# Optimal filtering with Kalman filters and smoothers – a Manual for Matlab toolbox EKF/UKF

Jouni Hartikainen and Simo Särkkä
Laboratory of Computational Engineering,
Helsinki University of Technology,
P.O.Box 9203, FIN-02015 TKK, Espoo, Finland
*jmjharti@cc.hut.fi, simo.sarkka@hut.fi*

September 4, 2007

Version 1.0

**Abstract**

In this paper we present a documentation for optimal filtering toolbox for mathematical software package Matlab. The methods in the toolbox include Kalman filter, extended Kalman filter and unscented Kalman filter for discrete time state space models. Also included in the toolbox are the Rauch-Tung-Striebel and Forward-Backward smoother counter-parts for each filter, which can be used to smooth the previous state estimates, after obtaining new measurements. The usage and function of each method are illustrated with five demonstrations problems.

# Contents

# Preface

Most of the software provided with this toolbox are originally created by Simo Särkkä while he was doing research on his doctoral thesis (Särkkä, 2006b) in the Laboratory of Computational Engineering (LCE) at Helsinki University of Technology (HUT). This document has been written by Jouni Hartikainen at LCE during spring 2007 with a little help from Simo Särkkä. Jouni also checked and commented the software code thoroughly. Many (small) bugs were fixed, and also some new functions were implemented (for example 2nd order EKF and augmented form UKF). Jouni also provided the software code for first three demonstrations, modified the two last ones a bit, and ran all the simulations.

First author would like to thank Doc. Aki Vehtari for helpful comments during the work and for coming up with the idea of this toolbox in the first place. Prof. Jouko Lampinen also deserves thanks for ultimately making this work possible.

# 1 Introduction

The term optimal filtering refers to methodology used for estimating the *state* of a time varying system, from which we observe indirect noisy measurements. The state refers to the physical state, which can be described by dynamic variables, such as position, velocity and acceleration of a moving object. The noise in the measurements means that there is a certain degree of uncertainty in them. The dynamic system evolves as a function of time, and there is also noise in the dynamics of system, *process noise*, meaning that the dynamic system cannot be modelled entirely deterministically. In this context, the term filtering basically means the process of filtering out the noise in the measurements and providing an optimal estimate for the state given the observed measurements and the assumptions made about the dynamic system.

This toolbox provides basic tools for estimating the state of a linear dynamic system, the Kalman filter, and also two extensions for it, the extended Kalman filter (EKF) and unscented Kalman filter (UKF), both of which can be used for estimating the states of nonlinear dynamic systems. Also the smoother counterparts of the filters are provided. Smoothing in this context means giving an estimate of the state of the system on some time step given all the measurements including ones encountered after that particular time step, in other words, the smoother gives a smoothed estimate for the history of the system's evolved state given all the measurements obtained so far.

This documentation is organized as follows:

- First we briefly introduce the concept of discrete-time state space models. After that we consider linear, discrete-time state space models in more detail and review Kalman filter, which is the basic method for recursively solving the linear state space estimation problems. Also Kalman smoother is introduced. After that the function of Kalman filter and smoother and their usage in this toolbox in demonstrated with one example (CWPA-model).
- Next we move from linear to nonlinear state space models and review the extended Kalman filter (and smoother), which is the classical extension to Kalman filter for nonlinear estimation. The usage of EKF in this toolbox is illustrated exclusively with one example (Tracking a random sine signal), which also compares the performances of EKF, UKF and their smoother counter-parts.
- After EKF we review unscented Kalman filter (and smoother), which is a newer extension to traditional Kalman filter to cover nonlinear filtering problems. The usage of UKF is illustrated with one example (UNGM-model), which also demonstrates the differences between different nonlinear filtering techniques.
- To give a more thorough demonstration to the provided methods two more classical nonlinear filtering examples are provided (Bearings Only Tracking and Reentry Vehicle Tracking).
- Lastly we list and describe briefly all the functions included in the toolbox.

The mathematical notation used in this document follows the notation used in (Särkkä, 2006b).

# 2 Discrete-Time State Space Models

In this section we shall consider models where the states are defined in discrete-time models. The models are defined recursively in terms of distributions

$$\begin{aligned} \mathbf{x}_k &\sim p(\mathbf{x}_k \,|\, \mathbf{x}_{k-1}) \\ \mathbf{y}_k &\sim p(\mathbf{y}_k \,|\, \mathbf{x}_k), \end{aligned} \tag{1}$$

where

- $\mathbf{x}_k \in \mathbb{R}^n$ is the *state* of the system on the time step $k$.
- $\mathbf{y}_k \in \mathbb{R}^m$ is the measurement on the time step $k$.
- $p(\mathbf{x}_k \,|\, \mathbf{x}_{k-1})$ is the dynamic model which charaterizes the dynamic behaviour of the system. Usually the model is a probability density (continous state), but it can also be a counting measure (discrete state), or a combination of them, if the state is both continuous and discrete.
- $p(\mathbf{y}_k \,|\, \mathbf{x}_k)$ is the model for measurements, which describes how the measurements are distributed given the state. This model characterizes how the dynamic model is perceived by the observers.

A system defined this way has the so called *Markov*-property, which means that the state $\mathbf{x}_k$ given $\mathbf{x}_{k-1}$ is independent from the history of states and measurements, which can also be expressed with the following equality:

$$p(\mathbf{x}_k \,|\, \mathbf{x}_{1:k-1}, \mathbf{y}_{1:k-1}) = p(\mathbf{x}_k \,|\, \mathbf{x}_{k-1}). \tag{2}$$

The past doesn't depend on the future given the present, which is the same as

$$p(\mathbf{x}_{k-1} \,|\, \mathbf{x}_{k:T}, \mathbf{y}_{k:T}) = p(\mathbf{x}_{k-1} \,|\, \mathbf{x}_k). \tag{3}$$

The same applies also to measurements meaning that the measurement $\mathbf{y}_k$ is independent from the histories of measurements and states, which can be expressed with equality

$$p(\mathbf{y}_k \,|\, \mathbf{x}_{1:k}, \mathbf{y}_{1:k-1}) = p(\mathbf{y}_k \,|\, \mathbf{x}_k). \tag{4}$$

In actual application problems, we are interested in predicting and estimating dynamic system's state given the measurements obtained so far. In probabilistic terms, we are interested in the predictive distribution for the state at the next time step

$$p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k-1}), \tag{5}$$

and in the marginal posterior distribution for the state at the current time step

$$p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k}). \tag{6}$$

The formal solutions for these distribution are given by the following recursive Bayesian filtering equations (e.g. Särkkä, 2006b):

$$p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k-1}) = \int p(\mathbf{x}_k \,|\, \mathbf{x}_{k-1})\, p(\mathbf{x}_{k-1} \,|\, \mathbf{y}_{1:k-1})\, \mathrm{d}\mathbf{x}_{k-1}. \tag{7}$$

and

$$p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k}) = \frac{1}{Z_k} p(\mathbf{y}_k \,|\, \mathbf{x}_k)\, p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k-1}), \tag{8}$$

where the normalization constant $Z_k$ is given as

$$Z_k = \int p(\mathbf{y}_k \,|\, \mathbf{x}_k)\, p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k-1})\, \mathrm{d}\mathbf{x}_k. \tag{9}$$

In many cases we are also interested in smoothed state estimates of previous time steps given the measurements obtained so far. In other words, we are interested in the marginal posterior distribution

$$p(\mathbf{x}_k \,|\, \mathbf{y}_{1:T}), \tag{10}$$

where $T > k$. As with the filtering equations above also in this case we can express the formal solution as a set of recursive Bayesian equations (e.g. Särkkä, 2006b):

$$\begin{aligned} p(\mathbf{x}_{k+1} \,|\, \mathbf{y}_{1:k}) &= \int p(\mathbf{x}_{k+1} \,|\, \mathbf{x}_k)\, p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k})\, \mathrm{d}\mathbf{x}_k \\ p(\mathbf{x}_k \,|\, \mathbf{y}_{1:T}) &= p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k}) \int \left[ \frac{p(\mathbf{x}_{k+1} \,|\, \mathbf{x}_k)\, p(\mathbf{x}_{k+1} \,|\, \mathbf{y}_{1:T})}{p(\mathbf{x}_{k+1} \,|\, \mathbf{y}_{1:k})} \right] \mathrm{d}\mathbf{x}_{k+1}. \end{aligned} \tag{11}$$

## 2.1 Linear state space estimation

The simplest of the state space models considered in this documentation are linear models, which can be expressed with equations of the following form:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}_{k-1}\,\mathbf{x}_{k-1} + \mathbf{q}_{k-1} \\ \mathbf{y}_k &= \mathbf{H}_k\,\mathbf{x}_k + \mathbf{r}_k, \end{aligned} \tag{12}$$

where

- $\mathbf{x}_k \in \mathbb{R}^n$ is the *state* of the system on the time step $k$.
- $\mathbf{y}_k \in \mathbb{R}^m$ is the measurement on the time step $k$.
- $\mathbf{q}_{k-1} \sim \mathrm{N}(\mathbf{0}, \mathbf{Q}_{k-1})$ is the process noise on the time step $k-1$.
- $\mathbf{r}_k \sim \mathrm{N}(\mathbf{0}, \mathbf{R}_k)$ is the measurement noise on the time step $k$.
- $\mathbf{A}_{k-1}$ is the transition matrix of the dynamic model.
- $\mathbf{H}_k$ is the measurement model matrix.
- The prior distribution for the state is $\mathbf{x}_0 \sim \mathrm{N}(\mathbf{m}_0, \mathbf{P}_0)$, where parameters $\mathbf{m}_0$ and $\mathbf{P}_0$ are set using the information known about the system under the study.

The model can also be equivalently expressed in probabilistic terms with distributions

$$\begin{aligned} p(\mathbf{x}_k \,|\, \mathbf{x}_{k-1}) &= \mathrm{N}(\mathbf{x}_k \,|\, \mathbf{A}_{k-1}\,\mathbf{x}_{k-1}, \mathbf{Q}_{k-1}) \\ p(\mathbf{y}_k \,|\, \mathbf{x}_k) &= \mathrm{N}(\mathbf{y}_k \,|\, \mathbf{H}_k\,\mathbf{x}_k, \mathbf{R}_k). \end{aligned} \tag{13}$$

### 2.1.1 Discretization of continuous-time linear time-invariant systems

Often many linear time-invariant models are described with continous-time state equations of the following form:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{F}\mathbf{x}(t) + \mathbf{L}\mathbf{w}(t), \tag{14}$$

where

- the initial conditions are $\mathbf{x}(0) \sim \mathrm{N}(\mathbf{m}(0), \mathbf{P}(0))$,
- $\mathbf{F}$ and $\mathbf{L}$ are constant matrices, which characterize the behaviour of the model,
- $\mathbf{w}(t)$ is a white noise process with a power spectral density $\mathbf{Q}_c$.

To be able to use the Kalman filter defined in the next section the model (14) must be discretized somehow, so that it can be described with a model of the form (12). The solution for the discretized matrices $\mathbf{A}_k$ and $\mathbf{Q}_k$ can be given as (e.g. Särkkä, 2006b; Bar-Shalom et al, 2001)

$$\mathbf{A}_k = \exp(\mathbf{F}\,\Delta t_k) \tag{15}$$

$$\mathbf{Q}_k = \int_0^{\Delta t_k} \exp(\mathbf{F}\,(\Delta t_k - \tau))\,\mathbf{L}\,\mathbf{Q}_c\,\mathbf{L}^T\,\exp(\mathbf{F}\,(\Delta t_k - \tau))^T \mathrm{d}\tau, \tag{16}$$

where $\Delta t_k = t_{k+1} - t_k$ is the stepsize of the discretization. In some cases the $\mathbf{Q}_k$ can be calculated analytically, but in cases where it isn't possible, the matrix can still be calculated efficiently using the following matrix fraction decomposition:

$$\begin{pmatrix} \mathbf{C}_k \\ \mathbf{D}_k \end{pmatrix} = \exp\left\{ \begin{pmatrix} \mathbf{F} & \mathbf{L}\,\mathbf{Q}_c\,\mathbf{L}^T \\ \mathbf{0} & -\mathbf{F}^T \end{pmatrix} \Delta t_k \right\} \begin{pmatrix} \mathbf{0} \\ \mathbf{I} \end{pmatrix}. \tag{17}$$

The matrix $\mathbf{Q}_k$ is then given as $\mathbf{Q}_k = \mathbf{C}_k \mathbf{D}_k^{-1}$.

In this toolbox the discretization can be done with the function `lti_disc` (see page 44), which uses the matrix fractional decomposition.

### 2.1.2 Kalman filter

The classical Kalman filter was first introduced by Rudolph E. Kalman in his seminal paper (Kalman, 1960). The purpose of the discrete-time Kalman filter is to provide the closed form recursive solution for estimation of linear discrete-time dynamic systems, which can be described by equations of the form (12).

Kalman filter has two steps: the prediction step, where the next state of the system is predicted given the previous measurements, and the update step, where the current state of the system is estimated given the measurement at that time step. The steps translate to equations as follows (see, e.g., Särkkä, 2006b, Bar-Shalom et al.,2001, for derivation):

- *Prediction:*

$$\begin{aligned} \mathbf{m}_k^- &= \mathbf{A}_{k-1}\, \mathbf{m}_{k-1} \\ \mathbf{P}_k^- &= \mathbf{A}_{k-1}\, \mathbf{P}_{k-1}\, \mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1}. \end{aligned} \tag{18}$$

- *Update:*

$$\begin{aligned} \mathbf{v}_k &= \mathbf{y}_k - \mathbf{H}_k\, \mathbf{m}_k^- \\ \mathbf{S}_k &= \mathbf{H}_k\, \mathbf{P}_k^-\, \mathbf{H}_k^T + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_k^-\, \mathbf{H}_k^T\, \mathbf{S}_k^{-1} \\ \mathbf{m}_k &= \mathbf{m}_k^- + \mathbf{K}_k\, \mathbf{v}_k \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\, \mathbf{S}_k\, \mathbf{K}_k^T, \end{aligned} \tag{19}$$

where

- $\mathbf{m}_k^-$ and $\mathbf{P}_k^-$ are the predicted mean and covariance of the state, respectively, on the time step $k$ before seeing the measurement.
- $\mathbf{m}_k$ and $\mathbf{P}_k$ are the estimated mean and covariance of the state, respectively, on time step $k$ after seeing the measurement.
- $\mathbf{v}_k$ is the innovation or the measurement residual on time step $k$.
- $\mathbf{S}_k$ is the measurement prediction covariance on the time step $k$.
- $\mathbf{K}_k$ is the filter gain, which tells how much the predictions should be corrected on time step $k$.

Note that in this case the predicted and estimated state covariances on different time steps do not depend on any measurements, so that they could be calculated off-line before making any measurements provided that the matrices $\mathbf{A}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{H}$ are known on those particular time steps. Usage for this property, however, is not currently provided explicitly with this toolbox.

It is also possible to predict the state of system as many steps ahead as wanted just by looping the predict step of Kalman filter, but naturally the accuracy of the estimate decreases with every step.

The prediction and update steps can be calculated with functions `kf_predict` and `kf_update`, described on pages 44 and 44, respectively.

### 2.1.3 Kalman smoother

The discrete-time Kalman smoother, also known as the Rauch-Tung-Striebel-smoother (RTS), (Rauch et al., 1965; Gelb, 1974; Bar-Shalom et al., 2001) can be used for computing the smoothing solution for the model (12) given as distribution

$$p(\mathbf{x}_k \mid \mathbf{y}_{1:T}) = \mathrm{N}(\mathbf{x}_k \mid \mathbf{m}_k^s, \mathbf{P}_k^s). \tag{20}$$

The mean and covariance $\mathbf{m}_k^s$ and $\mathbf{P}_k^s$ are calculated with the following equations:

$$\begin{aligned}
\mathbf{m}_{k+1}^- &= \mathbf{A}_k\, \mathbf{m}_k \\
\mathbf{P}_{k+1}^- &= \mathbf{A}_k\, \mathbf{P}_k\, \mathbf{A}_k^T + \mathbf{Q}_k \\
\mathbf{C}_k &= \mathbf{P}_k\, \mathbf{A}_k^T\, [\mathbf{P}_{k+1}^-]^{-1} \\
\mathbf{m}_k^s &= \mathbf{m}_k + \mathbf{C}_k\, [\mathbf{m}_{k+1}^s - \mathbf{m}_{k+1}^-] \\
\mathbf{P}_k^s &= \mathbf{P}_k + \mathbf{C}_k\, [\mathbf{P}_{k+1}^s - \mathbf{P}_{k+1}^-]\, \mathbf{C}_k^T,
\end{aligned} \tag{21}$$

where

- $\mathbf{m}_k^s$ and $\mathbf{P}_k^s$ are the smoother estimates for the state mean and state covariance on time step $k$.
- $\mathbf{m}_k$ and $\mathbf{P}_k$ are the filter estimates for the state mean and state covariance on time step $k$.
- $\mathbf{m}_{k+1}^-$ and $\mathbf{P}_{k+1}^-$ are the predicted state mean and state covariance on time step $k+1$, which are the same as in the Kalman filter.
- $\mathbf{C}_k$ is the smoother gain on time step $k$, which tells how much the smooted estimates should be corrected on that particular time step.

The difference between Kalman filter and Kalman smoother is that the recursion in filter moves forward and in smoother backward, as can be seen from the equations above. In smoother the recursion starts from the last time step T with $\mathbf{m}_T^s = \mathbf{m}_T$ and $\mathbf{P}_T^s = \mathbf{P}_T$.

The smoothed estimate for states and covariances using the RTS smoother can be calculated with the function `rts_smooth`, which is described on page 45.

In addition to RTS smoother it is possible to formulate the smoothing operation as a combination of two optimum filters (Fraser and Potter, 1969), of which the first filter sweeps the data forward going from the first measurement towards the last one, and the second sweeps backwards towards the opposite direction. It can be shown, that combining the estimates produced by these two filters in a suitable way produces an smoothed estimate for the state, which has lower mean square error than any of these two filters alone (Gelb, 1974). With linear models the forward-backward smoother gives the same error as the RTS-smoother, but in non-linear cases the error behaves differently in some situations. In this toolbox forward-backward smoothing solution can be calculated with function `tf_smooth` (see page 45).

### 2.1.4 Demonstration: 2D CWPA-model

Let's now consider a very simple case, where we track an object moving in two dimensional space with a sensor, which gives measurements of target's position in cartesian coordinates $x$ and $y$. In addition to position target also has state variables for its velocities and accelerations toward both coordinate axes, $\dot{x}$, $\dot{y}$, $\ddot{x}$ and $\ddot{y}$. In other words, the state of a moving object on time step $k$ can be expressed as a vector

$$\mathbf{x}_k = \begin{pmatrix} x_k & y_k & \dot{x}_k & \dot{y}_k & \ddot{x}_k & \ddot{y}_k \end{pmatrix}^T. \tag{22}$$

In continous case the dynamics of the target's motion can be modelled as a linear, time-invariant system

$$\frac{d\mathbf{x}(t)}{dt} = \underbrace{\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{\mathbf{F}} \mathbf{x}(t) + \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\mathbf{L}} \mathbf{w}(t), \tag{23}$$

where $\mathbf{x}(t)$ is the target's state on the time $t$ and $\mathbf{w}(t)$ is a white noise process with power spectral density

$$\mathbf{Q}_c = \begin{pmatrix} q & 0 \\ 0 & q \end{pmatrix} = \begin{pmatrix} 0.2 & 0 \\ 0 & 0.2 \end{pmatrix}. \tag{24}$$

As can be seen from the equation the acceleration of the object is perturbed with a white noise process and hence this model has the name continous Wiener process acceleration (CWPA) model. There is also other models similar to this, as for example the continous white noise acceleration (CWNA) model (Bar-Shalom et al., 2001), where the velocity is perturbed with a white noise process.

