M2R PROJECT

GROUP 21

DEPARTMENT OF MATHEMATICS

Stochastic Filtering For State Space Models

AUTHORS
Alexander Pinches, *Kyeongwhan Rho*, *Qingxin Geng*, *Yifei Wang*, *Ziruo Zhao*

SUPERVISOR
Dr. Nikolas Kantas

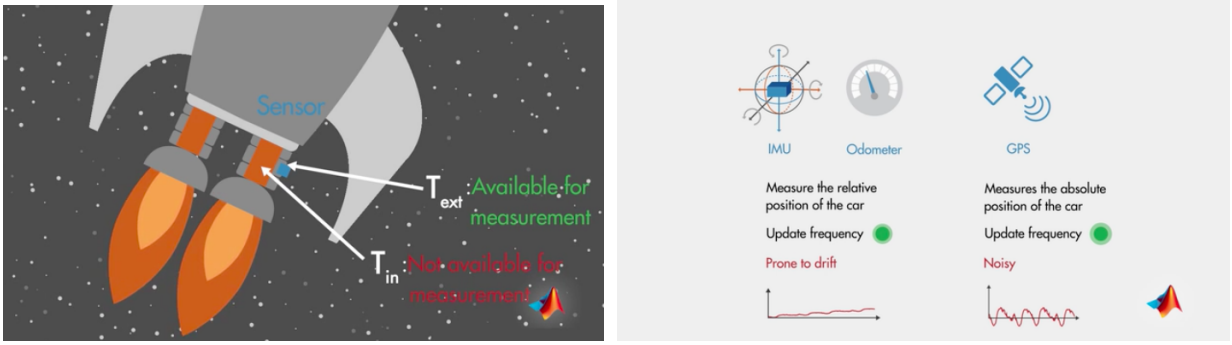
June 18, 2018

Contents

1 Introduction

A **Kalman filter** is an optimal estimation algorithm used to estimate states of a system from indirect and uncertain measurements. Why do we use the Kalman filters?

Let's see an example of spacecraft. The spacecraft's engine can burn fuel at a high temperature, it creates thrust that allows the spacecraft to fly. But too high a temperature can put the mechanical components of the engine at risk, and this can lead to the failure of some of the mechanical parts. To prevent such a situation, we should closely monitor the internal temperature of the combustion chamber. This is not an easy task, since a sensor placed inside the chamber would melt. Instead, it needs to be placed on a cooler surface close to the chamber. In this situation, we can use a Kalman filter to find the best estimate of the internal temperature from an indirect measurement. This way, you're extracting information about what you can't measure from what you can.



(a) Rocket conbustion chamber

(b) Sensors in a car

Figure 1: Example of using the KF
[4]

Now let's take a look at another example. When you're using your car's navigation system, your sensors(IMU and the odometer) measuring the relative position of your car give you fast updates, but they are prone to drift. The GPS receiver provides your absolute location, but it gets updated less frequently and it may be noisy(e.g.in a tunnel). In this scenario, a Kalman filter can be used to fuse these three measurements to find the optimal estimate of the exact position of the car.

Let's summarize these two examples. Kalman filters are used to optimally estimate the variables of interest when they can't be measured directly, but an indirect measurement is available. They're also used to find the best estimate of states by combining measurements from various sensors in the presence of noise. [4]

1.1 Aim of the report

State space models are used ubiquitously in many scientific disciples, engineering, finance and econometrics. They are composed of a sequences of hidden states that evolve as a Markov Chain and a sequence of given observations that depend on each state. When state space model equations are linear and Gaussian then one can compute the conditional distribution of the state sequence given all the observations up to that time. This is done by the Kalman filter that recursively computes sample mean and covariance matrices. For non-linear state space models, the Extended Kalman Filter is used for the same purpose and is based on local linearisation, while the Unscented Kalman Filter is for non-linear systems that are not locally linear.

In this report we present the derivation of the algorithms and some numerical examples when they are used in practice.

1.2 Outline

We will cover the following sections :
State space models, Bayesian Filtering, the Kalman Filter, the Extended Kalman Filter and the Unscented Kalman Filter.

2 State Space Models

State space models are a very popular class of time series models that use state variables to describe a system by a set of first-order differentials. Of which the structure is a good choice for quick estimation.[3]

2.1 State Space Models and Hidden Markov Models

State space models (SSMs), also known as **hidden Markov models (HMMs)**, are composed of a sequence of hidden states $\{X_t\}_{t\geq 0}$ that evolve as a Markov Chain (e.g. unknown positions of moving object, velocity) and a sequence of given observations $\{Y_t\}_{t\geq 0}$ that depend on each state. $\{X_t\}_{t\geq 0}$ is a discrete time Markov Chain with initial density η_θ(x) and transition density $f_\theta(x_t|x_{t-1})$. A given observation sequence $\{Y_t\}_{t\geq 0}$ is conditional on X_t i.i.d with conditional likelihood density $g_\theta(y_t|x_t)$.

Consider a canonical probability space $(\Omega,\mathscr{F},\mathbb{P})$ with $\Omega=\prod_{t=0}^{\infty}(\mathscr{X}\times\mathscr{Y})^t$ and $\mathscr{F}$ the associated Borel σ algebra. We can write the HMMs quite formally for $0\leq t\leq T$ as :

$$\mathbb{P}[X_t\in A\mid(X_{0:T}=x_{0:T},Y_{0:T}=x_{0:T})]=\int_A f_\theta(x\mid x_{t-1})dx,$$

$$\mathbb{P}[Y_t\in B\mid(X_{0:T}=x_{0:T},Y_{0:T}=x_{0:T})]=\int_B f_\theta(y\mid x_t)dy.$$

Or more casually :

$$X_n\sim f_\theta(\cdot\mid x_{n-1}),$$
$$Y_n\sim g_\theta(\cdot\mid x_n).$$

θ are static model parameters, i.e. not time varying or dynamic. For the parameter we assume $\theta\in\Theta\subset\mathbb{R}^{n_\theta}$, Θ is open.

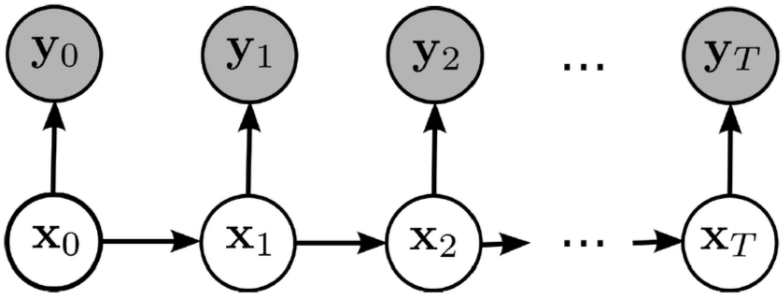


Figure 2: A Hidden Markov Model

HMMs are often described in terms of non-linear and non-Gaussian state space models

$$X_{t+1}=\psi_\theta(X_t,V_{t+1})$$

$$Y_t = \phi_\theta(X_t, W_t)$$

where $\{V_t\}_{t\geq 1}$ and $\{W_t\}_{t\geq 0}$ are arbitrary iid noise sequences and $(\psi_\theta, \phi_\theta)$ are nonlinear functions. θ are model parameters. Often these models are time discretisations of continuous time models, e.g. stochastic differential equations.

2.2 Examples of SSMs

For the Linear Gaussian State Space Model, let $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ and consider

$$X_t = \alpha X_{t-1} + \sigma_v V_t$$

$$Y_t = X_{t-1} + \sigma_w W_t$$

where $W_n, V_n \sim^{i.i.d} \mathcal{N}(0,1)$, $X_0 \sim^{i.i.d} \mathcal{N}(0,1)$. In this case, $\theta = (\alpha, \sigma_v, \sigma_w)$.

One can also define a multi-dimensional version with $\mathcal{X} = \mathbb{R}^{d_x}$ and $\mathcal{Y} = \mathbb{R}^{d_y}$ refer to figure 1 :

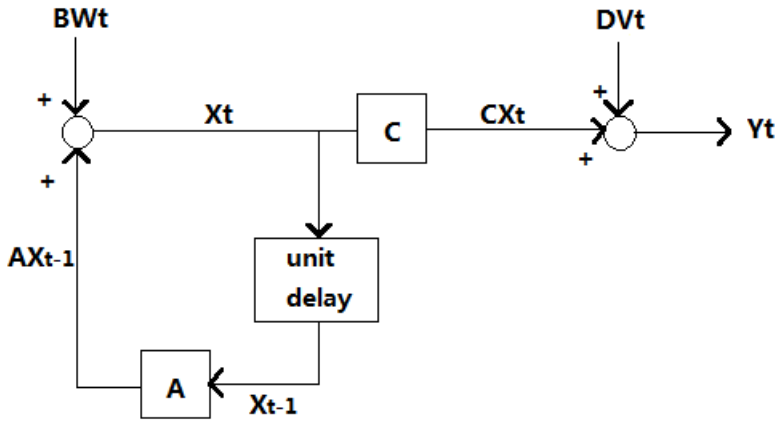


Figure 3: Linear Gaussian state-space model.

$$X_t = AX_{t-1} + BW_t \tag{1}$$

$$Y_t = CX_t + DV_t \tag{2}$$

W_n, V_n i.i.d zero mean Gaussian vectors and A,B,C,D are appropriate matrices.

The Linear Gaussian model is quite a generic model that can be used in many applications, mainly because one can perform many calculations analytically.

Another popular model with $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ is the Stochastic Volatility Model :

$$X_n = \alpha X_{t-1} + \sigma_v V_t$$

$$Y_n = \beta exp(\frac{X_t}{2}) W_t$$

where $W_n, V_n \sim^{i.i.d} \mathcal{N}(0,1)$, $X_0 \sim^{i.i.d} \mathcal{N}(0,1)$. In this case, $\theta = (\alpha, \sigma_v, \beta)$. In this model, X_n is the volatility of an asset and Y_n is its observed log-return.

3 Bayesian Filtering

3.1 State Estimation

Given $y_{0:n}$, to estimate x_n recursively, the object of interest is the **filtering density**

$$\pi_t(x_t) = p(x_t \mid y_{0:t})$$

where $p(x_t \mid y_{0:t})$ is full conditional density.

3.2 Filtering Recursions using Bayes Rule

$\pi_0(x_0)$ gives the initial condition, the initial density is

$$\pi_0(x_0) = p(x_0 \mid y_0) = \frac{p(x_0, y_0)}{p(x_0)}$$

Note

$$p(x_0) = \int p(x_0, y_0) dx_0$$

and if X is discrete, then $\int$ is $\sum$.

At time n we are given observed data $y_{0:n}$ and the filter $\pi_{n-1}(n-1)$ at time $n-1$. Then using π_{n-1} and $y_{0:n}$ to get π_n recursively :
STEP 1 [Prediction] (Chapman Kolmogorov)

$$p(x_t \mid y_{0:t-1}) = \int f(x_t \mid x_{t-1}) p(x_{t-1} \mid y_{0:n-1}) dx_{n-1} = \int p(x_t, x_{t-1} \mid y_{0:t-1}) dx_{t-1}$$

STEP 2 [update] (Bayes Rule)

$$p(x_t \mid y_{0:t}) = \frac{p(x_t, y_t \mid y_{0:t-1})}{p(y_t \mid y_{0:t-1})} = \frac{p(x_t \mid y_{0:t-1}) g(y_t \mid x_t)}{p(y_t \mid y_{0:n-1})} = \frac{p(x_t \mid y_{0:t-1}) g(y_t \mid x_t)}{\int p(x_t \mid y_{0:t-1}) g(y_t \mid x_t) dx_t}$$

In update procedure, $p(x_t \mid y_{0:t-1})$ is the prior, $g(y_t \mid x_t)$ is the likelihood and $p(y_t \mid y_{0:t-1})$ is the evidence term.[1]

If model is linear and Gaussian, this computation is the Kalman Filter.

4 The Kalman Filter (KF)

The Kalman Filter is named after Rudolf E. Kálmán, one of the primary developers of its theory. This section, describes the filter in its original formulation, derivation and numerical example.

4.1 Algorithm

A linear Gaussian state space model is defined as (1) and (2) with $W_t, V_t \sim \mathcal{N}(0, I)$ i.i.d..By the filtering recursion in **3.2**, given the initial distribution $\eta = \mathcal{N}(\hat{X}_{0|0}, P_{0|0})$and f, g of the model are all Gaussian, then we will always obtain a Gaussian distribution.

$$p(x_t \mid y_{1:t-1}) = \mathcal{N}_{x_t}(\hat{X}_{t|t-1}, P_{t|t-1}) \tag{3}$$

$$p(y_t \mid y_{1:t-1}) = \mathcal{N}_{y_t}(m_t, S_t) \tag{4}$$

$$p(x_t \mid y_{1:t}) = \mathcal{N}_{x_t}(\hat{X}_{t|t}, P_{t|t}) \tag{5}$$

The Kalman Filter (KF) computes $\hat{X}_{t|t}$, $P_{t|t}$, m_t, S_t, $\hat{X}_{t|t-1}$,$P_{t|t-1}$ recursively[1] :

$$\hat{X}_{t|t-1} = A\hat{X}_{t-1|t-1} \tag{6}$$

$$P_{t|t-1} = AP_{t-1|t-1}A^T + BB^T \tag{7}$$

$$m_t = C\hat{X}_{t|t-1} \tag{8}$$

$$S_t = CP_{t|t-1}C^T + DD^T \tag{9}$$

$$K_t = P_{t|t-1}C^T S_t^{-1} \tag{10}$$

$$\hat{X}_{t|t} = \hat{X}_{t|t-1} + K_t(Y_t - m_t) \tag{11}$$

$$P_{t|t} = P_{t|t-1} - K_t C P_{t|t-1} \tag{12}$$

4.2 Derivation

4.2.1 Method 1: Based on Linear Minimum Variance (LMV) estimation.

By the linearity of expectation: $E[AX] = AE[X]$ and manipulations of covariances with Gaussians, we have that $BW \sim N(0, BB^T)$ and $DV \sim N(0, DD^T)$.

STEP 1 [Assumptions]

The process and measurement noise random processes W_t and V_t are uncorrelated, zero-mean white-noise processes with known covariance matrices[5]. Then,

$$E[BW_t BW_l^T] = \begin{cases} BB^T & t = l \\ 0 & otherwise \end{cases}$$

$$E[DV_t DV_l^T] = \begin{cases} DD^T & t = l \\ 0 & otherwise \end{cases}$$

$$E[BW_t DV_l^T] = 0 \quad for \quad all \quad t, l$$

where BB^T and DD^T are symmetric positive semi-definite matrices.

