Learning nonlinear dynamical models using particle filters
Part 2 – Sequential Monte Carlo, the particle filter

Thomas Schön
Division of Systems and Control
Department of Information Technology
Uppsala University.
Email: thomas.schon@it.uu.se,
www: user.it.uu.se/~thosc112

Outline – Part 2

Aim: Derive the bootstrap particle filter and provide some of its properties.

1. Summary of Part 1
2. The Monte Carlo idea
3. Importance sampling
4. Derive the bootstrap particle filter
5. Particle smoothers (if there is time)
6. Sequential Monte Carlo
   a) General problem formulation
   b) The underlying pdf
   c) Convergence results

Summary of Part 1 (I/II)

The two identification strategies we are concerned with are:

- **Marginalization** Deal with the states by marginalizing (integrating) them out.
- **Data augmentation** Deal with the states by treating them as auxiliary variables to be estimated along with the parameters.

<table>
<thead>
<tr>
<th>ML</th>
<th>Bayesian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct optimization</td>
<td>Metropolis Hastings</td>
</tr>
<tr>
<td>Expectation Maximization</td>
<td>Gibbs sampling</td>
</tr>
</tbody>
</table>

Summary of Part 1 (II/II)

The measurement update

\[
p(x_t | y_{1:t}) = \frac{g(y_t | x_t) p(x_t | y_{1:t-1})}{p(y_t | y_{1:t-1})},
\]

and time update

\[
p(x_t | y_{1:t-1}) = \int f(x_{t+1} | x_t) p(x_t | y_{1:t-1}) \, dx_{t-1}.
\]

The joint smoothing pdf \( p(x_{1:T} | y_{1:T}) \) is given by

\[
p(x_{1:T} | y_{1:T}) = p(x_T | y_{1:T}) \prod_{t=1}^{T-1} \frac{f(x_{t+1} | x_t) p(x_t | y_{1:t})}{p(x_{t+1} | y_{1:t})}.
\]
Why do we need Monte Carlo methods?

In solving inference problems we are typically faced with various integration problems, which tend to be intractable and live in high dimensional spaces.

For example expectation arising in obtaining a point estimate. A commonly used point estimate is the conditional mean

\[ \hat{x}_{\mid t} = E \left[ x_{\mid t} \mid y_{1:t} \right] = \int x_{\mid t} p(x_{\mid t} \mid y_{1:t}) \, dx_{\mid t}. \]

Monte Carlo methods provide computational solutions where the distributions of interest are approximated by a large number of \( N \) random samples called particles.

Monte Carlo methods respects the model and the expressions we are trying to approximate.

The Monte Carlo idea (I/II)

(Very) restrictive assumption: Assume that we have \( N \) samples \( \{x^i\}_{i=1}^N \) from the target density \( \pi(x) \),

\[ \hat{\pi}(x) = \frac{1}{N} \sum_{i=1}^{N} \delta_{x^i}(x) \]

Allows for the following approximation of the integral,

\[ E[\varphi(x)] = \int \varphi(x) \pi(x) \, dx \approx \int \varphi(x) \sum_{i=1}^{N} \frac{1}{N} \delta_{x^i}(x) \, dx = \frac{1}{N} \sum_{i=1}^{N} \varphi(x^i) \]

\[ \int + \delta \rightarrow \sum \]

The Monte Carlo idea (II/II)

The integral

\[ I(\varphi(x)) \triangleq E[\varphi(x)] = \int \varphi(x) \pi(x) \, dx. \]

is approximated by

\[ \hat{I}_N(\varphi(x)) = \frac{1}{N} \sum_{i=1}^{N} \varphi(x^i). \]

The strong law of large numbers tells us that

\[ \hat{I}_N(\varphi(x)) \xrightarrow{\text{a.s.}} I(\varphi(x)), \quad N \to \infty, \]

and the central limit theorem state that

\[ \frac{\sqrt{N} \left( \hat{I}_N(\varphi(x)) - I(\varphi(x)) \right)}{\sigma_{\varphi}} \xrightarrow{d} \mathcal{N}(0, 1), \quad N \to \infty. \]

