# Formal Semantics of Probabilistic Programming Languages: Issues, Results, and Opportunities

Hongseok Yang
KAIST, Korea

Based on several joint projects with Bradley Gram-hansen, Chris Heunen, Ohad Kammar, Tobias Kohn, Tom Rainforth, Sam Staton, Frank Wood, and Yuan Zhou

Formal semantics describes a mapping from probabilistic programs to probabilistic models.

Formal semantics describes a mapping from probabilistic programs to probabilistic models.

```
(let [x (sample (normal 0 1))
      y (observe (normal x 1) 2)]
  x)
```

Formal semantics describes a mapping from probabilistic programs to probabilistic models.

```
(let [x (sample (normal 0 1))
      y (observe (normal x 1) 2)]
  x)
```

Formal semantics describes a mapping from probabilistic programs to probabilistic models.

```
(let [x (sample (normal 0 1))
      y (observe (normal x 1) 2)]
  x)
```

Formal semantics describes a mapping from probabilistic programs to probabilistic models.

```
(let [x (sample (normal 0 1))
      y (observe (normal x 1) 2)]
  x)
```

Formal semantics describes a mapping from probabilistic programs to probabilistic models.

$$\left[\!\left[ \boxed{\begin{array}{l}\texttt{(let [x (sample (normal 0 1))} \\ \qquad\texttt{y (observe (normal x 1) 2)]} \\ \quad\texttt{x)}\end{array}} \right]\!\right] = p(x \mid y=2)$$

Formal semantics describes a mapping from probabilistic programs to probabilistic models.

$$\left[\!\left[ \boxed{\texttt{(let [x (sample (normal 0 1))} \quad \texttt{y (observe (normal x 1) 2)]} \quad \texttt{x)}} \right]\!\right] = p(x \mid y=2)$$

$$\left[\!\left[ \boxed{\texttt{(let [x (sample (normal 0 1))} \quad \texttt{y (observe (normal x 1) 2)]} \quad \texttt{x)}} \right]\!\right]' = p(x, y=2)$$

Why should one care about formal semantics?

Why should one care about formal semantics?

Why should one care about formal semantics?

Why should one care about formal semantics?

1. Specification of inference algorithms.

2. Compiler optimisation.

3. Detection of ill-defined models.

Why should one care about formal semantics?

1. Specification of inference algorithms.
2. Compiler optimisation.
3. Detection of ill-defined models.

```
(let [x        (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y        (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(let [x         (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y         (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(let [x        (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y        (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

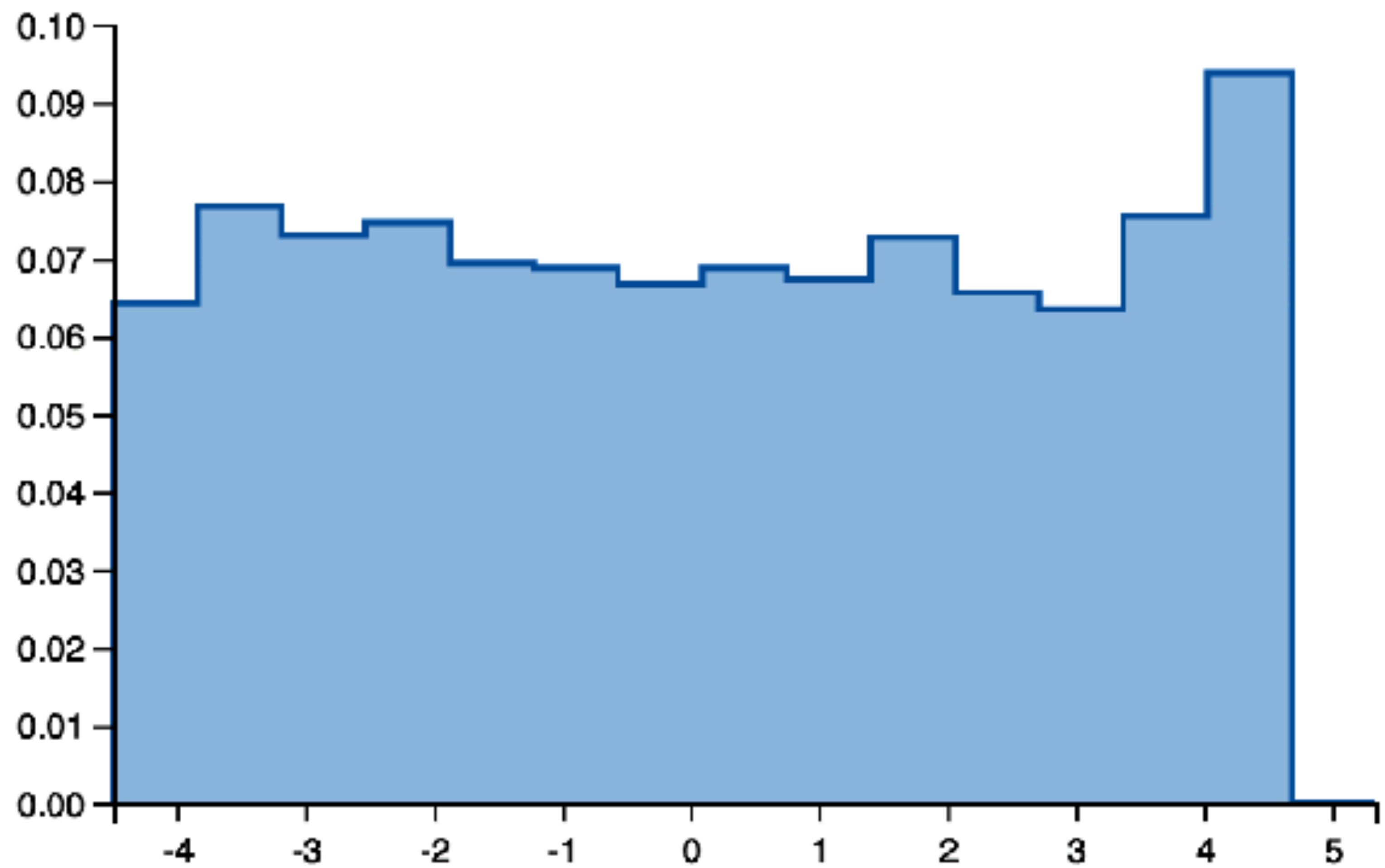
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(def lazy-samples (doquery :lmh example1 []))
(def samples (map :result (take-nth 100 (take 500000 (drop 100000 lazy-samples)))))
(plot/histogram samples :normalize :probability)
```