The initial system state, X_0 is a random vector that is uncorrelated to both the system and measurement noise processes. **(c)** The initial system has a known mean and covariance matrix:

$$\hat{X}_{0|0} = E[X_0] \quad and \quad \hat{P}_{0|0} = E[(\hat{X}_{0|0} - X_0)(\hat{X}_{0|0} - X_0)^T] \tag{13}$$

The task is to determine, given a set of observations Y_1, ..., Y_{t-1}, the estimation filter that at t^{th} instance in time generates an optimal estimate of the state X_t, which we denote by $\hat{X}_t$, that minimises the expectation of the squared-error loss function:

$$E[||X_t - \hat{X}_t||^2] = E[(X_t - \hat{X}_t)^T(X_t - \hat{X}_t)] \tag{14}$$

STEP 2 [The Predicted State]

Now the solution to the minimisation of Equation (14) is the expectation of the state at time t conditioned on the observations up to time t-1.

$$\hat{X}_{t|t-1} = E[X_t \mid Y_1, ..., X_{t-1}] = E[X_t \mid Y^{t-1}]$$

The predicted state:

$$\begin{aligned}\hat{X}_{t|t-1} &= E[X_t \mid Y^{t-1}] \\ &= E[AX_{t-1} + BW_t \mid Y^{t-1}] \\ &= AE[X_{t-1} \mid Y^{t-1}] + E[BW_t \mid Y^{t-1}] \\ &= A\hat{X}_{t-1|t-1}\end{aligned}$$

because the process noise has zero mean. Therefore, (6) is proved.

The **estimate variance matrix** $P_{t|t-1}$ is the mean squared error in the estimate $\hat{X}_{t|t-1}$.

$$\begin{aligned}P_{t|t-1} &= E[(A(X_{t-1} - \hat{X}_{t-1|t-1}) + BW_t)(A(X_{t-1} - \hat{X}_{t-1|t-1}) + BW_t)^T] \\ &= AE[(X_{t-1} - \hat{X}_{t-1|t-1})(X_{t-1} - \hat{X}_{t-1|t-1})^T]A^T + AE[(X_{t-1} - \hat{X}_{t-1|t-1})W_t^T] \\ &\quad + E[W_t(X_{t-1} - \hat{X}_{t-1|t-1})^T] + E[BW_tBW_t^T].\end{aligned}$$

As W_t and $\hat{X}_{t-1|t-1}$ are uncorrelated:

$$E[(X_{t-1} - \hat{X}_{t-1|t-1})W_t^T] = E[W_t(X_{t-1} - \hat{X}_{t-1|t-1})^T] = 0$$

as the error term and the noise follows normal distribution with expectation 0.
Therefore,

$$\begin{aligned}P_{t|t-1} &= AE[(X_{t-1} - \hat{X}_{t-1|t-1})(X_{t-1} - \hat{X}_{t-1|t-1})^T \mid Y^{t-1}]A^T + E[BW_tBW_t^T] \\ &= AP_{t-1|t-1}A^T + BB^T\end{aligned}$$

Hence (7) is proved, and where BB^T is the **process noise covariance matrix**.

STEP 3 [The Updated State]

We now take one more observation t. In order to derive out $\hat{X}_{t|t}$, we assume it is a linear weighted sum of the prediction and the new observation and can be described by the following equation. Hence we can express K_t' in terms of K_t by making use of the unbiased condition below, and therefore get the formula for the estimate in terms of K_t.

$$\hat{X}_{t|t} = K_t'\hat{X}_{t|t-1} + K_tY_t \tag{15}$$

where K_t' and K_t are gain matries.
Our problem now is to find K_t and K_t' that minimise the conditional mean squared estimation error:

$$\tilde{X}_{t|t} = \hat{X}_{t|t} - X_t.$$

(A) The Unbiased Condition

For our filter to be unbiased, we require: $E[\hat{X}_{t|t}] = E[X_t]$. We assume $\hat{X}_{t|t}$ is an unbiased estimate.

Combining (15) and (2) and take the expectation:

$$\begin{aligned} E[\hat{X}_{t|t}] &= E[K_t']\hat{X}_{t|t-1} + K_t C_t X_t + K_t D V_t] \\ &= K_t' E[X_{\hat{t}|t-1}] + K_t C_t E[X_t] + K_t E[DV_t] \end{aligned} \tag{16}$$

where the last term is zero.

$$E[\hat{X}_{t|t-1}] = E[A\hat{X}_{t-1|t-1}] = AE[\hat{X}_{t-1|t-1}] = E[X_t] \tag{17}$$

By combining (16) and (17):

$$E[\hat{X}_{t|t}] = (K_t' + K_t C_t)E[X_t]$$

As $\hat{X}_{t|t}$ is unbiased,

$$K_t' = I - K_t C_t$$

We plug back into (15):

$$\begin{aligned} \hat{X}_{t|t} &= (I - K_t C_t)\hat{X}_{t|t-1} + K_t Y_t \\ &= \hat{X}_{t|t-1} + K_t[Y_t - C_t\hat{X}_{t|t-1}] \\ &= \hat{X}_{t|t-1} + K_t[Y_t - m_t] \end{aligned} \tag{18}$$

Hence (11) is proved. Here $m_t = C\hat{X_{t|t-1}}$ ((8) is proved) and **K** is known as the **Kalman gain**.

Since $C\hat{X}_{t|t-1}$ can be interpreted as a predicted observation $Y_{t|t-1}$, equation (18) can be interpreted as the sum of a prediction and a fraction of the difference between the predicted and actual observation.[20]

(B) Finding the Error Covariance

We determined the prediction error covariance in equation (18), and we now tun to the updated error covariance:

$$\begin{aligned} P_{t|t} &= E[(\tilde{X}_{t|t}\tilde{X}_{t|t})^T \mid Y^{t-1}] \\ &= E[(X_t - \hat{X}_{t|t})^T(X_t - \hat{X}_{t|t})^T] \\ &= (I - K_t C_t)E[(\tilde{X}_{t|t-1}\tilde{X}_{t|t-1}^T)](I - K_t C_t)^T \\ &+ K_t E[DV_t DV_t^T]]K_t^T + 2(I - K_t C_t)E[\tilde{X}_{t|t-1}DV_t^T]K_{t+1}^T \end{aligned}$$

with

$$E[DV_t DV_t^T] = 0E[(\tilde{X}_{t|t-1}\tilde{X}_{t|t-1}^T)] = P_{t|t-1}E[\tilde{X}_{t|t-1}DV_t^T] = 0$$

we obtain

$$P_{t|t} = (I - K_t C_t)P_{t|t-1}((I - K_t C_t)^T + K_t DD^T K_t^T \tag{19}$$

where DD^T is the **observation noise matrix**.
Expand equation (19) :

$$\begin{aligned} P_{t|t} &= P_{t|t-1} - K_t C_t P_{t|t-1} - P_{t|t-1}C_t^T K_t^T + K_t(C_t P_{t|t-1}C_t^T + DD^T)K_t^T \\ &= P_{t|t-1} - K_t C_t P_{t|t-1} - P_{t|t-1}C_t^T K_t^T + K_t(S_t)K_t^T \end{aligned} \tag{20}$$

where $S_t = CP_{t|t-1}C^T + DD^T$, which is just equation (9).

(C) Properties of MVUE

A minimum variance unbiased estimator (MVUE):

$$\hat{x} = arg\min_{\hat{X}} E[||\hat{X} - X||^2 \mid Y] = E[X \mid Y]$$

The term $E[||X - \hat{X}||^2]$ is the so-called **variance of error**. The variance of error of an estimator is equal to the **trace of the error covariance matrix**:

$$E[||X - \hat{X}||^2] = traceE[(X - \hat{X})(X - \hat{X})^T]$$

(D) Choosing the Kalman Gain

Now our goal is to minimise the conditional mean-squared estimation error with respect to the Kalman gain, K.

$$L = \min_{K_t} E[\tilde{X}_{t|t}^T \tilde{X}_{t|t} \mid Y^t)] = \min_{K_t} trace(E[\tilde{X}_{t|t}\tilde{X}_{t|t}^T \mid Y^t]) = \min_{K_t} trace(P_{t|t}) \tag{21}$$

For any matrix A and a symmetric matrix B

$$\frac{\partial}{\partial A}(trace(ABA^T)) = 2AB$$

(consider writing the trace as $\sum i a_i^T B a_i$where a_i are the columns of A^T, and then differentiating w.r.t.the a_i to see this). Combining (19) and (42) and differentiate with respect to the gain matrix, and set it to be zero:

$$\frac{dL}{dK_t} = -2(CP_{t|t-1})^T + 2K_t(CP_{t|t-1}C^T + DD^T) = 0 \tag{22}$$

Rearrange formula (22):

$$K_t = P_{t|t-1}C^T[CP_{t|t-1}C^T + DD^T]^{-1} = P_{t|t-1}C^T[S_t]^{-1}$$

which is equation (10) we want to prove.
Plug (10) into (20):

$$\begin{aligned} P_{t|t} &= P_{t|t-1} - K_tC_tP_{t|t-1} - P_{t|t-1}H_t^TK_t^T + K_t(S_t)K_t^T \\ &= (I - K_tC_t)P_{t|t-1} - P_{t|t-1}C_t^TK_t^T + P_{t|t-1}C_t^TK_t^T \\ &= (I - K_tC_t)P_{t|t-1} \end{aligned}$$

Hence (12) is proved.

Together with the initial conditions on the estimate and its error covariance matrix (13), this defines the discrete-time sequential, recursive algorithm for determining the linear minimum variance estimate known as the Kalman filter.

4.2.2 Method 2: A probabilistic derivation using Gaussian integral

In this method, we substituting the Gaussian distributions into the predicted and updated equations in the Filtering Recursion of the Kalman Filter, and the key point of the computation in this proof is the Gaussian integral:

$$\int_{\infty}^{\infty} e^{-ax^2+bx+c}dx = \frac{\pi}{a}e^{\frac{b^2}{4a}+c}$$

or in N_x dimensions

$$\int e^{-\frac{1}{2}x^T\Sigma x+x^Tb}dx = \sqrt{\frac{(2\pi)^{N_x}}{det\Sigma}}e^{\frac{1}{2}b^T\Sigma^{-1}b}. \tag{23}$$

STEP 1 [The Predicted State]
For the predicted step :
As we know that $X_t = AX_{t-1} + BW$, Then from equation (3) in the algorithm we have

$$\begin{aligned} p(x_t \mid y_{1:t-1}) &= \int f(x_t \mid x_{t-1})p(x_{t-1} \mid y_{1:t-1})dx_{t-1}[1] \\ &= \frac{1}{\sqrt{2\pi\left|BB^T\right|}}\frac{1}{\sqrt{2\pi\left|P_{t-1|t-1}\right|}}\int exp\big(-\frac{1}{2}(x_t - Ax_{t-1})^T(BB^T)^{-1}(x_t - Ax_{t-1}) \\ &-\frac{1}{2}(x_{t-1} - \hat{x}_{t-1|t-1})^T{P_{t-1|t-1}}^{-1}(x_{t-1} - \hat{x}_{t-1|t-1})\big)dx_{t-1} \end{aligned}$$

Rearranging the terms in exp in the the integral part of the equation and gathering them in the quadratic form about x_{t-1} which is $-\frac{1}{2}x_{t-1}^T\Sigma x_{t-1} + x_{t-1}^Tb$ as in equation (23), then we get that in this case:

$$\Sigma = A^T(BB^T)^{-1}A + {P_{t-1|t-1}}^{-1}$$

and

$$b = \frac{1}{2}A^T(BB^T)^{-1}x_t - \frac{1}{2}{P_{t-1|t-1}}^{-1}\hat{x}_{t-1|t-1}$$

Then using the equation (23) to do the integration by x_{t-1} ,we have

$$\begin{aligned} p(x_t|y_{1:t-1}) &= \frac{1}{\sqrt{2\pi\left|BB^T\right|}}\frac{1}{\sqrt{2\pi\left|P_{t-1|t-1}\right|}}\sqrt{\frac{(2\pi)}{\left|A^T(BB^T)^{-1}A + {P_{t-1|t-1}}^{-1}\right|}} \\ &exp\big(-\frac{1}{2}(\frac{1}{2}A^T(BB^T)^{-1}x_t - \frac{1}{2}{P_{t-1|t-1}}^{-1}\hat{x}_{t-1|t-1})^T\big(A^T(BB^T)^{-1}A + {P_{t-1|t-1}}^{-1}\big)^{-1} \\ &(\frac{1}{2}A^T(BB^T)^{-1}x_t - \frac{1}{2}{P_{t-1|t-1}}^{-1}\hat{x}_{t-1|t-1})\big) \\ &= \frac{1}{\sqrt{2\pi\left|AP_{t-1|t-1}A^T + BB^T\right|}}exp\big(-\frac{1}{2}(x_t - A\hat{x}_{t-1|t-1})(AP_{t-1|t-1}A^T + BB^T)^{-1}(x_t - A\hat{x}_{t-1|t-1})\big) \end{aligned}$$

And from equation (3), we also know that

$$p(x_t \mid y_{1:t-1}) = N(x_t; \hat{x}_{t|t-1}, P_{t|t-1}) = \frac{1}{\sqrt{2\pi\left|P_{t|t-1}\right|}}exp(-\frac{1}{2}(x_t - \hat{x}_{t|t-1})^T(P_{t|t-1})^{-1}(x_t - \hat{x}_{t|t-1}))[7]$$

Comparing these two equations above, we can then get the equations (6) and (7) in the algorithm. Similarly, as we know that $Y_t = Cx_t + DV$, plugging equation (3) into equation (4) we have:

$$\begin{aligned}
p(y_t \mid y_{1:t-1}) &= \int p(x_t \mid y_{1:t-1}) g(y_t \mid x_t) dx_t \\
&= \frac{1}{\sqrt{2\pi \left|DD^T\right|}} \frac{1}{\sqrt{2\pi \left|P_{t|t-1}\right|}} \int exp\big(-\frac{1}{2}(y_t - Cx_t)^T (DD^T)^{-1}(y_t - Cx_t) \\
&- \frac{1}{2}(x_t - \hat{x}_{t|t-1})^T {P_{t|t-1}}^{-1}(x_t - \hat{x}_{t|t-1})\big) dx_t \\
&= N(y_t; m_t, s_t) \\
&= \frac{1}{\sqrt{2\pi |s_t|}} exp(-\frac{1}{2}(y_t - m_t)^T (s_t)^{-1}(y_t - m_t))
\end{aligned}$$

Applying exactly the same method as above, then we can get equations (8) and (9).

STEP 2 [The Updated State]
For the updated step (equation (5)) which uses the Bayes Rule, we have

$$\begin{aligned}
p(x_t|y_{1:t}) &= \frac{g(y_t \mid x_t) p(x_t \mid y_{1:t-1})}{\int g(y_t \mid x_t) p(x_t \mid y_{1:t-1}) dx_t} = \frac{g(y_t \mid x_t) p(x_t \mid y_{1:t-1})}{p(y_t \mid y_{1:t-1})} \\
&= \frac{1}{\sqrt{2\pi \left|DD^T\right|}} \frac{1}{\sqrt{2\pi \left|P_{t|t-1}\right|}} exp\big(-\frac{1}{2}(y_t - Cx_t)^T (DD^T)^{-1}(y_t - Cx_t) \\
&- \frac{1}{2}(x_t - \hat{x}_{t|t-1})^T {P_{t|t-1}}^{-1}(x_t - \hat{x}_{t|t-1})\big) \Big/ \frac{1}{\sqrt{2\pi s_t}} exp(-\frac{1}{2}(y_t - m_t)^T {s_t}^{-1}(y_t - m_t)) \\
&= N(x_t; \hat{x}_{t|t}, P_{t|t}) \\
&= \frac{1}{\sqrt{2\pi \left|P_{t|t}\right|}} exp(-\frac{1}{2}(x_t - \hat{x}_{t|t})^T {P_{t|t}}^{-1}(x_t - \hat{x}_{t|t}))
\end{aligned}$$

Taking log transforms and rearranging the terms in this equation. Then with some algebra we can get the equations (10) – (12).