The measurement matrix is set to

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{25}$$

which means that the observe only the position of the moving object. To be able to estimate this system with a discrete-time Kalman filter the differential equation defined above must be discretized somehow to get a discrete-time state equation of the form (12). It turns out, that the matrices $\mathbf{A}$ and $\mathbf{Q}$ can be calculated analytically with equations (15) and (16) to give the following:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & \Delta\mathrm{t} & 0 & \frac{1}{2}\Delta\mathrm{t}^2 & 0 \\ 0 & 1 & 0 & \Delta\mathrm{t} & 0 & \frac{1}{2}\Delta\mathrm{t}^2 \\ 0 & 0 & 1 & 0 & \Delta\mathrm{t} & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta\mathrm{t} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{26}$$

$$\mathbf{Q} = \begin{pmatrix} \frac{1}{20}\Delta\mathrm{t}^5 & 0 & \frac{1}{8}\Delta\mathrm{t}^4 & 0 & \frac{1}{6}\Delta\mathrm{t}^3 & 0 \\ 0 & \frac{1}{20}\Delta\mathrm{t}^5 & 0 & \frac{1}{8}\Delta\mathrm{t}^4 & 0 & \frac{1}{6}\Delta\mathrm{t}^3 \\ \frac{1}{8}\Delta\mathrm{t}_\mathrm{k}\,4 & 0 & \frac{1}{6}\Delta\mathrm{t}^3 & 0 & \frac{1}{2}\Delta\mathrm{t}^2 & 0 \\ 0 & \frac{1}{8}\Delta\mathrm{t}^4 & 0 & \frac{1}{6}\Delta\mathrm{t}^3 & 0 & \frac{1}{2}\Delta\mathrm{t}^2 \\ \frac{1}{6}\Delta\mathrm{t}^3 & 0 & \frac{1}{2}\Delta\mathrm{t}^2 & 0 & \Delta\mathrm{t} & 0 \\ 0 & \frac{1}{6}\Delta\mathrm{t}^3 & 0 & \frac{1}{2}\Delta\mathrm{t}^2 & 0 & \Delta\mathrm{t} \end{pmatrix} q, \tag{27}$$

where the stepsize is set to $\Delta\mathrm{t} = 0.5$. These matrices can also calculated using the function `lti_disc` introduced in section 2.1 with the following code line:

```
[A,Q] = lti_disc(F,L,Qc,dt);
```

where matrices `F` and `L` are assumed to contain the matrices from equation (23).

The object starts from origo with zero velocity and acceleration and the process is simulated 50 steps. The variance for the measurements is set to

$$R = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix}, \tag{28}$$

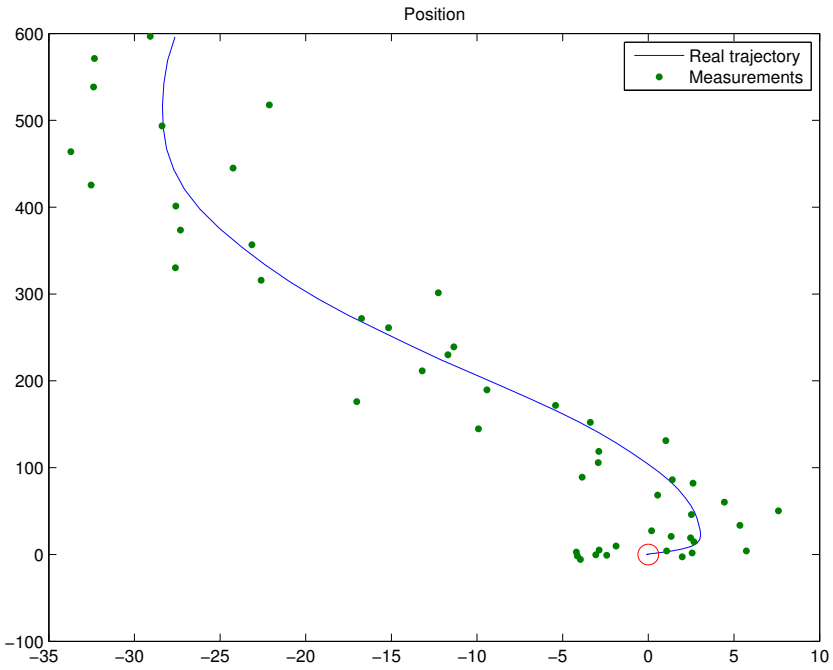


Figure 1: The real position of the moving object and the simulated measurements of it using the CWPA model. The circle marks the starting point of the object.

which is relatively high so that the the difference between the filtering and smoothing (described in next section) estimates becomes clearer. The real position of the object and measurements of it are plotted in the figure 1.

The filtering is done with the following code fragment:

```
MM = zeros(size(m,1), size(Y,2));
PP = zeros(size(m,1), size(m,1), size(Y,2));

for i = 1:size(Y,2)
   [m,P] = kf_predict(m,P,A,Q);
   [m,P] = kf_update(m,P,Y(:,i),H,R);
   MM(:,i) = m;
   PP(:,:,i) = P;
end
```

In the first 2 lines the space for state mean and covariance estimates is reserved, and the rest of the code contains the actual filtering loop, where we make the predict and update steps of the Kalman filter. The variables `m` and `P` are assumed to contain the initial guesses for the state mean and covariance before reaching the for-statement. Variable `Y` is assumed to contain the measurements made from the system (See the full source code of the example (`kf_cwpa_demo.m`) provided with the toolbox to see how we generated the measurements by simulating the dynamic system). In the end of each iteration the acquired estimates are stored to matrices `MM` and `PP`, for which we reserved space earlier. The estimates for object's position and velocity with Kalman filter and are plotted in figure 2.

The smoothed estimates for the state mean and covariance can be calculated with the following code line:

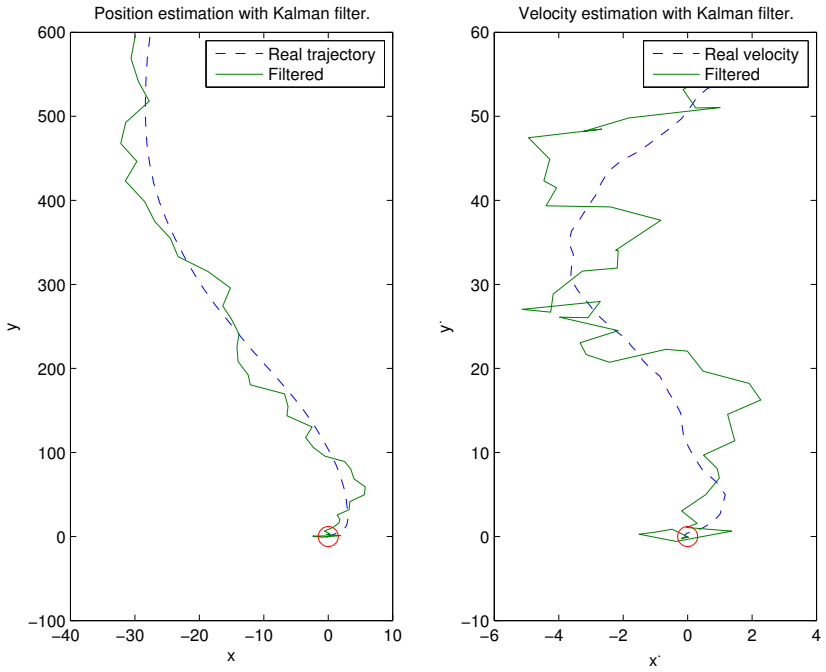


Figure 2: Estimates for position and velocity of the moving object using the Kalman filter and CWPA model.

```
[SM,SP] = rts_smooth(MM,PP,A,Q);
```

The calculated smoothed estimates for object's position and velocity for the earlier demonstration are plotted in figure 3. As expected the smoother produces more accurate estimates than the filter as it uses all measurements for the estimation each time step. Note that the difference between the smoothed and filtered estimated would be smaller, if the measurements were more accurate, as now the filter performs rather badly due to the great uncertaintity in the measurements. The smoothing results of a forward-backward smoother are not plotted here, as the result are exactly the same as with the RTS smoother.

As one would expect the estimates for object's velocity are clearly less accurate than the estimates for the object's position as the positions are observed directly and the velocities only indirectly. If the velocities were also observed not only the velocity estimates would get more accurate, but also the position ones as well.

## 2.2 Nonlinear state space estimation

In many cases interesting dynamic systems are not linear by nature, so the traditional Kalman filter cannot be applied in estimating the state of such a system. In these kind of systems, both the dynamics and the measurement processes can be nonlinear, or only one them. In this section, we describe two extensions to the traditional Kalman filter, which can be applied for estimating nonlinear dynamical systems by forming Gaussian approximations to the joint distribution of the state $\mathbf{x}$ and measurement $\mathbf{y}$. First we present the Extended Kalman filter (EKF), which is based on Taylor series approximation of the joint distribution, and then the Unscented Kalman filter (UKF), which is respectively based on the unscented transformation of the joint distribution.

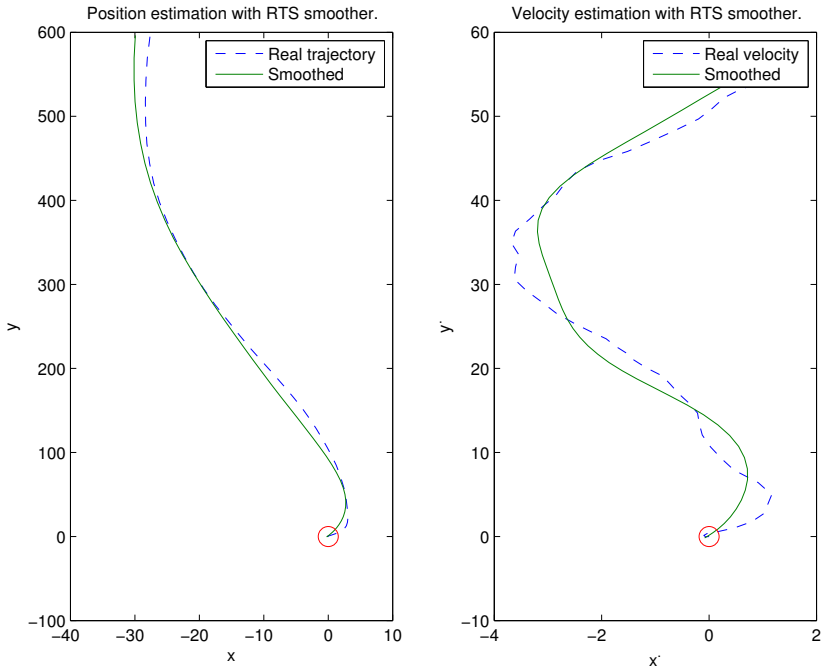


Figure 3: Estimate for position and velocity of the moving object using the RTS smoother and CWPA model.

### 2.2.1 Taylor Series Based Approximations

Next we present linear and quadratic approximations for the distribution of variable $\mathbf{y}$, which is generated with a non-linear transformation of a Gaussian random variable $\mathbf{x}$ as follows:

$$\begin{aligned} \mathbf{x} &\sim \mathrm{N}(\mathbf{m}, \mathbf{P}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}), \end{aligned} \tag{29}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, and $\mathbf{g} : \mathbb{R}^n \mapsto \mathbb{R}^m$ is a general non-linear function. Solving the distribution of $\mathbf{y}$ formally is in general not possible, because it is non-Gaussian for all by linear $\mathbf{g}$, so in practice it must be approximated somehow. The joint distribution of $\mathbf{x}$ and $\mathbf{y}$ can be formed with, for example, linear and quadratic approximations, which we present next. See, for example, (Bar-Shalom et al., 2001) for the derivation of these approximations.

#### Linear Approximation

The linear approximation based Gaussian approximation of the joint distribution of variables $\mathbf{x}$ and $\mathbf{y}$ defined by equations (29) is given as

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \sim \mathrm{N}\left( \begin{pmatrix} \mathbf{m} \\ \boldsymbol{\mu}_L \end{pmatrix}, \begin{pmatrix} \mathbf{P} & \mathbf{C}_L \\ \mathbf{C}_L^T & \mathbf{S}_L \end{pmatrix} \right), \tag{30}$$

where

$$\begin{aligned} \boldsymbol{\mu}_L &= \mathbf{g}(\mathbf{m}) \\ \mathbf{S}_L &= \mathbf{G}_{\mathbf{x}}(\mathbf{m})\, \mathbf{P}\, \mathbf{G}_{\mathbf{x}}^T(\mathbf{m}) \\ \mathbf{C}_L &= \mathbf{P}\, \mathbf{G}_{\mathbf{x}}^T(\mathbf{m}), \end{aligned} \tag{31}$$

and $\mathbf{G}_{\mathbf{x}}(\mathbf{m})$ is the Jacobian matrix of $\mathbf{g}$ with elements

$$[\mathbf{G}_{\mathbf{x}}(\mathbf{m})]_{j,j'} = \left.\frac{\partial g_j(\mathbf{x})}{\partial x_{j'}}\right|_{\mathbf{x}=\mathbf{m}}. \tag{32}$$

**Quadratic Approximation**

The quadratic approximations retain also the second order terms of the Taylor series expansion of the non-linear function:

$$\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} \sim \mathrm{N}\left(\begin{pmatrix}\mathbf{m}\\ \boldsymbol{\mu}_Q\end{pmatrix}, \begin{pmatrix}\mathbf{P} & \mathbf{C}_Q\\ \mathbf{C}_Q^T & \mathbf{S}_Q\end{pmatrix}\right), \tag{33}$$

where the parameters are

$$\begin{aligned}
\boldsymbol{\mu}_Q &= \mathbf{g}(\mathbf{m}) + \frac{1}{2}\sum_i \mathbf{e}_i \ \mathrm{tr}\left\{\mathbf{G}_{\mathbf{xx}}^{(i)}(\mathbf{m})\,\mathbf{P}\right\}\\
\mathbf{S}_Q &= \mathbf{G}_{\mathbf{x}}(\mathbf{m})\,\mathbf{P}\,\mathbf{G}_{\mathbf{x}}^T(\mathbf{m}) + \frac{1}{2}\sum_{i,i'} \mathbf{e}_i\,\mathbf{e}_{i'}^T \ \mathrm{tr}\left\{\mathbf{G}_{\mathbf{xx}}^{(i)}(\mathbf{m})\,\mathbf{P}\,\mathbf{G}_{\mathbf{xx}}^{(i')}(\mathbf{m})\,\mathbf{P}\right\}\\
\mathbf{C}_Q &= \mathbf{P}\,\mathbf{G}_{\mathbf{x}}^T(\mathbf{m}),
\end{aligned} \tag{34}$$

$\mathbf{G}_{\mathbf{x}}(\mathbf{m})$ is the Jacobian matrix (32) and $\mathbf{G}_{\mathbf{xx}}^{(i)}(\mathbf{m})$ is the Hessian matrix of $g_i(\cdot)$ evaluated at $\mathbf{m}$:

$$\left[\mathbf{G}_{\mathbf{xx}}^{(i)}(\mathbf{m})\right]_{j,j'} = \left.\frac{\partial^2 g_i(\mathbf{x})}{\partial x_j\,\partial x_{j'}},\right|_{\mathbf{x}=\mathbf{m}}. \tag{35}$$

$\mathbf{e}_i = (0\ \cdots\ 0\ 1\ 0\ \cdots\ 0)^T$ is the unit vector in direction of the coordinate axis $i$.

### 2.2.2 Extended Kalman filter

The extended Kalman filter (see, for instance, Jazwinski, 1970; Maybeck, 1982a; Bar-Shalom et al., 2001; Grewal and Andrews, 2001; Särkkä, 2006) extends the scope of Kalman filter to nonlinear optimal filtering problems by forming a Gaussian approximation to the joint distribution of state $\mathbf{x}$ and measurements $\mathbf{y}$ using a Taylor series based transformation. First and second order extended Kalman filters are presented, which are based on linear and quadratic approximations to the transformation. Higher order filters are also possible, but not presented here.

The filtering model used in the EKF is

$$\begin{aligned}
\mathbf{x}_k &= \mathbf{f}(\mathbf{x}_{k-1}, k-1) + \mathbf{q}_{k-1}\\
\mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k, k) + \mathbf{r}_k,
\end{aligned} \tag{36}$$

where $\mathbf{x}_k \in \mathbb{R}^n$ is the state, $\mathbf{y}_k \in \mathbb{R}^m$ is the measurement, $\mathbf{q}_{k-1} \sim \mathrm{N}(\mathbf{0}, \mathbf{Q}_{k-1})$ is the process noise, $\mathbf{r}_k \sim \mathrm{N}(\mathbf{0}, \mathbf{R}_k)$ is the measurement noise, $\mathbf{f}$ is the (possibly nonlinear) dynamic model function and $\mathbf{h}$ is the (again possibly nonlinear) measurement model function. The first and second order extended Kalman filters approximate the distribution of state $\mathbf{x}_k$ given the observations $\mathbf{y}_{1:k}$ with a Gaussian:

$$p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k}) \approx \mathrm{N}(\mathbf{x}_k \,|\, \mathbf{m}_k, \mathbf{P}_k). \tag{37}$$

**First Order Extended Kalman Filter**

Like Kalman filter, also the extended Kalman filter is separated to two steps. The steps for the first order EKF are as follows:

- *Prediction:*

$$\begin{aligned} \mathbf{m}_k^- &= \mathbf{f}(\mathbf{m}_{k-1}, k-1) \\ \mathbf{P}_k^- &= \mathbf{F}_{\mathbf{x}}(\mathbf{m}_{k-1}, k-1)\,\mathbf{P}_{k-1}\,\mathbf{F}_{\mathbf{x}}^T(\mathbf{m}_{k-1}, k-1) + \mathbf{Q}_{k-1}. \end{aligned} \tag{38}$$

- *Update:*

$$\begin{aligned} \mathbf{v}_k &= \mathbf{y}_k - \mathbf{h}(\mathbf{m}_k^-, k) \\ \mathbf{S}_k &= \mathbf{H}_{\mathbf{x}}(\mathbf{m}_k^-, k)\,\mathbf{P}_k^-\,\mathbf{H}_{\mathbf{x}}^T(\mathbf{m}_k^-, k) + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_k^-\,\mathbf{H}_{\mathbf{x}}^T(\mathbf{m}_k^-, k)\,\mathbf{S}_k^{-1} \\ \mathbf{m}_k &= \mathbf{m}_k^- + \mathbf{K}_k\,\mathbf{v}_k \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\,\mathbf{S}_k\,\mathbf{K}_k^T, \end{aligned} \tag{39}$$

where the matrices $\mathbf{F}_{\mathbf{x}}(\mathbf{m}, k-1)$ and $\mathbf{H}_{\mathbf{x}}(\mathbf{m}, k)$ are the Jacobians of $\mathbf{f}$ and $\mathbf{h}$, with elements

$$[\mathbf{F}_{\mathbf{x}}(\mathbf{m}, k-1)]_{j,j'} = \left.\frac{\partial f_j(\mathbf{x}, k-1)}{\partial x_{j'}}\right|_{\mathbf{x}=\mathbf{m}} \tag{40}$$

$$[\mathbf{H}_{\mathbf{x}}(\mathbf{m}, k)]_{j,j'} = \left.\frac{\partial h_j(\mathbf{x}, k)}{\partial x_{j'}}\right|_{\mathbf{x}=\mathbf{m}}. \tag{41}$$

Note that the difference between first order EKF and KF is that the matrices $\mathbf{A}_k$ and $\mathbf{H}_k$ in KF are replaced with Jacobian matrices $\mathbf{F}_{\mathbf{x}}(\mathbf{m}_{k-1}, k-1)$ and $\mathbf{H}_{\mathbf{x}}(\mathbf{m}_k^-, k)$ in EKF. Predicted mean $\mathbf{m}_k^-$ and residual of prediction $\mathbf{v}_k$ are also calculated differently in the EKF. In this toolbox the prediction and update steps of the first order EKF can be computed with functions `ekf_predict1` and `ekf_update1` (see page 45), respectively.

