most common operators and functions are then implemented using the definition in eqn (1). Some examples follow:

```
function "+" (A, B : ESTIMATE) return ESTIMATE is
  begin
    return ( A.EST + B.EST , A.COV + B.COV );
  end "+";

function "*" (A, B : ESTIMATE) return ESTIMATE is
  begin
    return ( A.EST*B.EST , A.EST*B.COV + B.EST*A.COV );
  end "*";

function SIN (A : ESTIMATE) return ESTIMATE is
  begin
    return ( SIN(A.EST) , COS(A.EST)*A.COV );
  end SIN;
```

A package containing these functions should be available to the normal user. If a more complex function is required, it can usually be put together from these simple functions. For instance, a function SINAB which calculates sin($a*b$) will be

```
function SINAB (A, B : ESTIMATE) return ESTIMATE is
  begin
    return SIN(A*B);
  end SINAB;
```

This function will call the previously defined functions "*" and SIN. This enables the user to implement functions $\tilde{\mathbf{f}}$ and $\tilde{\mathbf{g}}$.

The procedures for updating and triangularization should also exist as a software package together with utilities for initialization etc.

6. A process example

The example is a second order system where two states and two parameters are to be estimated. Both states are measured, but there are noise on the measurements.

The system is described by the equations:

$$x_1(k+1) = \theta_1(k) \cdot x_1(k) + v(k)$$

$$x_2(k+1) = \theta_2(k) \cdot x_2(k) + (x_1(k) + x_1(k+1))/2$$

$$\theta_1(k+1) = \theta_1(k)$$

$$\theta_2(k+1) = \theta_2(k)$$

$$y_1(k) = x_1(k) + w_1(k)$$

$$y_2(k) = x_2(k) + w_2(k)$$

The vectors can be defined:

$$\mathbf{x} = [x_1, x_2, \theta_1, \theta_2]^T$$

$$\mathbf{v} = [v]^T$$

$$\mathbf{y} = [y_1, y_2]^T$$

$$\mathbf{w} = [w_1, w_2]^T$$

To simplify the implementation, a new data type is declared:

```
type STATE is
  record
    THIS : ESTIMATE;      -- x(k)
    NEXT : ESTIMATE;      -- x(k+1)
  end record;
```

Then the variables are declared

```
X1,X2,TH1,TH2   :STATE;
V               :ESTIMATE;
W1, W2          :ESTIMATE;
Y               :ESTIMATE;
```

The procedure MODEL, implementing $\tilde{\mathbf{f}}$, will be:

```
procedure MODEL is

procedure ORDER1 ( X : in out STATE; TIMECONST, INPUT : in ESTIMATE)
is
  begin
      X.NEXT := TIMECONST * X.THIS + INPUT; -- first order process
  end ORDER1;

procedure PARAMETER ( P : in out STATE) is
  begin
      P.NEXT := P.THIS;            -- a parameter is constant
  end PARAMETER;

begin
  ORDER1 ( X1, TH1, V);                              -- X1(k+1)
  ORDER1 ( X2, TH2, (X1.THIS + X1.NEXT)/2 );         -- X2(k+1)
  PARAMETER ( TH1 );                                 -- TH1(k+1)
  PARAMETER ( TH2 );                                 -- TH2(k+1)
end MODEL;
```

Modularity has been extensively used for the implementation of the process equations.

The procedure MEASURE, implementing the functions $\tilde{g}_i$ will be:

```
procedure MEASURE (I : in INTEGER) is
  begin
    case I is
      when 1    => Y := X1.THIS + W1;
      when 2    => Y := X2.THIS + W2;
  end case;
  end MEASURE;
```

This completes the model specification. The estimates and covariances should be initialized, and the declared variables put into vectors for use by the measurement update and factorization algorithms. This is achieved by using utilities which are available as a part of the software supporting this estimation method.

7. Practical experience

The method described has been used for the estimation of states and parameters in a crushing and screening circuit (Solberg 1988). The possibility of making a modular implementation of the process model was utilized to a large extent. The original Kalman filter equations were used instead of a square root formulation, which in fact made the programs more complex. Pascal was adopted for the implementation, however since overloading could not be applied, procedure calls were used for all operators and functions.

The modularity and the fact that no partial differentiation was necessary made it very easy to modify the model. There was no problem of inconsistency between the estimate and covariance updates. It was also possible to simulate the process using the implemented model directly. This made it quite easy to find errors in the model, and make a set of simulated measurements that could be used for testing the convergence of the estimator.

8. A possible extension

The extended Kalman filter is based on the approximations:

$$E(\mathbf{f}(\mathbf{x})) \approx \mathbf{f}(E(\mathbf{x}))$$

$$E((\mathbf{f}(\hat{\mathbf{x}}) - \mathbf{f}(\mathbf{x}))(\mathbf{f}(\hat{\mathbf{x}}) - \mathbf{f}(\mathbf{x}))^{\mathrm{T}}) \approx \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}} E((\hat{\mathbf{x}} - \mathbf{x})(\hat{\mathbf{x}} - \mathbf{x})^{\mathrm{T}}) \left(\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}}\right)^{\mathrm{T}} \tag{2}$$

These approximations are sometimes rather unsatisfactory. A simple example is for $f(a, b) = a \cdot b$ where a and b are close to 0. In this case the partial derivatives are both close to 0 so the calculated covariance of the product will almost vanish.

Using the presented method the arguments to a function will also carry covariance information. This makes it possible to detect cases where the approximations do not hold. If so, a warning message can be given or some remedial action can be taken.

