Web Based Particle Filters

by

Xingpu Wang

A thesis submitted in partial fulfillment of the requirements for the degree of

Masters of Science

in

Statistics

Department of Mathematical and Statistical Sciences

University of Alberta

Abstract

In this thesis, we first introduce two basic problems of filter, the nonlinear filtering and model selection problem. We show that both of them can be solved by the unnormalized filter approach. Then several web based particle filter algorithms will be discussed. We extend the resampled and branching system on single computer platform to a web based platform. The performance and execution time of these algorithms will be compared upon two simulation models. We define a parameter, called "Bootstrap Factor", which is a reasonable way to compare different particle filters. By Bootstrap Factor, we show that the web based branching system performs much better than the double resampled system.

Acknowledgment

I would like to express my gratitude to all those guys who gave me help to complete this project and thesis. I am deeply thankful to my supervisor Dr. Michael Kouritzin. He gave a lot of valuable suggestions on programming the model and writing this thesis.

I am also thankful to my classmates Chi and Huiting. They helped me a lot on theoretical questions and correcting the code.

Finally, thanks for my parents. They give me the chance and support me to complete this project.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In section 1.1, we will review the history and discuss the current applications of particle filters. In section 1.2, two main problems, the tracking and model selection, of particle filters will be described. We will show how to use the unnormalized filter approach to solve these two problems. In section 1.3, the ideas of numeric approximation and the computer implementation will be briefly explained. Finally in section 1.4, we will show the main contributions and outline of this thesis.

1.1 History and Current Application

The filtering theory was first proposed by Norbert Wiener during the Second World War, in the 1940s (see [1]). The main objective of the Wiener filter is to estimate an unknown *linear time-invariant* signal by inputting and filtering a related signal to output the estimation. The Wiener filter mainly deals with the case that the time is continuous. The discrete-time version of Wiener filtering was derived by Andrey Kolmogorov in 1941 thus the theory is also called the Wiener-Kolmogorov filtering theory (see [2]). It is a frequency-based approach and as such can not be used effectively in real time applications. Nevertheless, it is used in communications and motivated many other filter theories, including the very famous Kalman filter.

The Kalman filter is named after Rudolf Emil Kalman, who was the inventor and developer of the Kalman filter. Richard S. Bucy also contributed to the theory thus the filter is also called the Kalman-Bucy filter. This filter was first described and published in technical papers by Swerling (1958), Kalman (1960) and Kalman and Bucy (1961) separately (see [3], [4], [5]). It deals with the linear Gaussian dynamic system and the main objective is to estimate the internal state of the system from a series of noisy measurements. It is a recursive estimator which means that to estimate the current state, one only needs to know the state of the previous time step and the current measurement. The most famous application of Kalman filter is that it was used in the Apollo program to help send rockets to the moon.

Following after Kalman's work, Duncan, Mortensen and Zakai, from 1967 to 1969, considered the non-linear filtering problem and independently developed an equation for the unnormalized conditional probability density function of the nonlinear filtering problem, which is known as Duncan-Mortensen-Zakai equation (see [6]).

Stratonavich was the first to develop the probabilistic approach to nonlinear filtering problem in Russia while Kalman and Bucy were formulating the Kalman filter theory in United States, in 1959 (see [7]). Kushner developed the continuous nonlinear filter theory independently and a lot of subsequent work in nonlinear filtering field was done by him in the 1970s, including the Kushner-Stratonovich equation, which characterises the evolution of the conditional distribution of nonlinear filtering (see [8]). In 1972, Fujisaki, Kallianpur and Kunita rigorously developed the theory of stochastic differential equations for the nonlinear filtering problem. Their work was so important that he Kushner-Stratonovich equation is often called the Fujisaki-Kallianpur-Kunita equation (see [9]).

In the meantime, Kallianpur collaborated with Charlotte Striebel on the stochastic differential equations which occurs in the estimation of the continuous parameter stochastic processes in 1969. They developed a Bayes formula to convert the unnormalized filter to the normalized one and their work is known as the Kallianpur-Striebel formula (see [10]).

The Kushner-Stratonovich and Duncan-Mortensen-Zakai equations are multiply infinite dimensional and hence not implementable on a computer. However, in 1970 Handschin and Mayne proposed the sequential Monte Carlo method to implement an approximation of nonlinear filters on a computer (see [11]). Their solution has become known as *weighted particle filter* and it is still used today even though it often suffers from particle spread.

Then in 1993, Gordon et al. proposed the first resampled particle filter that would move particles with low weights to locations of high weights (in an unbiased manner) thus avoiding particle spread (see [12]). They named their algorithm the *bootstrap filter*. There is no assumption about the state-space nor the noise of the system for either the weighted or bootstrap algorithms. The noise can be non-Gaussian and the signal equation nonlinear.

Nowadays, there are variety of exciting applications of the particle filters in wide fields, including the defense, communications, aerospace, econometrics, medical imaging, clickstream analysis, earthquake data analysis and weather prediction. In the defense field, one may use the particle filter to track an aircraft or submarine by the information obtained from radar or sonar. In the communications field, one may use the particle filter to eliminate the noise in signal transmission. In finance, particle filters are used to predict stock price or estimate volatility. For the clickstream analysis, particle filters can characterize the user and select the most appropriate ad or adapt a website to optimize the user experience. However, the current driving application of particle filter research is weather prediction.

1.2 Problem Description

1.2.1 Nonlinear Filtering

Nonlinear filtering deals with estimating the current state of a non-observable signal X based on the history of a distorted, corrupted partial observation process Y living on the same probability space $(\Omega, \mathcal{F}, P)$ as X. In many practical problems the signal model lives in some complete, separable metric space (E, ρ) and is a time-homogeneous discrete-time Markov process while the observations are finite dimensional. Thus, in this thesis, we take the signal and observation model as the following:

$$SignalModel : P\left(X_k \in A \,\middle|\, \mathcal{F}^X_{k-1}\right) = K\left(X_{k-1}, A\right), \quad X_0 \sim \pi_0, \tag{1.1}$$

$$ObservationModel : Y_k = h(X_{k-1}) + V_k, \quad Y_0 = 0, \tag{1.2}$$

where K is the transition kernel of the signal, $\mathcal{F}^X_{k-1}$ is the σ-algebra which contains all the information from the signal up to previous time and h is the sensor function, which is a measurable mapping from E to $\mathbb{R}^d$. $\{V_k\}$ is an independent sequence of random variables each with a bounded, strictly positive density function g and π_0 is the distribution for initial value of signal.

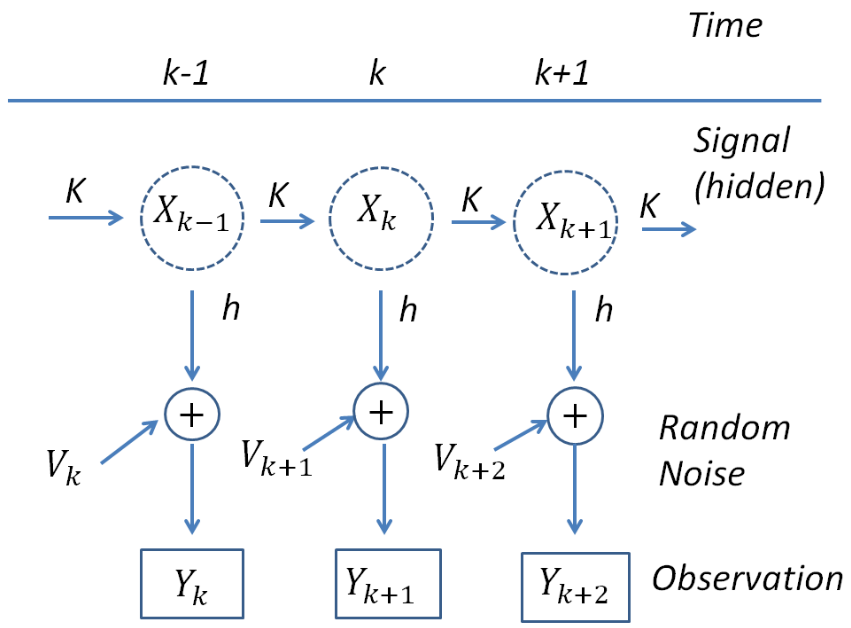


Figure 1: Signal and Observation Model

Figure 1 is a visualization of a signal and observation model. The signal, which is in the dashed circle, is a hidden Markov process that mutates by the transition kernal K. At each time step, the model outputs an observation in the solid rectangle, which consists of a distorted (by the sensor function h) signal measurement that is corrupted by random independent noise.

The main objective of the nonlinear filtering problem is to calculate the conditional probabilities:

$$\pi_k(A) = P\left(X_k \in A \left| \mathcal{F}_k^Y \right.\right), k = 1, 2, ..., \tag{1.3}$$

for all Borel sets A where $\mathcal{F}_k^Y \doteq \sigma\{Y_l, l = 1, ..., k\}$ contains all the information from the previous observations, or equivalently, to compute the conditional expectations:

$$\pi_k(f) = E^P\left(f(X_k) \left| \mathcal{F}_k^Y \right.\right), k = 1, 2, ..., \tag{1.4}$$

for all bounded and measurable functions f mapping from E to $\mathbb{R}$. In the sequel, we refer to $\{\pi_k\}$ as the filter or the filter process.

Example 1.1. The signal and observation models are defined by:

$$\begin{aligned} X_k &= 0.95X_{k-1} + 0.3W_k \\ Y_k &= X_{k-1} + V_k, \end{aligned}$$

where X_0, $\{W_k\}$ and $\{V_k\}$ are independent with standard Cauchy distribution. It is a non-Gaussian problem. In Figure 2, we use the residual branching particle filter (to be explained in the sequel) to track the signal X_k. The main objective of this example is to show how a particle filter works on tracking an unknown signal using the observations. From the picture, it is obvious that our filter is much closer to the true signal than the observations. In another word, the filter uses the information from observations but eliminates some of the noise.

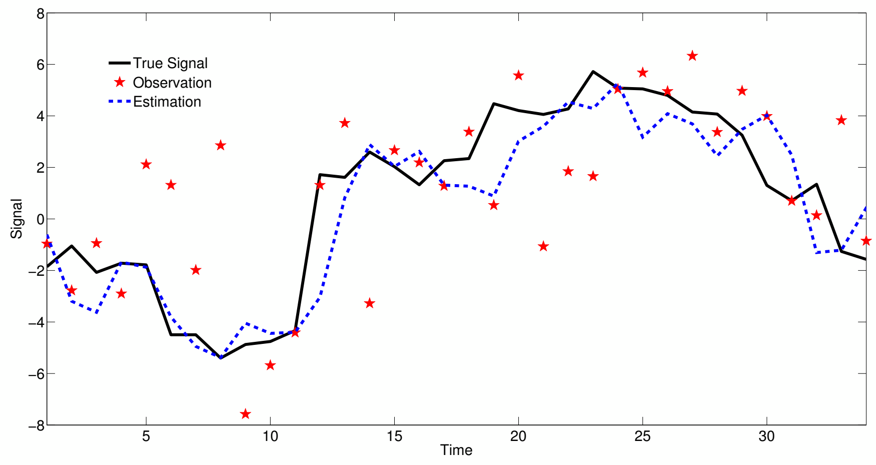


Figure 2: Example of the nonlinear filtering problem

1.2.2 Model Selection

Most often, the transition kernel K in the signal model (1.1) is unknown. In these cases, we can only guess the model by our experience or some physical and chemical laws. However, there may be a number of possible models so we need to choose the most likely one based upon the data and this is the model selection problem. For example, in tracking financial data we may have different stochastic volatility models, knowing any of them may perform best for a period of time. Even if the model is known in form up to an unknown parameter, we can still treat the model form with different parameters as different models and use the model selection to identify the parameter.

The Bayesian model selection deals with determining which of a class of signal model $\{x^{(i)}\}_{i\in I}$ best fits the back observations $\{Y_1, Y_2, ..., Y_k\}$ by comparing pairwise Bayes' factors, like:

$$B_k^{12} = \frac{E^Q[l_k(Y|x^{(1)})|\mathcal{F}_k^Y]}{E^Q[l_k(Y|x^{(2)})|\mathcal{F}_k^Y]}, k = 1, 2, ..., \tag{1.5}$$

where Q is a reference probability, Y is the pure noise with respect to Q and $l_k(Y|x^{(i)}), k = 1, 2, ...$ is the likelihood process that turns $Y_{k+1} = V_{k+1}$ into the observations $Y_{k+1} = h(x_k^{(i)}) + V_{k+1}$ for each i.

We provide a simple example to demonstrate Bayesian model selection.

Example 1.2. Consider five different signal models with the same observation:

Model Number	Signal Equation	Observation Equation
-2	$X_k = 0.5X_{k-1} + 0.3W_k$	$Y_k = X_{k-1} + V_k$
-1	$X_k = 0.6X_{k-1} + 0.3W_k$	$Y_k = X_{k-1} + V_k$
0	$X_k = 0.7X_{k-1} + 0.3W_k$	$Y_k = X_{k-1} + V_k$
1	$X_k = 0.8X_{k-1} + 0.3W_k$	$Y_k = X_{k-1} + V_k$
2	$X_k = 0.9X_{k-1} + 0.3W_k$	$Y_k = X_{k-1} + V_k$

X_0, $\{W_k\}$ and $\{V_k\}$ are independent with standard Cauchy distribution.

Model (0) is the true model and others are incorrect guesses. The main objective is to select the correct model by minimizing the final Bayes Factor:

$$B_k^{0,i} = \frac{E^Q[l_k(Y|x^{(0)})|\mathcal{F}_k^Y]}{E^Q[l_k(Y|x^{(i)})|\mathcal{F}_k^Y]}, k = 1, 2, ..., i = -2, -1, 0, 1, 2 \qquad (1.6)$$

In this way, Model (i) is more likely to be the true model than Model (j) if $B_k^{0,i}$ is closer to 1 than $B_k^{0,j}$ and we show the result in Figure 3. It shows that Model (-1) and Model (1) are more closer to the true model, which is Model 0.

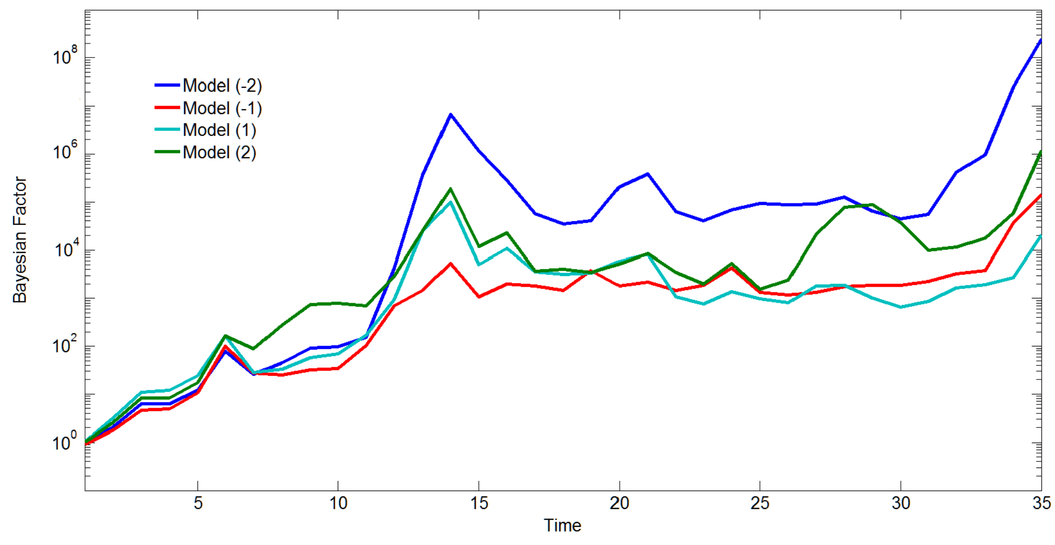


Figure 3: Model Selection Example

1.2.3 Unnormalized Filter Approach

The nonlinear filtering and model selection problems both can be solved simultaneously by the unnormalized filter approach.

By the notion of the unnormalized filter and the Bayes rule, we can express the conditional expectation $\pi_k(f)$, or equivalently the normalized filter by:

$$\pi_k(f) = \frac{\sigma_k(f)}{\sigma_k(1)}, k = 1, 2..., \tag{1.7}$$

where $\sigma_k(f)$ is the unnormalized filter defined by:

$$\sigma_k(f) = E^Q \left(l_k f\left(X_k\right) \middle| \mathcal{F}_k^Y \right), k = 1, 2... \tag{1.8}$$

We can also express the Bayes factor for the model selection by taking the ratio

$$B_k^{12} = \frac{\sigma_k^{(1)}(1)}{\sigma_k^{(2)}(1)}, k = 1, 2, ..., \tag{1.9}$$

of the unnormalized filters for two models. More specifically speaking, if we have unnormalized filters

$$\sigma_k^{(1)}(f) = E^Q\left(l_k^{(1)} f\left(X_k^{(1)}\right) | \mathcal{F}_k^Y\right) \tag{1.10}$$

and

$$\sigma_k^{(2)}(f) = E^Q\left(l_k^{(2)} f\left(X_k^{(2)}\right) | \mathcal{F}_k^Y\right) \tag{1.11}$$

for two different signal models $X^{(1)}$ and $X^{(2)}$, then B_k^{12} provides the Bayes factor for Model 1:$\{Y_m = h(X_{m-1}^{(1)}) + V_m\}_{m=1}^k$ over Model 2: $\{Y_m = h(X_{m-1}^{(2)}) + V_m\}_{m=1}^k$. The detailed proof of the equation (1.7) and (1.9) are in the appendix.

Figure 4 shows how the unnormalized filter approach works on a web based platform. There are four different models which describe the fair price of a financial asset and the main objective here is to select the correct one and track the fair price as well which are all in the presence of trading noise. In a web based platform, each computer can deal with one model thus we can do Bayes model selection and tracking for these four models at the same time. Each computer outputs a curve of Bayes factor and we pick the one with highest Bayes factor, which is Model 3. Then, the Model 3 tracker would be used to track the fair price.