The Monte Carlo idea – toy illustration

\[ \pi(x) = 0.3N(x \mid 2, 2) + 0.7N(x \mid 9, 19) \]

\[ \text{5000 samples} \quad \text{50000 samples} \]

Obvious problem: In general we are not able to directly sample from the density we are interested in.
**Importance sampling – problem and idea**

Importance sampling can be used to evaluate integrals of the form

\[ I(\varphi(x)) = E[\varphi(x)] = \int \varphi(x) \pi(x) dx, \]

where it is hard to generate samples from the target density \( \pi(x) = \frac{\gamma(x)}{Z} \).

Note that:

\[ \int \varphi(x) \pi(x) dx = \int \varphi(x) \frac{\pi(x)}{q(x)} q(x) dx. \]

**Idea:** Chose the proposal density \( q(x) \) such that it is easy to generate samples from it and somehow compensate for the mismatch between the target and the proposal.

---

**Importance sampling (IS)**

The target density can be written

\[ \pi(x) = \frac{\gamma(x)}{Z} = \frac{\gamma(x) q(x)}{q(x) Z} = \frac{w(x) q(x)}{Z}, \]

where \( w(x) \) is the unnormalized weight function

\[ w(x) = \frac{\gamma(x)}{q(x)}, \]

and

\[ Z = \int w(x) q(x) dx. \]

Upon normalization:

\[ \pi(x) = \frac{w(x) q(x)}{Z} \text{ and } Z = \int w(x) q(x) dx. \]

**Algorithm 1** Importance sampler (IS)

1. Draw \( N \) samples \( x^i \sim q(x), \ i = 1, \ldots, N \).
2. Compute the weights \( w_i = \gamma(x^i) / q(x^i) \).
3. Normalize the weights \( w^i = w_i / \sum_{j=1}^{N} w_j \).

Each step is carried out for \( i = 1, \ldots, N \).

The convergence of the resulting approximation \( \tilde{\pi}(x) = \sum_{i=1}^{N} w^i \delta_{x^i}(x) \) is since long well established.

Sampling from a user-chosen proposal distribution \( q \) is corrected for by the weights, which accounts for the discrepancy between the proposal \( q \) and the target \( \pi \).
The importance of a good proposal density

\[ q_1(x) = \mathcal{N}(5, 20) \text{ (dashed curve)} \]
\[ q_2(x) = \mathcal{N}(1, 20) \text{ (dashed curve)} \]

50,000 samples used in both simulations.

**Lesson learned:** It is important to be careful in selecting the proposal density.

Outline – Part 2

**Aim:** Derive the bootstrap particle filter and provide some of its properties.

1. Summary of Part 1
2. The Monte Carlo idea
3. Importance sampling
4. Derive the bootstrap particle filter
5. Sequential Monte Carlo
   a) General problem formulation
   b) The underlying pdf
   c) Convergence results

Using IS for our purposes

Recall that the nonlinear filtering problem amounts to computing the filter PDF

\[ p(x_t | y_{1:t}) \]

when the model is given by

\[ x_{t+1} | x_t \sim f(x_{t+1} | x_t), \]
\[ y_t | x_t \sim g(y_t | x_t), \]
\[ x_1 \sim \mu(x_1). \]

We have showed that the solution is

\[ p(x_t | y_{1:t}) = \frac{g(y_t | x_t)p(x_t | y_{1:t-1})}{p(y_t | y_{1:t-1})}, \]
\[ p(x_t | y_{1:t-1}) = \int f(x_t | x_{t-1})p(x_{t-1} | y_{1:t-1})dx_{t-1}. \]

**Relevant idea:** Try to solve this using importance sampling!

Finding a proposal (I/II)

Assume (in an “induction-like” fashion) that we at time \( t-1 \) have

\[ \hat{p}(x_{t-1} | y_{1:t-1}) = \sum_{i=1}^{N} w_{t-1}^i \delta_{x_{t-1}^i} (x_{t-1}), \]

allowing us to approximate the integral for \( p(x_t | y_{1:t-1}) \),

\[ \hat{p}(x_t | y_{1:t-1}) = \int f(x_t | x_{t-1}) \sum_{i=1}^{N} w_{t-1}^i \delta_{x_{t-1}^i} (x_{t-1})dx_{t-1} \]
\[ = \sum_{i=1}^{N} w_{t-1}^i f(x_t | x_{t-1}). \]