### Second Order Extended Kalman Filter

The corresponding steps for the second order EKF are as follows:

- *Prediction:*

$$\begin{aligned} \mathbf{m}_k^- &= \mathbf{f}(\mathbf{m}_{k-1}, k-1) + \frac{1}{2}\sum_i \mathbf{e}_i\ \mathrm{tr}\left\{\mathbf{F}_{\mathbf{xx}}^{(i)}(\mathbf{m}_{k-1}, k-1)\,\mathbf{P}_{k-1}\right\} \\ \mathbf{P}_k^- &= \mathbf{F}_{\mathbf{x}}(\mathbf{m}_{k-1}, k-1)\,\mathbf{P}_{k-1}\,\mathbf{F}_{\mathbf{x}}^T(\mathbf{m}_{k-1}, k-1) \\ &\quad + \frac{1}{2}\sum_{i,i'} \mathbf{e}_i\,\mathbf{e}_{i'}^T\,\mathrm{tr}\left\{\mathbf{F}_{\mathbf{xx}}^{(i)}(\mathbf{m}_{k-1}, k-1)\mathbf{P}_{k-1}\mathbf{F}_{\mathbf{xx}}^{(i')}(\mathbf{m}_{k-1}, k-1)\mathbf{P}_{k-1}\right\} \\ &\quad + \mathbf{Q}_{k-1}. \end{aligned} \tag{42}$$

- *Update:*

$$\begin{aligned} \mathbf{v}_k &= \mathbf{y}_k - \mathbf{h}(\mathbf{m}_k^-, k) - \frac{1}{2}\sum_i \mathbf{e}_i\ \mathrm{tr}\left\{\mathbf{H}_{\mathbf{xx}}^{(i)}(\mathbf{m}_k^-, k)\,\mathbf{P}_k^-\right\} \\ \mathbf{S}_k &= \mathbf{H}_{\mathbf{x}}(\mathbf{m}_k^-, k)\,\mathbf{P}_k^-\,\mathbf{H}_{\mathbf{x}}^T(\mathbf{m}_k^-, k) \\ &\quad + \frac{1}{2}\sum_{i,i'} \mathbf{e}_i\,\mathbf{e}_{i'}^T\ \mathrm{tr}\left\{\mathbf{H}_{\mathbf{xx}}^{(i)}(\mathbf{m}_k^-, k)\,\mathbf{P}_k^-\,\mathbf{H}_{\mathbf{xx}}^{(i')}(\mathbf{m}_k^-, k)\,\mathbf{P}_k^-\right\} + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_k^-\,\mathbf{H}_{\mathbf{x}}^T(\mathbf{m}_k^-, k)\,\mathbf{S}_k^{-1} \\ \mathbf{m}_k &= \mathbf{m}_k^- + \mathbf{K}_k\,\mathbf{v}_k \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\,\mathbf{S}_k\,\mathbf{K}_k^T, \end{aligned} \tag{43}$$

where matrices $\mathbf{F}_{\mathbf{x}}(\mathbf{m}, k-1)$ and $\mathbf{H}_{\mathbf{x}}(\mathbf{m}, k)$ are Jacobians as in the first order EKF, given by Equations (40) and (41). The matrices $\mathbf{F}^{(i)}_{\mathbf{xx}}(\mathbf{m}, k-1)$ and $\mathbf{H}^{(i)}_{\mathbf{xx}}(\mathbf{m}, k)$ are the Hessian matrices of $f_i$ and $h_i$:

$$\left[\mathbf{F}^{(i)}_{\mathbf{xx}}(\mathbf{m}, k-1)\right]_{j,j'} = \left.\frac{\partial^2 f_i(\mathbf{x}, k-1)}{\partial x_j \, \partial x_{j'}}\right|_{\mathbf{x}=\mathbf{m}} \tag{44}$$

$$\left[\mathbf{H}^{(i)}_{\mathbf{xx}}(\mathbf{m}, k)\right]_{j,j'} = \left.\frac{\partial^2 h_i(\mathbf{x}, k)}{\partial x_j \, \partial x_{j'}}\right|_{\mathbf{x}=\mathbf{m}}, \tag{45}$$

$\mathbf{e}_i = (0 \ \cdots \ 0 \ 1 \ 0 \ \cdots \ 0)^T$ is a unit vector in direction of the coordinate axis $i$, that is, it has a 1 at position $i$ and 0 at other positions.

The prediction and update steps of the second order EKF can be computed in this toolbox with functions `ekf_predict2` and `ekf_update2`, respectively. By taking the second order terms into account, however, doesn't quarantee, that the results get any better. Depending on problem they might even get worse, as we shall see in the later examples.

**The Limitations of EKF**

As discussed in, for example, (Julier and Uhlmann, 2004) the EKF has a few serious drawbacks, which should be kept in mind when it's used:

1. As we shall see in some of the later demonstrations, the linear and quadratic transformations produces realiable results only when the error propagation can be well approximated by a linear or a quadratic function. If this condition is not met, the performance of the filter can be extremely poor. At worst, its estimates can diverge altogether.

2. The Jacobian matrices (and Hessian matrices with second order filters) need to exist so that the transformation can be applied. However, there are cases, where this isn't true. For example, the system might be jump-linear, in which the parameters can change abruptly (Julier and Uhlmann, 2004).

3. In many cases the calculation of Jacobian and Hessian matrices can be a very difficult process, and its also prone to human errors (both derivation and programming). These errors are usually very hard to debug, as its hard to see which parts of the system produces the errors by looking at the estimates, especially as usually we don't know which kind of performance we should expect. For example, in the last demonstration (Reentry Vehicle Tracking) the first order derivatives were quite troublesome to calcute, even though the equations themselves were relatively simple. The second order derivatives would have even taken many more times of work.

### 2.2.3 Extended Kalman smoother

The difference between the first order extended Kalman smoother (Cox, 1964; Sage and Melsa, 1971) and the traditional Kalman smoother is the same as the difference between first order EKF and KF, that is, matrix $\mathbf{A}_k$ in Kalman smoother is replaced with Jacobian $\mathbf{F}_{\mathbf{x}}(\mathbf{m}_{k-1}, k-1)$, and $\mathbf{m}^-_{k+1}$ is calculated using the model function $\mathbf{f}$. Thus, the equations for the extended Kalman smoother can

be written as

$$\begin{aligned}
\mathbf{m}_{k+1}^{-} &= \mathbf{f}(\mathbf{m}_k, k) \\
\mathbf{P}_{k+1}^{-} &= \mathbf{F}_{\mathbf{x}}(\mathbf{m}_k, k)\, \mathbf{P}_k\, \mathbf{F}_{\mathbf{x}}^{T}(\mathbf{m}_k, k) + \mathbf{Q}_k \\
\mathbf{C}_k &= \mathbf{P}_k\, \mathbf{F}_{\mathbf{x}}^{T}(\mathbf{m}_k, k)\, [\mathbf{P}_{k+1}^{-}]^{-1} \\
\mathbf{m}_k^s &= \mathbf{m}_k + \mathbf{C}_k\, [\mathbf{m}_{k+1}^s - \mathbf{m}_{k+1}^{-}] \\
\mathbf{P}_k^s &= \mathbf{P}_k + \mathbf{C}_k\, [\mathbf{P}_{k+1}^s - \mathbf{P}_{k+1}^{-}]\, \mathbf{C}_k^T.
\end{aligned} \tag{46}$$

First order smoothing solutiong with a RTS type smoother can be computed with function `erts_smooth1`, and with forward-backward type smoother the computation can be done with function `etf_smooth1`.

Higher order smoothers are also possible, but not described here, as they are not currently implemented in this toolbox.

### 2.2.4 Demonstration: Tracking a random sine signal

Next we consider a simple, yet practical, example of a nonlinear dynamic system, in which we estimate a random sine signal using the extended Kalman filter. By random we mean that the angular velocity and the amplitude of the signal can vary through time. In this example the nonlinearity in the system is expressed through the measurement model, but it would also be possible to express it with the dynamic model and let the measurement model be linear.

The state vector in this case can be expressed as

$$\mathbf{x}_k = \begin{pmatrix} \theta_k & \omega_k & a_k \end{pmatrix}^T, \tag{47}$$

where $\theta_k$ is the parameter for the sine function on time step $k$, $\omega_k$ is the angular velocity on time step $k$ and $a_k$ is the amplitude on time step $k$. The evolution of parameter $\theta$ is modelled with a discretized Wiener velocity model, where the velocity is now the angular velocity:

$$\frac{d\theta}{dt} = \omega. \tag{48}$$

The values of $\omega$ and $a$ are perturbed with one dimensional white noise processes $w_a(t)$ and $w_w(t)$, so the signal isn't totally deterministic:

$$\frac{da}{dt} = w_a(t) \tag{49}$$

$$\frac{dw}{dt} = w_w(t). \tag{50}$$

Thus, the continous-time dynamic equation can be written as

$$\frac{d\mathbf{x}(t)}{dt} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{x}(t) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{w}(t), \tag{51}$$

where the white noise process $\mathbf{w}(t)$ has power spectral density

$$\mathbf{Q}_c = \begin{pmatrix} q_1 & 0 \\ 0 & q_2 \end{pmatrix}. \tag{52}$$

Variables $q_1$ and $q_2$ describe the strengths of random perturbations of the angular velocity and the amplitude, respectively, which are in this demonstration are set to $q_1 = 0.2$ and $q_2 = 0.1$. By using the equation (15) the discretized form of the dynamic equation can written as

$$\mathbf{x}_k = \begin{pmatrix} 1 & \Delta\mathrm{t} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{x}_{k-1} + \mathbf{q}_{k-1}, \tag{53}$$

where $\Delta\mathrm{t}$ is the step size (with value $\Delta\mathrm{t} = 0.01$ in this case), and using the equation (16) the covariance matrix $\mathbf{Q}_{k-1}$ of the discrete Gaussian white noise process $\mathbf{q}_{k-1} \sim N(\mathbf{0}, \mathbf{Q}_{k-1})$ can be easily computed to give

$$\mathbf{Q}_{k-1} = \begin{pmatrix} \frac{1}{3}\Delta\mathrm{t}^3\, q_1 & \frac{1}{2}\Delta\mathrm{t}^2\, q_1 & 0 \\ \frac{1}{2}\Delta\mathrm{t}^2\, q_1 & \Delta\mathrm{t}\, q_1 & 0 \\ 0 & 0 & \Delta\mathrm{t}\, q_2 \end{pmatrix}. \tag{54}$$

As stated above, the non-linearity in this case is expressed by the measurement model, that is, we propagate the current state through a non-linear measurement function $\mathbf{h}(\mathbf{x}_k, k)$ to get an actual measurement. Naturally the function in this case is the actual sine function

$$\mathbf{h}(\mathbf{x}_k, k) = a_k \sin(\theta_k). \tag{55}$$

With this the measurement model can be written as

$$y_k = \mathbf{h}(\mathbf{x}_k, k) + r_k = a_k \sin(\theta_k) + r_k, \tag{56}$$

where $r_k$ is white, univariate Gaussian noise with zero mean and variance $\sigma_r = 1$.

The derivatives of the measurement function with respect to state variables are

$$\begin{aligned} \frac{\partial \mathbf{h}(\mathbf{x}_k, k)}{\partial \theta_k} &= a_k \cos(\theta_k) \\ \frac{\partial \mathbf{h}(\mathbf{x}_k, k)}{\partial \omega_k} &= 0 \\ \frac{\partial \mathbf{h}(\mathbf{x}_k, k)}{\partial a_k} &= \sin(\theta_k), \end{aligned} \tag{57}$$

so the Jacobian matrix (actually in this case, a vector, as the measurements are only one dimensional) needed by the EKF can be written as

$$\mathbf{H}_{\mathbf{x}}(\mathbf{m}, k) = \begin{pmatrix} a_k \cos(\theta_k) & 0 & \sin(\theta_k) \end{pmatrix}. \tag{58}$$

We also filter the signal with second order EKF, so we need to evaluate the Hessian matrix of the measurement model function. In this case the second order derivatives of $\mathbf{h}$ with respect to all state variables can written as

$$\begin{aligned} \frac{\partial^2 \mathbf{h}(\mathbf{x}_k, k)}{\partial \theta_k \partial \theta_k} &= -a_k \sin(\theta_k) \\ \frac{\partial^2 \mathbf{h}(\mathbf{x}_k, k)}{\partial \theta_k \partial \omega_k} &= 0 \\ \frac{\partial^2 \mathbf{h}(\mathbf{x}_k, k)}{\partial \theta_k \partial a_k} &= \cos(\theta_k) \\ \frac{\partial^2 \mathbf{h}(\mathbf{x}_k, k)}{\partial \omega_k \partial \omega_k} &= 0 \\ \frac{\partial^2 \mathbf{h}(\mathbf{x}_k, k)}{\partial \omega_k \partial a_k} &= 0 \\ \frac{\partial^2 \mathbf{h}(\mathbf{x}_k, k)}{\partial a_k \partial a_k} &= 0. \end{aligned} \tag{59}$$

With these the Hessian matrix can expressed as

$$\mathbf{H}_{\mathbf{xx}}(\mathbf{m}, k) = \begin{pmatrix} -a_k \sin(\theta_k) & 0 & \cos(\theta_k) \\ 0 & 0 & 0 \\ \cos(\theta_k) & 0 & 0 \end{pmatrix}. \tag{60}$$

Note that as the measurements are only one dimensional we need to evaluate only one Hessian, and as the expressions are rather simple the computation of this Hessian is trivial. In case of higher dimensions we would need to evaluate the Hessian for each dimension separately, which could easily result in high amount of dense algebra.

In this demonstration program the measurement function (55) is computed with the following code:

```
function Y = ekf_demo1_h(x,param)
f = x(1,:);
a = x(3,:);
Y = a.*sin(f);
if size(x,1) == 7
    Y = Y + x(7,:);
end
```

where the parameter `x` is a vector containing a single state value, or a matrix containing multiple state values. It is also necessary to include the parameter `param`, which contains the other possible parameters for the functions (not present in this case). The last three lines are included for the augmented version of unscented Kalman filter (UKF), which is described later in this document. The Jacobian matrix of the measurement function (eq. (58)) is computed with the following function:

```
function dY = ekf_demo1_dh_dx(x, param)
f = x(1,:);
w = x(2,:);
a = x(3,:);
dY = [(a.*cos(f))' zeros(size(f,2),1) (sin(f))'];
```

The Hessian matrix of the measurement function (eq. 60) is computed with the following function:

```
function df = ekf_sine_d2h_dx2(x,param)
f = x(1);
a = x(3);

df = zeros(1,3,3);
df(1,:,:) = [-a*sin(f) 0 cos(f);
             0         0      0;
             cos(f)    0      0];
```

These functions are defined in files `efk_sine_h.m`, `ekf_sine_dh_dx.m` and `ekf_sine_d2h_dx2.m`, respectively. The handles of these functions are saved in the actual demonstration script file (`ekf_sine_demo.m`) with the following code lines:

```
h_func = @ekf_sine_h;
dh_dx_func = @ekf_sine_dh_dx;
d2h_dx2_func = @ekf_sine_d2h_dx2;
```

It is also important to check out that the implementation on calculating the derivatives is done right, as it is, especially with more complex models, easy to make errors in the equations. This can be done with function `der_check` (see page 54 for more details):

```
der_check(h_func, dh_dx_func, 1, [f w a]');
```

The third parameter with value `1` signals that we want to test the derivative of function's first (and in this case the only) dimension. Above we have assumed, that the variable `f` contains the parameter value for the sine function, `w` the angular velocity of the signal and `a` the amplitude of the signal.

After we have discretized the dynamic model and generated the real states and measurements same as in the previous example (the actual code lines are not stated here, see the full source code at end of this document), we can use the EKF to get the filtered estimates for the state means and covariances. The filtering (with first order EKF) is done almost the same as in the previous example:

```
MM = zeros(size(M,1),size(Y,2));
PP = zeros(size(M,1),size(M,1),size(Y,2));

for k=1:size(Y,2)
   [M,P] = ekf_predict1(M,P,A,Q);
   [M,P] = ekf_update1(M,P,Y(:,k),dh_dx_func,R*eye(1),h_func);
   MM(:,k)   = M;
   PP(:,:,k) = P;
end
```

As the model dynamics are in this case linear the prediction step functions exactly the same as in the case of traditional Kalman filter. In update step we pass the handles to the measurement model function and it's derivative function and the variance of measurement noise (parameters 6, 4 and 5, respectively), in addition to other parameters. These functions also have additional parameters, which might be needed in some cases. For example, the dynamic and measurement model functions might have parameters, which are needed when those functions are called. See the full function specifications in section 3 for more details about the parameters.

With second order EKF the filtering loop remains almost the same with the exception of update step:

```
MM2 = zeros(size(M,1),size(Y,2));
PP2 = zeros(size(M,1),size(M,1),size(Y,2));

for k=1:size(Y,2)
   [M,P] = ekf_predict1(M,P,A,Q);
   [M,P] = ekf_update2(M,P,Y(:,k),dh_dx_func,...
                       d2h_dx2_func,R*eye(1),h_func);
   MM(:,k)   = M;
   PP(:,:,k) = P;
end
```

The smoothing of state estimates using the extended RTS smoother is done sameways as in the previous example:

```
[SM1,SP1] = erts_smooth1(MM,PP,A,Q);
```

With the extended forward-backward smoother the smoothing is done with the following function call:

```
[SM2,SP2] = etf_smooth1(MM,PP,Y,A,Q,[],[],[],...
                        dh_dx_func,R*eye(1),h_func);
```

Here we have assigned empty vectors for parameters 6,7 and 8 (inverse prediction, its derivative w.r.t. to noise and its parameters, respectively), because they are not needed in this case.

To visualize the filtered and smoothed signal estimates we must evaluate the measurement model function with every state estimate to project the estimates to the measurement space. This can be done with the built-in Matlab function `feval`:

```
Y_m = feval(h_func, MM);
```

The filtered and smoothed estimates of the signals using the first order EKF, ERTS and ETF are plotted in figures 4, 5 and 6, respectively. The estimates produced by second order EKF are not plotted as they do not differ much from first order ones. As can be seen from the figures both smoothers give clearly better estimates than the filter. Especially in the beginning of the signal it takes a while for the filter to catch on to right track.

The difference between the smoothers doesn't become clear just by looking these figures. In some cases the forward-backward smoother gives a little better estimates, but it tends to be more sensitive about numerical accuracy and the process and measurement noises. To make a comparison between the performances of different methods we have listed the average of root mean square errors (RMSE) on 100 Monte Carlo simulations with different methods in table 1. In addition to RMSE of each state variable we also provide the estimation error in measurement space, because we might be more interested in estimating the actual value of signal than its components. Usually, however, the primary goal of these methods is to estimate the hidden state variables. The following methods were used:

- EKF1: First order extended Kalman filter.
- ERTS1: First order extended Rauch-Tung-Striebel smoother.
- ETF1: First order extended Forward-Backward smoother.
- EKF2: Second order extended Kalman filter.
- ERTS2: First order extended Rauch-Tung-Striebel smoother applied to second order EKF estimates.
- ETF2: First order extended Forward-Backward smoother applied to second order EKF estimates.
- UKF: unscented Kalman filter.
- URTS: unscented Rauch-Tung-Striebel smoother.

From the errors we can see that with filters EKF2 gives clearly the lowest errors with variables $\theta$ and $a$. Due to this also with smoothers ERTS2 and ETF2 give clearly lower errors than others. On the other hand EKF1 gives the lowest estimation error with variable $\omega$. Furthermore, with filters EKF1 also gives lowest error in measurement space. Each smoother, however, gives approximately the same error in measurement space. It can also be seen, that the UKF functions the worst in this case. This is due to linear and quadratic approximations used in EKF working well with this model. However, with more nonlinear models UKF is often superior over EKF, as we shall see in later sections.

In all, none of the used methods proved to be clearly superior over the others with this model. It is clear, however, that EKF should be preferred over UKF as it gives lower error and is slightly less demanding in terms of computation power. Whether first or second order EKF should be used is ultimately up to the goal of application. If the actual signal value is of interest, which is usually the case, then one should use first order EKF, but second order one might better at predicting new signal values as the variables $\theta$ and $a$ are closer to real ones on average.

### 2.2.5 Unscented Transform

Like Taylor series based approximation presented above also the *unscented transform* (UT) (Julier and Uhlmann, 1995, 2004; Wan and van der Merwe, 2001) can be used for forming a Gaussian approximation to the joint distribution of random variables $\mathbf{x}$ and $\mathbf{y}$, which are defined with equations (29). In UT we deterministically choose a fixed number of sigma points, which capture the

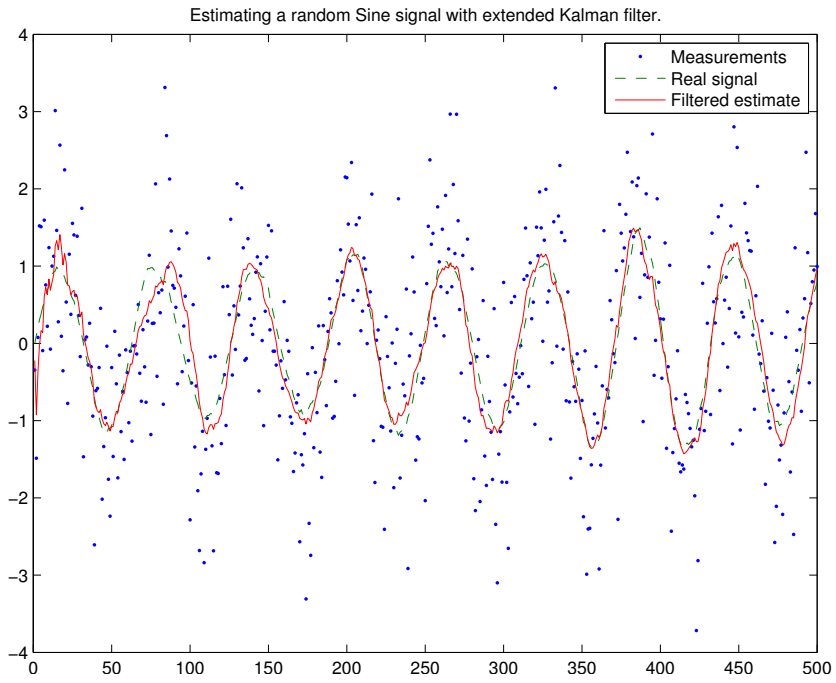


Figure 4: Filtered estimate of the sine signal using the first order extended Kalman filter.

