Aim: Introduce probabilistic programming as a new and exciting way of probabilistic modeling and inference, explain the basic concepts and demonstrate how SMC can be used for inference.

Outline:

1. Probabilistic programming
2. Inference (importance sampling and SMC)
Probabilistic programming
Probabilistic modeling in a nutshell:

1. Write down the model in the language of mathematics
2. Derive a (bespoke) inference algorithm
3. Implement the algorithm as a computer program and run it

Developing probabilistic models and inference algorithms is a time-consuming and error-prone process.
Developing probabilistic models and inference algorithms is a time-consuming and error-prone process.

*Probabilistic programming* is a (relatively) new approach to change this:

- The generative model is written as a computer program
- Automatic inference (as an integral part of the programming language)
Why probabilistic programming?

Fast development of models

• Easy to write generative models as programs
• No need for deriving and implementing a bespoke inference algorithm

More expressive models

• Programs can use stochastic branching and recursion, and therefore more expressive than graphical models

Development of widely-applicable inference algorithms
Consider the following toy model:

\[
\begin{align*}
    x_1 & \sim \mathcal{N}(0, 5^2) \\
    y_1 & \sim \mathcal{N}(x_1, 1) \\
    x_2 & \sim \mathcal{N}(0.5x_1, 1) \\
    y_2 & \sim \mathcal{N}(x_2, 1)
\end{align*}
\]

The model in Matlab:

```matlab
1  x1 = normrnd(0, 5);
2  y1 = normrnd(x1, 1);
3  x2 = normrnd(0.5*x1, 1);
4  y2 = normrnd(x2, 1);
5  fprintf(’x1=%f, x2=%f, y1=%f, y2=%f\n’, x1, x2, y1, y2);
```

We can run the script multiple times to get samples from the joint distribution \( p(x_1, x_2, y_1, y_2) \).
We observed that $y_1 = 4.78$ and $y_2 = 3.12$, and want to know the posterior distribution $p(x_1|y_1 = 4.78, y_2 = 3.12)$. Can we change the Matlab script to get samples from this posterior distribution?
A probabilistic programming language (PPL) is a programming language that provides ergonomic support for random variables and automatic inference.

Probabilistic constructs:

• **assume** – declaring a random variable by specifying its probability distribution:

  \[ \text{variable} \sim \text{Distribution}(...) \]

• **observe** – conditioning on the observed data:

  \[ \text{observe} \ \text{Distribution}(...) \ \text{value} \]
Programs may use random variables as though any ordinary variable, even to control the flow of the execution.

Example:

\[ x \sim \text{Normal}(0, 1) \]
\[
\text{if } x > 0.5 \text{ then}
\quad y \sim \text{Normal}(x, 1)
\]
\[
\text{else}
\quad y \sim \text{Exponential}(1)
\]
end if
Example: Birth-death model for generating trees (birth rate $\lambda$, death rate $\mu$)

function $\text{TREE}(\tau)$

$\Delta \sim \text{Exponential}(\lambda + \mu)$

$\tau' \leftarrow \tau - \Delta$

if $\tau' < 0$ then

    return $(0, \emptyset)$

end if

$b \sim \text{Bernoulli}(\lambda/(\lambda + \mu))$

if $b$ then

    return $(\tau', \{\text{TREE}(\tau'), \text{TREE}(\tau')\})$

else

    return $(\tau', \emptyset)$

end if

end function
Toy model as a probabilistic program

\[
y_1 \leftarrow 4.78
y_2 \leftarrow 3.12
\]

\[
x_1 \sim \text{Normal}(0, 5)
\text{observe Normal}(x_1, 1) \ y_1
x_2 \sim \text{Normal}(0.5 \times x_1, 1)
\text{observe Normal}(x_2, 1) \ y_2
\text{return } x_1
\]

Note: A probabilistic program encodes a posterior distribution.

Let’s implement this model in WebPPL, a simple PPL you can run in your browser (http://webppl.org).
Toy model in WebPPL

```javascript
var model = function() {
  var x1 = sample(Gaussian({mu: 0, sigma: 5}));
  observe(Gaussian({mu: x1, sigma: 1}), 4.78);
  var x2 = sample(Gaussian({mu: 0.5*x1, sigma: 1}));
  observe(Gaussian({mu: x2, sigma: 1}), 3.12);
  return x1;
}

var dist = Infer({method: 'SMC', particles: 10000}, model);
viz.auto(dist);
```
Deterministic programming vs. statistics vs. PPL

(Deterministic) programming

Parameters

Program

Output

Statistics

$\theta$

$p(X|\theta)$

$X$

Observations

Probabilistic programming

Parameters

Program

Observations

Based on a figure by Frank Wood.
Inference
Automatic inference is a difficult task.

- **Exact inference**
  - Analytical solutions (e.g. Kalman filtering)
  - Enumeration (for discrete models of limited dimension)

- **Approximate inference**
  - Monte Carlo inference
    - Markov chain Monte Carlo (MCMC)
    - Sequential Monte Carlo (SMC)
    - Hamiltonian Monte Carlo (HMC)
  - Variational inference
Note: We make some simplifications for pedagogical reasons.

