A Gauss Implementation of Particle Filters
The PF library

Thierry Roncalli
University of Evry

Guillaume Weisang
Bentley University

This version: December 24, 2008

Contents

1 Introduction **3**
1.1 Installation . . . 3
1.2 Getting started . . . 3
1.2.1 readme.txt file . . . 3
1.2.2 Setup . . . 3
1.3 What is **PF**? . . . 4
1.4 Using Online Help . . . 4

2 An introduction to particle filters **5**
2.1 Framework . . . 5
2.1.1 Definition of the tracking problem . . . 5
2.1.2 Bayesian filters . . . 5
2.2 Particle filters . . . 6
2.3 Numerical Algorithms . . . 8

3 Command Reference **13**

4 Some examples **21**

Bibliography **25**

Chapter 1

Introduction

1.1 Installation

1. The file *pf.zip* is a zipped archive file. Copy this file under the root directory of Gauss, for example **D:\GAUSS60**.

2. Unzip the file. Directories will then be created and files will be copied over them:

target_path	*readme.txt*
*target_path***dlib**	DLLs
*target_path***lib**	library file
*target_path***pf\examples**	example and tutorial files
*target_path***pf\src**	source code files
*target_path***src**	source code files

3. If your root of Gauss is **D:\GAUSS60**, the installation is finished, otherwise you have to modify the paths of the library using notepad or the LibTool. Another way to update the library is to run Gauss, l**og on to the *pf\src* directory**, delete the path with the command **lib pf -n** and add the path to the library with the command **lib pf -a**.

1.2 Getting started

Gauss 6.0.57+ for Windows is required to use the **PF** routines.

1.2.1 readme.txt file

The file *readme.txt* contains last minute information on the **PF** procedures. Please read it before using them.

1.2.2 Setup

In order to use these procedures, the **PF** library must be active. This is done by including **PF** in the LIBRARY statement at the top of your program:

```
library pf;
```

To reset global variables in subsequent executions of the program and in order to load DLLs, the following instruction should be used:

pfSet;

1.3 What is PF?

PF is a Gauss library for computing particle filters. **PF** contains the procedures whose list is given below:

- Particle_Filter_Set
- Generic_Particle_Filter
- Particle_Smoother
- Regularized_Particle_Filter
- Simulate_Tracking_Problem
- SIR_Particle_Filter
- SIS_Particle_Filter

1.4 Using Online Help

PF library supports Windows Online Help. Before using the browser, you have to verify that the **PF** library is activated by the `library` command.

Chapter 2

An introduction to particle filters

2.1 Framework

We have developed this library in the context of tracking problems [9]. In the next paragraphs, we recall the definition of tracking problems and we present Bayesian filters which are generally used to solve them.

2.1.1 Definition of the tracking problem

We follow [1] and [8] in their definition of the general tracking problem. We note $\mathbf{x}_k \in \mathbb{R}^{n_x}$ the vector of states and $\mathbf{z}_k \in \mathbb{R}^{n_z}$ the measurement vector at time index k. In our setting, we assume that the evolution of $\mathbf{x}_k$ is given by a first-order Markov model:

$$\mathbf{x}_k = f\left(t_k, \mathbf{x}_{k-1}, \boldsymbol{\nu}_k\right) \tag{2.1}$$

where f is a non-linear function and $\boldsymbol{\nu}_k$ a noise process. In general, the state $\mathbf{x}_k$ is not observed directly, but partially through the measurement vector $\mathbf{z}_k$. Thus, it is further assumed that the measurement vector is linked to the target state vector through the following measurement equation:

$$\mathbf{z}_k = h\left(t_k, \mathbf{x}_k, \boldsymbol{\eta}_k\right) \tag{2.2}$$

where h is a non-linear function, and $\boldsymbol{\eta}_k$ is a second noise process independent from $\boldsymbol{\nu}_k$. Our goal is thus to estimate $\mathbf{x}_k$ from the set of all available measurements $\mathbf{z}_{1:k} = \{\mathbf{z}_i, i = 1, \ldots, k\}$. The goal in a tracking problem is to estimate the state variable $\mathbf{x}_k$, the current state of the system at time t_k, using all available measurement $\mathbf{z}_{1:k} = \{\mathbf{z}_\ell\}_{\ell=1:k}$.

2.1.2 Bayesian filters

The prior density of the state vector at time k is given by the Chapman-Kolmogorov equation:

$$p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}\right) = \int p\left(\mathbf{x}_k \mid \mathbf{x}_{1:k-1}\right) p\left(\mathbf{x}_{k-1} \mid \mathbf{z}_{1:k-1}\right) \, \mathrm{d}\mathbf{x}_{k-1} \tag{2.3}$$

where we used the fact that our model is a first-order Markov model to write $p\left(\mathbf{x}_k \mid \mathbf{x}_{1:k-1}, \mathbf{z}_{1:k-1}\right) = p\left(\mathbf{x}_k \mid \mathbf{x}_{1:k-1}\right)$. This equation is known as the *Bayes prediction step*. It gives an estimate of the probability density function of $\mathbf{x}_k$ given all available information until

$k-1$. At time k, as a new measurement value $\mathbf{z}_k$ becomes available, one can update the probability density of $\mathbf{x}_k$:

$$p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k}\right) \propto p\left(\mathbf{z}_k \mid \mathbf{x}_k\right) p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}\right) \tag{2.4}$$

This equation is known as the *Bayes update step*. The Bayesian filter corresponds to the system of the two recursive equations (2.3) and (2.4). In order to initialize the recurrence algorithm, we assume the probability distribution of the initial state vector $p\left(\mathbf{x}_0\right)$ to be known.