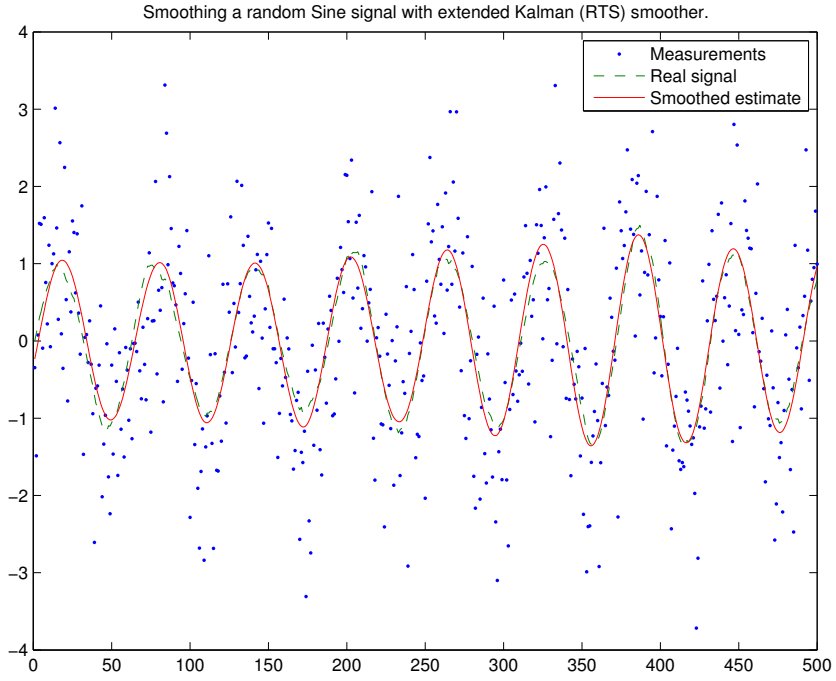


Figure 5: Smoothed estimate of the sine signal using the extended Kalman (RTS) smoother.

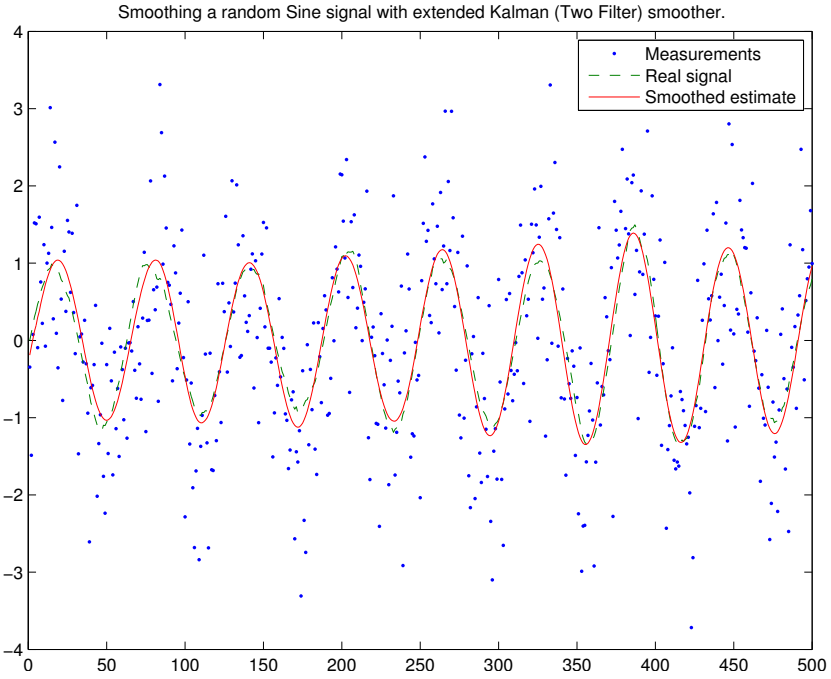


Figure 6: Smoothed estimate of the sine signal using a combination of two extended Kalman filters.

| *Method* | *RMSE*$[\theta]$ | *RMSE*$[\omega]$ | *RMSE*$[a]$ | *RMSE*$[y]$ |
|---|---|---|---|---|
| EKF1 | 0.64 | 0.53 | 0.40 | 0.24 |
| ERTS1 | 0.52 | 0.31 | 0.33 | 0.15 |
| ETF1 | 0.53 | 0.31 | 0.34 | 0.15 |
| EKF2 | 0.34 | 0.54 | 0.31 | 0.29 |
| ERTS2 | 0.24 | 0.30 | 0.18 | 0.15 |
| ETF2 | 0.24 | 0.30 | 0.18 | 0.15 |
| UKF | 0.59 | 0.56 | 0.39 | 0.27 |
| URTS | 0.45 | 0.30 | 0.30 | 0.15 |

Table 1: RMSEs of estimating the random sinusoid over 100 Monte Carlo simulations.

desired moments (atleast mean and covariance) of the original distribution of $\mathbf{x}$ exactly. After that we propagate the sigma points through the non-linear function $\mathbf{g}$ and estimate the moments of the transformed variable from them.

The advantage of UT over the Taylor series based approximation is that UT is better at capturing the higher order moments caused by the non-linear transform, as discussed in (Julier and Uhlmann, 2004). Also the Jacobian and Hessian matrices are not needed, so the estimation procedure is in general easier and less error-prone.

The unscented transform can be used to provide a Gaussian approximation for the joint distribution of variables $\mathbf{x}$ and $\mathbf{y}$ of the form

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \sim \mathrm{N}\left( \begin{pmatrix} \mathbf{m} \\ \boldsymbol{\mu}_U \end{pmatrix}, \begin{pmatrix} \mathbf{P} & \mathbf{C}_U \\ \mathbf{C}_U^T & \mathbf{S}_U \end{pmatrix} \right). \tag{61}$$

The (nonaugmented) transformation is done as follows:

1. Compute the set of $2n+1$ sigma points from the columns of the matrix $\sqrt{(n+\lambda)\,\mathbf{P}}$:

$$\begin{aligned} \mathbf{x}^{(0)} &= \mathbf{m} \\ \mathbf{x}^{(i)} &= \mathbf{m} + \left[\sqrt{(n+\lambda)\,\mathbf{P}}\right]_i, \quad i = 1, \ldots, n \\ \mathbf{x}^{(i)} &= \mathbf{m} - \left[\sqrt{(n+\lambda)\,\mathbf{P}}\right]_i, \quad i = n+1, \ldots, 2n \end{aligned} \tag{62}$$

   and the associated weights:

$$\begin{aligned} W_m^{(0)} &= \lambda/(n+\lambda) \\ W_c^{(0)} &= \lambda/(n+\lambda) + (1 - \alpha^2 + \beta) \\ W_m^{(i)} &= 1/\{2(n+\lambda)\}, \quad i = 1, \ldots, 2n \\ W_c^{(i)} &= 1/\{2(n+\lambda)\}, \quad i = 1, \ldots, 2n. \end{aligned} \tag{63}$$

   Parameter $\lambda$ is a scaling parameter, which is defined as

$$\lambda = \alpha^2\,(n + \kappa) - n. \tag{64}$$

   The positive constants $\alpha$, $\beta$ and $\kappa$ are used as parameters of the method.

2. Propagate each of the sigma points through non-linearity as

$$\mathbf{y}^{(i)} = \mathbf{g}(\mathbf{x}^{(i)}), \quad i = 0, \ldots, 2n. \tag{65}$$

3. Calculate the mean and covariance estimates for $\mathbf{y}$ as

$$\boldsymbol{\mu}_U \approx \sum_{i=0}^{2n} W_m^{(i)}\,\mathbf{y}^{(i)} \tag{66}$$

$$\mathbf{S}_U \approx \sum_{i=0}^{2n} W_c^{(i)}\,(\mathbf{y}^{(i)} - \boldsymbol{\mu}_U)\,(\mathbf{y}^{(i)} - \boldsymbol{\mu}_U)^T. \tag{67}$$

4. Estimate the cross-covariance between $\mathbf{x}$ and $\mathbf{y}$ as

$$\mathbf{C}_U \approx \sum_{i=0}^{2n} W_c^{(i)}\,(\mathbf{x}^{(i)} - \mathbf{m})\,(\mathbf{y}^{(i)} - \boldsymbol{\mu}_U)^T. \tag{68}$$

The square root of positive definite matrix $\mathbf{P}$ is defined as $\mathbf{A} = \sqrt{\mathbf{P}}$, where

$$\mathbf{P} = \mathbf{A}\mathbf{A}^T. \tag{69}$$

To calculate the matrix $\mathbf{A}$ we can use, for example, lower triangular matrix of the Cholesky factorialization, which can be computed with built-in Matlab function `chol`. For convience, we have provided a function (`schol`, see page 3.2), which computes the factorialization also for positive semidefinite matrices.

**The Matrix Form of UT**

The unscented transform described above can be written conviently in matrix form as follows:

$$\mathbf{X} = \begin{bmatrix}\mathbf{m} & \cdots & \mathbf{m}\end{bmatrix} + \sqrt{c}\begin{bmatrix}\mathbf{0} & \sqrt{\mathbf{P}} & -\sqrt{\mathbf{P}}\end{bmatrix} \tag{70}$$

$$\mathbf{Y} = \mathbf{g}(\mathbf{X}) \tag{71}$$

$$\boldsymbol{\mu}_U = \mathbf{Y}\,\mathbf{w}_m \tag{72}$$

$$\mathbf{S}_U = \mathbf{Y}\,\mathbf{W}\,\mathbf{Y}^T \tag{73}$$

$$\mathbf{C}_U = \mathbf{X}\,\mathbf{W}\,\mathbf{Y}^T, \tag{74}$$

where $\mathbf{X}$ is the matrix of sigma points, function $\mathbf{g}(\cdot)$ is applied to each column of the argument matrix separately, $c = \alpha^2\,(n + \kappa)$, and vector $\mathbf{w}_m$ and matrix $\mathbf{W}$ are defined as follows:

$$\mathbf{w}_m = \begin{bmatrix}W_m^{(0)} & \cdots & W_m^{(2n)}\end{bmatrix}^T \tag{75}$$

$$\begin{aligned}\mathbf{W} = &\left(\mathbf{I} - \begin{bmatrix}\mathbf{w}_m & \cdots & \mathbf{w}_m\end{bmatrix}\right) \\ &\times \operatorname{diag}(W_c^{(0)} \cdots W_c^{(2n)}) \\ &\times \left(\mathbf{I} - \begin{bmatrix}\mathbf{w}_m & \cdots & \mathbf{w}_m\end{bmatrix}\right)^T.\end{aligned} \tag{76}$$

See (Särkkä, 2006) for proof for this.

### 2.2.6 Unscented Kalman filter

The *unscented Kalman filter* (UKF) (Julier et al., 1995; Julier and Uhlmann, 2004; Wan and van der Merwe, 2001) makes use of the *unscented transform* described above to give a Gaussian approximation to the filtering solutions of non-linear optimal filtering problems of form (same as eq. (36), but restated here for convience)

$$\begin{aligned}\mathbf{x}_k &= \mathbf{f}(\mathbf{x}_{k-1}, k-1) + \mathbf{q}_{k-1} \\ \mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k, k) + \mathbf{r}_k,\end{aligned} \tag{77}$$

where $\mathbf{x}_k \in \mathbb{R}^n$ is the state, $\mathbf{y}_k \in \mathbb{R}^m$ is the measurement, $\mathbf{q}_{k-1} \sim \mathrm{N}(\mathbf{0}, \mathbf{Q}_{k-1})$ is the Gaussian process noise, and $\mathbf{r}_k \sim \mathrm{N}(\mathbf{0}, \mathbf{R}_k)$ is the Gaussian measurement noise.

Using the matrix form of UT described above the *prediction* and *update* steps of the UKF can computed as follows:

- *Prediction:* Compute the predicted state mean $\mathbf{m}_k^-$ and the predicted covariance $\mathbf{P}_k^-$ as

$$\begin{aligned}\mathbf{X}_{k-1} &= \begin{bmatrix}\mathbf{m}_{k-1} & \cdots & \mathbf{m}_{k-1}\end{bmatrix} + \sqrt{c}\begin{bmatrix}\mathbf{0} & \sqrt{\mathbf{P}_{k-1}} & -\sqrt{\mathbf{P}_{k-1}}\end{bmatrix} \\ \hat{\mathbf{X}}_k &= \mathbf{f}(\mathbf{X}_{k-1}, k-1) \\ \mathbf{m}_k^- &= \hat{\mathbf{X}}_k\,\mathbf{w}_m \\ \mathbf{P}_k^- &= \hat{\mathbf{X}}_k\,\mathbf{W}\,[\hat{\mathbf{X}}_k]^T + \mathbf{Q}_{k-1}.\end{aligned} \tag{78}$$

- *Update:* Compute the predicted mean $\boldsymbol{\mu}_k$ and covariance of the measurement $\mathbf{S}_k$, and the cross-covariance of the state and measurement $\mathbf{C}_k$:

$$\begin{aligned}
\mathbf{X}_k^- &= \begin{bmatrix}\mathbf{m}_k^- & \cdots & \mathbf{m}_k^-\end{bmatrix} + \sqrt{c}\begin{bmatrix}\mathbf{0} & \sqrt{\mathbf{P}_k^-} & -\sqrt{\mathbf{P}_k^-}\end{bmatrix} \\
\mathbf{Y}_k^- &= \mathbf{h}(\mathbf{X}_k^-, k) \\
\mu_k &= \mathbf{Y}_k^-\,\mathbf{w}_m \\
\mathbf{S}_k &= \mathbf{Y}_k^-\,\mathbf{W}\,[\mathbf{Y}_k^-]^T + \mathbf{R}_k \\
\mathbf{C}_k &= \mathbf{X}_k^-\,\mathbf{W}\,[\mathbf{Y}_k^-]^T.
\end{aligned} \tag{79}$$

  Then compute the filter gain $\mathbf{K}_k$ and the updated state mean $\mathbf{m}_k$ and covariance $\mathbf{P}_k$:

$$\begin{aligned}
\mathbf{K}_k &= \mathbf{C}_k\,\mathbf{S}_k^{-1} \\
\mathbf{m}_k &= \mathbf{m}_k^- + \mathbf{K}_k\,[\mathbf{y}_k - \boldsymbol{\mu}_k] \\
\mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\,\mathbf{S}_k\,\mathbf{K}_k^T.
\end{aligned} \tag{80}$$

The prediction and update steps of the nonaugmented UKF can be computed with functions `ukf_predict1` and `ukf_update1`, respectively.

**Augmented UKF**

It is possible to modify the UKF procedure described above by forming an *augmented* state variable, which concatenates the state and noise components together, so that the effect of process and measurement noises can be used to better capture the odd-order moment information. This requires that the sigma points generated during the predict step are also used in the update step, so that the effect of noise terms are truly propagated through the nonlinearity (Wu et al., 2005). If, however, we generate new sigma points in the update step the augmented approach give the same results as the nonaugmented, if we had assumed that the noises were additive. If the noises are not additive the augmented version should produce more accurate estimates than the nonaugmented version, even if new sigma points are created during the update step.

The *prediction* and *update* steps of the augmented UKF in matrix form are as follows:

- *Prediction:* Form a matrix of sigma points of the augmented state variable $\tilde{\mathbf{x}}_{k-1} = \begin{bmatrix}\mathbf{x}_{k-1}^T & \mathbf{q}_{k-1}^T & \mathbf{r}_{k-1}^T\end{bmatrix}^T$ as

$$\tilde{\mathbf{X}}_{k-1} = \begin{bmatrix}\tilde{\mathbf{m}}_{k-1} & \cdots & \tilde{\mathbf{m}}_{k-1}\end{bmatrix} + \sqrt{c}\begin{bmatrix}\mathbf{0} & \sqrt{\tilde{\mathbf{P}}_{k-1}} & -\sqrt{\tilde{\mathbf{P}}_{k-1}}\end{bmatrix}, \tag{81}$$

  where

$$\tilde{\mathbf{m}}_{k-1} = \begin{bmatrix}\mathbf{m}_{k-1}^T & \mathbf{0} & \mathbf{0}\end{bmatrix}^T \text{ and } \tilde{\mathbf{P}}_{k-1} = \begin{bmatrix}\mathbf{P_{k-1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{k-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{R}_{k-1}\end{bmatrix}. \tag{82}$$

  Then compute the predicted state mean $\mathbf{m}_k^-$ and the predicted covariance $\mathbf{P}_k^-$ as

$$\begin{aligned}
\hat{\mathbf{X}}_k &= \mathbf{f}(\mathbf{X^x}_{k-1}, \mathbf{X^q}_{k-1}, k-1) \\
\mathbf{m}_k^- &= \hat{\mathbf{X}}_k\,\mathbf{w}_m \\
\mathbf{P}_k^- &= \hat{\mathbf{X}}_k\,\mathbf{W}\,[\hat{\mathbf{X}}_k]^T,
\end{aligned} \tag{83}$$

  where we have denoted the components of sigma points which correspond to actual state variables and process noise with matrices $\mathbf{X}_{k-1}^x$ and $\mathbf{X}_{k-1}^q$, respectively. The state transition

function $\mathbf{f}$ is also augmented to incorporate the effect of process noise, which is now passed to the function as a second parameter. In additive noise case the process noise is directly added to the state variables, but other forms of noise effect are now also allowed.

- *Update:* Compute the predicted mean $\boldsymbol{\mu}_k$ and covariance of the measurement $\mathbf{S}_k$, and the cross-covariance of the state and measurement $\mathbf{C}_k$:

$$\begin{aligned} \mathbf{Y}_k^- &= \mathbf{h}(\hat{\mathbf{X}}_k, \mathbf{X}_{k-1}^r, k) \\ \mu_k &= \mathbf{Y}_k^- \, \mathbf{w}_m \\ \mathbf{S}_k &= \mathbf{Y}_k^- \, \mathbf{W} \, [\mathbf{Y}_k^-]^T \\ \mathbf{C}_k &= \hat{\mathbf{X}}_k \, \mathbf{W} \, [\mathbf{Y}_k^-]^T, \end{aligned} \tag{84}$$

  where we have denoted the component of sigma points corresponding to measurement noise with matrix $\mathbf{X}_{k-1}^r$. Like the state transition function $\mathbf{f}$ also the measurement function $\mathbf{h}$ is now augmented to incorporate the effect of measurement noise, which is passed as a second parameter to the function.

  Then compute the filter gain $\mathbf{K}_k$ and the updated state mean $\mathbf{m}_k$ and covariance $\mathbf{P}_k$:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{C}_k \, \mathbf{S}_k^{-1} \\ \mathbf{m}_k &= \mathbf{m}_k^- + \mathbf{K}_k \, [\mathbf{y}_k - \boldsymbol{\mu}_k] \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k \, \mathbf{S}_k \, \mathbf{K}_k^T. \end{aligned} \tag{85}$$

Note that nonaugmented form UKF is computationally less demanding than augmented form UKF, because it creates a smaller number of sigma points during the filtering procedure. Thus, the usage of the nonaugmented version should be preferred over the nonaugmented version, if the propagation of noise terms doesn't improve the accuracy of the estimates.

The prediction and update steps of the augmented UKF can be computed with functions `ukf_predict3` and `ukf_update3`, respectively. These functions concatenates the state variables, process and measurements noises to the augmented variables, as was done above.

It is also possible to separately concatenate only the state variables and process noises during prediction step and state variables and measurement noises during update step. Filtering solution based on this formulation can be computed with functions `ukf_predict2` and `ukf_update2`. However, these functions create new sigma points during the update step in addition to ones created during prediction step, and hence the higher moments might not get captured so effectively in cases, where the noise terms are additive.

### 2.2.7 Unscented Kalman smoother

The Rauch-Rung-Striebel type smoother using the unscented transformation (Särkkä, 2006c) can be used for computing a Gaussian approximation to the smoothing distribution of the step $k$:

$$p(\mathbf{x}_k|\mathbf{y}_{1:T}) \sim N(\mathbf{x}_k|\mathbf{m}_k^s, \mathbf{P}_k^s), \tag{86}$$

as follows (using again the matrix form):

- Form a matrix of sigma points of the augmented state variable $\tilde{\mathbf{x}}_{k-1} = \begin{bmatrix} \mathbf{x}_{k-1}^T & \mathbf{q}_{k-1}^T \end{bmatrix}^T$ as

$$\tilde{\mathbf{X}}_{k-1} = \begin{bmatrix} \tilde{\mathbf{m}}_{k-1} & \cdots & \tilde{\mathbf{m}}_{k-1} \end{bmatrix} + \sqrt{c} \begin{bmatrix} \mathbf{0} & \sqrt{\tilde{\mathbf{P}}_{k-1}} & -\sqrt{\tilde{\mathbf{P}}_{k-1}} \end{bmatrix}, \tag{87}$$

  where

$$\tilde{\mathbf{m}}_{k-1} = \begin{bmatrix} \mathbf{m}_{k-1}^T & \mathbf{0} \end{bmatrix}^T \text{ and } \tilde{\mathbf{P}}_{k-1} = \begin{bmatrix} \mathbf{P_{k-1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{k-1} \end{bmatrix}. \tag{88}$$

- Propagate the sigma points through the dynamic model:

$$\tilde{\mathbf{X}}_{k+1}^{-} = \mathbf{f}(\tilde{\mathbf{X}}_{k}^{x}, \tilde{\mathbf{X}}_{k}^{q}, k), \tag{89}$$

  where $\tilde{\mathbf{X}}_{k}^{x}$ and $\tilde{\mathbf{X}}_{k}^{q}$ denotes the parts of sigma points, which correspond to $\mathbf{x}_k$ and $\mathbf{q}_k$, respectively.