#'uppsala-pp17/lazy-samples

#'uppsala-pp17/samples

5K samples



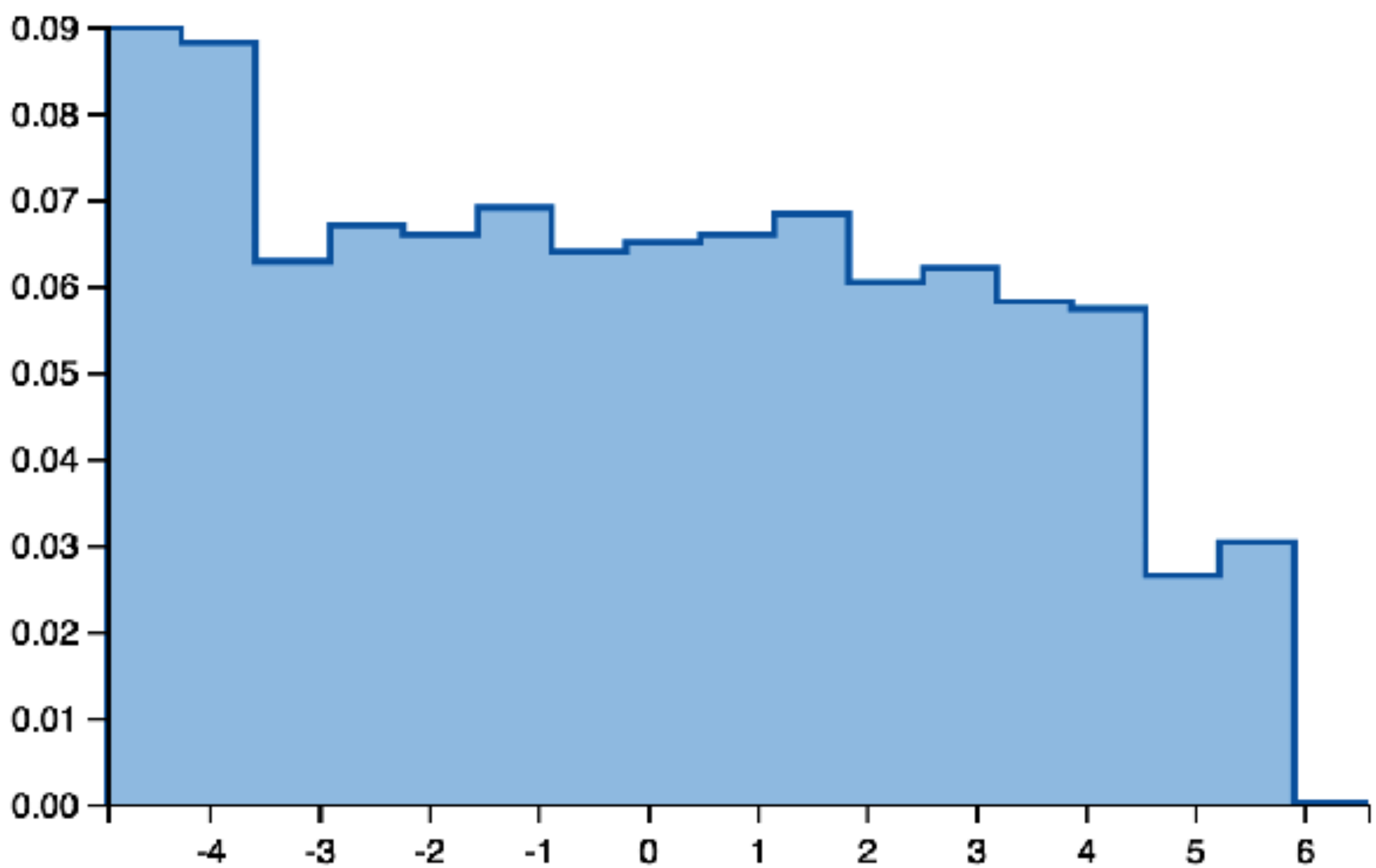
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(def lazy-samples (doquery :lmh example1 []))
(def samples (map :result (take-nth 100 (take 1000000 (drop 100000 lazy-samples)))))
(plot/histogram samples :normalize :probability)
```

#'uppsala-pp17/lazy-samples

#'uppsala-pp17/samples

10K samples



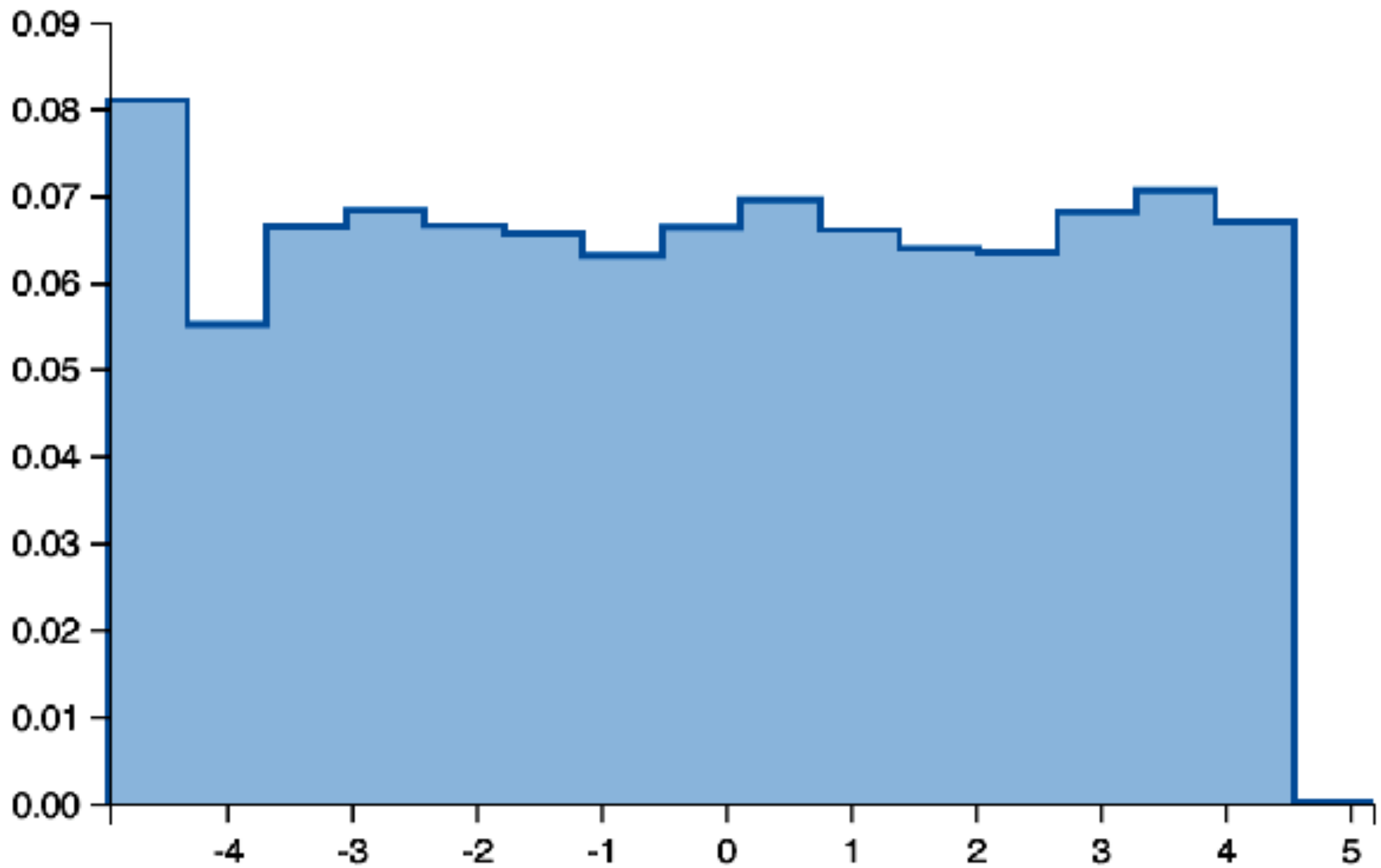
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(def lazy-samples (doquery :lmh example1 []))
(def samples (map :result (take-nth 100 (take 1500000 (drop 100000 lazy-samples)))))
(plot/histogram samples :normalize :probability)
```

```
#'uppsala-pp17/lazy-samples

#'uppsala-pp17/samples
```

