AI Programming

Lecture 1

Markus Böck and Jürgen Cito

Research Unit of Software Engineering

What is Probabilistic Programming?

What is Probabilistic Programming?

Textbook definition:

> *Probabilistic programming is a* ***programming paradigm*** *in which* ***probabilistic models*** *are specified and* ***inference*** *for these models is performed automatically.*

- ▷ Probabilistic models as programs
- ▷ Automatic posterior inference

(Explained later)

Where is the AI?

Probabilistic Programming is AI!

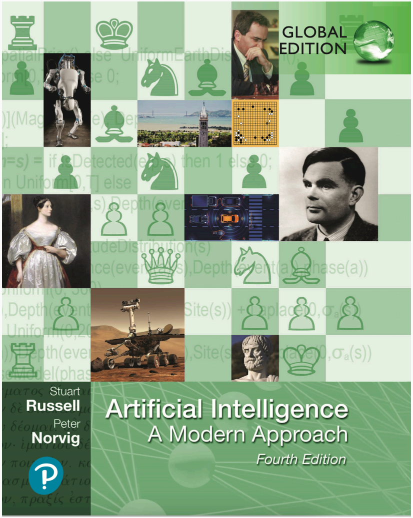
Machine Learning

- Programs define neural networks
- Data: input-output pairs
- Encodes how input maps to output
- Optimise parameters with automatic differentiation to minimise error in mapping
- Black-box approach

Probabilistic Programming

- Programs define probabilistic models
- Data: some observed data
- Encodes how unknown variables generated data
- Find distribution over unknown variables with automatic inference that ”fits” the data
- Explicit modelling + uncertainty quantification

Probabilistic Programming is AI!



IV Uncertain knowledge and reasoning

Probabilistic Programming is AI!

What is thinking?

How can we describe the intelligent inferences made in everyday human reasoning? ⟺ How can we engineer intelligent machines?

Computational theory of mind

mind = computer

mental representations = computer programs

run(program)

thinking = running a program

Probabilistic Programming is AI!

What kind of programs can represent thinking?

Why Probabilistic Programming?

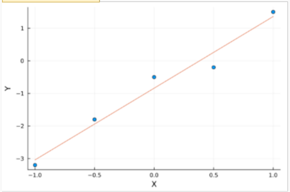
- Probabilistic models allow us to
  - incorporate prior knowledge
  - describe dependencies between variables
  - handle uncertainty
- Probabilistic programs specify probabilistic models
- Inference is concerned about updating our knowledge / belief about unknown or uncertain quantities in the program
- This is achieved by conditioning / constraining the model with observed data

Why Probabilistic Programming?

- Traditionally statisticians developed probabilistic models on paper and implemented inference algorithms
- **Probabilistic programming separates modelling from inference**
- **Expressivity:** Any probabilistic model can be implemented as a probabilistic program
- **General-purpose inference algorithms** + inference engineering
- **Enable incorporation of programming language and software engineering advances** (program analysis, debugging, visualisations,...)

First Look at Probabilistic Programming

Data



$$y = \underbrace{k}_{\text{slope}} \cdot x + \underbrace{d}_{\text{intercept}}$$

Probabilistic Model

```
using Turing
@model function linear_regression(x, y)
    # prior over latents
    slope ~ Normal(0, 3)
    intercept ~ Normal(0, 3)

    # likelihood
    for i in 1:length(x)
        # y ≈ slope * x + intercept
        y[i] ~ Normal(slope * x[i] + intercept, 1.)
    end
end
```