- Compute the predicted mean $\mathbf{m}_{k+1}^{-}$, covariance $\mathbf{P}_{k+1}^{-}$ and cross-covariance $\mathbf{C}_{k+1}$:

$$\begin{aligned} \mathbf{m}_{k+1}^{-} &= \tilde{\mathbf{X}}_{k+1}^{-x}\, \mathbf{w}_m \\ \mathbf{P}_{k+1}^{-} &= \tilde{\mathbf{X}}_{k+1}^{-x}\, \mathbf{W}\, [\tilde{\mathbf{X}}_{k+1}^{-x}]^T \\ \mathbf{C}_{k+1} &= \tilde{\mathbf{X}}_{k+1}^{-x}\, \mathbf{W}\, [\tilde{\mathbf{X}}_{k}^{x}]^T, \end{aligned} \tag{90}$$

  where $\tilde{\mathbf{X}}_{k+1}^{-x}$ denotes the part of propagated sigma points $\tilde{\mathbf{X}}_{k+1}^{-}$, which corresponds to $\mathbf{x}_k$.

- Compute the smoother gain $\mathbf{D}_k$, the smoothed mean $\mathbf{m}_k^s$ and the covariance $\mathbf{P}_k^s$:

$$\begin{aligned} \mathbf{D}_k &= \mathbf{C}_{k+1}[\mathbf{P}_{k+1}^{-}]^{-1} \\ \mathbf{m}_k^s &= \mathbf{m}_k + \mathbf{D}_k[\mathbf{m}_{k+1}^s - \mathbf{m}_{k+1}^{-}] \\ \mathbf{P}_k^s &= \mathbf{P}_k + \mathbf{D}_k[\mathbf{P}_{k+1}^s - \mathbf{P}_{k+1}^{-}]\mathbf{D}_k^T. \end{aligned} \tag{91}$$

The smoothing solution of this augmented type RTS smoother can be computed with function `urts_smooth2`. Also a nonaugmented version of this type smoother has been implemented, and a smoothing solution with that can be computed with function `urts_smooth1`.

### 2.2.8 Demonstration: UNGM-model

To illustrate some of the advantages of UKF over EKF and augmented form UKF over non-augmented lets now consider an example, in which we estimate a model called Univariate Nonstationary Growth Model (UNGM), which is previously used as benchmark, for example, in (Kotecha and Djuric, 2003) and (Wu et al., 2005). What makes this model particularly interesting in this case is that its highly nonlinear and bimodal, so it is really challenging for traditional filtering techniques. We also show how in this case the augmented version of UKF gives better performance than the nonaugmented version.

The dynamic state space model for UNGM can be written as

$$x_n = \alpha x_{n-1} + \beta \frac{x_{n-1}}{1 + x_{n-1}^2} + \gamma \cos(1.2(n-1)) + u_n \tag{92}$$

$$y_n = \frac{x_n^2}{20} + v_n, n = 1, \ldots, N \tag{93}$$

where $u_n \sim N(0, \sigma_u^2)$ and $v_n \sim N(0, \sigma_u^2)$. In this example we have set the parameters to $\sigma_u^2 = 1$, $\sigma_v^2 = 1$, $x_0 = 0.1$, $\alpha = 0.5$, $\beta = 25$, $\gamma = 8$, and $N = 500$. The cosine term in the state transition equation simulates the effect of time-varying noise.

In this demonstration the state transition is computed with the following function:

```
function x_n = ungm_f(x,param)
n = param(1);
x_n = 0.5*x(1,:) + 25*x(1,:)./(1+x(1,:).*x(1,:)) + 8*cos(1.2*(n-1));
if size(x,1) > 1
  x_n = x_n + x(2,:);
end
```

where the input parameter `x` contains the state on the previous time step. The current time step index $n$ needed by the cosine term is passed in the input parameter `param`. The last three lines in the function adds the process noise to the state component, if the augmented version of the UKF is used. Note that in augmented UKF the state, process noise and measurement noise terms are all concatenated together to the augmented variable, but in URTS the measurement noise term is left out. That is why we must make sure that the functions we declare are compatible with all cases (nonaugmented, augmented with and without measurement noise). In this case we check whether the state variable has second component (process noise) or not.

Similarly, the measurement model function is declared as

```
function y_n = ungm_h(x_n,param)
y_n = x_n(1,:).*x_n(1,:) ./ 20;
if size(x_n,1) == 3
   y_n = y_n + x_n(3,:);
end
```

The filtering loop for augmented UKF is as follows:

```
for k = 1:size(Y,2)
   [M,P,X_s,w] = ukf_predict3(M,P,f_func,u_n,v_n,k);
   [M,P] = ukf_update3(M,P,Y(:,k),h_func,v_n,X_s,w,[]);
   MM_UKF2(:,k)   = M;
   PP_UKF2(:,:,k) = P;
end
```

The biggest difference in this in relation to other filters is that now the predict step returns the sigma points (variable `X_s`) and their weigths (variable `w`), which must be passed as parameters to update function.

To compare the EKF and UKF to other possible filtering techniques we have also used a bootstrap filtering approach (Gordon et al., 1993), which belongs to class of *Sequential Monte Carlo* (SMC) methods (also known as *particle filters*). Basically the idea in SMC methods that is during each time step they draw a set of weighted particles from some appropriate importance distribution and after that the moments (that is, mean and covariance) of the function of interest (e.g. dynamic function in state space models) are estimated approximately from drawn samples. The weights of the particles are adjusted so that they are approximations to the relative posterior probabilities of the particles. Usually also a some kind of *resampling* scheme is used to avoid the problem of degenerate particles, that is, particles with near zero weights are removed and those with large weights are duplicated. In this example we used a *stratified resampling* algorithm (Kitagawa, 1996), which is optimal in terms of variance. In bootstrap filtering the dynamic model $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ is used as importance distribution, so its implementation is really easy. However, due to this a large number of particles might be needed for the filter to be effective. In this case 1000 particles were drawn on each step. The implementation of the bootstrap filter is commented out in the actual demonstration script (`ungm_demo.m`), because the used resampling function (`resampstr.m`) was originally provided in MCMCstuff toolbox (Vanhatalo et al., 2006), which can be found at http://www.lce.hut.fi/research/mm/mcmcstuff/.

In figure 7 we have plotted the 100 first samples of the signal as well as the estimates produced by EKF, augmented form UKF and bootstrap filter. The bimodality is easy to see from the figure. For example, during samples $10-25$ UKF is able to estimate the correct mode while the EKF estimates it wrong. Likewise, during steps $45-55$ and $85-95$ UKF has troubles in following the correct mode while EKF is more right. Bootstrap filter on the other hand tracks the correct mode on almost ever time step, although also it produces notable errors.

In figure 8 we have plotted the absolute errors and $3\sigma$ confidence intervals of the previous figures filtering results. It can be seen that the EKF is overoptimistic in many cases while UKF and boostrap

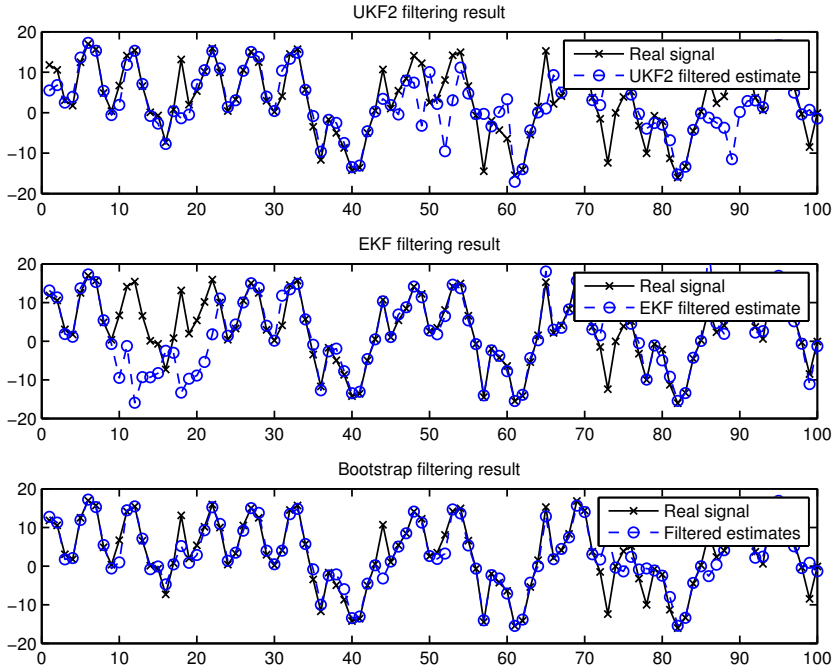


Figure 7: First 100 samples of filtering results of EKF, augmented form UKF and bootstrap filter for UNGM-model.

filter are better at telling when their results are unreliable. Also the lower error of bootstrap filter can be seen from the figure. The bimodality is also easy to notice on those samples, which were mentioned above.

The make a comparison between nonaugmented and augmented UKF we have plotted 100 first samples of their filtering results in figure 9. Results are very surprising (although same as in (Wu et al, 2005)). The reason why nonaugmented UKF gave so bad results is not clear. However, the better performance of augmented form UKF can be explained by the fact, that the process noise is taken into account more effectively when the sigma points are propagated through nonlinearity. In this case it seems to be very crucial, as the model is highly nonlinear and multi-modal.

Lastly in figure 10 we have plotted the mean square errors of each tested methods of 100 Monte Carlo runs. Average of those errors are listed in table 2. Here is a discussion for the results:

- It is surprising that the nonaugmented UKF seems to be better than EKF, while in above figures we have shown, that the nonaugmented UKF gives very bad results. Reason for this is simple: the variance of the actual signal is approximately 100, which means that by simply guessing zero we get better performance than with EKF, on average. The estimates of nonaugmented UKF didn't variate much on average, so they were better than those of EKF, which on the other hand variated greatly and gave huge errors in some cases. Because of this neither of the methods should be used for this problem, but if one has to choose between the two, that would be EKF, because in some cases it still gave (more or less) right answers, whereas UKF were practically always wrong.

- The second order EKF were also tested, but that diverged almost instantly, so it were left out from comparison.

- Augmented form UKF gave clearly the best performance from the tested Kalman filters. As

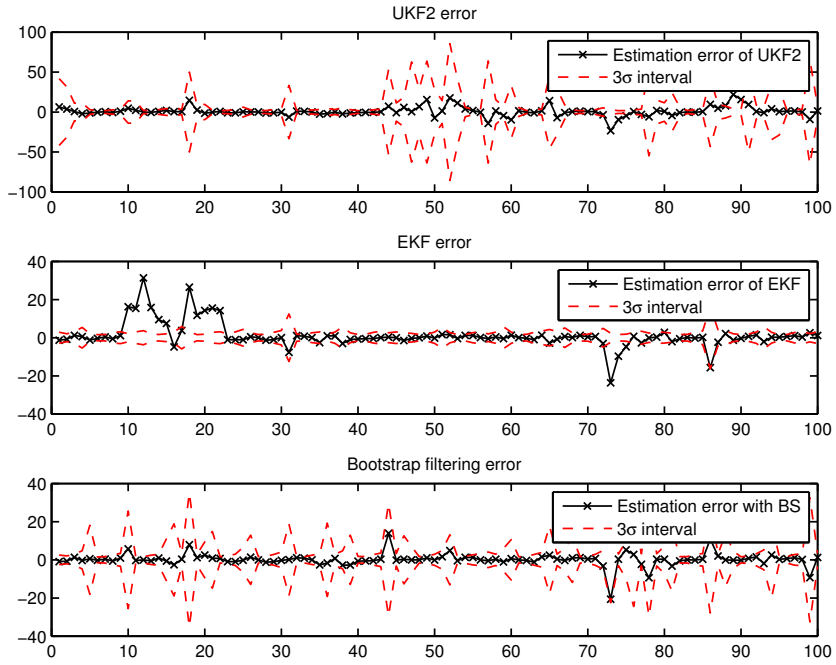


Figure 8: Absolute errors of and $3\sigma$ confidence intervals of EKF, augmented form UKF and bootstrap in 100 first samples.

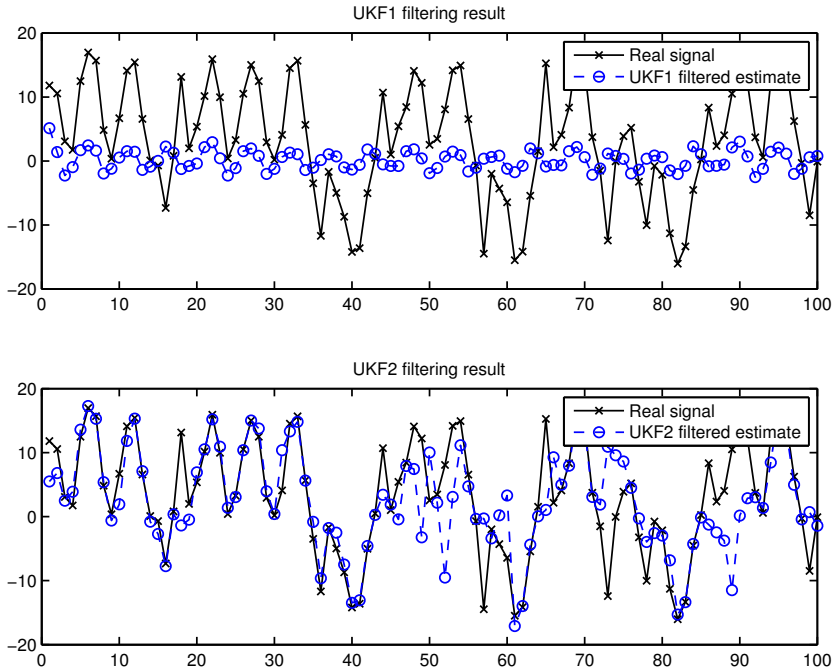


Figure 9: Filtering results of nonaugmented UKF (UKF1) and augmented UKF (UKF2) of 100 first samples.

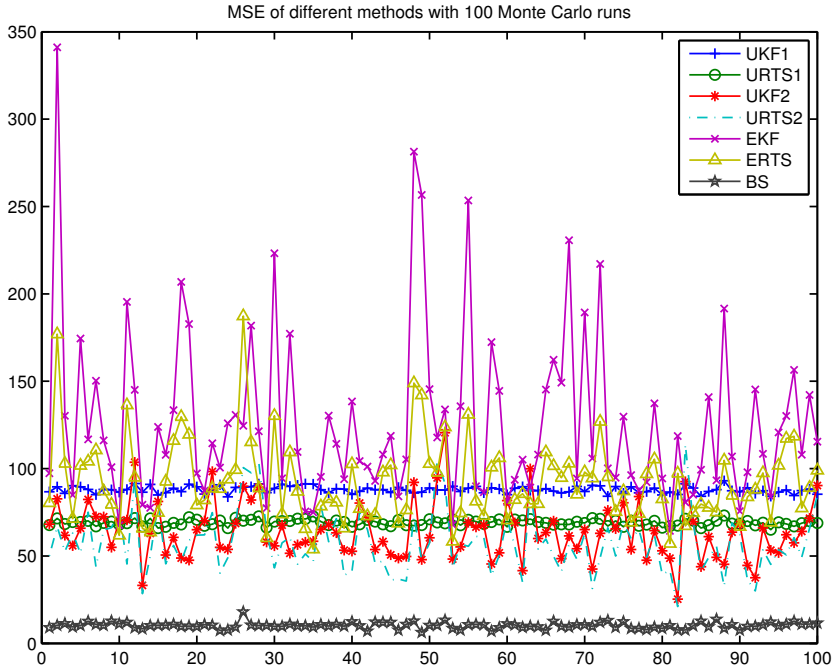


Figure 10: MSEs of different methods in 100 Monte Carlo runs.

discussed above, this is most likely due to the fact that the process noise terms are propagated through the nonlinearity, and hence odd-order moment information is used to obtain more accurate estimates. The usage of RTS smoother seemed to improve the estimates in general, but oddly in some cases it made the estimates worse. This is most likely due to the multi-modality of the filtering problem.

- Bootstrap filtering solution was clearly superior over all other tested methods. The results had been even better, if greater amount of particles had been used.

The reason why Kalman filters didn't work that well in this case is because Gaussian approximations do not in general apply well for multi-modal cases. Thus, a particle filtering solution should be preferred over Kalman filters in such cases. However, usually the particle filters need a fairly large amount of particles to be effective, so they are generally more demanding in terms of computational power than Kalman filters, which can be a limiting factor in real world applications. The errors, even with bootstrap filter, were also relatively large, so one must be careful when using the estimates in, for example, making financial decisions. In practice this means that one has to monitor the filter's covariance estimate, and trust the state estimates and predictions only when the covariance estimates are low enough, but even then there is a change, that the filter's estimate is completely wrong.

### 2.2.9 Demonstration: Bearings Only Tracking

Next we review a classical filtering application (see, e.g., Bar-Shalom et al., 2001), in which we track a moving object with sensors, which measure only the bearings (or angles) of the object with respect positions of the sensors. There is a one moving target in the scene and two angular sensors for tracking it. Solving this problem is important, because often more general multiple target tracking problems can be partitioned into sub-problems, in which single targets are tracked separately at a time (Särkkä, 2006b).

| *Method* | *MSE[x]* |
|---|---|
| UKF1 | 87.9 |
| URTS1 | 69.09 |
| UKF2 | 63.7 |
| URTS2 | 57.7 |
| EKF | 125.9 |
| ERTS | 92.2 |
| BS | 10.2 |

Table 2: MSEs of estimating the UNGM model over 100 Monte Carlo simulations.

The state of the target at time step $k$ consists of the position in two dimensional cartesian coordinates $x_k$ and $y_k$ and the velocity toward those coordinate axes, $\dot{x}_k$ and $\dot{y}_k$. Thus, the state vector can be expressed as

$$\mathbf{x}_k = \begin{pmatrix} x_k & y_k & \dot{x}_k & \dot{y}_k \end{pmatrix}^T. \tag{94}$$

The dynamics of the target is modelled as a linear, discretized Wiener velocity model (Bar-Shalom et al., 2001)

$$\mathbf{x}_k = \begin{pmatrix} 1 & 0 & \Delta\mathrm{t} & 0 \\ 0 & 1 & 0 & \Delta\mathrm{t} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{pmatrix} + \mathbf{q}_{k-1}, \tag{95}$$

where $\mathbf{q}_{k-1}$ is Gaussian process noise with zero mean and covariance

$$E[\mathbf{q}_{k-1}\mathbf{q}_{k-1}^T] = \begin{pmatrix} \frac{1}{3}\Delta\mathrm{t}^3 & 0 & \frac{1}{2}\Delta\mathrm{t}^2 & 0 \\ 0 & \frac{1}{3}\Delta\mathrm{t}^3 & 0 & \frac{1}{2}\Delta\mathrm{t}^2 \\ \frac{1}{2}\Delta\mathrm{t}^2 & 0 & \Delta\mathrm{t} & 0 \\ 0 & \frac{1}{2}\Delta\mathrm{t}^2 & 0 & \Delta\mathrm{t} \end{pmatrix} q, \tag{96}$$

where $q$ is the spectral density of the noise, which is set to $q = 0.1$ in the simulations. The measurement model for sensor $i$ is defined as

$$\theta_k^i = \arctan\left(\frac{y_k - s_y^i}{x_k - s_x^i}\right) + r_k^i, \tag{97}$$

where $(s_x^i, s_y^i)$ is the position of sensor $i$ and $r_k^i \sim N(0, \sigma^2)$, with $\sigma = 0.05$ radians. In figure 11 we have plotted a one realization of measurements in radians obtained from both sensors. The sensors are placed to $(s_x^1, s_y^1) = (-1, -2)$ and $(s_x^2, s_y^2) = (1, 1)$.

