A Particle filter Using SVD Based Sampling Kalman Filter to Obtain the Proposal Distribution

Bin Liu  
Graduate University, The Chinese Academy of Sciences  
Institute of Acoustics, The Chinese Academy of Sciences  
Beijing, China  
binliu1981@yahoo.com

Xiao-chuan Ma, Chao-huan Hou  
Institute of Acoustics, The Chinese Academy of Sciences  
Beijing, China  
{maxc,hch}@mail.ioa.ac.cn

Abstract—In this paper, we propose a novel particle filter (PF), which uses a bank of singular-value-decomposition based sampling Kalman filters (SVDSKF) to obtain the importance proposal distribution. This proposal has two properties. Firstly, it allows the particle filter to incorporate the latest observations into a prior updating routine and, secondly it inherits advantage of having good numerical stability from the singular-value-decomposition (SVD). The convergence results of the numerical simulations we made confirm that the proposed PF method outperforms the standard bootstrap PF as well as other local linearization based PFs.

Keywords—Particle Filter, SVD, proposal distribution, SRUKF

I. INTRODUCTION

To solve the problem of nonlinear problems, the best known filtering algorithms are EKF (extended Kalman filter)\cite{1,2} and UKF (unscented Kalman filter) \cite{1,3-8}. Furthermore, particle filters (PFs) are proposed to solve the problem of non-Gaussian\cite{8-12}. PFs rely on importance sampling techniques which lead to the requirement for the design of proposal distributions in order to approximate the posterior distribution reasonably well. In general, it is hard to design such proposals. The most common strategy is to sample from the probabilistic model of the states evolution (transition prior) which results in the generation of standard bootstrap PF. However, this strategy is easy to fail if the new measurements appear in the tail of the prior or if the likelihood is too peaked in comparison to the prior. These situations do indeed arise in several areas of engineering and finance, where one can encounter sensors that are very accurate (peaked likelihoods) or data that undergo sudden changes (nonstationarities)\cite{12}. To overcome this problem, researchers proposed to use local linearization techniques, such as EKF or UKF, to generate the proposal distribution of PFs\cite{12}. However, EKF has its inherent drawbacks due to its first-order linearization and UKF and its basis untransformed transform (UT) often encounter the ill-conditioned problem of covariance matrix in practice (though it is theoretically positive semi-definite)\cite{8}.

The singular-value-decomposition (SVD) concerns the factorization of an arbitrary matrix $A$ into a product $UDV^T$ of orthogonal matrices $U$ and $V$ and a “diagonal” matrix $D$. It’s applied frequently in numerical linear algebra and proved to be a robust method for solving ill-conditioned least squares problems\cite{13-15}. Youmin Zhang proposed a SVD based EKF with a good numerical stability in application to aircraft flight state and parameter estimation\cite{5}. Z.Chen presented a SVD sampling Kalman filter (SVDSKF) in \cite{8} which is similar as UKF but can avoid the ill-condition problems that may be encountered when using UKF.

In this paper, we use the SVDSKF to obtain the importance proposal distribution of the PF. Then we get a novel PF method.

II. DYNAMIC STATE SPACE MODEL

Since we are interested in nonlinear, non-Gaussian regression, the state space model can be expressed as follows

$$x_i = f(x_{i-1}) + v_{i-1}$$
$$y_i = h(x_i) + n_i,$$

where $x_i \in \mathbb{R}^n$ denotes the unobserved states (or parameters) of the model, $y_i \in \mathbb{R}^m$ the observations, $v_i \in \mathbb{R}^k$ the process noise and $n_i \in \mathbb{R}^m$ the measurement noise. To complete the specification of the model, the prior distribution (at $t = 0$) is denoted by $p(x_0)$. Our goal will be to approximate the posterior distribution $p(x_t \mid y_{1:t})$ and one of its marginals, the filtering density $p(x_t \mid y_{1:t})$, where $y_{1:t} = \{y_1, y_2, \ldots, y_t\}$. By computing the filtering density recursively, we do not need to keep track of the complete history of the states.

III. GENERIC PARTICLE FILTER

For completeness, we present a generic PF algorithm here, which involves the following steps as in table I. PFs allow us to approximate the posterior distribution $p(x_{1:t} \mid y_{1:t})$ using a set of $N$ weighted samples (particles) $\{x_{i}^{(n)}; i = 1, \ldots, N\}$, which are drawn from an importance proposal distribution $q(x_{t} \mid y_{1:t})$. In framework of the PF algorithm, we can restrict ourselves to importance functions of the form \cite{9}

$$q(x_{t} \mid y_{1:t}) = q(x_{t}) \prod_{i=1}^{N} q(x_{t} \mid y_{1:t}, x_{t-1})$$

(2)

to obtain a recursive formula to evaluate the importance weights\cite{9}

$$w_j \propto \frac{p(y_t \mid x_{t+1}, \{x_{t}^{(n)}\}) p(x_{t}^{(n)} \mid x_{t})}{q(x_{t}^{(n)} \mid y_{1:t}, x_{t-1})}$$