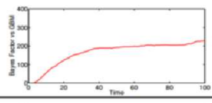
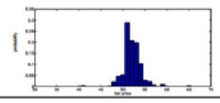
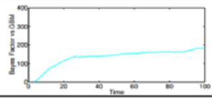
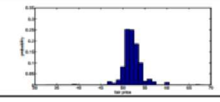
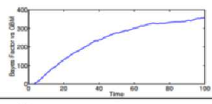
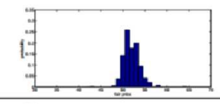
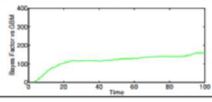
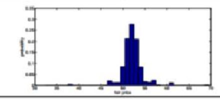
Computer	Model	Bayes Factor	Tracker
1	Model 1		
2	Model 2		
3	Model 3		
4	Model 4		

Figure 4: Web Based Model Selection and Tracking

1.3 Numeric Approximation

1.3.1 Particle Filter

To implement filters on computers, one needs a numeric approximation to the filter. The particle filter, a sequential Monte Carlo method, is a popular way to estimate the normalized filter. A particle filter estimates the unnormalized filter $\sigma_k(f)$ by

$$\sigma_k^n(f) = \sum_{i=1}^{n} l_k^i f(x_k^i), k = 1, 2, ..., \tag{1.12}$$

where n is the number of particles and initial particles $\{x_0^i\}_{i=1}^n$ are drawn independently from π_0.

Recalling the normalized filter $\pi_k(f)$ is recovered form the unnormalized filter by

$$\pi_k(f) = \frac{\sigma_k(f)}{\sigma_k(1)}, \tag{1.13}$$

one can find particle filter estimates for $\pi_k(f)$ by

$$\pi_k^n(f) = \frac{\sigma_k^n(f)}{\sigma_k^n(1)} = \frac{\sum\limits_{i=1}^{n} l_k^i f(x_k^i)}{\sum\limits_{i=1}^{n} l_k^i}, \tag{1.14}$$

where n is the number of particles.

1.3.2 Web Based Particle Filter

To estimate the filter accurately, one may need a large number of particles especially when the signal and observation models are complicated. Therefore, one computer can not deal with such a huge particle filter system and a web based platform is necessary. On a web based platform, one can estimate the unnormalized filter $\sigma_k(f)$ as

$$\sigma_k^{N,n}(f) = \sum_{i=1}^{N}\sum_{j=1}^{n} l_k^{i,j} f(x_k^{i,j}), k = 1, 2, ..., \tag{1.15}$$

and the normalized filter $\pi_k^{N,n}(f)$ as

$$\pi_k^{N,n}(f) = \frac{\sigma_k^{N,n}(f)}{\sigma_k^{N,n}(1)} = \frac{\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{n} l_k^{i,j} f(x_k^{i,j})}{\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{n} l_k^{i,j}}, \tag{1.16}$$

where N is the number of computers and n is the number of particles on each computer. Compared to the single computer platform, one can expand particle size N times as there are totally Nn particles on the web based platform, which is meaningful for both the performance and execution time of particle filter.

1.4 Contributions and Outline

The main contribution of this thesis is that with my supervisor Dr. Michael Kouritzin, I extend the resampled and branching system based on the single computer platform (see Kouritzin [13]) to a web based platform. Although the double bootstrap method was first introduced by Christelle et. al. (see [14]), the double residual, stratified, systematic, combined and web based branching are totally new algorithms. By simulation results, we show that the branching system is not only easy to be parallel implemented but also performs much better than any double resampled system.

In Chapter 2, we will review resampled and branching system on the single computer platform and explain how to implement resampled and branching system on a web based platform. The algorithms will be provided. In Chapter 3, we compare the performance and execution time of the web based resampled and branching system on the two simulation models. A parameter for comparing, which we call it "Bootstrap Factor", is defined and used to show the web based branching system is so much better than the web based resampled system. For the convenience of readers, we put all the proofs and calculations in Chapter 4, the Appendix.

Chapter 2

Particle Filter Methods

In section 2.1, we will review the resampled and branching systems on single computer platforms. Then in section 2.2, these two systems will be extended to a web based platform. We will explain the general idea of parallel implementation and the algorithms will be given.

2.1 Single Computer Platform

2.1.1 Weighted Particle Filter

The weighted particle system does not resample. In this case, the weighted conditional expectation

$$\sigma_k(f) = E^Q[l_k f(x_k) \big| \mathcal{F}_k^Y], \tag{2.1}$$

with respect to the reference probability Q is replaced with an independent sample average to arrive at

$$\sigma_k^n(f) = \frac{1}{n}\sum_{i=1}^{n} l_k^i f\left(x_k^i\right), \tag{2.2}$$

where n is the number of particles, f is a bounded function and the *particles* $\{x^i\}_{i=1}^{\infty}$ are independent (π_0, K)-Markov processes that are independent of the observations Y and the *weights* satisfy

$$l_k^i = \prod_{j=1}^{k} w_j^i, i = 1, 2, ..., n, \tag{2.3}$$

with

$$w_j^i = \alpha_j(x_{j-1}^i) = \frac{g\left(Y_j - h\left(x_{j-1}^i\right)\right)}{g\left(Y_j\right)}, i = 1, 2, ..., n. \tag{2.4}$$

Then, $\sigma_k^n(f)$ is our weighted-particle estimator of $\sigma_k(f)$ and

$$\sigma_0^n(f) = \frac{1}{n}\sum_{i=1}^{n} f\left(x_0^i\right), \tag{2.5}$$

as all particles have the same weight $l_0^i = 1, i = 1, 2, ...,$ at the initial time step.

Example 2.1. Suppose there is a hidden Markov process defined by $x_k = 0.95x_{k-1} + 0.3w_k, x_0 \sim \pi_0$ for $k = 1, 2, ...,$, where w_k is a random noise with standard Cauchy distribution. Now, we only have observations Y_k defined by $Y_k = x_{k-1} + v_k$, where v_k is a random noise with strictly, positive bounded density function $g = \frac{1}{\pi(1+x^2)}$. The bounded function f in this flow chart example is defined by:

$$f(x) = \begin{cases} 30 & : \quad x > 30 \\ x & : \quad -30 \leq x \leq 30 \\ -30 & : \quad x < -30 \end{cases}.$$

The flow chart in Figure 5 gives the process of weighted particle system.

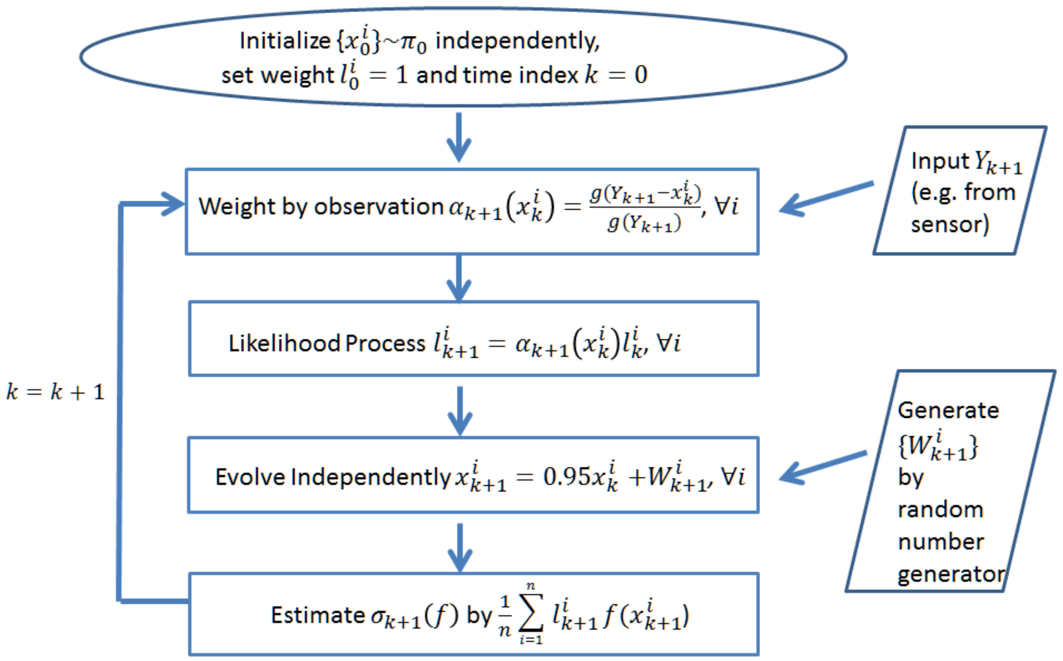

Initialize $\{x_0^i\} \sim \pi_0$ independently, set weight $l_0^i = 1$ and time index $k = 0$
Weight by observation $\alpha_{k+1}(x_k^i) = \frac{g(Y_{k+1} - x_k^i)}{g(Y_{k+1})}$, $\forall i$
Input Y_{k+1} (e.g. from sensor)
Likelihood Process $l_{k+1}^i = \alpha_{k+1}(x_k^i) l_k^i$, $\forall i$
$k = k + 1$
Generate $\{W_{k+1}^i\}$ by random number generator
Evolve Independently $x_{k+1}^i = 0.95 x_k^i + W_{k+1}^i$, $\forall i$
Estimate $\sigma_{k+1}(f)$ by $\frac{1}{n}\sum_{i=1}^{n} l_{k+1}^i f(x_{k+1}^i)$

Figure 5: Weighted Particle Filter

2.1.2 Bootstrap Particle Filter

Sometimes, the iteration of evolution of particles leads to a degeneracy problem, that only a few particles have significant weights and all the other particles have very small weights. Doucet et. al. have illustrated that the degeneracy problem is unavoidable in the weighted particle filter (see [15]). To solve the degeneracy problem, people introduce resampling into particle filters. Here, we still use the example in 2.1 and a flow chart, Figure 6, to visualize the resampled system, which includes bootstrap, residual resampling, stratified, systematic and combined stratified-residual resampling particle filters.

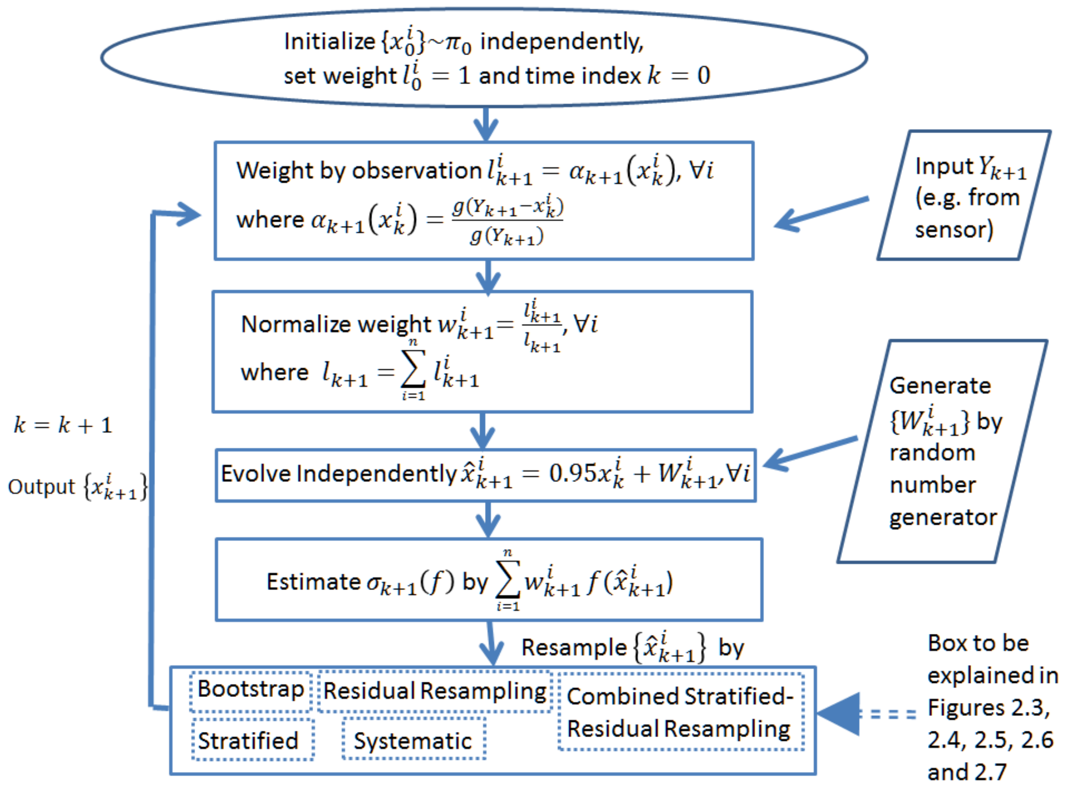

Initialize $\{x_0^i\} \sim \pi_0$ independently, set weight $l_0^i = 1$ and time index $k = 0$
Weight by observation $l_{k+1}^i = \alpha_{k+1}(x_k^i), \forall i$ where $\alpha_{k+1}(x_k^i) = \frac{g(Y_{k+1} - x_k^i)}{g(Y_{k+1})}$
Input Y_{k+1} (e.g. from sensor)
Normalize weight $w_{k+1}^i = \frac{l_{k+1}^i}{l_{k+1}}, \forall i$ where $l_{k+1} = \sum_{i=1}^{n} l_{k+1}^i$
Generate $\{W_{k+1}^i\}$ by random number generator
$k = k + 1$
Output $\{x_{k+1}^i\}$
Evolve Independently $\hat{x}_{k+1}^i = 0.95 x_k^i + W_{k+1}^i, \forall i$
Estimate $\sigma_{k+1}(f)$ by $\sum_{i=1}^{n} w_{k+1}^i f(\hat{x}_{k+1}^i)$
Resample $\{\hat{x}_{k+1}^i\}$ by
Bootstrap
Residual Resampling
Combined Stratified-Residual Resampling
Stratified
Systematic
Box to be explained in Figures 2.3, 2.4, 2.5, 2.6 and 2.7

Figure 6: Resampled Particle Filter

Due to resampling, the weight in the resampled particle filters is just the one-step likelihood,

$$l_k^i = \alpha_k\left(x_{k-1}^i\right), i = 1, 2, ..., n, \tag{2.6}$$

instead of the multi-steps $\prod_{j=1}^{k} \alpha_j^i\left(x_{j-1}^i\right)$ as the weighted one, where

$$\alpha_k(x) = \frac{g\left(Y_k - h(x)\right)}{g\left(Y_k\right)}. \tag{2.7}$$

Before resampling, the weight is normalized by

$$w_k^i = \frac{l_k^i}{l_k}, i = 1, 2, ..., n, \tag{2.8}$$

where $l_k = \sum_{i=1}^{n} l_k^i$.

Notice here we estimate the unnormalized filter $\sigma_k(f)$ before resampling to avoid introducing more noise. To be clearer on notation, we use $\widehat{x}_{k+1}^i$ for particle location after evolve but before resampling and x_{k+1}^i for particles after resampling.

The bootstrap method, by Gordan et. al. (1993) (see [12]), was the first and is still the most popular resampled particle filter. The idea of the bootstrap method is particle x_k^i will be selected by the probability of its normalized weight w_k^i. One possible implementation is to divide the interval $[0, 1]$ into n small intervals with the length of i'th interval being $w_k^i, i = 1, 2..., n$, which is the weight of corresponding particle. More specifically, the small interval is $[p_{i-1}, p_i]$, for some $i = 1, 2, ..., n$, where

$$p_i = \sum_{j=1}^{i} w_k^j, \ p_0 = 0. \tag{2.9}$$

Then, each time before selecting a particle, a random number u_m is generated by $[0, 1]$-uniform distribution. If u_m drops into the interval $[p_{i-1}, p_i)$, then the particle location $\widehat{x}_k^i$ will be selected for a new particle x_{k+1}^i.

However, this approach is not efficient when implemented on computer since one must compare each u_m to each interval $[p_{i-1}, p_i]$. As there are n intervals and we need to select n particles, the computer will have $O(n^2)$ operations when resampling. To implement bootstrap resampling more efficiently, we create the order statistics v_m by

$$v_m = u_m^{\frac{1}{m}} v_{m+1}, \ m = n, n-1, ...1, \tag{2.10}$$

where $v_{n+1} = 1$ and u_m has $[0, 1]$-uniform distribution. Since v_m increases in m, then if v_{m+1} drops into the interval $[p_{i-1}, p_i)$, we only need to compare v_m with the intervals $[p_{j-1}, p_j]$ for $j \leq i$ instead of all $j = 1, 2, ..., n$. Therefore the computer will only have $O(n)$ operations when resampling. In Figure 7, we visualize the bootstrap resampling by a flow chart that fits in the lowest box in Figure 6