The derivatives of the measurement model, which are needed by EKF, can be computed as

$$\begin{aligned} \frac{\partial \mathbf{h}^i(\mathbf{x}_k)}{\partial x_k} &= \frac{-(y_k - s_y^i)}{(x_k - s_x^i)^2 + (y_k - s_y^i)^2} \\ \frac{\partial \mathbf{h}^i(\mathbf{x}_k)}{\partial y_k} &= \frac{(x_k - s_x^i)}{(x_k - s_x^i)^2 + (y_k - s_y^i)^2} \\ \frac{\partial \mathbf{h}^i(\mathbf{x}_k)}{\partial \dot{x}_k} &= 0 \\ \frac{\partial \mathbf{h}^i(\mathbf{x}_k)}{\partial \dot{y}_k} &= 0, i = 1, 2. \end{aligned} \tag{98}$$

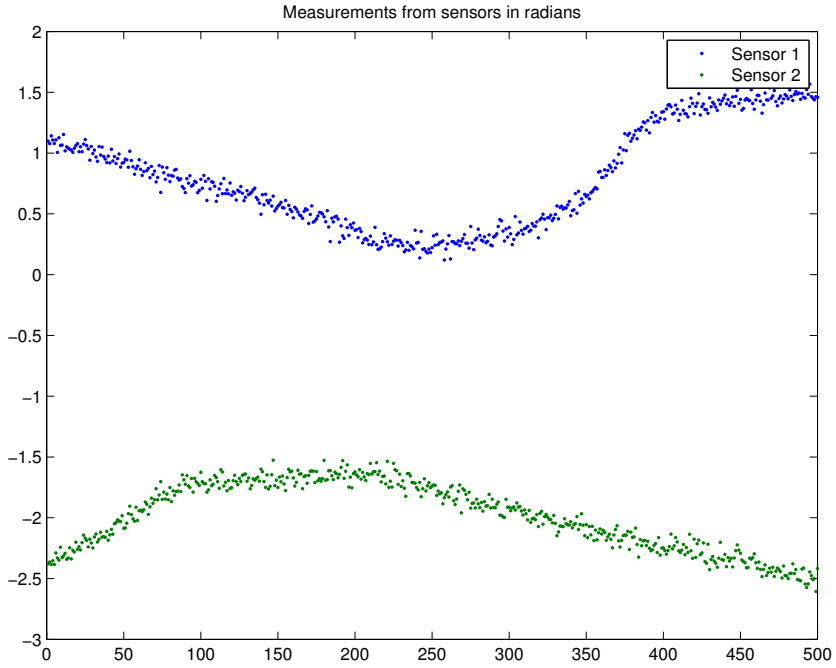


Figure 11: Measurements from sensors (in radians) in bearings only tracking problem .

With these the Jacobian can written as

$$\mathbf{H}_{\mathbf{x}}(\mathbf{x}_k, k) = \begin{pmatrix} \frac{(x_k - s_x^1)}{(x_k - s_x^1)^2 + (y_k - s_y^1)^2} & \frac{-(y_k - s_y^1)}{(x_k - s_x^1)^2 + (y_k - s_y^1)^2} & 0 & 0 \\ \frac{(x_k - s_x^2)}{(x_k - s_x^2)^2 + (y_k - s_y^2)^2} & \frac{-(y_k - s_y^2)}{(x_k - s_x^2)^2 + (y_k - s_y^2)^2} & 0 & 0 \end{pmatrix}. \tag{99}$$

The non-zero second order derivatives of the measurement function are also relatively easy to compute in this model:

$$\begin{aligned} \frac{\partial^2 \mathbf{h}^i(\mathbf{x}_k)}{\partial x_k \partial x_k} &= \frac{-2(x_k - s_x^i)}{((x_k - s_x^i)^2 + (y_k - s_y^i)^2)^2} \\ \frac{\partial^2 \mathbf{h}^i(\mathbf{x}_k)}{\partial x_k \partial y_k} &= \frac{(y_k - s_y^i)^2 - (x_k - s_x^i)^2}{((x_k - s_x^i)^2 + (y_k - s_y^i)^2)^2} \\ \frac{\partial^2 \mathbf{h}^i(\mathbf{x}_k)}{\partial y_k \partial y_k} &= \frac{-2(y_k - s_y^i)}{((x_k - s_x^i)^2 + (y_k - s_y^i)^2)^2}. \end{aligned} \tag{100}$$

Thus, the Hessian matrices can be written as

$$\mathbf{H}_{\mathbf{xx}}^i(\mathbf{x}_k, k) = \begin{pmatrix} \frac{-2(x_k - s_x^i)}{((x_k - s_x^i)^2 + (y_k - s_y^i)^2)^2} & \frac{(y_k - s_y^i)^2 - (x_k - s_x^i)^2}{((x_k - s_x^i)^2 + (y_k - s_y^i)^2)^2} & 0 & 0 \\ \frac{(y_k - s_y^i)^2 - (x_k - s_x^i)^2}{((x_k - s_x^i)^2 + (y_k - s_y^i)^2)^2} & \frac{-2(y_k - s_y^i)}{((x_k - s_x^i)^2 + (y_k - s_y^i)^2)^2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, i = 1, 2. \tag{101}$$

We do not list the program code for the measurement function and it's derivatives here as they are straightforward to implement, if the previous examples have been read.

| *Method* | *RMSE* |
|---|---|
| EKF1 | 0.114 |
| ERTS1 | 0.054 |
| ETF1 | 0.054 |
| EKF2 | 0.202 |
| ERTS2 | 0.074 |
| ETF2 | 0.063 |
| UKF | 0.113 |
| URTS | 0.055 |
| UTF | 0.055 |

Table 3: RMSEs of estimating the position in Bearings Only Tracking problem over 1000 Monte Carlo runs.

The target starts with state $\mathbf{x}_0 = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$, and in the estimation we set the prior distribution for the state to $\mathbf{x}_0 \sim N(\mathbf{0}, \mathbf{P}_0)$, where

$$\mathbf{P}_0 = \begin{pmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{pmatrix}, \tag{102}$$

which basically means that we are fairly certain about the target's origin, but very uncertain about the velocity. In the simulations we also give the target an slightly randomized acceleration, so that it achieves a curved trajectory, which is approximately the same in different simulations. The trajectory and estimates of it can be seen in figures 12, 13 and 14. As can be seen from the figures EKF1 and UKF give almost identical results while the estimates of EKF2 are clearly worse. Especially in the beginning of the trajectory EKF2 has great difficulties in getting on the right track, which is due to the relatively big uncertainty in the starting location. After that the estimates are fairly similar.

In table 3 we have listed the root mean square errors (mean of position errors) of all tested methods (same as in random sine signal example on page 20 with the addition of UTF) over 1000 Monte Carlo runs. The numbers prove the previous observations, that the EKF1 and UKF give almost identical performances. Same observations apply also to smoothers. Had the prior distribution for the starting velocity been more accurate the performance difference between EKF2 and other methods would have been smaller, but still noticable.

### 2.2.10 Demonstration: Reentry Vehicle Tracking

As a last example we review a challenging filtering problem, which was used in (Julier and Uhlmann, 2004b) to demonstrate the performance of UKF. Later they released few corrections to the model specifications and simulation parameters in (Julier and Uhlmann, 2004a).

This example conserns a reentry tracking problem, where radar is used for tracking a space vehicle, which enters the atmosphere at high altitude and very high speed. Figure 15 shows a sample trajectory of the vehicle with respect to earth and radar. The dynamics of the vehicle are affected with three kinds of forces: aerodynamic drag, which is a function of vehicle speed and has highly nonlinear variations in altitude. The second type of force is gravity, which causes the vehicle to accelerate toward the center of the earth. The third type of forces are random buffeting terms. The state space in this model consists of vehicles position ($x_1$ and $x_2$), its velocity ($x_3$ and $x_4$) and a parameter of its aerodynamic properties ($x_5$). The dynamics in continuous case are defined as

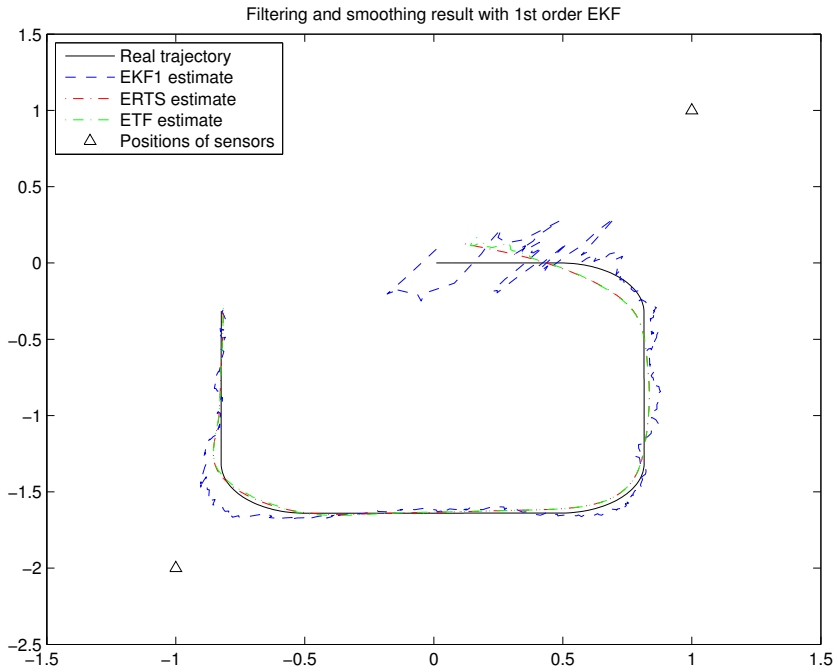


Figure 12: Filtering and smoothing results of first order EKF.

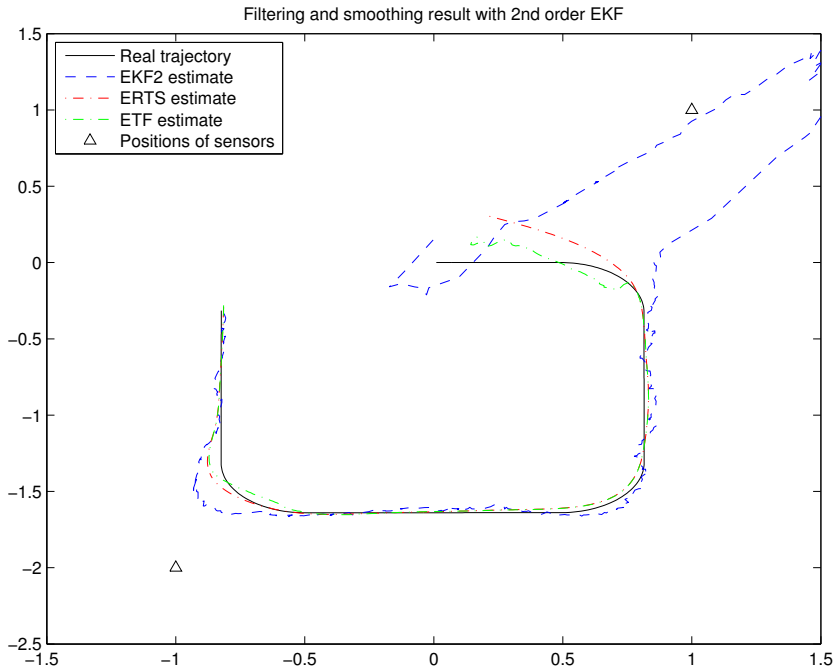


Figure 13: Filtering and smoothing results of second order EKF.

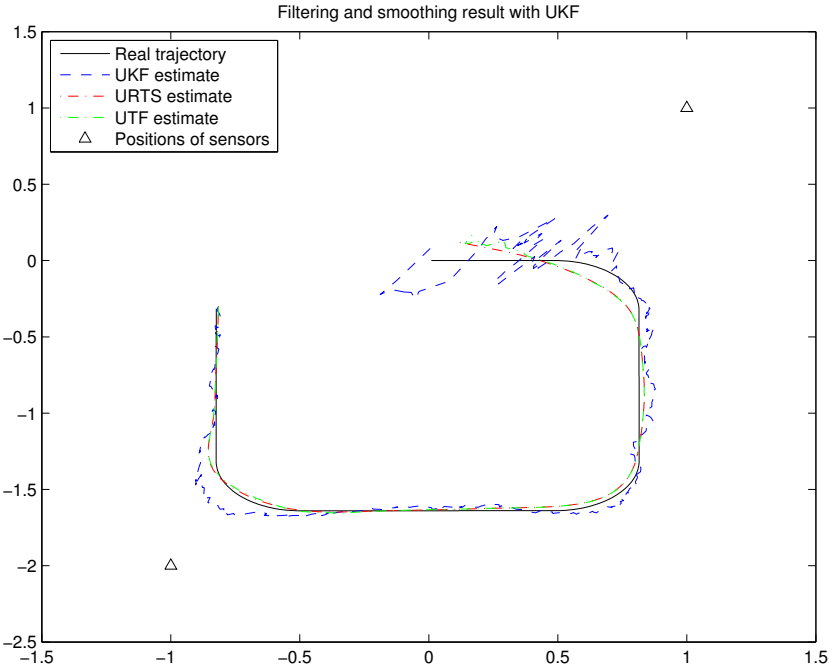


Figure 14: Filtering and smoothing results of UKF

(Julier and Uhlmann, 2004b)

$$\begin{aligned}
\dot{x}_1(t) &= x_3(t) \\
\dot{x}_2(t) &= x_4(t) \\
\dot{x}_3(t) &= D(t)x_3(t) + G(t)x_1(t) + v_1(t) \\
\dot{x}_4(t) &= D(t)x_4(t) + G(t)x_2(t) + v_2(t) \\
\dot{x}_5(t) &= v_3(t),
\end{aligned} \tag{103}$$

where $\mathbf{w}(t)$ is the process noise vector, $D(t)$ the drag-related force and $G(t)$ the gravity-related force. The force terms are given by

$$\begin{aligned}
D(k) &= \beta(t) \exp\left\{\frac{[R_0 - R(t)]}{H_0}\right\} V(t) \\
G(t) &= -\frac{Gm_0}{R^3(t)} \\
\beta(t) &= \beta_0 \exp x_5(t),
\end{aligned} \tag{104}$$

where $R(t) = \sqrt{x_1^2(t) + x_2^2(t)}$ is the vehicle's distance from the center of the earth and $V(t) = \sqrt{x_3^2(t) + x_4^2(t)}$ is the speed of the vehicle. The constants in previous definition were set to

$$\begin{aligned}
\beta_0 &= -0.59783 \\
H_0 &= 13.406 \\
Gm_0 &= 3.9860 \times 10^5 \\
R_0 &= 6374.
\end{aligned} \tag{105}$$

To keep the implementation simple the continuous-time dynamic equations were discretized using a simple Euler integration scheme, to give

$$\begin{aligned}
x_1(k+1) &= x_1(k) + \Delta\mathrm{t}\, x_3(k) \\
x_2(k+1) &= x_2(k) + \Delta\mathrm{t}\, x_4(k) \\
x_3(k+1) &= x_3(k) + \Delta\mathrm{t}(D(k)x_3(k) + G(k)x_1(k)) + w_1(k) \\
x_4(k+1) &= x_4(k) + \Delta\mathrm{t}(D(k)x_4(k) + G(k)x_2(k)) + w_2(k) \\
x_5(k+1) &= x_5(k) + w_3(k),
\end{aligned} \tag{106}$$

where the step size between time steps was set to $\Delta\mathrm{t} = 0.1s$. Note that this might be too simple approach in real world applications due to high nonlinearities in the dynamics, so more advanced integration scheme (such as Runge-Kutta) might be more preferable. The discrete process noise covariance in the simulations was set to

$$Q(k) = \begin{pmatrix} 2.4064 \times 10^{-5} & 0 & 0 \\ 0 & 2.4064 \times 10^{-5} & 0 \\ 0 & 0 & 10^{-6} \end{pmatrix}. \tag{107}$$

The lower right element in the covariance was initially in (Julier and Uhlmann, 2004b) set to zero, but later in (Julier and Uhlmann, 2004a) changed to $10^{-6}$ to increase filter stability.

The non-zero derivatives of the discretized dynamic equations with respect to state variables are

straightforward (although rather technical) to compute:

$$
\begin{aligned}
&\frac{\partial x_1(k+1)}{\partial x_1(k)} = 1 \\
&\frac{\partial x_1(k+1)}{\partial x_3(k)} = \Delta \mathrm{t} \\
&\frac{\partial x_2(k+1)}{\partial x_2(k)} = 1 \\
&\frac{\partial x_2(k+1)}{\partial x_4(k)} = \Delta \mathrm{t} \\
&\frac{\partial x_3(k+1)}{\partial x_1(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_1(k)} x_3(k) + \frac{\partial G(k)}{\partial x_1(k)} x_1(k) + G(k)) \\
&\frac{\partial x_3(k+1)}{\partial x_2(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_2(k)} x_3(k) + \frac{\partial G(k)}{\partial x_2(k)} x_1(k)) \\
&\frac{\partial x_3(k+1)}{\partial x_3(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_3(k)} x_3(k) + D(k)) + 1 \\
&\frac{\partial x_3(k+1)}{\partial x_4(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_4(k)} x_3(k)) \\
&\frac{\partial x_3(k+1)}{\partial x_4(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_5(k)} x_3(k)) \\
&\frac{\partial x_4(k+1)}{\partial x_1(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_1(k)} x_4(k) + \frac{\partial G(k)}{\partial x_1(k)} x_2(k)) \\
&\frac{\partial x_4(k+1)}{\partial x_2(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_2(k)} x_4(k) + \frac{\partial G(k)}{\partial x_2(k)} x_2(k) + G(k)) \\
&\frac{\partial x_4(k+1)}{\partial x_3(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_3(k)} x_4(k)) \\
&\frac{\partial x_4(k+1)}{\partial x_4(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_4(k)} x_4(k) + D(k)) + 1 \\
&\frac{\partial x_4(k+1)}{\partial x_5(k)} = \Delta \mathrm{t} * (\frac{\partial D(k)}{\partial x_5(k)} x_4(k)) \\
&\frac{\partial x_5(k+1)}{\partial x_5(k)} = 1,
\end{aligned}
\tag{108}
$$

where the (non-zero) derivatives of the force, position and velocity related terms with respect to state

variables can be computed as

$$
\begin{aligned}
\frac{\partial R(k)}{\partial x_1(k)} &= x_1(k)\frac{1}{R(k)} \\
\frac{\partial R(k)}{\partial x_2(k)} &= x_2(k)\frac{1}{R(k)} \\
\frac{\partial V(k)}{\partial x_3(k)} &= x_3(k)\frac{1}{V(k)} \\
\frac{\partial V(k)}{\partial x_4(k)} &= x_4(k)\frac{1}{V(k)} \\
\frac{\partial \beta(k)}{\partial x_5(k)} &= \beta(k)\frac{1}{R(k)} \\
\frac{\partial D(k)}{\partial x_1(k)} &= -\frac{\partial R(k)}{\partial x_1(k)}\frac{1}{H_0} * D(k) \\
\frac{\partial D(k)}{\partial x_2(k)} &= -\frac{\partial R(k)}{\partial x_2(k)}\frac{1}{H_0} * D(k) \\
\frac{\partial D(k)}{\partial x_3(k)} &= \beta(k)\exp\left\{\frac{[R_0 - R(k)]}{H_0}\right\}\frac{\partial V(k)}{\partial x_3} \\
\frac{\partial D(k)}{\partial x_4(k)} &= \beta(k)\exp\left\{\frac{[R_0 - R(k)]}{H_0}\right\}\frac{\partial V(k)}{\partial x_4} \\
\frac{\partial D(k)}{\partial x_5(k)} &= \frac{\partial \beta(k)}{x_5(k)}\exp\left\{\frac{[R_0 - R(k)]}{H_0}\right\}V(k) \\
\frac{\partial G(k)}{\partial x_1(k)} &= \frac{3Gm_0}{(R(k))^4}\frac{\partial R(k)}{\partial x_1(k)} \\
\frac{\partial G(k)}{\partial x_2(k)} &= \frac{3Gm_0}{(R(k))^4}\frac{\partial R(k)}{\partial x_2(k)}.
\end{aligned}
\tag{109}
$$

The prior distribution for the state was set to multivariate Gaussian, with mean and covariance (same as in (Julier and Uhlmann, 2004b))

$$
\begin{aligned}
\mathbf{m}_0 &= \begin{pmatrix} 6500.4 \\ 349.14 \\ -1.8093 \\ -6.7967 \\ 0 \end{pmatrix} \\
\mathbf{P}_0 &= \begin{pmatrix} 10^{-6} & 0 & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 & 0 \\ 0 & 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 0 & 10^{-6} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.
\end{aligned}
\tag{110}
$$

In the simulations the initial state were drawn from multivariate Gaussian with mean and covariance

$$\mathbf{m}_0 = \begin{pmatrix} 6500.4 \\ 349.14 \\ -1.8093 \\ -6.7967 \\ 0.6932 \end{pmatrix}$$
$$\mathbf{P}_0 = \begin{pmatrix} 10^{-6} & 0 & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 & 0 \\ 0 & 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 0 & 10^{-6} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{111}$$

that is, vehicle's aerodynamic properties were not known precisely beforehand.

The radar, which is located at $(s_x, s_y) = (R_0, 0)$, is used to measure the range $r_k$ and bearing $\theta_k$ in relation to the vehicle on time step $k$. Thus, the measurement model can be written as

$$\begin{aligned} r_k &= \sqrt{(x_1(k) - s_x)^2 + (x_2(k) - s_y)^2} + q_1(k) \\ \theta_k &= \tan^{-1}\left(\frac{x_2(k) - s_y}{x_1(k) - s_x}\right) + q_2(k), \end{aligned} \tag{112}$$

where the measurement noise processes $q_1(k)$ and $q_2(k)$ are Gaussians with zero means and standard deviations $\sigma_r = 10^{-3}$km and $\sigma_\theta = 0.17$mrad, respectively. The derivatives of $\theta_k$ with respect to state variables can computed with equations (98). For $r_k$ the derivatives can be written as

$$\begin{aligned} \frac{\partial r_k}{\partial x_1(k)} &= x_1(k)\frac{1}{r_k} \\ \frac{\partial r_k}{\partial x_2(k)} &= x_2(k)\frac{1}{r_k}. \end{aligned} \tag{113}$$

In the table 4 we have listed the RMS errors of position estimates with tested methods, which were

- EKF1: first order extended Kalman filter.
- ERTS: first order Rauch-Tung-Striebel smoother.
- UKF: augmented form unscented Kalman filter.
- URTS1: unscented Rauch-Tung-Striebel smoother with non-augmented sigma points.
- URTS2: unscented Rauch-Tung-Striebel smoother with augmented sigma points.
- UTF: unscented Forward-Backward smoother.