Using Bayesian filters, we do not only derive the probability distributions $p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}\right)$ and $p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k}\right)$, but we may also compute the best estimates $\hat{\mathbf{x}}_{k|k-1}$ and $\hat{\mathbf{x}}_{k|k}$ which are given by:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbb{E}\left[\mathbf{x}_k \mid \mathbf{z}_{1:k-1}\right] = \int \mathbf{x}_k p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}\right) \, \mathrm{d}\mathbf{x}_k$$

and:

$$\hat{\mathbf{x}}_{k|k} = \mathbb{E}\left[\mathbf{x}_k \mid \mathbf{z}_{1:k}\right] = \int \mathbf{x}_k p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k}\right) \, \mathrm{d}\mathbf{x}_k$$

When looking at Bayesian filters, the first distinction should be between the type of state variables. On the one hand, in the case of a state variable with a finite number of discrete states, one can use Grid-based methods to get an optimal solution to the Bayesian filter, independently of the form of the density functions. On the other hand, if the state variable is continuous, then there exists no method in general providing an optimal solution, except for the normal case. Since the Gaussian family is its own conjugate, models with Gaussian densities have a particular attraction. If, furthermore, the functions f and h in (2.1) and (2.2) are linear, then the optimal solution of the Bayesian filter is given by the Kalman filter. Moreover, in the case where the noise densities are Gaussian but the functions f and h are nonlinear, one can use an approximate method called Extended Kalman filter (EKF) where the functions f and h are replaced by local linear approximation using their first derivatives at each recursion. In the more general case of non Gaussian densities, one has to resort to sub-optimal algorithms, called particle filters, to approximate the solution to the Bayesian filter. The idea behind particle filters is rather simple. Since no closed-form solution to the tracking problem can be found in general, one simply simulate at each step a sample of particles which will be used to provide a discrete estimation of the density function, the filtering density, $p(\mathbf{x}_k|\mathbf{z}_{1:k})$.

2.2 Particle filters

Particle filtering methods are techniques to implement recursive Bayesian filters using Monte-Carlo simulations. The key idea is to represent the posterior density function by a set of random samples with associated weights and to compute estimates based on these samples and weights [1, 3, 5, 6, 7, 8]. As the samples become very large $N_s \gg 1$, this Monte-Carlo approximation becomes an equivalent representation on the functional description of the posterior pdf. To clarify ideas[1], let $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1}^{N_s}$ denotes a set of support points $\left\{\mathbf{x}_k^i, i = 1, \ldots, N_s\right\}$ and their associated weights $\left\{w_k^i, i = 1, \ldots, N_s\right\}$ characterizing the posterior density $p\left(\mathbf{x}_k \mid \mathbf{z}_{0:k}\right)$. The posterior density at time k can then be approximated as:

$$p\left(\mathbf{x}_k \mid \mathbf{z}_k\right) \approx \sum_{i=1}^{N_s} w_k^i \delta\left(\mathbf{x}_k - \mathbf{x}_k^i\right) \tag{2.5}$$

[1]Note that the succinct presentation given here of particle filters is adapted to our first-order Markovian framework.

We have thus a discrete weighted approximation to the true posterior distribution. One common way of choosing the weights is by way of *importance sampling* — see for example [1, 3, 5, 8]. This principle relies on the following idea. In the general case, the probability density $p(\mathbf{x}_k \mid \mathbf{z}_k)$ is such that it is difficult to draw samples from it. Assume for a moment that $p(x) \propto \pi(x)$ is a probability density from which it is difficult to draw sample from, but for which $\pi(x)$ is easy to evaluate. Hence, up to proportionality, so is $p(x)$. Also, let $x^s \sim q(x)$ be samples that are easily drawn from a proposal $q(\cdot)$, called an *importance density*. Then, similarly to 2.5, a weighted approximation of the density $p(\cdot)$ can be obtained by using:

$$p(x) \approx \sum_{i=1}^{N_s} w^i \delta\left(x - x^i\right)$$

where:

$$w^i \propto \frac{\pi\left(x^i\right)}{q\left(x^i\right)}$$

is the normalized weight of the i-th particle. Thus, if the samples $\{\mathbf{x}_k^i\}$ were drawn from a proposal density $q(\mathbf{x}_k \mid \mathbf{z}_k)$, then the weights in (2.5) are defined to be:

$$w_k^i \propto \frac{p\left(\mathbf{x}_k^i \mid \mathbf{z}_k\right)}{q\left(\mathbf{x}_k^i \mid \mathbf{z}_k\right)} \tag{2.6}$$

The PF sequential algorithm can thus be subsumed in the following steps. At each iteration, one has samples constituting an approximation of $p\left(\mathbf{x}_{k-1}^i \mid \mathbf{z}_{k-1}\right)$ and wants to approximate $p\left(\mathbf{x}_k^i \mid \mathbf{z}_k\right)$ with a new set of samples. If the importance density can be chosen so as to factorize in the following way:

$$q(\mathbf{x}_k \mid \mathbf{z}_k) = q(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{z}_k) \times q(\mathbf{x}_{k-1} \mid \mathbf{z}_{k-1}) \tag{2.7}$$

then one can obtain samples $\{\mathbf{x}_k^i\}$ by drawing samples from $q\left(\mathbf{x}_k^i \mid \mathbf{z}_k\right)$. To derive the weight update equation:

$$\begin{aligned}
p(\mathbf{x}_k \mid \mathbf{z}_k) &= \frac{p(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{z}_{k-1}) \times p(\mathbf{x}_k \mid \mathbf{z}_{k-1})}{p(\mathbf{z}_k \mid \mathbf{z}_{k-1})} \\
&= \frac{p(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{z}_{k-1}) \times p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{z}_{k-1})}{p(\mathbf{z}_k \mid \mathbf{z}_{k-1})} \times p(\mathbf{x}_{k-1} \mid \mathbf{z}_{k-1}) \\
&= \frac{p(\mathbf{z}_k \mid \mathbf{x}_k) \times p(\mathbf{x}_k \mid \mathbf{x}_{k-1})}{p(\mathbf{z}_k \mid \mathbf{z}_{k-1})} \times p(\mathbf{x}_{k-1} \mid \mathbf{z}_{k-1}) \\
&\propto p(\mathbf{z}_k \mid \mathbf{x}_k) \times p(\mathbf{x}_k \mid \mathbf{x}_{k-1}) \times p(\mathbf{x}_{k-1} \mid \mathbf{z}_{k-1})
\end{aligned} \tag{2.8}$$

By substituting (2.7) and (2.8) into (2.6), the weight equation can be derived to be:

$$w_k^i \propto w_{k-1}^i \frac{p\left(\mathbf{z}_k \mid \mathbf{x}_k^i\right) \times p\left(\mathbf{x}_k^i \mid \mathbf{x}_{k-1}^i\right)}{q\left(\mathbf{x}_k^i \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k\right)} \tag{2.9}$$

and the posterior density $p(\mathbf{x}_k|\mathbf{z}_k)$ can be approximated using (2.5). We refer the reader to [1] for a more detailed but concise exposé of the differences between the different PF algorithms: sequential importance sampling (SIS), generic particle filter, sampling importance resampling (SIR), auxiliary particle filter (APF), and regularized particle filter (RPF). We provide a succinct exposé of the SIS, SIR algorithms as well as the generic particle filter's and the regularized particle filter's in the next section. One important feature of PF is that not one implementation is better than all the others. In different contexts, different PFs may have wildly different performances.