Posterior Inference

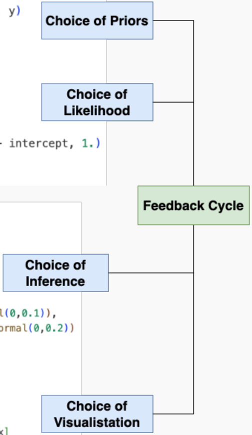
```
using AdvancedMH
function do_inference()
    x = [-1., -0.5, 0.0, 0.5, 1.0]
    y = [-3.2, -1.8, -0.5, -0.2, 1.5]
    model = linear_regression(x, y)
    res = sample(model,
        MH(
            :slope => RandomWalkProposal(Normal(0,0.1)),
            :intercept => RandomWalkProposal(Normal(0,0.2))
        ),
        1000
    )
    maximum_a_posteriori_ix = argmax(res[:lp])
    return (
        res[:slope][maximum_a_posteriori_ix],
        res[:intercept][maximum_a_posteriori_ix]
    )
end
```

First Look at Probabilistic Programming

```julia
using Turing
@model function linear_regression(x, y)
    # prior over latents
    slope ~ Normal(0, 3)
    intercept ~ Normal(0, 3)

    # likelihood
    for i in 1:length(x)
        # y ≈ slope * x + intercept
        y[i] ~ Normal(slope * x[i] + intercept, 1.)
    end
end
```

```julia
using AdvancedMH
function do_inference()
    x = [-1., -0.5, 0.0, 0.5, 1.0]
    y = [-3.2, -1.8, -0.5, -0.2, 1.5]
    model = linear_regression(x, y)
    res = sample(model,
        MH(
            :slope => RandomWalkProposal(Normal(0,0.1)),
            :intercept => RandomWalkProposal(Normal(0,0.2))
        ),
        1000
    )
    maximum_a_posteriori_ix = argmax(res[:lp])
    return (
        res[:slope][maximum_a_posteriori_ix],
        res[:intercept][maximum_a_posteriori_ix]
    )
end
```

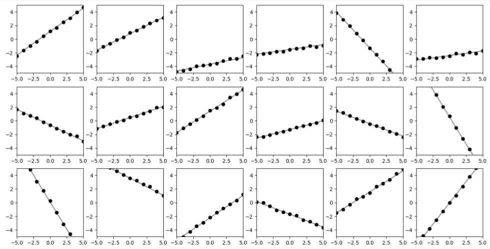


SE for PPL Research in our research group

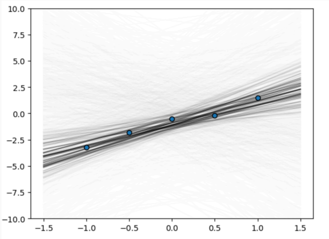
- Program Comprehension (Reasoning about Programs)
- Software Evolution (Reasoning about Change)
- Software Visualization (Reasoning about Large-scale Traces)
- Software Testing (Reasoning about Correctness)

First Look at Probabilistic Programming: Visualisation

Possible worlds according to model



Posterior distribution



Probabilistic Modelling (and Primer in Probability Theory)

Probabilistic Modelling

- The primitives in probabilistic modelling are **random variables**
- Two types of random variables:
  - **Latent variables** Θ (Unknown parameter variables)
  - **Observed variables** X (data variables)
- By relating the variables with mathematical functions, we can model dependencies between the variables
- The model denotes the **joint distribution** over latent and observed variables

Random variables

A random variable X can be viewed as a distribution on some sample space Ω – the set of possible outcomes.

Example. Bernoulli distribution parameterised by p, $\Omega = \{0, 1\}$:

$$X \sim \text{Bernoulli}(p) \iff \begin{cases} X = 1 & \text{with probability } p \\ X = 0 & \text{with probability } 1 - p \end{cases}$$

Example. Uniform distribution parameterised by $a < b$, $\Omega = [a, b]$:

$$\mathrm{P}(X \in [c, d]) = \frac{\min(b, d) - \max(a, c)}{b - a}$$

Probability mass function and density function

- A discrete variable X is fully described by its **probability mass function** p_X:

$$\mathrm{P}(X \in A) = \sum_{x \in A} p_X(x)$$

- A continuous variable X is fully described by its **probability density function** f_X:

$$\mathrm{P}(X \in A) = \int_A f_X(x) dx$$

Basic properties of random variables

- $\mathrm{P}(X \in \Omega) = 1$
- $\mathrm{P}(X \in \emptyset) = 0$
- For disjoint outcomes $A \cap B = \emptyset$ we have $\mathrm{P}(X \in A \cup B) = \mathrm{P}(X \in A) + \mathrm{P}(X \in B)$
- Expected value for discrete variables $\mathbb{E}[X] = \sum_{x \in \Omega} x \cdot p_X(x)$
- Expected value for continuous variables $\mathbb{E}[X] = \int_\Omega x \cdot f_X(x) dx$