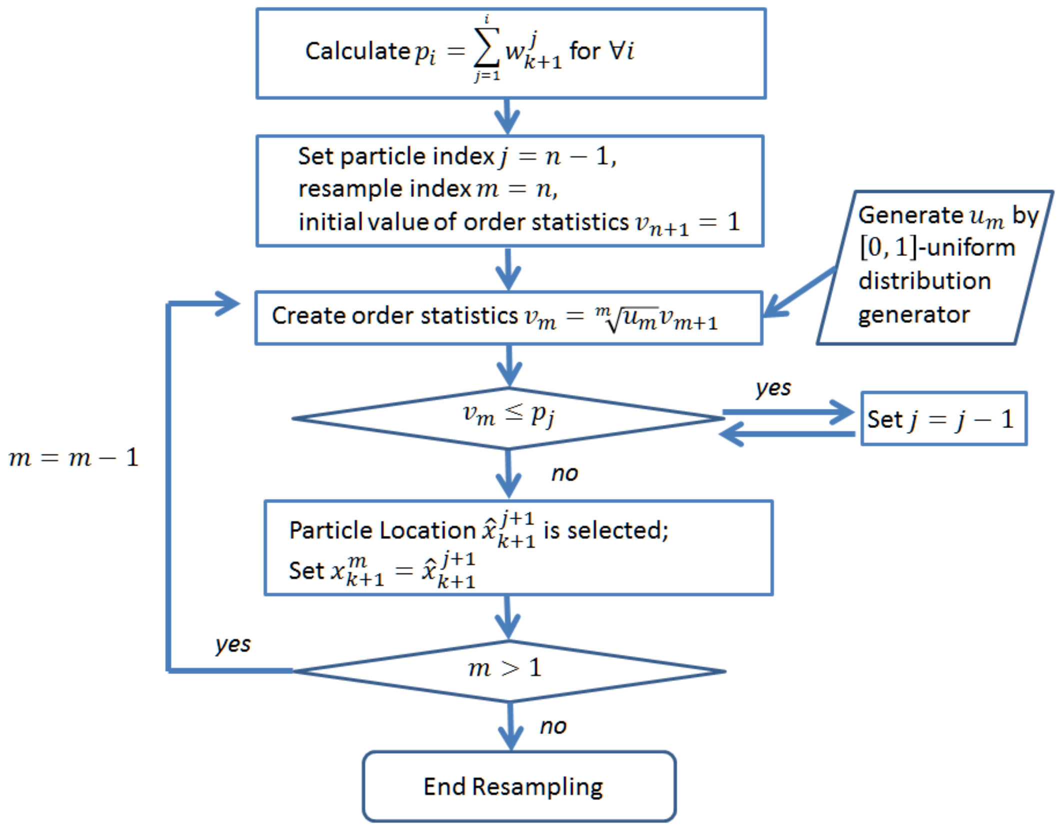
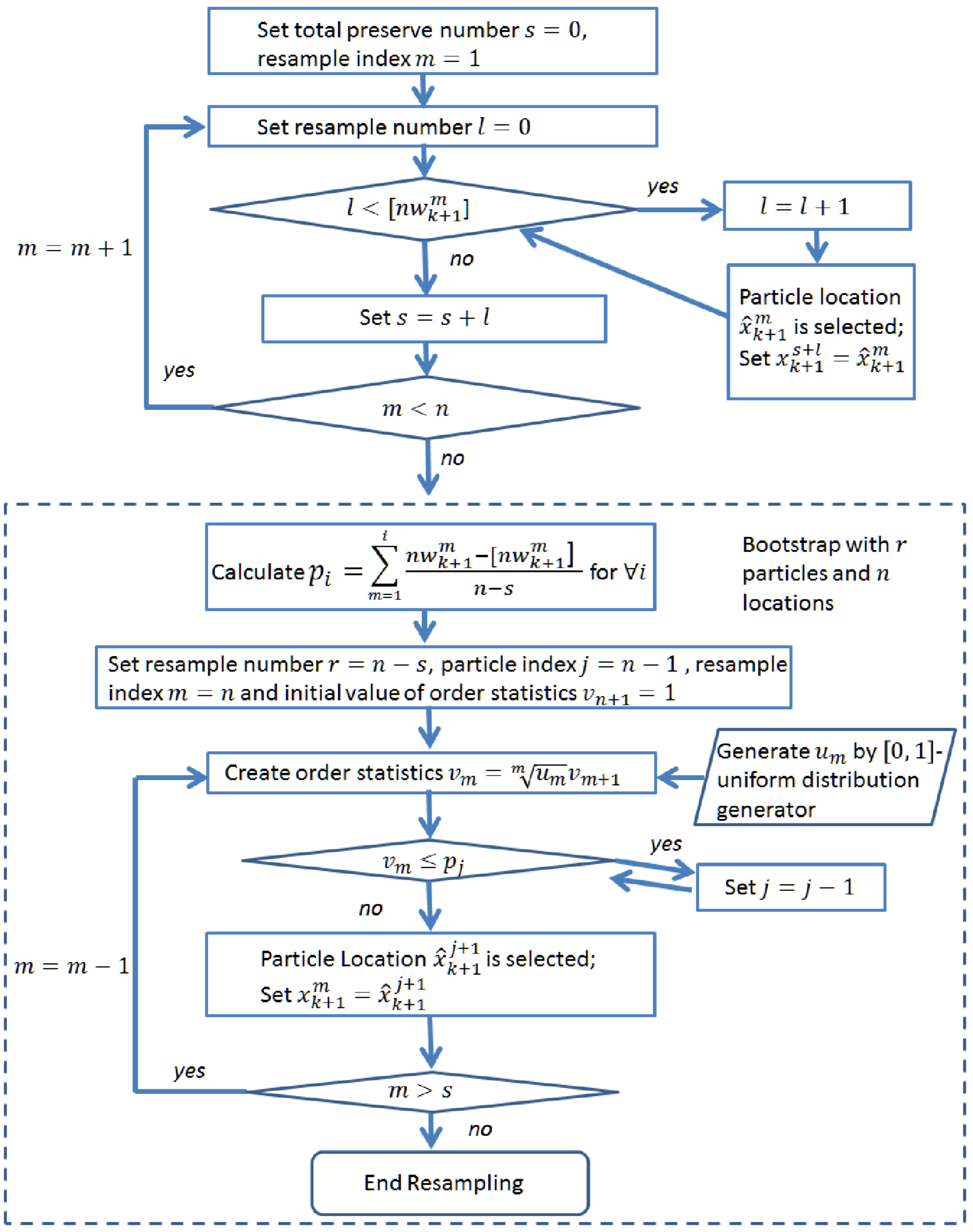

Calculate $p_i = \sum_{j=1}^{i} w_{k+1}^j$ for $\forall i$
Set particle index $j = n - 1$, resample index $m = n$, initial value of order statistics $v_{n+1} = 1$
Generate u_m by $[0, 1]$-uniform distribution generator
Create order statistics $v_m = \sqrt[m]{u_m} v_{m+1}$
$v_m \leq p_j$
yes
Set $j = j - 1$
no
$m = m - 1$
Particle Location $\hat{x}_{k+1}^{j+1}$ is selected; Set $x_{k+1}^m = \hat{x}_{k+1}^{j+1}$
yes
$m > 1$
no
End Resampling

Figure 7: Bootstrap Method at time step *k+1*

2.1.3 Residual Resampling Particle Filter

The idea of the residual resampling is redistributing fewer particles by preserving the particles with high weights, which implies less noise is introduced when resampling. This method was first introduced by Liu and Chen (1998) (see [16]). The residual resampling method has two steps: preserve and resample. In preserve step, the multiplicity to preserve the particle location $\widehat{x}_k^i$ is defined by $\lfloor n w_{k+1}^i \rfloor$, where n is the number of particles. We count the number of preserved particles as preserve number s. Then in the resample step, we only need to select $n - s$ particle locations by the bootstrap method. In Figure 8, we visualize the residual resampling method by a flow chart.



Set total preserve number $s = 0$, resample index $m = 1$
Set resample number $l = 0$
$l < [nw_{k+1}^{m}]$
yes
$l = l + 1$
Particle location $\hat{x}_{k+1}^{m}$ is selected; Set $x_{k+1}^{s+l} = \hat{x}_{k+1}^{m}$
no
Set $s = s + l$
$m = m + 1$
yes
$m < n$
no
Calculate $p_i = \sum_{m=1}^{i} \frac{nw_{k+1}^{m} - [nw_{k+1}^{m}]}{n-s}$ for $\forall i$
Bootstrap with r particles and n locations
Set resample number $r = n - s$, particle index $j = n - 1$, resample index $m = n$ and initial value of order statistics $v_{n+1} = 1$
Create order statistics $v_m = \sqrt[m]{u_m} v_{m+1}$
Generate u_m by $[0,1]$-uniform distribution generator
$v_m \leq p_j$
yes
Set $j = j - 1$
no
Particle Location $\hat{x}_{k+1}^{j+1}$ is selected; Set $x_{k+1}^{m} = \hat{x}_{k+1}^{j+1}$
$m = m - 1$
yes
$m > s$
no
End Resampling

Figure 8: Residual Resampling Method at time step *k+1*

2.1.4 Stratified Particle Filter

The idea of the stratified resampling method is to generate the uniform random number in a smaller interval, whose length is $\frac{1}{n}$, rather than the interval $[0, 1]$. Thus, less randomness is introduced when resampling. This method was first introduced by Kitagawa (1996) (see [17]). In Figure 9, we visualize the stratified resampling method by a flow chart.

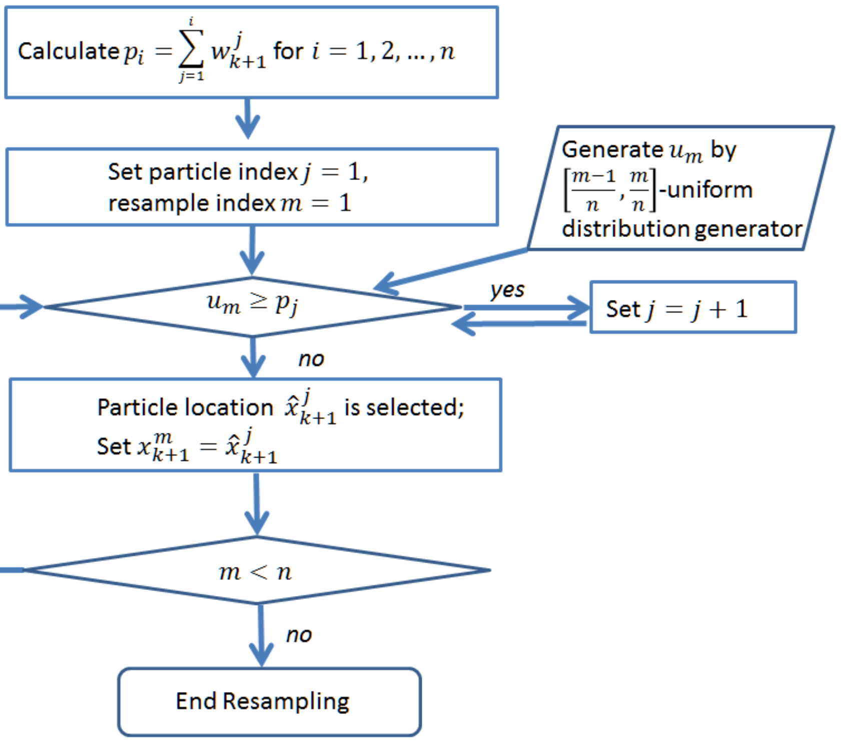

Calculate $p_i = \sum_{j=1}^{i} w_{k+1}^j$ for $i = 1, 2, \dots, n$
Set particle index $j = 1$, resample index $m = 1$
Generate u_m by $\left[\frac{m-1}{n}, \frac{m}{n}\right]$-uniform distribution generator
$u_m \geq p_j$
yes
Set $j = j + 1$
no
Particle location $\hat{x}_{k+1}^j$ is selected; Set $x_{k+1}^m = \hat{x}_{k+1}^j$
$m = m + 1$
yes
$m < n$
no
End Resampling

Figure 9: Stratified Method at time step *k+1*

2.1.5 Systematic Particle Filter

The systematic resampling method is a modification of the stratified method by Carpenter et. al. (1999) (see [18]), which is easier and more efficient to implement. The idea of the systematic method is that instead of generating the uniform random number each time before selecting a particle, we only generate one random number u from $[-\frac{1}{n}, 0]$-uniform distribution prior to resampling. Then each time before selecting a particle location, the random number u_m is defined as

$$u_m = u + \frac{m}{n}, \ \ m = 1, 2, ...n, \tag{2.11}$$

where n is the number of particles. We visualize the systematic resampling method by a flow chart in Figure 10.

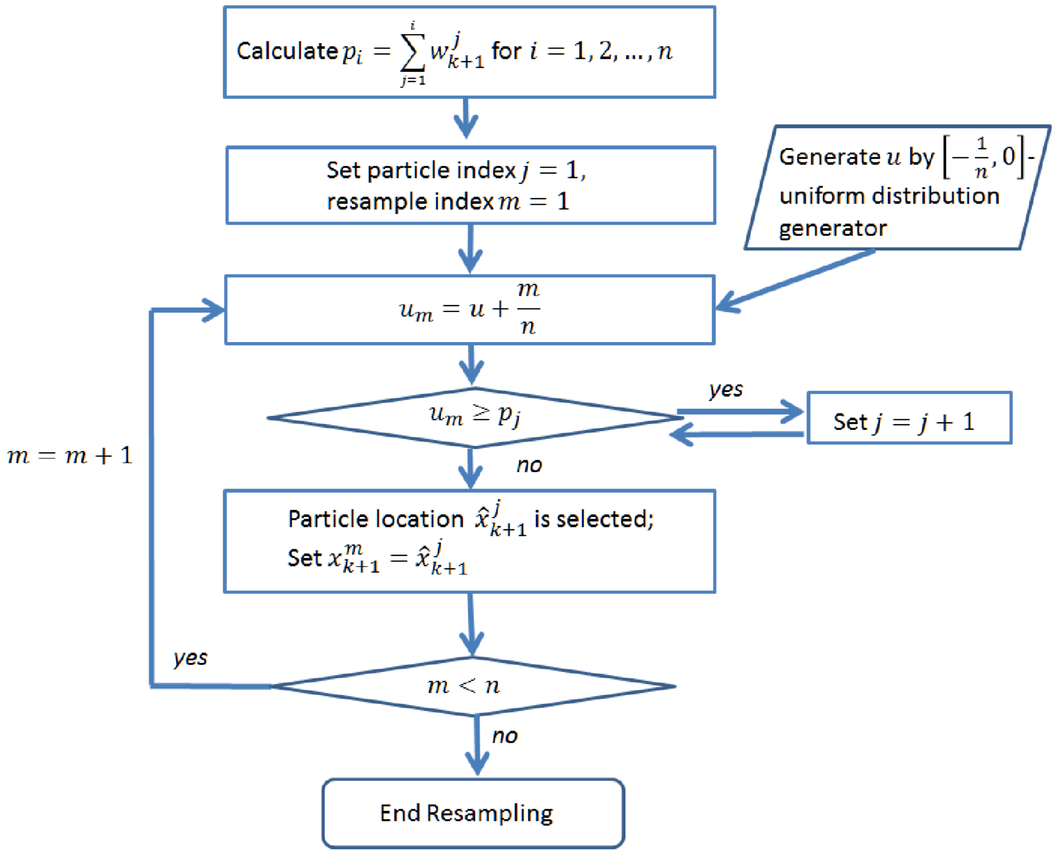

Calculate $p_i = \sum_{j=1}^{i} w_{k+1}^{j}$ for $i = 1, 2, ..., n$
Set particle index $j = 1$, resample index $m = 1$
Generate u by $\left[-\frac{1}{n}, 0\right]$-uniform distribution generator
$u_m = u + \frac{m}{n}$
$u_m \geq p_j$
yes
Set $j = j + 1$
no
$m = m + 1$
Particle location $\hat{x}_{k+1}^{j}$ is selected; Set $x_{k+1}^{m} = \hat{x}_{k+1}^{j}$
yes
$m < n$
no
End Resampling

Figure 10: Systematic Method at time step *k+1*

2.1.6 Combined Stratified-Residual Resampling Particle Filter

The residual resampling method reduces noise in resampling by preserving the particles with high weights and stratified resampling method reduces noise by generating uniform random number in a smaller interval. As they are two totally different ways and Douc et. al. have showed that they are all unbiased (see [19]), it is reasonable to combine these two methods together, which means we still preserve the particles with high weights but resample the left over particles by the stratified method. Notice that after preserving s particles by the residual method, we only need to select $n - s$ more particles, which can be done using the stratified method. Therefore we divide the $[0, 1]$ interval to $n - s$ smaller intervals here instead of n smaller intervals in the stratified method. We also visualize combined resampling method by a flow chart in Figure 11.

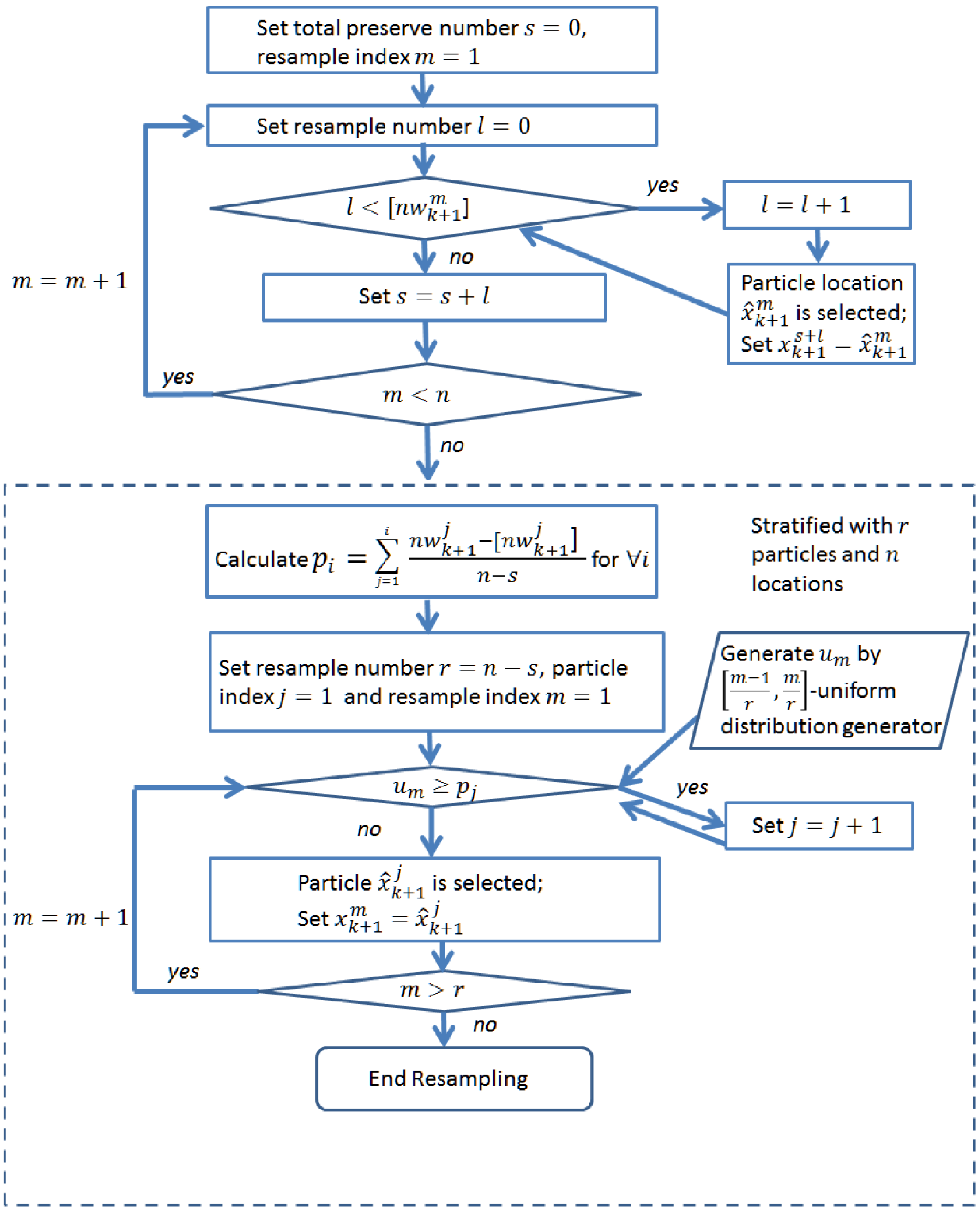

Set total preserve number $s = 0$, resample index $m = 1$
Set resample number $l = 0$
$l < [nw_{k+1}^m]$
yes
$l = l + 1$
Particle location $\hat{x}_{k+1}^m$ is selected; Set $x_{k+1}^{s+l} = \hat{x}_{k+1}^m$
no
Set $s = s + l$
$m = m + 1$
yes
$m < n$
no
Calculate $p_i = \sum_{j=1}^{i} \frac{nw_{k+1}^j - [nw_{k+1}^j]}{n-s}$ for $\forall i$
Stratified with r particles and n locations
Set resample number $r = n - s$, particle index $j = 1$ and resample index $m = 1$
Generate u_m by $\left[\frac{m-1}{r}, \frac{m}{r}\right]$-uniform distribution generator
$u_m \geq p_j$
yes
Set $j = j + 1$
no
Particle $\hat{x}_{k+1}^j$ is selected; Set $x_{k+1}^m = \hat{x}_{k+1}^j$
$m = m + 1$
yes
$m > r$
no
End Resampling

Figure 11: Combined Stratified-Residual Resampling Method at time *k+1*

2.1.7 Branching Particle Filter

The branching particle filter starts with n initial particles and each particle may be preserved, duplicated or killed during branching process based on its weight. Instead of complete resampling in the resampled systems and no resampling in the weighted particle filter, branching particle filter is a partial resampling method. Similar to the weighted particle filter, the weight in branching particle filter is also defined by the multi-steps likelihood

$$l_{k+1}^i = \alpha_{k+1}(x_k^i) l_k^i, \;\; l_0^i = 1, \;\; i = 1, 2, ..., n_k, \tag{2.12}$$

where n_k is the number of particles at time step k. We show the general steps of the branching particle filter in Figure 12 and discuss the more detailed branching process in Figure 13 and 14.

