Sequential Monte Carlo Methods

Lecture 17 – SMC for Probabilistic Programs

Lawrence Murray, Uppsala University
2017-08-29
Aim:

- Introduce probabilistic programming as a modeling paradigm.
- Demonstrate SMC as an appropriate inference method.

Outline:

1. Probabilistic programs.
2. Some examples in Birch.
3. SMC for probabilistic programs.
Probabilistic programs
Consider a program that depends on random numbers.
Execute that program on a processor.
As it runs, its memory state evolves \textit{dynamically} and \textit{stochastically} in time.
Consider a program that depends on random numbers.

Execute that program on a processor.

As it runs, its memory state evolves *dynamically* and *stochastically* in time.

We can think of the running program as a *stochastic process*. 
• Let \( k = 1, 2, \ldots \) denote a sequence of **checkpoints**.

• Let \( (x_{1:k})_{k\geq 1} \) denote the (memory) state of the running program at checkpoint \( k \), where \( x_{1:k} \in \mathcal{X}_{1:k} \) and \( \mathcal{X}_{1:k} = \mathcal{X}_k \times \mathcal{X}_{1:k-1} \).

• The state transitions according to \( p_k(x_k | x_{1:k-1}) \).
• Let $k = 1, 2, \ldots$ denote a sequence of **checkpoints**.
• Let $(x_{1:k})_{k \geq 1}$ denote the (memory) state of the running program at checkpoint $k$, where $x_{1:k} \in \mathcal{X}_{1:k}$ and $\mathcal{X}_{1:k} = \mathcal{X}_k \times \mathcal{X}_{1:k-1}$.
• The state transitions according to $p_k(x_k | x_{1:k-1})$.

At each checkpoint we can manipulate the running program: pause execution, inspect memory state, consider distributions over that state, modify that state. This is what facilitates inference.
What is probabilistic programming?

Probabilistic programming is a programming paradigm that emphasises this perspective on programs.

Consider other programming paradigms that emphasise other perspectives: functional, imperative, object-oriented, aspect-oriented.
A probabilistic programming language (PPL) is a programming language that provides ergonomic support for the probabilistic programming paradigm.

A PPL may provide, for example:

- A library of probability distributions with the ability to evaluate and simulate them.
- Specialised language features for specifying probabilistic models.
- Specialised language features for writing probabilistic inference methods.
A probabilistic program encodes a probabilistic model according to the semantics of a particular probabilistic programming language.

Probabilistic programs extend graphical models with support for stochastic branching.
The particular PPL that we will use is **Birch**, which is currently being developed at Uppsala University.

- It is the successor of **LibBi** (www.libbi.org).
- It is a probabilistic and object-oriented language.
- It compiles down to C++.
Example #1

\[ x \sim \text{Gaussian}(0.0, 1.0); \]
\[ y \sim \text{Gaussian}(x, 1.0); \]
\[ z \sim \text{Gaussian}(y, 1.0); \]
x ~ Gaussian(0.0, 1.0);
y ~ Gaussian(x, 1.0);
z ~ Gaussian(y, 1.0);

Adopting operational semantics, the interpretation of a program is defined by its execution. Here, the program encodes a joint distribution.
Example #1

\[
x \sim \text{Gaussian}(0.0, 1.0);
\]
\[
y \sim \text{Gaussian}(x, 1.0);
\]
\[
z \sim \text{Gaussian}(y, 1.0);
\]
Example #1

\[ x \sim \text{Gaussian}(0.0, 1.0); \]
\[ y \sim \text{Gaussian}(x, 1.0); \]
\[ z \sim \text{Gaussian}(y, 1.0); \]

\[ p(y \mid x) \]
Example #1

\[x \sim \text{Gaussian}(0.0, 1.0);\]
\[y \sim \text{Gaussian}(x, 1.0);\]
\[z \sim \text{Gaussian}(y, 1.0);\]