Hence, \( p(x_t | y_{1:t-1}) \) is approximated by a mixture distribution, where there is one mixture component for each of the \( N \) particles.
Finding a proposal (I/II)

Inserting
\[ \tilde{p}(x_t \mid y_{1:t-1}) = \sum_{i=1}^{N} w_{t-1}^i f(x_t \mid x_{t-1}^i) \]
into
\[ p(x_t \mid y_{1:t}) = \frac{g(y_t \mid x_t)p(x_t \mid y_{1:t-1})}{p(y_t \mid y_{1:t-1})}, \]
results in
\[ p(x_t \mid y_{1:t}) \approx \frac{g(y_t \mid x_t)}{p(y_t \mid y_{1:t-1})} \sum_{i=1}^{N} w_{t-1}^i f(x_t \mid x_{t-1}^i). \] (1)

The task is now to approximate (1) using importance sampling.

Importance sampling reminder

Algorithm 2 Importance sampler
1. Sample \( x^t \sim r(x) \).
2. Compute the weights \( \nu(x^t) = \gamma(x^t)/r(x^t) \).
3. Normalize the weights \( \nu^t = \nu(x^t)/\sum_j^N \nu(x^j) \).

Choose a proposal density of the same form as \( p(x_t \mid y_{1:t}) \),
\[ r(x_t \mid y_{1:t}) = \sum_{i=1}^{N} \nu_{t-1}^i r(x_t \mid x_{t-1}^i, y_t), \]
where both the mixture components \( r(x_t \mid x_{t-1}^i, y_t) \) and the mixture weights \( \nu_{t-1}^i \) are design choices.

Sampling from the proposal

Two step procedure to sample from mixture proposal \( r(x_t \mid y_{1:t}) \).

Notation: The ancestor index \( a_t^i \) denotes the index of the mixture component selected for the \( i^{th} \) component at time \( t \).

1. Select one of the components (resampling),
\[ \mathbb{P}(a_t^i = j) = \nu_{t-1}^j, \quad j = 1, \ldots, N. \]
2. Generate a sample from that component,
\[ x_t^i \sim r(x_t \mid x_{t-1}^i, y_t). \]
Repeat this \( N \) times, for \( i = 1, \ldots, N. \)

Resampling (I/III)

The particle \( \bar{x}_{t-1} = x_{t-1}^{a_{t-1}^i} \) is referred to as the ancestor particle of \( x_t^i \), since \( x_t^i \) is generated conditionally on \( \bar{x}_{t-1} \).

This also explains why \( a_t^i \) is referred to as the ancestor index, since it indexes the ancestor of particle \( x_t^i \) at time \( t - 1 \).

Resampling is the procedure that (randomly) turns a weighted set of samples \( \{x_{t-1}^i, w_{t-1}^i\}_{i=1}^N \) into an unweighted set of samples \( \{\bar{x}_{t-1}^i, 1/N\}_{i=1}^N \) according to
\[ \mathbb{P}(a_t^i = j) = \nu_{t-1}^j, \quad \bar{x}_{t-1}^i = x_{t-1}^{a_{t-1}^i}. \]
Resampling (II/III)

Illustrating how resampling works (using 7 particles).
1. Compute the cumulative sum of the weights.
2. Generate \( u \sim \mathcal{U}[0, 1] \).

Three new samples are generated in the figure above, corresponding to sample 2, 3, and 4.