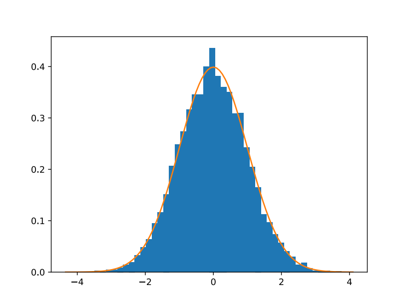
Monte Carlo Simulation

By the **law of large numbers** the arithmetic mean of a sample approaches the expected value and the histogram approaches the density function when increasing the sample size.

```
torch.manual_seed(0)
sample = dist.Normal(0,1).sample((10_000,))
plt.hist(sample, bins=50, density=True)
x = torch.linspace(sample.min(),sample.max(), 100)
plt.plot(x, dist.Normal(0,1).log_prob(x).exp())
plt.savefig("lecture_1_figs/normal_hist.pdf")
sample.mean()
```

✓ 0.1s

```
tensor(-0.0107)
```



First probabilistic model

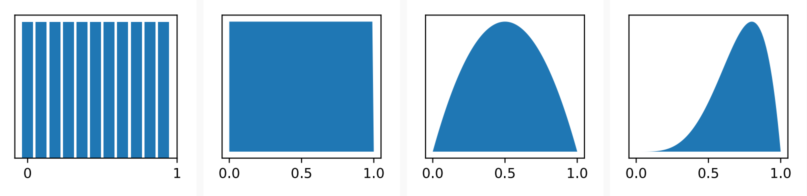
Scenario: A friend comes to us and wants to play a game of flipping coins. We are suspicious of the coin that the friend brought and we want to infer **whether the coin is fair**.

Observed variable: results of coin flips head/tail X.

Unknown variable: the probability of flipping heads p.

i-th coin flip: $X_i \sim \text{Bernoulli}(p)$

$p \sim$??



$p \sim \text{Uniform}(0, 1)$ is a choice

First probabilistic program

```julia
using Turing

@model function coinflip(y)
    p ~ Uniform(0,1)
    N = length(y)
    for n in 1:N
        y[n] ~ Bernoulli(p)
    end
end

y = [0,1,1,0,1,1,1,0,1,1]
```

Bayesian Inference

Bayesian view of probability

Frequentist probability:

The probability of an event is its relative frequency over time

Bayesian probability:

Probability is a measure of the *degree of belief* of the individual assessing the uncertainty of a particular situation.

Probability represents a *state of knowledge*.

Bayesian statistics

Bayesian modelling

- Prior $\Theta \sim \mathrm{P}(\Theta)$
  Encodes our prior information/belief about the latent variables
- Likelihood $X \sim \mathrm{P}(X|\Theta)$
  Encodes the way the observations are believed to be generated from the latents
- Joint $(\Theta, X) \sim \mathrm{P}(X|\Theta) \cdot \mathrm{P}(\Theta)$
  Specifies the full probabilistic model
- Posterior $\Theta \sim \mathrm{P}(\Theta|X)$
  Is the distribution of latent variables *given* that we have observed the data. It denotes the updated information/belief about the latent variables after the experiment

Coin flip model

- $p \sim \mathrm{Uniform}(0, 1)$
- $X_i \sim \mathrm{Bernoulli}(p)$
- How to find posterior?

Bayes' Theorem

Θ … latent/unknown variables, X … data/observed variables

$$\underbrace{\mathrm{P}(\Theta|X)}_{\text{posterior}} = \frac{\overbrace{\mathrm{P}(X|\Theta)}^{\text{likelihood}} \cdot \overbrace{\mathrm{P}(\Theta)}^{\text{prior}}}{\underbrace{\mathrm{P}(X)}_{\text{evidence}}}$$

We can compute likelihood and prior.