2.3 Numerical Algorithms

In the previous section, we presented the algorithm, known under the name Sequential Importance Sampling (SIS), which forms the basis for most sequential Monte Carlo filters developed over the past decade [1]. We start by providing its pseudo code in Algorithm 1, before exposing the more advanced algorithms we used: a generic Particle Filter (GPF), a Sampling Importance Resampling (SIR) algorithm, and a regularized Particle Filter (RPF).

Algorithm 1 SIS Particle Filter

procedure SIS_PARTICLE_FILTER($\mathbf{z}_{1:T}$,N_s) ▷ Runs a SIS Particle Filter
  $\left\{\mathbf{x}_0^i,\, w_0^i\right\}_{i=1:N_s} \sim p_0(.)$ ▷ Initialization
  $k \leftarrow 1$
  while $k < T$ **do**
    $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s} \leftarrow$ SIS_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$)
    $k \leftarrow k + 1$
  end while
  return $\left\{\mathbf{x}_{1:T}^i, w_{1:T}^i\right\}_{i=1:N_s}$
end procedure

procedure SIS_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$) ▷ Propagates the sample from state $k-1$ to state k
  for $i = 1 : N_s$ **do**
    Draw $\mathbf{x}_k^i \sim q\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k\right)$
    Assign the particle a weight, w_k^i, according to 2.9
  end for
  return $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s}$
end procedure

The SIS algorithm is thus a very simple algorithm, easy to implement. However, it commonly suffers from a degeneracy phenomenon, where after only a few iterations, all but one particle will have negligible weights. This degeneracy problems implies that a large computational effort will be devoted to updating particles whose contribution to the approximation of the filtering density $p\left(\mathbf{x}_k \mid \mathbf{z}_{1:k}\right)$ is quasi null. In order to alleviate this problem, more advanced algorithm have been devised. One way to deal with degeneracy is to carefully choose the importance density function $q\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k\right)$. We leave to the reader to consult [1] for a discussion of the importance of the choice of the importance density. Another simple idea is to resample the particles when a certain measure of degeneracy becomes too large (or too small). For example, one could calculate the effective sample size N_{eff} defined as:

$$N_{\text{eff}} = \frac{N_s}{1 + \sigma\left(w_k^{*i}\right)^2}$$

where $w_k^{*i} = p\left(\mathbf{x}_k^i \mid \mathbf{z}_{1:k}\right) / q\left(\mathbf{x}_k^i \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k\right)$ is referred to as the "true weight." As this cannot be valued exactly, this quantity can be estimated using:

$$\hat{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_s}\left(w_k^i\right)^2} \tag{2.10}$$

We provide in Algorithm 2 and in Algorithm 3 respectively the resampling algorithm we used and the generic Particle Filter which is deduced from the SIS algorithm by adding this resampling step to avoid degeneracy.

Algorithm 2 Resampling Algorithm

procedure RESAMPLE($\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s}$)
  $c_1 \leftarrow 0$ ▷ Initialise the CDF
  for $i = 2 : N_s$ **do** ▷ Construct the CDF
    $c_i \leftarrow c_{i-1} + w_k^i$
  end for

  $i \leftarrow 1$ ▷ Start at the bottom of the CDF
  $u_1 \sim \mathbb{U}\left[0, N_s^{-1}\right]$ ▷ Draw a starting point
  for $j = 1 : N_s$ **do**
    $u_j \leftarrow u_1 + N_s^{-1}(j-1)$ ▷ Move along the CDF
    while $u_j > c_i$ **do**
      $i \leftarrow i + 1$
    end while
    $\mathbf{x}_k^{j*} = \mathbf{x}_k^i$ ▷ Assign sample
    $w_k^j = N_s^{-1}$ ▷ Assign weight
    $\text{parent}_j \leftarrow i$ ▷ Assign parent
  end for
  return $\left\{\mathbf{x}_k^{j*}, w_k^j, \text{parent}_j\right\}_{j=1:N_s}$
end procedure

Algorithm 3 Generic Particle Filter

procedure GENERIC_PARTICLE_FILTER($\mathbf{z}_{1:T}$,N_s) ▷ Runs a Generic Particle Filter
  $\left\{\mathbf{x}_0^i,\, w_0^i\right\}_{i=1:N_s} \sim p_0(.)$ ▷ Initialization
  $k \leftarrow 1$
  while $k < T$ **do**
    $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s} \leftarrow$ PF_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$)
    $k \leftarrow k + 1$
  end while
  return $\left\{\mathbf{x}_{1:T}^i, w_{1:T}^i\right\}_{i=1:N_s}$
end procedure

procedure PF_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$)
  for $i = 1 : N_s$ **do**
    Draw $\mathbf{x}_k^i \sim q\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k\right)$
    Assign the particle a weight, w_k^i, according to 2.9
  end for
  $t \leftarrow \sum_{i=1}^{N_s} w_k^i$ ▷ Calculate total weight
  for $i = 1 : N_s$ **do**
    $w_k^i \leftarrow t^{-1} w_k^i$
  end for
  Calculate $\widehat{N_{\text{eff}}}$ using 2.10
  if $\widehat{N_{\text{eff}}} < N_s$ **then**
    $\left\{\mathbf{x}_k^i, w_k^i, -\right\}_{i=1:N_s} \leftarrow$ RESAMPLE($\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s}$)
  end if
end procedure