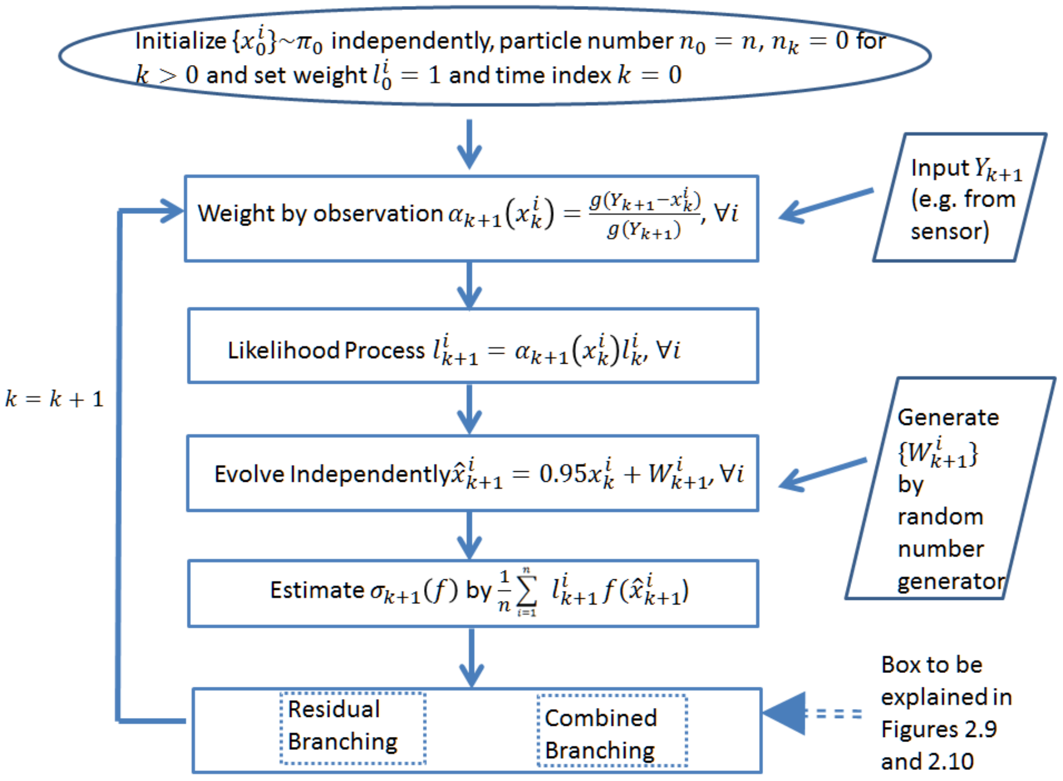

Initialize $\{x_0^i\} \sim \pi_0$ independently, particle number $n_0 = n$, $n_k = 0$ for $k > 0$ and set weight $l_0^i = 1$ and time index $k = 0$
Weight by observation $\alpha_{k+1}(x_k^i) = \frac{g(Y_{k+1} - x_k^i)}{g(Y_{k+1})}$, $\forall i$
Input Y_{k+1} (e.g. from sensor)
Likelihood Process $l_{k+1}^i = \alpha_{k+1}(x_k^i) l_k^i$, $\forall i$
$k = k + 1$
Evolve Independently $\hat{x}_{k+1}^i = 0.95 x_k^i + W_{k+1}^i$, $\forall i$
Generate $\{W_{k+1}^i\}$ by random number generator
Estimate $\sigma_{k+1}(f)$ by $\frac{1}{n}\sum_{i=1}^{n} l_{k+1}^i f(\hat{x}_{k+1}^i)$
Residual Branching
Combined Branching
Box to be explained in Figures 2.9 and 2.10

Figure 12: Branching System

The idea of the branching particle filter is that we preserve particles whose weights are close to the average weight. More specifically speaking, if the weight l^i_{k+1} satisfies

$$l^i_{k+1} \in (a_k A_{k+1}, b_k A_{k+1}),\ a_k \le b_k,\ i = 1, 2, ..., n_k, \tag{2.13}$$

then the particle location $\widehat{x}^i_{k+1}$ will be preserved and its weight l^i_{k+1} will be inherited. However, if $l^i_{k+1} \notin (a_k A_{k+1}, b_k A_{k+1})$, then the particle location $\widehat{x}^i_{k+1}$ will branch and its offspring number n^i_k is

$$n^i_{k+1} = \left\lfloor \frac{l^i_{k+1}}{A_{k+1}} \right\rfloor + \rho^i_{k+1}, \tag{2.14}$$

where ρ^i_{k+1} is a $(\frac{l^i_{k+1}}{A_{k+1}} - \left\lfloor \frac{l^i_{k+1}}{A_{k+1}} \right\rfloor)$-Bernoulli random number so there is no bias. We visualize residual branching process in Figure 13.

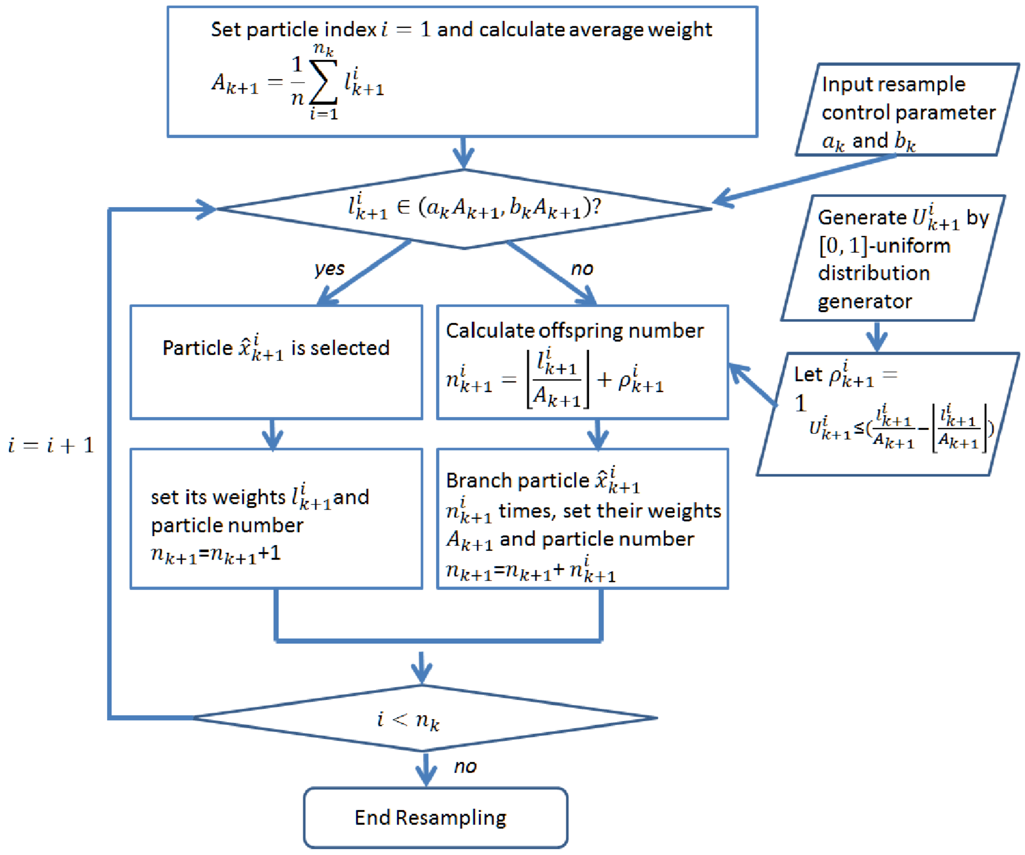

Set particle index $i = 1$ and calculate average weight $A_{k+1} = \frac{1}{n}\sum_{i=1}^{n_k} l^i_{k+1}$
Input resample control parameter a_k and b_k
$l^i_{k+1} \in (a_k A_{k+1}, b_k A_{k+1})$?
yes
no
Generate U^i_{k+1} by $[0,1]$-uniform distribution generator
Particle $\hat{x}^i_{k+1}$ is selected
Calculate offspring number $n^i_{k+1} = \left\lfloor \frac{l^i_{k+1}}{A_{k+1}} \right\rfloor + \rho^i_{k+1}$
Let $\rho^i_{k+1} = 1$ $U^i_{k+1} \le (\frac{l^i_{k+1}}{A_{k+1}} - \left\lfloor \frac{l^i_{k+1}}{A_{k+1}} \right\rfloor)$
$i = i + 1$
set its weights l^i_{k+1} and particle number n_{k+1}=n_{k+1}+1
Branch particle $\hat{x}^i_{k+1}$ n^i_{k+1} times, set their weights A_{k+1} and particle number n_{k+1}=n_{k+1}+ n^i_{k+1}
$i < n_k$
no
End Resampling

Figure 13: Residual Branching System at time step *k+1*

Similar as the resampled system, there are also different ways to redistribute particles in the branching particle system. By the idea of stratified resampling, we can add it into our branching system to help control particle number and decrease noise. The idea is instead of generating the random number by $[0,1]$-uniform distribution, we generate the random number in a smaller interval $[\frac{k-1}{n_k-m}, \frac{k}{n_k-m}]$, where m is the non-resample count. Therefore the particle number can be more stable. We visualize the combined branching process in Figure 14.

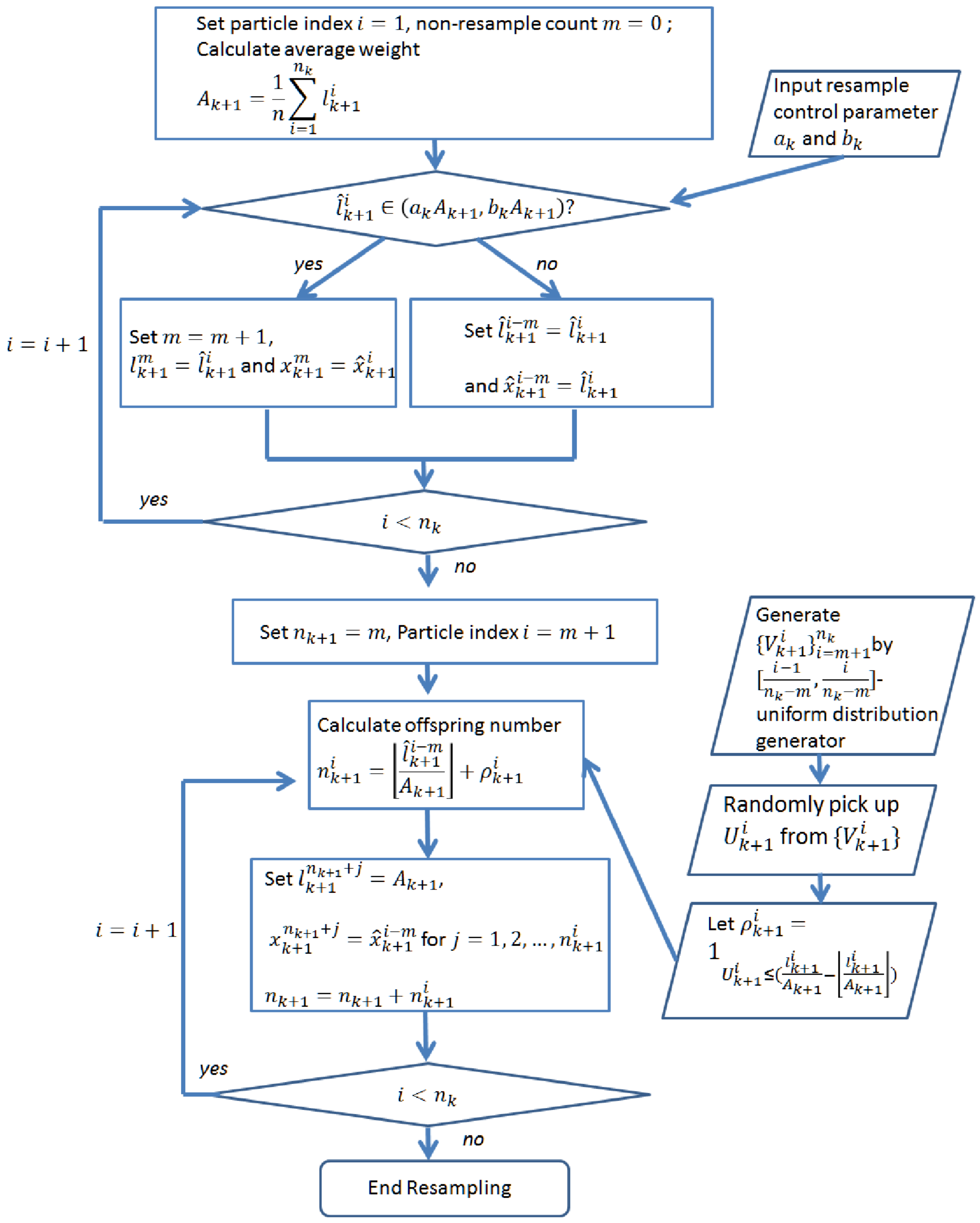

Set particle index $i = 1$, non-resample count $m = 0$; Calculate average weight $A_{k+1} = \frac{1}{n}\sum_{i=1}^{n_k} l_{k+1}^i$
Input resample control parameter a_k and b_k
$\hat{l}_{k+1}^i \in (a_k A_{k+1}, b_k A_{k+1})$?
yes
no
Set $m = m + 1$, $l_{k+1}^m = \hat{l}_{k+1}^i$ and $x_{k+1}^m = \hat{x}_{k+1}^i$
Set $\hat{l}_{k+1}^{i-m} = \hat{l}_{k+1}^i$ and $\hat{x}_{k+1}^{i-m} = \hat{l}_{k+1}^i$
$i = i + 1$
yes
$i < n_k$
no
Set $n_{k+1} = m$, Particle index $i = m + 1$
Generate $\{V_{k+1}^i\}_{i=m+1}^{n_k}$by $[\frac{i-1}{n_k-m}, \frac{i}{n_k-m}]$-uniform distribution generator
Calculate offspring number $n_{k+1}^i = \left\lfloor \frac{\hat{l}_{k+1}^{i-m}}{A_{k+1}} \right\rfloor + \rho_{k+1}^i$
Randomly pick up U_{k+1}^i from $\{V_{k+1}^i\}$
Let $\rho_{k+1}^i = 1_{U_{k+1}^i \le (\frac{l_{k+1}^i}{A_{k+1}} - \left\lfloor \frac{l_{k+1}^i}{A_{k+1}} \right\rfloor)}$
$i = i + 1$
Set $l_{k+1}^{n_{k+1}+j} = A_{k+1}$, $x_{k+1}^{n_{k+1}+j} = \hat{x}_{k+1}^{i-m}$ for $j = 1, 2, \ldots, n_{k+1}^i$ $n_{k+1} = n_{k+1} + n_{k+1}^i$
yes
$i < n_k$
no
End Resampling

Figure 14: Combined Branching System at time step *k+1*

2.2 Web Based Platform

2.2.1 Double Bootstrap

The double bootstrap, which is also called the island particle method, is developed and studied by Christelle et. al. in 2013 (see [14]). The idea of the island particle method is as follows: the total population of particles is divided into the sub-populations, referred to as islands. In this way, resampling of the total population of particles is divided into two steps: resampling between islands and within each island. In practice, we can consider each computer as a platform to implement one island and then a large population of particles, which is unable to be implemented efficiently on one computer, can be divided to multiple computers so that each computer only needs to deal with a smaller sub-population of particles.

This web based particle filter can speed up the computation as we can divide the whole population of particles into multiple computers.This filter will be closer to the optimal filter, as more particles can be used at the same time.

Hereafter, we suppose that there are N islands each with n particles. Thus, the sub-population of particles on the island i at time step k can be expressed by $\mathbb{X}_k^i = (x_k^{i,1}, x_k^{i,2}, ..., x_k^{i,n})$. In the double bootstrap method, bootstrap will be used to resample between islands before resampling within each island. An island also has its own weight, defined by:

$$\widehat{\mathbb{L}}_k^i = \sum_{j=1}^{n} \widehat{l}_k^{i,j}, i = 1, 2, ..., N, \tag{2.15}$$

the sum of weights of all the particles on the island. Then, during resampling, each island can be selected by the probability of its normalized weight.

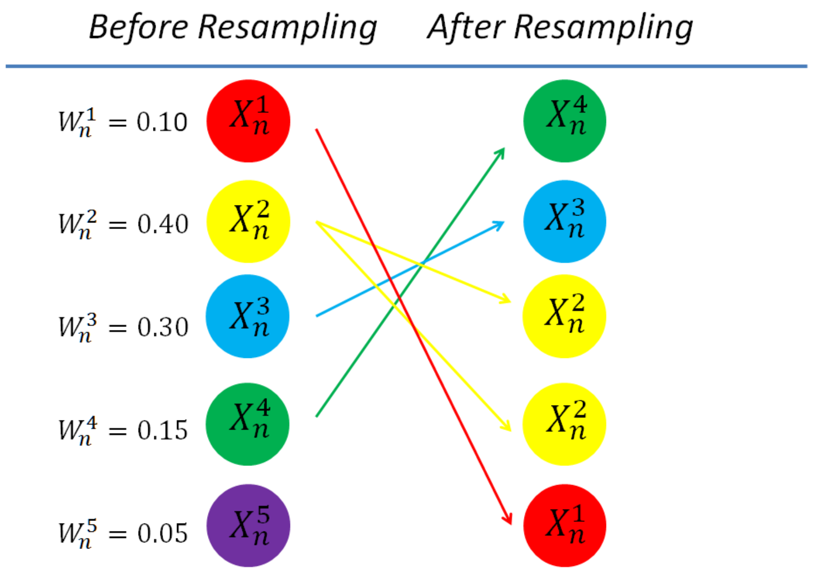


Figure 15: Resampling Between Islands of Double Bootstrap

Example 2.2. Here is a simple example to show how resampling between islands works. There are 5 islands $\mathbb{X}_k^1, \mathbb{X}_k^2, \mathbb{X}_k^3, \mathbb{X}_k^4, \mathbb{X}_k^5$, with normalized weight $0.10, 0.40, 0.30, 0.15$ and 0.05, which are displayed in Figure 15.

