Nonlinear Filtering using Particles and Quadrature

A presentation in Professor B. Rozovskii's Stochastic PDE class

Andreas Klöckner <kloeckner@dam.brown.edu>

Table of contents

Sources and Program Code

The material for Sections 1 through 2.5 originates from [1] and [2], for Section 2.6 from [4], for Section 8 from [5]. You may browse and download the code belonging to this presentation at `http://git.tiker.net/?p=spde-smc.git;a=tree`.

1 Nonlinear Filtering in Discrete Time

The setup for the *nonlinear filtering problem* in discrete time consists of

- An unobserved *Markov process* $\boldsymbol{x}_t$ for $t = 0, 1, 2, \ldots$ with initial distribution $p(\boldsymbol{x}_0)$ and transition density $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$. By the Markov property,

$$p(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_0) \prod_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}).$$

- Conditionally independent *observations* $\boldsymbol{y}_t$ for $t = 1, 2, 3...$ characterized by the density $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$. Conditional independence manifests itself as

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{x}_t).$$

Our goal is to obtain (an estimate of) the *posterior distribution* $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$. A first step in this direction is made by obtaining the joint distribution $p(\boldsymbol{x}_{0:t}|\boldsymbol{y}_{1:t})$, which is nothing but *Bayes' Theorem*:

$$p(X|Y) = \frac{p(X \cap Y)}{p(Y)} = \frac{p(Y|X)p(X)}{p(Y)} = \frac{p(Y|X)p(X)}{\int P(Y \cap X)\mathrm{d}X} = \frac{p(Y|X)p(X)}{\int p(Y|X)p(X)\mathrm{d}X}.$$

$$p(\boldsymbol{x}_{0:t}|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_{1:t}|\boldsymbol{x}_{0:t})p(\boldsymbol{x}_{0:t})}{\int p(\boldsymbol{y}_{1:t}|\boldsymbol{x}_{0:t})p(\boldsymbol{x}_{0:t})\mathrm{d}\boldsymbol{x}_{0:t}}$$

This is nice, but it has an important shortcoming: It is not very suitable for on-line processing, i.e. as the process is happening, because we need to store and reprocess data for all time at each time step, which is prohibitive for long-running processes. This has several important consequences:

- We need a *recursive algorithm* that can simply update the results obtained at time $t-1$ for time t. In particular, we assume that we cannot afford to store data for all time, much less process it.
- More precisely: Storage and effort required for the recursion should be roughly constant in time, it should not grow with t.

Somewhat surprisingly, there is an *update formula* for the joint posterior that allows us to turn the joint posterior at step t into the joint posterior for step $t+1$. The two assumptions on our processes, namely Markov and conditional independence, are crucial for its derivation:

$$\begin{aligned}
\frac{p(\boldsymbol{x}_{0:t}|\boldsymbol{y}_{1:t})}{p(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1})} &= \frac{p(\boldsymbol{y}_{1:t}|\boldsymbol{x}_{0:t})p(\boldsymbol{x}_{0:t})}{p(\boldsymbol{y}_{1:t})} \cdot \frac{p(\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{y}_{1:t-1}|\boldsymbol{x}_{0:t-1})p(\boldsymbol{x}_{0:t-1})} \\
&\overset{\text{CI}}{=} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_{0:t})p(\boldsymbol{x}_{0:t})}{p(\boldsymbol{y}_{1:t})} \cdot \frac{p(\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{x}_{0:t-1})} \\
&\overset{(*)}{=} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_{0:t})p(\boldsymbol{x}_{0:t})p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}{p(\boldsymbol{y}_{1:t})} \cdot \frac{p(\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{x}_{0:t})} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_{0:t})p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}{p(\boldsymbol{y}_{1:t})} \cdot p(\boldsymbol{y}_{1:t-1}) \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})},
\end{aligned}$$

where we have used

$$p(\boldsymbol{x}_{0:t}) = p(\boldsymbol{x}_{0:t-1})p(\boldsymbol{x}_t|\boldsymbol{x}_{0:t-1}) \overset{\text{Markov}}{=} p(\boldsymbol{x}_{0:t-1})p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$$

in step $(*)$. Altogether, we have derived

$$p(\boldsymbol{x}_{0:t}|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} p(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1}). \tag{1}$$

By realizing that

$$p(\boldsymbol{y}_t) = \int p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t)\mathrm{d}\boldsymbol{x}_t,$$

the denominator above can be expanded as

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_t.$$

As we see, this update formula is not recursive yet, because we require the marginal $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$. Further, at time t, we really only care about the marginal distribution of $\boldsymbol{x}_t$ and not of $\boldsymbol{x}_{0:t}$, so the above joint dis-

tribution really has too much information.

Starting with the equality

$$p(\boldsymbol{x}_{t+1}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_t)\mathrm{d}\boldsymbol{x}_t,$$

we find

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_t \tag{2}$$

for a *prediction formula*.

Note that we are looking for a *recursion formula* for $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$, but (2) is not recursive yet. It is only one half of our final recursion formula.

To complete the recursion and with the observation $\boldsymbol{y}_t$ on board, we find

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) &= \int p(\boldsymbol{x}_{0:t}|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_{0:t-1} \\
&\overset{(1)}{=} \int \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} p(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_{0:t-1} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_{0:t-1} \\
&\overset{\text{Markov}}{=} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} \int p(\boldsymbol{x}_t|\boldsymbol{x}_{0:t-1})p(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_{0:t-1} \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) \\
&= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int\ p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_t},
\end{aligned}$$

yielding the *update formula for the marginal distribution* of $\boldsymbol{x}_t$

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int\ p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_t} \tag{3}$$

which completes our recursion.