15K samples



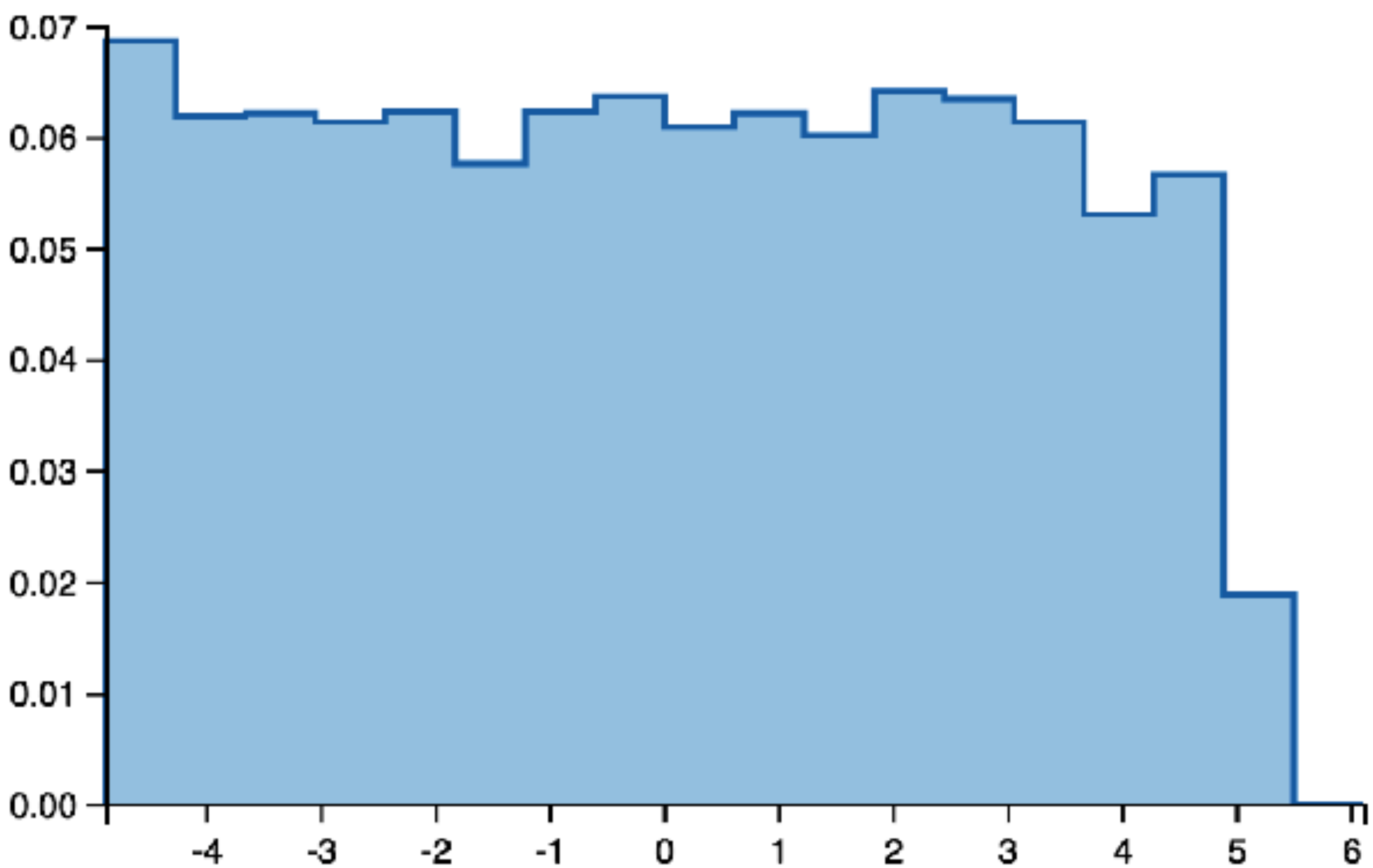
```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(def lazy-samples (doquery :lmh example1 []))
(def samples (map :result (take-nth 100 (take 2000000 (drop 100000 lazy-samples)))))
(plot/histogram samples :normalize :probability)
```

#'uppsala-pp17/lazy-samples

#'uppsala-pp17/samples

20K samples



```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

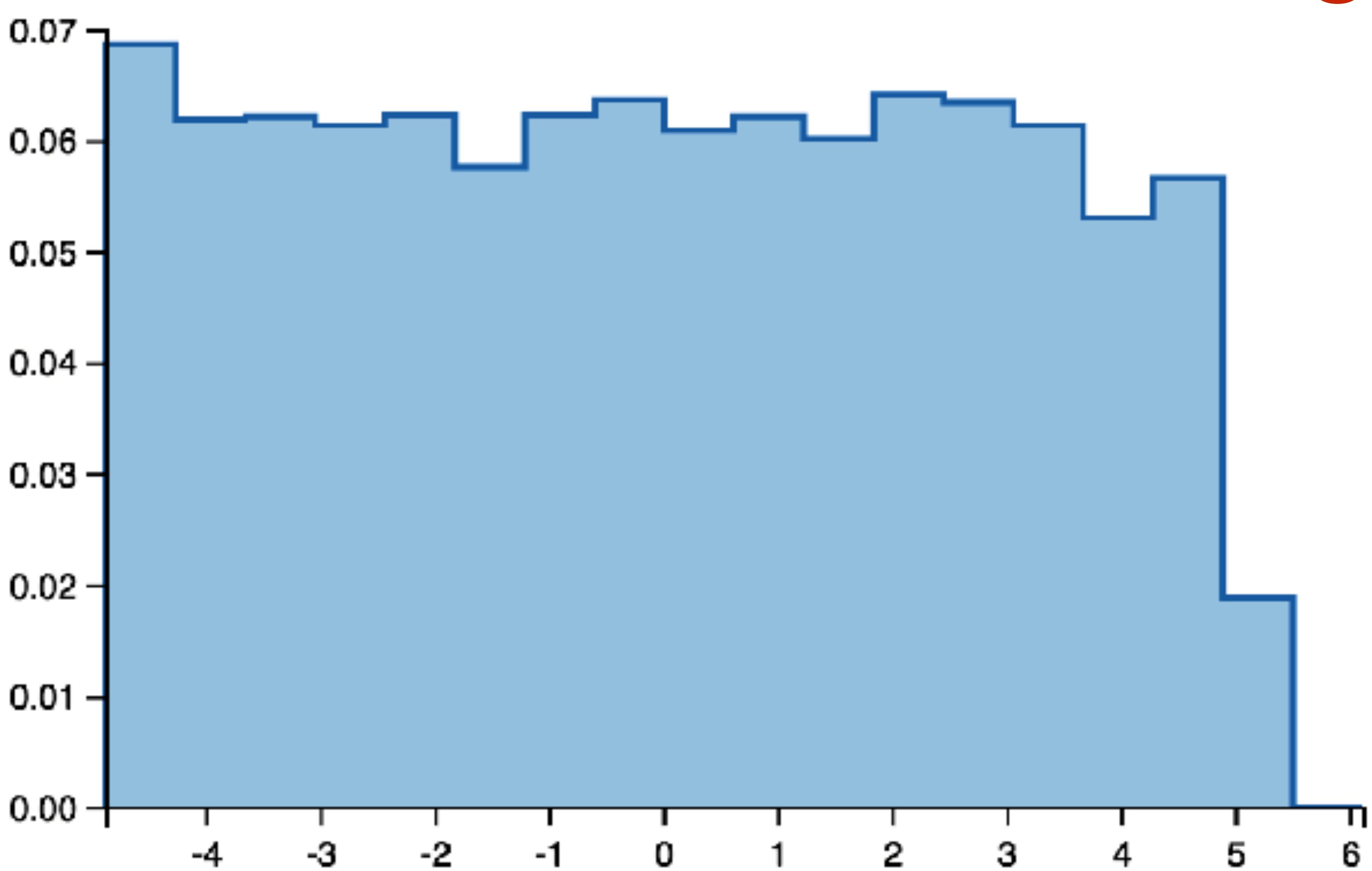
```
(def lazy-samples (doquery :lmh example1 []))
(def samples (map :result (take-nth 100 (take 2000000 (drop 100000 lazy-samples)))))
(plot/histogram samples :normalize :probability)
```