Ancestral indices and ancestral paths

At time \( t = 1 \), particle \( x_1^1 \) is resampled twice and particle \( x_3^1 \) is resampled once. Hence, at time \( t = 2 \), the \textit{ancestor indices} are

\[
a_1^2 = 2, \quad a_2^2 = 2 \quad \text{and} \quad a_3^2 = 3.
\]

Similarly, at time \( t = 3 \), the \textit{ancestor indices} are given by

\[
a_1^3 = 2, \quad a_2^3 = 3 \quad \text{and} \quad a_3^3 = 3.
\]

The \textit{ancestral path} of particle \( x_3^1 \), which we denote \( x_{1:3}^1 \), is shown as a thick line in the figure. This path is given recursively from the ancestor indices,

\[
x_{1:3}^1 = (x_2^1, x_3^1, x_3^1) = (x_1^2, x_2^2, x_3^1).
\]

Next step – computing the weights

\textbf{Algorithm 3 Importance sampler}

1. Sample \( x_i \sim r(x) \).
2. Compute the weights \( w(x_i) = \gamma(x_i)/r(x_i) \).
3. Normalize the weights \( w_i = w(x_i)/\sum_{j=1}^{N} w(x_j) \).

Compute the weights

\[
w_i(x_i) = \frac{g(y_i | x_i) \hat{p}_i(x_i | y_{1:t-1})}{r(x_i | y_{1:t})} = \frac{g(y_i | x_i) \sum_{j=1}^{N} w_{i-1}^j f(x_i | x_{i-1}^j)}{\sum_{j=1}^{N} r_i \sum_{j=1}^{N} r(x_i | x_{i-1}^j, y_t)}
\]

Computational complexity: \( \mathcal{O}(N^2) \)!
Pragmatic solution to quadratic complexity

A pragmatic solution to the quadratic computational complexity is to use the freedom available in the proposal density,

\[ r(x_t | y_{1:t}) = \sum_{j=1}^{N} w^j_{t-1} f(x_t | x^j_{t-1}). \]

Resulting weight computation

\[ w_t(x^i_t) = g(y_t | x^i_t). \]

Result – A first particle filter

Algorithm 4 Bootstrap particle filter (for \( i = 1, \ldots, N \))

1. Initialization \((t = 1)\):
   (a) Sample \( x^i_1 \sim \mu(x_1) \).
   (b) Compute the weights \( \bar{w}^i_1 = g(y_1 | x^i_1) \) and normalize,
       \( w^i_1 = \bar{w}^i_1 / \sum_{j=1}^{N} \bar{w}^j_1 \).

2. for \( t = 2 \) to \( T \) do
   (a) **Resample** \( \{x^i_{t-1}, w^i_{t-1}\} \) resulting in equally weighted particles \( \{\bar{x}^i_{t-1}, 1/N\} \).
   (b) **Propagate** by sampling \( x^i_t \sim f(x_t | \bar{x}^i_{t-1}) \).
   (c) **Weight** by computing \( \bar{w}^i_t = g(y_t | x^i_t) \) and normalize
       \( w^i_t = \bar{w}^i_t / \sum_{j=1}^{N} \bar{w}^j_t \).

SMC structure

The structure is the same for all SMC algorithms. For the bootstrap PF we have,

**Resampling**: \( \{x^i_{t-1}, w^i_{t-1}\}_{i=1}^{N} \rightarrow \{\bar{x}^i_{t-1}, 1/N\}_{i=1}^{N} \).

**Propagation**: \( x^i_t \sim f(x_t | \bar{x}^i_{t-1}) \).

**Weighting**: \( \bar{w}^i_t = W_t(x^i_t) = g(y_t | x^i_t) \) and normalize.

The result is a new weighted set of particles \( \{x^i_t, w^i_t\}_{i=1}^{N} \).

Important “design” considerations

1. Adaptive resampling – only resample “when needed”.
2. Be careful when selecting the importance density.
3. Exploit analytically tractable sub-structures (Rao-Blackwellization).
4. \ldots
Application – indoor localization (I/III)

**Aim:** Compute the position of a person moving around indoors using sensors (inertial, magnetometer and radio) located in an ID badge and a map.

The sensors (IMU and radio) and the DSP are mounted inside an ID badge.

The inside of the ID badge.

Application – indoor localization (II/III)

Targeting the JSD instead

Our derivation of the PF is rather **non-standard**. The reason I like it is that it clearly shows why the resampling step is needed and where the need for the resampling step comes from.