2 Monte Carlo Filters

In order to be able to give a meaningful intuition of the behavior of the algorithms introduced, we begin by fixing a stochastic process to which we will apply our methods.

2.1 The Example Process

Throughout this presentation, we will focus on one particular example process, given by

$$\begin{aligned}
x_t &= \frac{1}{2}x_{t-1} + 25\frac{x_{t-1}}{1+x_{t-1}^2} + 8\cos(1.2t) + v_t \\
y_t &= \frac{x_t^2}{20} + w_t
\end{aligned}$$

with $x_0 \sim \mathcal{N}(0, \sigma_1^2)$, $v_t \sim \mathcal{N}(0, \sigma_v^2)$, $w_t \sim \mathcal{N}(0, \sigma_w^2)$. With the time dependency omitted, the dynamics of this process can be guessed from Figure 1. In particular, there are stable fixed points at $x = \pm\ 7$ and an unstable fixed point at $x = 0$. Note in particular that we are observing the square of the process, so just from the observations, we will never be sure about the sign of x_t. The resulting probability distribution will thus have two modes and is a somewhat canonical example of a nonlinear process that makes the Kalman filter fail.

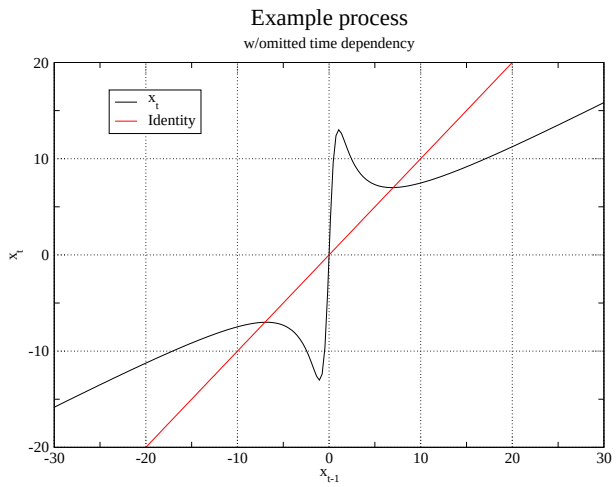

Figure 1. Plot of the Transition function of the example process, omitting the time dependency.

2.2 The Particle Filtering Framework

We begin by assuming that we have a discrete approximation $\hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ to the exact distribution $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$, given by a number N_t of particles $\boldsymbol{x}_t^{(i)}$, such that

$$\hat{p}(\mathrm{d}\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) = \frac{1}{N_t}\sum_{i=1}^{N_t} \delta_{\boldsymbol{x}_t^{(i)}}(\mathrm{d}\boldsymbol{x}_t). \tag{4}$$

We might then apply both halves of our recursion, first rewriting the prediction formula as

$$\hat{p}(\mathrm{d}\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) = \frac{1}{N_t}\sum_{i=1}^{N_t} p(\mathrm{d}\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)}) \tag{5}$$

and then sampling a new set of particles $\boldsymbol{x}_{t+1}^{(i)}$ from the distribution

$$\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t+1}) = \frac{1}{\tilde{C}_{t+1}} \cdot p(\boldsymbol{y}_{t+1}|\boldsymbol{x}_{t+1})p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}),$$

which is to say

$$\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t+1}) = \frac{1}{\tilde{C}_{t+1}\cdot N_t}\sum_{i=1}^{N_t} p(\boldsymbol{y}_{t+1}|\boldsymbol{x}_{t+1})p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)}). \tag{6}$$

Note that we named the normalization constant in (3):

$$C_t := \int p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\mathrm{d}\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}),$$

which, in the current framework, would be approximated by

$$\tilde{C}_t \approx \int p(\boldsymbol{y}_t|\boldsymbol{x}_t)\frac{1}{N_t}\sum_{i=1}^{N_t} p(\mathrm{d}\boldsymbol{x}_t|\boldsymbol{x}_{t-1}^{(i)}). \tag{7}$$

However, integrating (7) and sampling from (6) is usually still nontrivial. These holes in the framework are to be filled by the concrete filtering methods in the rest of this presentation.

2.3 Importance Sampling

Let us begin this section with a few words on *importance sampling*. Importance sampling solves the problem of sampling from a probability density $p(x)$ which is only known up to a proportionality constant. The trick is to allow each particle a *weight* in addition to its position.

More precisely, suppose we know that $p(x) \propto q(x)$ is a probability density from which we cannot easily draw samples, but where $q(x)$ is known. Next assume we have a different density $\pi(x)$, the so-called *importance density*, that is easy to sample from. Let $x^{(i)}$ be samples drawn from $\pi(x)$. Next, compute the

weights

$$w^{(i)} \propto \frac{p(x^{(i)})}{\pi(x^{(i)})}$$

and the normalized weights

$$\tilde{w}^{(i)} := \frac{w^{(i)}}{\sum_j w^{(j)}}.$$

Then

$$\hat{p}(x) := \sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x - x^{(i)})$$

is an approximate density of p. To see this, consider

$$\begin{aligned}
E_{\hat{p}}[f(x)] &= \int f(x)\hat{p}(x)\pi(x)\mathrm{d}x \\
&= \int f(x) \sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x - x^{(i)})\pi(x)\mathrm{d}x \\
&= \sum_{i=1}^{N} f(x^{(i)})\tilde{w}^{(i)}\pi(x^{(i)}) \\
&= \sum_{i=1}^{N} f(x^{(i)}) \frac{w^{(i)}}{\sum_j w^{(j)}} \pi(x^{(i)}) \\
&= \sum_{i=1}^{N} f(x^{(i)}) \frac{\frac{C\,p(x^{(i)})}{\pi(x^{(i)})}}{\sum_j \frac{C\,p(x^{(i)})}{\pi(x^{(i)})}} \pi(x^{(i)}) \\
&= \frac{1}{\sum_j \frac{p(x^{(i)})}{\pi(x^{(i)})}} \sum_{i=1}^{N} f(x^{(i)})\pi(x^{(i)}).
\end{aligned}$$

In addition to solving the problem of sampling from a distribution known up to a multiplicative constant, importance sampling gives us the possibility of choosing an importance density that allows us to "herd" particles into interesting areas of the state space. This freedom, however, can quickly become a liability if an unsuitable density is chosen.