4.3 Numerical Examples for KF

In this section, numerical examples for Kalman filtering will be introduced. In the section 4.3.1, we will briefly explain the Kalman filter algorithms in a general example. All the examples from 4.3.2 to 4.3.4 are implemented by **R**. At a section 4.3.2, the simplified model with simple constant for Kalman filter will be described. The examples can be divided into two part by dimension, Firstly, $1-D$ case will be analysed at section 4.3.3, and $2-D$ case will be followed at section 4.3.4.

4.3.1 1-D case for tracking problem

In this one-dimensional(scalar) example, considering a train is moving along a railway line. At every measurement epoch we wish to know the best possible estimate of the location of the train. We can then consider some example vectors and matrices in this problem:

The state vector X_t contains the position and velocity of the train: $X_t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix}$

The control vector u_t contains the acceleration applied into the system which can be expressed as a function

of an applied force f_t and the mass of the train m : $u_t = \frac{f_t}{m}$.[8]
Therefore the position and velocity of the train is given by the following equations:

$$x_t = x_{t-1} + (\dot{x}_{t-1} \times \tau) + \frac{f_t(\tau)^2}{2m}$$

$$\dot{x_t} = \dot{x}_{t-1} + \frac{f_t \tau}{m}$$

These linear equations can be written in matrix form as:

$$\begin{bmatrix} x_t \\ \dot{x}_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ \dot{x}_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{(\tau)^2}{2} \\ \tau \end{bmatrix} \frac{f_t}{m}.$$

By comparison with the state equation $X_t = AX_{t-1} + Bu_t + W_t$, we can see for this example that

$$A = \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} \frac{(\tau)^2}{2} \\ \tau \end{bmatrix}.[8]$$

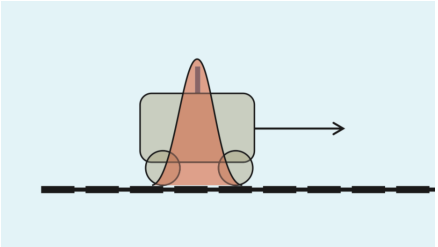


Figure 4: The initial state of the system at time t = 0s is known to a reasonable accuracy. And the location of the train is given by a Gaussian pdf. [8]

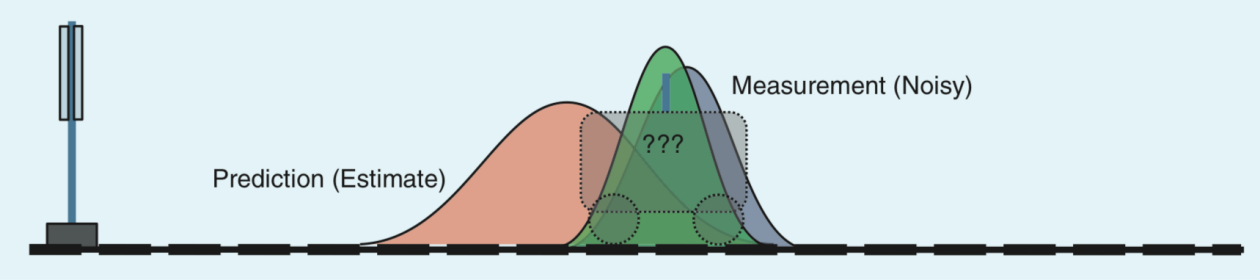


Figure 5: The state of system at the next time epoch t = 1s, the red pdf is the prediction, the blue pdf is the measurement and the green one is the optimal estimate. [8]

At every measurement epoch,the Kalman filter combines the measurement of the location Y_t (eg. we assume the measurements comes from a radio ranging system deployed at the track side) and the prediction $\hat{X}_{t|t-1}$ (in this example,the predictions are based on the last known position and velocity of the train,its maximum possible acceleration and deceleration, etc.) to find the optimal estimate $\hat{X}_{t|t}$ (which is the best possible estimate of the location of the train). Which means that the best current estimate of the system is given by the product of the red Gaussian function Y_1 and blue Gaussian function Y_2. And because the product of two Gaussian functions is another Gaussian function, the Kalman filter has the recursive properties.

To present a general case, note that in reality a function is usually required to map predictions and measurements into the same domain to allow the two pdfs to be multiplied together, And it is standard practice to map the predictions into the measurement domain via the transformation matrix.[8]

And in this example, we consider the distribution $Y2$ to represent the time of flight in seconds for a radio signal propagating from a transmitter positioned at the origin to the antenna on the train. The spatial prediction pdf $Y1$ is converted into the measurement domain by scaling the function by c, the speed of light. Therefore we can write:

$$Y_1(s;\mu_1,\sigma_1,c) = \frac{1}{\sqrt{2\pi(\frac{\sigma_1}{c})^2}} e^{-\frac{(s-\frac{\mu_1}{c})^2}{2(\frac{\sigma_1}{c})^2}}$$

and

$$Y_2(s;\mu_2,\sigma_2) = \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(s-\mu_2)^2}{2\sigma_2^2}} \text{ [8]}$$

where both distributions are now defined in the measurement unit: the second. Therefore

$$\begin{aligned} Y_{fused}(s;\mu_1,\sigma_1,c,\mu_2,\sigma_2) &= Y_1(s;\mu_1,\sigma_1,c) \times Y_2(s;\mu_2,\sigma_2) \\ &= \frac{1}{\sqrt{2\pi(\frac{\sigma_1}{c})^2{\sigma_2}^2}} e^{-\frac{(s-\frac{\mu_1}{c})^2}{2(\frac{\sigma_1}{c})^2}-\frac{(s-\mu_2)^2}{2{\sigma_2}^2}} \end{aligned} \tag{24}$$

The quadratic terms in the new function Y_{fused} in (24) can be expanded and then the whole expression can be rewritten in Gaussian form:

$$Y_{fused}(r;\mu_{fused},\sigma_{fused}) = \frac{1}{\sqrt{2\pi\sigma_{fused}^2}} e^{-\frac{(r-\mu_{fused})^2}{2{\sigma_{fused}}^2}}$$

where

$$\frac{\mu_{fused}}{c} = \frac{\mu_1}{c} + \frac{(\frac{\sigma_1}{c})^2(\mu_2 - \frac{\mu_1}{c})}{(\frac{\sigma_1}{c})^2 + \sigma_2^2}$$

Therefore

$$\mu_{fused} = \mu_1 + \big(\frac{\frac{\sigma_1^2}{c}}{(\frac{\sigma_1}{c})^2 + \sigma_2^2}\big)(\mu_2 - \frac{\mu_1}{c})$$

Substituting $H = 1/c$ and $K = (H\sigma_1^2)/(H^2\sigma_1^2 + \sigma_2^2)$ results in

$$\mu_{fused} = \mu_1 + K(\mu_2 - H\mu_1).$$

Similarly the fused variance estimate is:

$$\frac{\sigma_{fused}^2}{c^2} = (\frac{\sigma_1}{c})^2 - \frac{(\frac{\sigma}{c})^4}{(\frac{\sigma}{c})^2 + \sigma_2^2}$$

Therefore,

$$\sigma_{fused}^2 = \sigma_1^2 - \big(\frac{\frac{\sigma_1^2}{c}}{(\frac{\sigma_1}{c})^2 + \sigma_2^2}\big)\frac{\sigma_1^2}{c} = \sigma_1^2 - KH\sigma_1^2 \text{[8]}$$

We can now compare certain terms from this scalar example with the standard vectors and matrices used in the Kalman filter algorithm from equations (6)−(12): $\mu_{fused} \to \hat{x}_{t|t}$, $\mu_1 \to \hat{x}_{t|t-1}$, $\sigma_{fused}^2 \to P_{t|t}$, $\sigma_1^2 \to P_{t|t-1}$, $\mu_2 \to Y_t$, $\sigma_2^2 \to D$ and $H \to C$. And then we can get exactly the same predicted and updated equations as above.

4.3.2 1-D and 2-D case for implementing to code

In order to implement examples into code, the models which is utilised for Kalman filter should be defined at first. the coefficients of a state model and an observation model will be defined at this section. Each model which was introduced at section 4.1 will be replicated as below;

In the examples, 4.3.3 and 4.3.4, which are implemented by **R** code, the coefficients of the state model and observational model will be set as 1 (or column vectors with 1-entry), and variance of error term will be set as 1 (or identity matrix) as below;

To simplify the models and implement them as code, set $A = 1$, $B = 1$, $C = 1$, and $D = 1$ in the equations (1) and (2), and the model now becomes $X_t = X_{t-1} + W_t$ and $Y_t = X_t + V_t$ where $X_0 \sim N(0, I)$. And we take values of observations defined for $t = 1, .., T$.[11] [12]

4.3.2.1 Numerical Example for 1-D Case

In this section, in terms of 1-D observed data, its filtering mean and variance, which the Kalman filter is applied, will be analysed.

The raw data (observed data) which is utilised for simulation is generated randomly. In this example, the raw data follows sin(x) as a mean with different values of variances. It means that the raw data is randomly scattered from the sin(x) with the amount of a specific value of variance. Therefore, the randomly selected data has a noise following the variance.

From the raw data, filtering mean and variance is generated by Kalman filtering. Hence, with respect to each point of raw data will be utilised for a filtering mean and a filtering variance. Therefore, the mean and variance can be plotted as points and a credible interval respectively.[13]

First case: When the raw data is randomly generated with the amount of the variance of 0.3 from the sin(x), its filtering mean is plotted over time, and the 99% credible intervals are drawn over time as below;

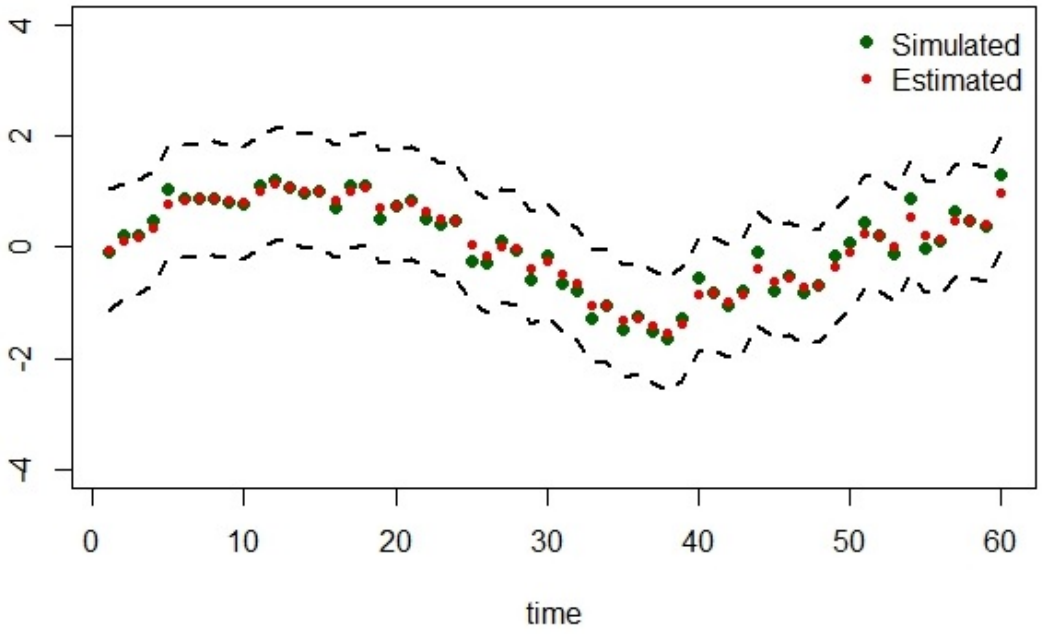


Figure 6: The plot for simulated data and its filtering mean with the 99% credible intervals, in case of SD = 0.3.

Second case: When the raw data is randomly generated with the amount of the variance of 0.5 from the sin(x),

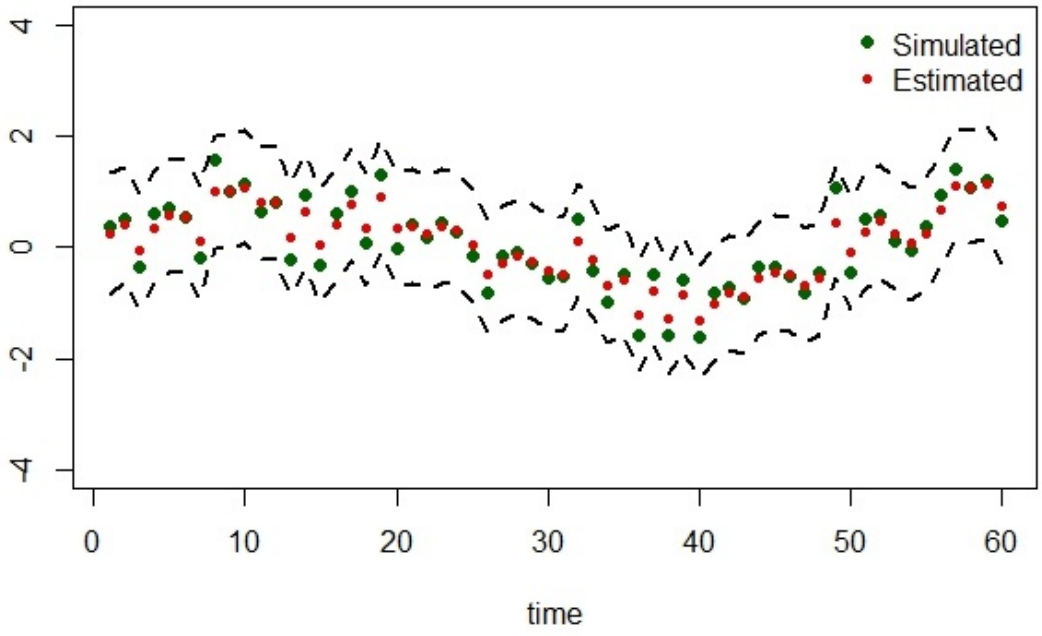


Figure 7: The plot for simulated data and its filtering mean with the 99% credible intervals, in case of SD = 0.5.