Because of the randomness of resampling, here we just describe one possible case. During resampling, firstly the forth island $\mathbb{X}_k^4$ is selected, which means the data of all particles on Island 4 will be copied and passed to Computer 1 to form a new Island 1. Then Island 3 is selected once and Island 2 is selected two times due to its high weight. Thus the data of all particles on Island 3 will be copied then passed to computer 2 and the data of Island 2 will be copied twice then passed to both Computer 3 and 4. Finally, island 1 is selected then its data will be copied and passed to Computer 5 to form new Island 5. Island 5 will be killed because of its low weight. Suppose there are n particles on each island, then totally there will be $5n$ particles passing between islands in this example.

After resampling between islands, the particles on each island will be resampled by bootstrap independently. The algorithm of the double bootstrap particle filter is as the following. We first give the common steps for the double resampled system and then give the detailed steps of bootstrap resampling between islands and within an island.

Algorithm 1 General Steps of Double Resampled System

```
1: Initialization:
2: for $i = 1$ to $N$ do
3:     $\mathbb{X}_0^i = \left\{x_0^{i,j}\right\}_{j=1}^n$ are independent initial particle samples of $\pi_0$, $V_{N+1} = 1$
4: end for
5: for $k = 0$ to $T$ do
6:     Weight by Observation: $\widehat{l_{k+1}^{i,j}} = \alpha_{k+1}\left(x_k^{i,j}\right)$ for $i = 1, 2, ..., N, j = 1, 2, ..., n$
7:     Normalize Particles' Weight: $w_{k+1}^{i,j} = \frac{\widehat{l_{k+1}^{i,j}}}{\widehat{\mathbb{L}}_{k+1}^i}$ for $i = 1, 2, ..., N, j = 1, 2, ..., n$ where $\widehat{\mathbb{L}}_{k+1}^i = \sum_{j=1}^n \widehat{l}_{k+1}^{i,j}$
8:     Normalize Islands' Weight: $W_{k+1}^i = \frac{\widehat{\mathbb{L}_{k+1}^i}}{\widehat{\mathbb{L}}_{k+1}}$ for $i = 1, 2, ..., N$ where $\widehat{\mathbb{L}}_{k+1} = \sum_{i=1}^N \widehat{\mathbb{L}}_{k+1}^i$
9:     Evolve Independently: $P^Y(\widehat{x}_{k+1}^{i,j} \in \Gamma_{i,j} \quad \forall \quad i, j | \mathcal{F}_k^{\mathbb{X}}) = \prod_{i=1}^N \prod_{j=1}^n K(x_k^{i,j}, \Gamma_{i,j})$ for all $\Gamma_{i,j}$
10:    Estimate $\pi_{k+1}$: $\mathbb{P}_{k+1}^{N,n} = \sum_{i=1}^N \sum_{j=1}^n w_{k+1}^{i,j} \delta_{\widehat{x}_{k+1}^{i,j}}$
11:    Resampling Between Islands
12:    Resampling Within Island
13: end for
```

Algorithm 2 Double Bootstrap Particle Filter

```
1: Resampling between Islands
2: $P_i = \sum_{m=1}^{i} w_{k+1}^{m}$ for $i = 1, ..., N$, $L = N - 1$
3: for $m = N$ to 1 do
4:     Draw $[0,1]$-uniform $U_m$ and set $V_m = U_m^{\frac{1}{m}} V_{m+1}$
5:     While $V_m \leq P_L$ set $L = L - 1$
6:     Set $\mathbb{X}_{k+1}^{m} \doteq \widehat{\mathbb{X}}_{k+1}^{l+1}$
7: end for
8: Resampling Within Island
9: for $i = 1$ to $n$ do
10:     $p_{i,j} = \sum_{m=1}^{j} w_{k+1}^{i,j}$ for $j = 1, ..., n$, $l = n - 1$
11:     for $m = n$ to 1 do
12:         Draw $[0,1]$-uniform $u_m$ and set $v_m = u_m^{\frac{1}{m}} v_{m+1}$
13:         While $v_m \leq p_{i,l}$ set $l = l - 1$
14:         Set $x_{k+1}^{i,j} \doteq \widehat{x}_{k+1}^{i,l+1}$
15:     end for
16: end for
```

2.2.2 Double Residual Resampling

The double residual resampling is an extension for the residual resampling method to a web based platform. Similar as the double bootstrap, we also divide the total population of particles into several islands. On each island, they are resampled by the residual resampling method. Between islands, the residual resampling will be used as well.

Resampling between islands is separated into two steps: preserve and resample. In the double bootstrap resampling, some islands with higher weight than average may still be killed by the bootstrap method. To avoid these possible cases and eliminate unnecessary noise, we preserve islands with weight, which is higher than the average weight of all islands, and then do resampling. The multiplicity of each island to be preserved is $\lfloor NW_{k+1}^{i} \rfloor$, where N is the number of islands and W^i_{k+1} is the normalized weight of each island.

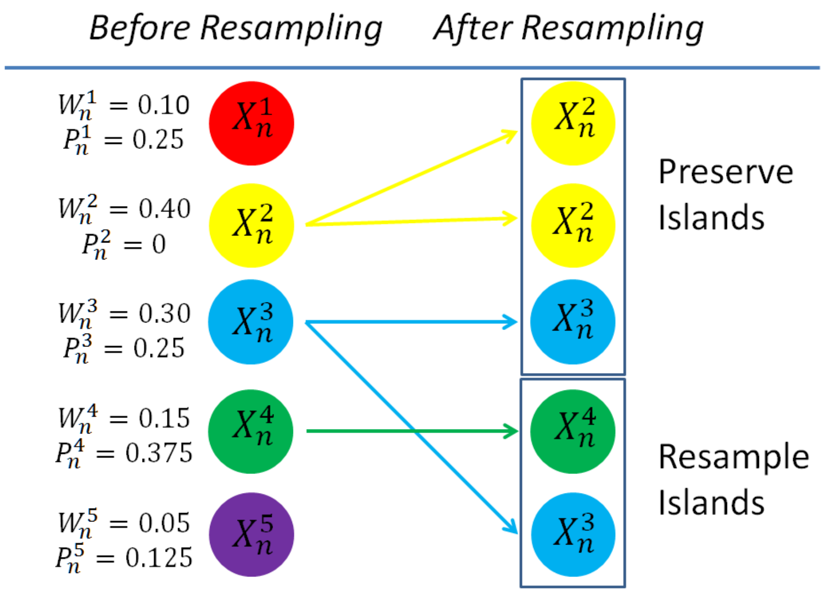


Figure 16: Resampling Between Islands of Double Residual Resampling

We also use a simple example to show how it works during resampling between islands.

Example 2.3. In this example, there are 5 islands $\mathbb{X}^1_k, \mathbb{X}^2_k, \mathbb{X}^3_k, \mathbb{X}^4_k, \mathbb{X}^5_k$, with normalized weight $0.10, 0.40, 0.30, 0.15$ and 0.05, which are displayed in Figure 16.

First of all, we need to select preserved islands. It is shown in the picture that Island 2 and Island 3 should be preserved as their weights are higher than the average weight of 0.2. Then, by the formula $\lfloor NW^i_{k+1} \rfloor$, Island 2 should be selected twice and Island 3 be selected once. Thus, we copy the data of all particles of Island 2 twice and then pass them to Computer 1 and 2 to form new Island 1 and 2. As Island 3 should be selected once then the data of Island 3 is copied once and pass to Computer 3 to form new Island 3. Therefore, the preserve number S in this simple example is 3.

After preserving islands with higher weights, we re-calculate the weight of each island by the formula $P^i_k = \frac{NW^i_k - \lfloor NW^i_k \rfloor}{R}$ for $i = 1, ..., N$, where $R = N - S$. The new weight P^i_k is also displayed on the left side of island. Because of the randomness of resampling, here we just describe one possible case of resampling. As Island 3 and Island 4 have higher new weight, thus they are selected and copied to pass the data to Computer 4 and 5 to form new Island 4 and 5. Island 1 and Island 5 are killed due to their lower weight.

By double residual resampling method, there are also $5n$ particles passed between islands when resampling between islands in this example.

The algorithm of resampling between islands and within island for double residual resampling particle filter is as follows:

Algorithm 3 Double Residual Resampling Particle Filter

```
1: Resampling between Islands
2: Preserve: $S = 0$
3: for $i = 1$ to $N$ do
4:     $M = 0$
5:     While $M < \lfloor NW^i_{k+1} \rfloor$ set $M = M + 1$, $\mathbb{X}^{S+M}_{k+1} \doteq \widehat{\mathbb{X}}^i_{k+1}$
6:     $S = S + M$
7: end for
8: Resample: $R = N - S$; $\overline{P}_i = \sum_{m=1}^{i} \frac{N_2 W^m_{k+1} - \lfloor N_2 w^k_{n+1} \rfloor}{R}$ for $i = 1, ..., N$; $j = n - 1$,
   $L = R - 1$
9: for $M = N$ to $S + 1$ do
10:     Draw $[0, 1]$-uniform $U_M$ and set $V_M = U_M^{\frac{1}{M}} V_{M+1}$
11:     While $V_M \leq P_L$ set $L = L - 1$
12:     Set $\mathbb{X}^M_{k+1} \doteq \widehat{\mathbb{X}}^{L+1}_{k+1}$
13: end for
14: Resampling Within Island
15: for $i = 1$ to $N$ do
16:     Preserve: $s = 0$
17:     for $j = 1$ to $n$ do
18:         $m = 0$
19:         While $m < \lfloor nw^{i,j}_{k+1} \rfloor$ set $m = m + 1$, $x^{i,s+m}_{k+1} \doteq \widehat{x}^{i,j}_{k+1}$
20:     end for
21:     Resample: $r = n - s$; $\overline{p}_{i,j} = \sum_{m=1}^{j} \frac{nw^{i,m}_{k+1} - \lfloor nw^{i,m}_{k+1} \rfloor}{r}$ for $j = 1, ..., n$; $l = n - 1$
22:     for $m = n$ to $s + 1$ do
23:         Draw $[0, 1]$-uniform $u_m$ and set $v_m = U_m^{\frac{1}{m}} V_{m+1}$
24:         While $v_m \leq p_{i,l}$ set $l = l - 1$
25:         Set $x^{i,m}_{k+1} \doteq \widehat{x}^{i,l+1}_{k+1}$
26:     end for
27: end for
```

2.2.3 Double Stratified Resampling

The double stratified resampling is an extension for stratified resampling method to a web based platform. Similar to the double bootstrap, we also divide the total population of particles to several islands. On each island, they are resampled by the stratified resampling method. Between islands, the stratified resampling method will be used to resample as well.

There will be less randomness when resampling between islands by double stratified resampling because of generating uniform random number over a smaller interval, whose length is $\frac{1}{N}$, instead of $[0, 1]$.

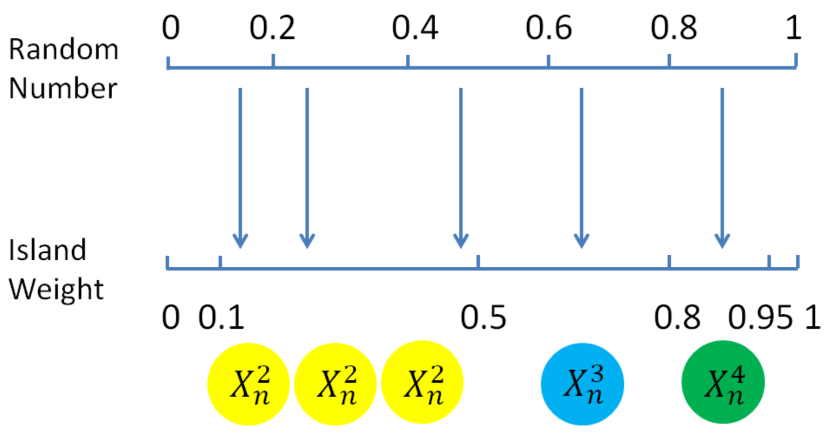


Figure 17: Resampling Between Islands of Double Stratified Resampling

We also use a simple example to show how it works during resampling between islands.

Example 2.4. In this example, there are 5 islands $\mathbb{X}_k^1, \mathbb{X}_k^2, \mathbb{X}_k^3, \mathbb{X}_k^4, \mathbb{X}_k^5$, with normalized weight $0.10, 0.40, 0.30, 0.15$ and 0.05. We put them onto the number line to get a serial of intervals of particle weight in Figure 17. Because of the randomness of resampling, here we just describe one possible case. According to the stratified resampling method, the length of random number interval is $\frac{1}{5} = 0.2$. Thus at first time, it generates a uniform random number from $[0, 0.2]$ and this number drops into interval $[0.1, 0.5]$, which represents Island 2.

Similarly, second random number, which is generated from interval $[0.2, 0.4]$, still drops into interval $[0.1, 0.5]$. From the picture, we can see that Island 2 will be copied 3 times and passed to other computers due to its high weight. Island 3 and Island 4 will be copied once. However, Island 1 and Island 5 will be killed as no random number drops into their intervals. This can happen when island has a low weight.

By double stratified resampling method, there are also $5n$ particles passed between islands when resampling between islands in this example.

The algorithm of resampling between islands and within island for double stratified resampling particle filter is as follows:

Algorithm 4 Double Stratified Resampling Particle Filter

```
1: Resampling between Islands
2: $P_i = \sum_{m=1}^{i} W_{k+1}^{m}$ for $i = 1, ..., N$, $L = 1$
3: for $m = 1$ to $N$ do
4:     Draw $\left[\frac{m-1}{N}, \frac{m}{N}\right]$-uniform $U_m$
5:     While $U_m \geq P_L$ set $L = L + 1$
6:     Set $\mathbb{X}_{k+1}^{m} \doteq \widehat{\mathbb{X}}_{k+1}^{j}$
7: end for
8: Resampling Within Island
9: for $i = 1$ to $N$ do
10:     $p_{i,j} = \sum_{m=1}^{j} w_{k+1}^{i,m}$ for $j = 1, ..., n$, $l = 1$
11:     for $m = 1$ to $n$ do
12:         Draw $\left[\frac{m-1}{n}, \frac{m}{n}\right]$-uniform $u_m$
13:         While $u_m \geq p_{i,j}$ set $l = l + 1$
14:         Set $\mathbb{X}_{k+1}^{i,m} \doteq \widehat{x}_{k+1}^{i,l}$
15:     end for
16: end for
```

2.2.4 Double Systematic Resampling

The double systematic resampling is an extension for the systematic resampling method to a web based platform. As for the double systematic particle filter, we divide the total population of particles into several islands. On each island, they are resampled by systematic resampling method. Between islands, systematic resampling method will be used to resample as well.

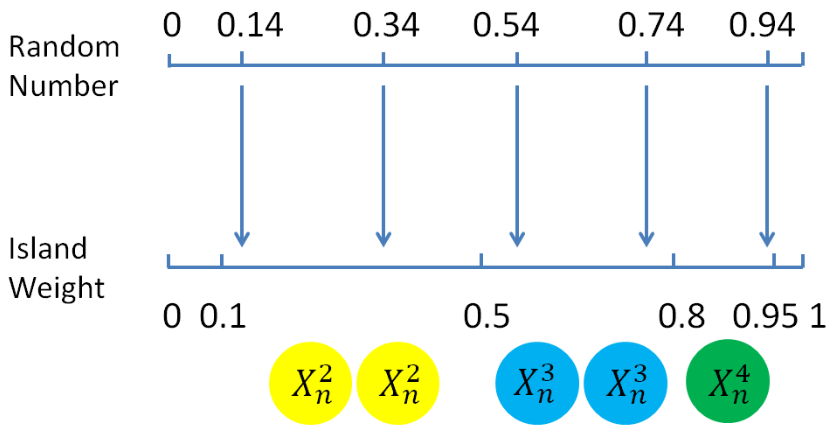


Figure 18: Resampling Between Islands of Double Systematic Resampling

Systematic resampling method is a modification of stratified resampling method which is more computationally efficient. It only generates one uniform random number. Let's show how it works for resampling between islands by a simple example.

Example 2.5. In this example, there are 5 islands $\mathbb{X}_k^1, \mathbb{X}_k^2, \mathbb{X}_k^3, \mathbb{X}_k^4, \mathbb{X}_k^5$, with normalized weight $0.10, 0.40, 0.30, 0.15$ and 0.05. We put them onto the number line to get a serial of intervals of particle weight in Figure18. Because of the randomness of resampling, here we just describe one possible case. According to the systematic resampling method, the length of random number interval is $\frac{1}{5} = 0.2$ and we start from a uniform random number in $[-0.2, 0]$, which is -0.06. Then the random numbers for selecting island are $0.14, 0.34, 0.54, 0.74$ and 0.94. Two of them drop into interval $[0.1, 0.5]$, which represents Island 2; two of them drop into interval $[0.5, 0.8]$, which represents Island 3 and one of them drops into interval $[0.8, 0.95]$, which represents Island 4. Thus Island 2 and 3 will be copied twice and Island 4 will be copied once. Then they will be passed to other computers to form new islands.

By double systematic resampling method, there are also $5n$ particles passing between islands when resampling between islands in this example.

The algorithm of resampling between islands and within island for double systematic resampling particle filter is as follows:

Algorithm 5 Double Systematic Resampling Particle Filter

```
1: Resampling between Islands
2: $P_i = \sum_{m=1}^{i} W_{k+1}^{m}$ for $i = 1, ..., N$, $j = 1$
3: Draw $\left[-\frac{1}{N}, 0\right]$-uniform $U$
4: for $m = 1$ to $N$ do
5:     Set $U_k = U + \frac{k}{N_2}$
6:     While $U_k \geq P_j$ set $j = j + 1$
7:     Set $\mathbb{X}_{k+1}^{m} \doteq \widehat{\mathbb{X}}_{k+1}^{j}$
8: end for
9: Resampling Within Island
10: for $i = 1$ to $N$ do
11:     $P_{i,j} = \sum_{m=1}^{j} w_{k+1}^{i,m}$ for $m = 1, ..., n$, $l = 1$
12:     Draw $\left[-\frac{1}{N_1}, 0\right]$-uniform $U_i$
13:     for $k = 1$ to $N_1$ do
14:         Set $U_{i,k} = U_i + \frac{k}{N_1}$
15:         While $U_{i,k} \geq P_j$ set $l = l + 1$
16:         Set $x_{n+1}^{i,k} \doteq \widehat{x}_{n+1}^{i,l}$
17:     end for
18: end for
```

2.2.5 Double Combined Resampling

The double combined resampling is an extension for combined resampling method to the web based platform. As for the double combined particle filter, we also divide the total population of particles to several islands. On each island, they are resampled by combined resampling method. Between islands, combined resampling method will be used to resample as well.

