Probabilistic Programming (WS22/23)
Prof. Dr. Ir. Dr. h.c. Joost-Pieter Katoen
Lutz Klinkenberg, Philipp Schroer, Tobias Winkler

Lehrstuhl für Informatik 2 Softwaremodellierung und Verifikation

RWTH AACHEN UNIVERSITY

Exercise Sheet 1

General remarks:

- **Due date:** October 21st 12:30 (before the exercise class).
- Please submit your solutions via MOODLE. Remember to provide your matriculation number. It is necessary to hand in your solutions in groups of **three**. You may use the MOODLE forum to form groups.
- Solutions must be written in English.
- While we will publish sketches of exercise solutions, we do *not* guarantee that these sketches contain all details that are necessary to properly solve an exercise. Hence, it is recommended to attend the exercise classes.
- If you have any questions regarding the lecture or the exercise, please use the forum in MOODLE.
- Provide solutions to programming exercises in a separate plain text file. Please name the files according to the exercise they belong to (e.g. 'sheet01_ex1a.wppl'). All your programs are required to run without errors in the online interpreter available at `http://webppl.org`.

Exercise 1 (Monty Hall problem) $\boxed{\textbf{25P}}$

The *Monty Hall problem*—probably one of the most famous brain teasers in history—goes as follows[1]:

> Suppose you are on a game show, and you are given the choice of three doors. Behind one door there is a car; behind the other two, goats. You pick a door, say No. 1, and the host, who knows what is behind the doors, opens another door, say No. 3, which has a goat. He then asks "Do you want to pick door No. 2 instaed of No. 1?" Is it to your advantage to switch your choice?

While many of you might already know the answer to the problem, your task in this exercise is to solve it with the help of a WEBPPL program. More specifically, you are asked to provide WEBPPL code that models both

- the situation where the player does *not* switch his or her choice, and
- the situation where the player *does* switch.

Solve the Monty Hall problem by comparing the outcome of your models for the following host strategies! Also state the probabilities of winning the car. **Note:** *You are allowed to use the full feature set of* WEBPPL[2].

(a) [15P] Monty chooses a door uniformly at random, but neither the door chosen by the player nor the one with the car behind it.

Solution: Switching has about a $\frac{2}{3}$ chance of winning.

[1] https://en.wikipedia.org/wiki/Monty_Hall_problem
[2] `https://webppl.readthedocs.io/en/master/gettingstarted.html`

```
var change = true;
var model = function() {
    var prize = randomInteger(3);
    var myChoice = randomInteger(3);

    var montyChoice = uniformDraw(
        filter(function(door) {
            return door != myChoice && door != prize }, [0, 1, 2])
    );

    return change ? prize == otherChoice : prize == myChoice;
}
var dist = Infer({method: 'rejection', samples: 1000}, model);
print(dist)
expectation(dist)
```

(b) [5P] Monty always chooses the door with the lowest number which is neither the player's choice nor the door with the car behind it.

Solution: Same as before, $\frac{2}{3}$ chance of winning when one switches.

```
var change = true;
var model = function() {
    var prize = randomInteger(3);
    var myChoice = randomInteger(3);

    var montyChoice =
        filter(function(door) {
            return door != myChoice && door != prize }, [0, 1, 2])[0]
    var otherChoice = filter(function(door) {
        return door != myChoice && door != montyChoice }, [0, 1, 2])[0];

    return change ? prize == otherChoice : prize == myChoice;
}
var dist = Infer({method: 'rejection', samples: 1000}, model);
print(dist)
expectation(dist)
```

(c) [5P] Monty chooses a door uniformly at random, but not the player's choice.

Solution: If Monty ignores where the car is, we do not gain information from his choice. So we have a $\frac{1}{3}$ chance of winning.

```
var change = true;
var model = function() {
    var prize = randomInteger(3);
    var myChoice = randomInteger(3);

    var montyChoice = uniformDraw(
        filter(function(door) { return door != myChoice }, [0, 1, 2])
    );
    var otherChoice = filter(function(door) {
        return door != myChoice && door != montyChoice }, [0, 1, 2])[0];

    return change ? prize == otherChoice : prize == myChoice;
}
var dist = Infer({method: 'rejection', samples: 1000}, model);
print(dist)
expectation(dist)
```

Exercise 2 (Gambler's ruin problem) **25P**

Consider a coin-flipping game with two players where each player has a 50% chance of winning with each flip of the coin. Assume that player 1 initially has n_1 pennies and player 2 has n_2 pennies. After each flip of the coin the loser transfers one penny to the winner. The game ends when one player has all the pennies.

