Probabilistic Programming Languages

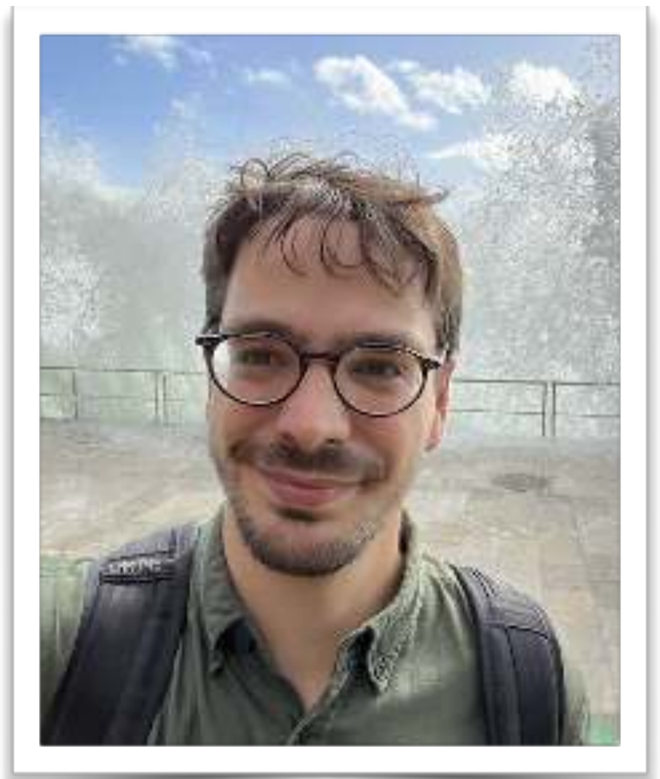
Guillaume Baudart, Hugo Paquet, Xavier Rival, Christine Tasson

MPRI 2025-2026

MPRI PPL

Course content
- Language design (probabilistic construction, static analysis, compilation)
- Inference methods (Monte Carlo methods, symbolic inference, variational inference)
- Semantics (measure theory, quasi-Borel spaces, integrable cones)
- Static analysis of inference algorithms (variational inference)

Evaluation: max(exam, (project + exam) / 2)
- Article-based project
- Final exam



Guillaume Baudart
Inria / IRIF

Hugo Paquet
Inria / DIENS

Xavier Rival
Inria / DIENS

Christine Tasson
ISAE Supaero

https://github.com/mpri-probprog/probprog-25-26

Introduction

MPRI-PPL

Probabilistic programming languages

General purpose programming languages extended with probabilistic constructs

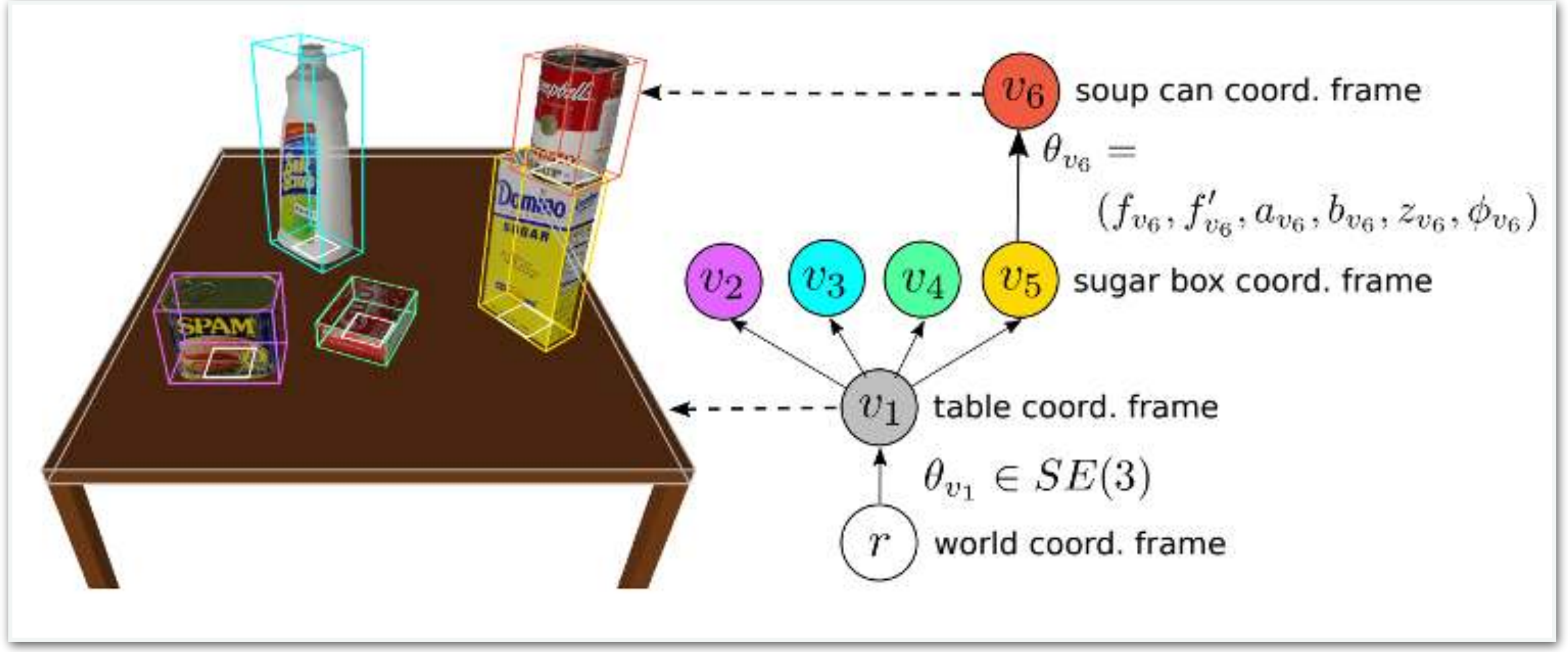
- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on inputs (e.g., observed data)
- `infer`: compute the posterior distribution of a model given the inputs

Multiple examples:

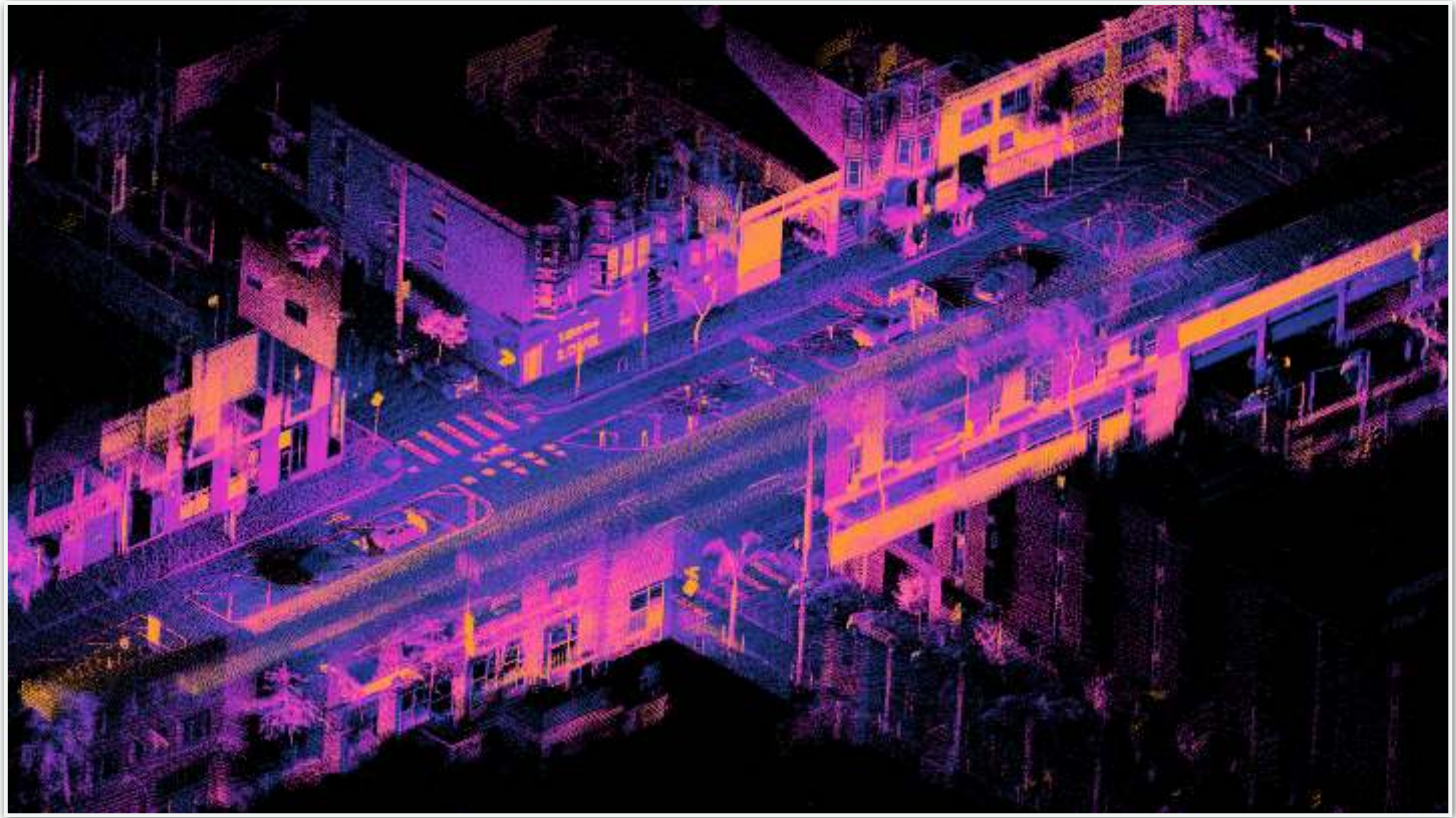
- Church, Anglican (lisp, clojure), 2008
- WebPPL (javascript), 2014
- Pyro/NumPyro (python), 2017/2019
- Gen (julia), 2018
- ProbZelus (Zelus), 2019
- ...

More and more, incorporating new ideas:

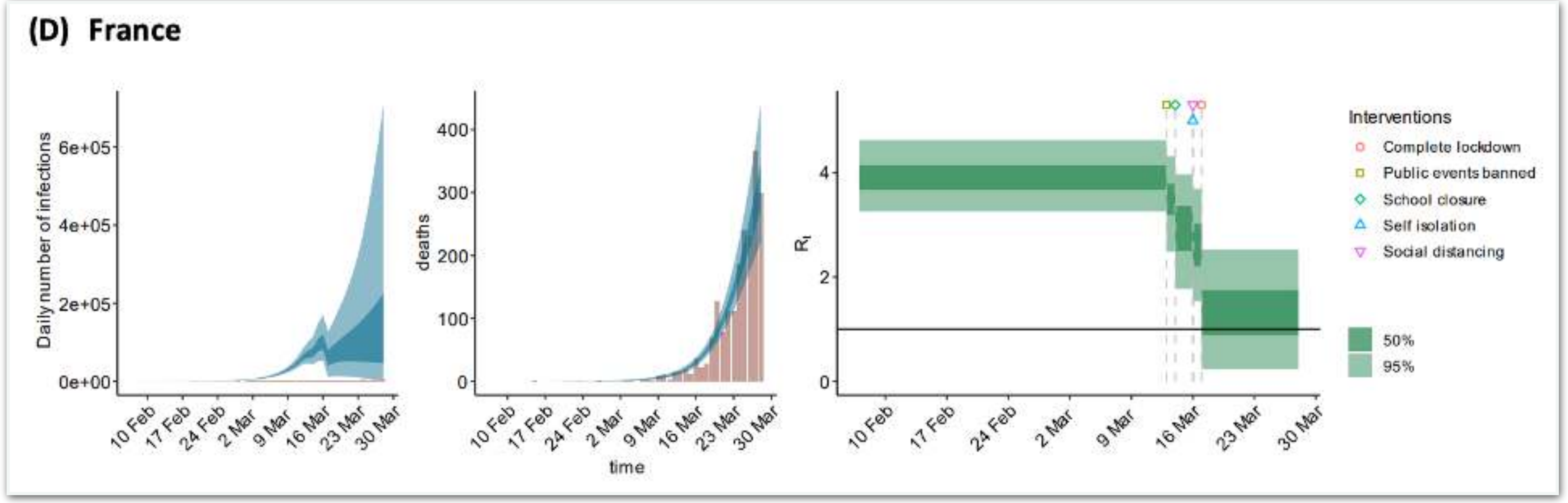
- New inference techniques, e.g., stochastic variational inference (SVI)
- Interaction with neural nets (deep probabilistic programming)