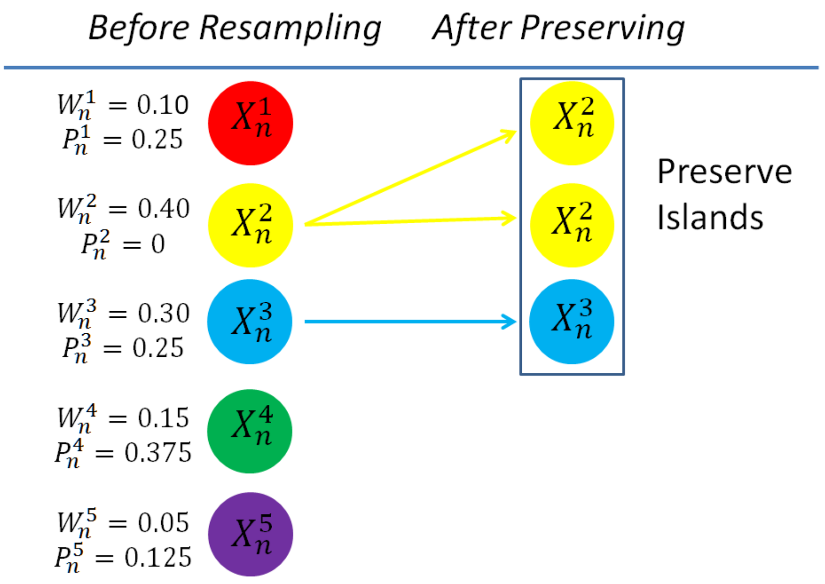


Figure 19: Preserving Islands via Double Combined Resampling

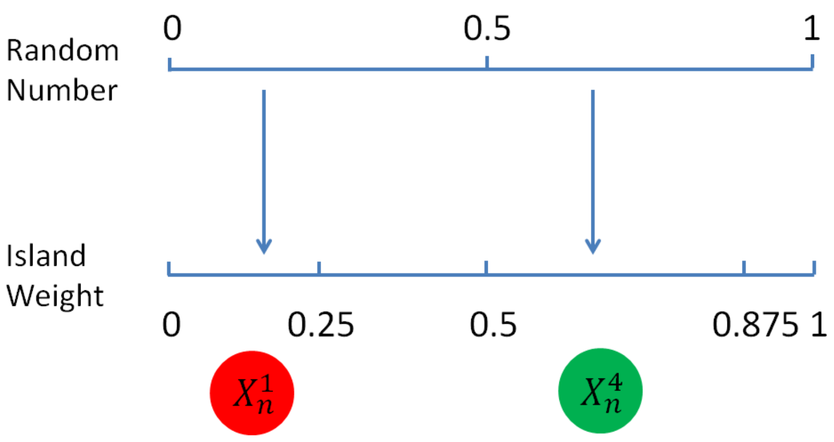


Figure 20: Resampling Islands via Double Combined Resampling

We use the same example to show how combined resampling method can be applied to resampling between islands.

Example 2.6. In this example, there are 5 islands $\mathbb{X}_k^1, \mathbb{X}_k^2, \mathbb{X}_k^3, \mathbb{X}_k^4, \mathbb{X}_k^5$, with normalized weight $0.10, 0.40, 0.30, 0.15$ and 0.05, which are displayed in Figure 19. Because of the randomness of resampling, here we just describe one possible case. In preserving step, it's same as double residual resampling, that Island 2 will be copied twice and Island 3 will be copied once. Then in resampling step, we re-calculate the weight of each particle after preserving and put them onto the number line. As we only need to select 2 more islands. thus the length of random number interval is $\frac{1}{2} = 0.5$ as we show in Fig20. At first time of resampling, it generates a uniform random number from $[0, 0.5]$ and this number drops into interval $[0, 0.25]$, which represents Island 1. Similarly, second random number, which is generated from interval $[0.5, 1]$, drops into the interval $[0.5, 0.875]$, which represents Island 4. From the picture, we can see that Island 1 and island 4 are both selected once.

By double combined resampling method, there are also $5n$ particles passed between islands when resampling between islands in this example.

The algorithm of resampling between islands and within island for double combined resampling particle filter is as follows:

Algorithm 6 Double Combined Particle Filter

```
1: Resampling between Islands
2: Preserve: $S = 0$
3: for $i = 1$ to $N$ do
4:     $m = 0$
5:     While $m < \lfloor Nw_{k+1}^{i} \rfloor$ set $m = m + 1$, $\mathbb{X}_{k+1}^{S+m} \doteq \widehat{\mathbb{X}}_{k+1}^{i}$
6:     $S = S + k$
7:     Resample: $R = N - S$; $\overline{P}_i = \sum_{m=1}^{i} \frac{Nw_{k+1}^{m} - \lfloor Nw_{k+1}^{m} \rfloor}{R}$ for $i = 1, ..., N$; $j = N - 1$
8:     for $m = N$ to $S + 1$ do
9:         Draw $\left[\frac{m-1}{R}, \frac{m}{R}\right]$-uniform $U_m$
10:        While $U_m \geq P_j$ set $j = j + 1$
11:        Set $\mathbb{X}_{k+1}^{m} \doteq \widehat{\mathbb{X}}_{k+1}^{j}$
12:    end for
13:    Resampling Within Island
14:    for $i = 1$ to $N$ do
15:        Preserve: $s = 0$
16:        for $j = 1$ to $n$ do
17:            $m = 0$
18:            While $m < \lfloor nw_{k+1}^{i,j} \rfloor$ set $m = m + 1$, $\mathbb{X}_{k+1}^{i,S+m} \doteq \widehat{\mathbb{X}}_{k+1}^{i,j}$
19:        end for
20:        Resample: $r = n - s$; $\overline{p}_{i,j} = \sum_{m=1}^{j} \frac{nw_{k+1}^{i,m} - \lfloor nw_{k+1}^{i,l} \rfloor}{r}$ for $j = 1, ..., n$; $l = n - 1$
21:        for $m = n$ to $s + 1$ do
22:            Draw $\left[\frac{m-1}{r}, \frac{m}{r}\right]$-uniform $u_m$
23:            While $u_m \geq P_j$ set $l = l + 1$
24:            Set $\mathbb{X}_{k+1}^{i,m} \doteq \widehat{\mathbb{X}}_{k+1}^{i,l}$
25:        end for
26:    end for
27: end for
```

2.2.6 Web Based Branching System

The web based residual branching particle filter is an extension of the residual branching system to a web based platform. It is relatively easy to implement a web based branching system compared to a resampled one. We still use the idea of island particle method, that divide the total population of particles to several islands. Between islands, we only need to pass the weight of the island to calculate the average weight and return it back to each island. Sometimes, to avoid all particles on one island being killed, we may need to pass a small amount of particles from the island with highest weight to the one with lowest weight. Compared to the resampled system, which copy all the data on the island to others, branching system are extremely time efficient on passing data.

The algorithm is the same as branching system on single computer platform. Figure 21 shows the idea of web based branching system.

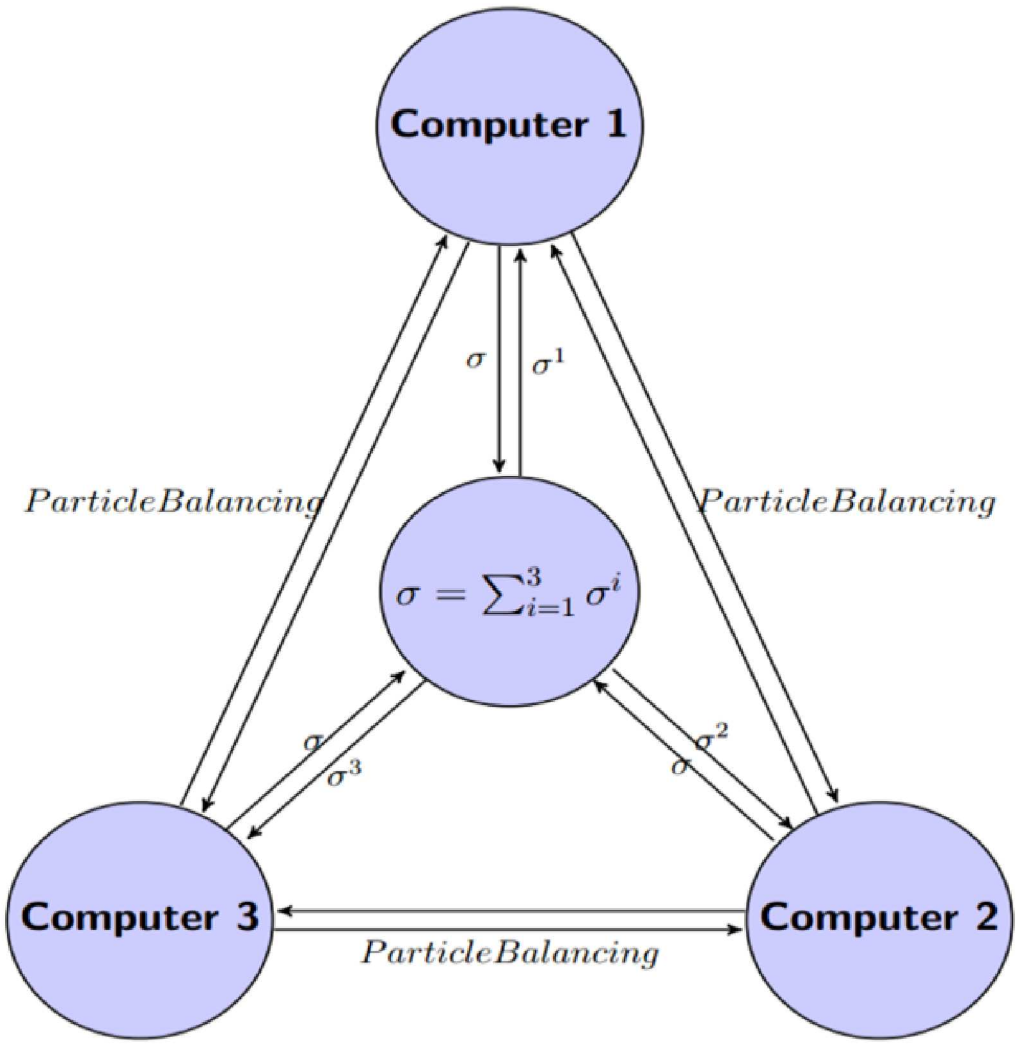


Figure 21: Web Based Branching System

Chapter 3

Simulation Results

This section is organized as follows: we first introduce the two simple problems, our *Test* and *Range-Only* problems, that will be used for the comparison purposes. Then, we compare the various double resampled particle systems discussed above on these two problems. Next, we compare the worst of our web based branching algorithms to the gamut of the double resampled particle systems discussed above and show even this most basic branching algorithm significantly outperforms all the resampled particle systems. Finally, we compare all our branching algorithms to determine which variation performs the best. For consistency, all results herein are either a *typical path* or an *average over 200 different sample paths.*

3.1 Test Model

The *Test Model* refers to the scalar signal and observation pair:

$$\begin{aligned} X_n &= 0.95X_{n-1} + 0.3W_n \\ Y_n &= X_{n-1} + V_n, \end{aligned}$$

where X_0, $\{W_n\}$ and $\{V_n\}$ are independent with standard Cauchy distribution. This is a linear, non-Gaussian filtering problem. The Kalman filter does not apply since the noise is the Cauchy random number. Indeed, conditional expectations of state do not exist since the noise is heavy-tailed. However, this problem is in other respects simple.

For the model selection, we introduce the alternative models and show that we select the correct one. We keep most of the *Test Model* the same and just vary two coefficients:

Model Number	Signal Equation	Observation Equation
-2	$X_n = 0.93X_{n-1} + 0.28W_n$	$Y_n = X_{n-1} + V_n$
-1	$X_n = 0.94X_{n-1} + 0.29W_n$	$Y_n = X_{n-1} + V_n$
0	$X_n = 0.95X_{n-1} + 0.3W_n$	$Y_n = X_{n-1} + V_n$
1	$X_n = 0.96X_{n-1} + 0.31W_n$	$Y_n = X_{n-1} + V_n$
2	$X_n = 0.97X_{n-1} + 0.32W_n$	$Y_n = X_{n-1} + V_n$

Hence, the real model is model 0, the null model.

3.2 Range Only Model

The *Range Only Model* refers to the four dimensional signal and scalar observation model:

Position	*Velocity*
$X_n = \alpha X_{n-1} + U_{n-1} + 0.3\alpha_n$	$U_n = 0.95U_{n-1} + \gamma_{n-1}$
$Z_n = \alpha Z_{n-1} + V_{n-1} + 0.3\beta_n$	$V_n = 0.95V_{n-1} + \theta_{n-1}$

$$\alpha = 0.5, \quad Y_n = \sqrt{X_{n-1}^2 + Z_{n-1}^2} + 0.1\psi_n,$$

Table 1: Range Only Model

where X_0, Z_0, U_0, V_0, $\{\gamma_n\}$, $\{\theta_n\}$, $\{\alpha_n\}$, $\{\beta_n\}$ are independent. X_0, Z_0 have 10 times the standard Cauchy distribution and U_0, V_0 have 5 times the standard

Normal distribution. Signal noise sources γ_n and θ_n have the standard normal distribution while α_n and β_n have the standard Cauchy distribution. ψ_n is the observation noise with the standard Cauchy distribution. This is a nonlinear and non-Gaussian problem but otherwise simple.

The idea of this model comes from the radar detection. Suppose that there is a radar station at the origin in the plane, (X_n, Z_n) describes the position of a ship and (U_n, V_n) its velocity. The radar produces a noise-corrupted distance observation between the ship and itself, which is Y_n in our model. The object of this problem is to estimate the state of the ship using the (back) observations.

For the model selection alternative models, we will keep most of the *Range Only Model* the same and just vary the coefficient in front of X_n and Z_n slightly:

Model Number	Signal Equation	Observation Equation
-2	$\alpha = 0.48$	$Y_n = \sqrt{X_{n-1}^2 + Z_{n-1}^2} + 0.1\psi_n$
-1	$\alpha = 0.49$	$Y_n = \sqrt{X_{n-1}^2 + Z_{n-1}^2} + 0.1\psi_n$
0	$\alpha = 0.50$	$Y_n = \sqrt{X_{n-1}^2 + Z_{n-1}^2} + 0.1\psi_n$
1	$\alpha = 0.51$	$Y_n = \sqrt{X_{n-1}^2 + Z_{n-1}^2} + 0.1\psi_n$
2	$\alpha = 0.52$	$Y_n = \sqrt{X_{n-1}^2 + Z_{n-1}^2} + 0.1\psi_n$

Hence, the real model is model 0 with the others differing slightly through α.

3.3 Comparison within Double Resampled Particle Systems

First, we compare the double resampled particle systems based on both the error (of root-mean-square type between positional tracking estimation and the real value) and the execution time.

For our *Test Model*, the error is defined as

$$error = \sqrt{\frac{1}{T}\sum_{k=1}^{T}(\pi_k^{N,n}(f) - f(X_k))^2}, \tag{3.1}$$

where $f(x)$ defined by

$$f(x) = \begin{cases} 30 & : \quad x > 30 \\ x & : \quad -30 \le x \le 30 \\ -30 & : \quad x < -30 \end{cases} .$$

where $\pi_k^{N,n}(f)$ is the filter approximation at time instant k with N islands and n particles on each island. X_k is the signal and T is the total time steps which is 35 in our experiments.

For our *Range Only Model*, the error is defined as

$$error = \frac{1}{T}\sum_{k=1}^{T}\sqrt{(\pi_k^{N,n}(g_x) - g(X_k))^2 + (\pi_k^{N,n}(g_z) - g(Z_k))^2}, \tag{3.2}$$

where $\pi_k^{N,n}$ is the normalized filter approximation at time instant k with N islands and n particles on each island, X_k, Z_k are the positional components of the real signal,

$$g(x) = \begin{cases} 1000 & : \quad x > 1000 \\ x & : \quad -1000 \le x \le 1000 \\ -1000 & : \quad x < -1000 \end{cases} .$$

and g_x, g_z denote g applied to the x and z (positional) components of the signal.

The results for error of *Test Model* and *Range Only Model* are shown in Table 2 and Table 3 respectively for the double resampled system algorithms defined in Chapter 2.

Particle Number N	**500**	**2000**	**10000**	**50000**
Double Bootstrap	7.5408	6.9203	5.3155	4.6635
Double Residual	6.6210	5.9027	5.2012	4.6467
Double Stratified	6.3037	5.6543	5.1858	4.5144
Double Systematic	6.5690	5.7674	5.2321	4.5549
Double Combined	6.2077	5.5335	5.1398	4.5096

Table 2: Average Error of Test Model

All four improved resampling methods show a significant improvement over the bootstrap algorithm with a small particle number, 500, 2000, 10000 in the *Test Model* and *Range Only Model*. When the particle number increases enough all these methods approach the optimal filter. As these two problems are not difficult as most real life problems where one has to limit the number of particles for computational reasons, the performance with these lower particle numbers are most important.

Particle Number N	**500**	**2000**	**10000**	**50000**
Double Bootstrap	53.6267	48.1276	47.5782	46.4845
Double Residual	52.5242	48.0185	47.1093	46.0613
Double Stratified	49.5776	47.8253	46.6373	46.0015
Double Systematic	51.2376	48.2155	47.3729	46.2636
Double Combined	51.7858	47.9969	47.0044	45.9785

Table 3: Average Error of Range Only Model

We should not just pick the method with lowest error as the speed is also important. The average execution time results are shown in Table 4 and Table 5 respectively.

Particle Number N	**500**	**2000**	**10000**	**50000**
Double Bootstrap	0.0043	0.0235	0.1237	0.6196
Double Residual	0.0031	0.0160	0.0892	0.4513
Double Stratified	0.0042	0.0213	0.1163	0.5927
Double Systematic	0.0035	0.0182	0.0917	0.4610
Double Combined	0.0030	0.0157	0.0792	0.4066

Table 4: Average Execution Times for Test Model

Although the residual and combined resampling are more complicated than the bootstrap, they preserve some particles without resampling thus saving a portion of the resampling computations. The stratified, combined and systematic resampling, save computations related to ordering the uniform random variables in the bootstrap method. It is reasonable that this speed advantage will be more significant with larger numbers of particles. For simple signal models (like Test), a large portion of the time is consumed generating and ordering the uniform resampling random numbers. This is efficiently done with stratification so it is reasonable that stratified, combined and systematic method can improve the speed of the *Test Model* greatly.