\(p(z \mid y)\)

\[
\begin{array}{c}
\text{x} \\
\rightarrow
\text{y} \\
\rightarrow
\text{z}
\end{array}
\]
Example #1

\[
x \sim \text{Gaussian}(0.0, 1.0); \\
y \sim \text{Gaussian}(x, 1.0); \\
z \sim \text{Gaussian}(y, 1.0);
\]

\[
p(x) \quad p(y \mid x) \quad p(z \mid y)
\]
\[ \beta \sim \text{Bernoulli}(0.5); \]
\[ x \sim \text{Gaussian}(0.0, 1.0); \]
\[ \text{if } (\beta) \{ \]
\[ \quad y \sim \text{Gaussian}(x, 1.0); \]
\[ \} \text{ else } \{ \]
\[ \quad y \sim \text{Gaussian}(0.0, 1.0); \]
\[ \} \]
Example #2

\[
\begin{align*}
\beta & \sim \text{Bernoulli}(0.5); \\
x & \sim \text{Gaussian}(0.0, 1.0); \\
\text{if} \ (\beta) \ {\{ } & \\
\quad y & \sim \text{Gaussian}(x, 1.0); \\
\text{\} else \ {\{ } & \\
\quad y & \sim \text{Gaussian}(0.0, 1.0); \\
\text{\} }
\end{align*}
\]
\( \beta \sim \text{Bernoulli}(0.5); \)
\( x \sim \text{Gaussian}(0.0, 1.0); \)
if (\( \beta \)) {
    y \sim \text{Gaussian}(x, 1.0);
} else {
    y \sim \text{Gaussian}(0.0, 1.0);
}
Example #2

\[
\beta \sim \text{Bernoulli}(0.5);
\]

\[
x \sim \text{Gaussian}(0.0, 1.0);
\]

\[
\text{if } (\beta) \{
\quad y \sim \text{Gaussian}(x, 1.0);
\}
\]

\[
\text{else } \{
\quad y \sim \text{Gaussian}(0.0, 1.0);
\}
\]

\[
p(x)
\]
\[ \beta \sim \text{Bernoulli}(0.5); \]
\[ x \sim \text{Gaussian}(0.0, 1.0); \]
\[ \text{if } (\beta) \{ \]
\[ \quad y \sim \text{Gaussian}(x, 1.0); \]
\[ \} \text{ else } \{ \]
\[ \quad y \sim \text{Gaussian}(0.0, 1.0); \]
\[ \} \]

\[ p(y \mid x, \beta) \]
Example #2

\[ \beta \sim \text{Bernoulli}(0.5); \]
\[ x \sim \text{Gaussian}(0.0, 1.0); \]
if (\( \beta \)) {
    \[ y \sim \text{Gaussian}(x, 1.0); \]
} else {
    \[ y \sim \text{Gaussian}(0.0, 1.0); \]
}
Example #3

\[x[1] \sim \text{Gaussian}(0.0, 1.0);\]
\[y[1] \sim \text{Gaussian}(x[1], 1.0);\]

for (t in 2..T) {
    \[x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);\]
    \[y[t] \sim \text{Gaussian}(x[t], 1.0);\]
}
Example #3

\[ x[1] \sim \text{Gaussian}(0.0, 1.0); \]
\[ y[1] \sim \text{Gaussian}(x[1], 1.0); \]
\[ \text{for (t in 2..T) } \{ \]
\[ x[t] \sim \text{Gaussian}(a*x[t - 1], 1.0); \]
\[ y[t] \sim \text{Gaussian}(x[t], 1.0); \]
\[ \} \]
Example #3

\[
x[1] \sim \text{Gaussian}(0.0, 1.0); \\
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]

for (t in 2..T) {
    \[
x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
    \]
    \[
y[t] \sim \text{Gaussian}(x[t], 1.0);
    \]
}
Example #3

