Probabilistic Programming

An Introduction to Applications in Machine Learning

Amirabbas Asadi

amir.asadi78@sharif.edu

Sharif University of Technology
Department of Mathematical Sciences

2024

Presentation Outline

1. Bayesian Learning
2. Probabilistic Programs
3. Approximate Bayesian Inference
   - Markov Chain Monte Carlo
   - Variational Inference
4. Differentiable Programming

Bayesian Learning

Can you guess the next number in the following sequence?

$$2, 4, 6, 8, 10, ?$$

Bayesian Learning

Can you guess the next number in the following sequence?

$$2, 4, 6, 8, 10, ?$$

What function could have generated this sequence?

Bayesian Learning

Can you guess the next number in the following sequence?

$$2, 4, 6, 8, 10, ?$$

What function could have generated this sequence?

$$f_1(n) = 2n$$

Bayesian Learning

Can you guess the next number in the following sequence?

$$2, 4, 6, 8, 10, ?$$

What function could have generated this sequence?

$$f_1(n) = 2n$$

$$f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

Bayesian Learning

Can you guess the next number in the following sequence?

$$2, 4, 6, 8, 10, ?$$

What function could have generated this sequence?

$$f_1(n) = 2n$$

$$f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

$$f_3(n) = 0.05n^5 - 0.75n^4 + 4.25n^3 - 11.25n^2 + 15.7n - 6$$

Bayesian Learning

$$H_1 : f_1(n) = 2n$$

$$H_2 : f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

$$H_3 : f_3(n) = 0.05n^5 - 0.75n^4 + 4.25n^3 - 11.25n^2 + 15.7n - 6$$

Bayesian Learning

$$H_1 : f_1(n) = 2n$$

$$H_2 : f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

$$H_3 : f_3(n) = 0.05n^5 - 0.75n^4 + 4.25n^3 - 11.25n^2 + 15.7n - 6$$

All functions reproduce the data exactly

Bayesian Learning

$$H_1 : f_1(n) = 2n$$

$$H_2 : f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

$$H_3 : f_3(n) = 0.05n^5 - 0.75n^4 + 4.25n^3 - 11.25n^2 + 15.7n - 6$$

All functions reproduce the data exactly

All have the same *likelihood*

Bayesian Learning

$$H_1 : f_1(n) = 2n$$

$$H_2 : f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

$$H_3 : f_3(n) = 0.05n^5 - 0.75n^4 + 4.25n^3 - 11.25n^2 + 15.7n - 6$$

All functions reproduce the data exactly

All have the same *likelihood*

$$p(D|H_1) = p(D|H_2) = p(D|H_3)$$

Bayesian Learning

$$H_1 : f_1(n) = 2n$$

$$H_2 : f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

$$H_3 : f_3(n) = 0.05n^5 - 0.75n^4 + 4.25n^3 - 11.25n^2 + 15.7n - 6$$

All functions reproduce the data exactly

All have the same *likelihood*

$$p(D|H_1) = p(D|H_2) = p(D|H_3)$$

Then why do people choose the first one for the same data???

Bayesian Learning

$$H_1 : f_1(n) = 2n$$

$$H_2 : f_2(n) = 0.0167n^5 - 0.25n^4 + 1.4167n^3 - 3.75n^2 + 6.5667n - 2$$

$$H_3 : f_3(n) = 0.05n^5 - 0.75n^4 + 4.25n^3 - 11.25n^2 + 15.7n - 6$$

All functions reproduce the data exactly

All have the same *likelihood*

$$p(D|H_1) = p(D|H_2) = p(D|H_3)$$

Then why do people choose the first one for the same data???

It seems people as a **prior belief** prefer H_1 over other options

Bayesian Learning

But How can we quantify and take into account a prior belief?

Bayesian Learning

But How can we quantify and take into account a prior belief?

We can formulate our prior belief $p(H)$ as distribution over all hypothesis

Bayesian Learning

But How can we quantify and take into account a prior belief?

We can formulate our prior belief $p(H)$ as distribution over all hypothesis

$$p(D|H)$$

Bayesian Learning

But How can we quantify and take into account a prior belief?

We can formulate our prior belief $p(H)$ as distribution over all hypothesis

$$p(D|H)p(H)$$

Bayesian Learning

But How can we quantify and take into account a prior belief?

We can formulate our prior belief $p(H)$ as distribution over all hypothesis

$$p(D|H)p(H)$$

To be a valid probability distribution, a normalization constant is needed

$$p(H|D) = \frac{p(D|H)p(H)}{p(D)}$$

Bayesian Learning

But How can we quantify and take into account a prior belief?

We can formulate our prior belief $p(H)$ as distribution over all hypothesis

$$p(D|H)p(H)$$

To be a valid probability distribution, a normalization constant is needed

$$p(H|D) = \frac{p(D|H)p(H)}{p(D)}$$

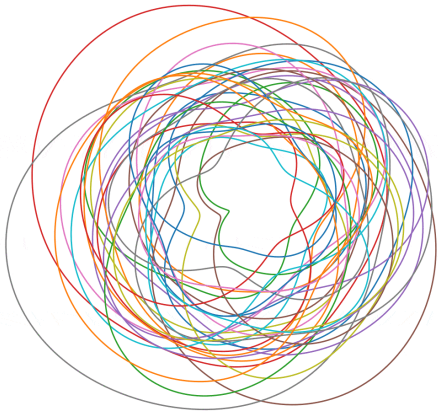
Bayes Theorem

Bayesian Learning

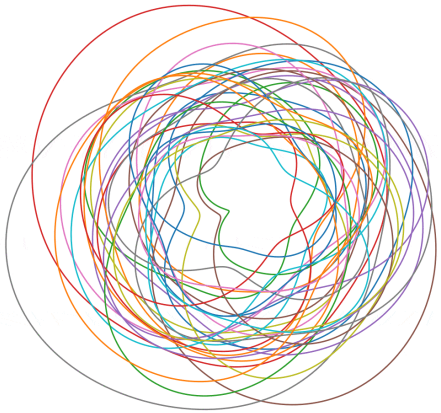
Bayesian Learning provides a natural framework for updating our beliefs.