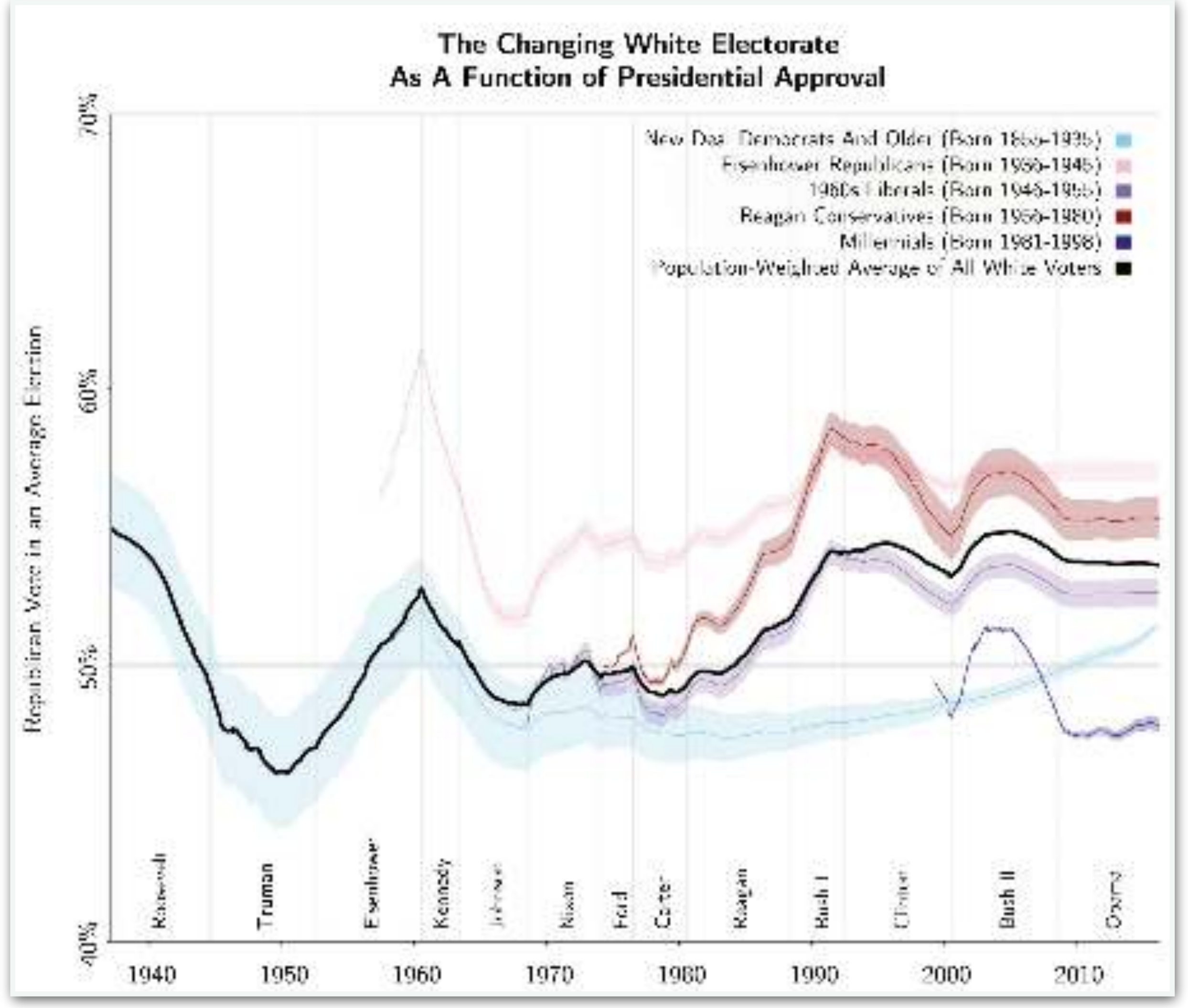
Gothoskar et al. *3dp3: 3d scene perception via probabilistic programming*
NeurIPS 2021



SLAM: Simultaneous localization and mapping
https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping#/media/File:Ouster_OS1-64_lidar_point_cloud_of_intersection_of_Folsom_and_Dore_St,_San_Francisco.png



Report 13: Estimating the number of infections and the impact of non-pharmaceutical interventions on COVID-19 in 11 European countries, Imperial College COVID-19 Response Team, Mach 2020



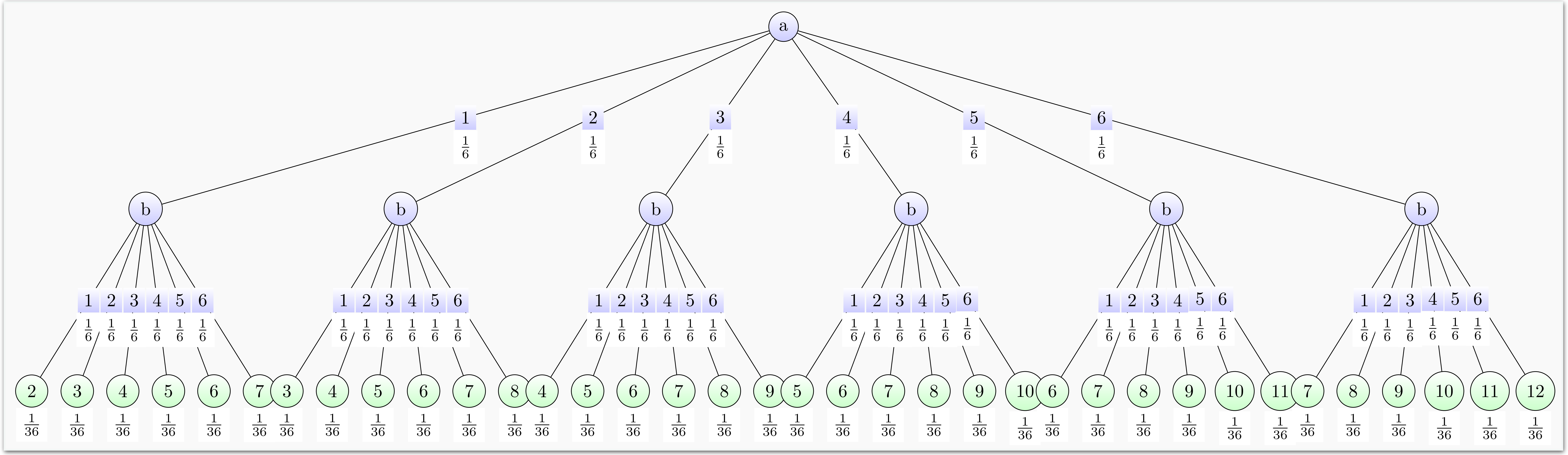
Ghitza et al. The Great Society, Reagan's Revolution, and Generations of Presidential Voting AJPS 2022

Examples: mu-ppl

What is this?

```
def dice() → int:
  a = sample(RandInt(1, 6), name="a")
  b = sample(RandInt(1, 6), name="b")
  return a + b
```

The sum of the values output by the two independent fair dice.



Credit: C. Tasson

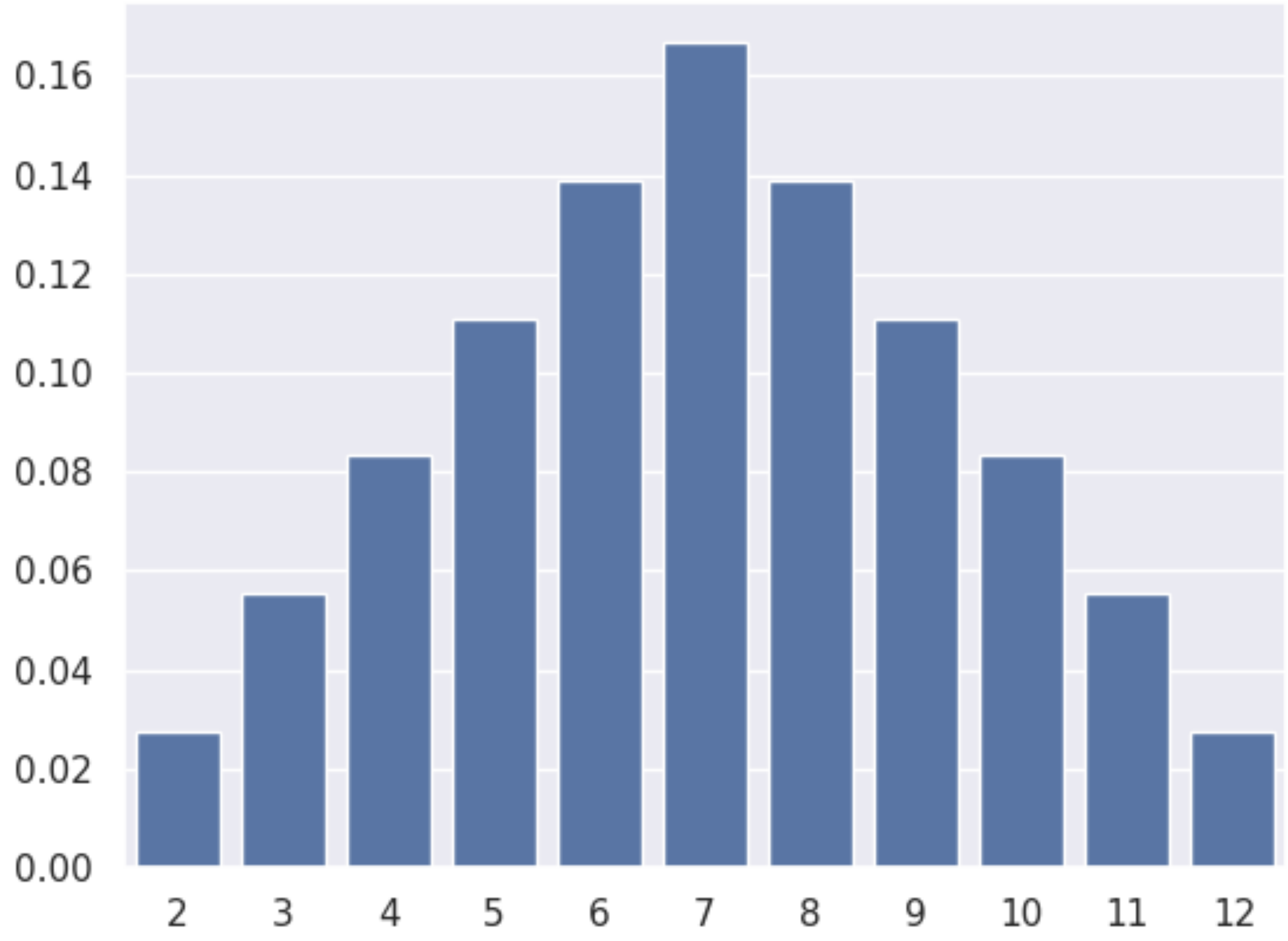
What is this?

```
def dice() → int:
  a = sample(RandInt(1, 6), name="a")
  b = sample(RandInt(1, 6), name="b")
  return a + b

with Enumeration():
  dist: Categorical[int] = infer(dice)
  viz(dice)
```

The sum of the values output by the two independent fair dice.

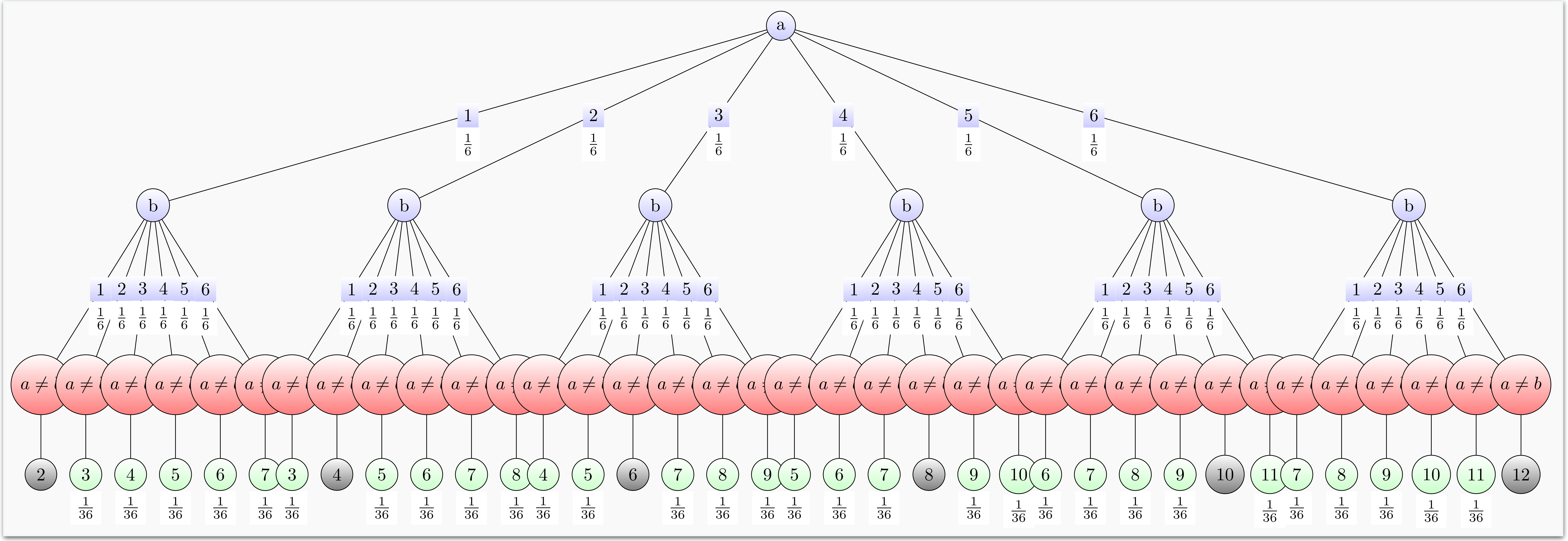
$$
\begin{aligned}
[\![\text{dice}]\!] &:= \mathbb{P}(\text{dice}() = k) \\
&= \sum_{a=1}^{6} \sum_{b=1}^{6} \frac{1}{36} \mathbb{1}_{\{a+b=k\}}
\end{aligned}
$$



What is this?

```python
def hard_dice() → int:
  a = sample(RandInt(1, 6), name=”a”)
  b = sample(RandInt(1, 6), name=”b”)
  assume (a ≠ b)
  return a + b
```

The sum of two independent fair dice, given that the output value are distinct.