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]

\[
p(y_1 | x_1)
\]

for (t in 2..T) {
  x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
  y[t] \sim \text{Gaussian}(x[t], 1.0);
}

\[
\begin{align*}
x[1] \\
y[1]
\end{align*}
\]
Example #3

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
\]
\[
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]
\[
\text{for } (t \text{ in } 2..T) \{
\]
\[
x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
\]
\[
y[t] \sim \text{Gaussian}(x[t], 1.0);
\]
\[
\}
\]

\[
p(X_t \mid X_{t-1})
\]
\[
x[1] \sim \text{Gaussian}(0.0, 1.0); \\
y[1] \sim \text{Gaussian}(x[1], 1.0); \\
\text{for (t in 2..T)} \{ \\
\quad x[t] \sim \text{Gaussian}(a*x[t - 1], 1.0); \\
\quad y[t] \sim \text{Gaussian}(x[t], 1.0); \\
\}
\]

\[p(y_t | x_t)\]
Example #3

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
\]
\[
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]
\[
\text{for (} t \text{ in 2..T) }
\]
\[
\quad x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
\]
\[
\quad y[t] \sim \text{Gaussian}(x[t], 1.0);
\]

\[
p(x_t \mid x_{t-1})
\]
x[1] ~ Gaussian(0.0, 1.0);  
y[1] ~ Gaussian(x[1], 1.0);  
for (t in 2..T) {
  x[t] ~ Gaussian(a*x[t - 1], 1.0);  
  y[t] ~ Gaussian(x[t], 1.0);  
}  

\[ p(y_t | x_t) \]
Example #3

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
\]

\[
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]

for (t in 2..T) {
    \[
x[t] \sim \text{Gaussian}(a \cdot x[t - 1], 1.0);
    \]
    \[p(x_t \mid x_{t-1})\]
    \[
y[t] \sim \text{Gaussian}(x[t], 1.0);
    \]
}

\[
x[1]
\]

\[
\text{y}[1]
\]

\[
x[2]
\]

\[
\text{y}[2]
\]

\[
x[3]
\]

\[
\text{y}[3]
\]

\[
x[4]
\]
x[1] ~ Gaussian(0.0, 1.0);
y[1] ~ Gaussian(x[1], 1.0);
for (t in 2..T) {
    x[t] ~ Gaussian(a*x[t - 1], 1.0);
    y[t] ~ Gaussian(x[t], 1.0);
}

\[ p(y_t | x_t) \]
Example #3

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]

\[
\text{for (t in 2..T) }
\]
\[
x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
y[t] \sim \text{Gaussian}(x[t], 1.0);
\]

\[
p(x_t | x_{t-1})
\]
Example #3

\[ x[1] \sim \text{Gaussian}(0.0, 1.0); \]
\[ y[1] \sim \text{Gaussian}(x[1], 1.0); \]
\[
\text{for (t in 2..T) }
\begin{align*}
  x[t] & \sim \text{Gaussian}(a \times x[t - 1], 1.0); \\
  y[t] & \sim \text{Gaussian}(x[t], 1.0);
\end{align*}
\]

\[ p(y_t | x_t) \]

\[
\begin{array}{l}
  x[1] \\
  y[1] \\
  x[2] \\
  y[2] \\
  x[3] \\
  y[3] \\
  x[4] \\
  y[4] \\
  x[5] \\
  y[5]
\end{array}
\]
Example #3

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
\]
\[
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]
for (t in 2..T) { 
  \[
x[t] \sim \text{Gaussian}(a*x[t - 1], 1.0);
\]
  \[
y[t] \sim \text{Gaussian}(x[t], 1.0);
\]
}
The PPL will define checkpoints when interesting events happen in the running of the program. A typical setup uses two categories of checkpoint:

1. **Sample** when \( x \) is distributed according to some distribution \( p \) and should be sampled.
2. **Observe** when \( x \) is distributed according to some distribution \( p \) and should be observed to have some given value.