The more **standard** way of deriving the PF is by targeting the sequence of joint smoothing densities (JSD) \( \{p(x_{1:t} | y_{1:t})\}_{t \geq 1} \).

Enlightening derivation as well! Shows that

\[
SMC = SIS + \text{Resampling}
\]

Show movie

Johan Kihlberg, Simon Tegelid, Manon Kok and Thomas B. Schön. **Map aided indoor positioning using particle filters.** Reglermåls (Swedish Control Conference), Linköping, Sweden, June 2014.
Problems?

Can you see any problems with the algorithm producing approximations of the JSD according to

\[ p(x_{1:t} \mid y_{1:t}) = \sum_{i=1}^{N} w_i^t \delta_{x_i^t}(x_{1:t}) \]

The resampling step removes particles with small weights and duplicate particles with large weights.

This results in path degeneracy, which we explain using a simple example.

Illustration of path degeneracy (I/II)

Left plot: At each point in time all particles are plotted using a black dot and each particle is connected with its ancestor using a black line.

Right plot: The grey dots represent the \( p(x_t \mid y_{1:t}) \) at each point in time. The black lines show the particle trajectories \( \{x_i^{1:25}\}_{i=1}^{30} \) at time \( t = 25 \).

The right plot corresponds to the left plot with all trajectories that are not resampled removed (all particles are still visualized using gray dots).

Illustration of path degeneracy (II/II)

Mitigating the path degeneracy problem

This implies that if we are interested in the smoothing distribution

\[ p(x_{1:T} \mid y_{1:T}) \]

or some of its marginals we are forced to use different algorithms, which leads us to particle smoothers. Backward simulation is key here (and elsewhere).
Outline – Part 2

**Aim:** Derive the bootstrap particle filter and provide some of its properties.

1. Summary of Part 1
2. The Monte Carlo idea
3. Importance sampling
4. Derive the bootstrap particle filter
5. Particle smoothers (if there is time)
6. Sequential Monte Carlo
   - a) General problem formulation
   - b) The underlying pdf
   - c) Convergence results

---

Recall – backwards simulation

A first particle smoother

**Algorithm 5** Forward filtering/Backward smoothing (FFBSm)

1. Run the PF and store the particles and their weights \( \{ (w_i^t, x_i^t) \} \) for \( t = 1, \ldots, T \).
2. Initialise the smoothed weights \( w_T^T = w_T \), for \( i = 1, \ldots, N \).
3. for \( t = T - 1 \) to 1 do
   - a) Compute the smoothed weights
   \[
   w_i^{t+1} = w_i^t \sum_{k=1}^N w_{i+1}^t \frac{f(x_{i+1}^t | x_i^t)}{v_i^t}, \quad v_i^t = \sum_{k=1}^N w_i^t f(x_{i+1}^t | x_i^t),
   \]

The resulting approximation \( \hat{p}(x_t | y_{1:T}) = \sum_{i=1}^N w_i^t \delta_{x_i^t}(x_t) \) does not suffer from degeneracy.

---

Recall – backward computations

By marginalizing
\[
p(x_1:T | y_1:T) = p(x_T | y_1:T) \prod_{t=1}^{T-1} \frac{f(x_{t+1} | x_t)p(x_t | y_{1:t})}{p(x_{t+1} | y_{1:t})}.
\]

w.r.t. \( x_{1:t-1} \) and \( x_{t+1:T} \) we obtain the following expression for the marginal smoothing pdf
\[
p(x_t | y_{1:T}) = \frac{\int f(x_{t+1} | x_t)p(x_{t+1} | y_{1:T}) \, dx_{t+1}}{p(x_T | y_{1:T})}.
\]

**Strategy:** Run a forward filter and generate samples backwards in time:

- Sample \( \tilde{x}_T \sim p(x_T | y_{1:T}) \)
- Sample \( \tilde{x}_t \sim p(x_t | \tilde{x}_{t+1}, y_{1:t}) \)
- Sample \( \hat{x}_1 \sim p(x_1 | \tilde{x}_2, y_1) \)