Credit: C. Tasson

What is this?

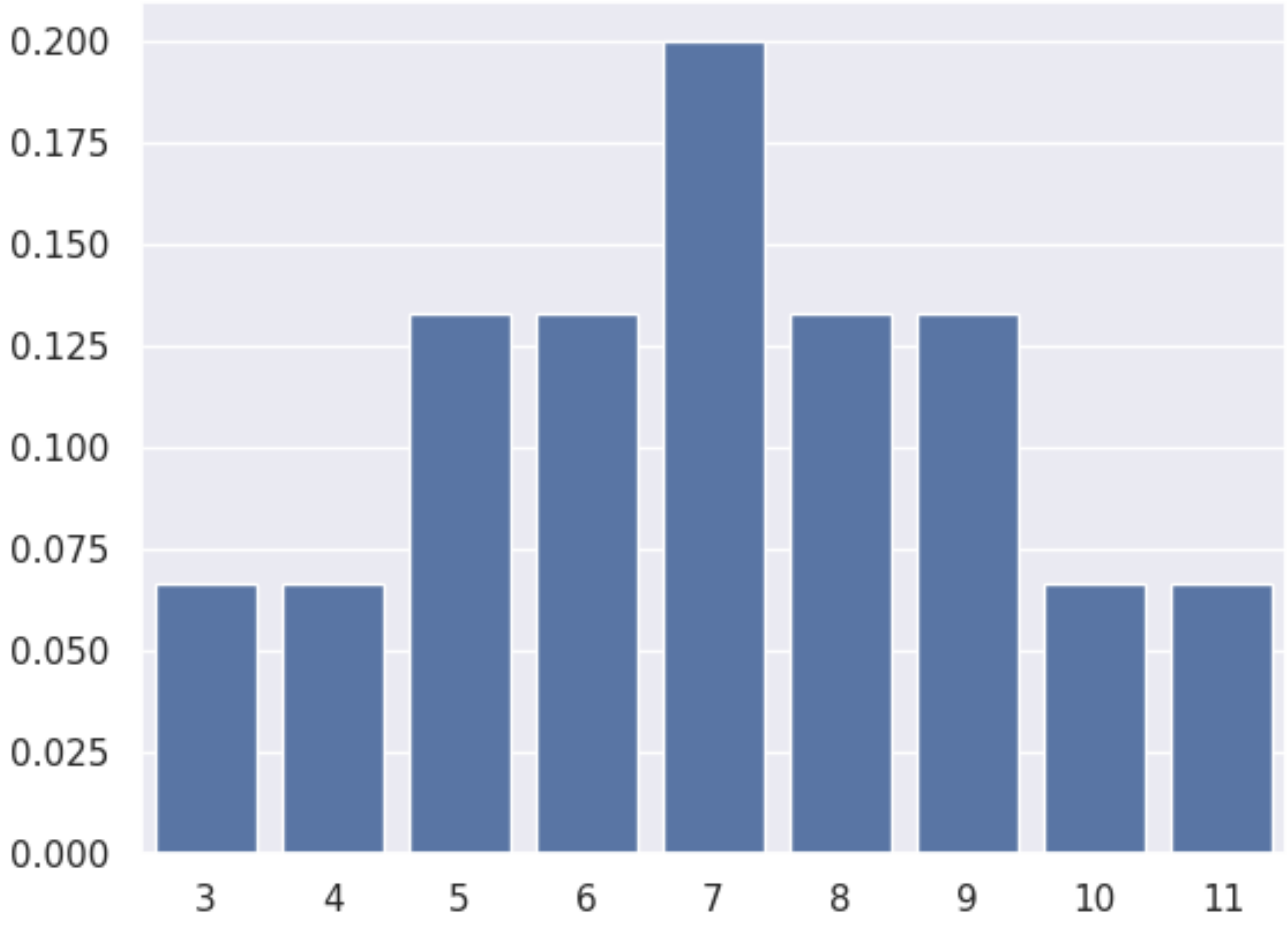
```
def hard_dice() → int:
  a = sample(RandInt(1, 6), name=”a”)
  b = sample(RandInt(1, 6), name=”b”)
  assume (a ≠ b)
  return a + b

with Enumeration():
  dist: Categorical[int] = infer(dice)
  viz(dice)
```

The sum of two independent fair dice, given that the output value are distinct.

$$[\![\texttt{hard\_dice}]\!](k) = \mathbb{P}(a+b=k|a\neq b)$$

$$= \frac{\sum_{a\neq b\in\{1,\ldots,6\}^2}\frac{1}{36}\mathbb{1}_{\{a+b=k\}}}{\sum_{a\neq b\in\{1,\ldots,6\}^2}\frac{1}{36}}$$



Bayesian programming

MPRI-PPL

Probabilistic programming

Notation: $x \sim \mu$

- parameter (output): `sample(mu)`
- observation (input): `observe(mu)`

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$\underbrace{p(x \mid y_1, \ldots, y_n)}_{\text{posterior}} = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto \underbrace{p(x)}_{\text{prior}} \underbrace{p(y_1, \ldots, y_n \mid x)}_{\text{likelihood}} \qquad \text{(Data are constants)}$$

Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer(m, y)`: compute posterior distribution of `m` given `y`

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

A Bayesian model is described by a program

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$\underbrace{p(x \mid y_1, \ldots, y_n)}_{\text{posterior}} = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto \underbrace{p(x)}_{\text{prior}}\, \underbrace{p(y_1, \ldots, y_n \mid x)}_{\text{likelihood}} \qquad \text{(Data are constants)}$$

```
def model(y1, ... , yn):
  x = sample prior
  observe (likelihood(x), (y1, ... , yn))
  return x

infer(model, (y1, ... , yn))
```

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

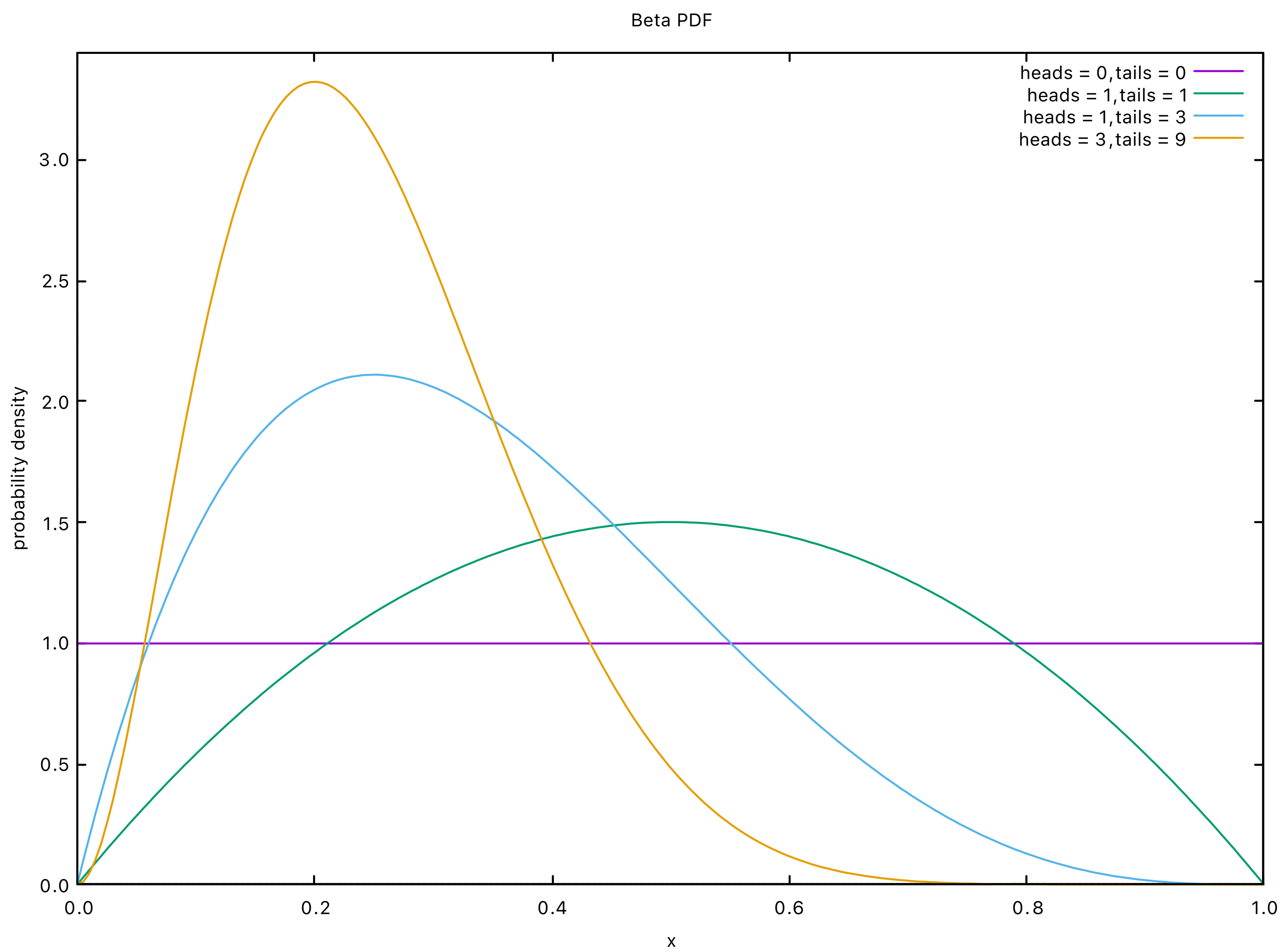
Example: coin

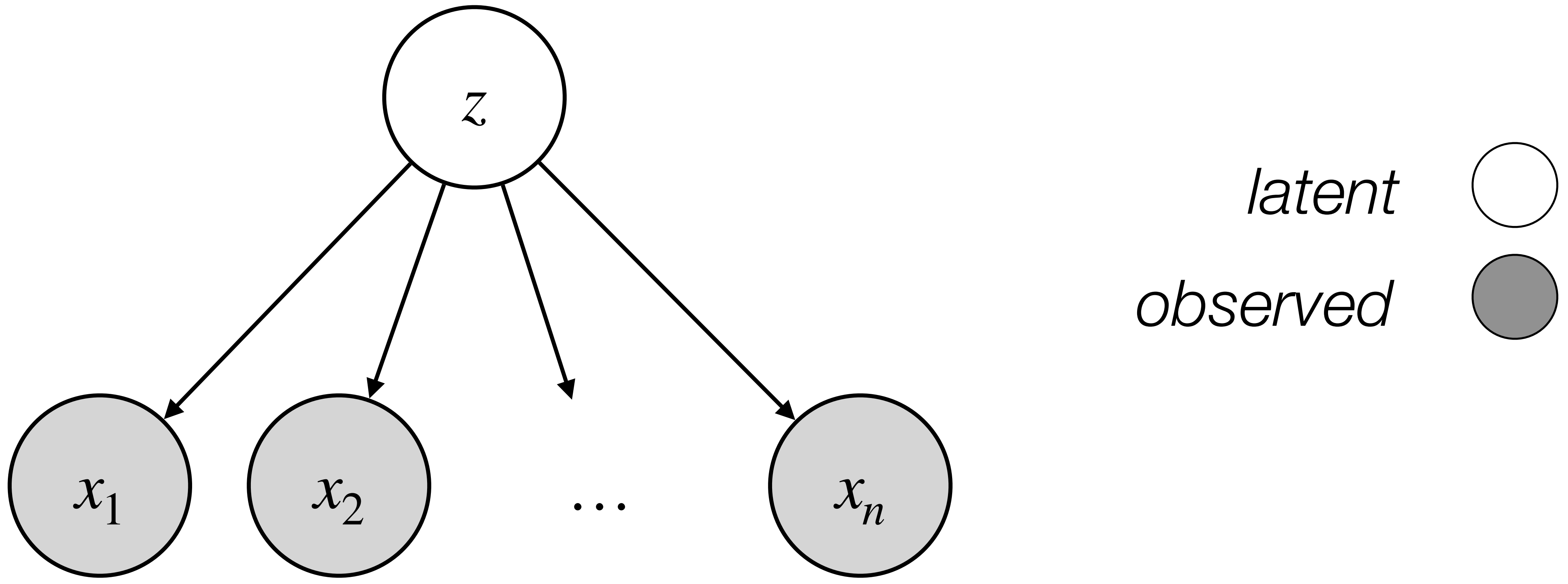
Consider a series of coin tosses

- Observations: head or tail
- Each toss is independant
- What is the probability of getting head at the next toss?