```
#'uppsala-pp17/lazy-samples

#'uppsala-pp17/samples
```

20K samples

Uniform[-6,6]?

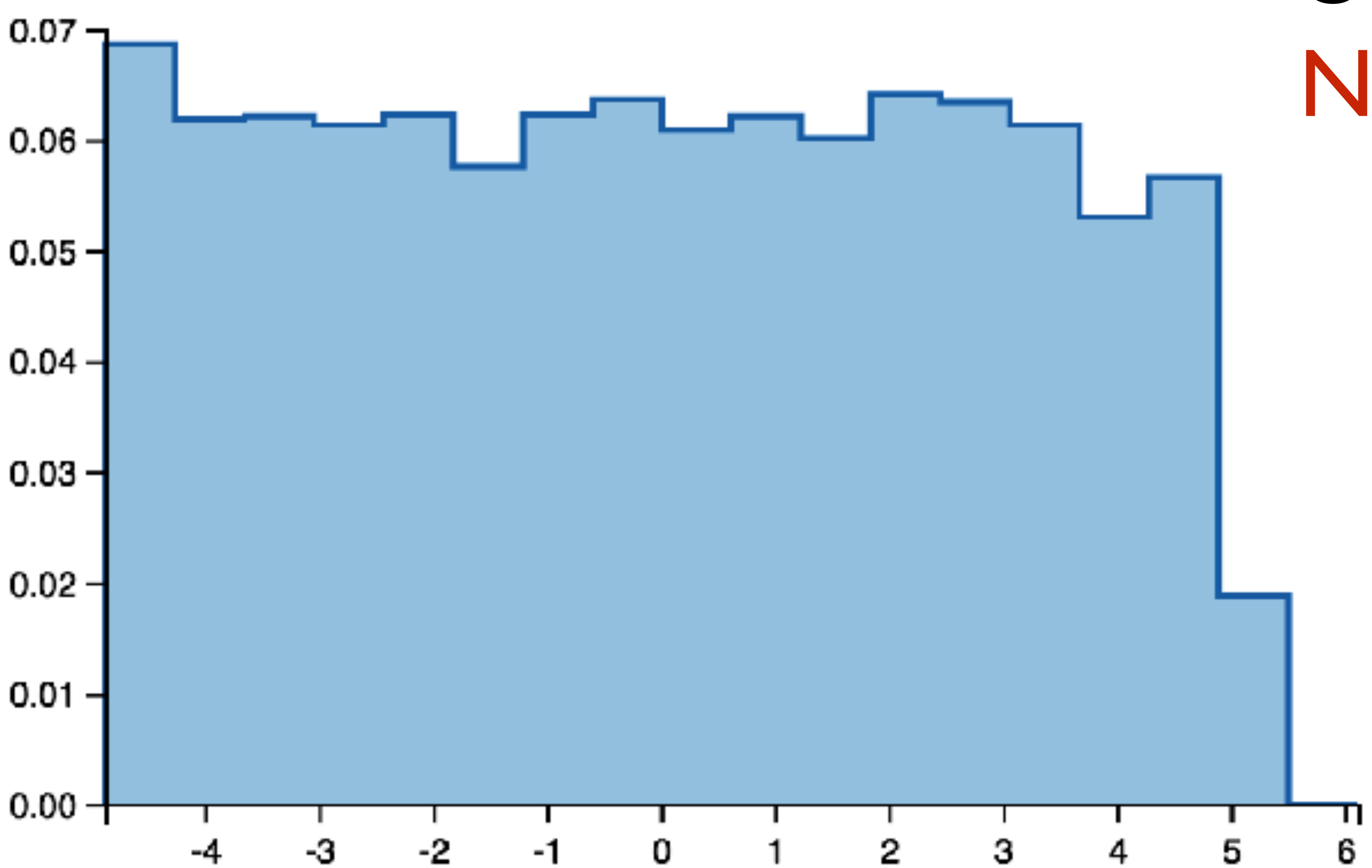


```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(def lazy-samples (doquery :lmh example1 []))
(def samples (map :result (take-nth 100 (take 2000000 (drop 100000 lazy-samples)))))
(plot/histogram samples :normalize :probability)
```