Particle Number N	**500**	**2000**	**10000**	**50000**
Double Bootstrap	0.0152	0.0362	0.1893	0.8195
Double Residual	0.0081	0.0359	0.1377	0.7230
Double Stratified	0.0123	0.0373	0.1591	0.8519
Double Systematic	0.0083	0.0291	0.1368	0.6579
Double Combined	0.0098	0.0351	0.1639	0.8391

Table 5: Average Execution Times for Range Only Model

With slightly larger signals such as our *Range Only Model*, which has a four dimensional signal, a lot of time is spent copying particles. Thus, the execution time may depend more on how many particles need to be copied or resampled. Hence, it is also reasonable that the residual and combined methods, which reduce the number of particles resampled, can improve execution time a lot. Naturally, the systematic method is very fast as it only uses one uniform random variable.

The number of interactions between islands is a very important parameter to describe the efficiency of the web based particle filter algorithm. The smaller this number is, the less data needs to be passed between computers. Hereafter, we define this parameter by the the number of particles passing between islands in T time steps. In our experiments, $T = 35$. Given N islands and n particles on each island in resampled system, there are nN particles passing between islands according to the algorithms in Chapter 2. Thus, during T time steps, there are totally nNT particles passing between islands. As in C++ platform, the data of particle and its weight are both in "double type", which takes 8 bytes. Based on the formula $1mb = 1024kb$ and $1kb = 1024b$, there will be $\frac{8nNT}{1024^2}mb$ data passing between islands during T time steps. Based on the data of Ookla, a broadband research company that crowdsourced research and download and upload speeds through its website SpeedTest.net, the average

Internet speed in Canada is $16.6mb/s$. Therefore we can estimate the time spent on passing data for the resampled system on web based platform by:

$$time = \frac{1}{16.6}\frac{8nNT}{1024^2} \tag{3.3}$$

We show the results in Table 6 and Table 7

Particle Number	**Interaction Number**	**Approximate Time**
500	17500	0.0080
2000	70000	0.0322
10000	350000	0.1609
50000	1750000	0.8043

Table 6: Interaction Number and Approximate Time of Passing Data for Resampled system based on *Test Model*

Particle Number	**Interaction Number**	**Approximate Time**
500	70000	0.0322
2000	280000	0.1287
10000	1400000	0.6434
50000	7000000	3.2172

Table 7: Interaction Number and Approximate Time of Passing Data for Resampled system based on *Range Only Model*

The execution time of resampled system on web based platform are composed by two factors: execution time on single computer and time spent on passing data. We show the total time, which is added by these two factors and the bootstrap factor based on this total time in Table 8 and 9.

To combine performance and speed, we define the "Bootstrap Factor" as:

$$Bootstrap \quad Factor = \frac{t_{bootstrap}}{t}, \tag{3.4}$$

where $t_{bootstrap}$ and t are the approximate execution times that the bootstrap and improved version take to reach a fixed error.

Now, we fix the *error* = 5.0 in the *Test Model* and *error* = 46.0 in the *Range Only Model* and show the minimum particle number, execution time (including execution time on single computer platform and time of passing data) and "Bootstrap Factor" for both in Tables 8 and 9.

	N	**Time**	**Bootstrap Factor**
Bootstrap	30000	0.8955	1.0000
Residual Resampling	25000	0.6536	1.3701
Stratified Resampling	15000	0.4177	2.1439
Systematic Resampling	16000	0.4146	2.1599
Combined Resampling	12000	0.2892	3.0965

Table 8: Bootstrap Factor of *Test Model* with fixed *Error*=5.0

	N	**Time**	**Bootstrap Factor**
Bootstrap	60000	4.8736	1.0000
Residual Resampling	52000	4.1393	1.1774
Stratified Resampling	50000	4.0691	1.1977
Systematic Resampling	55000	4.2762	1.1397
Combined Resampling	49000	3.9719	1.2270

Table 9: Bootstrap Factor of *Test Model* with fixed *Error*=46.0

The Bootstrap Factor compares speed of a method versus bootstrap algorithm for a given performance, thus combining accuracy and efficiency factors. The Combined Resampling method is the best choice for both models with

Bootstrap Factors of 3.0965 and 1.2270. However, we will see below that *every* branching algorithm will significantly outperform this.

3.4 Comparison between Web Based Branching and Double Resampled Particle Systems

In this section, we compare the double bootstrap and the best double resampled particle system to the web based residual branching, the most basic branching system. We show that our web based residual branching can improve both performance and execution time. For now, we define $(a_n, b_n) = (1/r, r)$, where $r \in [1, \infty]$, and refer to r as the resampling parameter. All particles will resample when $r = 1$, which we call complete resampling. No particle will resample when $r = \infty$, which means we have the weighted particle filter. We find a good fixed choice of r for the *Test Model* is 2.25 and for the *Range Only Model* is 5.

The error comparison is shown in Tables 10 and 11. Residual branching is much better than the bootstrap and even the best resampled system, combined resampling.

Particle Number N	**500**	**2000**	**10000**	**50000**
Bootstrap	7.5408	6.9203	5.3155	4.6635
Combined Resampling	6.2077	5.5335	5.1398	4.5096
Residual Branching	4.8211	4.6056	4.4185	4.2046

Table 10: Average Error of Test Model

Particle Number N	**500**	**2000**	**10000**	**50000**
Bootstrap	53.6267	48.1276	47.5782	46.4845
Combined Resampling	51.7858	47.9969	47.3044	46.1985
Residual Branching	46.7922	45.9687	45.5428	45.0540

Table 11: Average Error of Range Only Model

Speed is compared in Tables 12 and 13. The residual branching algorithm is the fastest one for both models.

Particle Number N	**500**	**2000**	**10000**	**50000**
Bootstrap	0.0043	0.0235	0.1237	0.6196
Combined Resampling	0.0030	0.0157	0.0792	0.4066
Residual Branching	0.0016	0.0078	0.0408	0.1969

Table 12: Average Execution Time of Test Model

Particle Number N	**500**	**2000**	**10000**	**50000**
Bootstrap	0.0152	0.0362	0.1893	0.8195
Combined Resampling	0.0098	0.0351	0.1639	0.8391
Residual Branching	0.0035	0.0108	0.0512	0.2513

Table 13: Average Execution Time of Range Only Model

Compared with the double resampled system, the web based branching system has huge advantage on time of passing data as it only needs to pass the average weight and small amount of particles between computers, instead of all particles in double resampled system. At each time step, we move 2% of all particles from the computer which has the highest weight to the one with lowest weight to avoid all particles are killed on it. The interaction number and approximate time of passing data for web based branching system can be calculated as the follows: given N computers, at each time step, each computer passes the average weight to one computer to calculate overall average weight and then it returns back. Thus, the interaction number for passing weight is $2N$. The interaction number for passing particles is $2\% \times nN$. Similar as calculate passing data time in double resampled system, we can estimate the time spent on passing data for web based branching system by:

$$time = \frac{1}{16.6} \frac{8(2N + 2\% \times nN)T}{1024^2} \tag{3.5}$$

We show the results in Table 14 and Table 15

Particle Number	**Interaction Number**	**Approximate Time**
500	1750	0.0008
2000	2800	0.0013
10000	8400	0.0039
50000	36400	0.0167

Table 14: Interaction Number and Approximate Time of Passing Data for Web Based Branching System on *Test Model*

Particle Number	**Interaction Number**	**Approximate Time**
500	2800	0.0013
2000	7000	0.0032
10000	29400	0.0135
50000	141400	0.0650

Table 15: Interaction Number and Approximate Time of Passing Data for Web Based Branching System on *Range Only Model*

Finally, to evaluate the advantage of residual branching on both performance and speed, we provide the Bootstrap Factor in Tables 16 and 17. In the *Test Model*, residual branching is 426.4286 and 137.7131 times better than bootstrap and combined resampled respectively. In *Range Only Model*, the improvement is also very significant at 348.1143 and 283.7117. Our better branching algorithms will be shown below to outperform yet a lot more.

	N	**Time**	**Bootstrap Factor**
Bootstrap	30000	0.8955	1.0000
Combined Resampling	12000	0.2892	3.0965
Residual Branching	400	0.0021	426.4286

Table 16: Bootstrap Factor of Test Model with Error 5.0

	N	**Time**	**Bootstrap Factor**
Bootstrap	60000	4.8736	1.0000
Combined Resampling	49000	3.9719	1.2270
Residual Branching	2000	0.0140	348.1143

Table 17: Bootstrap Factor of Range Only Model with Error 46

The model selection ability is also critical. For comparison purposes, we fix the initial number of particles to be $N = 10,000$ for all model selection experiments and show the execution time for model selection in Table 18. Web based residual branching is the fastest one for model selection as it was for tracking. Indeed, branching has another small inherent advantage here since model selection is based upon the unnormalized filter, which is already computed in the branching methods.

Model	Test Model	Range Only Model
Bootstrap	0.0973	0.1291
Combined Resampling	0.0812	0.1223
Residual Branching	0.0465	0.0831

Table 18: Average Execution Time of Model Selection

As the performance results are very similar for both the Test and Range Only models, we just apply each of these three algorithm to our *Range Only Model*. We define $Bayes\ Factor = \frac{\sigma^0(1)}{\sigma^k(1)}$, where k is the index for the different models described in Sections 3.1 and 3.2.

All three algorithms select the correct model rather convincingly. Based on Bayes Factor, it appears from these single-outcome pictures that bootstrap had the hardest time distinguishing models. Next, it appears that Combined Resampling distinguished the correct model from model 1 the best while Residual Branching distinguished the other three models better. Typically, bootstrap has the most difficult time distinguishing models and residual branching is slightly better at distinguishing models than combined resampling.

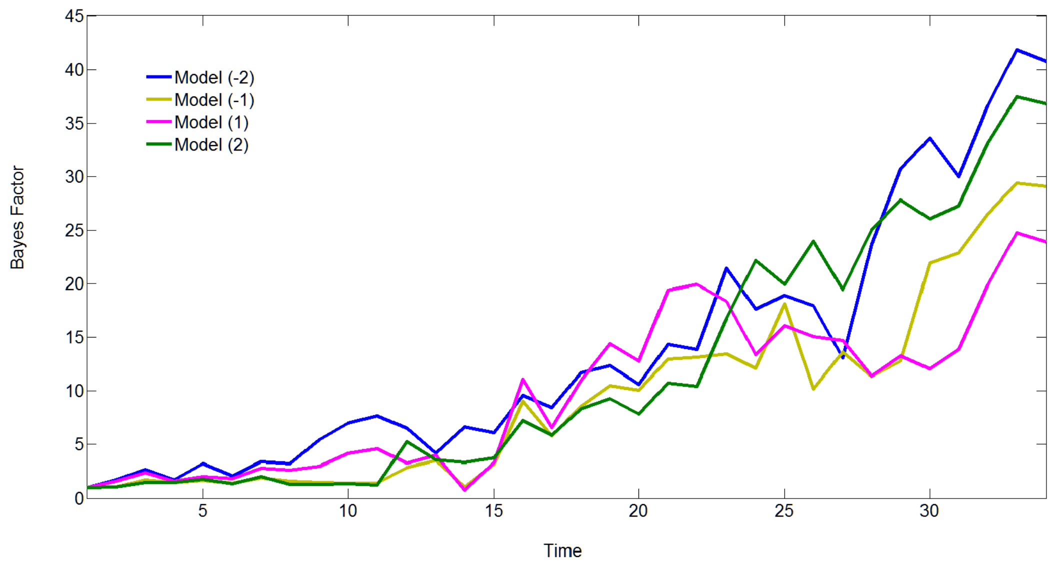


Figure 22: Bootstrap Model Selection of *Range Only Model*

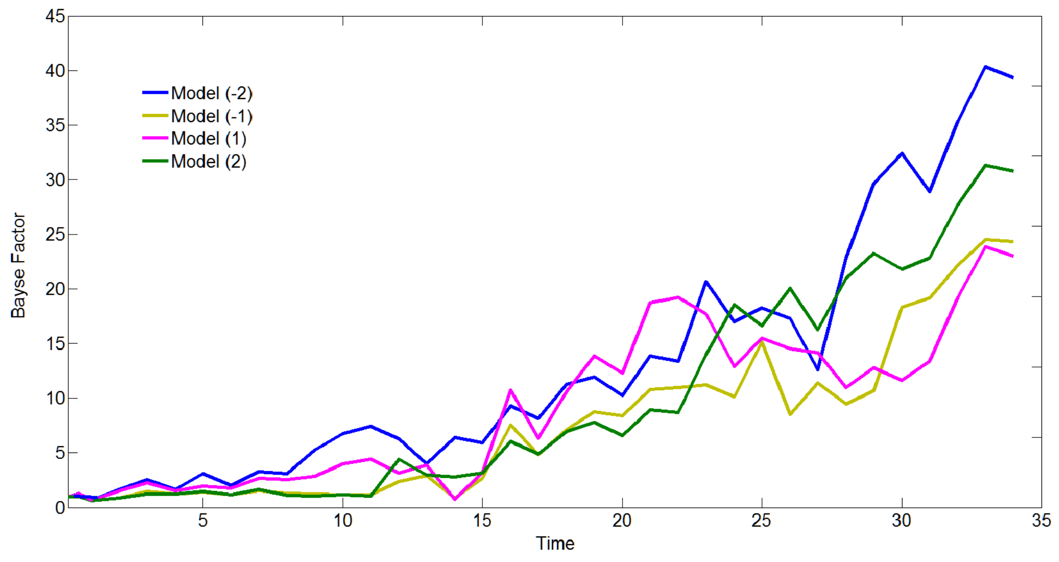


Figure 23: Combined Resampling Model Selection of *Range Only Model*

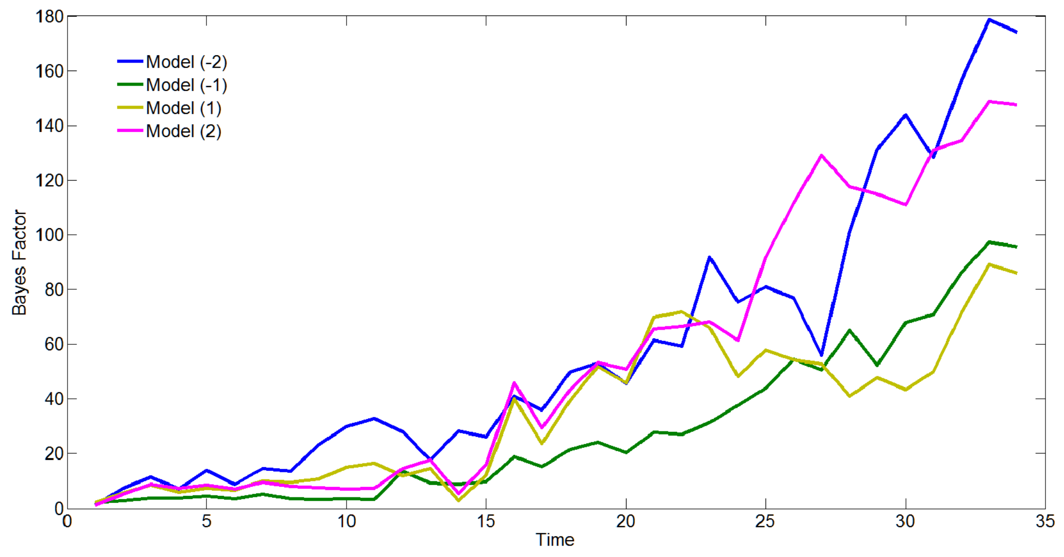


Figure 24: Residual Branching Model Selection of *Range Only Model*

3.5 Comparison within Web Based Branching Particle Systems

There are many ways to produce unbiased branching filters. The residual branching and combined branching algorithms, introduced in Chapter 3, are two of the simplest to implement. Neither requires any fundamental changes to work on multiple computers. We just pass total island weight to the other islands and transfer a very small amount of particles to even the load as shown in Figure 21. Remember, stratification is done separately on each computer so this does not present a problem.

We continue comparing error, speed and "Bootstrap Factor" but now within branching particle systems. First, we consider simple residual branching and combined branching on our test model and show combined is much better than residual and hence two levels above the double resampled particle systems. The average error is shown in Table 19

Particle Number N	**500**	**2000**	**10000**	**50000**
Residual Branching	4.8211	4.6056	4.4185	4.2046
Combined Branching	4.3926	4.2332	4.1397	4.1093

Table 19: Average Error of Test Model

and the expected execution time is shown in Table 20

Particle Number N	**500**	**2000**	**10000**	**50000**
Residual Branching	0.0016	0.0078	0.0408	0.1969
Combined Branching	0.0024	0.0081	0.0362	0.1897

Table 20: Average Execution Time of Test Model

Finally, we combine performance and execution time (including execution time on single computer and time of passing data):

	N	**Time**	**Bootstrap Factor**
Residual Branching	400	0.0021	426.4286
Combined Branching	150	0.0014	639.6429

Table 21: Bootstrap Factor of Test Model with fixed Error 5.0

Next, we apply residual branching (with $r = 5$), combined branching (with $r = 5$), dynamic branching and effective particle branching to our Range Only Model. We show the results in Table 22 and Table 23.

Particle Number N	**500**	**2000**	**10000**	**50000**
Residual Branching	46.7922	45.9687	45.5428	45.0540
Combined Branching	45.8054	45.5855	45.0531	44.9357

Table 22: Average Error of Range Only Model

Particle Number N	**500**	**2000**	**10000**	**50000**
Residual Branching	0.0035	0.0108	0.0512	0.2513
Combined Branching	0.0027	0.0113	0.0549	0.2549

Table 23: Average Execution Time of Range Only Model

Similarly, we we combine performance and execution time (including execution time on single computer and time of passing data):

	N	**Time**	**Bootstrap Factor**
Residual Branching	2000	0.0140	348.1143
Combined Branching	500	0.0037	1317.1892

Table 24: Bootstrap Factor of Range Only with Error 46.0

These results are clearly shocking. The web based combined branching is 1317 times more powerful than the double bootstrap, which is easily the most popular method today.

To demonstrate the significant improvement of branching systems over resampled systems, we show typical *Range Only Model* error versus time.