Probabilistic model

- Prior: $z \sim \textit{Uniform}(0, 1)$
- Observations: for $i \in [1, n]$, $x_i \sim \textit{Bernoulli}(z)$
- Posterior: $p(z \mid x_1, \ldots, x_n)$?





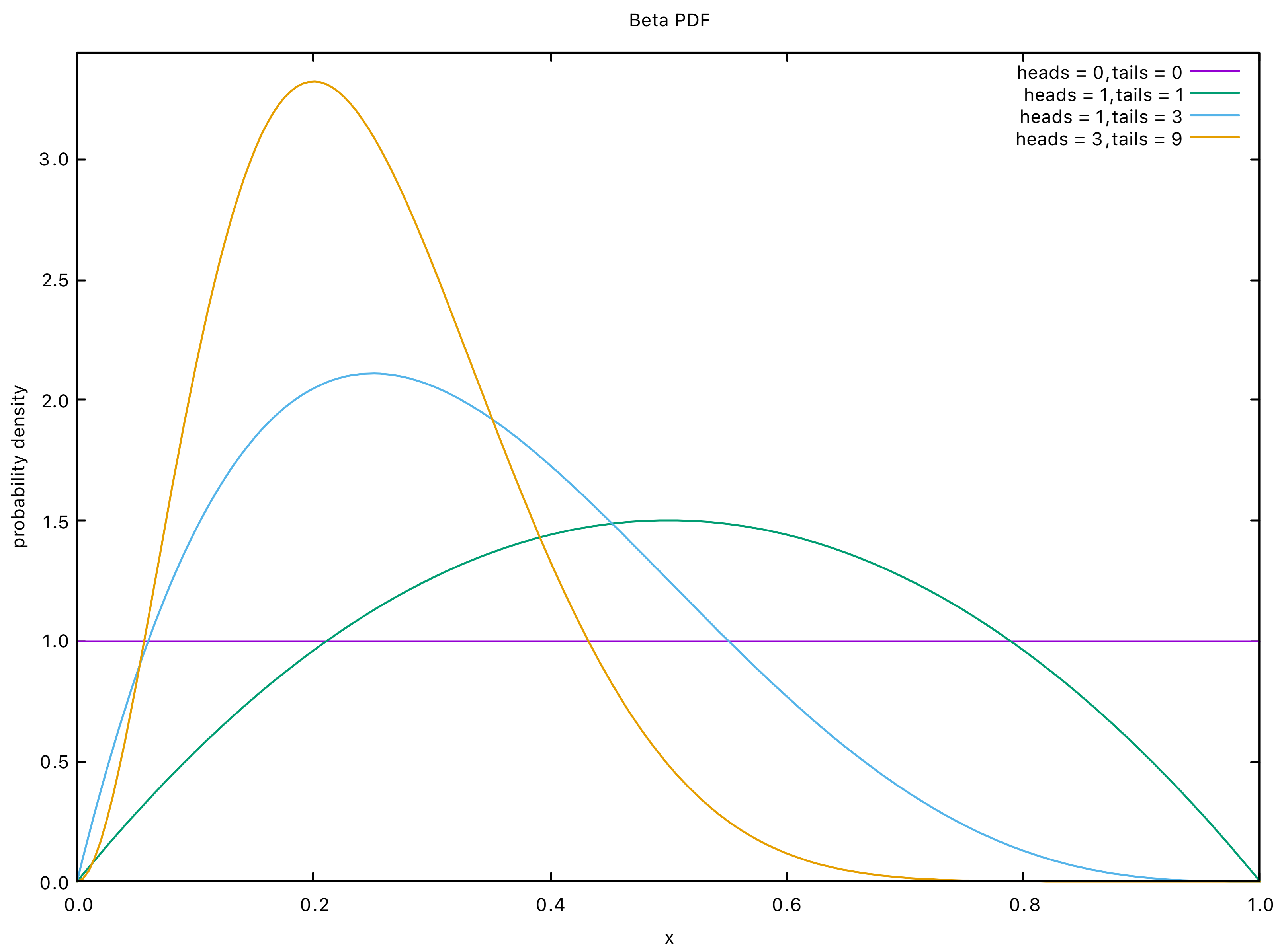
Example: coin

Consider a series of coin tosses

- Observations: head or tail
- Each toss is independant
- What is the probability of getting head at the next toss?

Probabilistic model

- Prior: $z \sim \mathit{Uniform}(0, 1)$
- Observations: for $i \in [1, n]$, $x_i \sim \mathit{Bernoulli}(z)$
- Posterior: $p(z \mid x_1, \ldots, x_n)$?




```
def coin(obs: List[int]) → float:
    p = sample(Uniform(0, 1))
    for o in obs:
        observe(Bernoulli(p), o)
    return p
```

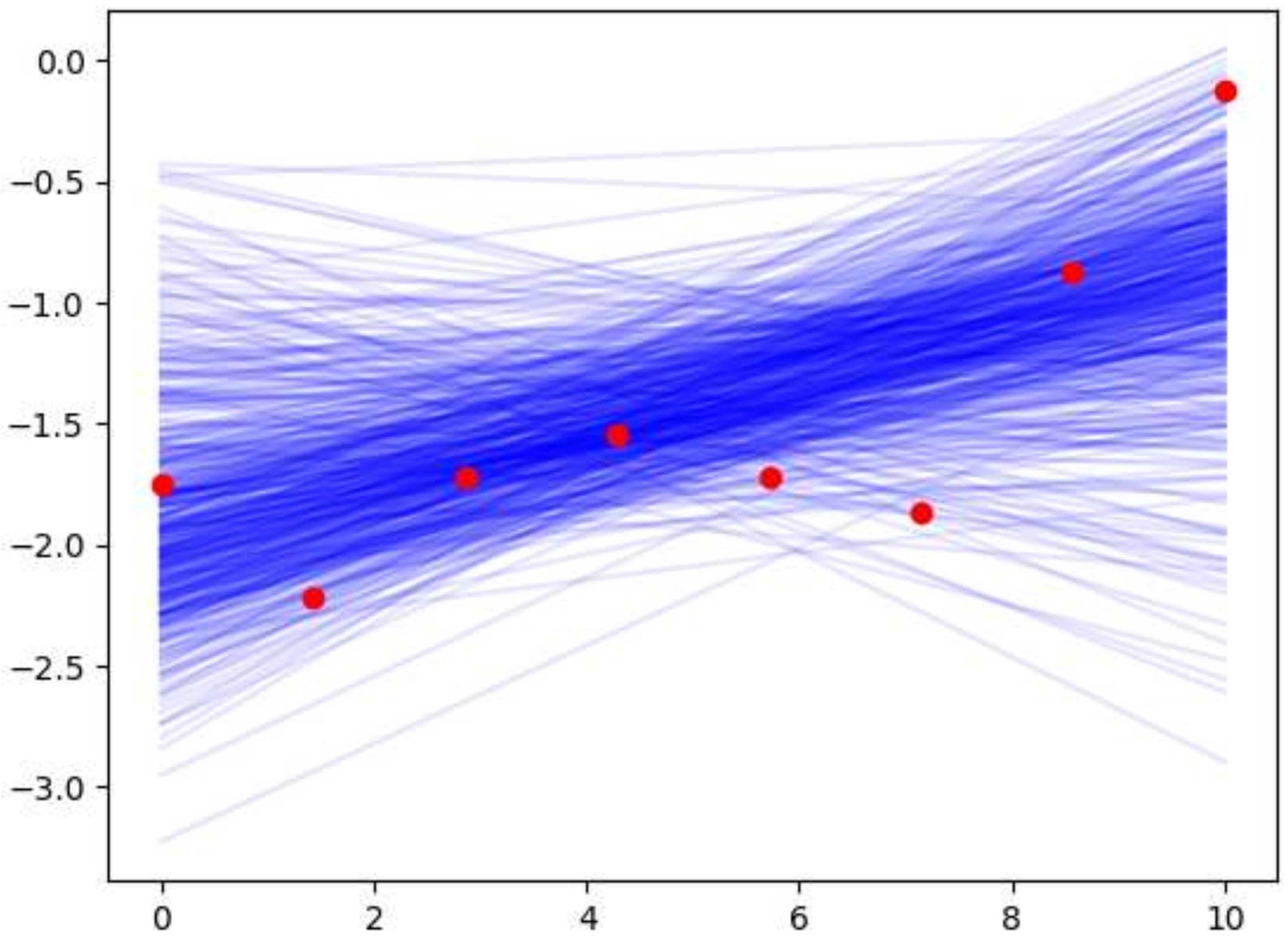
Example: linear regression

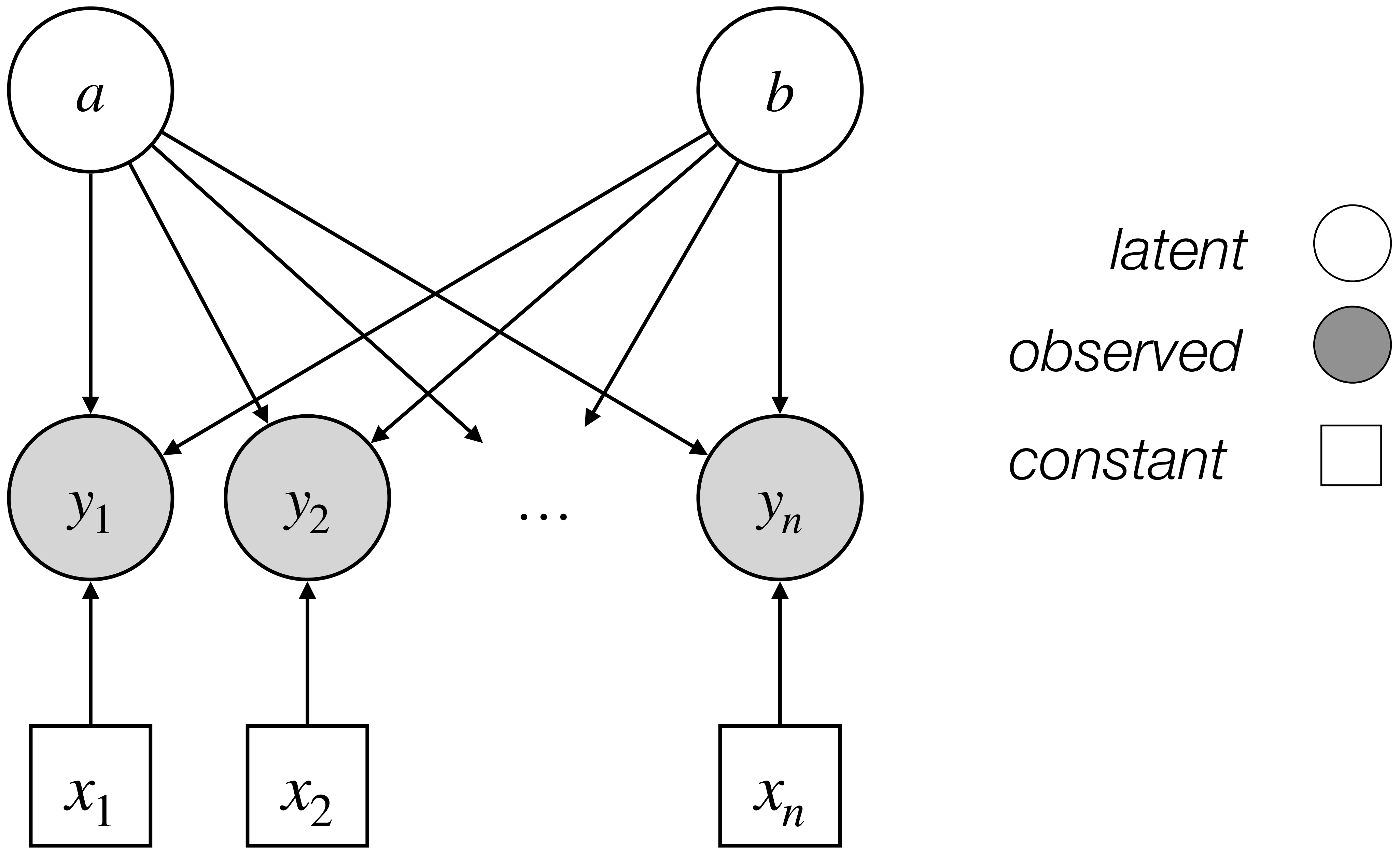
Consider a series of observations

- Observation: point (x, y)
- Each point is independent from others
- Find the distribution of possible regressions

Probabilistic model

- Prior: $a \sim \mathcal{N}(0, 1)$, and $b \sim \mathcal{N}(0, 1)$
- Observations: for $i \in [1, n]$, $y_i \sim \mathcal{N}(a \times x_i + b, \sigma)$
- Posterior: $p(a, b \mid (x_1, y_1) \ldots, (x_n, y_n))$?