```
#'uppsala-pp17/lazy-samples

#'uppsala-pp17/samples
```




20K samples

Uniform[-6,6]?

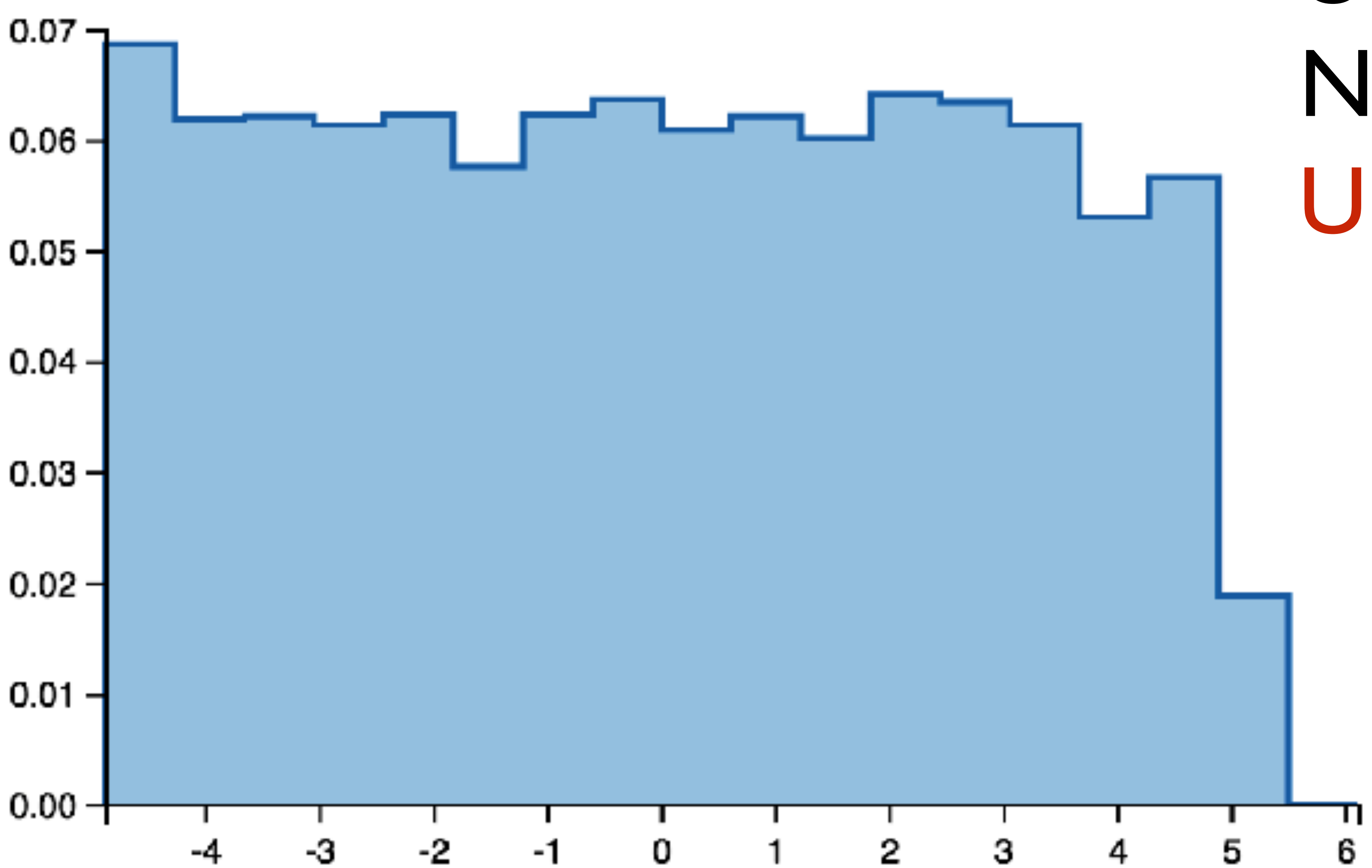
No posterior.

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

```
(def lazy-samples (doquery :lmh example1 []))
(def samples (map :result (take-nth 100 (take 2000000 (drop 100000 lazy-samples)))))
(plot/histogram samples :normalize :probability)
```

#'uppsala-pp17/lazy-samples

#'uppsala-pp17/samples




20K samples

Uniform[-6,6]?

No posterior.

Uniform[-∞,∞].

```
(let [x        (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y        (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

p(x) = normal-pdf(x;0,1)

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

p(x) = normal-pdf(x;0,1)

p(y=0|x) = exponential-pdf(y=0;1/p(x))

```
(let [x        (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y        (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

$p(x)$ = normal-pdf$(x;0,1)$

$p(y{=}0|x)$ = exponential-pdf$(y{=}0;1/p(x))$ $= 1/p(x)$

```
(let [x        (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y        (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

p(x) = normal-pdf(x;0,1)

p(y=0|x) = exponential-pdf(y=0;1/p(x)) = 1/p(x)

p(x,y=0) = p(x) * p(y=0|x)

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

p(x) = normal-pdf(x;0,1)

p(y=0|x) = exponential-pdf(y=0;1/p(x)) = 1/p(x)

p(x,y=0) = p(x) * p(y=0|x) = p(x) * 1/p(x) = 1

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf)) 0)]
  x)
```

p(x) = normal-pdf(x;0,1)

p(y=0|x) = exponential-pdf(y=0;1/p(x)) = 1/p(x)

p(x,y=0) = p(x) * p(y=0|x) = p(x) * 1/p(x) = 1

p(y=0) = ∫ p(x,y=0)dx = ∫ dx = ∞

$$\left[\!\left[\;\boxed{\begin{array}{l}\texttt{(let [x}\quad\texttt{(sample (normal 0 1))}\\ \quad\texttt{x-pdf}\quad\texttt{(normal-pdf x 0 1)}\\ \quad\texttt{y}\quad\texttt{(observe (exponential (/ 1 x-pdf)) 0)]}\\ \;\texttt{x)}\end{array}}\;\right]\!\right]$$

$= p(x, y=0) = \mathrm{Lebesgue}(x)$

# Issues and results

# Issue 1:
# Normalised posterior or unnormalized posterior?

$\llbracket$

```
(let [x        (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y        (observe (exponential (/ 1 x-pdf) 0))]
  x)
```

$\rrbracket$

$= p(x, y=0) = \mathrm{Lebesgue}(x)$

$$
\left[\!\left[
\boxed{
\begin{array}{l}
\texttt{(let [x \quad (sample (normal 0 1))} \\
\texttt{\qquad x-pdf \quad (normal-pdf x 0 1)} \\
\texttt{\qquad y \quad (observe (exponential (/ 1 x-pdf) 0))]} \\
\texttt{\ \ x)}
\end{array}
}
\right]\!\right]
$$

= p(x, y=0) = Lebesgue(x)

$$
\left[\!\left[
\boxed{
\begin{array}{l}
\texttt{(let [x \quad (sample (normal 0 1))} \\
\texttt{\qquad x-pdf \quad (normal-pdf x 0 1)} \\
\texttt{\qquad y \quad (observe (exponential (/ 1 x-pdf) 0))]} \\
\texttt{\ \ x)}
\end{array}
}
\right]\!\right]'
$$

= p(x|y=0) = undefined

⟦
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
⟧

= p(x, y=0) = Lebesgue(x)

⟦
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
⟧’

= p(x|y=0) = undefined

```
(let [x'     • • •


      • • •                          ]
  x')
```

⟦
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
⟧

= p(x, y=0) = Lebesgue(x)

⟦
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
⟧'

= p(x|y=0) = undefined

```
(let [x' (let [x      (sample (normal 0 1))
               x-pdf  (normal-pdf x 0 1)
               y      (observe ...)] x)
    • • •                            ]
  x')
```

$[\![$

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf) 0))]
  x)
```

$]\!]$

= p(x, y=0) = Lebesgue(x)

$[\![$

```
(let [x       (sample (normal 0 1))
      x-pdf   (normal-pdf x 0 1)
      y       (observe (exponential (/ 1 x-pdf) 0))]
  x)
```

$]\!]'$

= p(x|y=0) = undefined

```
(let [x'  (let [x       (sample (normal 0 1))
                x-pdf   (normal-pdf x 0 1)
                y       (observe ...)]  x)
      y'  (observe (normal x' 1) 0)]
  x')
```

$\llbracket$
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
$\rrbracket$

= p(x, y=0) = Lebesgue(x)

$\llbracket$
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
$\rrbracket'$

= p(x|y=0) = undefined

$\llbracket$
```
(let [x'  (let [x      (sample (normal 0 1))
                x-pdf  (normal-pdf x 0 1)
                y      (observe ...)]  x)
      y'  (observe (normal x' 1) 0)]
  x')
```
$\rrbracket'$