If the distribution of the arguments is known, a function could utilize this information to achieve a better result. Assuming a multinormal distribution of the arguments, the following can be used for the product (Solberg 1988):

$$\tilde{a} \cdot \tilde{b} = \{\hat{a} \cdot \hat{b} + \overset{*}{\mathbf{a}}^{\mathrm{T}}\overset{*}{\mathbf{b}},\ \hat{a}\overset{*}{\mathbf{b}}^{\mathrm{T}} + \hat{b}\overset{*}{\mathbf{a}}^{\mathrm{T}} + (\overset{*}{\mathbf{a}}^{\mathrm{T}}\overset{*}{\mathbf{a}}\overset{*}{\mathbf{b}}^{\mathrm{T}}\overset{*}{\mathbf{b}} + (\overset{*}{\mathbf{a}}^{\mathrm{T}}\overset{*}{\mathbf{b}})^2)^{1/2}\mathbf{e}_v\}$$

where $\mathbf{e}_v$ is the unit vector associated with an extra noise variable. This gives correct values for the estimate and covariance, but the product will not have a normal probability distribution. The effect of such a modification to the extended Kalman filter has not been thoroughly investigated.

9. Computational requirements

The covariance update represents most of the computational burden when using an extended Kalman filter. This method utilizes the sparsity of the matrices of partial derivatives, and can thus be faster than methods based on straightforward matrix multiplication. When a square root method is applied, the factorizations involved will often be the bottleneck of the filter. This means that the way the model is implemented becomes a matter of minor importance.

As modularity is implemented on a very low level (basic operators and functions), the calculations can be freely distributed to more processors. The method is well suited to parallel processing.

10. Conclusion

The presented method solves two problems involved with the implementation of the extended Kalman filter:

The need to specify partial derivatives.

The problem of achieving consistency between the process model and the partial derivatives (due to error-prone differentiation and coding).

The properties of the filter (convergence, numerical properties, etc.) remain unchanged.

With the filtering, the factorization, the low level procedures (basic operators and functions) and the set of initialization procedures available, the implementation of a process model is very simple. The task has about the same complexity as the implementation of a discrete simulation model.

The method makes it very easy to make a modular implementation of a process model. The different modules can even be executed on different processors, thus making parallel processing feasible.

Appendix

Proof of Theorem 1.

A function $f(\mathbf{x})$ and a matrix S is given, such that $\mathbf{x}$ is an n-dimensional vector, and S has n rows. The matrix S can be split into row vectors:

$$S = [\overset{*}{\mathbf{x}}_1, \overset{*}{\mathbf{x}}_2 \ldots \overset{*}{\mathbf{x}}_n]^{\mathrm{T}}$$

A vector $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x}_2 \dots \tilde{x}_n)^T$ is defined with components

$$\tilde{x}_i = \{x_i, \overset{*}{\mathbf{x}}_i^T\}$$

A function $\tilde{f}(\tilde{\mathbf{x}})$ based on $f(\mathbf{x})$ is defined to be

$$\tilde{f}(\mathbf{x}) = \left\{f(\mathbf{x}), \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \overset{*}{\mathbf{x}}_i^T\right\} = \left\{f(\mathbf{x}), \frac{\partial f}{\partial \mathbf{x}} S\right\}$$

This is consistent with the definition (1), the only difference is that the arguments have been put into a vector.

For a vector function $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}) \dots f_m(\mathbf{x}))^T$ the extension is quite simple:

$$\tilde{\mathbf{f}}(\tilde{\mathbf{x}}) = (\tilde{f}_1(\tilde{\mathbf{x}}), \tilde{f}_2(\mathbf{x}) \dots \tilde{f}_m(\tilde{\mathbf{x}}))^T$$

where

$$\tilde{f}_i(\tilde{\mathbf{x}}) = \left\{f_i(\mathbf{x}), \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \overset{*}{\mathbf{x}}_j^T\right\} = \left\{f_i(\mathbf{x}), \frac{\partial f_i}{\partial \mathbf{x}} S\right\}$$

When stacking the values and covariance components, this yields

$$\begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} = \mathbf{f}(\mathbf{x}) \qquad \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}} S \\ \frac{\partial f_2}{\partial \mathbf{x}} S \\ \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}} S \end{bmatrix} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} S$$

which proves the theorem.

For a function $\mathbf{f}(\mathbf{g}(\mathbf{x}))$, the two composite functions $\tilde{\mathbf{f}}$ and $\tilde{\mathbf{g}}$ are defined. If $\tilde{\mathbf{x}}$ contains $\mathbf{x}$ and S, $\tilde{\mathbf{g}}(\mathbf{x})$ gives $\mathbf{g}(\mathbf{x})$ and $(\partial \mathbf{g}/\partial \mathbf{x})S$. $\tilde{f}(\tilde{\mathbf{g}}(\tilde{\mathbf{x}}))$ then gives $\mathbf{f}(\mathbf{g}(\mathbf{x}))$ and $(\partial \mathbf{f}/\partial \mathbf{g})((\partial \mathbf{g}/\partial \mathbf{x})S)$, which shows that the chain rule is automatically satisfied.

References

Bierman, G. J. (1977). *Factorization Methods for Discrete Sequential Estimation* (Academic Press, New York).

Carlson, N. A. (1973). Fast triangular formulation of the square root filter. *AIAA Journal*, **11**, 1259–1265.

MACSYMA. Symbolics Inc., Four Cambridge Center (Cambridge, MA 02143).

Rall, L. B. (1981). *Automatic Differentiation: Techniques and Applications* (Springer Verlag, Berlin).

Solberg, I. (1988). A modular implementation of the extended Kalman filter with application to a crushing and screening circuit. Dr. Ing. thesis, Norwegian Institute of Technology, Division of Engineering Cybernetics, Trondheim.