Extended Forward-Backward smoother was also tested, but it produced in many cases divergent estimates, so it was left out from comparison. Second order EKF was also left out, because evaluating the Hessians would have taken too much work while considering the fact, that the estimates might have gotten even worse.

From the error estimates we can see, that EKF and UKF give almost identical performances, altough in the article (Julier and Uhlmann, 2004b) UKF was clearly superior over EKF. The reason for this might be the fact that they used numerical approximations (central differences) for calculating the Jacobian in EKF rather than calculating the derivatives in closed form, as was done in this demonstration.

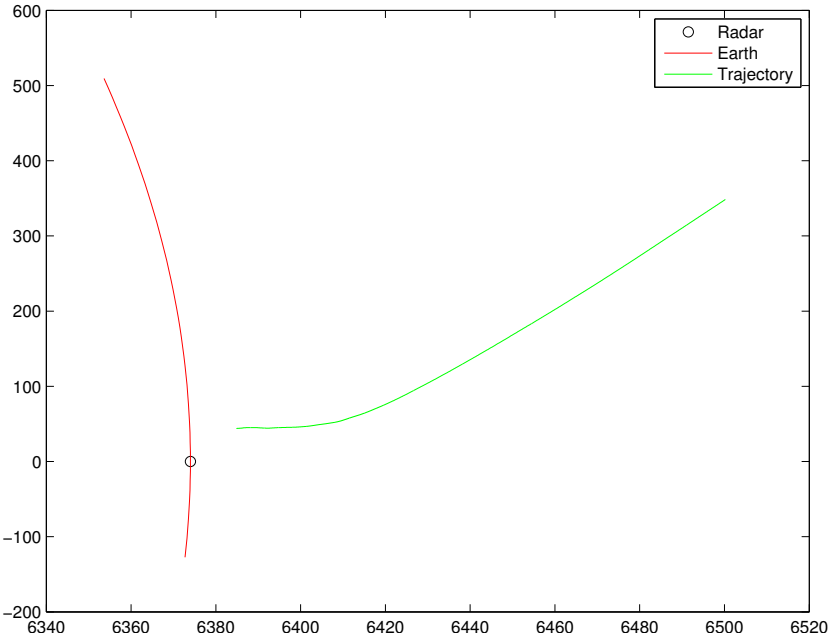


Figure 15: Sample vehicle trajectory, earth and position of radar in Reentry Vehicle Tracking problem.

In figure 16 we have plotted the mean square errors and variances in estimating $x_1$, $x_3$ and $x_5$ with EKF and ERTS over 100 Monte Carlo runs. It shows that using smoother always gives better estimates for positions and velocities, but for $x_5$ the errors are practically the same after $\sim 45$ seconds. This also shows that both methods are pessimistic in estimating $x_5$, because variances were bigger than the true errors. Figures for $x_2$ and $x_4$ are not shown, because they are very similar to the ones of $x_1$ and $x_3$. Also by using UKF and URTS the resulting figures were in practically identical, and therefore left out.

| *Method* | *RMSE* |
|---|---|
| EKF1 | 0.0083 |
| ERTS | 0.0043 |
| UKF | 0.0083 |
| URTS1 | 0.0043 |
| URTS2 | 0.0043 |
| UTF | 0.0044 |

Table 4: Average RMSEs of estimating the position in Reentry Vehicle Tracking problem over 100 Monte Carlo runs.

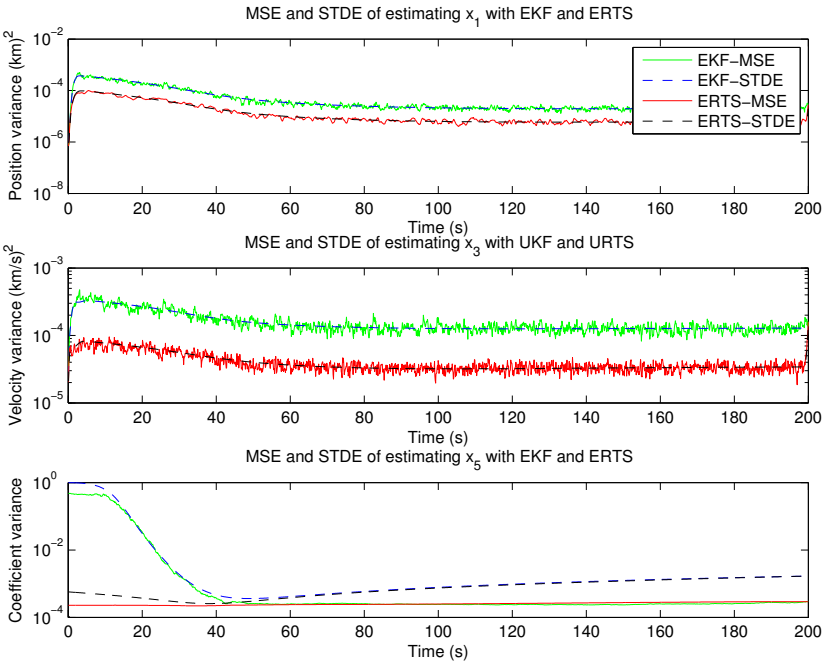


Figure 16: MSEs and variances in estimating of $x_1$, $x_3$ and $x_5$ using EKF and ERTS over 100 Monte Carlo runs.

| Function | Page |
|---|---|
| lti_disc | 44 |
| kf_predict | 44 |
| kf_update | 44 |
| rts_smooth | 45 |
| tf_smooth | 45 |
| ekf_predict1 | 45 |
| ekf_update1 | 45 |
| ekf_predict2 | 46 |
| ekf_update2 | 46 |
| erts_smooth1 | 47 |
| etf_smooth1 | 48 |
| ut_weights | 49 |
| ut_mweights | 49 |
| ut_sigmas | 49 |
| ut_transform | 49 |
| ukf_predict1 | 49 |
| ukf_update1 | 50 |
| ukf_predict2 | 50 |
| ukf_update2 | 50 |
| ukf_predict3 | 51 |
| ukf_update3 | 50 |
| urts_smooth1 | 52 |
| urts_smooth2 | 53 |
| utf_smooth1 | 53 |
| der_check | 54 |
| gauss_pdf | 54 |
| gauss_rnd | 54 |
| rk4 | 55 |
| schol | 55 |

Table 5: Functions provided with the toolbox. Functions are sorted according to page numbers, from which function descriptions begin.

# 3 Functions in the toolbox

In table 5 we have listed all the functions in the toolbox and the page numbers (sorted by page numbers in increasing order), where the function description begins. Some of the functions have optional input parameters. The default values for these parameters are listed in end of parameter descriptions.

## 3.1 Discrete-time state space estimation

| **lti_disc**<br>Discretizes a linear, continous-time dynamic system using the matrix fraction decomposition (eq. (17)). | | | |
|---|---|---|---|
| **Syntax:** `[A,Q] = lti_disc(F,L,Qc,dt)` | | | |
| **Input:** | `F` | $N \times N$ feedback matrix | |
| | `L` | $N \times L$ noise effect matrix | default: identity |
| | `Qc` | $L \times L$ diagonal diffusion matrix | default: zeros |
| | `dt` | time step | default: 1 |
| **Output:** | `A` | The transition matrix | |
| | `Q` | The covariance of the discrete process | |

### 3.1.1 Linear models

#### Kalman filter

| **kf_predict**<br>Calculates the **prediction step** of the Kalman filter (eq. (18)). | | | |
|---|---|---|---|
| **Syntax:** `[X,P] = kf_predict(X,P,A,Q,B,U)` | | | |
| **Input:** | `X` | $N \times 1$ vector containing the state mean estimate of the previous step | |
| | `P` | $N \times N$ matrix containing the state covariance of the previous step | |
| | `A` | Transition matrix of the discrete model | default: identity |
| | `Q` | Process noise of the discrete model | default: zero |
| | `B` | Input effect matrix | default: identity |
| | `U` | Constant input | default: empty |
| **Output:** | `X` | Predicted mean for the state on the next time step | |
| | `P` | Predicted covariance for the state on the next time step | |

| **kf_update**<br>Calculates the **update step** of the Kalman filter (eq. 19). | | |
|---|---|---|
| **Syntax:** `[X,P,K,IM,IS,LH] = kf_update(X,P,Y,H,R)` | | |
| **Input:** | `X` | $N \times 1$ vector containing the predicted state mean estimate from the prediction step |
| | `P` | $N \times N$ matrix containing the predicted state covariance of the prediction step |
| | `Y` | $D \times 1$ vector containing the measurement on this time step |
| | `H` | $N \times D$ measurement matrix on this time step |
| | `R` | $S \times D$ measurement noise covariance matrix on this time step |
| **Output:** | `X` | $N \times 1$ vector containing the updated state mean |
| | `P` | $N \times N$ matrix containing the updated state covariance |
| | `K` | Computed gain for the Kalman filter |
| | `IM` | Mean of predictive distribution of `Y` |
| | `IS` | Covariance or predictive mean of `Y` |
| | `LH` | Predictive probability of the measurement |

**Kalman smoother**

**`rts_smooth`**
Computes **The smoothed estimate** for the state mean and covariance using **the RTS smoother** (eq. (21)).

**Syntax:** `[M,P,S] = rts_smooth(M,P,A,Q)`

| | | |
|---|---|---|
| **Input:** | `M` | $N \times K$ matrix containing $K$ mean estimates calculated with Kalman filter |
| | `P` | $N \times N \times K$ matrix of $K$ state covariances calculated with Kalman filter |
| | `A` | $N \times N$ state transition matrix or $N \times N \times K$ matrix of $K$ state covariances for each time step |
| | `Q` | $N \times N$ process noise covariance matrix or $N \times N \times K$ matrix of $K$ process noise covariance matrices for each time step |
| **Output:** | `M` | Smoothed state mean on each time step |
| | `P` | Smoothed state covariance sequence on each time step |
| | `S` | Smoother gain on each time step |

**`tf_smooth`**
Calculates the **The smoothed estimate** for the state mean and covariance using **the TF smoother**.

**Syntax:** `[M,P] = tf_smooth(M,P,Y,A,Q,H,R,use_inf)`

| | | |
|---|---|---|
| **Input:** | `M` | $N \times K$ matrix containing $K$ mean estimates calculated with Kalman filter |
| | `P` | $N \times N \times K$ matrix of $K$ state covariances calculated with Kalman filter |
| | `Y` | $D \times K$ matrix containing the sequence of $K$ measurements |
| | `A` | $N \times N$ state transition matrix or $N \times N \times K$ matrix of $K$ state covariances for each time step |
| | `Q` | $N \times N$ process noise covariance matrix or $N \times N \times K$ matrix of $K$ process noise covariance matrices for each time step |
| | `H` | $D \times N$ measurement matrix |
| | `R` | $D \times D$ measurement noise covariance |
| | `use_inf` | if information filter should be used |
| **Output:** | `M` | Smoothed state mean on each time step |
| | `P` | Smoothed state covariance sequence on each time step |

### 3.1.2 Nonlinear models

**Extended Kalman filter**

**`ekf_predict1`**
Calculates the **The prediction step** of the **first order extended Kalman filter** (eq. (38)).

**Syntax:** `[M,P] = ekf_predict1(M,P,[A,Q,a,W,param])`

| | | | |
|---|---|---|---|
| **Input:** | `M` | $N \times 1$ vector containing the state mean estimate of the previous step | |
| | `P` | $N \times N$ matrix containing the state covariance of the previous step | |
| | `A` | Derivative of `a` with respect to state as matrix, inline function, function handle or name of the function in form `A(x,param)` | default: identity |
| | `Q` | Process noise of the discrete model | default: zero |
| | `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a(x,param)` | default: `A(x)*M` |
| | `W` | Derivative of `a` with respect to noise as matrix, inline function, function handle or name of the function in form `W(x,param)` | default: identity |
| | `param` | Parameters of function a | default: empty |
| **Output:** | `M` | Predicted mean for the state on the next time step | |
| | `P` | Predicted covariance for the state on the next time step | |

**ekf_update1**

Calculates the **The update step** of the **first order extended Kalman filter** (eq. (39)).

**Syntax:** `[M,P,K,IM,S,LH] = ekf_update1(M,P,Y,H,R,h,V,[param])`

**Input:**

| | | |
|---|---|---|
| `M` | $N \times 1$ vector containing the state mean estimate after the prediction step | |
| `P` | $N \times N$ matrix containing the state covariance after the prediction step | |
| `Y` | $D \times 1$ measurement vector | |
| `H` | Derivative of `h` with respect to state as matrix, inline function, function handle or name of the function in form `H(x,param)` | |
| `R` | Measurement noise covariance | |
| `h` | Measurement model function as vector, inline function, function handle or name of the function in form `h(x,param)` | default: `H(x)*M` |
| `V` | Derivative of `h` with respect to noise as matrix, inline function, function handle or name of the function in form `V(x,param)` | default: identity |
| `param` | Parameters of function h | default: empty |

**Output:**

| | |
|---|---|
| `M` | Updated mean for the state on the next time step |
| `P` | Updated covariance for the state on the next time step |
| `K` | Computed Kalman gain |
| `MU` | Predictive mean of Y |
| `S` | Predictive covariance of Y |
| `LH` | Predictive probability (likelihood) of measurement |

**ekf_predict2**

Calculates the **The prediction step** of the **second order extended Kalman filter** (eq. (42)).

**Syntax:** `[M,P] = ekf_predict2(M,P,[A,F,Q,a,W,param])`

**Input:**

| | | |
|---|---|---|
| `M` | $N \times 1$ vector containing the state mean estimate of the previous step | |
| `P` | $N \times N$ matrix containing the state covariance of the previous step | |
| `A` | Derivative of `a` with respect to state as matrix, inline function, function handle or name of the function in form `A(x,param)` | default: identity |
| `F` | $N \times N \times N$ Hessian matrix of the state transition w.r.t. state variables as matrix, function handle of name of function in form `F(x,param)` | default: identity |
| `Q` | Process noise of the discrete model | default: zero |
| `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a(x,param)` | default: $A(x) * M$ |
| `W` | Derivative of `a` with respect to noise as matrix, inline function, function handle or name of the function in form `W(x,param)` | default: identity |
| `param` | Parameters of function a | default: empty |

**Output:**

| | |
|---|---|
| `M` | Predicted mean for the state on the next time step |
| `P` | Predicted covariance for the state on the next time step |

| **ekf_update2**<br>Calculates the **The update step** of the **second order extended Kalman filter** (eq. (43)). | | | |
|---|---|---|---|
| **Syntax:** `[M,P,K,IM,S,LH] = ekf_update2(M,P,Y,H,H_xx,R,h,V,[param])` | | | |
| **Input:** | `M` | $N \times 1$ vector containing the state mean estimate after the prediction step | |
| | `P` | $N \times N$ matrix containing the state covariance after the prediction step | |
| | `Y` | $D \times 1$ measurement vector | |
| | `H` | Derivative of `h` with respect to state as matrix, inline function, function handle or name of the function in form `H(x,param)` | |
| | `H_xx` | $D \times N \times N$ Hessian of `h` w.r.t. state variables as matrix, inline function, function handle or name of function in form `H_xx(x,param)` | |
| | `R` | Measurement noise covariance | |
| | `h` | Measurement model function as vector, inline function, function handle or name of the function in form `h(x,param)` | default: `H(x)*M` |
| | `V` | Derivative of `h` with respect to noise as matrix, inline function, function handle or name of the function in form `V(x,param)` | default: identity |
| | `param` | Parameters of function h | default: empty |
| **Output:** | `M` | Updated mean for the state on the next time step | |
| | `P` | Updated covariance for the state on the next time step | |
| | `K` | Computed Kalman gain | |
| | `MU` | Predictive mean of Y | |
| | `S` | Predictive covariance of Y | |
| | `LH` | Predictive probability (likelihood) of measurement | |

**Extended Kalman smoothers**

In extended and unscented smoothers the parameters of dynamic and measurement function are passed as a single cell array, vector or a matrix containing the same parameters for each step, or if different parameters are used on each step they must be a cell array of the format `{param_1,param_2,...}`, where `param_x` contains the parameters for step x as a cell array, vector or matrix.

**erts_smooth1**

Calculates **The smoothed estimate** for the state mean and covariance using **the extended RTS smoother** (eq. (46)).

**Syntax:** `[M,P,D] = erts_smooth1(M,P,[A,Q,s,W,param,same_p])`

| | | | |
|---|---|---|---|
| **Input:** | `M` | $N \times K$ matrix containing the state mean estimates produced by the EKF | |
| | `P` | $N \times N \times K$ matrix containing the state covariances produced by the EKF | |
| | `A` | Derivative of `a` with respect to state as matrix, inline function, function handle or name of the function in form `A(x,param)` | default: zero |
| | `Q` | Process noise of discrete model | default: zero |
| | `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a(x,param)` | default: `H(x)*M` |
| | `W` | Derivative of `a` with respect to noise as matrix, inline function, function handle or name of the function in form `V(x,param)` | default: identity |
| | `param` | Parameters of function `h` | default: empty |
| | `same_p` | If 1 uses the same parameters on every time step | default: 1 |
| **Output:** | `M` | Smoothed state mean sequence for all time steps | |
| | `P` | Smoothed state covariance for all time steps | |
| | `D` | Smoother gain for all time steps | |

**etf_smooth1**

Calculates **The smoothed estimate** for the state mean and covariance using **the extended TF smoother**.

**Syntax:** `[M,P] = etf_smooth1(M,P,Y,A,Q,ia,W,aparam,H,R,h,V,hparam,s_p_a,s_p_h)`

| | | | |
|---|---|---|---|
| **Input:** | `M` | $N \times K$ matrix containing the state mean estimates produced by the EKF | |
| | `P` | $N \times N \times K$ matrix containing the state covariances produced by the EKF | |
| | `Y` | Measurement vector | |
| | `A` | Derivative of `ia` with respect to state as matrix, inline function, function handle or name of the function in form `A(x,param)` | default: identity |
| | `Q` | Process noise of discrete model | default: zero |
| | `ia` | Inverse prediction function as vector, inline function, function handle or name of function in form `ia(x,param)` | default: `inv(A(x))*M` |
| | `W` | Derivative of `ia` with respect to noise as matrix, inline function, function handle or name of the function in form `V(x,param)` | default: identity |
| | `aparam` | Parameters of function `ia` | default: empty |
| | `H` | Derivative of `h` with respect to state as matrix, inline function, function handle or name of function in form `H(x,param)` | |
| | `R` | Measurement noise covariance | |
| | `h` | Measurement model function as vector, inline function, function handle or name of the function in form `h(x,param)` | default: `H(x)*M` |
| | `V` | Derivative of `h` w.r.t. noise as matrix, inline function function handle or name of function in form `V(x,param)` | default: identity |
| | `hparam` | Parameters of h | default: `aparam` |
| | `s_p_a` | If 1 uses the same parameters on every time step for `a` | default: 1 |
| | `s_p_h` | If 1 uses the same parameters on every time step for `h` | default: 1 |
| **Output:** | `M` | Smoothed state mean sequence for all time steps | |
| | `P` | Smoothed state covariance for all time steps | |

**Unscented Kalman Filter**

| **ut_weights** | | | |
|---|---|---|---|
| Calculates the **weights** for the **unscented transform** using the **summation** form (eq. (63)). | | | |
| **Syntax:** `[WM,WC,c] = ut_weights(n,alpha,beta,kappa)` | | | |
| **Input:** | `n` | Dimensionality of random variable, | |
| | `alpha` | Transformation parameter $\alpha$ | default: 0.5 |
| | `beta` | Transformation parameter $\beta$ | default: 2 |
| | `kappa` | Transformation parameter $\kappa$ | default: 3-N |
| **Output:** | `WM` | $(2n+1) \times 1$ vector containing the weights for mean estimation | |
| | `WC` | $(2n+1) \times 1$ vector containing the weights for covariance estimation | |
| | `c` | Scaling constant | |

| **ut_mweights** | | | |
|---|---|---|---|
| Calculates the **weights** for the **unscented transform** using the **matrix** form (eq. (75) and (76)). | | | |
| **Syntax:** `[WM,W,c] = ut_mweights(n,alpha,beta,kappa)` | | | |
| **Input:** | `n` | Dimensionality of the state variable | |
| | `alpha` | Transformation parameter $\alpha$ | default: 0.5 |
| | `beta` | Transformation parameter $\beta$ | default: 2 |
| | `kappa` | Transformation parameter $\kappa$ | default: 3-N |
| **Output:** | `WM` | $(2n+1) \times 1$ vector containing the weights for mean estimation | |
| | `W` | $(2n+1) \times (2n+1)$ matrix containing the weights for covariance estimation | |
| | `c` | Scaling constant | |

| **ut_sigmas** | | |
|---|---|---|
| Generates The **sigma points** for the **unscented transform** with eq. (62). | | |
| **Syntax:** `[X] = ut_sigmas(M,P,c)` | | |
| **Input:** | `M` | $N \times 1$ vector containing the initial state mean |
| | `P` | $N \times N$ matrix containing the initial state covariance |
| | `c` | Parameter returned by `ut_weights` |
| **Output:** | `X` | $(2n+1) \times (2n+1)$ matrix containing the sigma points in its columns |

| **ut_transform** | | | |
|---|---|---|---|
| Calculates the **unscented transform** of function **g** using the summation or matrix form. | | | |
| **Syntax:** `[mu,S,C,X,Y,w] = ut_transform(g,M,P,[param,alpha,beta,kappa,mat])` | | | |
| **Input:** | `g` | Transformation function of the form `g(x,param)` as matrix, inline function, function name or function reference | |
| | `M` | $N \times 1$ containing mean of $\mathbf{x}$ | |
| | `P` | $N \times N$ matrix containing the covariance of $\mathbf{x}$ | |
| | `param` | Parameters of function `g` | default: empty |
| | `alpha` | Transformation parameter $\alpha$ | default: 0.5 |
| | `beta` | Transformation parameter $\beta$ | default: 2 |
| | `kappa` | Transformation parameter $\kappa$ | default: 3-N |
| | `mat` | 1 if the matrix form is used, otherwise 0 | default: 0 |
| **Output:** | `mu` | Estimated mean of y | |
| | `S` | Estimated covariance of y | |
| | `C` | Estimated cross-covariance of x and y | |
| | `X` | Sigma points of x | |
| | `Y` | Sigma points of y | |
| | `w` | Weights of sigma points as cell-array of form `{mean-weights,cov-weights,c}` | |

**ukf_predict1**

Calculates the **The prediction step** of the **non-augmented form unscented Kalman filter** (eq. (78)).