To compare the above figures 6 and 7, in general, the filtering mean is plotted with less scattered than observed data. Furthermore, if the data has more noise, the filtering mean is more scattered. In this case,

majority of data and filtering mean is plotted in the 99% credible interval. The R code for these examples is in Appendix 10.2.1

4.3.2.2 Numerical Example for 2-D Case

In this section, practical example for 2-D case will be introduced. With respect to 2-D raw data, all the data which represents the motion of moving car is also generated randomly by R code. [14]

First and foremost, the simulated data is a position of a moving car. The observed data is a vector which represents a position analysed in 2-D Cartesian coordinates. The car moves on extremely winding path following two mountains. Before analysing the data, in order to help the understanding of the motion of the car, a contour map, which the car follows, is added as below at figure 8; [15] [17]

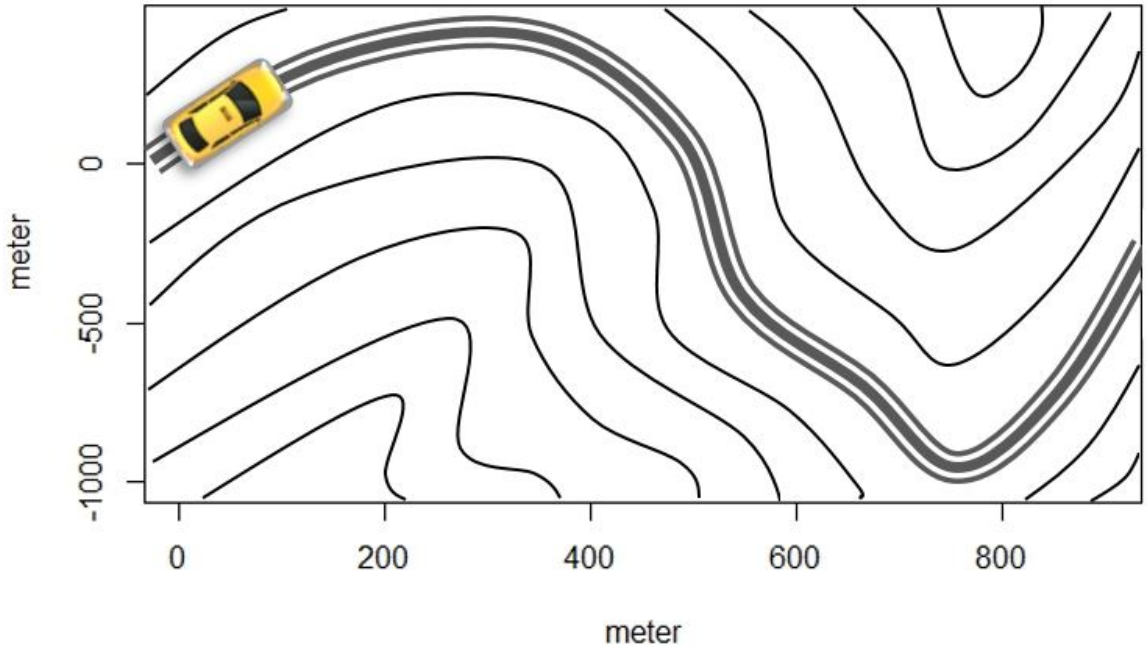


Figure 8: The contour map which a car follows.

The position of the moving car is analysed as a vector in x and y direction. The each measurement of the car is taken every 5 seconds by GPS. The GPS measurements of the position of the car is measured over 300 seconds. The horizontal and vertical position (measured in meters) represent position x and position y respectively. The plot of the position and its estimated position are given as below at figure 9.

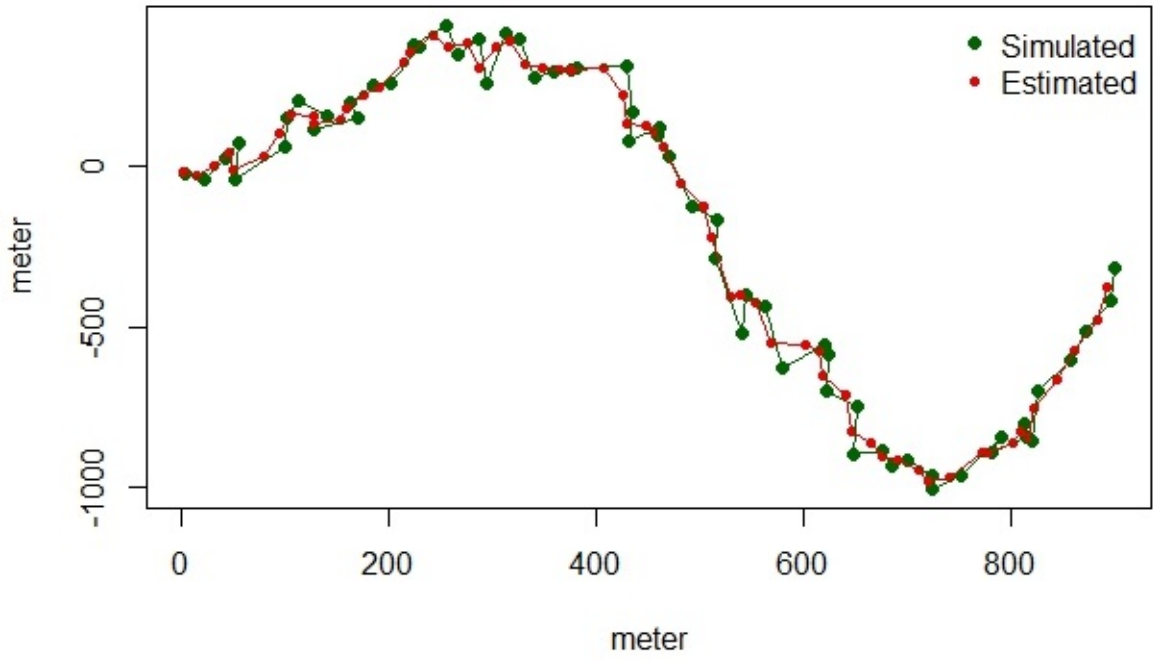


Figure 9: Plot of position of the moving car (GPS data) and its estimated position.

From the position data, the speed of the car can also be analysed. The estimated speed can be computed by Kalman filtering. The Figure 10 shows the speed of the car at the specific position (or time) and its estimated speed.

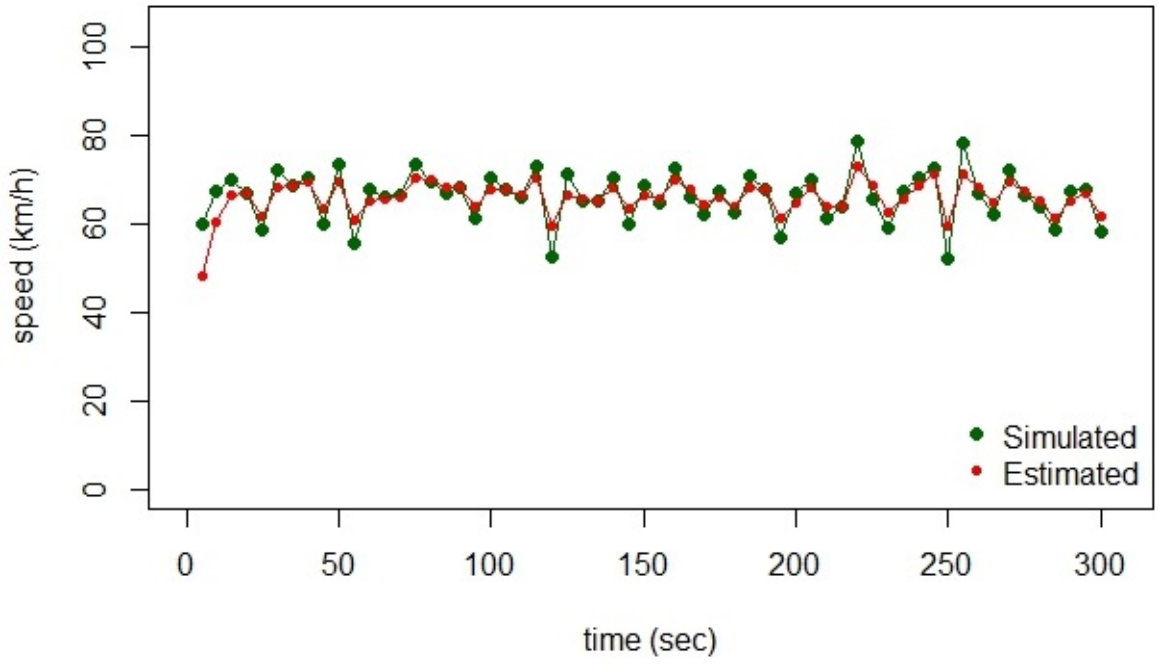


Figure 10: Plot of speed of the car and its estimated speed.

To conclude the section 6.4, Kalman filter is applied to analyse GPS position data. From the estimated

position, estimated speed of the car can be computed. This process is actively applied in the real world. For instance, "Google Map" can be a proper example. The estimated data can be utilised when Wifi or mobile data are not connected during the moving car is passing through a tunnel. Hence, the estimated position will be used to predict a position of the car in the tunnel, and the estimated speed will be used to keep predicting the estimated arrival time to destination.

5 The Extended Kalman Filter (EKF)

5.1 Introduction to the EKF

Previously, we used simplified linear models to discuss state estimation through Kalman filters. However, in a general system, either the state transition function, or the measurement function or both may be nonlinear. Here we need to use a non-linear state estimator instead of a Kalman filter.

If the state transition function is nonlinear, then the resulting state distribution after undergoing the linear transformation may not be Gaussian. Therefore, the Kalman filter algorithm may not converge. In this case, we can implement an extended Kalman filter (EKF), which linearises the nonlinear function around the mean of the current state estimate.[21] At each time step, the linearisation is performed locally and the resulting Jacobian matrices are then used in the prediction and update states of the Kalman filter algorithm. When the system is non-linear and can be well approximated by linearisation, then extended Kalman filter is a good option for state estimation.

However, the complicated derivation of Jacobians, the high computational cost, the restriction of systems (only continuous) that are applicable to and the bad behavior for highly nonlinear systems are the drawbacks for the EKF.[4]

5.2 Algorithm

The nonlinear and non-Gaussian model is defined as

$$X_{t+1} = \psi_\theta(X_t, V_{t+1}), Y_t = \phi_\theta(X_t, W_t)$$

Replace with an approximate linear Gaussian model

$$X_t = a_t + A_t X_{t-1} + B_t W_t \tag{25}$$

$$Y_t = c_t + C_t X_t + D_t V_t \tag{26}$$

Linearisation of ψ, ϕ around $\hat{X}_{t-1|t-1}$, $\hat{X}_{t|t-1}$ resp. gives

$$A_t = \nabla_X \psi_\theta \mid_{\hat{X}_{t-1|t-1}}, C_t = \nabla_X \phi_\theta \mid_{\hat{X}_{t|t-1}}$$

and similarly assuming the noises are zero mean we can set

$$B_t = \nabla_W \psi_\theta \mid_0, D_t = \nabla_V \phi_\theta \mid_0$$

so

$$a_t = \psi_\theta(\hat{X}_{t-1|t-1}, 0) - A_t \hat{X}_{t-1|t-1}, c_t = \phi_\theta(\hat{X}_{t|t-1}, 0) - C_t \hat{X}_{t|t-1}$$

The EKF computes $\hat{X}_{t|t}$, $P_{t|t}$, m_t, S_t, $\hat{X}_{t|t-1}$,$P_{t|t-1}$ recursively[1] :

$$\hat{X}_{t|t-1} = a_t + A_t \hat{X}_{t-1|t-1} \tag{27}$$

$$P_{t|t-1} = A_t P_{t-1|t-1} A_t^T + B_t B_t^T \tag{28}$$

$$m_t = c_t + C_t\hat{X}_{t|t-1} \tag{29}$$

$$S_t = C_tP_{t|t-1}C_t^T + D_tD_t^T \tag{30}$$

$$K_t = P_{t|t-1}C_t^TS_t^{-1} \tag{31}$$

$$\hat{X}_{t|t} = \hat{X}_{t|t-1} + K_t(Y_t - m_t) \tag{32}$$

$$P_{t|t} = P_{t|t-1} - K_tC_tP_{t|t-1} \tag{33}$$

5.3 Derivation

Now in the EKF case, we know that A,B,C and D are dependent on t, and in our model, each of X_t and Y_t has one more contributive term, namely a_t and c_t. The derivation of EKF is actually based on the derivation of KF, they are similar except these differences.

To see the proof, see Appendix 8.2.

5.4 Numerical Examples for the EKF

In this section, numerical examples in the one dimensional case of the Extended Kalman Filter will be provided.

In order to implement the examples into R code, we must define a model similarly to 4.4 but with some more time dependent variables. According to (25) and (26), and set B and D to be 1 while a_t and c_t are 0, we can rearrange the model as $X_t = A_t(X_{t-1}) + W_t$ and $Y_t = C_t(X_t) + V_t$ where A_t and C_t are functions with respect to t. Here set the covirance of both W and V to be 0.8 and $X_0 \sim N(0,4)$.

Using the R code for the Extended Kalman Filter provided in the appendix and the variables set as below:

$T = 50$	$C = t$	$A = sin(t)$
$x = Asin(t)$	$R = 0.8$	$\hat{X}_0 = 0$
$P_0 = 4$	$B = 2$	$Q = 0.8$
$\alpha = 0.05$	$y = Cx + N(0,\sqrt{R})$	

The data is generated with random noise from a sine function.

The function produces 3 graphs each the predictor, the Kalman Filter and the Rauch–Tung–Striebel Smoothed Kalman Filter.The average (μ_p, μ_f, μ_s respectively) is plotted as a black line, the credible interval for the given α is plotted as a red dotted line and the observations Y_t are plotted as a green line.