(3)
1. Sequential importance sampling step
   - For $i = 1, \ldots, N$, sample $\tilde{x}_{t,i} = q(x_t | x_{t-1,i}, y_t)$ and update the trajectories $x_{t,i} = x_{t-1,i}$
   - For $i = 1, \ldots, N$, evaluate the weights $w_{t,i}^{(i)} = \frac{p(x_{t,i}^{(i)})}{q(x_{t,i}^{(i)} | x_{t-1,i}, y_t)}$
   - For $i = 1, \ldots, N$, normalize the weights $w_{t,i}^{(i)} = \frac{w_{t,i}^{(i)}}{\sum_{i=1}^{N} w_{t,i}^{(i)}}$

2. Selection step
   - Multiply/suppress samples $\left(\tilde{x}_{t,i}, w_{t,i}^{(i)}\right)$ with high/low importance weights $w_{t,i}^{(i)}$, respectively, to obtain $N$ random samples $\left(x_{t,i}^{(i)}\right)$ approximately distributed according to $p(x_t | y_t)$
   - For $i = 1, \ldots, N$, set $w_{t,i}^{(i)} = 1/N$

3. Output:
   - The output of the algorithm is a set of samples that can be used to approximate the posterior distribution as follows:
     $p(x_t | y_t) = \hat{p}(x_t | y_t) = \frac{1}{N} \sum_{i=1}^{N} \delta(x_{t,i}^{(i)}) (dx_{t,i}^{(i)})$
   - One obtains straightforwardly the following estimate of $E(g_t(x_{t,i}^{(i)}), E(g_t(x_{t,i}^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} g_t(x_{t,i}^{(i)})$.

There are infinitely many possible choices for $q(x_{t,i}^{(i)} | y_t)$, the only condition being that its support must include that of $p(x_{t,i}^{(i)} | y_t)$. The simplest choice is to just sample from the prior, $p(x_t | y_t)$, in which case the importance weights is equal to the likelihood, $p(y_t | x_{t,i}^{(i)}, x_{t-1,i})$. This is the most widely used distribution and it leads to the most general bootstrap PF[9]. This proposal is simple to compute but also can be inefficient since it ignores the most recent evidence, $y_t$. The selection (resampling) step in Table I is used to eliminate the particles having low importance weights and to multiply particles having high importance weights[9,12].

IV. SRUKF BASED PF

Compared with the original formulation of the UKF, the SRUKF has added benefit of numerical stability[7]. A PF using SRUKF as its proposal distributions has been proposed in [16]. The SRUKF based PF is a candidate algorithm for performance comparison with the proposed SVDSKF based PF.

V. SVDSKF BASED PF

The SVDSKF is close in spirit to UKF[8]. Similarly to the implementations of PFs based on EKF or UKF[12], we propose the SVDSKF based PF as follows in Table II.

<table>
<thead>
<tr>
<th>TABLE I. GENERIC PF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initialization: $t = 0$</td>
</tr>
<tr>
<td>- For $i = 1, \ldots, N$, draw the states (particles) $x_{t,i}^{(i)}$ from the prior $p(x_t</td>
</tr>
<tr>
<td>$x_{0}^{(i)} = E[x_{0}^{(i)}]$</td>
</tr>
<tr>
<td>$P_{0}^{(i)} = E[(x_{0}^{(i)} - x_{0}) (x_{0}^{(i)} - x_{0})^T]$</td>
</tr>
<tr>
<td>$x_{0}^{(i)} = E[x_{0}^{(i)}]$ = $\left[\left(x_{0}^{(i)} - x_{0}\right) \left(x_{0}^{(i)} - x_{0}\right)^T\right]$</td>
</tr>
<tr>
<td>$P_{0}^{(i)} = E[\left(x_{0}^{(i)} - x_{0}\right) \left(x_{0}^{(i)} - x_{0}\right)^T]$</td>
</tr>
<tr>
<td>$\begin{bmatrix} P_{0}^{(i)} &amp; 0 &amp; 0 \ 0 &amp; Q &amp; 0 \ 0 &amp; 0 &amp; R \end{bmatrix}$</td>
</tr>
<tr>
<td>where $Q$ = process noise cov, $R$ = measurement noise cov</td>
</tr>
<tr>
<td>2. For $t = 1, 2, \ldots$</td>
</tr>
<tr>
<td>(a) Importance sampling step</td>
</tr>
<tr>
<td>- Update the particles with the SVD-based KF:</td>
</tr>
<tr>
<td>* Compute the SVD and eigen-point covariance matrix</td>
</tr>
<tr>
<td>$P_{v,t}^{(i)} = USV^T$</td>
</tr>
<tr>
<td>$\chi_{t,j,t}^{(i)} = x_{t,j,t}^{(i)}$</td>
</tr>
<tr>
<td>$\chi_{j,t,j,t}^{(i)} = x_{t,j,t}^{(i)} + \rho U_{j} \sqrt{s_{j,t}}, j = 1, \ldots, n_{z}$</td>
</tr>
<tr>
<td>$\chi_{j,t,j,t}^{(i)} = x_{t,j,t}^{(i)} - \rho U_{j} \sqrt{s_{j,t}}, j = n_{z} + 1, \ldots, 2n_{z}$</td>
</tr>
<tr>
<td>where $s_{j}$ is the $j$th diagonal element of $S$.</td>
</tr>
<tr>
<td>* Propagate particles into future (time update)</td>
</tr>
<tr>
<td>$x_{t+1,j,t}^{(i)} = f(\chi_{t+1,j,t}^{(i)}, \chi_{t,j,t}^{(i)})$</td>
</tr>
<tr>
<td>$x_{t,j+1,t}^{(i)} = x_{t,j,t}^{(i)} + \sum_{j=0}^{2n_{z}} w_{j}^{(i)} \chi_{j,t,j+1,t}^{(i)}$</td>
</tr>
<tr>
<td>$P_{t,j+1,t}^{(i)} = \sum_{j=0}^{2n_{z}} w_{j}^{(i)} \left[\chi_{j,t,j+1,t}^{(i)} - x_{t,j+1,t}^{(i)}\right] \left[\chi_{j,t,j+1,t}^{(i)} - x_{t,j+1,t}^{(i)}\right]^T$</td>
</tr>
<tr>
<td>$y_{t,j+1,t}^{(i)} = h(\chi_{t,j,t}^{(i)}, \chi_{t,j+1,t}^{(i)})$</td>
</tr>
<tr>
<td>$y_{t,j+1,t}^{(i)} = \sum_{j=0}^{2n_{z}} w_{j}^{(i)} \chi_{j,t,j+1,t}^{(i)}$</td>
</tr>
</tbody>
</table>