Algorithm 4 SIR Particle Filter

procedure SIR_PARTICLE_FILTER($\mathbf{z}_{1:T}$,N_s) ▷ Runs a SIR Particle Filter
  $\left\{\mathbf{x}_0^i,\, w_0^i\right\}_{i=1:N_s} \sim p_0(.)$ ▷ Initialization
  $k \leftarrow 1$
  while $k < T$ **do**
    $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s} \leftarrow$ SIR_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$)
    $k \leftarrow k + 1$
  end while
  return $\left\{\mathbf{x}_{1:T}^i, w_{1:T}^i\right\}_{i=1:N_s}$
end procedure

procedure SIR_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$)
  for $i = 1 : N_s$ **do**
    Draw $\mathbf{x}_k^i \sim p(\mathbf{x}_k \mid \mathbf{x}_{k-1}^i)$
    $w_k^i \leftarrow p(\mathbf{z}_k \mid \mathbf{x}_k^i)$
  end for
  $t \leftarrow \sum_{i=1}^{N_s} w_k^i$ ▷ Calculate total weight
  for $i = 1 : N_s$ **do**
    $w_k^i \leftarrow t^{-1} w_k^i$
  end for
  $\left\{\mathbf{x}_k^i, w_k^i, -\right\}_{i=1:N_s} \leftarrow$ RESAMPLE($\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s}$) ▷ Systematic resampling
end procedure

In many particle filters implementations, one uses the prior density $p\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}^i\right)$ as the importance density $q\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k\right)$ for even though it is often suboptimal, it simplifies the weights update equation 2.9 into:

$$w_k^i \propto w_{k-1}^i \times p\left(\mathbf{z}_k \mid \mathbf{x}_k^i\right)$$

Furthermore, if resampling is applied at every step — this particular implementation is called the Sampling Importance Resampling (SIR) of which we give the algorithm in pseudo code in Algorithm 4 — then we have $w_{k-1}^i = 1/N_s \,\forall i$, and so:

$$w_k^i \propto p\left(\mathbf{z}_k \mid \mathbf{x}_k^i\right) \tag{2.11}$$

The weights given in 2.11 are normalized before the resampling stage.

The regularized Particle Filter is based on the same idea as the Generic Particle Filter, with the same resampling condition, but the resampling step provides an entirely new sample based on a continuous approximation of the posterior filtering density $p\left(\mathbf{x}_k \mid \mathbf{z}_k\right)$, such that we have the following approximation:

$$\hat{p}\left(\mathbf{x}_k \mid \mathbf{z}_k\right) = \sum_{i=1}^{N_s} w_k^i K_h\left(\mathbf{x}_k - \mathbf{x}_k^i\right) \tag{2.12}$$

where:

$$K_h\left(\mathbf{x}\right) = \frac{1}{h^{n_x}} K\left(\frac{\mathbf{x}}{h}\right)$$

is the re-scaled Kernel density $K\left(\cdot\right)$, $h > 0$ is the Kernel bandwidth, n_x is the dimension of the state vector $\mathbf{x}$, and w_k^i, $i = 1, \ldots, N_s$ are normalized weights. The Kernel $K\left(\cdot\right)$ and bandwidth

h should be chosen to minimize the Mean Integrated Square Error (MISE), between the true posterior density and the corresponding regularized empirical representation in 2.12, defined as:

$$\text{MISE}\left(\hat{p}\right) = \mathbb{E}\left[\int \left[\hat{p}\left(\mathbf{x}_k \mid \mathbf{z}_k\right) - p\left(\mathbf{x}_k \mid \mathbf{z}_k\right)\right]^2 \mathrm{d}\mathbf{x}_k\right]$$

One can show that in the case where all the samples have the same weight, the optimal choice of the Kernel is the Epanechnikov Kernel:

$$K_{\text{opt}} = \begin{cases} \frac{n_x+2}{2c_{n_x}}\left(1 - \|x\|^2\right) & \text{if } \|x\| < 1, \\ 0 & \text{otherwise} \end{cases}$$

where c_{n_x} is the volume of the unit hypersphere in $\mathbb{R}^{n_x}$. Furthermore, when the underlying density is Gaussian with a unit covariance matrix, the optimal choice for the bandwidth is:

$$\begin{aligned} h_{\text{opt}} &= AN_s^{-\frac{1}{n_x+4}} \\ A &= \left[8c_{n_x}^{-1}\left(n_x+4\right)\left(2\sqrt{\pi}\right)^{n_x}\right]^{-\frac{1}{n_x+4}} \end{aligned}$$

We can now provide the algorithm for the regularized Particle Filter in Algorithm 5.

Algorithm 5 Regularized Particle Filter

procedure REGULARIZED_PARTICLE_FILTER($\mathbf{z}_{1:T}$,N_s) ▷ Runs a Regularized Particle Filter
  $\left\{\mathbf{x}_0^i,\, w_0^i\right\}_{i=1:N_s} \sim p_0(.)$ ▷ Initialization
  $k \leftarrow 1$
  while $k < T$ **do**
    $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s} \leftarrow$ RPF_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$)
    $k \leftarrow k + 1$
  end while
  return $\left\{\mathbf{x}_{1:T}^i, w_{1:T}^i\right\}_{i=1:N_s}$
end procedure

procedure RPF_STEP($\mathbf{x}_{k-1}^i,\, w_{k-1}^i,\, \mathbf{z}_k$)
  for $i = 1 : N_s$ **do**
    Draw $\mathbf{x}_k^i \sim q\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k\right)$
    Assign the particle a weight, w_k^i, according to 2.9
  end for
  $t \leftarrow \sum_{i=1}^{N_s} w_k^i$ ▷ Calculate total weight
  for $i = 1 : N_s$ **do**
    $w_k^i \leftarrow t^{-1} w_k^i$
  end for
  Calculate $\widehat{N_{\text{eff}}}$ using 2.10
  if $\widehat{N_{\text{eff}}} < N_s$ **then**
    Compute the empirical covariance matrix S_k of $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s}$
    Compute $\mathbf{D}_k \leftarrow \text{Chol}(S_k)$ ▷ Cholesky decomposition of S_k: $\mathbf{D}_k\mathbf{D}_k^\top = S_k$
    $\left\{\mathbf{x}_k^i, w_k^i, -\right\}_{i=1:N_s} \leftarrow$ RESAMPLE($\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s}$)
    for $i = 1 : N_s$ **do**
      Draw $\epsilon^i \sim K_{\text{opt}}$ from the Epanechnikov Kernel
      $\mathbf{x}_k^{i*} \leftarrow \mathbf{x}_k^i + h_{\text{opt}} \mathbf{D}_k \epsilon^i$
    end for

    return $\left\{\mathbf{x}_k^{i*}, w_k^i\right\}_{i=1:N_s}$
  else
    return $\left\{\mathbf{x}_k^i, w_k^i\right\}_{i=1:N_s}$
  end if
end procedure

Chapter 3

Command Reference

The following global variables and procedures are defined in **PF**. They are the *reserved words* of **PF**.