The evidence and posterior are in general infeasible.

However, we can compute ratios $\mathrm{P}(\Theta = \theta_1|X)/\mathrm{P}(\Theta = \theta_2|X)$.

Probabilistic Programming Automates Bayesian Inference

```julia
using Turing

@model function coinflip(y)
    p ~ Uniform(0,1)
    N = length(y)
    for n in 1:N
        y[n] ~ Bernoulli(p)
    end
end

y = [0,1,1,0,1,1,1,0,1,1]

Turing.Random.seed!(0)
res = sample(coinflip(y), NUTS(), 1000)
```

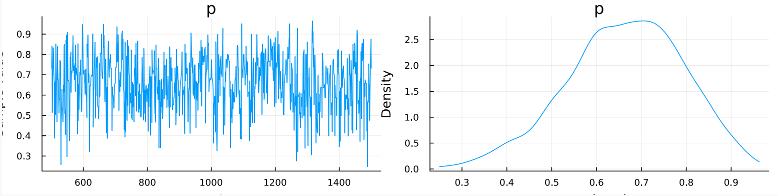
First inference result

Summary Statistics

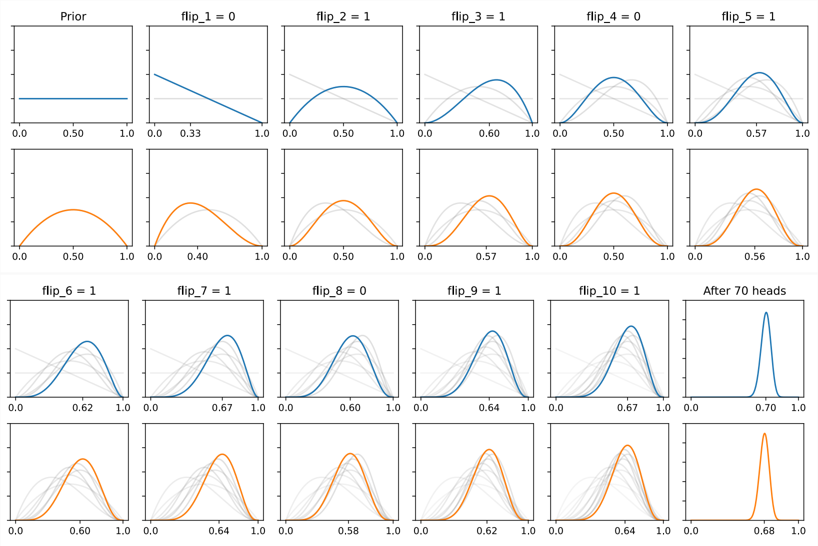
parameters	mean	std	mcse	ess_bulk	ess_tail	rhat	ess_per_sec
Symbol	Float64	Float64	Float64	Float64	Float64	Float64	Float64
p	0.6632	0.1296	0.0069	351.9368	604.9492	1.0033	4399.2097

Quantiles

parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
p	0.3878	0.5817	0.6691	0.7590	0.8974

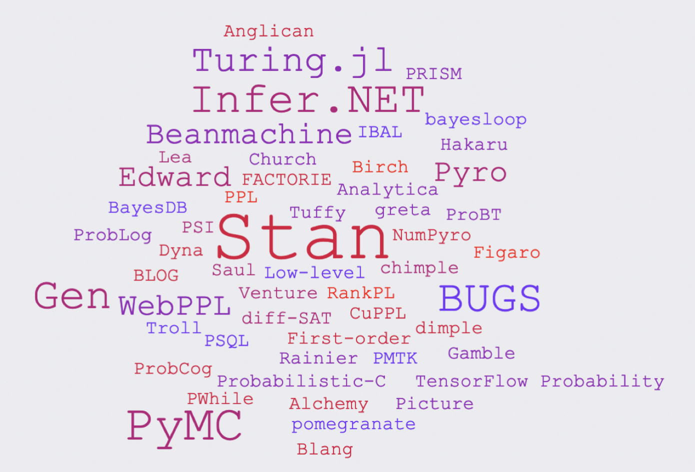


Belief updating



Probabilistic Programming Languages (PPLs)