= undefined

⟦

```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```

⟧

$= p(x, y{=}0) = \text{Lebesgue}(x)$

⟦

```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```

⟧'

$= p(x|y{=}0) = \text{undefined}$

⟦

```
(let [x'  (let [x      (sample (normal 0 1))
                x-pdf  (normal-pdf x 0 1)
                y      (observe ...)]  x)
      y'  (observe (normal x' 1) 0)]
  x')
```

⟧'

$= \text{undefined} \neq p(x'|y{=}0,y'{=}0) = \text{normal-pdf}(x';0,1)$

⟦
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
⟧

= p(x, y=0) = Lebesgue(x)

⟦
```
(let [x      (sample (normal 0 1))
      x-pdf  (normal-pdf x 0 1)
      y      (observe (exponential (/ 1 x-pdf) 0))]
  x)
```
⟧'

= p(x|y=0) = undefined

⟦
```
(let [x'  (let [x      (sample (normal 0 1))
                x-pdf  (normal-pdf x 0 1)
                y      (observe ...)]  x)
      y'  (observe (normal x' 1) 0)]
  x')
```
⟧

= p(x, y=0, y'=0) = normal-pdf(x;0,1)

In general, semantics interprets programs as unnormalized conditional distributions [Staton17].

In general, semantics interprets programs as unnormalized conditional distributions [Staton17].

$$\left[\!\left[\ \boxed{\begin{array}{l}\texttt{(let [x (sample (normal z 1))} \\ \quad\texttt{x-pdf (normal-pdf x 0 1)} \\ \quad\texttt{y (observe (exponential (/ 1 x-pdf) 0))]} \\ \ \texttt{x)}\end{array}}\ \right]\!\right]$$

$= p(x, y{=}0 \mid z)$

In general, semantics interprets programs as unnormalized conditional distributions [Staton17].

$$\left[\!\left[ \boxed{\begin{array}{l}\texttt{(let [x (sample (normal z 1))} \\ \quad\texttt{x-pdf (normal-pdf x 0 1)} \\ \quad\texttt{y (observe (exponential (/ 1 x-pdf) 0))]} \\ \texttt{ x)}\end{array}} \right]\!\right]$$

$= p(x, y{=}0 \mid z)$

Always s-finite unnormalised cond. distributions (also called s-finite kernels) [Staton17].

# Issue 2:
# Should marginalise or not?

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y   (observe (normal x2 1) -1)]
 x1)
```

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y   (observe (normal x2 1) -1)]
 x1)
```

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y   (observe (normal x2 1) -1)]
 x1)
```

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y   (observe (normal x2 1) -1)]
 x1)
```

$x_1$

$z_2$

$y$

$[\![$

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

$]\!]$

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y=-1)$

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y{=}{-1})$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1};\ \text{if } (x_1{>}0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y{=}{-1})$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1}; \text{if } (x_1{>}0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧'

$= p(x_1, y{=}{-1}) = \int p(x_1, z_2, y{=}{-1})\, dz_2$

⟦

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

⟧

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y=-1)$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y=-1; \text{if } (x_1>0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

⟦

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

⟧'

$= p(x_1, y=-1) = \int p(x_1, z_2, y=-1)\, dz_2$

$= f(x_1;0,1) * [x_1>0] * \cdots$

$+ f(x_1;0,1) * [x_1\leq 0] * \cdots$

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y=-1)$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y=-1; \text{if } (x_1>0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧'

$= p(x_1, y=-1) = \int p(x_1, z_2, y=-1)\, dz_2$

$= f(x_1;0,1) * [x_1>0] * f(y=-1; 2, 1)$

$\quad + f(x_1;0,1) * [x_1 \le 0] * \ldots$

$[\![$

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

$]\!]$

$x_1$
$z_2$
$y$

$= p(x_1, z_2, y{=}{-1})$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1}; \text{if } (x_1{>}0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

$[\![$

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

$]\!]'$

$= p(x_1, y{=}{-1}) = \int p(x_1, z_2, y{=}{-1})\, dz_2$

$= f(x_1;0,1) * [x_1{>}0] * f(y{=}{-1}; 2, 1)$

$\quad + f(x_1;0,1) * [x_1 \leq 0] * \int f(z_2;-2,1) * f(y{=}{-1}; z_2, 1)\, dz_2$

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y{=}{-1})$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1};\text{ if } (x_1{>}0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧'

$= p(x_1, y{=}{-1}) = \int p(x_1, z_2, y{=}{-1})\, dz_2$

$= f(x_1;0,1) * [x_1{>}0] * f(y{=}{-1}; 2, 1)$

$\quad + f(x_1;0,1) * [x_1{\leq}0] * \int f(z_2;-2,1) * f(y{=}{-1}; z_2, 1)\, dz_2$

$[\![$
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
$]\!]$

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y{=}{-1})$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1};\ \text{if } (x_1{>}0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma) = \text{normal-pdf}(x;\mu,\sigma)$.

**Graphical model**

$[\![$
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
$]\!]'$

$= p(x_1, y{=}{-1}) = \int p(x_1, z_2, y{=}{-1})\, dz_2$

$= f(x_1;0,1) * [x_1{>}0] * f(y{=}{-1}; 2, 1)$

$\quad + f(x_1;0,1) * [x_1{\leq}0] * \int f(z_2;-2,1) * f(y{=}{-1}; z_2, 1)\, dz_2$

⟦

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

⟧

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y{=}{-1})$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1}; \text{if } (x_1{>}0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

$\boxed{\text{Graphical model}}$

⟦

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

⟧'

$= p(x_1, y{=}{-1}) = \int p(x_1, z_2, y{=}{-1})\, dz_2$

$= f(x_1;0,1) * [x_1{>}0] * f(y{=}{-1}; 2, 1)$

$\quad + f(x_1;0,1) * [x_1{\le}0] * \int f(z_2;-2,1) * f(y{=}{-1}; z_2, 1)\, dz_2$

$\boxed{\text{Model on execution traces}}$

⟦

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

⟧

$x_1$

$z_2$

$y$

$= p(x_1, z_2, y{=}{-1})$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1}; \text{if } (x_1{>}0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-pdf$(x;\mu,\sigma)$.

Graphical model

⟦

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

⟧'

$= p(x_1, y{=}{-1}) = \int p(x_1, z_2, y{=}{-1})\, dz_2$

$= f(x_1;0,1) * [x_1{>}0] * f(y{=}{-1}; 2, 1)$

$+ f(x_1;0,1) * [x_1 \leq 0] * \int f(z_2;-2,1) * f(y{=}{-1}; z_2, 1)\, dz_2$

Model on execution traces

Semantics without marginalisation.

1. Converts prob. progs to graphical models.
2. Basis for graph-based inference algorithm.

Semantics with marginalisation.