_pf_Ft, _pf_Ht, _pf_importance_density, _pf_importance_sampling, _pf_initial_density, _pf_initial_sampling, _pf_likelihood_density, _pf_Partial_Gaussian, _pf_prior_density, _pf_prior_sampling, _pf_Qt, _pf_Resampling_Threshold, _pf_Rt, _pf_Save_Particles, _rpf_Bounds; _rpf_Kernel_N.

The default global control variables are

`_pf_Resampling_Threshold`	∞
`_pf_Save_Particles`	1
`_rpf_Bounds`	0
`_pf_Partial_Gaussian`	0
`_rpf_Kernel_N`	201

Particle_Filter_Set

■ **Purpose**
Set all the information to run a PF.

■ **Format**
call Particle_Filter_Set(F,Q,H,R,
importance_pdf,prior_pdf,likelihood_pdf,initial_pdf,
importance_sampling,prior_sampling,initial_sampling);

■ **Input**

F	scalar, pointer to a procedure that computes $F(t,x)$
Q	scalar, pointer to a procedure that computes $Q(t,x)$
H	scalar, pointer to a procedure that computes $H(t,x)$
R	scalar, pointer to a procedure that computes $R(t,x)$
importance_pdf	scalar, pointer to a procedure that computes the importance density function $q(x_k \mid x_{k-1}, z_k)$
prior_pdf	scalar, pointer to a procedure that computes the prior density function $p(x_k \mid x_{k-1})$
likelihood_pdf	scalar, pointer to a procedure that computes the likelihood density function $p(z_k \mid x_k)$
initial_pdf	scalar, pointer to a procedure that computes the initial density function $p(x_0)$
importance_sampling	scalar, pointer to a procedure that simulates the state vector according to the importance density
prior_sampling	scalar, pointer to a procedure that simulates the state vector according to the prior density
initial_sampling	scalar, pointer to a procedure that simulates the initial state vector x_0

■ **Output**

■ **Globals**

■ **Remarks**
The function F, Q, H and R are only need if you want to perform Monte Carlo runs. In this case, the model considered by the procedure `Simulate_Particle_Filter` is the following:

$$\begin{cases} \mathbf{x}_k = F(t_k, \mathbf{x}_{k-1}) + \boldsymbol{\nu}_k \\ \mathbf{z}_k = H(t_k, \mathbf{x}_k) + \boldsymbol{\eta}_k \end{cases}$$

with $\boldsymbol{\nu}_k \sim \mathcal{N}(\mathbf{0}, Q(t_k, \mathbf{x}_{k-1}))$ and $\boldsymbol{\eta}_k \sim \mathcal{N}(\mathbf{0}, R(t_k, \mathbf{x}_k))$. If you don't need to use this procedure, you may set the pointers to these functions equal to 0.

The procedure `initial_pdf` computes the initial weights w_0. If its pointer is set to zero, the initial weights are uniform.

■ **Source**
pf.src

Generic_Particle_Filter

Purpose

Run a generic particle filter.

Format

{x,w,m,cov} = Generic_Particle_Filter(z,Ns);

Input

z	matrix $N \times G$, observed values z_k
Ns	scalar, number of particles

Output

x	array $N \times N_s \times P$, sampled particles at each step
w	matrix $N \times N_s$, weights associated with the samples
m	matrix $N \times P$, mean vector of the sample
cov	array $N \times P \times P$, covariance matrix of the sample

Globals

_pf_Save_Particles	scalar, 1 to save the particles (default), 0 if not

Remarks

Combining the outputs provides the empirical posterior density at each step which can be approximated by:

$$p\left(\mathbf{x}_k = \mathbf{x} \mid \mathbf{z}_k\right) = \sum_{i=1}^{N_s} w_k^i \delta\left(\mathbf{x} - \mathbf{x}_k^i\right)$$

Source

pf.src

Particle_Smoother

■ **Purpose**
Run a properly defined particle smoother, by drawing N_s realizations of $p\left(x_k \mid z_{1:K}\right)$.

■ **Format**
{ps_x,ps_w} = Particle_Smoother(pf_x,pf_w,Ns);

■ **Input**

pf_x	array $N \times N_s \times P$, stored particles from the run of a particle filter
pf_w	matrix $N \times N_s$, stored weights associated to the samples from the PF run
Ns	scalar, number of simulations

■ **Output**

ps_x	array $N \times N_s \times P$, smoothed particles at each step
ps_w	matrix $N \times N_s$, weights associated with the samples ps_x

■ **Globals**

■ **Remarks**
Running a particle smoother requires to run a particle filter first, and feeding the smoothing procedure with the output of the PF run. This algorithm assumes that the procedures used are based on Importance Resampling. Finally, the size of the samples of smoothed particles is equal to the size of the samples from the particle filter.

■ **Source**
pf.src

Regularized_Particle_Filter

Purpose

Run a regularized particle filter.