TO BE CONTINUED
is a scaling parameter \( \beta \) for the following process model:

\[ \dot{x}_t = f(x_{t-1}, w_t), \quad y_t = h(x_t) + v_t, \]

\( w_t \) is draw from a Gaussian distribution \( N(0, \sigma^2) \).

\( y_t \) is a random variable modeling the process noise, \( w_t \) is a \( Gamma(3,2) \) random variable modeling the process noise, and \( \phi \) is a scalar parameter.

For this experiment, a time-series is generated by the following process model:

\[ x_{t+1} = 1 + \sin(w_{t+1}) + \phi x_t + v_t, \]

where \( v_t \) is a \( Gamma(3,2) \) random variable modeling the process noise, and \( w = 4 - 2 \) are scalar parameters.

VI. SIMULATION

For this experiment, a time-series is generated by the following process model:

\[ x_{t+1} = 1 + \sin(w_{t+1}) + \phi x_t + v_t, \]

where \( v_t \) is a \( Gamma(3,2) \) random variable modeling the process noise, and \( w = 4 - 2 \) are scalar parameters.

**Parameter and weights of the SVD-based KF module:**

\[ \begin{align*}
\mu_0^{(a)} &= \frac{\kappa}{n_{\alpha} + \kappa}, \\
\mu_j^{(a)} &= \frac{1}{2(n_{\alpha} + \kappa)}(j \neq 0), \\
\mu_0^{(c)} &= \frac{\kappa \cdot n_{\alpha}}{1.96 \times (n_{\alpha} + \kappa)}, \\
\mu_j^{(c)} &= \frac{n_{\alpha}}{1.96 \times 2(n_{\alpha} + \kappa)}(j \neq 0),
\end{align*} \]

where \( \kappa \) is a small tuning parameter.

**Selection step**

Multiply/Suppress particles \((x_{0t}, P_{0t})\) with high/low importance weights \( \tilde{w}_t \), respectively, to obtain \( N \) random particles \((x_{0t}, P_{0t})\).

**Output**

The output is generated in the same manner as for the generic particle filter.

**TABLE III. The mean and variance of the MSE calculated over 100 independent runs**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MSE mean</th>
<th>MSE variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootstrap PF</td>
<td>0.357</td>
<td>0.055</td>
</tr>
<tr>
<td>EKF based PF</td>
<td>0.315</td>
<td>0.018</td>
</tr>
<tr>
<td>SRUKF based PF</td>
<td>0.320</td>
<td>0.069</td>
</tr>
<tr>
<td>SVDSKF based PF (proposed)</td>
<td>0.167</td>
<td>0.015</td>
</tr>
</tbody>
</table>

VII. CONCLUSIONS

This paper proposes a new PF algorithm. It incorporates a robust numerical method, i.e., SVD, into the framework of PF. Meanwhile, it incorporates the latest observations into a prior updating routine of PF. Both of the two characters guarantee this method’s superiority to the generic bootstrap PF algorithm. Furthermore, simulation results show that the proposed algorithm also has better estimation performance than other existing similar PF algorithms that are based on local linearization techniques such as the EKF and SRUKF. However, the proposed method needs some more computing costs than the others mentioned above.
REFERENCES


Figure 2. Comparison of computing burdens indexed by average computing time in a simulation including 100 independent runs of each PF algorithm respectively.