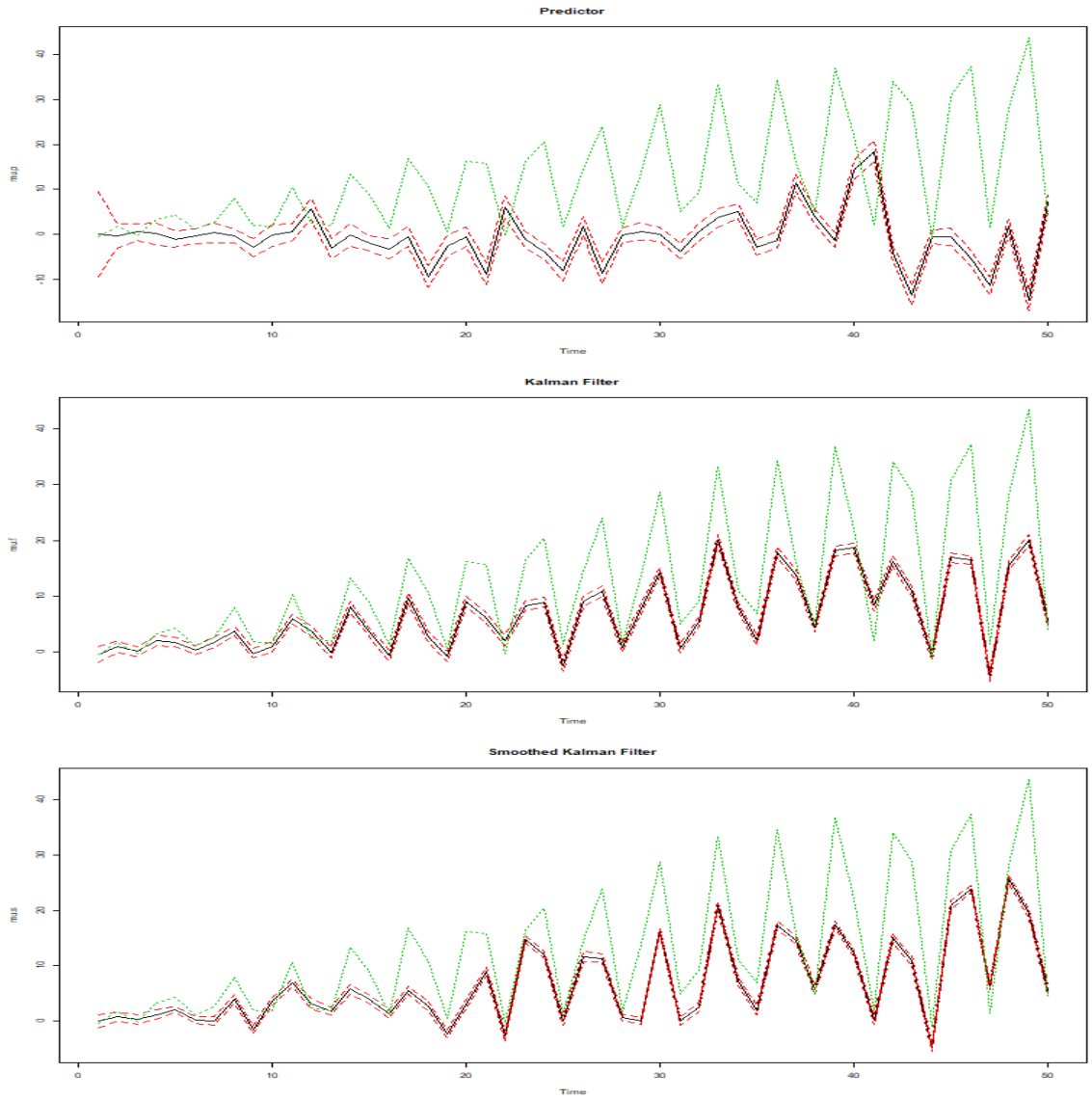


Figure 11: Graphs of the Predictor, Extended Kalman Filter and The Smoothed Extended Kalman Filter.

In figure 11 we can see the credible interval is smaller after the filter is applied and after smoothing showing that the filter has filtered out noise in the system. Thus this shows that the true mean is more concentrated around the mean produced after filtering and after smoothing. This shows that the filter has filtered noise out of the system.

6 The Unscented Kalman Filter(UKF)

6.1 Introduction to the UKF

The UKF is an alternative filter with performance superior to that of the EKF at an equivalent computational complexity. The EKF algorithms provides only an approximation to optimal nonlinear estimation. In the EKF, the state distribution is approximated by a GRV (Gaussian random variables),which is then propagated analytically through the first-order linearisation of the nonlinear system. Therefore large errors in the true

posterior mean and covariance of the transfered GRV are introduced which may lead to sub-optimal performance and sometimes divergence of the filter.[22] However, The UKF addresses this problem by using a deterministic sampling approach. The state distribution is again approximated by a GRV,but is now represented using a minimal set of carefully chosen sample points. These sample points completely capture the true mean and covariance accurately to the 2nd order(Taylor series expansion) for any nonlinearity. The EKF, in contrast, only achieves first-order accuracy. [9]

6.2 Algorithm

Using the model given in 4.1 with the addition of the variables κ and λ. The unscented transform for $Y = f(X)$, where $X \sim N(\bar{X}, P_X)$. To calculate the statistics of Y, we form a matrix χ of 2L+1 sigma vectors χ_i. We initialise with: [9]

$$\chi_0 = \bar{X} \tag{34}$$

For $i = 1, ..., L$ set

$$\begin{aligned} \chi_i &= \bar{X} + \left(\sqrt{(L+\lambda)P_x}\right)_i \\ \chi_{i+n} &= \bar{X} - \left(\sqrt{L+\lambda)P_x}\right)_{i+n} \end{aligned} \tag{35}$$

Where $(\sqrt{P_x})_i$ is defined as the i-th column of a matrix Q such that $P_x = QQ^T$. which can be found through methods such as Choleski Decomposition.

Then propagate the sigma points through $f(\cdot)$

$$Y_i = f(X_i), \qquad i = 0, ..., 2L \tag{36}$$

Use quadrature estimates

$$\mathbb{E}[f(X)] = \hat{X}_h \approx \sum_{i=0}^{2L} W_i^{(m)} Y^i \tag{37}$$

$$\mathrm{Cov}[f(X)] = P_{Xh} \approx \sum_{i=0}^{2L} W_i^{(c)} \left(Y_i - \hat{X}_h\right)\left(Y_i - \hat{X}_h\right)^T \tag{38}$$

with the weights W_i defined as

$$W_0^{(m)} = \frac{\lambda}{L+\lambda} \tag{39}$$

$$W_0^{(c)} = \frac{\lambda}{L+\lambda} + \kappa \tag{40}$$

$$W_i^{(m)} = W_i^{(c)} = \frac{1}{2(L+\lambda)}, \qquad i = 1, ..., 2L \tag{41}$$

where κ is a scaling parameter.

We initialize with:

$$\hat{X}_0 = E[X_0]$$

$$P_0 = E[(X_0 - \hat{X}_0)(X_0 - \hat{X}_0)^T]$$

$$\hat{X}_0^a = E[X^a] = \begin{bmatrix} \hat{x}_0^T & 0 & 0 \end{bmatrix}$$

$$P_0^a = E[(X_0^a - \hat{X}_0^a)(X_0^a - \hat{X}_0^a)^T] = \begin{bmatrix} P_0 & 0 & 0 \\ 0 & R^v & 0 \\ 0 & 0 & R^n \end{bmatrix}$$

For $t \in 1,\ldots,\infty$, calculating sigma points :

$$\chi_{t-1}^a = \begin{bmatrix} \hat{X}_{t-1}^a & \hat{X}_{t-1}^a + \gamma\sqrt{P_{t-1}^a} & \hat{X}_{t-1}^a - \gamma\sqrt{P_{t-1}^a} \end{bmatrix}$$

Time update :

$$\chi_{t|t-1}^x = A[\chi_{t-1}^x, u_{t-1}, \chi_{t-1}^v]$$

$$\hat{x}_{t|t-1} = \sum_{i=0}^{2L} W_i^{(m)} \chi_{i,t|t-1}^x \tag{42}$$

$$P_{t|t-1} = \sum_{i=0}^{2L} W_i^{(c)} [\chi_{i,t|t-1}^x - \hat{x}_{t|t-1}][\chi_{i,t|t-1}^x - \hat{x}_{t|t-1}]^T \tag{43}$$

$$y_{t|t-1} = C[\chi_{t|t-1}^x, \chi_{t-1}^n]$$

$$\hat{y}_{t|t-1} = \sum_{i}^{2L} W_i^{(m)} y_{i,t|t-1}$$

Measurement update equations:

$$P_{\tilde{y}_t\tilde{y}_t} = \sum_{i=0}^{2L} W_i^{(c)} [y_{i,t|t-1} - \hat{y}_{t|t-1}][y_{i,t|t-1} - \hat{y}_{t|t-1}]^T$$

$$P_{x_t y_t} = \sum_{i=0}^{2L} W_i^{(c)} [\chi_{i,t|t-1} - \hat{x}_{t|t-1}][y_{i,t|t-1} - \hat{y}_{t|t-1}]^T \tag{44}$$

$$K_t = P_{x_t y_t} P_{\tilde{y}_t\tilde{y}_t}^{-1} \tag{45}$$

$$\hat{x}_t = \hat{x}_{t|t-1} + K_t(y_t - \hat{y}_{t|t-1}) \tag{46}$$

$$P_t = P_{t|t-1} - K_t P_{\tilde{y}_t\tilde{y}_t} K_t^T \tag{47}$$

where, $x^a = \begin{bmatrix} x^T & v^T & n^T \end{bmatrix}^T$, $\chi^a = \begin{bmatrix} (\chi^x)^T & (\chi^v)^T & (\chi^n)^T \end{bmatrix}^T$, $\gamma = \sqrt{L+\lambda}$,$\lambda$ =composite scaling parameter, L =dimension of augmented state, R^v =process noise cov., R^n =measurement noise cov., W_i =weight as calculated from (39) – (41). [9]

6.3 Derivation

The UKF is actually not off the frame of KF, except predicting the further status by extending sigma points and non-linear mapping.

To see the proof, see Appendix 8.3.

6.4 Numerical Example for the UKF

Following the algorithm defined in the section 6.2 , using the R code for the Unscented Kalman Filter provided in the appendix 8.4.3 and with the same variables as example 5.4 and weights with zero mean shown below:

$$T = 50 \qquad C = t \qquad A = sin(t)$$
$$x = Asin(t) \qquad R = 0.8 \qquad \hat{X}_0 = 0$$
$$P_0 = 4 \qquad B = 2 \qquad Q = 0.8$$
$$\alpha = 0.05 \qquad \lambda = 0.00000001 \qquad \kappa = 0$$
$$y = Cx + N(0, \sqrt{R})$$

The observed data is generated with random noise as above following a sin function. The unscented kalman filter is applied producing a graph of the mean calculated in the predictive step, after the update step and after smoothing. A 95% credible interval is shown in red and the raw data in green.

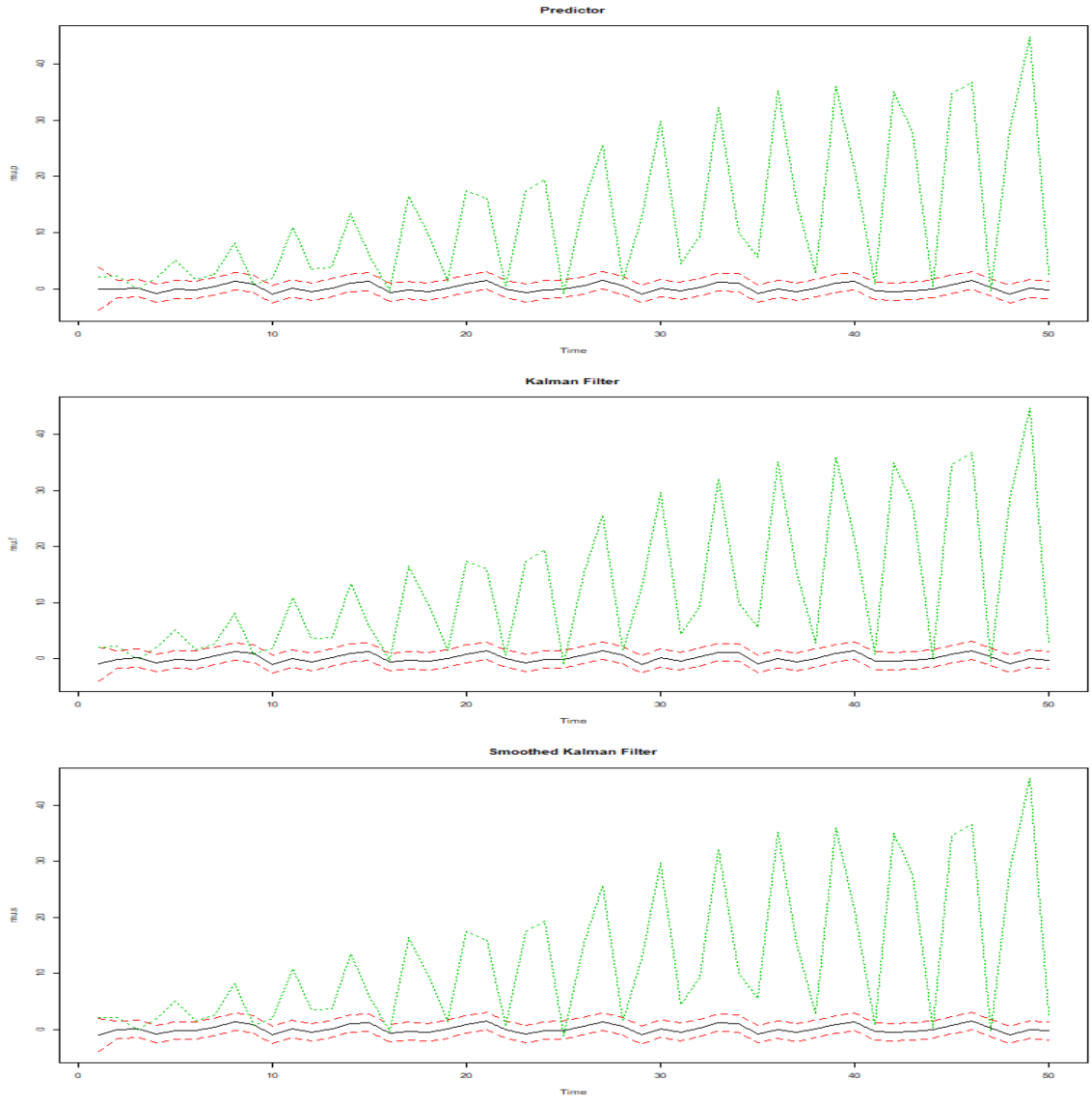


Figure 12: Graphs of the Predictor, Unscented Kalman Filter and Smoothed Unscented Kalman Filter.

In figure 12, we can see that the the Unscented Kalman Filter produces a mean values that don't vary

much with respect to time and with a small credible interval. This shows that the UKF may not be useful in this case as it doesn't appear to follow the observations well as time progresses.

7 Conclusion

State Estimator	Model	Assumed Distribution	Computational Cost
Kalman filter (KF)	Linear	Gaussian	Low
Extended Kalman filter (EKF)	Locally Linear	Gaussian	Low (if the Jacobians need to be computed analytically) Medium (if the Jacobians can be computed numerically)
Unscented Kalman filter (UKF)	Nonlinear	Gaussian	Medium
Particle filter (PF)	Nonlinear	Non - Gaussian	High

Figure 13: Comparison of KF, UKF and EKF[4]

.

7.1 Comparison of the KF, EKF and UKF

7.1.1 Linear Gaussian Distributed Models

If a particular model is linear and is assumed to be distributed according to a Gaussian distribution, one would use a Kalman Filter. This can be seen in the table shown in figure 13. As the Extended and Unscented Kalman Filters are for models that are non-linear and have a higher computational cost as they require the calculation of Jacobians or Sigma Points.[23]

7.1.2 Non-Linear Gaussian Distributed Models

If the model is non-linear and is also assumed to be distributed according to a Gaussian distribution, then one has two options the Extended Kalman Filter and the Unscented Kalman Filter. For models that are locally linear you could use the Extended Kalman Filter.[24] The computational cost of this approach depends on the difficulty of finding the Jacobians. However, if a model is not locally linear or has difficult to calculate Jacobians or if they do not exist then an Unscented Kalman Filter is more appropriate.[25]

7.1.2.1 Comparison Of the 1-D Examples 5.4 and 6.4

In the examples 5.4 and 6.4 the same parameters are used so we can overlay the graphs in figures 11 and 6.4 to compare the two filters for the same input parameters.