2.4 Sequential Importance Sampling

We now return to the framework from Section 2.2 and apply the importance sampling technique. To maintain the beneficial property of recursive updates, we demand that the importance density decompose according to

$$\pi(\boldsymbol{x}_{0:t}|\boldsymbol{y}_{1:t}) = \pi(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1})\pi(\boldsymbol{x}_t|\boldsymbol{x}_{0:t-1}, \boldsymbol{y}_{1:t}).$$

Iteration of this equality yields

$$\pi(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{1:T}) = \pi(\boldsymbol{x}_0) \prod_{t=1}^{T} \pi(\boldsymbol{x}_t|\boldsymbol{x}_{0:t-1}, \boldsymbol{y}_{1:t}).$$

Recall the update formula (1) for the joint posterior density

$$\begin{aligned}
p(\boldsymbol{x}_{0:t}|\boldsymbol{y}_{1:t}) &= p(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1}) \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} \\
&\propto p(\boldsymbol{x}_{0:t-1}|\boldsymbol{y}_{1:t-1})p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}).
\end{aligned}$$

Thereby we obtain the *weight update equation*

$$w_t^{(i)} \propto \frac{p(\boldsymbol{x}_{0:t}^{(i)}|\boldsymbol{y}_{1:t})}{\pi(\boldsymbol{x}_{0:t}^{(i)}|\boldsymbol{y}_{1:t})} = \frac{p(\boldsymbol{x}_{0:t-1}^{(i)}|\boldsymbol{y}_{1:t-1})}{\pi(\boldsymbol{x}_{0:t-1}^{(i)}|\boldsymbol{y}_{1:t-1})} \cdot \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})p(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)})}{\pi(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{0:t-1}^{(i)}, \boldsymbol{y}_{1:t})} = w_{t-1}^{(i)} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})p(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)})}{\pi(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{0:t-1}^{(i)}, \boldsymbol{y}_{1:t})}. \tag{8}$$

We will additionally assume that

$$\pi(\boldsymbol{x}_t|\boldsymbol{x}_{0:t-1}, \boldsymbol{y}_{1:t}) = \pi(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{y}_t),$$

allowing our method to get by without storing a complete history of all particle paths and observations. This simplifies (8) to

$$w^{(i)} \propto w_{t-1}^{(i)} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})p(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)})}{\pi(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{y}_t)}. \tag{9}$$

Remember that it suffices to specify the weights up to a proportionality constant.

The final question separating us from an actually implementable method is now the actual choice of the importance density π. The goal in choosing this density has to be to keep all particles "meaningful", i.e. keep the weights as uniform as possible. Without proof, and without even further formalizing the aforementioned requirement, we state that the optimal choice is

$$\pi_{\text{opt}}(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{y}_t) = p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{y}_t).$$

However, sampling from this density is not usually easy. A more convenient (and also more popular choice) is the prior

$$\pi_{\text{prior}}(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{y}_t) := p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}^{(i)}), \tag{10}$$

which is readily available and often Gaussian, thus easy to sample from. The weight update equation (9) takes a particularly simple form in this case:

$$w_t^{(i)} \propto w_{t-1}^{(i)} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})p(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)})}{\pi(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{y}_t)} = w_{t-1}^{(i)} p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)}).$$

In all that follows, we will use this choice of importance density. For a more detailed treatment of the choice of importance density, we refer the reader to [2].

We now have an acutally implementable algorithm that is summarized as source code in Figure 2. Instead of showing pseudocode, I opted for showing an actual implementation in the Python[1] programming language, which I deem just as readable, with the added benefit of being an actual, executable and verifiable program.

```
def sis(model, n=1000, max_time=20):
    particles = [model.sample_initial() for i in range(n)]
    particle_history = [particles]
    weights = [1/n for i in range(n)]
    weight_history = [weights]

    x = model.sample_initial()
    x_history = [x]

    for t in range(1, max_time):
        x = model.sample_transition(t, x)
        y = model.sample_observation(t, x)
        x_history.append(x)

        evolved_particles = [model.sample_transition(t, xp) for xp in particles]
        weights = [weight*model.observation_density(t, y, xtp)
                for weight, xtp in zip(weights, evolved_particles)]
        weight_sum = sum(weights)
        weights = [weight/weight_sum for weight in weights]
        particle_history.append(particles)
        weight_history.append(weights)

    return x_history, particle_history, weight_history
```

Figure 2. A Python implementation of Sequential Importance Sampling.

1. http://www.python.org

The next issue is to evaluate the performance of the Sequential Importance Sampling Algorithm. To that end, consider the plot in Figure 3. It's easy to see that after a few timesteps, the distribution degenerates, so that all the weight is concentrated on a single spot, and in fact concentrated on a single particle (which is not visible from the plot). This is obviously not very useful, so the next section will bring about a method that addresses this problem.