Write a WebPPL program that models the game and use it to answer the following questions.

(a) [10P] If the second player starts with 10 pennies, what is the minimum number of pennies the first player should bring so that she wins with probability at least 70%?

Solution:

```
var game = function(n1, n2, n){
  if(n1 == n){
     return "1 wins"
  }
  else if(n2 == n){
     return "2 wins"
} else
  flip(.5) ? game(n1+1, n2-1, n) : game(n1-1, n2+1, n);
}

var play = function(){
  var n1 =  24 //Trying different numbers for n1 to see where p1 > 0.7
              // With different algorithms and sampling methods and rates,
              // you may get different results, which is fine as long as
              // your model is logically corrent
  var n2 = 10 //setting n2 to 10
  return game(n1, n2, n1+n2)
}
```

```
var options = {method: "rejection", samples: 10000}
var result = Infer(options, play)
viz.auto(result)
print(result)
```

(b) [15P] For $n_1 = 5$ and $n_2 = 5$, what is the expected number of rounds until the game ends?

Solution:

```
var game = function(n1, n2, n){
  if(n1 == n || n2 == n){
    return 1 // the base for counting the number of rounds
  }else{
      flip(0.5) ? 1 + game(n1+1, n2-1, n) : 1 + game(n1-1, n2+1, n)
  }
}

var play = function(){
  var n1 = 5
  var n2 = 5
  game(n1, n2, n1+n2)
}

var options = {method: "rejection", samples: 10000}
var result = Infer(options, play)
viz.auto(result)
print("Expected number of rounds: "
           + expectation(result))
```

Expected number of rounds is approximately 26

Exercise 3 (Dungeons and Dragons) $\boxed{\textbf{25P}}$

Assume you are playing a Dungeons and Dragons[3] game. Suppose that you have earned yourself a bonus that allows you to throw n dice with f faces (instead of only one die) and take the maximum value. For example, if you throw $n = 3$ dice with $f = 6$ faces yielding the outcomes $(1, 4, 5)$ then this particular throw counts as a 5.

(a) [15P] Write a WebPPL program that computes the expected value of a bonus throw, given the global variables n and f.

Solution:

```
var n = 3;
var f = 20;
```

[3] https://de.wikipedia.org/wiki/Dungeons_%26_Dragons

```
var model = function(){
  var throwDice = repeat(n, function(){return randomInteger(f)+1});
  return sort(throwDice, gt)[0];
}

var dist = Infer(model)
var avg = expectation(dist)
print("Expected value is: " + avg)
```

(b) [3P] Let $f = 20$. For each $n \in \{2, 3, 4\}$ compute the ratio of the expected value against the number of faces. Can you spot an approximate pattern?

Solution:

$$\begin{aligned}
&\text{Let } n = 2. \quad \text{The ratio is } 0.69125 \approx \tfrac{2}{3}.\\
&\text{Let } n = 3. \quad \text{The ratio is } 0.774375 \approx \tfrac{3}{4}.\\
&\text{Let } n = 4. \quad \text{The ratio is } 0.824166875 \approx \tfrac{4}{5}.
\end{aligned}$$

(c) [7P] Lets investigate the case where $n = 3$ a bit further. Compute (by hand, not in WebPPL) the ratio of the expected value against the number of faces f when $f \to \infty$. Compare the distance between the estimated ratio and the ratio computed in the limit.
Hint: *The number of possibilities where the maximum is x is given by $3x(x-1)+1$.*

Solution:

$$\begin{aligned}
\lim_{f\to\infty} \frac{\mathbb{E}[\max]}{f} &= \lim_{f\to\infty} \frac{1}{f} \cdot \sum_{x=1}^{f} \Pr(\{\max = x\}) \cdot x\\
&= \lim_{f\to\infty} \frac{1}{f} \cdot \sum_{x=1}^{f} \frac{3x(x-1)+1}{f^3} \cdot x\\
&= \lim_{f\to\infty} \frac{(f+1)(3f-1)}{4f^2}\\
&= \frac{1}{4} \cdot 1 \cdot 3 = \frac{3}{4}.\\
\text{The distance is: } \left|0.774375 - \frac{3}{4}\right| &= 0.024375 = \frac{39}{1600}
\end{aligned}$$