**Program state**: the contents of the memory used by the program to store its data.

**Immediate sampling**: During execution of a probabilistic program, whenever the program encounters an unobserved random variable (i.e. an assume statement), it immediately samples its value from the distribution associated with it.
Graphical model of the execution

\[ \begin{align*}
\emptyset & \xrightarrow{f_1(X_1|\emptyset)} X_1 \\
X_1 & \xrightarrow{f_2(X_2|X_1)} X_2 \\
X_2 & \xrightarrow{f_3(X_3|X_2)} \ldots \\
X_t & \xrightarrow{f_{t+1}(X_{t+1}|X_t)} \ldots \\
y_1 & \xrightarrow{g_1(y_1|X_1)} X_1 \\
y_2 & \xrightarrow{g_2(y_2|X_2)} X_2 \\
y_t & \xrightarrow{g_t(y_t|X_t)} X_t
\end{align*} \]

\( X_i \) denotes the state at the \( i \)-th observe statement.

\[ p(X_{1:T}, y_{1:T}) = \prod_{t=1}^{T} f_t(X_t|X_{t-1})g_t(y_t|X_t), \]

where \( X_0 = \emptyset \). We are interested in the posterior probability

\[ p(X_{1:T}|y_{1:T}) = \frac{p(X_{1:T}, y_{1:T})}{p(y_{1:T})} \propto p(X_{1:T}, y_{1:T}). \]
Importance sampling I

We can use importance sampling (IS) to sample from $p(\mathcal{X}_{1:T}|y_{1:T})$.

Proposal distribution:

$$q(\mathcal{X}_{1:T}) = \prod_{t=1}^{T} f_t(\mathcal{X}_t|\mathcal{X}_{t-1}).$$

The importance weight:

$$w(\mathcal{X}_{1:T}) = \prod_{t=1}^{T} g_t(y_t|\mathcal{X}_t).$$

Exercise

How can we sample from $f_t(\mathcal{X}_t|\mathcal{X}_{t-1})$?
How can we calculate $g_t(y_t|\mathcal{X}_t)$?
Run the program forward with the following handling of probabilistic statements (checkpoints):

- **assume**: sample a value of the random variable (immediate sampling)
- **observe**: update the weight by multiplying it with the likelihood of the observed value w.r.t. its distribution and parameters (calculated during the execution).
Importance sampling for the toy example

```
var model = function() {
  var x1 = sample(Gaussian({mu: 0, sigma: 5}));
  observe(Gaussian({mu: x1, sigma: 1}), 4.78);
  var x2 = sample(Gaussian({mu: 0.5*x1, sigma: 1}));
  observe(Gaussian({mu: x2, sigma: 1}), 3.12);
  return x1;
}
```

\( w \leftarrow 1 \) (initial weight)
var model = function() {
    var x1 = sample(Gaussian({mu: 0, sigma: 5}));
    observe(Gaussian({mu: x1, sigma: 1}), 4.78);
    var x2 = sample(Gaussian({mu: 0.5*x1, sigma: 1}));
    observe(Gaussian({mu: x2, sigma: 1}), 3.12);
    return x1;
};

Sampling $x_1$ from $\mathcal{N}(0, 5^2)$: $x_1 \leftarrow 4.1$
Importance sampling for the toy example

```javascript
var model = function() {
  var x1 = sample(Gaussian({mu: 0, sigma: 5}));
  observe(Gaussian({mu: x1, sigma: 1}), 4.78);
  var x2 = sample(Gaussian({mu: 0.5*x1, sigma: 1}));
  observe(Gaussian({mu: x2, sigma: 1}), 3.12);
  return x1;
}

w ← w * N(4.78 | 4.1, 1) = 0.3166
```
importance sampling for the toy example

```javascript
var model = function() {
    var x1 = sample(Gaussian({mu: 0, sigma: 5}));
    observe(Gaussian({mu: x1, sigma: 1}), 4.78);
    var x2 = sample(Gaussian({mu: 0.5*x1, sigma: 1}));
    observe(Gaussian({mu: x2, sigma: 1}), 3.12);
    return x1;
}
```

Sampling $x_2$ from $\mathcal{N}(2.05, 1)$: $x_2 \leftarrow 2.7$
Importance sampling for the toy example

```javascript
var model = function() {
  var x1 = sample(Gaussian({mu: 0, sigma: 5}));
  observe(Gaussian({mu: x1, sigma: 1}), 4.78);
  var x2 = sample(Gaussian({mu: 0.5*x1, sigma: 1}));
  observe(Gaussian({mu: x2, sigma: 1}), 3.12);
  return x1;
}

w ← w * N(3.12 | 2.7, 1) = w * 0.3653 = 0.1156
```
var model = function() {
    var x1 = sample(Gaussian({mu: 0, sigma: 5}));
    observe(Gaussian({mu: x1, sigma: 1}), 4.78);
    var x2 = sample(Gaussian({mu: 0.5*x1, sigma: 1}));
    observe(Gaussian({mu: x2, sigma: 1}), 3.12);
    return x1;
}

Returned value 4.1, weight 0.1156
Recall the disadvantages of importance sampling. Better idea is to use SMC methods.

**Bootstrap particle filter**

- Start a set of $N$ parallel executions (*particles*) of the program.
- Repeat:
  - Run each particle until the next observe statement (incl. the weight calculation) and pause the execution.
  - Resample the particles and resume execution.
SMC II

Particles

- Run until observe $y_1$
- observe $y_1$
- Resampling
- Cont. until observe $y_2$
- observe $y_2$
Examples of PPLs

There already exist quite a few PPLs today with different programming paradigms, for example:

- Functional: Anglican and Venture
- Imperative: Probabilistic C, Turing, Stan, Edward and Pyro
- Object-oriented: Birch (with delayed sampling)
Want to learn more?

Noah D. Goodman and Andreas Stuhlmüller.  
The design and implementation of probabilistic programming languages.  
*Retrieved 2019-8-29 from http://dippl.org*

Jan-Willem van de Meent et al.  
An introduction to probabilistic programming.  

Lawrence M. Murray and Thomas B. Schön.  
Automated learning with a probabilistic programming language: Birch.  
*Annual Reviews in Control, 2018.*