1. Defines models based on prog. execution.
2. Basis for evaluation-based inference algo.

# Issue 3:
# Non-differentiability

$[\![$

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y   (observe (normal x2 1) -1)]
 x1)
```

$]\!]$

$x_1$

$z_2$

$y$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y=-1; \text{if } (x_1>0) \text{ then } 2 \text{ else } z_2,1)$

Here $f(x;\mu,\sigma) = \text{normal-prob}(x;\mu,\sigma)$.

$[\![$

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

$]\!]$

$x_1$ $z_2$ $y$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}-1;$ if $(x_1>0)$ then $2$ else $z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-prob$(x;\mu,\sigma)$.

Non-differentiable probability density.

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```
⟧

$x_1$

$z_2$

$y$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y=-1;\ \text{if } (x_1>0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-prob$(x;\mu,\sigma)$.

Non-differentiable probability density.

But non-differentiable at a Lebesgue-measure-0 set.

$[\![$

```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y  (observe (normal x2 1) -1)]
 x1)
```

$]\!]$

$x_1$ $z_2$ $y$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y=-1; \text{if } (x_1>0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-prob$(x;\mu,\sigma)$.

Non-differentiable probability density.

But non-differentiable at a Lebesgue-measure-0 set.

[Q] Always measure-0?

⟦
```
(let [x1 (sample (normal 0 1))
      x2 (if (> x1 0) 2 (sample (normal -2 1)))
      y   (observe (normal x2 1) -1)]
 x1)
```
⟧

$x_1$ $z_2$ $y$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y=-1;\ \text{if } (x_1>0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma)$ = normal-prob$(x;\mu,\sigma)$.

Non-differentiable probability density.

But non-differentiable at a Lebesgue-measure-0 set.

[Q] Always measure-0?

[Partial A] If all primitive operations are analytic [AISTATS'19].

(g x1)

```
⟦ (let [x1 (sample (normal 0 1))
        x2 (if (> x1 0) 2 (sample (normal -2 1)))
        y   (observe (normal x2 1) -1)]
    x1) ⟧
```

(h x2)

$x_1$ $z_2$ $y$

$= f(x_1;0,1) * f(z_2;-2,1) * f(y{=}{-1};\ \text{if } (x_1>0) \text{ then } 2 \text{ else } z_2, 1)$

Here $f(x;\mu,\sigma) = \text{normal-prob}(x;\mu,\sigma)$.

Non-differentiable probability density.

But non-differentiable at a Lebesgue-measure-0 set.

[Q] Always measure-0?

[Partial A] If all primitive operations are analytic [AISTATS'19].

# Issue 4:
# Higher-order functions

# Linear regression

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

    [s b])
```

# Linear regression

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

    [s b])
```

# Linear regression

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

   (observe (normal (f 0) .5) .6)
   (observe (normal (f 1) .5) .7)
   (observe (normal (f 2) .5) 1.2)
   (observe (normal (f 3) .5) 3.2)
   (observe (normal (f 4) .5) 6.8)
   (observe (normal (f 5) .5) 8.2)
   (observe (normal (f 6) .5) 8.4)

   [s b])
```

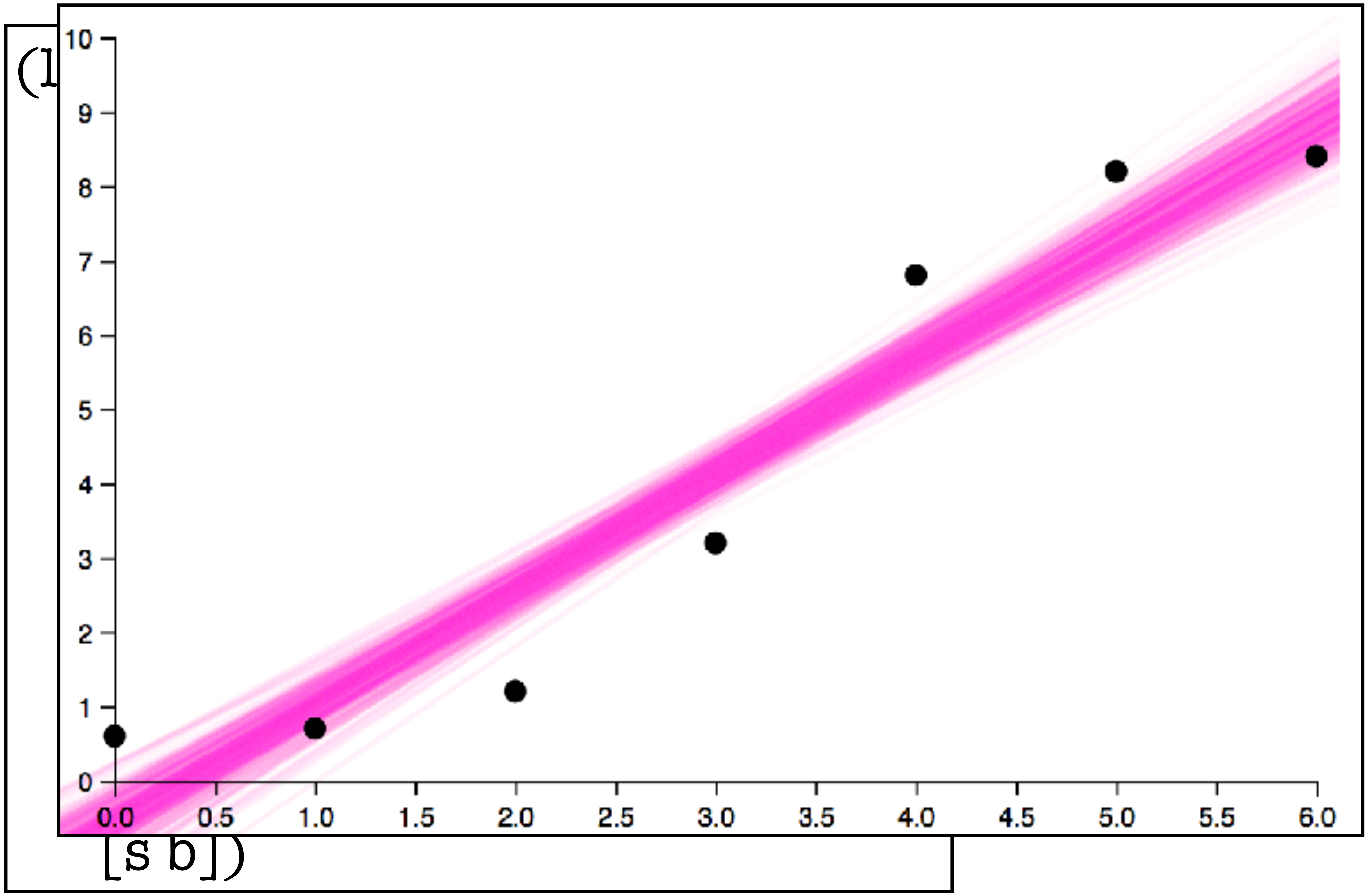
# Linear regression

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

    [s b])
```

# Linear regression



# Linear regression

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

    (observe (normal (f 0) .5) .6)
    (observe (normal (f 1) .5) .7)
    (observe (normal (f 2) .5) 1.2)
    (observe (normal (f 3) .5) 3.2)
    (observe (normal (f 4) .5) 6.8)
    (observe (normal (f 5) .5) 8.2)
    (observe (normal (f 6) .5) 8.4)

    [s b])
```