Exercise 4 (1-dimensional particle dynamics) $\boxed{\textbf{25P}}$

In this exercise we study the dynamics of a uniformly accelerated particle moving along one dimension. We assume that the initial position $x(t_0)$ of the particle has a Gaussian prior distribution with mean $\mu = 0$ and variance $\sigma = 1$. For the initial velocity $v(t_0)$ and acceleration a you can assume uniform priors in the interval $[1, 10]$ and $[1, 2]$, respectively:

$$\begin{aligned} x(t_0) &\sim \mathcal{N}(0,1) \\ v(t_0) &\sim \mathrm{Unif}(1,10) \\ a &\sim \mathrm{Unif}(1,2)\ . \end{aligned}$$

Note that the acceleration a is constant (it does not depend on t). To approximate the particle's dynamics we use the equations

$$\begin{aligned} x(t+h) &= x(t) + hv(t) \\ v(t+h) &= v(t) + ha \end{aligned}$$

for some step size $h > 0$. Now let $t_0 = 0$, $t_1 = 10$ and $h = 1$. Write a WebPPL program that models the situation and use it for the following tasks:

(a) [10P] Give an estimate of the expected value of the final position $x(t_1)$.

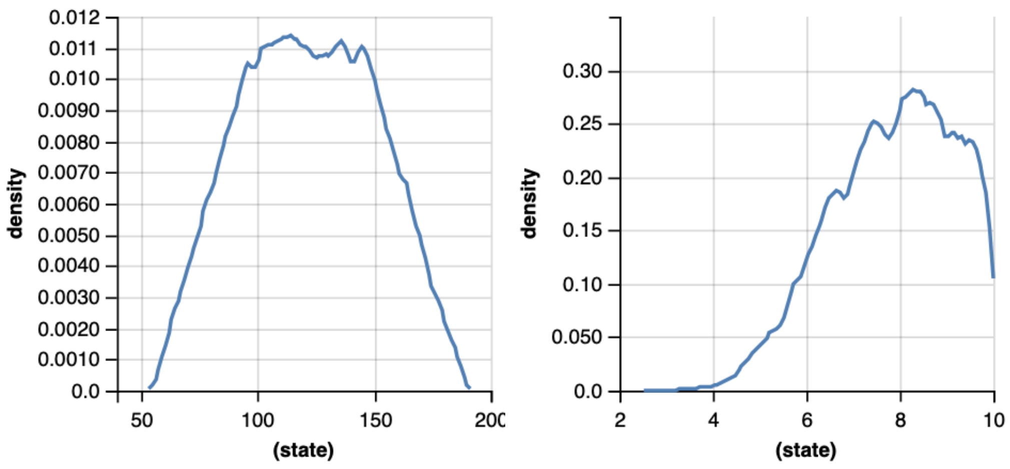
(b) [15P] Estimate the initial velocity $v(t_0)$ given that we have measured $x(t_1) = 150$. To account for noise in the measuring device, assume that the measurement itself is actually a Gaussian-distributed random variable with mean 150 and variance 10.

Hints:

- Remember that WebPPL does not support loops (neither `for` nor `while`) but it does support recursion.
- Use WebPPL's `observe` statement to condition on the value drawn from a *continuous* distribution being equal to another value.
- In this exercise you can read off your answers from the plots output by the `WebPPL` interpreter.

Solution:

```
// select part of exercise here
var exPart = 'a'

var model = function() {
    var x = gaussian(0, 1)
    var v = uniform(1, 10)
    var a = uniform(1,2)
    var t_1 = 10
    var h = 1
    var steps = t_1 / h

    var dynamics = function(x, v, steps) {
        if(steps == 0) {
            return x
        }
        var x_new = x + h * v
        var v_new = v + h * a
        return dynamics(x_new, v_new, steps - 1)
    }

    var x_t_1 = dynamics(x, v, steps)

    if(exPart == 'a'){
        return x_t_1
    }
    else if(exPart == 'b') {
        observe(Gaussian({mu: 150, sigma: 10}), x_t_1)
        return v

    }
}


viz.auto(Infer({method: 'MCMC', samples: 100000, model}));
```



Output of part (a) (left) and part (b) (right). From these plots we read off the following:

(a) The expected final position is approximately 125.

(b) A good estimate for the initial velocity is ≈ 8.