Format

{x,w,m,cov} = Regularized_Particle_Filter(z,Ns);

Input

z	matrix $N \times G$, observed values z_k
Ns	scalar, number of particles

Output

x	array $N \times N_s \times P$, sampled particles at each step
w	matrix $N \times N_s$, weights associated with the samples
m	matrix $N \times P$, mean vector of the sample
cov	array $N \times P \times P$, covariance matrix of the sample

Globals

_pf_kernel	scalar, defines the kernel for the regularization step (default = 1) 1 for the Epanechnikov kernel 2 for the Gaussian kernel
_pf_Save_Particles	scalar, 1 to save the particles (default), 0 if not

Remarks

Combining the outputs provides the empirical posterior density at each step which can be approximated by:

$$p\left(\mathbf{x}_k = \mathbf{x} \mid \mathbf{z}_k\right) = \sum_{i=1}^{N_s} w_k^i \delta\left(\mathbf{x} - \mathbf{x}_k^i\right)$$

Source

pf.src

Simulate_Tracking_Problem

■ **Purpose**
Simulate a tracking problem for Monte Carlo analysis.

■ **Format**
{t,x,z} = Simulate_Tracking_Problem(x0,N,Ns);

■ **Input**

x0	vector $P \times 1$, initial values x_0
N	scalar, number of time periods
Ns	scalar, number of simulations

■ **Output**

t	vector $N \times 1$, time index k
x	aray $N \times N_s \times P$, sample of x_k
z	aray $N \times N_s \times G$, sample of z_k

■ **Globals**

■ **Remarks**
The model considered by the procedure `Simulate_Tracking_Problem` is the following:

$$\begin{cases} \mathbf{x}_k = F\left(t_k, \mathbf{x}_{k-1}\right) + \boldsymbol{\nu}_k \\ \mathbf{z}_k = H\left(t_k, \mathbf{x}_k\right) + \boldsymbol{\eta}_k \end{cases}$$

with $\boldsymbol{\nu}_k \sim \mathcal{N}\left(\mathbf{0}, Q\left(t_k, \mathbf{x}_{k-1}\right)\right)$ and $\boldsymbol{\eta}_k \sim \mathcal{N}\left(\mathbf{0}, R\left(t_k, \mathbf{x}_{k-1}\right)\right)$. The functions F, Q, H and R are initialized using the procedure `Particle_Filter_Set`.

■ **Source**
pf.src

SIR_Particle_Filter

Purpose

Run a SIR particle filter.

Format

{x,w,m,cov} = SIR_Particle_Filter(z,Ns);

Input

z	matrix $N \times G$, observed values z_k
Ns	scalar, number of particles

Output

x	array $N \times N_s \times P$, sampled particles at each step
w	matrix $N \times N_s$, weights associated with the samples
m	matrix $N \times P$, mean vector of the sample
cov	array $N \times P \times P$, covariance matrix of the sample

Globals

_pf_Save_Particles	scalar, 1 to save the particles (default), 0 if not

Remarks

Combining the outputs provides the empirical posterior density at each step which can be approximated by:

$$p\left(\mathbf{x}_k = \mathbf{x} \mid \mathbf{z}_k\right) = \sum_{i=1}^{N_s} w_k^i \delta\left(\mathbf{x} - \mathbf{x}_k^i\right)$$

Source

pf.src

SIS_Particle_Filter

■ **Purpose**
Run a SIS particle filter.

■ **Format**
{x,w,m,cov} = SIS_Particle_Filter(z,Ns);

■ **Input**

z	matrix $N \times G$, observed values z_k
Ns	scalar, number of particles

■ **Output**

x	array $N \times N_s \times P$, sampled particles at each step
w	matrix $N \times N_s$, weights associated with the samples
m	matrix $N \times P$, mean vector of the sample
cov	array $N \times P \times P$, covariance matrix of the sample

■ **Globals**

_pf_Save_Particles	scalar, 1 to save the particles (default), 0 if not

■ **Remarks**
Combining the outputs provides the empirical posterior density at each step which can be approximated by:

$$p\left(\mathbf{x}_k = \mathbf{x} \mid \mathbf{z}_k\right) = \sum_{i=1}^{N_s} w_k^i \delta\left(\mathbf{x} - \mathbf{x}_k^i\right)$$

■ **Source**
pf.src

Chapter 4

Some examples

1. **example1.prg**
   We consider the example[1] of Arulampalam *et al.* [1]:

$$\begin{cases} p\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}\right) = \mathcal{N}\left(F_k\left(\mathbf{x}_{k-1}\right), Q_k\right) \\ p\left(\mathbf{z}_k \mid \mathbf{x}_k\right) = \mathcal{N}\left(\frac{\mathbf{x}_k^2}{20}, R_k\right) \end{cases}$$

   or equivalently:

$$\begin{cases} \mathbf{x}_k = F_k\left(\mathbf{x}_{k-1}\right) + \boldsymbol{\nu}_k \\ \mathbf{z}_k = \frac{\mathbf{x}_k^2}{20} + \boldsymbol{\eta}_k \end{cases}$$

   where:

$$F_k\left(\mathbf{x}_{k-1}\right) = \frac{\mathbf{x}_{k-1}}{2} + \frac{25\mathbf{x}_{k-1}}{1+\mathbf{x}_{k-1}^2} + 8\cos\left(1.2k\right)$$

   We have $\boldsymbol{\nu}_k \sim \mathcal{N}\left(\mathbf{0}, Q_k\right)$ and $\boldsymbol{\eta}_k \sim \mathcal{N}\left(\mathbf{0}, R_k\right)$. We use $Q_k = 1$ and $R_k = 10$. Using the procedure `Particle_Filter_Set`, we build the corresponding tracking problem. We consider 1000 particles and perform a Monte Carlo analysis in order to compare the RMSE between SIS, GPF, SIR and RPF algorithms (Figure 4.1). In Figure 4.2, we report the results of one MC trial.

2. **example2.prg**
   We use the previous tracking problem in order to illustrate the influence of the number of particles in the convergence of the SIS algorithm. Results are reported in Figure 4.3.

3. **example3.prg**
   Same example than the **example2.prg** program, but with the SIR algorithm. Results are reported in Figure 4.4.

4. **example4.prg**
   We estimate the probability density of the state variables using the RPF algorithm and represent it in Figures 4.5 and 4.6.

5. **example5.prg**
   In this program, we reproduce the example[2] of Roncalli and Weisang [9]. Results are reported in Figure 4.7.

[1] This example has been already studied by Carlin *et al.* [2] and Kitagawa [4].
[2] Appendix C, Figure 28, page 65.