# Linear regression

$$\left[\!\left[ \quad \right]\!\right] \in \text{Measure}(\mathbb{R}^2)$$

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

   (observe (normal (f 0) .5) .6)
   (observe (normal (f 1) .5) .7)
   (observe (normal (f 2) .5) 1.2)
   (observe (normal (f 3) .5) 3.2)
   (observe (normal (f 4) .5) 6.8)
   (observe (normal (f 5) .5) 8.2)
   (observe (normal (f 6) .5) 8.4)

   [s b])
```

# Linear regression

$\llbracket$

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

   (observe (normal (f 0) .5) .6)
   (observe (normal (f 1) .5) .7)
   (observe (normal (f 2) .5) 1.2)
   (observe (normal (f 3) .5) 3.2)
   (observe (normal (f 4) .5) 6.8)
   (observe (normal (f 5) .5) 8.2)
   (observe (normal (f 6) .5) 8.4)
   f)
```

~~[s b])~~

$\rrbracket \in \mathrm{Measure}(\mathbb{R}^2)$

# Linear regression

$[\![$

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

  (observe (normal (f 0) .5) .6)
  (observe (normal (f 1) .5) .7)
  (observe (normal (f 2) .5) 1.2)
  (observe (normal (f 3) .5) 3.2)
  (observe (normal (f 4) .5) 6.8)
  (observe (normal (f 5) .5) 8.2)
  (observe (normal (f 6) .5) 8.4)
  f)
```

~~[s b])~~

$]\!]$

$\text{Measure}(\mathbb{R} \to \mathbb{R})$

$\in \cancel{\text{Measure}(\mathbb{R}^2)}$

# Linear regression

Troublemaker
in measure theory

⟦

```
(let [s  (sample (normal 0 2))
      b (sample (normal 0 6))
      f  (fn [x] (+ (* s x) b))]

   (observe (normal (f 0) .5) .6)
   (observe (normal (f 1) .5) .7)
   (observe (normal (f 2) .5) 1.2)
   (observe (normal (f 3) .5) 3.2)
   (observe (normal (f 4) .5) 6.8)
   (observe (normal (f 5) .5) 8.2)
   (observe (normal (f 6) .5) 8.4)
   f)
```

~~[s b])~~

⟧

$\text{Measure}(\mathbb{R} \to \mathbb{R})$

$\in \cancel{\text{Measure}(\mathbb{R}^2)}$

# Measure-theoretic issue

Measure theory provides a foundation for modern probability theory.

But it doesn't support higher-order fns well.

$$ev : (\mathbb{R} \to_m \mathbb{R}) \times \mathbb{R} \to \mathbb{R}, \qquad ev(f,x) = f(x).$$

[Aumann 61] ev is not measurable no matter which $\sigma$-algebra is used for $\mathbb{R} \to_m \mathbb{R}$.

We formulated a new probability theory that puts random variable as primary concept [LICS'17].

Used it to define the semantics of expressive prob. programming languages, such as Anglican.

Quasi-Borel space - core notion of this theory.

We formulated a new probability theory that puts random variable as primary concept [LICS'17].

Used it to define the semantics of expressive prob. programming languages, such as Anglican.

Quasi-Borel space - core notion of this theory.

# Random variable α in X

# Random variable $\alpha$ in X

$$\alpha : \Omega \to X$$

- X - set of values.
- $\Omega$ - set of random seeds.
- Random seed generator.

# Random variable α in X
# in measure theory

$$\alpha : \Omega \to X$$

- X - set of values.
- Ω - set of random seeds.
- Random seed generator.

# Random variable α in X
# in measure theory

$$\alpha : \Omega \rightarrow X$$

- X - set of values.
- Ω - set of random seeds.
- Random seed generator.

1. $\Sigma \subseteq 2^{\Omega}$, $\Theta \subseteq 2^{X}$

# Random variable α in X
# in measure theory

$$\alpha : \Omega \to X$$

- X - set of values.
- Ω - set of random seeds.
- Random seed generator.

1. $\Sigma \subseteq 2^{\Omega}$, $\Theta \subseteq 2^{X}$
2. $\mu : \Sigma \to [0,1]$

# Random variable α in X
# in measure theory

$\alpha : \Omega \to X$ is a random variable
if $\alpha^{-1}(A) \in \Sigma$ for all $A \in \Theta$

- X - set of values.
- Ω - set of random seeds.
- Random seed generator.

1. $\Sigma \subseteq 2^{\Omega}$, $\Theta \subseteq 2^{X}$
2. $\mu : \Sigma \to [0,1]$

# Random variable $\alpha$ in X

$$\alpha : \Omega \to X$$

- X - set of values.
- $\Omega$ - set of random seeds.
- Random seed generator.

# Random variable α in X in quasi-Borel spaces

$$\alpha : \Omega \to X$$

- X - set of values.
- Ω - set of random seeds.
- Random seed generator.

# Random variable α in X in quasi-Borel spaces

$$\alpha : [0,1] \to X$$

- X - set of values.
- [0,1] - set of random seeds.
- Random seed generato

1. [0,1] random source
2. Borel subsets $\mathfrak{B} \subseteq 2^{[0,1]}$

# Random variable α in X in quasi-Borel spaces

$$\alpha : [0,1] \to X$$

- X - set of values.
- [0,1] - set of random seeds.
- Random seed generato

1. [0,1] random source
2. Borel subsets $\mathfrak{B} \subseteq 2^{[0,1]}$
3. Uniform[0,1]

# Random variable α in X
# in quasi-Borel spaces

$$\alpha : [0,1] \to X$$

- X - set of values.
- [0,1] - set of random seeds.
- Random seed generato

1. [0,1] random source
2. Borel subsets $\mathfrak{B} \subseteq 2^{[0,1]}$
3. Uniform[0,1]
4. $M \subseteq [[0,1] \to X]$

# Random variable α in X
## in quasi-Borel spaces

$\alpha : [0,1] \rightarrow X$ is a random variable
if $\alpha \in M$

- X - set of values.
- [0,1] - set of random seeds.
- Random seed generato

1. [0,1] random source
2. Borel subsets $\mathfrak{B} \subseteq 2^{[0,1]}$
3. Uniform[0,1]
4. $M \subseteq [[0,1] \rightarrow X]$

- Measure theory:
  - Measurable space $(X, \Theta \subseteq 2^X)$.
  - Random variable is an induced concept.
- Quasi-Borel space:
  - Quasi-Borel space $(X, M \subseteq [[0,1] \to X])$.
  - M is the set of random variables.

[Theorem] The category of measurable spaces is not cartesian closed.

[Theorem] The category of quasi-Borel spaces is cartesian closed.

Intuitively, cartesian closure means good support for higher-order functions.

Compositional inference algo. by Scibior et al. are justified using quasi-Borel spaces [POPL18].

# Issues

1. Unnormalised or normalized posterior?
2. Marginalised or un-marginalised?
3. Measure-0 non-differentiabilities? Maybe.
4. Focus on measurable sets or random vars?

1. Commutative semantics for probabilistic programs. Staton. ESOP'17.
2. A convenient category for higher-order probability theory. Heunen et al. LICS'17.
3. A domain theory for statistical probabilistic programming. Vakar et al. POPL'19.
4. LF-PPL: A low-level first order probabilistic programming language for non-differentiable models. Zhou et al. AISTATS'19.
5. An introduction to probabilistic programming. Van de Meent et al. 2019. Draft book.