The resulting sample is then by construction \( \tilde{x}_{1:T} \sim p(x_1:T | y_{1:T}) \).
Sampling from the backward kernel

**Key idea:** Generate a backward trajectory by sampling from the following approximation

\[
\hat{p}(x_t | \bar{x}_{t+1}, y_{1:t}) = \frac{\sum_i w_i^t f(\bar{x}_{t+1}^i | x_t^i)}{\sum_i \sum_k w_k^i f(\bar{x}_{t+1}^i | x_t^i) \delta_x(x_t^i)}
\]

of the backward kernel. This amounts to making use of the backwards simulation strategy

\[
\tilde{x}_T \sim \hat{p}(x_T | y_{1:T}), \\
\tilde{x}_t \sim \hat{p}(x_t | \tilde{x}_{t+1}, y_{1:T}),
\]

to sample an entire backward trajectory \( \tilde{x}_{1:T} \).

Still computationally very costly!

Particle based FFBSi

**Algorithm 6** Particle based FFBSi

1. Run a PF to generate \( \{x_t^i, w_t^i\}_{i=1}^N \) for \( t = 1, \ldots, T \).
2. **Initialization** \( (t = T) \):
   (a) Sample independently \( \{b_T(j)\}_{j=1}^M \sim C(\{w_T^i\}_{i=1}^N) \).
   (b) Set \( \tilde{x}_T^j = x_T^{br(j)} \) for \( j = 1, \ldots, M \).
3. for \( t = T - 1 \) to 1 do
4. for \( j = 1 \) to \( M \) do
   (a) Compute \( \tilde{w}_{t+1}^j \propto w_t^i f(\bar{x}_{t+1}^i | x_t^i) \) and normalize.
   (b) Sample independently \( b_t(j) \sim C(\{\tilde{w}_{t+1}^j\}_{i=1}^N) \)
   (c) Set \( \tilde{x}_t^j = x_t^{b(j)} \) and \( \tilde{x}_{1:T} = \{\tilde{x}_1^j, \tilde{x}_{1:t+1}^j\} \).

Particle smoother – backwards simulation

1. At time \( T \) draw among the particles according to
   \[
P(\tilde{x}_T = x_T^i) = w_T^i.
\]
2. At time \( t = T - 1, \ldots, 1 \) draw among the particles according to
   \[
P(\tilde{x}_t = x_t^i) = \tilde{w}_t^i / \tilde{w}_{T | T}.
\]

Repeat \( M \) times, resulting in

\[
\tilde{p}^M(x_{1:T} | y_{1:T}) = \frac{1}{M} \sum_{j=1}^M \delta_{\tilde{x}_{1:T}^j}. \]

Illustration of a particle system \( \{x_t^i, w_t^i\}_{i=1}^4 \) generated using \( N = 4 \) particles during \( T = 5 \) time steps. The size of the particle visualize the corresponding weight.
Particle based BSi – example (II/II)

The backward smoothing weights \( \tilde{w}_{t \mid T} \) are visualized using circles.

Sampling from the backward kernel - in words

The BSi samples a trajectory from the empirical joint smoothing density by first choosing a particle \( x_{m \mid T} \) with probability proportional to \( w_{m \mid T} \) and then generating the ancestral path \( b_{1 \mid T} \) according to the backward kernel.

The resulting trajectory is

\[
x_{1 \mid T}^* = \{ x_{b_1}, \ldots, x_{b_T} \}.
\]

The computational cost is \( O(N^2) \), which is too expensive to be practical.

Sampling from the backward kernel - pictures

Recall that the main problem with using the PF to sample from \( p(x_{1 \mid T} \mid y_{1 \mid T}) \) is the particle degeneracy problem.

In the figures we show two samples from \( p(x_{1 \mid T} \mid y_{1 \mid T}) \) generated using the PF. The diversity is completely gone from time \( \approx 30 \) and backwards.

Sampling from the backward kernel - pictures

The backward simulator resolves the path degeneracy problem.

Note that in the above samples there is no loss of diversity.
Towards a practical particle smoother

To obtain a practical algorithm we have to reduce the computational complexity, how?!