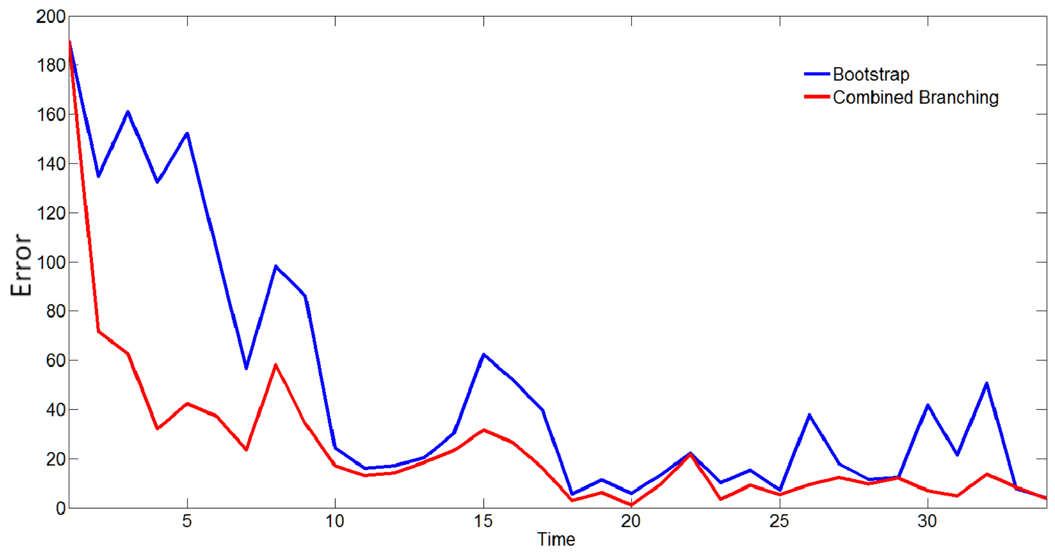


Figure 25: typical case in *Range Only Model*

Chapter 4

Appendix

4.1 Girsanov's theorem

One of the best ways of constructing particle filters is to transfer all of the information contained in the observations to a likelihood process by way of measure change.

In this reference probability method, a new fictitious probability measure Q is introduced under which the signal, observation process $\{(X_k, Y_{k+1}),\ k = 0, 1, 2, ...\}$ has the same (process) distribution as the signal, noise process $\{(X_k, V_{k+1}),\ k = 0, 1, 2, ...\}$ does under P. In another words, this means that the observations become i.i.d. random vectors with strictly-positive, bounded density function g that are independent of X under measure Q. All the observation information is absorbed into the likelihood ratio process $\{L_k,\ k = 1, 2, ...\}$ transforming Q back to P, which in our case has the form

$$\frac{dP}{dQ}\Big|_{\mathcal{F}_\infty^X \vee \mathcal{F}_k^Y} = L_k = \prod_{j=1}^{k} W_j, \quad W_j = \alpha_j(X_{j-1}), \tag{4.1}$$

and the weight function has the form

$$\alpha_j(x) = \frac{g\left(Y_j - h\left(x\right)\right)}{g\left(Y_j\right)}, \tag{4.2}$$

so $L_k = W_k L_{k-1}$ and $L_0 = 1$. The following result constructs the real probability P from the fictitious one Q.

Theorem 4.1. *Suppose $\{X_k,\ k = 0, 1, ...\}$ and $\{Y_k,\ k = 1, 2, ...\}$ are independent processes and $\{Y_k\}$ are i.i.d. with strictly-positive, bounded density g on $\mathbb{R}^d$ with some probability measure Q, and $(V_k = Y_k - h(X_{k-1}))$ for all $k = 1, 2, ...$ Then, there exists a probability measure P such that (4.1) holds, $\{V_k,\ k = 1, 2, ...\}$ are i.i.d. on $(\Omega, \mathcal{F}, P)$ with density g and $\{X_k\}$ is independent of $\{V_k\}$ with the same law as on $(\Omega, \mathcal{F}, Q)$.*

This is basically a discrete version of Girsanov's theorem. We give the proof for completeness, even though the ideas are well known.

Proof. Define P_n by Radon-Nykodym derivative

$$\frac{dP_n}{dQ} = L_n = \prod_{m=1}^{n} \frac{g\left(Y_m - h\left(X_{m-1}\right)\right)}{g\left(Y_m\right)} \tag{4.3}$$

and let $1 \leq j_1 < j_2 < \cdots < j_k \leq n$, $0 \leq i_1 < i_2 < \cdots < i_l$. Then, by the independence of X and Y under Q we have for $f_r \in B(\mathbb{R}^d)$ and $\phi_p \in B(E)$

$$\begin{aligned}
& E^{P_n}\left[\prod_{r=1}^{k} f_r(V_{j_r}) \prod_{p=1}^{l} \phi_p(X_{i_p})\right] \qquad (4.4)\\
= \;& E^{Q}\left[\prod_{m=1}^{n} \frac{g\left(Y_m - h\left(X_{m-1}\right)\right)}{g\left(Y_m\right)} \prod_{r=1}^{k} f_r(Y_{j_r} - h(X_{j_r - 1})) \prod_{p=1}^{l} \phi_p(X_{i_p})\right]\\
= \;& E^{Q}\left[\prod_{p=1}^{l} \phi_p(X_{i_p}) \int_{\mathbb{R}^d} g_1\left(y_1 - h\left(X_0\right)\right) dy_1 \cdots \int_{\mathbb{R}^d} g_n\left(y_n - h\left(X_{n-1}\right)\right) dy_n\right]\\
= \;& E^{Q}\left[\prod_{p=1}^{l} \phi_p(X_{i_p})\right] \int_{\mathbb{R}^d} g_1\left(v_1\right) dv_1 \cdots \int_{\mathbb{R}^d} g_n\left(v_n\right) dv_n\\
= \;& E^{Q}\left[\prod_{p=1}^{l} \phi_p(X_{i_p})\right] \prod_{r=1}^{k} \int_{\mathbb{R}^d} f_r(v_{j_r}) g\left(v_{j_r}\right) dv_{j_r},
\end{aligned}$$

$$\text{where } g_i = \begin{cases} g f_r & \text{if } i = j_r \\ g & \text{if } i \notin \{j_1, ..., j_k\} \end{cases} . \tag{4.5}$$

The $\{P_n\}$ are consistent by (4.4). The result follows by Kolmogorov's consistency.

4.2 Unnormalized Filter

Two nice features about σ_k are that:

1) $\sigma_k(1)$ provides the Bayes factor that $\{Y_m\}_{m=1}^k$ satisfies $Y_m = h(X_{m-1}) + V_m$ over $Y_m = V_m$, with $\{V_m\}$ being i.i.d. with density function g, since

$$\begin{aligned} \sigma_k(1) &= E^Q\left[L_k 1\left(X_k\right)\middle|\mathcal{F}_k^Y\right] \qquad (4.6) \\ &= E^Q\left[\prod_{m=1}^k \frac{g\left(Y_m - h\left(X_{m-1}\right)\right)}{g\left(Y_m\right)}\middle|\mathcal{F}_k^Y\right] \\ &= \frac{E^Q\left[\prod_{m=1}^k g\left(Y_m - h\left(X_{m-1}\right)\right)|\mathcal{F}_k^Y\right]}{E^Q\left[\prod_{m=1}^k g\left(Y_m\right)|\mathcal{F}_k^Y\right]} \end{aligned}$$

is the ratio of marginal likelihoods for these two models.

2) $\pi_k\left(f\right) = \frac{\sigma_k(f)}{\sigma_k(1)}$ by Bayes rule since

$$\begin{aligned} E^Q[\pi_k(f)\sigma_k(1)1_A] &= E^Q[E^P(f(X_k)|\mathcal{F}_k^Y)E^Q(L_k|\mathcal{F}_k^Y)1_A] \qquad (4.7) \\ &= E^Q[E^Q(L_k E^P(f(X_k)1_A|\mathcal{F}_k^Y)|\mathcal{F}_k^Y)] \\ &= E^Q[L_k E^P(f(X_k)1_A|\mathcal{F}_k^Y)] \\ &= E^P[f(X_k)1_A] \\ &= E^Q[L_k f(X_k)1_A] = E^Q[\sigma_k(f)1_A] \end{aligned}$$

for any $A \in \mathcal{F}_k^Y$.

4.3 Variance and Failure of Weighted Particle Filter

Now, it will be helpful in computing variances of the normalized filter and its approximations in the sequel to define the following.

Definition 1. The observation co-variability and variability functions are

$$\lambda(x,\xi) = \int \frac{g(y-h(x))g(y-h(\xi))}{g(y)}dy \text{ and } \overline{\lambda}(x) = \lambda(x,x). \tag{4.8}$$

Example 4.1. Suppose that g is $N(m,\gamma)$ i.e. normal with mean m and variance γ. Then, it follows easily that

$$\begin{aligned}\lambda(x,\xi) &= \frac{1}{\sqrt{2\pi}\sqrt{\gamma}}\int \exp\left(\frac{(y-m)^2-(y-h(x)-m)^2-(y-h(\xi)-m)^2}{2\gamma}\right)dy \\ &= \exp\left(\frac{h(x)h(\xi)}{\gamma}\right)\end{aligned}$$

so $\overline{\lambda}(x) = \exp\left(\frac{h^2(x)}{\gamma}\right)$.

The weighted particle filter can basically fail due to particle spread. Now we show in this section that this problem can be so bad that adding more particles still can not solve it. This failure is best explained by comparing K_λ and $\overline{K}_\lambda$, which are defined as,

$$\begin{aligned}\overline{K}_\lambda(x,dz) &= \overline{\lambda}(x)K(x,dz), && (4.9)\\ K_\lambda(x,\xi,dz,d\zeta) &= \lambda(x,\xi)K(x,dz)K(\xi,d\zeta). && (4.10)\end{aligned}$$

in a setting where explicit calculations are manageable.

Example 4.2. Suppose $h\left(x\right) = x$ and $g\left(x\right) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$ so $\lambda\left(x,\xi\right) = \exp\left(x\xi\right)$ by Example 4.1. Moreover, let $K\left(x,dz\right) = \frac{1}{\sqrt{\pi}}e^{-(x-z)^2}dz$ so $\overline{K}_\lambda\left(x,dz\right) =$

$\frac{1}{\sqrt{\pi}} e^{2xz - z^2} dz$ and we have

$$\begin{aligned} X_k &= X_{k-1} + W_k & (4.11) \\ Y_k &= X_{k-1} + V_k & (4.12) \end{aligned}$$

with $\{(W_k, V_k)\}_{k=1}^{\infty}$ being i.i.d. $(\mathcal{N}\left(0, \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \end{bmatrix}\right))$, so the filter could be solved by the Kalman filter if X_0 is independent and Gaussian. However, one might still use a particle filter if one wants to do model selection or if the initial condition is not Gaussian.

Using $(\overline{K}_\lambda^{l+1}(x, dz) = \int \overline{K}_\lambda^{l}(\zeta, dz)\, \overline{K}_\lambda(x, d\zeta))$, we find that

$$\begin{aligned} \overline{K}_\lambda^2(x, dz) &= \frac{1}{\pi}\left[\int \exp\left(2\zeta z - z^2 + 2x\zeta - \zeta^2\right) d\zeta\right] dz \\ &= \frac{1}{\pi}\int \exp\left(-\left(\zeta - x - z\right)^2\right) d\zeta \exp\left(\left(x+z\right)^2 - z^2\right) dz \\ &= \frac{1}{\sqrt{\pi}} \exp\left(x^2 + 2xz\right) dz \text{ for any } x, z \in \mathbb{R} \end{aligned}$$

and

$$\overline{K}_\lambda^3(x, dz) = \frac{1}{\pi}\left[\int \exp\left(\zeta^2 + 2\zeta z + 2x\zeta - \zeta^2\right) d\zeta\right] dz = \infty \text{ for any } x, z \in \mathbb{R}.$$

On the other hand,

$$K_\lambda(x, \xi, dz, d\zeta) = \frac{1}{\pi} e^{x\xi - x^2 - z^2 - \xi^2 - \zeta^2 + 2xz + 2\xi\zeta} dz d\zeta. \tag{4.13}$$

Hence, using $(K_\lambda^{l+1}(x, \xi, dz, d\zeta) = \int\int K_\lambda^l(y, \theta, dz, d\zeta)\, K_\lambda(x, \xi, dy, d\theta),)$ we find that

$$
\begin{aligned}
&\frac{K_\lambda^2(x,\xi,dz,d\zeta)}{dzd\zeta}\\
=\;&\frac{\exp\left(x\xi-x^2-\xi^2-z^2-\zeta^2\right)}{\pi^2}\\
\times\;&\int\int\exp\left(y\theta+2\left[xy+\xi\theta+yz+\theta\zeta-y^2-\theta^2\right]\right)dyd\theta\\
=\;&\frac{\sqrt{15}}{\pi}\exp\left(\frac{19x\xi+16xz+16\xi\zeta+4x\zeta+4z\xi+4z\zeta-7x^2-7\xi^2-7z^2-7\zeta^2}{15}\right).
\end{aligned}
$$

Moreover, letting $[a=\frac{44(16z+4\zeta+30x)+19(16\zeta+4z+30\xi)}{44^2-19^2}, b=\frac{19(16z+4\zeta+30x)+44(16\zeta+4z+30\xi)}{44^2-19^2},]$ we find

$$
\begin{aligned}
\frac{K_\lambda^3(x,\xi,dz,d\zeta)}{dzd\zeta} &= \int K_\lambda^2(y,\theta,dz,d\zeta)\,K_\lambda(x,\xi,dy,d\theta) \qquad (4.14)\\
&= \frac{\sqrt{15}}{\pi^2}\exp\left(\frac{22a^2+22b^2-19ab+4z\zeta-7z^2-7\zeta^2}{15}+x\xi-x^2-\xi^2\right)\\
&\times \int\exp\left(-\left[y-a\quad\theta-b\right]\begin{bmatrix}\frac{22}{15} & -\frac{19}{30}\\ -\frac{19}{30} & \frac{22}{15}\end{bmatrix}\begin{bmatrix}y-a\\ \theta-b\end{bmatrix}\right)dyd\theta\\
&= \frac{\sqrt{105}}{\pi}\exp\left(\frac{22a^2+22b^2-19ab+4z\zeta-7z^2-7\zeta^2}{15}+x\xi-x^2-\xi^2\right).
\end{aligned}
$$

Substituting $n=3$ as well as the values for $K_\lambda^1, K_\lambda^2, K_\lambda^3$ and $\overline{K}_\lambda^1, \overline{K}_\lambda^2, \overline{K}_\lambda^3$ into

$$
\begin{aligned}
E^Q[\gamma_n^W(f)] &= \pi_0 K_\lambda^n(f\times f)-\pi_0\times\pi_0 K_\lambda^n(f\times f) \qquad (4.15)\\
&+ \sum_{l=1}^{n}\pi_0\overline{K}_\lambda^{l-1}[\overline{K}_\lambda-K_\lambda]K_\lambda^{n-l}(f\times f)\ \forall f\in B(E)_+,
\end{aligned}
$$

we see the expected weighted particle filter variance $E^Q[\gamma_3^W(f)]=\infty$ (since $\overline{K}_\lambda^3=\infty$) for any non-trivial, non-negative f. Therefore, taking expectations in the L^2-rates, one finds $E^Q[(\sigma_3^N(1)-\sigma_3(1))^2]=\infty$ for all N so the weighted particle filter can not really work as a model selection nor a tracking device.

Notice the variance of the unnormalized filter itself $\sigma_3(f)$, given in

$$
E^Q[(\sigma_n(f)-E^Q(\sigma_n(f)))^2]=\pi_0\times\pi_0\left(K_\lambda^n(f\times f)\right)-(\pi_0(K^n f))^2, \quad (4.16)
$$

is finite as it only involves the kernals $K_\lambda^1, K_\lambda^2, K_\lambda^3$.

Remark 4.1. In the above example, one might notice that the weighted particle filter could work by just changing $h(x) = x$ to $h(x) = cx$ for small enough $c > 0$. However, this changes the (observation equation) signal-to-noise ratio and thereby fundamentally changes the problem. The observation equation is derived by the physics of a given problem. It is fine to multiply the whole observation by a constant c to obtain ($Y_k = cX_{k-1} + cV_k$) but not just one term of it. However, you will see that the $\lambda(x, \xi)$, and hence all calculations, do not change by this constant observation multiplication.

This example also illustrates the importance of the observation variability function. The unbounded nature of the observation variability function adversely affects expected variances.

Bibliography

[1] Wiener N. The interpolation, extrapolation and smoothing of stationary time series. *Report of the Services 19, Research Project DIC-6037, MIT, Cambridge, Massachusetts*, 1942.

[2] Kolmogorov A.N. Stationary sequences in hilbert space. *Bull, Moscow University*, 2(6):1–40, 1941.

[3] Swerling P. First-order error propagation in a stagewise smoothing procedure for satellite observations. *Research Memoranda. RM-2329. Santa Monica, CA: RAND Corporation.*, 1959.

[4] Kalman R.E. A new approach to linear filtering and prediction problems. *ASME. J. Basic Eng*, 82(1):35–45, 1960.

[5] Rudolph E. Kalman and Richard S. Bucy. New results in linear filtering and prediction theory. *Journal of Fluids Engineering*, 83(1):95–108, 1961.

[6] Zakai M. On the optimal filtering of diffusion processes. *Zeitschrift fr Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 11(3):230, 1969.

[7] Stratonovich R.L. Optimum nonlinear systems which bring about a separation of a signal with constant parameters from noise. *Radiofizika*, 2(6):892901, 1959.

[8] Kushner H.J. On the differential equations satisfied by conditional probability densities of markov processes, with applications. *SIAM Control Ser. A*, 2(1):106–119, 1964.

[9] Masatoshi Fujisaki, Kallianpur G., and Kunita H. Stochastic differential equations for the non linear filtering problem. *Osaka J. Math.*, 9(1):19–40, 1972.

[10] Kallianpur G. and Striebel C. Estimation of stochastic systems: Arbitrary system process with additive white noise observation errors. *Ann. Math. Statist.*, 39(3):785–801, 06 1968.

[11] Handschin J. E. Monte carlo techniques for prediction and filtering of non-linear stochastic processes. *Automatica*, 6(4):555–563, July 1970.

[12] Gordon N.J., Salmond D.J., and Smith A.F.M. Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107–113, Apr 1993.

[13] Kouritzin M. Residual and stratified branching particle filters.

[14] DelMoral P. Vergé C., Dubarry C. and Moulines E. An improved particle filter for nonlinear problems. *On parallel implementation of sequential Monte Carlo methods: the island particle model*, 23:91–107, 2013.

[15] Simon G. Doucet A. and Christophe A. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[16] Liu J.S. and Chen R. Novel approach to nonlinear/non-gaussian bayesian state estimation. *Sequential Monte-Carlo methods for dynamic systems*, 93:1032–1044, 1998.

[17] Kitagawa G. Monte-carlo filter and smoother for non-gaussian nonlinear state space models. *J. Comput. Graph. Statist.*, 1:1–25, 1996.

[18] Clifford P. Carpenter J. and Fearnhead P. An improved particle filter for nonlinear problems. *IEE Proc. Radar Sonar Navigation*, 146:2–7, 1999.

[19] Douc R. and Cappe O. Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69, 2005.