Figure 4.1: Density of the RMSE statistic

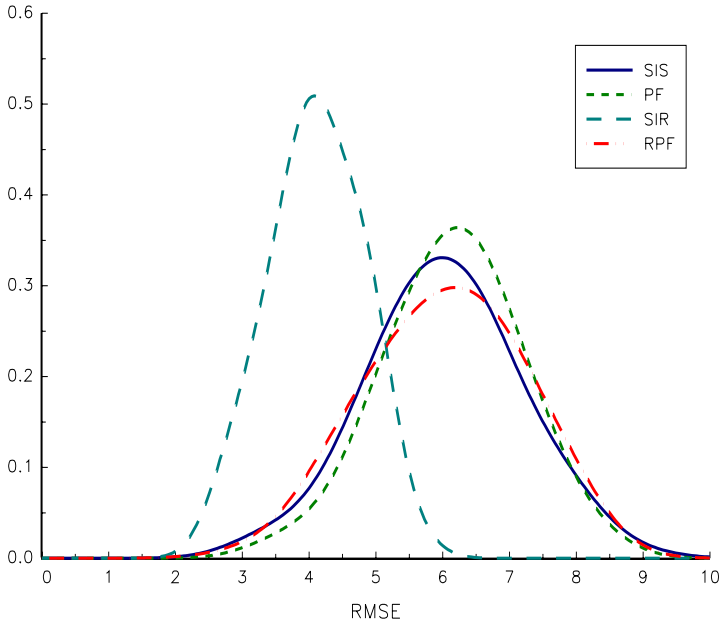


Figure 4.2: An example of a MC run

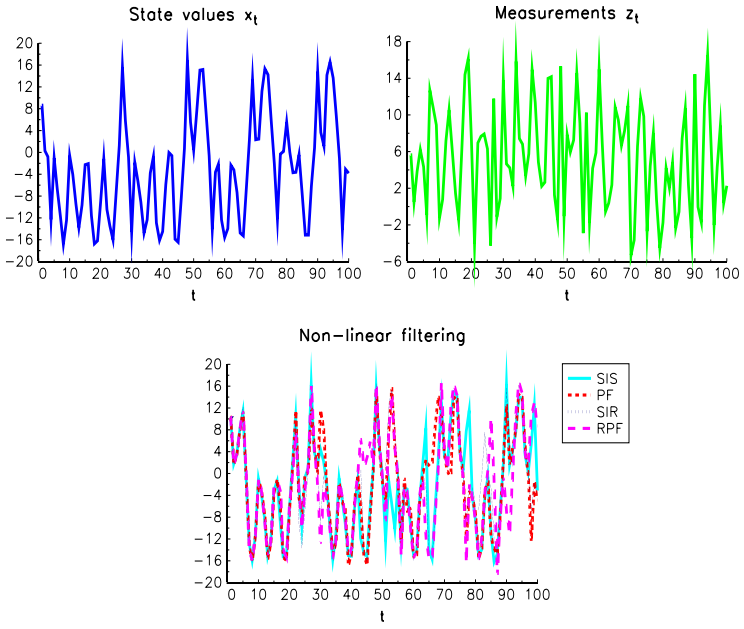


Figure 4.3: Density of the RMSE statistic for the SIS algorithm

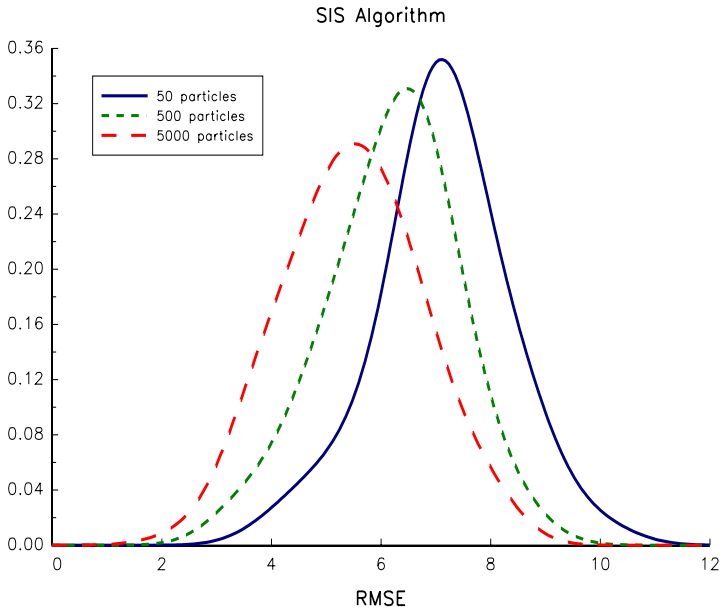


Figure 4.4: Density of the RMSE statistic for the SIR algorithm

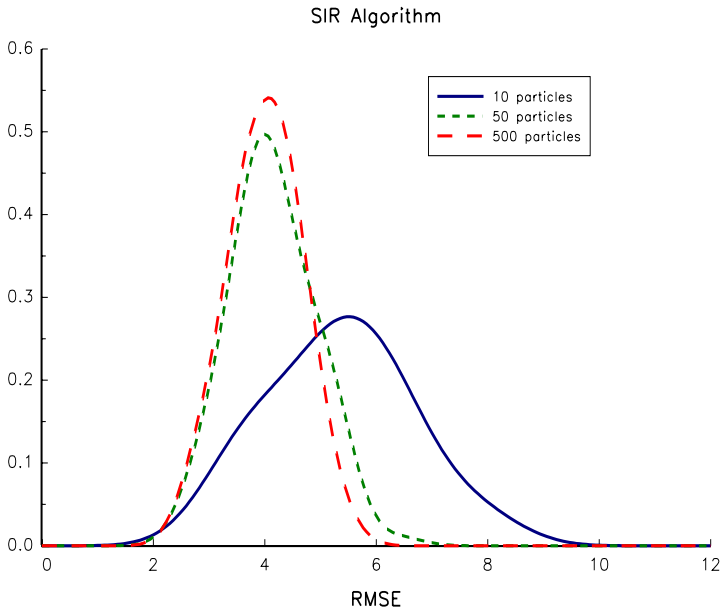


Figure 4.5: Probability density evolution (particles representation)