**Syntax:** `[M,P] = ukf_predict1(M,P,a,Q,[param,alpha,beta,kappa])`

| | | | |
|---|---|---|---|
| **Input:** | `M` | $N \times 1$ vector containing the state mean estimate of the previous step | |
| | `P` | $N \times N$ matrix containing the state covariance of the previous step | |
| | `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a(x,param)` | |
| | `Q` | Process noise of the discrete model | default: zero |
| | `param` | Parameters of function a | default: empty |
| | `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| | `beta` | Transformation parameter $\beta$ | default: $2$ |
| | `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| | `mat` | If 1 uses matrix form | default: $0$ |
| **Output:** | `M` | Predicted mean for the state on the next time step | |
| | `P` | Predicted covariance for the state on the next time step | |

**ukf_update1**

Calculates the **The update step** of the **nonaugmented form unscented Kalman filter** (eq. (79) and (80)).

**Syntax:** `[M,P,K,IM,S,LH] = ukf_update1(M,P,Y,h,R,[param,alpha,beta,kappa,mat])`

| | | | |
|---|---|---|---|
| **Input:** | `M` | $N \times 1$ vector containing the state mean estimate after the prediction step | |
| | `P` | $N \times N$ matrix containing the state covariance after the prediction step | |
| | `Y` | $D \times 1$ measurement vector | |
| | `h` | Measurement model function as vector, inline function, function handle or name of the function in form `h(x,param)` | |
| | `R` | Measurement noise covariance | |
| | `param` | Parameters of function h | default: empty |
| | `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| | `beta` | Transformation parameter $\beta$ | default: $2$ |
| | `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| | `mat` | If 1 uses matrix form | default: $0$ |
| **Output:** | `M` | Updated mean for the state on the next time step | |
| | `P` | Updated covariance for the state on the next time step | |
| | `K` | Computed Kalman gain | |
| | `MU` | Predictive mean of Y | |
| | `S` | Predictive covariance of Y | |
| | `LH` | Predictive probability (likelihood) of measurement | |

**ukf_predict2**

Calculates the **The prediction step** of the **augmented form unscented Kalman filter** which concatenates state variables and process noise terms together.

**Syntax:** `[M,P] = ukf_predict2(M,P,a,Q,[param,alpha,beta,kappa])`

**Input:**

| | | |
|---|---|---|
| `M` | $N \times 1$ vector containing the state mean estimate of the previous step | |
| `P` | $N \times N$ matrix containing the state covariance of the previous step | |
| `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a([x;q],param)` | |
| `Q` | Process noise of the discrete model | default: zero |
| `param` | Parameters of function a | default: empty |
| `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| `beta` | Transformation parameter $\beta$ | default: $2$ |
| `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| `mat` | If 1 uses matrix form | default: $0$ |

**Output:**

| | |
|---|---|
| `M` | Predicted mean for the state on the next time step |
| `P` | Predicted covariance for the state on the next time step |

**ukf_update2**

Calculates the **The update step** of the **augmented form unscented Kalman filter**, which concatenates state variables and measurement noise terms together. Note that this function creates a new set of sigma points, so some odd-order moment information carried by process noise terms might get lost.

**Syntax:** `[M,P,K,IM,S,LH] = ukf_update2(M,P,Y,h,R,[param,alpha,beta,kappa,mat])`

**Input:**

| | | |
|---|---|---|
| `M` | $N \times 1$ vector containing the state mean estimate after the prediction step | |
| `P` | $N \times N$ matrix containing the state covariance after the prediction step | |
| `Y` | $D \times 1$ measurement vector | |
| `h` | Measurement model function as vector, inline function, function handle or name of the function in form `h([x;r],param)` | |
| `R` | Measurement noise covariance | |
| `param` | Parameters of function h | default: empty |
| `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| `beta` | Transformation parameter $\beta$ | default: $2$ |
| `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| `mat` | If 1 uses matrix form | default: $0$ |

**Output:**

| | |
|---|---|
| `M` | Updated mean for the state on the next time step |
| `P` | Updated covariance for the state on the next time step |
| `K` | Computed Kalman gain |
| `MU` | Predictive mean of Y |
| `S` | Predictive covariance of Y |
| `LH` | Predictive probability (likelihood) of measurement |

**ukf_predict3**

Calculates the **The prediction step** of the **augmented form unscented Kalman filter** (eq. (81),(82) and (83)), which concatenates state variables, process and measurement noise terms together.

**Syntax:** `[M,P] = ukf_predict3(M,P,a,Q,[param,alpha,beta,kappa])`

**Input:**

| | | |
|---|---|---|
| `M` | $N \times 1$ vector containing the state mean estimate of the previous step | |
| `P` | $N \times N$ matrix containing the state covariance of the previous step | |
| `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a([x;q],param)` | |
| `Q` | Process noise of the discrete model | default: zero |
| `param` | Parameters of function a | default: empty |
| `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| `beta` | Transformation parameter $\beta$ | default: $2$ |
| `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| `mat` | If 1 uses matrix form | default: $0$ |

**Output:**

| | |
|---|---|
| `M` | Predicted mean for the state on the next time step |
| `P` | Predicted covariance for the state on the next time step |
| `X` | Sigma points of x |
| `w` | Weights as cell array of form `{mean-weights,cov-weights,c}` |

**ukf_update3**

Calculates the **The update step** of the **augmented form unscented Kalman filter** (eq. (84) and (85)), which concatenates state variables and measurement noise terms together. Note that this function uses the sigma points created in the prediction step, so that odd-order moment information could get captured better.

**Syntax:** `[M,P,K,IM,S,LH] = ukf_update3(M,P,Y,h,R,X,w,[param,alpha,beta,kappa,mat])`

**Input:**

| | | |
|---|---|---|
| `M` | $N \times 1$ vector containing the state mean estimate after the prediction step | |
| `P` | $N \times N$ matrix containing the state covariance after the prediction step | |
| `Y` | $D \times 1$ measurement vector | |
| `h` | Measurement model function as vector, inline function, function handle or name of the function in form `h([x;r],param)` | |
| `R` | Measurement noise covariance | |
| `X` | Sigma points of x | |
| `w` | Weights as cell array of form `{mean-weights,cov-weights,c}` | |
| `param` | Parameters of function h | default: empty |
| `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| `beta` | Transformation parameter $\beta$ | default: $2$ |
| `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| `mat` | If 1 uses matrix form | default: $0$ |

**Output:**

| | |
|---|---|
| `M` | Updated mean for the state on the next time step |
| `P` | Updated covariance for the state on the next time step |
| `K` | Computed Kalman gain |
| `MU` | Predictive mean of Y |
| `S` | Predictive covariance of Y |
| `LH` | Predictive probability (likelihood) of measurement |

**urts_smooth1**

Calculates **The smoothed estimate** for the state mean and covariance using **the nonaugmented form unscented RTS smoother**. Note that this assumes that the noises are additive.

**Syntax:** `[M,P,D] = urts_smooth1(M,P,a,Q,[param,alpha,beta,kappa,mat,same_p])`

| | | | |
|---|---|---|---|
| **Input:** | `M` | $N \times K$ matrix containing the state mean estimates produced by the EKF | |
| | `P` | $N \times N \times K$ matrix containing the state covariances produced by the EKF | |
| | `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a(x,param)` | |
| | `Q` | Process noise of discrete model | default: zero |
| | `param` | Parameters of function `a` | default: empty |
| | `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| | `beta` | Transformation parameter $\beta$ | default: $2$ |
| | `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| | `mat` | If 1 uses matrix form | default: $0$ |
| | `same_p` | If 1 uses the same parameters on every time step | default: $1$ |
| **Output:** | `M` | Smoothed state mean sequence for all time steps | |
| | `P` | Smoothed state covariance for all time steps | |
| | `D` | Smoother gain for all time steps | |

**urts_smooth2**

Calculates **The smoothed estimate** for the state mean and covariance using **the augmented form unscented RTS smoother** (eq. (87) - (91)). Note that in this case noises can be non-additive.

**Syntax:** `[M,P,D] = urts_smooth2(M,P,a,Q,[param,alpha,beta,kappa,mat,same_p])`

| | | | |
|---|---|---|---|
| **Input:** | `M` | $N \times K$ matrix containing the state mean estimates produced by the EKF | |
| | `P` | $N \times N \times K$ matrix containing the state covariances produced by the EKF | |
| | `a` | Dynamic model function as vector, inline function, function handle or name of the function in form `a([x;w],param)` | |
| | `Q` | Process noise of discrete model | default: zero |
| | `param` | Parameters of function `a` | default: empty |
| | `alpha` | Transformation parameter $\alpha$ | default: $0.5$ |
| | `beta` | Transformation parameter $\beta$ | default: $2$ |
| | `kappa` | Transformation parameter $\kappa$ | default: $3 - N$ |
| | `mat` | If 1 uses matrix form | default: $0$ |
| | `same_p` | If 1 uses the same parameters on every time step | default: $1$ |
| **Output:** | `M` | Smoothed state mean sequence for all time steps | |
| | `P` | Smoothed state covariance for all time steps | |
| | `D` | Smoother gain for all time steps | |

| **utf_smooth1** | | | |
|---|---|---|---|
| Calculates **The smoothed estimate** for the state mean and covariance using **the unscented TF smoother**. | | | |
| **Syntax:** `[M,P] = utf_smooth1(M,P,Y,[ia,Q,ap,h,R,hp,aa,bb,kk,mat,s_p_a,s_p_h])` | | | |
| **Input:** | `M` | $N \times K$ matrix containing the state mean estimates produced by the EKF | |
| | `P` | $N \times N \times K$ matrix containing the state covariances produced by the EKF | |
| | `Y` | Measurement vector | |
| | `ia` | Inverse prediction function as vector, inline function, function handle or name of function in form `ia(x,param)` | default: identity |
| | `Q` | Process noise of discrete model | default: zero |
| | `ap` | Parameters of function `ia` | default: empty |
| | `h` | Measurement model function as vector, inline function, function handle or name of the function in form `h(x,param)` | |
| | `R` | Measurement noise covariance | |
| | `hp` | Parameters of h | default: `aparam` |
| | `aa` | Transformation parameter $\alpha$ | default: $0.5$ |
| | `bb` | Transformation parameter $\beta$ | default: $2$ |
| | `kk` | Transformation parameter $\kappa$ | default: $3 - N$ |
| | `mat` | If 1 uses matrix form | default: $0$ |
| | `s_p_a` | If 1 uses the same parameters on every time step for `a` | default: $1$ |
| | `s_p_h` | If 1 uses the same parameters on every time step for `h` | default: $1$ |
| **Output:** | `M` | Smoothed state mean sequence for all time steps | |
| | `P` | Smoothed state covariance for all time steps | |

## 3.2 Other functions

For convience and to make this toolbox independent of any other toolboxes (e.g. Statistics toolbox) we provide some functions, which are useful for methods described above.

| **gauss_pdf** | | |
|---|---|---|
| Calculates the density of a multivariate Gaussian distribution. | | |
| **Syntax:** `P = gauss_pdf(X,M,S)` | | |
| **Input:** | `X` | $D \times N$ matrix containing $N$ vectors with size $D \times 1$ |
| | `M` | $D \times 1$ mean of distribution or $N$ values as $D \times N$ matrix |
| | `S` | $D \times D$ covariance matrix |
| **Output:** | `P` | Vector containing the probabilities of `X` |

| **gauss_rnd** | | | |
|---|---|---|---|
| Generates random samples from multivariate Gaussian distribution. | | | |
| **Syntax:** `P = gauss_rnd(M,S,N)` | | | |
| **Input:** | `M` | $D \times 1$ mean of distribution or $N$ values as $D \times N$ matrix | |
| | `S` | $D \times D$ covariance matrix | |
| | `N` | Number of samples | default: 1 |
| **Output:** | `X` | $D \times (K * N)$ matrix of samples | |

**der_check**

Evaluates the derivative of given functions derivative analytically and using finite differences. If no output parameters are not given, issues a warning if the derivatives differ too much. Otherwise the derivatives are stored in output parameters. The purpose of this is to check that the derivatives of the dynamic and measurement functions are bug free.

**Syntax:** `[D0,D1] = der_check(F,DF,INDEX,[P1,P2,P3,...])`

**Input:**

| | |
|---|---|
| `F` | Name of the function or inline function in form `F(P1,P2,...)` |
| `DF` | Derivative value as matrix, name of derivative function or inline function in form `DF(P1,P2,...)` |
| `INDEX` | Index of parameter of interest. `DF` should calculate the derivative w.r.t. parameter `Pn`, where `n` is the index. |
| `Pn` | Parameters of the function |

**Output:**

| | |
|---|---|
| `D0` | Actual derivative |
| `D1` | Approximate derivative |

**rk4**

Performs one fourth order Runge-Kutta iteration step for differential equation of the form
$d\mathbf{x}/dt = f(\mathbf{x}(t), \theta(t))$,
or for the chained case
$d\mathbf{x}/dt = f(\mathbf{x}(t), \mathbf{y}(t), \theta(t))$
$d\mathbf{y}/dt = g(\mathbf{x}(t), \theta(t))$,
where $\theta(t)$ is some parameter vector for functions $f$ and $g$ on time $t$.

See the source file for more details and examples.

**Syntax:** `[x,Y] = rk4(f,dt,x,[P1,P2,P3,Y])`

**Input:**

| | | |
|---|---|---|
| `f` | Name of the function in form `f(x,P(:))`, or inline function taking the same parameters. In chained case the function should be `f(x,y,P(:))` | |
| `dt` | Step size as scalar | |
| `x` | Value of x from the previous time step | |
| `P1` | Values of parameters of `f` at initial time $t$ as a cell array or scalar | default: empty |
| `P2` | Values of parameters of `f` at time $t + \Delta\mathrm{t}/2$ | default: `P1` |
| `P3` | Values of parameters of `f` at time $t + \Delta\mathrm{t}$ | default: `P2` |
| `Y` | Cell array of partial results $y_1, y_2, y_3, y_4$ in the RK algorithm of the second parameter in the integrated function. This can be used for chaining the integrators | default: empty |

**Output:**

| | |
|---|---|
| `x` | Next value of x |
| `Y` | Cell array of partial results in Runge-Kutta algorithm |

**schol**

Computes the lower triangular Cholesky factor $\mathbf{L}$ of symmetric positive semidefinite matrix $\mathbf{A}$ such that $\mathbf{A} = \mathbf{L}\mathbf{L}^T$.

**Syntax:** `[L,def] = schol(A)`

**Input:** `A` Matrix to be factorized

**Output:**

| | |
|---|---|
| `L` | Lower triangular Cholesky factor if `A` is not negative definite |
| `def` | Value 1,0,-1 denoting that `A` was positive definite, positive semidefinite of negative definite, respectively |

# References


[1] Bar-Shalom,Y., Li, X.-R., and Kirubarajan, T. (2001). *Estimation with Applications to Tracking and Navigation.* Wiley Interscience.

[2] Bernardo, J.M. and Smith, A.F.M. (1994). *Bayesian Theory.* John Wiley & Sons.

[3] Cox, H. (1964). *On the estimation of state variables and parameters for noisy dynamic systems.* IEEE Transactions on Automatic Control, 9(1):5-12.

[4] Doucet, A., de Freitas, N., and Gordon, N., editors (2001). *Sequential Monte Carlo Methods in Practice.* Springer.

[5] Fraser, D. C. and Potter, J. E. (1969) *The Optimum Linear Smoother as a Combination of Two Optimum Linear Filters.* IEEE Transactions on Automatic Control AC-14, 387-390.

[6] Gelb, A., editor (1974). *Applied Optimal Estimation.* The MIT Press.

[7] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. R. (1995). *Bayesian Data Analysis.* Chapman & Hall.

[8] Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). *Novel approach to nonlinear/non-Gaussian Bayesian state estimation* in IEEE Proceedings on Radar an Signal Processing, volume 140, pages 107-113.

[9] Grewal, M. S. and Andrews, A. P. (2001). *Kalman Filtering, Theory and Practice Using MATLAB.* Wiley Interscience.

[10] Jazwinski, A. H. (1970). *Stochastic Processes and Filtering Theory.* Academic Press.

[11] Julier, S. J. and uhlmann, J. K. (2004). *Corrections to "Unscented Filtering and Nonlinear Estimation"* In Proceedings of the IEEE, 92(12):1958.

[12] Julier, S. J. and Uhlmann, J. K. (2004). *Unscented filtering and nonlinear estimation.* In Proceedings of the IEEE, 92(3):401-422.

[13] Julier, S. J., Uhlmann, J. K., and Durrant-Whyte, H. F. (1995). *A new approach for filtering nonlinear systems.* In *Proceedings of the 1995 American Control, Conference, Seattle, Washington*, pages 1628-1632.

[14] Kalman, R. E. (1960). *A new approach to linear filtering and prediction problems.* Transactions of the ASME, Journal of Basic Engineering, 82:34-45.

[15] Kitagawa, G. (1996). *Monte Carlo filter and smoother for non-Gaussian nonlinear state space models.* Journal of Computational and Graphical Statistics, 5:1-25.

[16] Kotecha, J. H., and Djuric, P. M. (2003). *Gaussian Particle Filtering.* IEEE Transactions on Signal Processing, 51(10):2592-2600.

[17] Maybeck, P. (1982). *Stochastic Models, Estimation and Control, Volume 2.* Academic press.

[18] Ristic, B., Arulampalam, S., and Gordon, N. (2004). *Beyond the Kalman Filter.* Artech House.

[19] Sage, A. P. and Melsa, J. L. (1971). *Estimation Theory with Applications to Communications and Control.* McGraw-Hill Book Company.

[20] Särkkä, S. (2006). *Recursive Bayesian Inference on Stochastic Differential Equations.* Doctoral dissertation, Helsinki University of Technology.

[21] Särkkä, S., Vehtari, A. and Lampinen, J. (2007) *Rao-Blackwellized Particle Filter for Multiple Target Tracking.* Information Fusion, 8(1):2-15.

[22] Särkkä, S. (2006) *Unscented Rauch-Tung-Striebel Smoother.* Submitted.

[23] Vanhatalo, J., and Vehtari, A., (2006). *MCMC Methods for MLP-networks and Gaussian Process and Stuff - A documentation for Matlab Toolbox MCMCstuff.* Toolbox can be found at http://www.lce.hut.fi/research/mm/mcmcstuff/.

[24] Wan, E. A. and van der Merwe, R. (2001). *The unscented Kalman filter.* In Haykin, S., editor *Kalman filtering and Neural Networks*, chapter 7. Wiley.

[25] Wu, Y., Hu, D., Wu, M., and Hu, X. (2005). *Unscented Kalman Filtering for Additive Noise Case: Augmented versus Nonaugmented.* IEEE Signal Processing Letters 12(5):357-360.