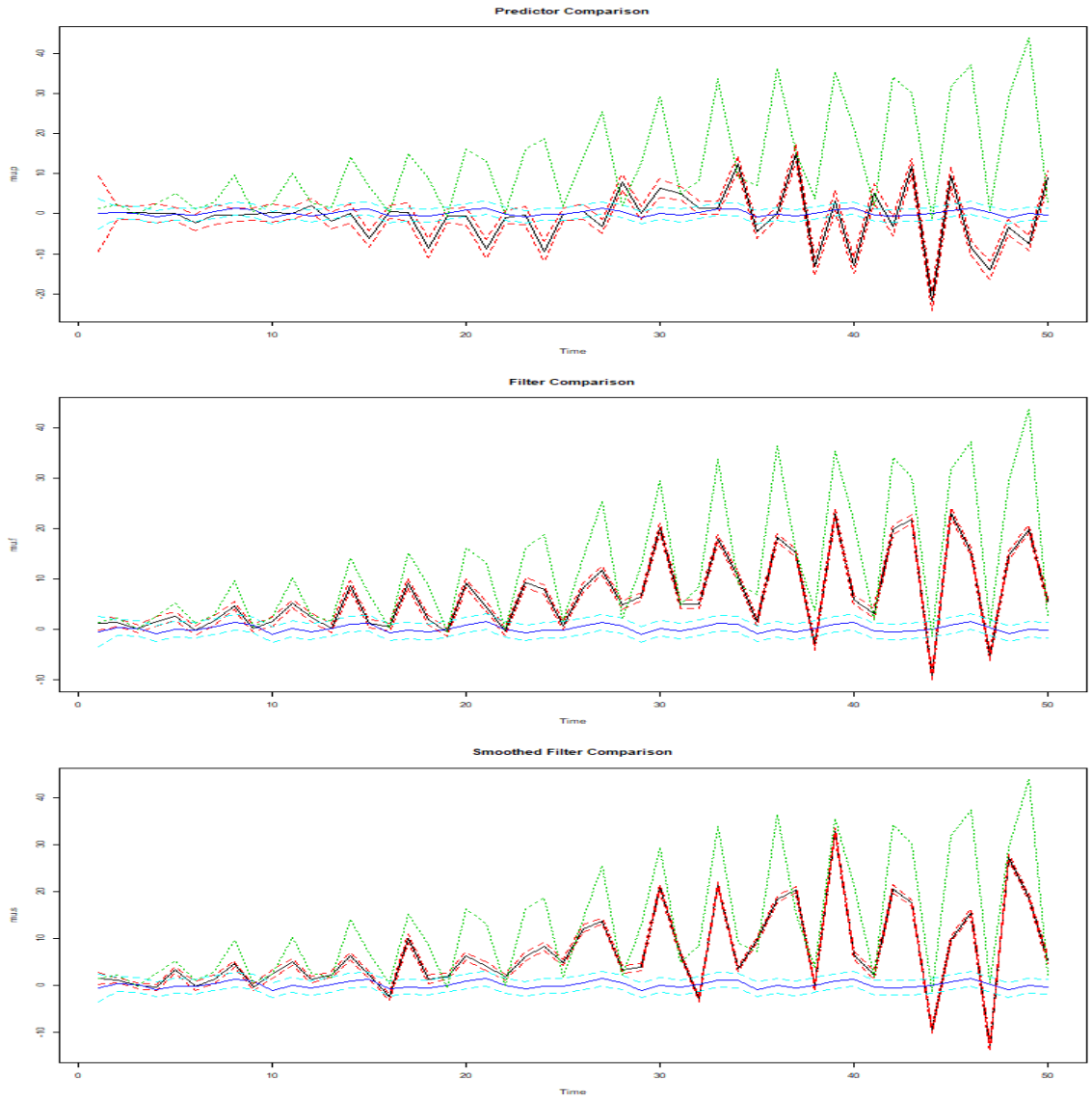


Figure 14: Graph comparing the EKF with the UKF

In figure 14 the two graphs are overlaid with the EKF mean in black and the 95% credible interval in red, the UKF mean in navy and the 95% credible interval in light blue and with the observed data in green.

The mean produced by the EKF is more variable but the mean produced by the UKF however doesn't vary much with time.Indicating that the Extended Kalman Filter may be preferable in this case as it matches the observed data better.[26]

7.1.2.2 Case where the UKF is preferable

In some cases for Jacobians that are difficult to calculate the UKF can be preferable for example using the R code outlined in the appendix for the EKF 8.4.2 and UKF 8.4.3, the parameter outlined below and overlaying

the graphs in figure.

$$T = 50 \qquad C = \frac{1}{t} \qquad A = sin(t)$$
$$x = Asin(t) \qquad R = 0.8 \qquad \mu_0 = 0$$
$$\sigma_0 = 4 \qquad B = 2 \qquad Q = 0.8$$
$$\alpha = 0.05 \qquad \lambda = 0.00000001 \qquad \kappa = 0$$
$$y = Cx + N(0, \sqrt{R})$$

The variables κ and λ are not used in the EKF as shown in the code in appendix 8.4.2.

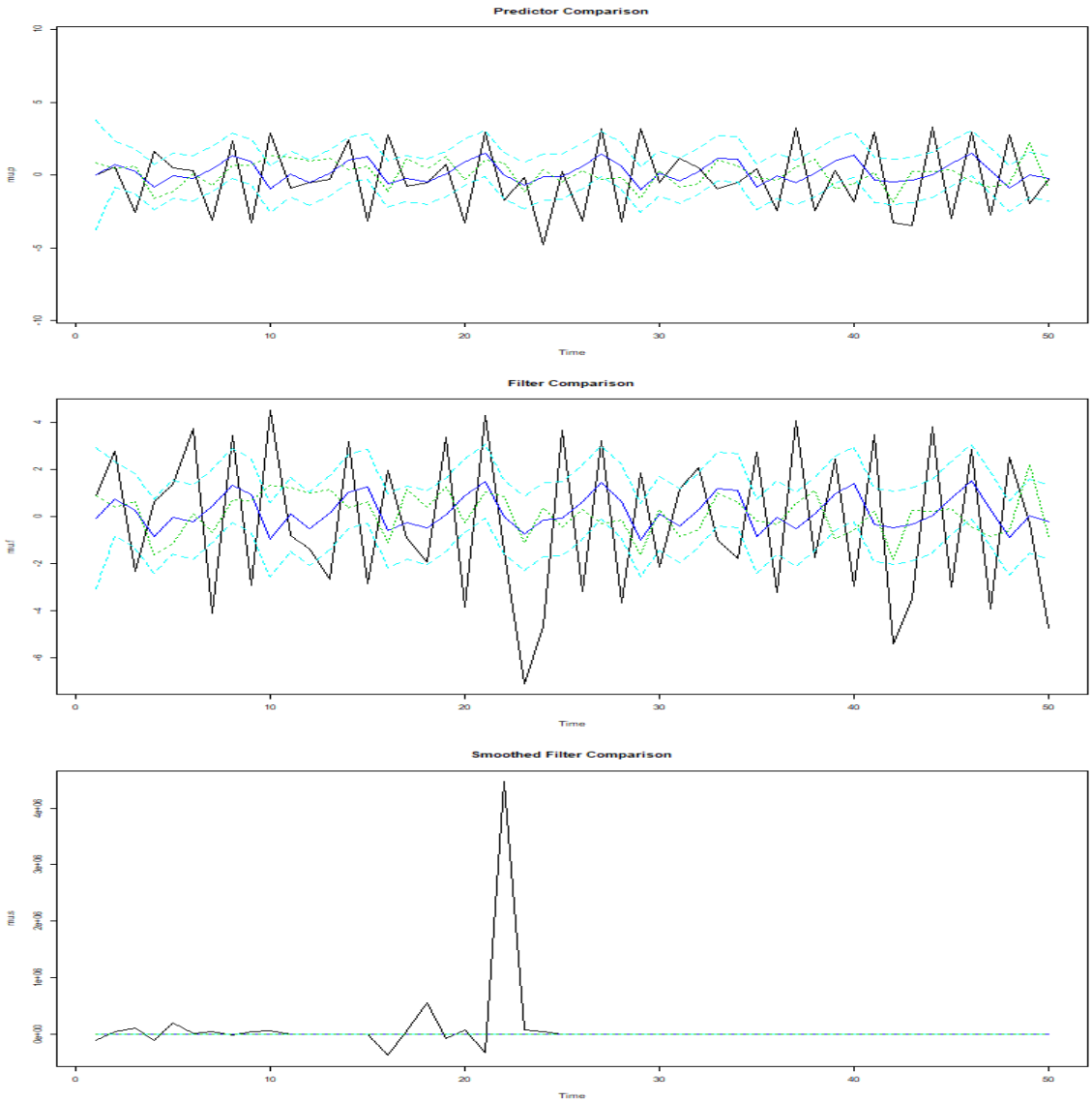


Figure 15: Graph comparing the EKF with the UKF for the parameters defined above

The same colours are used as in the graph in figure 14. We can see in this case the UKF fits the observed data much better than the EKF and the EKF also has a credible interval that is undefined as it is so large. This is due to the model not being locally linear.

7.2 Future Work

7.2.1 Continuous time

In some state space models time can be modeled continuously instead of at set time intervals.A general example of such a model is given below;

$$\frac{d}{dt}X(t) = A(t)X(t) + B(t)U(t) + W(t) \qquad State\ \ model$$
$$Y(t) = C(t)X(t) + V(t) \qquad Observation\ \ model$$

This can be calculated by a modification of the Kalman Filter called the Kalman-Bucy Filter. Which unlike the Kalman Filter does not have a distinguishable prediction and update step but uses the solutions to two differential equations. One for the mean and one for the covariance. [16]

7.2.2 Particle Filter for Non-Gaussian distributed models

The Kalman Filters assume that the model is distributed according to a Gaussian Distribution. However some models may be distributed differently meaning you cant apply the Kalman Filter but need another filter such as the the particle filter mentioned in figure 13. However, these are much more computationally complex as the probability distributions often can't be calculated analytically such as with a Gaussian Distribution. Thus require a numerical approach which is much more computationally expensive except in some specific cases where they can still be calculated analytically.[19]

8 Appendix

8.1 Lemmas

Lemmas used throughout with the sample space defined as Ω

8.1.1 Borel σ-algebra

Definition *For a sample space Ω. A collection of subsets of Ω is called a σ-algebra field or Borel field,$\mathscr{F}$, if:*

i $\emptyset \in \mathscr{F}$

ii $A \in \mathscr{F} \implies A^C \in \mathscr{F}$

iii $\mathscr{F}$ *is closed under countable unions, i.e, that*

$$A_1, A_2, ... \in \mathscr{F} \implies \bigcup_{i=1}^{\infty} A_i \in \mathscr{F}$$

[27]

8.1.2 Bayes Theorem

Theorem *Let $A_1, A_2, ...$ be a partition of Ω, contained in a Borel field $\mathscr{B}$. For $B \in \mathscr{B}$,*

$$Pr(A_i|B) = \frac{Pr(B \cap A_i)}{Pr(B)} = \frac{Pr(B|A_i)Pr(A_i)}{\sum_{i=1}^{\infty} Pr(B|A_i)Pr(A_i)}$$

[27]

8.1.3 Positive Definite Matrix

Definition *A matrix nxnM is positive definite if and only if for every non-zero column vector of z of n real numbers. The scalar $z^T Mz$ is strictly positive* [28].

8.1.4 Cholesky Decomposition

Algorithm *For a symmetric positive definite matrix as defined in 8.1.3 A with elements $(a)_{i,j}$. We can find a lower triangular matrix L with elements $(\ell)_{i,j}$, such that $A = LL^T$ using:[28]*

$$\ell_{1,j} = \frac{1}{\sqrt{a_{1,1}}} a_{j,1}, \qquad j = 1, ..., n$$
$$A^{(1)} = A - \ell_1 \ell_1^T$$
$$\ell_{2,,j} = \frac{1}{\sqrt{a_{2,2}}} a_{j,1}, \qquad j = 1, ..., n$$
$$A^{(2)} = A - \ell_2 \ell_2^T$$
$$... \qquad \text{continue till}$$
$$\ell_{n,j} = \frac{1}{\sqrt{a_{n,n}}} a_{j,n}$$
$$L = (\ell_1, \ell_2, ..., \ell_n)$$

8.1.5 Chapman Kolomogrov

Definition *The equation[2]*

$$f(x_k \| y_{1:k-1}) = \int f(x_k | x_k - 1) f(x_{k-1} | y_{1:k-1}) dx_{k-1} \tag{48}$$

8.1.6 Gaussian Distribution

Definition *Let X be a random variable with PDF*

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},\ for -\infty < x < +\infty$$

X is said to follow a gaussian distribution and we write $X \sim N(\mu, \sigma^2)$. [27]

8.2 Derivation of the EKF

8.2.1 Further assumptions based on the KF derivation

(a) the the state and observe models are differentiable.
(b) $X_t = \psi_\theta(X_{t-1}) + B_t W_t + a_t$, $Y_t = \phi_\theta(X_t) + D_t V_t + c_t$

8.2.2 The Predicted State

$$\begin{aligned}
\hat{X}_{t|t-1} &= E[X_t \mid Y^{t-1}] \\
&= E[\psi_\theta(X_{t-1}) + B_t W_t + a_t \mid Y^{t-1}] \\
&= E[\frac{\psi_\theta(X_{t-1})}{\hat{X}_{t-1|t-1}}(\hat{X}_{t-1|t-1}) a_t \mid Y^{t-1}] + E[B_t W_t \mid Y^{t-1}] + E[a_t \mid Y^{t-1}] \\
&= E[\frac{\psi_\theta(X_{t-1})}{\hat{X}_{t-1|t-1}}(\hat{X}_{t-1|t-1}) a_t \mid Y^{t-1}] + BB^T + a_t
\end{aligned}$$

Let

$$A = \frac{\psi_\theta(X_{t-1})}{\hat{X}_{t-1|t-1}}(\hat{X}_{t-1|t-1})$$

and then we have

$$\hat{X}_{t|t-1} = A\hat{X}_{t-1|t-1} + a_t$$

Hence, (27) is proved.

$$\begin{aligned}
P_{t|t-1} &= cov(X_t - \hat{X}_{t|t-1}) \\
&= cov(\psi_\theta(X_{t-1}) + B_t W_t + a_t - \psi_\theta(\hat{X}_{t-1|t-1})) \\
&= cov(\psi_\theta(X_{t-1}) - \psi_\theta(\hat{X}_{t-1|t-1}) + B_t W_t + a_t) \\
&= cov(\frac{\psi_\theta(X_{t-1}) - \psi_\theta(\hat{X}_{t-1|t-1})}{X_{t-1} - \hat{X}_{t-1|t-1}}(X_{t-1} - \hat{X}_{t-1|t-1})) + cov(B_t W_t) + cov(a_t)
\end{aligned}$$

Let[6]

$$\begin{aligned} A_t &= \frac{\partial \psi_\theta}{\partial X} \mid_{\hat{X}-1|X-1} \\ &= \frac{\psi_\theta(X_{t-1}) - \psi_\theta(\hat{X}_{t-1} \mid t-1)}{X_{t-1} - \hat{X}_{t-1|t-1}} \end{aligned}$$

then

$$\begin{aligned} P_{t|t-1} &= cov(A(x_{t-1} - \hat{X}_{t-1|t-1})) + cov(B_t W_t) + cov(a_t) \\ &= A_t cov(X_{t-1} - \hat{X}_{t-1|t-1}) A_t^T + BB^T \end{aligned}$$

which gives us (28).