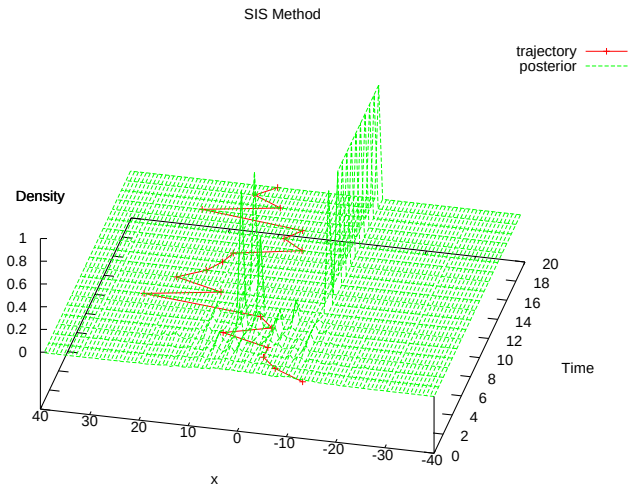

Figure 3. A typical density evolution for SIS on the process from Section 2.1 using $N = 1000$ particles. The "true" particle trajectory is shown in red, while the inferred posterior distribution is shown in green. Note how the distribution degenerates as time goes by.

2.5 Sequential Importance Resampling

The key idea behind SIR is to add an additional *resampling* step that kills off particles having low importance weights while multiplying the ones with high weights. The simplest and most popular way of doing so is to sample N times from the discrete distribution

$$\hat{p}(x) := \sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x - x^{(i)}).$$

This is often called "*multinomial branching*" because the number of particles $\boldsymbol{\xi} = (\xi^{(1)}, \xi^{(2)}, ..., \xi^{(N)})$ descending from the particles $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$ can be viewed as a realization of a multinomial distribution

$$\boldsymbol{\xi} \sim \text{Multinomial}(n, \tilde{w}^{(1)}, ..., \tilde{w}^{(N)}).$$

Adding this step, we obtain the algorithm shown in Figure 4. Note that the procedure is even simpler now because we do not need to propagate a set of weights between steps. Instead, at the end of each step, all particles are equally weighted, and the weights only exist briefly within each step to obtain this equal weighting.

```
def sir(model, n=1000, max_time=20):
    particles = [model.sample_initial() for i in range(n)]
    particle_history = [particles]

    x = model.sample_initial()
    x_history = [x]

    for t in range(1, max_time):
        x = model.sample_transition(t, x)
        y = model.sample_observation(t, x)
        x_history.append(x)

        evolved_particles = [model.sample_transition(t, xp) for xp in particles]
        iweights = [model.observation_density(t, y, xtp) for xtp in evolved_particles]
        particles = sample_discrete(evolved_particles, iweights, n)
        particle_history.append(particles)

    return x_history, particle_history, None
```

Figure 4. A Python implementation of Sequential Importance Resampling.

The added resampling step helps the SIR algorithm overcome the degeneracy problems encountered by the SIS method. As evidenced by Figure 5, it is a useful method for tracking our example process from Section 2.1. Observe that the seemingly unexplained peaks in the posterior distribution are caused by the fact that we are only observing the square of our process, thus we cannot know what sign x_t has at any given moment, causing two modes to appear on either side of zero.

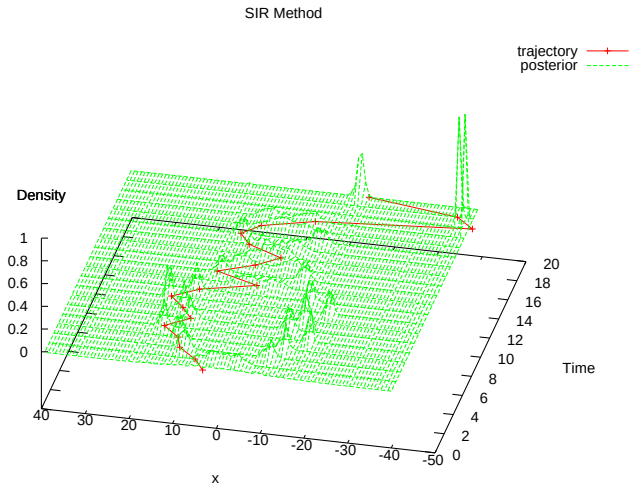

Figure 5. A typical density evolution for SIR on the process from Section 2.1 using $N = 1000$ particles. The "true" particle trajectory is shown in red, while the inferred posterior distribution is shown in green.

The resampling step introduces other problems, though.

- First, in the case of low process noise, the distribution will once again collapse to a single point within very few iterations. This issue is known as *sample impoverishment*, and counteracted by methods such as *Resample-Move* [3] or regularisation [2].
- Second, it reduces the embarassingly parallel SIS method to one that is not entirely trivial to parallelize.
- Third, with our particular choice (10) of importance density, the state space is explored without any knowledge of the observations. This can make this filter inefficient and sensitive to outliers.