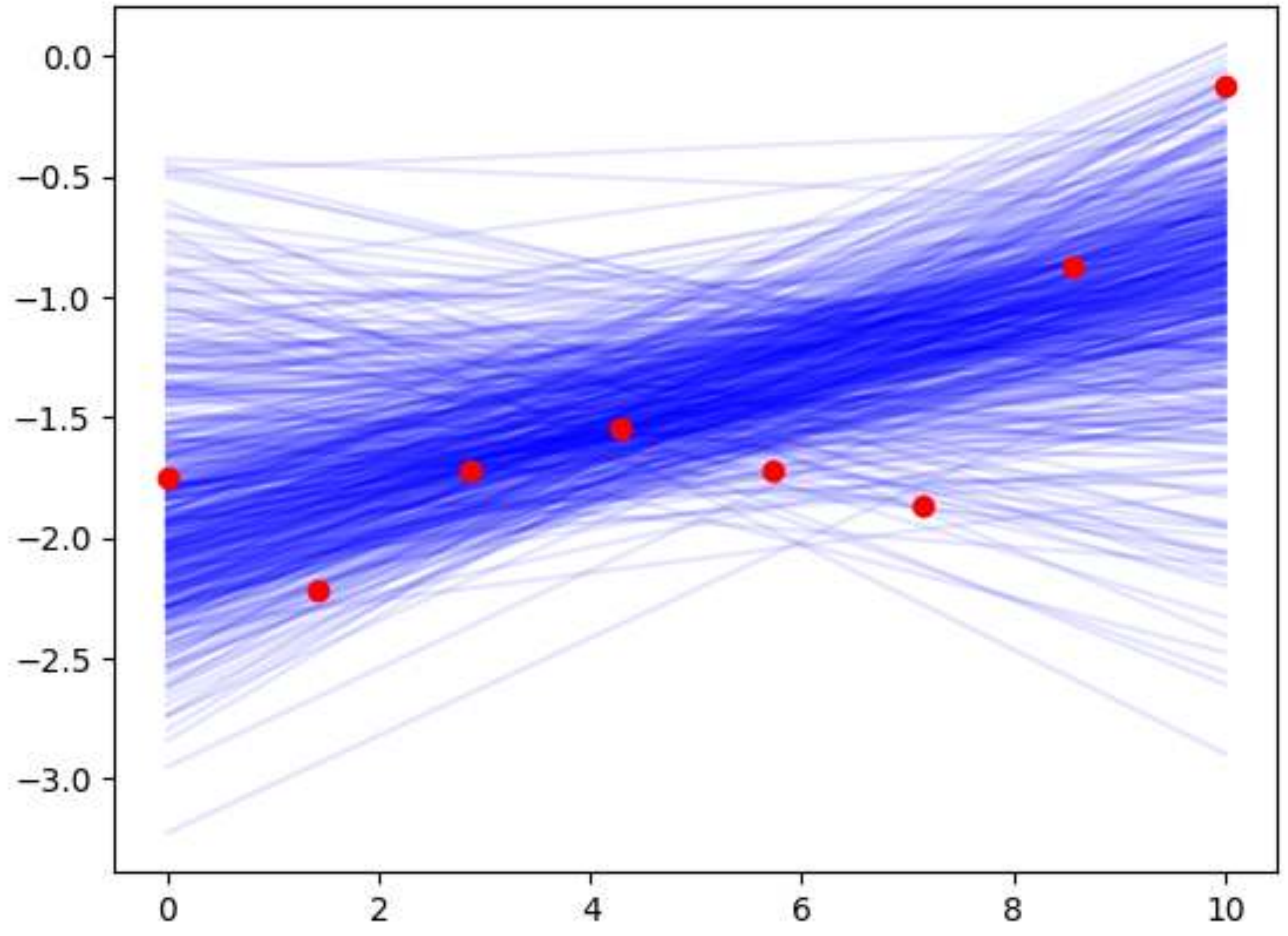
Example: linear regression

Consider a series of observations

- Observation: point (x, y)
- Each point is independent from others
- Find the distribution of possible regressions

Probabilistic model

- Prior: $a \sim \mathcal{N}(0, 1)$, and $b \sim \mathcal{N}(0, 1)$
- Observations: for $i \in [1, n]$, $y_i \sim \mathcal{N}(a \times x_i + b, \sigma)$
- Posterior: $p(a, b \mid (x_1, y_1) \ldots, (x_n, y_n))$?




```
def model(x_obs, y_obs):
    a = sample(Gaussian(0, 5))
    b = sample(Gaussian(0, 5))
    sigma = sample(Uniform(0, 2))
    for (xo, yo) in zip(x_obs, y_obs):
        observe(Gaussian(a*xo + b, sigma), yo)
    return [a, b, sigma]
```

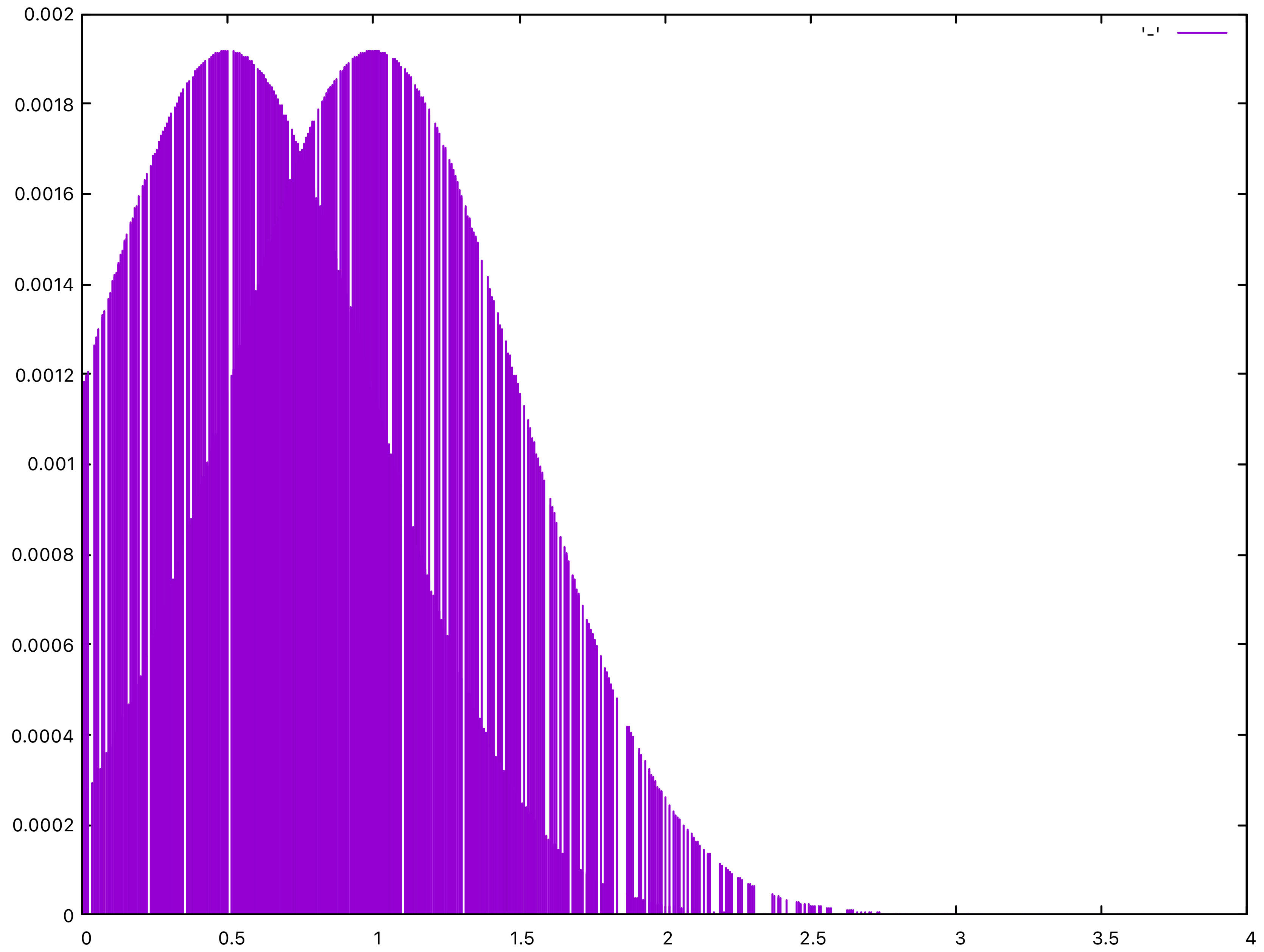
Probabilistic programming

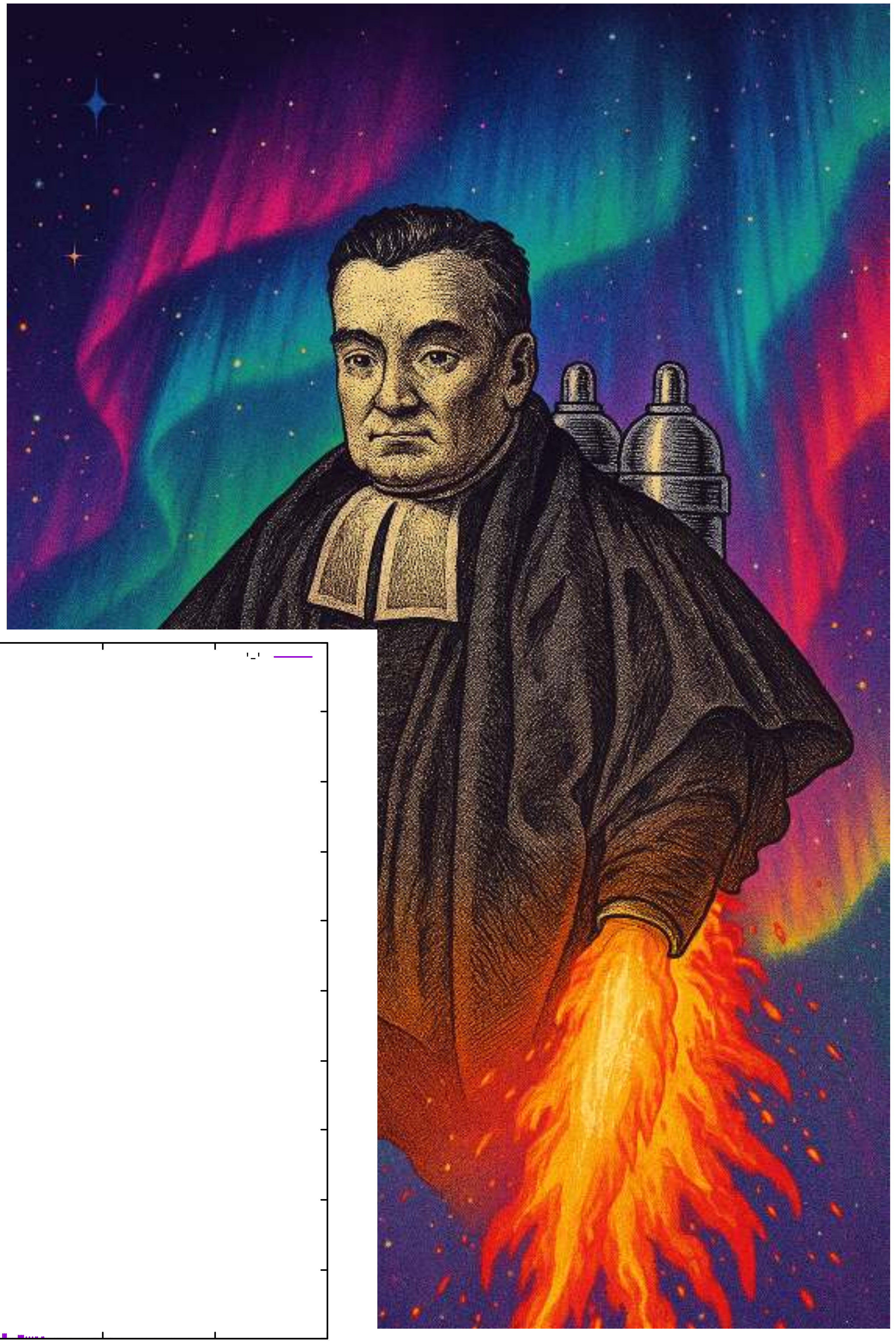
Probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on observed data
- `infer`: compute the posterior distribution of a model given the inputs

More general than classic Bayesian reasoning

```
def weird() =
  b = sample(Bernoulli(0.5))
  mu = 0.5 if (b = 1) else 1.0
  theta = sample(Gaussian(mu, 1.0))
  if theta > 0.:
    observe (Gaussian(mu, 0.5), theta)
    return theta
  else:
    return weird ()
```





Credit: ChatGPT

Probabilistic programming

Probabilistic constructs

- `sample`: draw a sample from a distribution
- `assume`, `factor`, `observe`: condition the model on observed data
- `infer`: compute the posterior distribution of a model given the inputs

More general than classic Bayesian reasoning

But: general case....