Probabilistic Programming Languages



Coin flip model in several PPLs

```
data {
    int N;
    int y[N];
}
parameters {
    real p;
}
model {
    p ~ uniform(0,1);
    for (n in 1:N)
        y[n] ~ bernoulli(o);
}
```

```python
import pyro
def coinflip(y):
    p = pyro.sample("p", dist.Uniform(0,1))
    with pyro.plate("flips"):
        pyro.sample("obs", dist.Bernoulli(p), obs=y)
```

```python
import pymc as pm
with pm.Model() as model:
    p = pm.Uniform("p", 0, 1)
    pm.Bernoulli("obs", p, observed=y)
```

```julia
using Gen
@gen function coinflip()
    p ~ uniform(0,1)
    N = length(y)
    for n in 1:N
        {:y => n} ~ bernoulli(p)
    end
end
```

```julia
using Turing
@model function coinflip(y)
    p ~ Uniform(0,1)
    N = length(y)
    for n in 1:N
        y[n] ~ Bernoulli(p)
    end
end
```

```python
import beanmachine as bm
@bm.random_variable
def p():
    return dist.Uniform(0,1)
@bm.random_variable
def y(i: int):
    return dist.Bernoulli(p())
```

Why so many Probabilistic Programming Languages?

Balance between expressivity and efficiency.

What class of models should I be able to implement?

How can we optimise inference for this class of models?

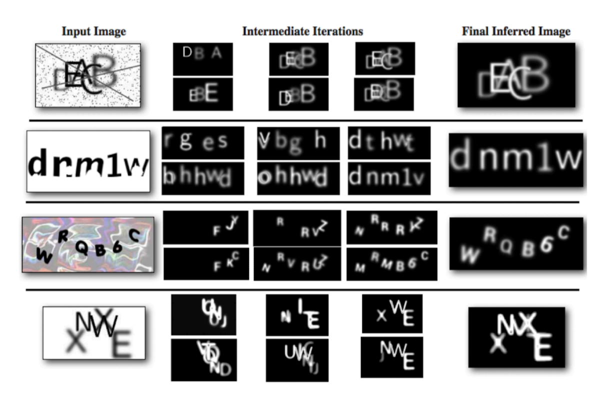
Why so many Probabilistic Programming Languages?

Balance between expressivity and efficiency.

- Stan: only continuous variables, optimised for HMC and ADVI
- Pyro: optimised for deep probabilistic programming (SVI)
- Pymc: optimised for static-structure finite-dimensional models
- Gen: facilitates inference programming
- Turing: facilitates combination of many inference algorithms
- Beanmachine: takes a declarative approach

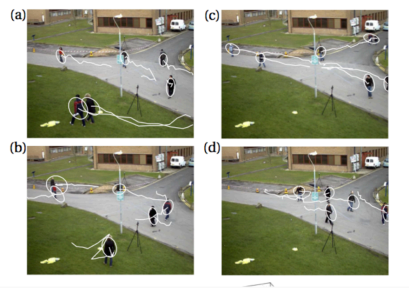
Applications

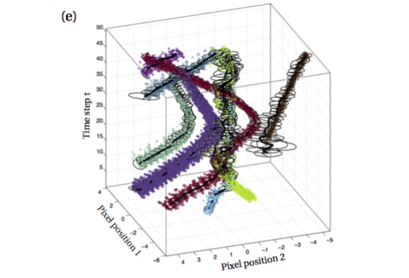
Mansinghka, V. K., Kulkarni, T. D., Perov, Y. N., & Tenenbaum, J. (2013). Approximate bayesian image interpretation using generative probabilistic graphics programs. Advances in Neural Information Processing Systems, 26.