2.5.1 Binary-Tree-Based Resampling

For completeness, we will briefly treat a different way of doing the resampling. The process is based on a binary tree as in Figure 6, whose leaves correspond to the particles $\boldsymbol{x}^{(i)}$ with weights $\tilde{w}^{(i)}$. Let each node receive a number m, and let m_1 and m_2 denote the two children of node m. By some technique (see [4]) it is then possible to construct a random variable ξ_m for each node such that

$$\xi_m = \xi_{m_1} + \xi_{m_2} \tag{11}$$

and further

$$\xi_m = \begin{cases} \lfloor w_m \rfloor & \text{with probability } 1 - w_m - \lfloor w_m \rfloor, \\ \lfloor w_m \rfloor + 1 & \text{with probability } \quad w_m - \lfloor w_m \rfloor, \end{cases}$$

where we define the leaf weights as $w_{m^{(i)}} := N\tilde{w}^{(i)}$ with N the desired number of particles after resampling. We then propagate the weights towards the root by defining

$$w_m := w_{m_1} + w_{m_2}.$$

Clearly, $w_r = N$. Direct computation verifies $E[\xi_m] = w_m$, and obviously ξ_m is integer. For leaf nodes, the value of ξ_m is then taken as the number of descendants of the particle corresponding to that node, whereas the remaining ξ_m are discarded–they were just scaffolding to obtain the correctly-distributed ξ_m with the sum property (11).

Intuitively, this method introduces less randomness than the multinomial resampling. For example, consider a particle that has a weight of, say, 3.72. In the tree-based approach, it is resampled into either 3 or 4 particles, while for the multinomial case, any number from 0 to N is possible.

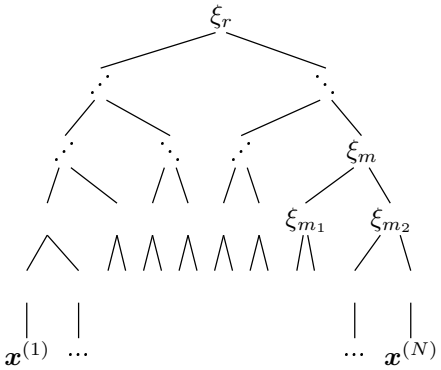

Figure 6. Binary-tree based Resampling.

2.6 A Convergence Result

Without proof, we state the following convergence result.

Theorem 1. ([4], Corollary 2.4.4) *Let the transition kernel $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ have the* Feller property*, that is for any bounded and continuous $f\colon \mathbb{R}^n \to \mathbb{R}$, the map*

$$\boldsymbol{x}_{t-1} \mapsto \int f(x) p(\mathrm{d}\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$$

must be bounded and continuous. Then as the number of particles $N \to \infty$, the approximated posterior densities of the SIR method with multinomial resampling converge

$$\hat{p}^N(\boldsymbol{x}_t|\boldsymbol{y}_{0:t}) \to p(\boldsymbol{x}_t|\boldsymbol{y}_{0:t})$$

p-almost surely.

3 Quadrature Filters

3.1 The Idea behind the Filter

The key idea in the construction of the quadrature filter is a clever probabilistic reinterpretation of Gauß

quadrature. Recall that Gauß-Legendre quadrature obtains an approximation to an integral

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{i=1}^{N} f(\xi^{(i)})\gamma^{(i)}$$

for some quadrature weights $\gamma^{(i)}$ and quadrature points $\xi^{(i)}$, the latter of which are found as the roots of Legendre polynomials.

Using the substitution rule, this formula is easily transformed to a general interval (A, B):

$$\int_{A}^{B} f(x)\mathrm{d}x \approx \sum_{i=1}^{N} f(x^{(i)})w^{(i)},$$

with

$$x^{(i)} = \frac{A+B}{2} + \xi^{(i)}\frac{B-A}{2}, \qquad w^{(i)} = \frac{B-A}{2}\gamma^{(i)}.$$

If we now consider a probability density function p supported on (A, B), then

$$E[f(x)] = \int_{A}^{B} f(x)p(x)\mathrm{d}x \approx \sum_{i=1}^{N} f(x^{(i)})p(x^{(i)})w^{(i)}.$$

In particular,

$$1 = E[1] = \int_{A}^{B} p(x)\mathrm{d}x \approx \sum_{i=1}^{N} p(x^{(i)})w^{(i)}.$$

This gives rise to the interpretation of

$$\hat{p}(\mathrm{d}x) = \sum_{i=1}^{N} \delta(\mathrm{d}x - x^{(i)})p(x^{(i)})w^{(i)}$$

as a discrete approximation of $\hat{p}(x)$. In analogy to what we did above, we may regard the quadrature points $x^{(i)}$ as our particles and

$$\tilde{p}(x^{(i)}) := p(x^{(i)})w^{(i)}$$

as our weights.

3.2 Construction of the Filter

Now recall from Section 1 the recursion for the posterior, consisting of the prediction formula (2)

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_t$$

and the update formula (3)

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) = \frac{1}{C_t} \cdot p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$$

with

$$C_t = \int p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_t.$$

Recall that in the quadrature approach, the particle locations $\boldsymbol{x}_t^{(i)}$ are fixed, so that the update equation takes the role of simply updating the particle posterior probabilities (a.k.a. weights). Given the approximate prediction $\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t-1})$ for each particle, we set

$$\tilde{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) = \frac{1}{\hat{C}_t} \cdot p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})w_t^{(i)} \underbrace{\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t-1})}_{\text{prediction}}, \tag{12}$$

which in fact is most easily computed by first finding

$$q^{(i)} = p(\boldsymbol{y}_t|\boldsymbol{x}_t)w_t^{(i)}\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t-1}) \quad (i = 1, ..., N)$$

and then computing

$$\hat{C}_t = \sum_{i=1}^{N} q^{(i)}$$

and applying the normalization

$$\tilde{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) = \frac{q^{(i)}}{\hat{C}_t}.$$

Note our use of the tilde in $\tilde{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t})$ to denote that this term is not just a particle approximation (denoted $\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t})$), but in addition carries the weighting introduced in Section 3.1.