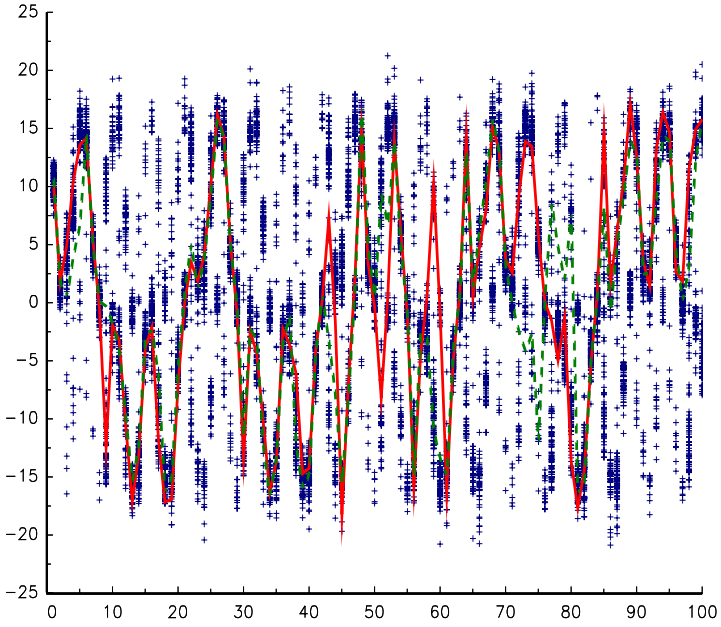
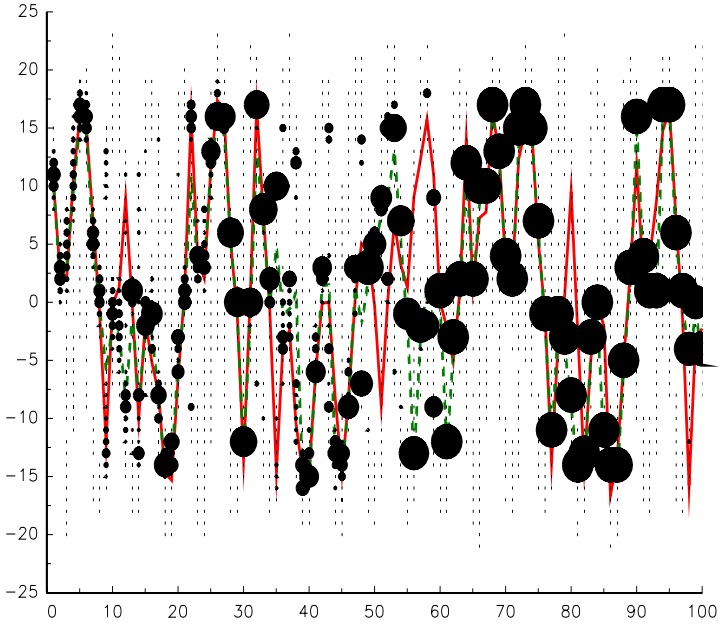
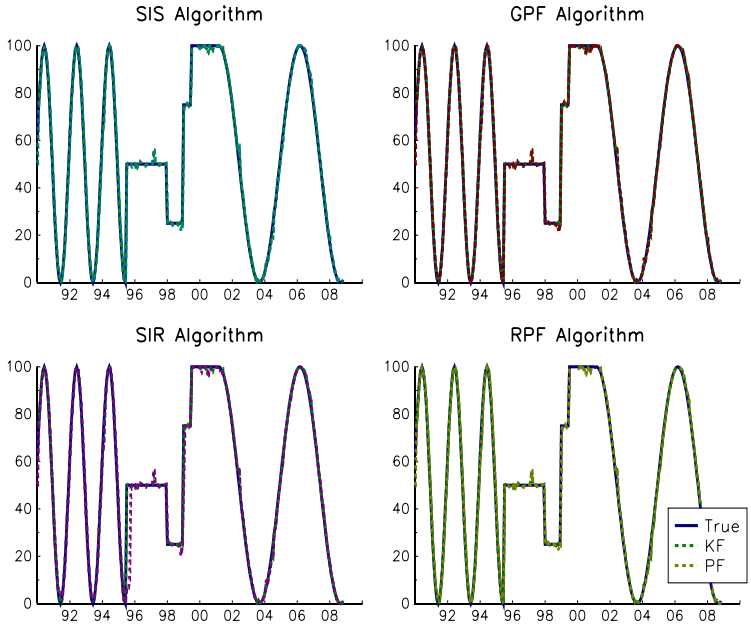


Figure 4.6: Probability density evolution (mass probability representation)



6. **example6.prg**
   An example to illustrate the procedure `Particle_Smoother`.

Figure 4.7: Solving a GTAA tracking problem with particle filters



Bibliography

[1] Sanjeev Arulampalam, Simon Maskell, Neil J. Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transaction on Signal Processing*, 50(2):174–188, February 2002.

[2] Bradley P. Carlin, Nicholas G. Polson, and David S. Stoffer. A monte carlo approach to nonnormal and nonlinear state space modeling. *Journal of the American Statistical Association*, 87(418):493–500, 1992.

[3] Michael S. Johannes and Nick Polson. Particle filtering and parameter learning. *University of Chicago, Working Paper*, March 2007.

[4] Genshiro Kitagawa. Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1–25, 1996.

[5] Michael K. Pitt and Neil Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, June 1999.

[6] George Poyiadjis, Arnaud Doucet, and Sumeetpal S. Singh. Maximum likelihood parameter estimation in general state-space models using particle methods. In *Proceedings of the American Statistical Association, JSM 05*, August 2005.

[7] George Poyiadjis, Arnaud Doucet, and Sumeetpal S. Singh. Particle methods for optimal filter derivative: application to parameter estimation. In *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, March 2005.

[8] Branko Ristic, Sanjeev Arulampalam, and Neil J. Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, Boston, 1st edition, 2004.

[9] Thierry Roncalli and Guillaume Weisang. Tracking problems, hedge fund replication and alternative beta. *Working Paper*, 2008. Available at SSRN: `http://ssrn.com/abstract=1325190`.