8.2.3 The Updated State

$$\begin{aligned} P_{t|t} &= cov(X_t - \hat{X}_{t|t}) \\ &= cov(X_t - \hat{X}_{t|t-1} - K_t) - K_t(\phi_\theta(X_t) - \phi_\theta(\hat{X}_{t|t-1}) - K_t D_t V_t) \\ &= cov(X_t - \hat{X}_{t|t-1} - K_t) - K_t(\phi_\theta(X_t) - \phi_\theta(\hat{X}_{t|t-1})) + cov(K_t D_t V_t) \\ &= cov(X_t - \hat{X}_{t|t-1} - K_t) - K_t(\phi_\theta(X_t) - \phi_\theta(\hat{X}_{t|t-1})) + K_t DD^T K_t^T \\ &= cov((I - K_t \frac{\phi_\theta(X_t) - \phi_\theta(\hat{X}_{t|t-1})}{X_t - \hat{X}_{t|t-1}}(X_t - \hat{X}_{t|t-1})) + K_t DD^T K_t^T \end{aligned}$$

Let

$$\begin{aligned} C_t &= \frac{\partial \phi_\theta}{\partial X} \mid_{\hat{X}|X-1} \\ &= \frac{\psi_\theta(X_t) - \psi_\theta(\hat{X}_{t|t-1})}{X_t - \hat{X}_{t|t-1}} \end{aligned}$$

then

$$\begin{aligned} P_{t|t} &= cov((I - K_t C_t)(X_t - \hat{X}_{t|t-1})) + K_t DD^T K_t^T \\ &= P_{t|t-1} - K_t C_t P_{t|t-1} - P_{t|t-1} C_t^T K_t^T + K_t(C_t P_{t|t-1} C_t^T C_t^T + DD^T) K_t^T \end{aligned}$$

which is just (19) above, then by exactly the same method for KF, we can get (30) and (31) easily. By the same method of KF, we can also get (32), but this time with (29). The further steps are just the repetition of the derivation of KF, and we can finally derive out (28) to (33).

8.3 Derivation of the UKF

8.3.1 The Predicted State

$$\begin{aligned} X_{t-1}^a &= [\hat{X}_{t-1}^T \quad E[W_t^T]]^T \\ P_{t-1}^a &= \begin{bmatrix} P_{t-1} & 0 \\ 0 & BB^T \end{bmatrix} \end{aligned}$$

We form the same matrix χ in equations (35).

Propagate the sigma points to new sigma points through predict function $f(\cdot)$

$$\chi i, t \mid t-1 = f(\chi i, t-1), i = 0, 1, ..., 2L$$

The new sigma points are used to predict $\hat{X}_{t|t-1} and P_{t|t-1}$.

Hence we could derive out (42) and (43) with weights (39)-(41).

8.3.2 The Updated State

$$X^a_{t|t-1} = [\hat{X}^T_{t|t-1} \quad E[V_t^T]]^T$$

$$P^a_{t|t-1} = \begin{bmatrix} P_{t|t-1} & 0 \\ 0 & DD^T \end{bmatrix}$$

Following the exactly same steps as above, propagate the sigma points to new sigma points through measurement function $h(\cdot)$

$$\Upsilon i,t = h(\chi i, t-1), i = 0,1,...,2L$$

The new sigma points are used to observe the estimate $\hat{Y}_t$ and the covirance $P_{y_t y_t}$

$$\hat{Y}_t = \sum_{i=0}^{2L} W_i^{(s)} \Upsilon_{i,t}$$

$$P_{y_t y_t} = \sum_{i=0}^{2L} W_i^{(c)} [\Upsilon_{i,t} - \hat{Y}_t][\Upsilon_{i,t} - \hat{Y}_t]^T \tag{49}$$

where (49) is used to derive equations (44) and (45).

Then by the same concept of linear weighted sum in the proof of the KF, we finally derive out (46) and (47).

8.4 R code

To make the code show a nicer format, here we set μ to be $\hat{X}$ above, Σ to be P above, Q to be BB^T above, and R to be DD^T above.

8.4.1 R code for the Kalman Filter (Section 4.3)

All the code written is based on the code from the reference [10].

```
# kalman function
kalman = function(y, A, B, Q, C, R, mu0, Sigma0) {
  dy = nrow(y)
  T = ncol(y)
  dx = length(mu0)
  I = diag(dx)

  # make placeholder whcih is zero matrix as below;
  mu.p = matrix(0, nrow = dx, ncol = T)
  Sigma.p = array(0, c(dx, dx, T))
  mu.f = matrix(0, nrow = dx, ncol = T)
  Sigma.f = array(0, c(dx, dx, T))

  # predict at time 1
  mu.p[, 1] = A %*% mu0
  Sigma.p[, , 1] = A %*% Sigma0 %*% t(A) + B %*% Q %*% t(B)
  nu = y[, 1] - H %*% mu.p[, 1]
  S = H %*% Sigma.p[, , 1] %*% t(C) + R
  K = Sigma.p[, , 1] %*% t(C) %*% solve(S)
  mu.f[, 1] = mu.p[, 1] + K %*% nu
  Sigma.f[, , 1] = (I - K %*% C) %*% Sigma.p[, , 1]

  # for loop for time 2:T
```

```
  for (t in (2:T)) {
    # Prediction
    mu.p[, t] = F %*% mu.f[, t - 1]
    Sigma.p[, , t] = A %*% Sigma.f[, , t - 1] %*% t(A) + B %*% Q %*% t(B)
    # Update
    nu = y[, t] - C %*% mu.p[, t]
    S = C %*% Sigma.p[, , t] %*% t(C) + R
    K = Sigma.p[, , t] %*% t(C) %*% solve(S)
    mu.f[, t] = mu.p[, t] + K %*% nu
    Sigma.f[, , t] = (I - K %*% H) %*% Sigma.p[, , t]
  }
  return(list(mu.f = mu.f, Sigma.f = Sigma.f))
}
```

```
# example in 1-D
# define A, B, C, D mu0 and Sigma0 in 1 dimension
A = 1
B = 1
Q = 1
C = 1
R = 1
mu0 = 0
Sigma0 = 1

# extract randomly generated data
T = 60
x = matrix(sin(c(1:T)/8), 1, T)
y = matrix(x + rnorm(T, sd = 0.2), nrow = 1, ncol = T)

# compute mean and covariance of the Kalman filtering
results.KF = kalman(y, A, B, Q, C, R, mu0, Sigma0)
mu.f = results.KF$mu.f
Sigma.f = results.KF$Sigma.f

# plot the filtering mean & credible interval
time = 1:60
q_z = qnorm(0.95)
plot(time, y, xlab="time", ylab="", pch=19, col="darkgreen", ylim=c(-3, 3))
points(time, mu.f, col = "red2", pch=20)
legend("topright", c("Simulated", "Estimated"), bty="n",
       pch=c(19, 20), col=c("darkgreen", "red2"))
lines(time, mu.f + q_z * Sigma.f[1,1,], lty=2, lwd=2)
lines(time, mu.f - q_z * Sigma.f[1,1,], lty=2, lwd=2)
```

The code for GPS analysis is based on the reference [18]

```
# example in 2-D
# code for plotting the position in x and y axis
# define coefficients of the models, mu0 and Sigma0 in 2-D
mu0 = rep(0, 2)
Sigma0 = diag(1, nrow =2)
F = diag(1, nrow=2)  # A = F
G = diag(1, nrow=2)  # G = B
Q = diag(1, nrow=2)  # I = Q
H = diag(1, nrow=2)  # C = H
```

```
R = diag(1, nrow=2)  # I = R
T = 60
time = 1:60

# extract randomly generated data
x_1 = matrix(20*(c(1:T)), nrow = 1, ncol = T)
y_1 = matrix(x_1 + rnorm(T, sd = 10), nrow = 1, ncol = T)
x_2 = matrix(20*(sin(c(1:T))+(c(1:T))*sin((c(1:T))/8)), nrow = 1, ncol = T)
y_2 = matrix(x_2 + rnorm(T, sd = 20), nrow = 1, ncol = T)
y = rbind(y_1, y_2)

# compute mean and covariance of the Kalman filtering
results.KF = kalman(y, A, B, Q, C, R, mu0, Sigma0)
mu.f = results.KF$mu.f
Sigma.f = results.KF$Sigma.f

# plot the position of simulated data and estimated data(filtering mean)
plot(y[1,],y[2,], col = "darkgreen", pch=19, xlab="meter", ylab="meter")
points(mu.f[1,], mu.f[2,], col = "red2", pch=20)
legend("topright", c("Simulated", "Estimated"), bty="n",
       pch=c(19, 20), col=c("darkgreen", "red2"))
lines(y[1,],y[2,], col = "darkgreen")
lines(mu.f[1,], mu.f[2,], col = "red2")

# code for plotting the speed
# define A, B, C, D mu0 and Sigma0
R = 1
mu0 = 0
Sigma0 = 3
Q = 1
A = 1
B = 1
C = 1
y = vel
T = 60
time = 1:60

# extract randomly generated data
x_1 = matrix(15*(c(1:T)), nrow = 1, ncol = T)
y_1 = matrix(x_1 + rnorm(T, sd = 10), nrow = 1, ncol = T)
x_2 = matrix(15*(c(1:T)), nrow = 1, ncol = T)
y_2 = matrix(6*x_2 + rnorm(T, sd = 6), nrow = 1, ncol = T)
y = rbind(y_1,y_2)

# make placeholder whcih is zero matrix as below;
spd = matrix(0, nrow=1, ncol=60)
d_1 = matrix(y[,1], nrow=1, ncol=2)  # for t = 1

# for loop for T = 2:60
for (t in (1:T-1)) {
  spd[1,1] = sqrt((d_1[1,1])^2+(d_1[1,2])^2)/5*3600/1000
  n = t
  d = matrix((y[,n+1]-y[,n]),nrow = 2, ncol=1)
  dist = sqrt(d[1,1]^2+d[2,1]^2)
  v = dist/5*3600/1000
  spd[1,t+1] = v
}

# compute mean and covariance of the Kalman filtering
```

```
results.KF = kalman(spd, A, B, Q, C, R, mu0, Sigma0)
mu.f = results.KF$mu.f
Sigma.f = results.KF$Sigma.f

# plot the speed of simulated data and estimated data(filtering mean)
plot(5*time, spd[1,], col="darkgreen", xlab="time (sec)", ylab="speed (km/h)",
     pch=c(19), xlim=c(0, 305), ylim=c(0, 105))
points(5*time, mu.f, col="red", pch=c(20))
lines(5*time, spd[1,], col="darkgreen")
lines(5*time, mu.f, col="red")
legend("bottomright", c("Simulated", "Estimated"),
       bty="n", pch=c(19, 20), col=c("darkgreen", "red2"))
```

8.4.2 R code for the Extended Kalman Filter (Section 5.4)

```
extendedkalmanf = function(y, A, B, Q, C, R, mu0, Sigma0, alpha) {

    library(numDeriv)

    dy = nrow(y)
    T = ncol(y)
    dx = length(mu0)
    I = diag(dx)

    ## p = PREDICTOR / f = KALMAN FILTER / s = SMOOTHED KALMAN FILTER##

    ##INITIALISE##
    mu.p = matrix(0, nrow = dx, ncol = T)
    Sigma.p = array(0, c(dx, dx, T))
    ci.p = matrix(0, nrow = 2, ncol = T)
    mu.f = matrix(0, nrow = dx, ncol = T)
    Sigma.f = array(0, c(dx, dx, T))
    ci.f = matrix(0, nrow = 2, ncol = T)
    mu.s = matrix(0, nrow = dx, ncol = T)
    Sigma.s = array(0, c(dx, dx, T))
    ci.s = matrix(0, nrow = 2, ncol = T)

    ##TIME 1##
    #PREDICTION#
    dA = grad(A,mu0)
    mu.p[, 1] = dA %*% mu0
    Sigma.p[,, 1] = dA %*% Sigma0 %*% t(dA) + B %*% Q %*% t(B)

    #UPDATE#
    dC = grad(C,mu.p[, 1])
    nu = y[, 1] - dC %*% mu.p[, 1]
    S = dC %*% Sigma.p[,, 1] + R
    K = Sigma.p[,, 1] %*% t(dC) %*% solve(S)
    mu.f[, 1] = mu.p[, 1] + K %*% nu
    Sigma.f[,, 1] = (I - K %*% dC) %*% Sigma.p[,, 1]

    ##RECURSION TIME 2:T##
    for (t in (2:T)) {
        #PREDICTION#
        dA = grad(A,mu.f[, t - 1])
```

```
        mu.p[, t] = dA %*% mu.f[, t - 1]
        Sigma.p[,, t] = dA %*% Sigma.f[,, t - 1] %*% t(dA) + B %*% Q %*% t(B)

        #UPDATE#
        dC = grad(C,mu.p[, t])
        nu = y[, t] - dC %*% mu.p[, t]
        S = dC %*% Sigma.p[,, t] + R
        K = Sigma.p[,, t] %*% t(dC) %*% solve(S)
        mu.f[, t] = mu.p[, t] + K %*% nu
        Sigma.f[,, t] = (I - K %*% dC) %*% Sigma.p[,, t]
    }

    ##SMOOTHING/BACKWARD RECURSION##
    mu.s[, T] = mu.f[, T]
    Sigma.s[,, T] = Sigma.f[,, T]
    for (t in (T - 1):1) {
        dA = grad(A, mu.s[, t + 1])
        J = Sigma.f[,, t] %*% t(dA) %*% solve(Sigma.p[,, t + 1])
        mu.s[, t] = mu.f[, t] + J %*% (mu.s[, t + 1] - mu.p[, t + 1])
        Sigma.s[,, t] = Sigma.f[,, t] + J %*% (Sigma.s[,, t + 1] - Sigma.p[,, t + 1])
            %*% t(J)
    }

    ##CREDIBLE INTERVAL##
    #UPPER = 1/LOWER = 2#
    for (t in (1:T)) {
        ci.p[1, t] = qnorm(alpha / 2, mean = mu.p[, t], sd = Sigma.p[1, 1, t])
        ci.p[2, t] = qnorm(1 - alpha / 2, mean = mu.p[, t], sd = Sigma.p[1, 1, t])
        ci.f[1, t] = qnorm(alpha / 2, mean = mu.f[, t], sd = Sigma.f[1, 1, t])
        ci.f[2, t] = qnorm(1 - alpha / 2, mean = mu.f[, t], sd = Sigma.f[1, 1, t])
        ci.s[1, t] = qnorm(alpha / 2, mean = mu.s[, t], sd = Sigma.s[1, 1, t])
        ci.s[2, t] = qnorm(1 - alpha / 2, mean = mu.s[, t], sd = Sigma.s[1, 1, t])
    }

    ##PLOT##
    png(filename = "ekalmanfilter.png", width = 768, height = 1280)
    par(mfrow = c(3, 1))
    #p#
    matplot((1:T),
            t(rbind(mu.p, ci.p,y)),
            type = "l",
            col = c(1, 2, 2,3),
            lty = c(1, 2, 2,3),
            main = "Predictor",
            xlab = "Time",
            ylab = "mu.p",
            cex = 8
            )
    #f#
    matplot((1:T),
            t(rbind(mu.f, ci.f,y)),
            type = "l",
            col = c(1, 2, 2,3),
            lty = c(1, 2, 2, 3),
            main = "Kalman Filter",
            xlab = "Time",
            ylab = "mu.f",
            cex = 8
            )
```

```
    #s#
    matplot((1:T),
            t(rbind(mu.s, ci.s,y)),
            type = "l",
            col = c(1, 2, 2,3),
            lty = c(1, 2, 2,3),
            main = "Smoothed Kalman Filter",
            xlab = "Time",
            ylab = "mu.s",
            cex = 8
            )
    dev.off()


}
```