In Birch, these are triggered by special operators:

1. \( x \sim p \)
2. \( x \rightarrow p \)
Example (Checkpoints)

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]
\[
\text{for}\ (t\ \text{in}\ 2..T)\ \{\n\quad x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);\n\quad y[t] \sim \text{Gaussian}(x[t], 1.0);\n\}\n\]

Now, the program explicitly states which variables must be sampled, and which have given values and should be observed. The program encodes a **posterior distribution**.
Example (Checkpoints)

```plaintext
x[1] <- Gaussian(0.0, 1.0);
y[1] ~> Gaussian(x[1], 1.0);
for (t in 2..T) {
    x[t] <- Gaussian(a*x[t - 1], 1.0);
y[t] ~> Gaussian(x[t], 1.0);
}
```

sample(x[1])
x[1] ~ Gaussian(0.0, 1.0);
y[1] ~> Gaussian(x[1], 1.0);

for (t in 2..T) {
    x[t] ~ Gaussian(a*x[t - 1], 1.0);
    y[t] ~> Gaussian(x[t], 1.0);
}

observe(x[1])
Example (Checkpoints)

\[
\begin{align*}
x[1] & \sim \text{Gaussian}(0.0, 1.0); \\
y[1] & \triangleright \text{Gaussian}(x[1], 1.0); \\
\text{for } (t \text{ in } 2..T) \{} \\
\quad x[t] & \sim \text{Gaussian}(a \times x[t - 1], 1.0); \\
\quad y[t] & \triangleright \text{Gaussian}(x[t], 1.0); \\
\}\end{align*}
\]

\text{sample}(x[t])
Example (Checkpoints)

```plaintext
x[1] <- Gaussian(0.0, 1.0);
y[1] ~> Gaussian(x[1], 1.0);
for (t in 2..T) {
  x[t] <- Gaussian(a*x[t - 1], 1.0);
  y[t] ~> Gaussian(x[t], 1.0);
  observe(x[t])
}
```

Diagram:

```
```
Example (Checkpoints)

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
y[1] \simarrow \text{Gaussian}(x[1], 1.0);
\]

for (t in 2..T) {
\[
x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
y[t] \simarrow \text{Gaussian}(x[t], 1.0);
\]
}

\text{sample}(x[t])
Example (Checkpoints)

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
\]

\[
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]

for (t in 2..T) {
    \[
x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
    \]
    \[
y[t] \sim \text{Gaussian}(x[t], 1.0);
    \]
    observe(x[t])
}
x[1] <- Gaussian(0.0, 1.0);
y[1] ~> Gaussian(x[1], 1.0);
for (t in 2..T) {
    x[t] <- Gaussian(a*x[t - 1], 1.0);
    y[t] ~> Gaussian(x[t], 1.0);
}

sample(x[t])
Example (Checkpoints)

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
y[1] \rightarrow \text{Gaussian}(x[1], 1.0);
\]

for \( t \in 2..T \) {
\[
x[t] \sim \text{Gaussian}(a \cdot x[t - 1], 1.0);
y[t] \rightarrow \text{Gaussian}(x[t], 1.0);
\]

\text{observe}(x[t])
\begin{align*}
x[1] & \sim \text{Gaussian}(0.0, 1.0); \\
y[1] & \sim \text{Gaussian}(x[1], 1.0); \\
\text{for } (t \text{ in } 2..T) \{ \\
& \quad x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0); \\
& \quad y[t] \sim \text{Gaussian}(x[t], 1.0); \\
& \}
\end{align*}

Sample \(x[t]\)
Example (Checkpoints)

\[
x[1] \sim \text{Gaussian}(0.0, 1.0);
y[1] \sim \text{Gaussian}(x[1], 1.0);
\]

for (t in 2..T) {
    x[t] \sim \text{Gaussian}(a \times x[t - 1], 1.0);
y[t] \sim \text{Gaussian}(x[t], 1.0);
}

\text{observe}(x[t])

\[
\begin{align*}
\end{align*}
\]
Example (Checkpoints)

```
x[1] <- Gaussian(0.0, 1.0);
y[1] ~> Gaussian(x[1], 1.0);
for (t in 2..T) {
    x[t] <- Gaussian(a*x[t - 1], 1.0);
y[t] ~> Gaussian(x[t], 1.0);
}
```