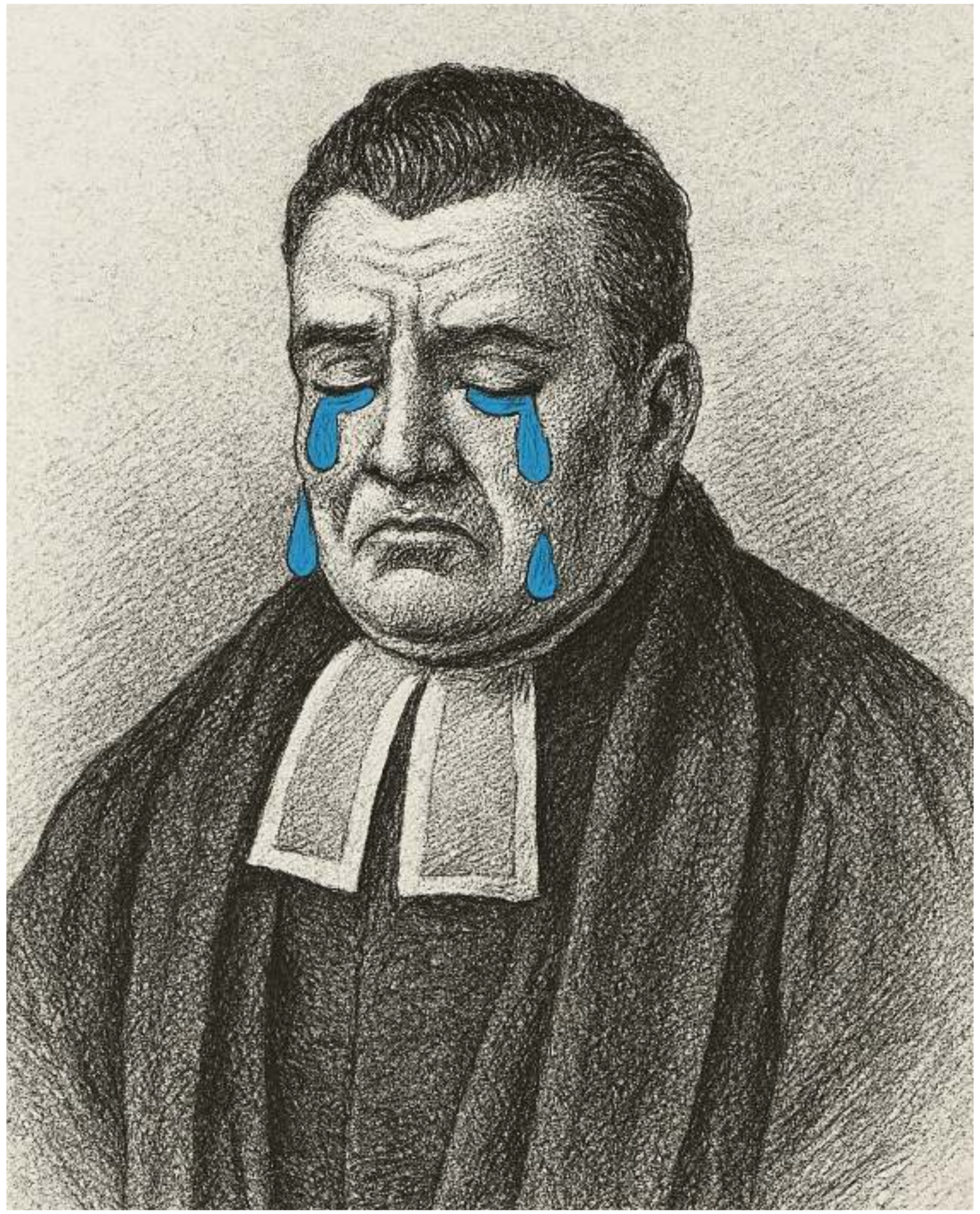
$$p(\theta \mid x_1, \ldots x_n) = \frac{p(\theta)\; p(x_1, \ldots, x_n \mid \theta)}{p(x_1, \ldots, x_n)}$$

(Bayes' theorem)

$$= \frac{p(\theta)\; p(x_1, \ldots, x_n \mid \theta)}{\int p(\theta)\; p(x_1, \ldots, x_n \mid \theta) d\theta}$$

=

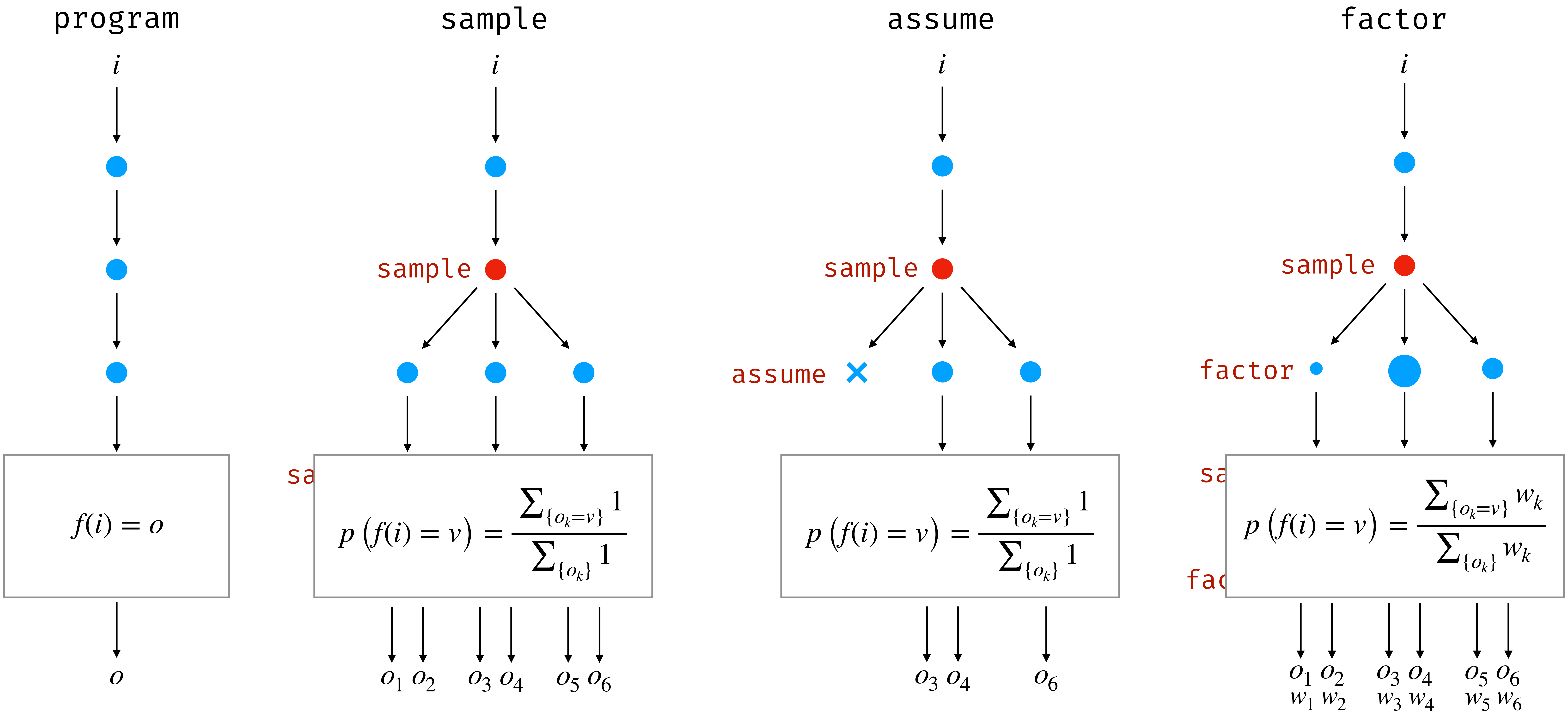
intractable...



Inference

Probabilistic Programming Languages

`infer` : $(\alpha \to \beta) \to \alpha \to \beta$ `dist`



Hard/soft conditioning

```python
# Reject if [p] is not true.
def assume(self, p):
  if not p: self.score += -np.inf

# Assume [x] was sampled from [d].
def observe(self, d, x):
  v = self.sample(d)
  self.assume(v = x)
```

Hard conditioning

```python
# Update the (log)score.
def factor(self, s):
  self.score += s

# Assume [x] was sampled from [d].
def observe(self, d, x):
  self.factor(d.logprob(x))
```

Soft conditioning

Importance sampling

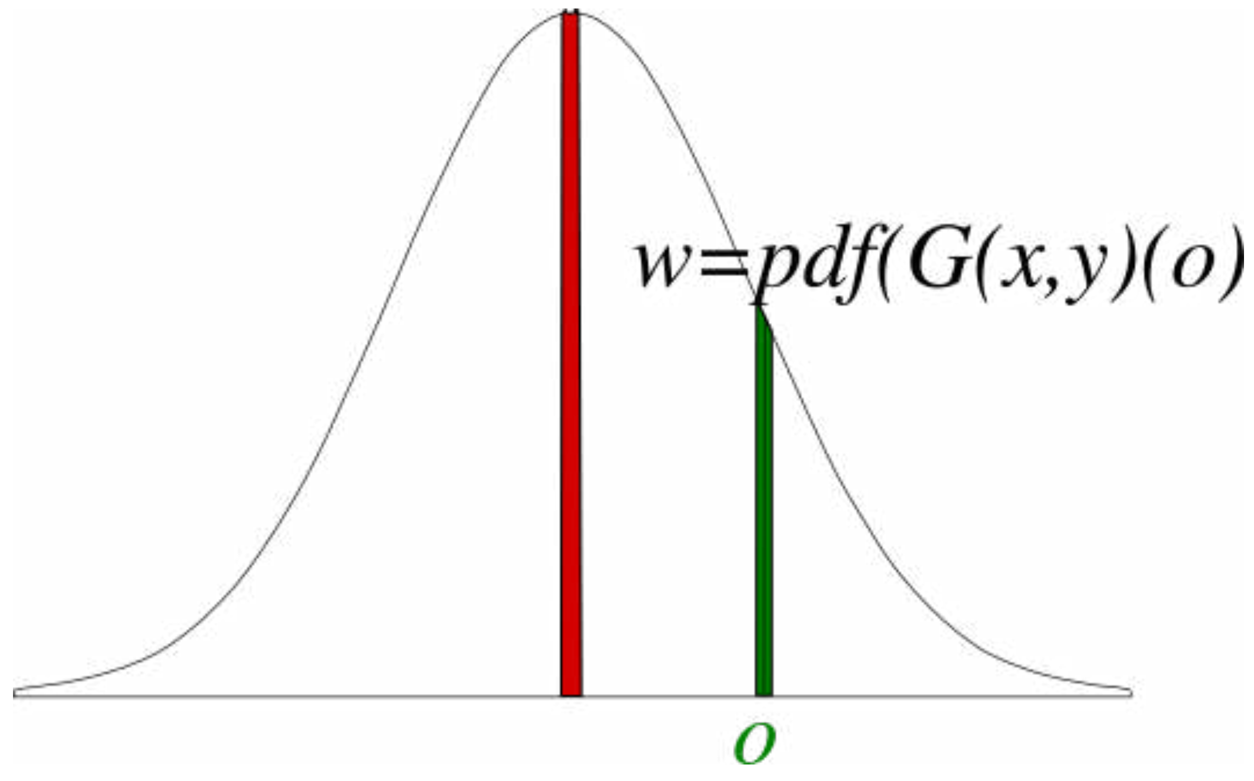
Inference algorithm

- Run a set of n independent executions
- `sample`: draw a sample from a distribution
- `factor`: associate a score to the current execution
- `infer`: gather output values v_i and score W_i to approximate the posterior distribution