8.4.3 R code for the Unscented Kalman Filter (Section 6.3)

```
unscentedkalmanf = function(lambda, kappa, y, A, B, Q, C, R, mu0, Sigma0, alpha) {

    dx = length(mu0)
    I = diag(dx)
    dy = nrow(y)
    T = ncol(y)
    N = NROW(mu0)

    ## p = PREDICTOR / f = KALMAN FILTER / s = SMOOTHED KALMAN FILTER##

    ##INITIALISE##
    Wc0 = kappa + lambda / (lambda + T)
    Wc = cbind(matrix(Wc0, nrow = dx, ncol = 1), matrix(1 / (2 * (lambda + dx)), nrow
        = dx, ncol = (2 * dx)))
    Wm0 = lambda / (lambda + T)
    Wm = cbind(matrix(Wm0, nrow = dx, ncol = 1), matrix(1 / (2 * (lambda + dx)), nrow
        = dx, ncol = (2 * dx)))

    ## p = PREDICTOR / f = KALMAN FILTER / s = SMOOTHED KALMAN FILTER##

    mu.p = matrix(0, nrow = dx, ncol = T)
    Yk = array(0, c(dx, (2 * dx + 1), T))
    Yk.p = matrix(0, nrow = dx, ncol = T)
    Xk.p = array(0, c(dx, (2 * dx + 1), T))
    Sigma.p = array(0, c(dx, dx, T))
    ci.p = matrix(0, nrow = 2, ncol = T)
    mu.f = matrix(0, nrow = dx, ncol = T)
    Sigma.f = array(0, c(dx, dx, T))
    ci.f = matrix(0, nrow = 2, ncol = T)
    mu.s = matrix(0, nrow = dx, ncol = T)
    Sigma.s = array(0, c(dx, dx, T))
    ci.s = matrix(0, nrow = 2, ncol = T)

    ##TIME 1##
    #CALCULATE SIGMAPOINTS#
    Qc = chol(Q)
    Sigmapoints = matrix(mu0,nrow = N, ncol = (2*dx+1))
```

```
    for (i in dx) {
        Sigmapoints[, i + 1] = Sigmapoints[, i + 1] + sqrt(dx + lambda) %*% Qc[, i]
        Sigmapoints[, i + dx + 1] = Sigmapoints[, i + dx + 1] + sqrt(dx + lambda) %*%
            Qc[, i]
    }

    #PREDICTION#
    Xk.p[, ,1] = A(1) %*% Sigmapoints
    mu.p[,1] = sum(Wm[,1:(2*dx+1)] * Xk.p[,1,1:(2*dx+1)])
    Sigma.p[,,1] = sum(Wc[, 1:(2 * dx + 1)] * (Xk.p[, 1:(2 * dx + 1),1] - mu.p[, 1]) %
        *% t(Xk.p[, 1:(2 * dx + 1),1] - mu.p[,1])) + Q
    Yk[,,1] = C(1) %*% Xk.p[,,1]
    Yk.p[, 1] = sum(Wm[, 1:(2 * dx + 1)] * Yk[, 1:(2 * dx + 1),1])

    #UPDATE#
    Pyy = sum(Wc[, 1:(2 * dx + 1)] * (Yk[, 1:(2 * dx + 1),1] - Yk.p[,1]) %*% t(Yk[,
        1:(2 * dx + 1),1] - Yk.p[,1])) + R
    Pxy = sum(Wc[, 1:(2 * dx + 1)] * (Xk.p[, 1:(2 * dx + 1),1] - mu.p[, 1]) %*% t(Yk[,
         1:(2 * dx + 1),1] - Yk.p[,1]))
    K = Pxy %*% solve(Pyy)
    mu.f[,1] = mu.p[, 1] + K %*% (y[, 1] - Yk.p[, 1])
    Sigma.f[, ,1] = Sigma.p[,, 1] - K %*% Pyy %*% t(K)

    ##RECURSION TIME 2:T##
    for (t in (2:T)) {
        #CALCULATE SIGMAPOINTS#
        Sigmapoints = matrix(mu.f[,t-1], nrow = N, ncol = (2 * dx + 1))
        for (i in dx) {
            Sigmapoints[, i + 1] = Sigmapoints[, i + 1] + sqrt(dx + lambda) %*% Qc[, i
                ]
            Sigmapoints[, i + dx + 1] = Sigmapoints[, i + dx + 1] + sqrt(dx + lambda)
                %*% Qc[, i]
        }

        #PREDICTION#
        Xk.p[, ,t] = A(t) %*% Sigmapoints
        mu.p[, t] = sum(Wm[, 1:(2 * dx + 1)] * Xk.p[, 1:(2 * dx + 1), t])
        Sigma.p[,, t] = sum(Wc[, 1:(2 * dx + 1)] * (Xk.p[, 1:(2 * dx + 1),t] - mu.p[,
            t]) %*% t(Xk.p[, 1:(2 * dx + 1),t] - mu.p[, t])) + Q
        Yk[, ,t] = C(t) %*% Xk.p[, ,t]
        Yk.p[, t] = sum(Wm[, 1:(2 * dx + 1)] * Yk[, 1:(2 * dx + 1),t])

        #UPDATE#
        Pyy = sum(Wc[, 1:(2 * dx + 1)] * (Yk[, 1:(2 * dx + 1),t] - Yk.p[,t]) %*% t(Yk
            [,  1:(2 * dx + 1),t] - Yk.p[,t])) + R
        Pxy = sum(Wc[, 1:(2 * dx + 1)] * (Xk.p[, 1:(2 * dx + 1),t] - mu.p[, t]) %*% t(
            Yk[, 1:(2 * dx + 1),t] - Yk.p[,t]))
        K = Pxy %*% solve(Pyy)
        mu.f[, t] = mu.p[,t] + K %*% (y[, t] - Yk.p[, t])
        Sigma.f[,, t] = Sigma.p[,, t] - K %*% Pyy %*% t(K)

    }

    ##SMOOTHING/BACKWARD RECURSION##
    mu.s[, T] = mu.f[, T]
    Sigma.s[,, T] = Sigma.f[,, T]
    for (t in (T - 1):1) {
        J = Sigma.f[,, t] %*% t(F(t)) %*% solve(Sigma.p[,, t + 1])
        mu.s[, t] = mu.f[, t] + J %*% (mu.s[, t + 1] - mu.p[, t + 1])
```

```
        Sigma.s[,, t] = Sigma.f[,, t] + J %*% (Sigma.s[,, t + 1] - Sigma.p[,, t + 1])
            %*% t(J)
    }

    ##CREDIBLE INTERVAL##
    #UPPER = 1/LOWER = 2#
    for (t in (1:T)) {
        ci.p[1, t] = qnorm(alpha / 2, mean = mu.p[, t], sd = Sigma.p[1, 1, t])
        ci.p[2, t] = qnorm(1 - alpha / 2, mean = mu.p[, t], sd = Sigma.p[1, 1, t])
        ci.f[1, t] = qnorm(alpha / 2, mean = mu.f[, t], sd = Sigma.f[1, 1, t])
        ci.f[2, t] = qnorm(1 - alpha / 2, mean = mu.f[, t], sd = Sigma.f[1, 1, t])
        ci.s[1, t] = qnorm(alpha / 2, mean = mu.s[, t], sd = Sigma.s[1, 1, t])
        ci.s[2, t] = qnorm(1 - alpha / 2, mean = mu.s[, t], sd = Sigma.s[1, 1, t])
    }

    ##PLOT##
    png(filename = "ukalmanfilter.png", width = 768, height = 1280)
    par(mfrow = c(3, 1))
    #p#
    matplot((1:T),
            t(rbind(mu.p, ci.p, y)),
            type = "l",
            col = c(1, 2, 2, 3),
            lty = c(1, 2, 2, 3),
            main = "Predictor",
            xlab = "Time",
            ylab = "mu.p",
            cex = 8
            )
    #f#
    matplot((1:T),
            t(rbind(mu.f, ci.f, y)),
            type = "l",
            col = c(1, 2, 2, 3),
            lty = c(1, 2, 2, 3),
            main = "Kalman Filter",
            xlab = "Time",
            ylab = "mu.f",
            cex = 8
            )
    #s#
    matplot((1:T),
            t(rbind(mu.s, ci.s, y)),
            type = "l",
            col = c(1, 2, 2, 3),
            lty = c(1, 2, 2, 3),
            main = "Smoothed Kalman Filter",
            xlab = "Time",
            ylab = "mu.s",
            cex = 8
            )
    dev.off()



}
```

code uses the algorithm presented in reference [9]

9 References

References

[1] N. Kantas. Sequential Monte Carlo Notes. Imperial College London.

[2] S. Sarkka. *Bayesian filtering and smoothing*. CUP Cambridge, 2013.

[3] MathWorks. What Are State-Space Models? Available from: https://uk.mathworks.com/help/ident/ug/what-are-state-space-models.html#brbu1g0[Accessed 17th June 2018.

[4] MathWorks Understanding Kalman Filters Available from: https://uk.mathworks.com/videos/series/understanding-kalman-filters.html[Accessed 17th June 2018].

[5] Ian Reid. Estimation II. Hilary Term, 2001. Available from: http://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf [Accessed 17th June 2018].

[6] Knowsky. Explanation of KF, EKF and UKF. Available from: http://www.knowsky.com/1054652.html [Accessed 17th June 2018].

[7] Michael Bensimhoun (reviewed by David Arnon). N-dimensional cumulative function, and other useful facts about gaussians and normal densities. Available from: https://upload.wikimedia.org/wikipedia/commons/a/a2/Cumulative_function_n_dimensional_Gaussians_12.2013.pdf. [Accessed 17th June 2018]

[8] Ramsey Faragher. Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation. *IEEE SIGNAL PROCESSING MAGAZINE* , September 2012, pp.132, available from: https://www.cl.cam.ac.uk/~rmf25/papers/Understanding%20the%20Basis%20of%20the%20Kalman%20Filter.pdf [Accessed 17th June 2018].

[9] Eric A. Wan and Rudolph van der Merwe. Chapter 7, The Unscented Kalman Filter. Available from: https://www.pdx.edu/biomedical-signal-processing-lab/sites/www.pdx.edu.biomedical-signal-processing-lab/files/ukf.wan_.chapt7_.pdf [Accessed 17th June 2018].

[10] Francois Caron. *Computational Statistics - Hidden Markov Models* April 2018.

[11] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[12] K.P. Murphy. *Machine Learning. A Probabilistic Perspective*. The MIT Press, 2012.

[13] Mohinder S, Angus P. *Kalman Filtering: Theory and Practice* $2nd$ ed. John Wiley & Sons Publishing, 2001.

[14] Florian Wilhelm. Handling GPS Data with Python. Jul 2016. Available from: https://ep2016.europython.eu//conference/talks/handling-gps-data-with-pythonhttps://www.youtube.com/watch?v=9Q8nEA_0ccg&t=2239s [Accessed 8th June 2018].

[15] Boaz Sobrado. Implementing a Kalman filter in R for Android GPS measurements. Aug 2017. Available from: http://boazsobrado.com/blog/2017/08/29/implementing-a-kalman-filter-in-r-for-android-gps-measurements/ [Accessed 10th June 2018].

[16] Yoshiki Takeuchi. Optimization of Linear Observations for the Stationary Kalman-Bucy Filter. *Transactions of the Institute of Systems, Control and Information Engineers*, 2015, Vol. 28, No. 12, pp. 467-475.

[17] Durbin, J. and Koopman, S. J. *Time Series Analysis by State Space Methods*. 2*nd* ed. Oxford University Press, 2001.

[18] Yeon-woo Jang. *Computational Statistics*. Oxford University, Practical no.215, 2018.

[19] Alexandros Beskos, Dan Crisan, Ajay Jasra, Kengo Kamatani, Yan Zhou. A stable particle filter for a class of high-dimensional state-space models,*Advances in Applied Probability, Applied Probability Trust, Vol 49, Issue 1*, pp24-48. Available from: `https://spiral.imperial.ac.uk:8443/bitstream/10044/1/49663/2/stable.pdf` [Accessed 17th June 2018].

[20] Huang, Guoquan P; Mourikis, Anastasios I; Roumeliotis, Stergios I. Analysis and improvement of the consistency of extended Kalman filter based SLAM. *Robotics and Automation,ICRA 2008. IEEE International Conference*, 2008, pp. 473–479.

[21] Soken, H.E., Hajiyev, C. Pico Satellite Attitude Estimation via Robust Unscented Kalman Filter in the Presence of Measurement Faults. ISA Transactions. 2010; 49: 249–256.

[22] Markley, F.L., Crassidis, J.L., Cheng, Y. Nonlinear Attitude Filtering Methods.. *AIAA Guidance, Navigation, and Control Conference and Exhibit, San Francisco*, 2005.

[23] Lefferts, E.J., Markley, F.L., Shuster, M.D. Kalman Filtering for Spacecraft Attitude Estimation. *Journal of Guidance, Control and Dynamics*, 1982, 417–422.

[24] Mehra, R.K. Approaches to Adaptive Filtering. *IEEE Transactions on Automatic Control 17*,1972, 693–698.

[25] Julier, S.J., Uhlmann, J.K., Durrant-Whyte, H.F. A New Approach for Filtering Nonlinear Systems. *American Control Conference, Seattle, vol. 3*,1995, pp. 1628–1632.

[26] Liu, J., Lu, M. An Adaptive UKF Filtering Algorithm for GPS Position Estimation..In: 5th International Conference on Wireless Communications, Networking and Mobile Computing, Beijing, 2009, pp. 1–4.

[27] David A van Dyk, Chris Hallsworth. *Probability and Statistics ii*. Imperial College London, October 2017. Available from: `https://bb.imperial.ac.uk/bbcswebdav/pid-1215706-dt-content-rid-3935374_1/courses/DSS-M2S1-17_18/notes%282%29.pdf` [Accessed 17th June 2018].

[28] Robert Nurnberg. *Intro to Numerical Analysis*. Imperial College London, 2018. Available from: `https://bb.imperial.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_12705_1&content_id=_1256767_1` [Accessed 17th June 2018].