Prediction, in principle the first step of the algorithm, now makes use of Gaussian quadrature using the weights we built into (12):

$$\hat{p}(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t-1}) = \sum_{j=1}^{N} p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{x}_t^{(j)})\tilde{p}(\boldsymbol{x}_t^{(j)}|\boldsymbol{y}_{1:t}) \quad (i = 1, ..., N). \tag{13}$$

Notice that, computationally, this is the most expensive step of the algorithm, of N^2 complexity. Consequently, this is the very point where the algorithm gets untractable for dimensions greater than roughly three–realize that Gaussian quadrature requires $N = m^d$ quadrature points, where m is the order of the quadrature and d is the dimension of the state space.

In a high-dimensional setting, knowledge about some strong correlation between state variables in principle opens up the possibility of employing a skewed grid that does a better job of covering the support of the probability density, allowing the use of fewer points. This however requires high-order interpolation between these grids, introducing additional error and tedious computations. The authors of [5] indicate that, in their experiments, this cost outweighed any potential speed gain.

3.3 Implementation and Evaluation

Figure 7 shows a reasonably straightforward Python implementation of quadrature filtering.

```
def quadrature(model, n=100, max_time=20, interval=(-40,40)):
    particles, gauss_weights = quad_data(n, interval)

    prob = [model.initial_density(x) for x in particles]
    assert abs(sum(p*w for p, w in zip(prob, gauss_weights))-1) < 1e-3
    prob_history = [prob]

    x = model.sample_initial()
    x_history = [x]

    for t in range(1, max_time):
        x = model.sample_transition(t, x)
        y = model.sample_observation(t, x)
        x_history.append(x)

        prediction_prob = [
                sum(model.transition_density(t, xj, xi) * prob[j]
                    for j, xj in enumerate(particles))
                for xi in particles]

        prob = [model.observation_density(t, y, xi) *
                gauss_weights[i] *
                prediction_prob[i]
                for i, xi in enumerate(particles)]
        normalizer = sum(prob)
        prob = [p/normalizer for p in prob]
        prob_history.append(prob)

    return x_history, particles, prob_history
```

Figure 7. A Python implementation of Quadrature Filtering.

In contrast to the methods discussed in Section 2, observe that the quadrature scheme is entirely deterministic. There is no sampling involved–given the same observations, the quadrature filter will give the same posterior distribution every time. This makes the scheme attractive for applications where such predictability is desirable, for example when comparing the likelihood

$$p(\boldsymbol{y}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \prod_{t=1}^{T} C_t$$

of a given observation with respect to different models, in particular similar models with different parameters.

Figure 8 shows a sample run of the algorithm on the process from Section 2.1 with $N = 100$. First, observe the clustering of particles near the domain endpoints typical of Gaussian quadrature. Next, note that despite the fact an order of magnitude fewer particles than we used in the Monte Carlo methods, the quadrature method is the only scheme to produce the symmetric posterior that is to be expected for a process of which we only observe the square. In general, the posteriors produced by quadrature are also less erratic than the Monte Carlo ones.

Another favorable aspect of quadrature methods is the fact that explicit estimates of the error are available, as we will show next.

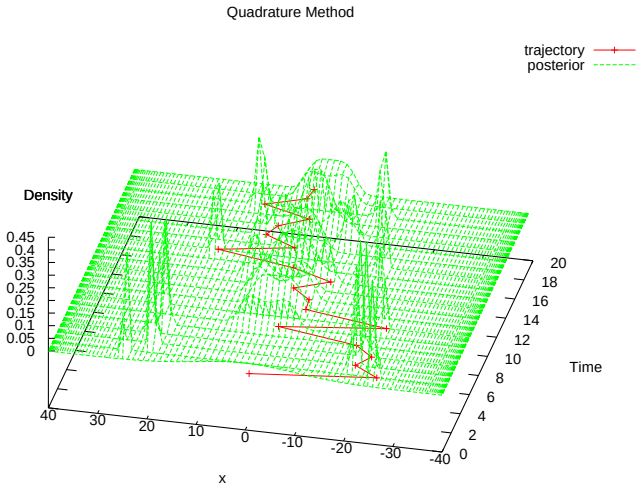

Figure 8. A typical density evolution for the quadrature filter on the process from Section 2.1 using $N = 100$ particles. The "true" particle trajectory is shown in red, while the inferred posterior distribution is shown in green. Observe the difference in particle spacing between the domain boundary and its center.

3.4 Error Estimates

To begin, define the error in the log-likelihood of the observation

$$\delta_t := \log\{\hat{p}(\boldsymbol{y}_{1:t})\} - \log\{p(\boldsymbol{y}_{1:t})\} = \sum_{s=1}^{t} \log\hat{C}_s - \sum_{s=1}^{t} \log C_s.$$

We will derive an error propagation result for

$$\varepsilon(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) := \exp(\delta_t)\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}).$$

which we take as indicator for the error in the prediction $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})$. It is only an indicator and not the true error because we ignored the error in the normalization constant, necessitating the term

$$\exp(\delta_t) = \frac{\hat{p}(\boldsymbol{y}_{1:t})}{p(\boldsymbol{y}_{1:t})} = \prod_{s=1}^{t} \frac{\hat{C}_s}{C_s}.$$