**Key insight:** We do not have to compute all the smoothing weights \( \{ \tilde{w}_{t|T}^{i,j} \} \) to be able to sample from the empirical backward kernel.

**How:** Use rejection sampling to sample the indices! This means that we should not have to compute \( N \) indices just to draw one sample.

For an introduction to particle smoothers, see


Outline – Part 2

**Aim:** Derive the bootstrap particle filter and provide some of its properties.

1. Summary of Part 1
2. The Monte Carlo idea
3. Importance sampling
4. Derive the bootstrap particle filter
5. Particle smoothers (if there is time)
6. Sequential Monte Carlo
   a) General problem formulation
   b) The underlying pdf
   c) Convergence results

SMC – (abstract) problem formulation

The distribution of interest, \( \pi_t(x_{1:t}) \) is called target distribution.

**Problem formulation:** Sample sequentially from a sequence of target distributions \( \{ \pi_t(x_{1:t}) \}_{t \geq 1} \) of increasing dimension, where

\[
\pi_t(x_{1:t}) = \frac{\gamma_t(x_{1:t})}{Z_t},
\]

such that \( \gamma_t(x_t) : X^t \to \mathbb{R}^+ \) is known pointwise and \( Z_t = \int \pi(x_{1:t}) dx_{1:t} \) is computationally challenging.

So far we have seen that this formulation includes nonlinear SSMs,

\[
\pi_t(x_{1:t}) = p(x_{1:t} | y_{1:t}), \quad \gamma_t(x_{1:t}) = p(x_{1:t}, y_{1:t}), \quad Z_t = p(y_{1:t}),
\]

but importantly it is more general than that.

Generality of SMC

The sequence of target distributions \( \{ \pi_t(x_{1:t}) \}_{t \geq 1} \) can be constructed in many different ways!

Two concrete examples:

1. When variables are not defined on product spaces, \( \pi : X \to \mathbb{R}^+ \) we can introduce an artificial sequence of (auxiliary) distributions, where we are only interested in one of the marginals.


2. Inference in probabilistic graphical models (PGM) is possible via such a sequence of auxiliary distributions. SMC provide consistent estimates and an unbiased estimate of the partition (normalization) constant (also for loopy PGMs!).

**Sequential Monte Carlo – particle filter**

SMC = resampling + sequential importance sampling

1. Resampling: \( P(a_i^t = j) = \bar{w}_i^{t-1}/\sum_l \bar{w}_l^{t-1} \).
2. Propagation: \( x_i^t \sim f_\theta(x_t | x_a_i^{1:t-1}) \) and \( x_{1:t} = \{ x_{1:t-1}, x_t^i \} \).
3. Weighting: \( \bar{w}_i^t = W_t(x_i^t) = g_\theta(y_t | x_i^t) \).

The ancestor indices \( \{ a_i^t \}_{i=1}^N \) are very useful auxiliary variables! They make the stochasticity of the resampling step explicit.

---

**Convergence results in one slide...**

Let \( \varphi : X \mapsto \mathbb{R} \) be some test function of interest. The expectation

\[
E \left[ \varphi(x_t) \mid y_{1:t} \right] = \int \varphi(x_t) p(x_t \mid y_{1:t}) dx_t,
\]

can be estimated by the particle filter

\[
\hat{\varphi}_t^N = \sum_{i=1}^N \bar{w}_i^t \varphi(x_i^t).
\]

The CLT governing the convergence of this estimator states

\[
\sqrt{N} \left( \hat{\varphi}_t^N - E \left[ \varphi(x_t) \mid y_{1:t} \right] \right) \xrightarrow{d} N(0, \sigma_t^2(\varphi)).
\]

The likelihood estimate \( \hat{p}(y_{1:t}) = \prod_{s=1}^t \left\{ \frac{1}{N} \sum_{i=1}^N \bar{w}_i^s \right\} \) from the PF is unbiased, \( E_{\psi} [\hat{p}(y_{1:t})] = p(y_{1:t}) \) for any value of \( N \) and there are CLTs available as well.