Algorithm:
1. Sample $x[1] \sim \text{Gaussian}(0.0, 1.0)$;
2. Observe $y[1] \sim \text{Gaussian}(x[1], 1.0)$;
3. For $t = 2, \ldots, T$:
   - Sample $x[t] \sim \text{Gaussian}(a \cdot x[t - 1], 1.0)$;
   - Observe $y[t] \sim \text{Gaussian}(x[t], 1.0)$.
A probabilistic program encodes a **probabilistic model**.

A running probabilistic program is a **stochastic process**.
SMC for probabilistic programs
Recall that SMC can be used to approximate a sequence of probability distributions on a sequence of probability spaces of increasing dimension.

Let \( \{ \pi_k(x_{1:k}) \}_{k \geq 1} \) be the sequence of target distributions

\[
\pi_k(x_{1:k}) = \frac{\tilde{\pi}_k(x_{1:k})}{Z_k}
\]

Where

\[
\pi_k(x_{1:k}) \approx \sum_{i=1}^{N} w_k^i \delta_{x_1^i}(x_{1:k})
\]

and the weighted particle populations \( \{ x_{1:k}^i, w_k^i \}_{i=1}^{N} \) are generated sequentially for \( k = 1, 2, \ldots \).
We can use SMC for inference on running probabilistic programs.

- Each of the $N$ particles is a running probabilistic program.
- We have:
  \[
  \tilde{\pi}_k(x_{1:k}) = p_k(x_k | x_{1:k-1})\tilde{\pi}_{k-1}(x_{1:k-1})
  
  q_k(x_k | x_{1:k-1}) = p_k(x_k | x_{1:k-1}).
  \]
- That is, the probabilistic program defines the target and the proposal, much like the bootstrap particle filter.
Assume that we have obtained \( \{x_{1:k-1}^i, w_{k-1}^i\}^N_{i=1} \).

1. **Resample:** Sample \( a_k^i \) with \( P(a_k^i = j) = \nu_{k-1}^j, j = 1, \ldots, N \).

2. If this is a **sample** checkpoint, then **propagate:**

\[
x_k^i \sim p_k(x_k | x_{1:k-1}^{a_k^i}) \text{ and } x_{1:k}^i = (x_{1:k-1}^{a_k^i}, x_k^i)
\]

3. If this is an **observe** checkpoint, then **weight:**

\[
w_k^i \propto \frac{w_{k-1}^{a_k^i} p_k(x_k^i | x_{1:k-1}^{a_k^i})}{\nu_{k-1}^{a_k^i}}.
\]

The result is a new weighted set of particles \( \{x_{1:k}^i, w_k^i\}^N_{i=1} \).
A probabilistic program encodes a probabilistic model according to the semantics of a particular probabilistic programming language.

The memory state of a running probabilistic program evolves dynamically and stochastically in time and so is a stochastic process.

General Sequential Monte Carlo can be applied to perform inference across a sequence of target distributions defined by the checkpoints of the running program.
Further study
## Probabilistic Programming Languages

<table>
<thead>
<tr>
<th>Year</th>
<th>PL</th>
<th>AI</th>
<th>ML</th>
<th>STATS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>Figaro</td>
<td>ProbLog</td>
<td>webPPL</td>
<td>STAN</td>
</tr>
<tr>
<td></td>
<td>HANSAI</td>
<td></td>
<td>Venture</td>
<td>LibBi</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Probabilistic-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Church</td>
<td>JAGS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Infer.NET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Factorie</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Blog</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>IBAL</td>
<td>Prism</td>
<td>KMP</td>
<td>WinBUGS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td></td>
<td></td>
<td></td>
<td>BUGS</td>
</tr>
</tbody>
</table>

(Figure courtesy of Frank Wood, Oxford.)
Probabilistic Programming Languages

(Figure courtesy of Frank Wood, Oxford.)