We will also simply state an error propagation result for δ_t. In notation and argument, we follow [5]. First up is a recursion formula for the prediction error indicator that is valid for *any* particle filter:

$$
\begin{aligned}
& \varepsilon(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
= \;& \exp(\delta_t)\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
\overset{(2),(13)}{=} \;& \exp(\delta_t)\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_t \\
= \;& \exp(\delta_t)\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) \\
& \underbrace{-\int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)\exp(\delta_t)\hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_t + \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)\exp(\delta_t)\hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_t}_{0} \\
& - \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_t \\
= \;& \underbrace{\exp(\delta_t)\Big[\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)\hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\mathrm{d}\boldsymbol{x}_t\Big]}_{\eta(\boldsymbol{x}_{t+1}):=} \\
& + \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)\Big(\exp(\delta_t)\hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\Big)\mathrm{d}\boldsymbol{x}_t \\
= \;& \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)\Big(\exp(\delta_t)\hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})\Big)\mathrm{d}\boldsymbol{x}_t + \eta(\boldsymbol{x}_{t+1}) \\
\overset{(3)}{=} \;& \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{y}_t|\boldsymbol{x}_t)\Big(\exp(\delta_t)\frac{1}{\hat{C}_t}\cdot\hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) - \frac{1}{C_t}\cdot p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\Big)\mathrm{d}\boldsymbol{x}_t + \eta(\boldsymbol{x}_{t+1}) \\
\overset{(*)}{=} \;& \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{y}_t|\boldsymbol{x}_t)\Big(\exp(\delta_t)\frac{1}{\hat{C}_t}\cdot\frac{\varepsilon(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) + p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\exp(\delta_{t-1})} - \frac{1}{C_t}\cdot p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\Big)\mathrm{d}\boldsymbol{x}_t + \eta(\boldsymbol{x}_{t+1}) \\
= \;& \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{y}_t|\boldsymbol{x}_t)\Big(\frac{1}{\hat{C}_t}\cdot\frac{\hat{C}_t}{C_t}[\varepsilon(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) + p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})] - \frac{1}{C_t}\cdot p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\Big)\mathrm{d}\boldsymbol{x}_t + \eta(\boldsymbol{x}_{t+1}) \\
= \;& \frac{1}{C_t}\int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{y}_t|\boldsymbol{x}_t)\underline{\varepsilon(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}\mathrm{d}\boldsymbol{x}_t + \eta(\boldsymbol{x}_{t+1}),
\end{aligned}
$$

where we've used

$$
(*) \quad \frac{\varepsilon(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) + p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\exp(\delta_{t-1})} = \hat{p}(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}).
$$

Note that the term $\eta(\boldsymbol{x}_{t+1})$ defined above is exactly the error contribution of the time step $t+1$.

A similar error recursion can be derived specifically for quadrature methods. Consider

$$
\begin{aligned}
& \varepsilon(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
= \;& \exp(\delta_t)\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
= \;& \exp(\delta_t)\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
= \;& \sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})\Big[\exp(\delta_t)\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - w_t^{(i)}p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t})\Big] \\
& + \sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})w_t^{(i)}p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
= \;& \sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})\Big[\hat{C}_t^{-1}\exp(\delta_t)p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})w_t^{(i)}\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t-1}) - w_t^{(i)}C_t^{-1}p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t-1})\Big] \\
& + \sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})w_t^{(i)}p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
= \;& C_t^{-1}\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})w_t^{(i)}\Big[\exp(\delta_{t-1})\hat{p}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t-1}) - p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\Big] \\
& + \sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})w_t^{(i)}p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})
\end{aligned}
$$

$$= C_t^{-1}\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})w_t^{(i)}\underline{\varepsilon(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})} + \sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})w_t^{(i)}p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})$$

Next, assume that we have enough particles so that

$$|\sum_j w^{(j)}p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{x}_t^{(j)})p(\boldsymbol{x}_t^{(j)}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t})| \leqslant \varepsilon p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t}). \tag{14}$$

Note that this is an approximation assumption based on the exact probabilities, not the approximations. Our goal is to prove a recursive estimate of the form

$$r_{t+1} \leqslant r_t(1+\varepsilon)+\varepsilon, \tag{15}$$

where r_t is a measure of the relative error in $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})$, namely, the smallest nonnegative real such that

$$|\varepsilon(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})| \leqslant r_t p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}).$$

We will try to prove (15) by induction. (14) is the first step in the recursion, namely $r_1 \leqslant \varepsilon$. For the step from t to $t+1$, consider

$$\begin{aligned}
& \varepsilon(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t}) \\
\leqslant\ & |C_t^{-1}\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})w_t^{(i)}\varepsilon(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})| \\
& + |\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})w_t^{(i)}p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t}) - p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{1:t})| \\
\overset{(14)}{\leqslant}\ & |C_t^{-1}\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})w_t^{(i)}\varepsilon(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})| + \varepsilon p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t}) \\
\overset{(15)\text{ for }t}{\leqslant}\ & |C_t^{-1}\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})w_t^{(i)}r_{t-1}p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})| + \varepsilon p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t}) \\
=\ & |r_{t-1}\sum_i p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^{(i)})w_t^{(i)}p(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_{1:t})| + \varepsilon p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t}) \\
\overset{(14)}{\leqslant}\ & r_{t-1}|p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t})(1+\varepsilon)| + \varepsilon p(\boldsymbol{x}_{t+1}^{(i)}|\boldsymbol{y}_{1:t}).
\end{aligned}$$

Therefore,

$$r_t \leqslant r_{t-1}(1+\varepsilon)+\varepsilon,$$

as claimed. Iterating this estimate for r_2 yields

$$\begin{aligned}
r_2 &= r_1(1+\varepsilon)+\varepsilon \\
&= \varepsilon(1+\varepsilon)+\varepsilon \\
&= \varepsilon^2+2\varepsilon \\
&= (1+\varepsilon)^2-1.
\end{aligned}$$

We hypothesize that

$$r_t \leqslant (1+\varepsilon)^t - 1, \tag{16}$$

which we can again easily prove by induction: Suppose it holds for t, then

$$\begin{aligned}
r_{t+1} &\leqslant r_t(1+\varepsilon)+\varepsilon \\
&\leqslant \big[(1+\varepsilon)^t-1\big](1+\varepsilon)+\varepsilon \\
&= (1+\varepsilon)^{t+1}-(1+\varepsilon)+\varepsilon \\
&= (1+\varepsilon)^{t+1}-1.
\end{aligned}$$

The estimate (16) gives a good idea about how the error in the prediction estimate (13) grows. By a similar argument, the likelihood error is estimated as

$$\delta_t \leqslant (t+1)\log(1+\varepsilon'),$$

yielding only linear growth of the error in this case (but keep in mind that δ_t is a logarithmic quantity).