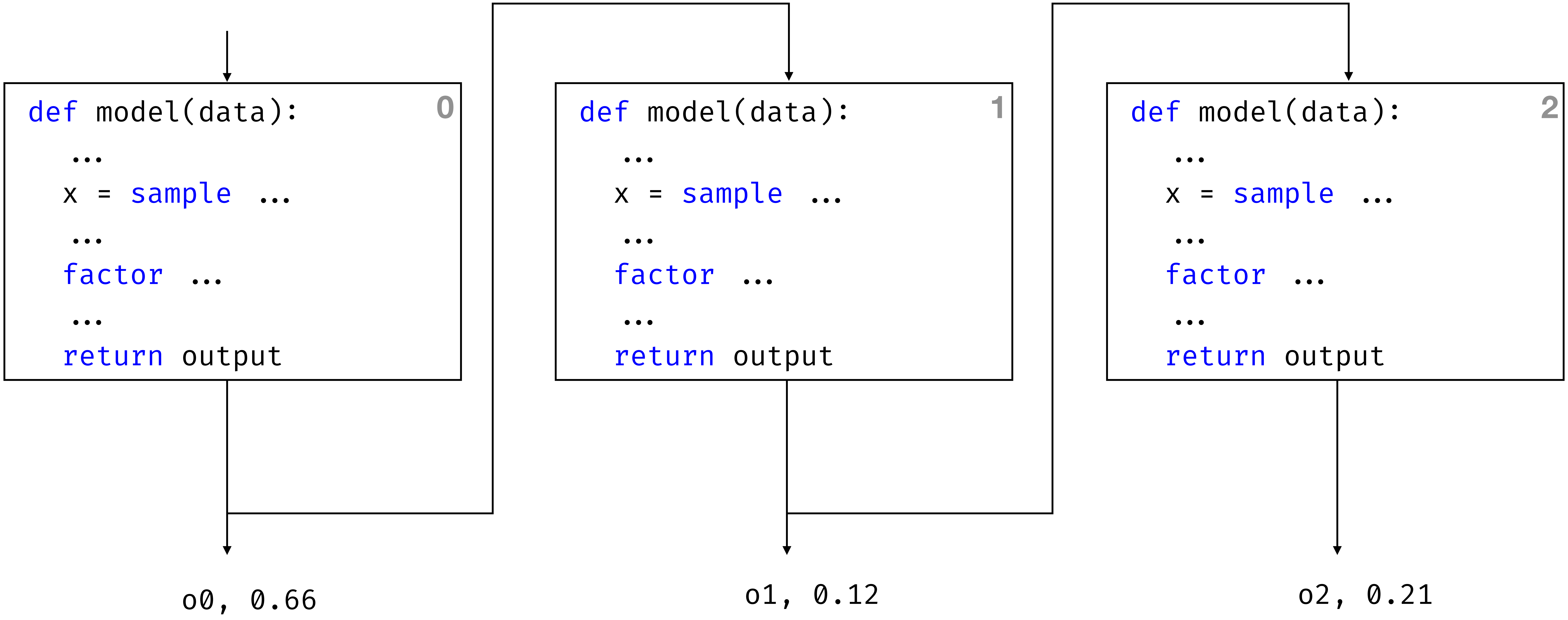
$$p_i = \frac{W_i}{\sum_{1 \leq i \leq n} W_i}$$

Conditioning

- `assume(p)`: sets the score to 0 if `p` is false
- `observe(d,v)`: multiply the score by the likelihood of `d` at `v` (density function of `d`)



Importance sampling

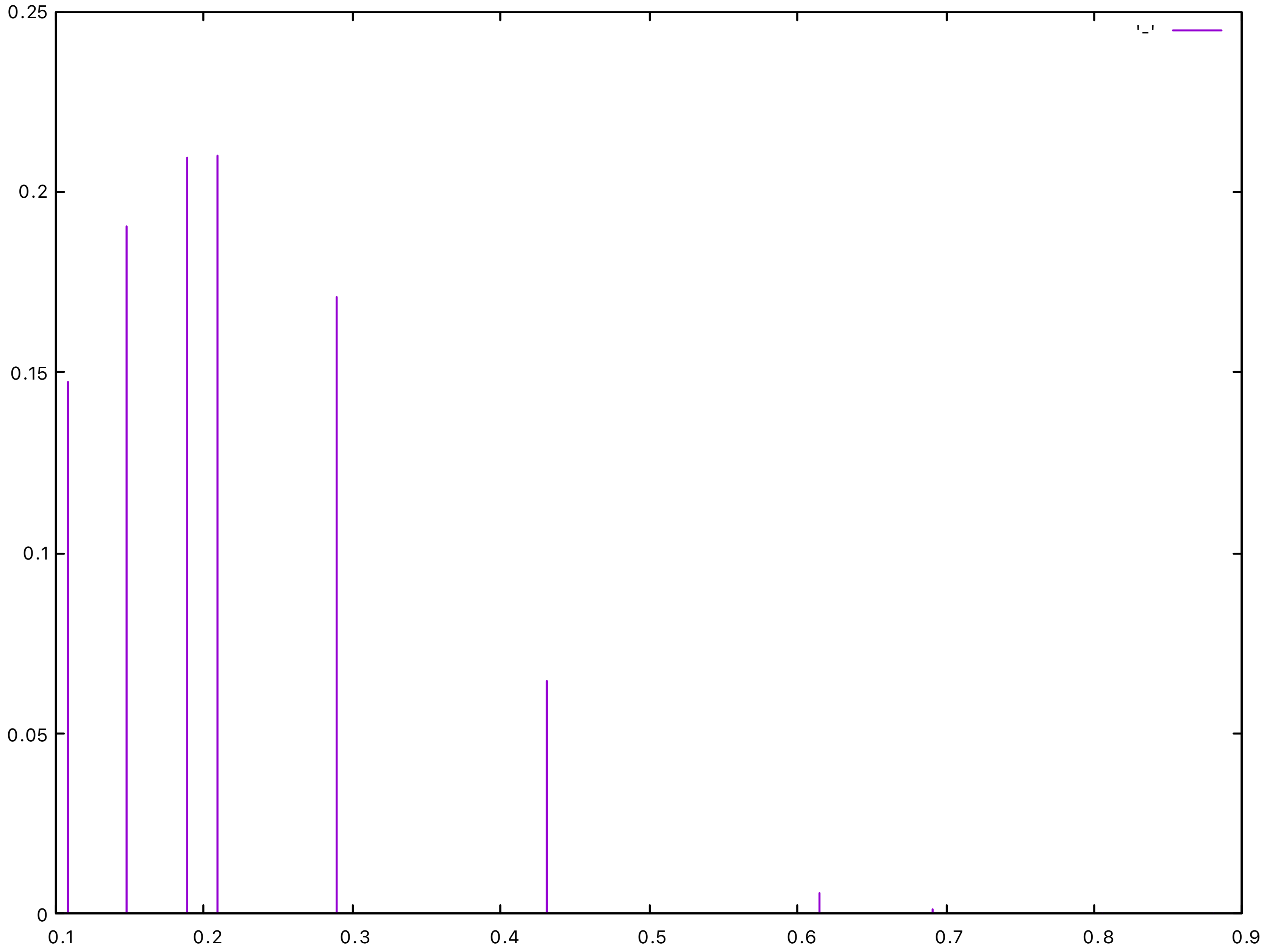


Example: coin

```python
def coin(obs: list[int]) -> float:
  p = sample(Uniform(0, 1))
  for o in obs:
    observe(Bernoulli(p), o)
  return p

with ImportanceSampling(num_particles=10):
  dist = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
  viz(dist)
```

10 particles

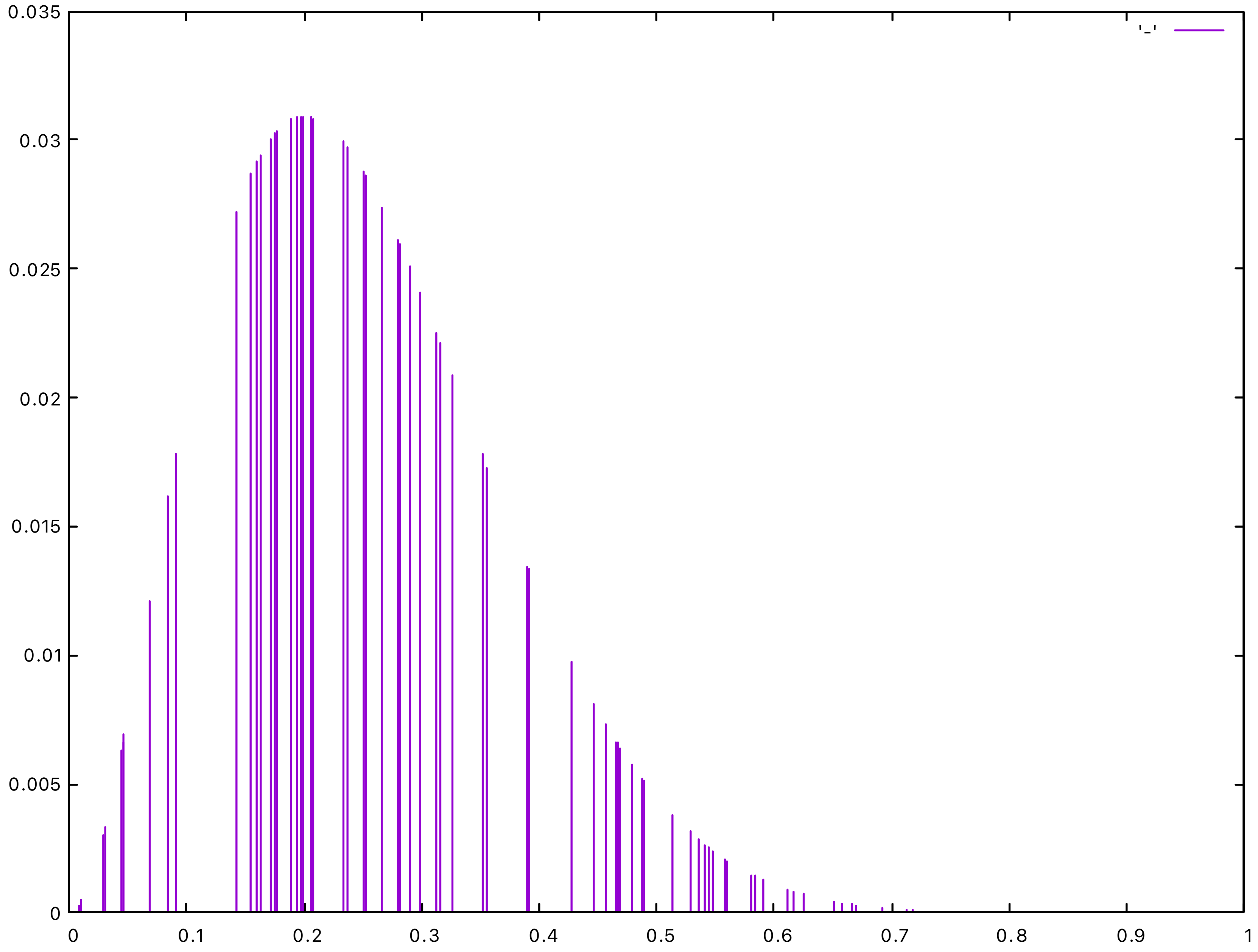


Example: coin

```
def coin(obs: list[int]) → float:
  p = sample(Uniform(0, 1))
  for o in obs:
    observe(Bernoulli(p), o)
  return p

with ImportanceSampling(num_particles=100):
  dist = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
  viz(dist)
```