Object Tracking

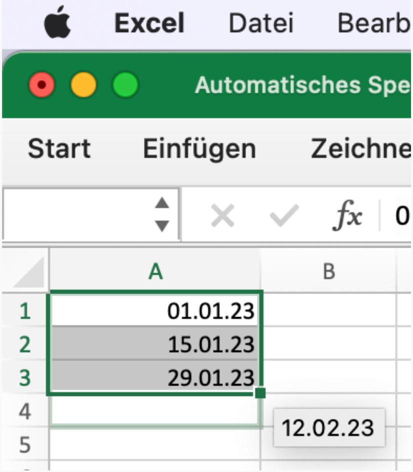
Neiswanger, W., Wood, F., & Xing, E. (2014, April). The dependent Dirichlet process mixture of objects for detection-free tracking and object modeling. In Artificial Intelligence and Statistics (pp. 660-668). PMLR.





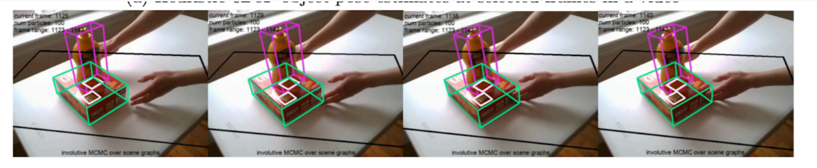
Excel Auto-Fill

Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. ACM Sigplan Notices, 46(1), 317-330.



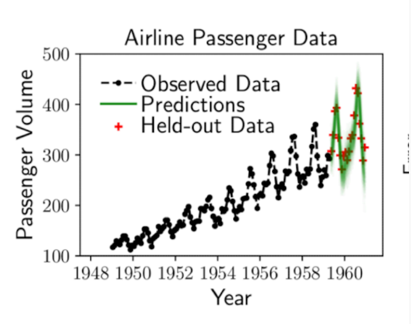
Pose Estimation

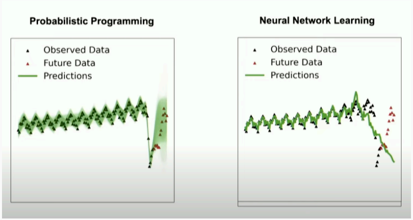
Cusumano-Towner, M. F. (2020). Gen: a high-level programming platform for probabilistic inference (Doctoral dissertation, Massachusetts Institute of Technology). Kulkarni, T. D., Kohli, P., Tenenbaum, J. B., & Mansinghka, V. (2015). Picture: A probabilistic programming language for scene perception. In Proceedings of the ieee conference on computer vision and pattern recognition (pp. 4390-4399).



(b) For each frame in (a), the inferred 6DoF object poses and object-object contact planes

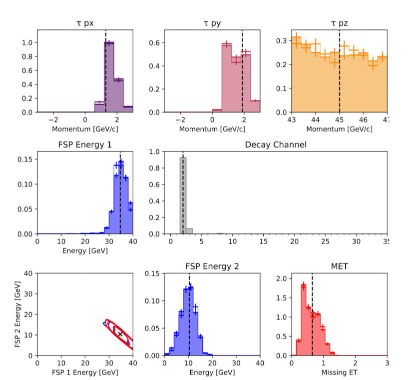
Cusumano-Towner, M. F. (2020). Gen: a high-level programming platform for probabilistic inference (Doctoral dissertation, Massachusetts Institute of Technology).





Hadron Collider

Baydin, A. G., Shao, L., Bhimji, W., Heinrich, L., Meadows, L., Liu, J., ... & Wood, F. (2019, November). Etalumis: Bringing probabilistic programming to scientific simulators at scale. In Proceedings of the international conference for high performance computing, networking, storage and analysis (pp. 1-24).



Nuclear Test Detection

Arora, N. S., Russell, S., & Sudderth, E. (2013). NET-VISA: Network processing vertically integrated seismic analysis. Bulletin of the Seismological Society of America, 103(2A), 709-729.