3.5 Quadrature vs. Monte Carlo

Altogether, quadrature and particle filters form a mostly complementary set of algorithms–one shines where the other one has weaknesses. Quadrature filters are rather time-consuming and suffer badly from the curse of dimensionality, but deliver exact and deterministic answers. Monte Carlo filters, on the other hand, are rather quick and deal with high-dimensional state spaces more easily. Their results are random and sometimes inaccurate, and they may require a lot of particles to give an adequate answer. Figure 9 shows a summary of the relative merits of each method.

Feature	SMC Filters	Quadrature Filters
Runtime cost/step	$O(N)$ (SIS) $O(N \log N)$ (SIR)	$O(N^2)$
Randomness	Randomized algorithm	Deterministic
Suitable for parameter finding	$-$	$+$
Curse of dimensionality	$+$	$--$
Error estimates	$-$	$+$
Particles required for given accuracy	many	relatively few

Figure 9. Comparison of performance criteria for quadrature vs. sequential Monte Carlo filters.

4 A Glance at Continuous-Time Filtering

The setup for the nonlinear filtering problem in continuous time consists of

- a *Markov diffusion process*
$$\mathrm{d}x^i(t) = b^i(\boldsymbol{x}(t))\mathrm{d}t + \sigma^{ij}(\boldsymbol{x}(t))\mathrm{d}W^j(t)$$
where $\boldsymbol{x} = (x^1, ..., x^d)$ and $\boldsymbol{x}(0) = \boldsymbol{x}_0$.
- An *observation*
$$\boldsymbol{y}(t) = \int_0^t \boldsymbol{h}(\boldsymbol{x}(s))\mathrm{d}s + \boldsymbol{V}(t).$$

All coefficients are assumed smooth, and the initial vector and the two Wiener processes $\boldsymbol{W}$ and $\boldsymbol{V}$ are assumed to be independent. (Setup and notation here originate from [7].) If we are interested in observing a function $f(\boldsymbol{x}(t))$, then the optimal filter (the best mean-square estimate) for this random variable is given by

$$\hat{f}(\boldsymbol{x}(t)) = \frac{\int_{\mathbb{R}^d} f(\boldsymbol{x})u(t,\boldsymbol{x})\mathrm{d}\boldsymbol{x}}{\int_{\mathbb{R}^d} u(t,\boldsymbol{x})\mathrm{d}\boldsymbol{x}},$$

where u is the *unnormalized filtering density*, somewhat in analogy to the update formula (3) above without the normalization constant C_t^{-1} at each time step. The function u can be found as a solution to the *Zakai SPDE*

$$\mathrm{d}u(t,\boldsymbol{x}) = \mathcal{L}^* u(t,\boldsymbol{x})\mathrm{d}t + h(\boldsymbol{x})u(t,\boldsymbol{x})\mathrm{d}\boldsymbol{y}(t), \tag{17}$$

where

$$\mathcal{L}^* u := \frac{1}{2}\frac{\partial^2}{\partial x_i x_j}((\sigma\,\sigma^*)^{ij}u) - \frac{\partial}{\partial x_i}(b^i u).$$

(An accessible derivation of the Zakai equation may be found in Chapter 13 of [6].) One very interesting approach of solving (17) is Sergey Lototsky's *spectral separating scheme* that expresses the unnormalized filtering density as an expansion into *Wick polynomials* ξ_α:

$$u(t,\boldsymbol{x}) = \sum_\alpha \frac{1}{\sqrt{\alpha!}}\varphi_\alpha(t,\boldsymbol{x})\xi_\alpha(\boldsymbol{y}).$$

The coefficients φ_α then satisfy a deterministic system of PDEs. The secheme is called *separating* because it separates the dependency on process parameters (φ_α) and observation (ξ_α), allowing the former to be obtained beforehand, making the scheme computationally attractive. We refer to [7] for the details.

Bibliography

[1] Arnaud Doucet, Nando de Freitas, and Neil Gordon. An Introduction to Sequential Monte Carlo Methods. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 1. Springer, 2001.

[2] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

[3] Carlo Berzuini and Walter Gilks. RESAMPLE-MOVE Filtering with Cross-Model Jumps. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 6. Springer, 2001.

[4] Dan Crisan. Particle Filters - A Theoretical Perspective. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 2. Springer, 2001.

[5] Erik Bolviken and Geir Storvik. Deterministic and Stochastic Particle Filters in State-Space Models. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 5. Springer, 2001.

[6] Nasir Uddin Ahmed. *Linear and Nonlinear Filtering for Scientists and Engineers*. World Scientific, 1998.

[7] Sergey Lototsky, Remigijus Mikulevicius, and Boris L. Rozovskii. Nonlinear Filtering Revisited: A Spectral Approach. *Siam J. Control Optim.*, 35(2):435–461, March 1997.