100 particles

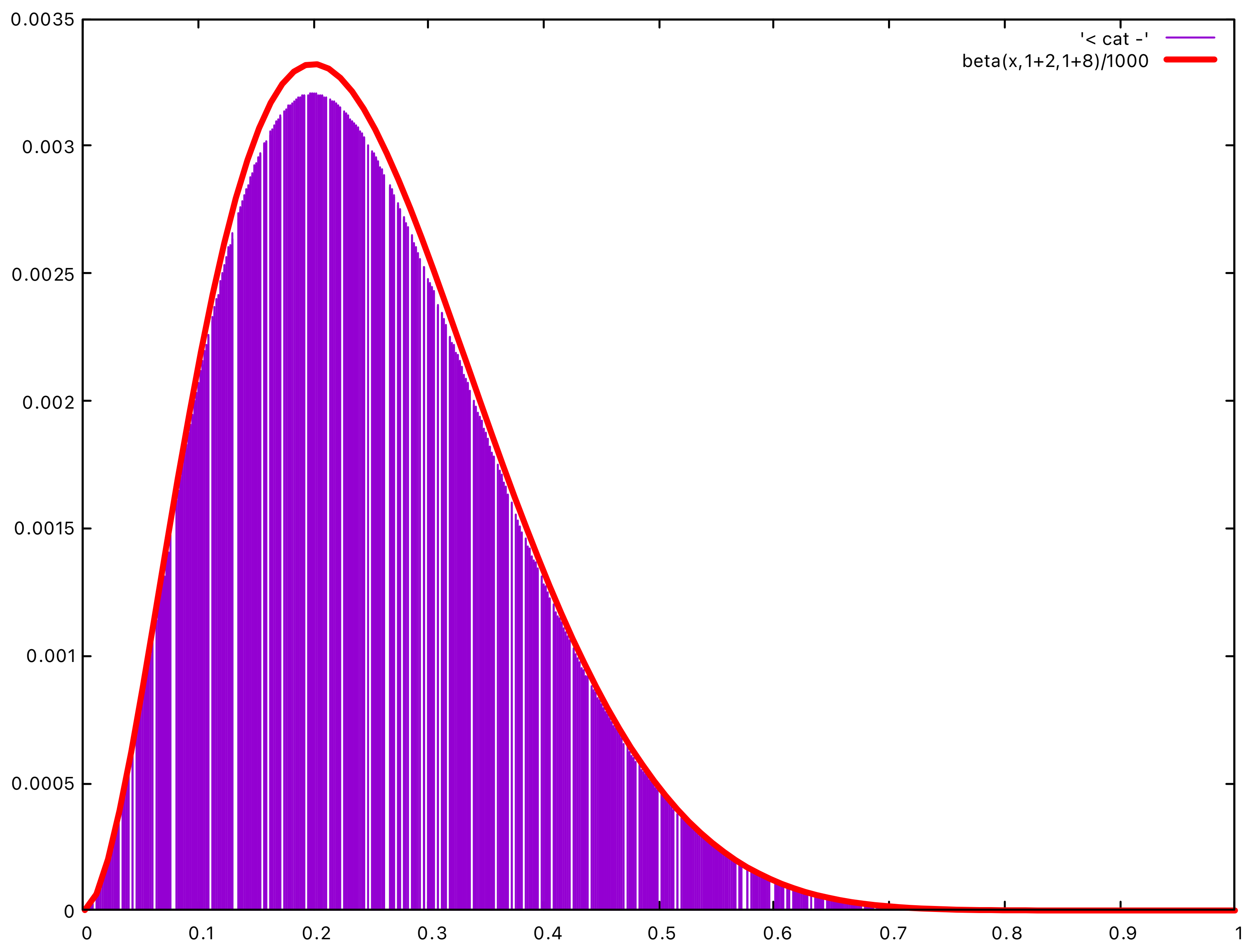


Example: coin

```
def coin(obs: list[int]) → float:
  p = sample(Uniform(0, 1))
  for o in obs:
    observe(Bernoulli(p), o)
  return p

with ImportanceSampling(num_particles=1000):
  dist = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
  viz(dist)
```

1000 particles



Exact solution: Beta(#heads + 1, #tails + 1)

Examples: mu-ppl

Reminders: Measure Theory

Probabilistic Programming Languages

Measurable Space

A σ-algebra on a set X is a collection of subsets

- containing $\emptyset$
- closed under complement
- closed under countable union

A measurable space is a pair (X, Σ_X)

- X: set
- Σ_X: measurable sets

Examples

- $\emptyset \ldots$
- Singleton: $\{\emptyset, \{\texttt{unit}\}\}$
- Booleans: $\{\emptyset, \{\texttt{true}\}, \{\texttt{false}\}, \{\texttt{true}, \texttt{false}\}\}$
- Borel sets (intervals on $\mathbb{R}$)
- Product: $\Sigma_{A \otimes B}$ generated by the rectangles

Measure

A measure maps a measurable set to a positive score: $\mu : \Sigma_X \to [0, \infty)$

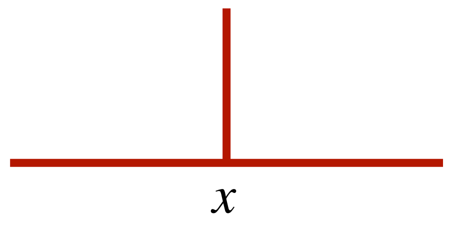
- $\Sigma_X, \mu(U) \geq 0$ for all set U in Σ_X
- $\mu(\emptyset) = 0$
- $\mu(\bigcup_{i \in I} U_i) = \sum_{i \in I} \mu(U_i)$ for all countable collections $\{U_i\}_{i \in I}$ of pairwise disjoint sets in Σ_X

Probability distributions are normalized measures, i.e., $\mu(X) = 1$

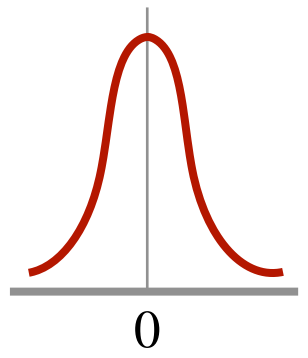
Examples

- Lebesgue measure
  $\lambda(a, b) = b - a$
- Bernoulli distribution (discret
  $\mathcal{B}(0.3)(\{1\}) = 0.3$
  $\mathcal{B}(0.3)(\{0, 1\}) = 1$

- Dirac delta measure
  $\delta_x(U) = \begin{cases} 1 & \text{if } x \in U \\ 0 & \text{otherwise} \end{cases}$



- Normal distribution
  $\mathcal{N}(0, 1)([0, 1]) \approx 0.34$
  $\mathcal{N}(0, 1)((-\infty, 0]) = 0.5$



Measurable Function

$f : X \to Y$ is measurable if $f^{-1}(U) \in \Sigma_X$ for all $U \in \Sigma_Y$

Pushforward: transfer a measure from a measurable space to another one

- $f : X \to Y$ measurable
- $\mu : \Sigma_X \to [0, \infty)$
- $f_*(\mu) : \Sigma_Y \to [0, \infty)$

Examples

- $\mu : \Sigma_{A \times B} \to [0, \infty)$
- $\pi_{1*}(\mu) : \Sigma_A \to [0, \infty)$
- $\pi_{2*}(\mu) : \Sigma_B \to [0, \infty)$

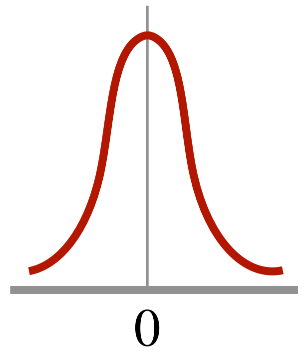
Integration

Given $f : X \to [0, \infty)$ a measurable function and a measure $\mu : X \to [0, \infty)$

- $\int_X f(x)\mu(dx) \in [0, \infty)$
- $\int_X \delta_x(U)\mu(dx) = \int_U \mu(dx) = \mu(U)$
- $\int_X f(x)\, r\, \mu(dx) = r \int_X f(x)\mu(dx)$
- $\int_X (f(x) + g(x))\mu(dx) = \int_X f(x)\mu(dx) + \int_X g(x)\mu(dx)$

We can also define new measures from integration

- $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$
- $\int_X f(x)\lambda(dx) = \int_X f(x)dx = \mathcal{N}(0,1)(X)$ using the Lebesgue measure λ
- $f(x)$ is the density w.r.t. the Lebesgue measure λ



Expected value

Expected value

- If $X \sim F$ with density f w.r.t. the Lebesgue measure
- If g is a measurable function

$$\mathbb{E}(g(X)) = \int xF(dx) = \int xf(x)dx$$

Kernel

A kernel $k : X \times \Sigma_Y \to [0, \infty)$ is a function such that:

- $k(x, \_) : \Sigma_Y \to [0, \infty)$ for all $x \in X$ is a measure
- $k(\_, U) : X \to [0, \infty)$ for all $U \in \Sigma_Y$ is measurable

A probability kernel is such that $k(x, Y) = 1$ for all $x \in X$

Law of large numbers

Sample average

- If $X_1, X_2, \ldots, X_n$ are iid.
- Sample average: $\overline{X}_n = \dfrac{1}{n}\sum_{i=1}^{n} X_i$

Strong law of large numbers

- If g is a measurable function and X, $X_1, \ldots, X_n$ are iid
- If the mean of $g(X)$ is finite
- Then the sample average approximates the mean

$$\overline{g(X)}_n \xrightarrow{a.s.} \mathbb{E}(g(X)) \text{ when n} \to \infty$$

Applications

- $g =$ *id* to compute the mean
- $g = \mathbb{1}_{[x>0]}$ to compute $P(X > 0)$
- $g = \pi_1$ to compute the first marginal of (X_1, X_2)

Correctness of importance sampling

Model

- Samples X_i from the priors with density $\pi(x)$
- Observed data: $Y = y$
- Likelihood (product of all observe statements): $\ell(x; y)$
- Posterior: $p(x|y) = \dfrac{\pi(x)\ell(x;y)}{Z}$ where $Z = \int \pi(x)\ell(x;y)dx$ (evidence)

Importance sampling estimation with n particles: $\text{IS}(g(X)|y)_n = \dfrac{\sum_{i=1}^{n} W_i\, g(X_i)}{\sum_{i=1}^{n} W_i} = \dfrac{\sum_{i=1}^{n} \ell(X_i;y)g(X_i)}{\sum_{i=1}^{n} \ell(X_i;y)} = \dfrac{\overline{(\ell(X;y)g(X))}_n}{\overline{\ell(X;y)}_n}$

Correctness with SLLN

- $\overline{(\ell(X;y)g(X))}_n \xrightarrow{a.s.} \mathbb{E}(\ell(X;y)g(X)) = \int g(x)\ell(x;y)\pi(x)dx$
- $\overline{\ell(X;y)}_n \xrightarrow{a.s.} \mathbb{E}(\ell(X;y)) = \int \ell(x;y)\pi(x)dx = Z$

Careful with 0, and ∞...

$$\text{IS}(g(X)|y)_n \xrightarrow{a.s.} \int g(x)\frac{\ell(x;y)\pi(x)}{Z}dx = \int g(x)p(x|y)dx = \mathbb{E}(g(X)|y)$$

Semantics

Probabilistic Programming Languages

Deterministic vs. probabilistic semantics

output value

$$[\![\text{prog}]\!]_\gamma : t$$

environment

$$[\![2+5]\!]_{[]} = 7$$

$$[\![\mathtt{x}+\mathtt{y}]\!]_{[\mathtt{x}\leftarrow 2,\mathtt{y}\leftarrow 5]} = 7$$

measure over possible outputs

$$\{\![\text{model}]\!\}_\gamma : \mathsf{Meas}([\![t]\!])$$

environment

$$\{\![2+5]\!\}_{[]}(U) = \delta(7)(U)$$

$$\{\![\mathtt{sample}(\mathtt{normal}(\mathtt{x},\mathtt{y}))]\!\}_{[\mathtt{x}\leftarrow 1,\mathtt{y}\leftarrow 0]}(U) = \mathcal{N}(0,1)(U)$$

(Un)normalized measures

measure over possible outputs

$$\{\!\![\,\text{model}\,]\!\!\}_\gamma : \textbf{Meas}([\![t]\!])$$

environment

Unnormalized measure

$$[\![\texttt{infer}(\text{model})]\!]_\gamma = \frac{\{\!\![\,\text{model}\,]\!\!\}_\gamma}{\{\!\![\,\text{model}\,]\!\!\}_\gamma\,([\![\mathit{typeOf}(\text{model})]\!])}$$

Distribution

normalize over all possible values

Takeaway

I - Probabilistic programming

- A program describes a distribution
- Sample: draw from distribution
- Bayesian reasoning: condition on data
- Inference computes the distribution: Intractable problem in general

II - Approximate inference

- Importance sampling: weighted sampler returns pairs (value, score)
- Correctness with the law of large numbers

III - Semantics

- Deterministic semantics: expression to value
- Probabilistic semantics: expression to measure of possible values

https://github.com/mpri-probprog/probprog-25-26