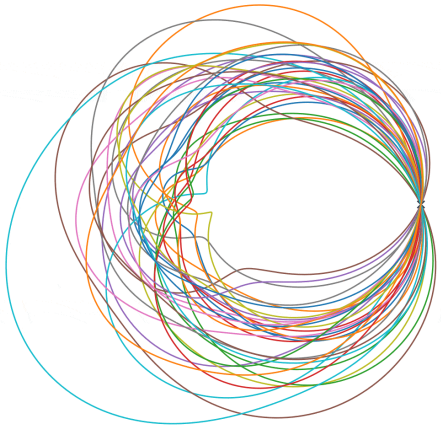
$$p(H) \rightarrow^{D_1} p(H|\{D_1\}) \rightarrow^{D_2} p(H|\{D_1, D_2\}) \rightarrow^{D_3} \cdots$$

Bayesian Learning

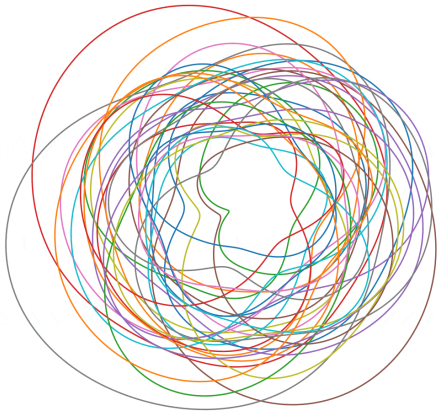


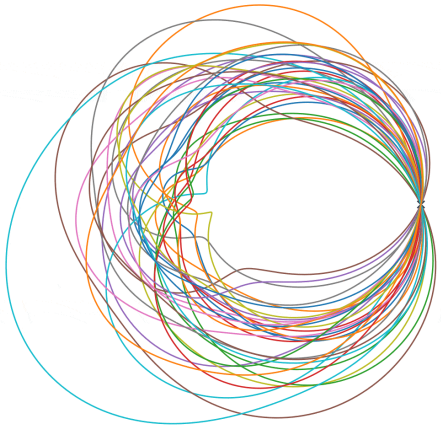
Bayesian Learning

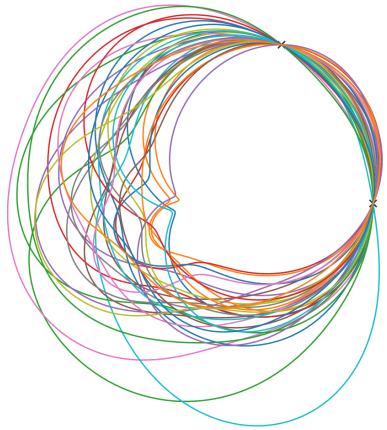




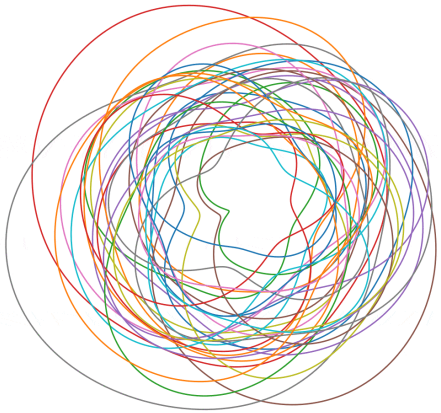
Bayesian Learning

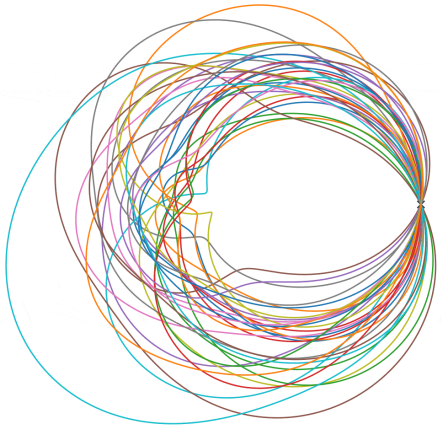


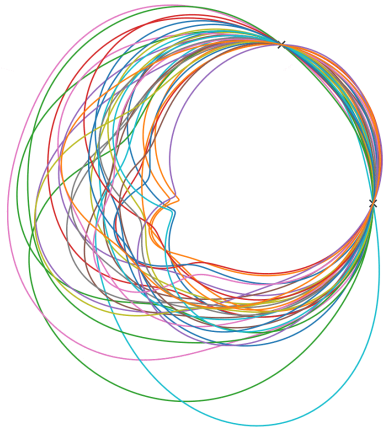


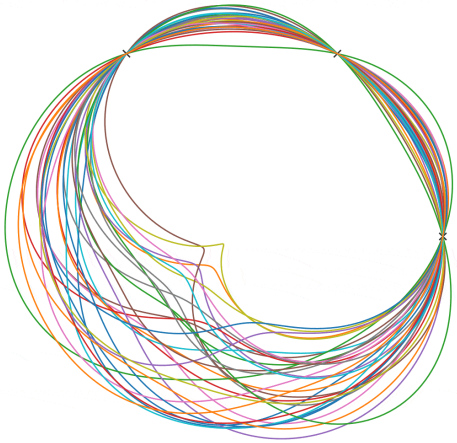


Bayesian Learning

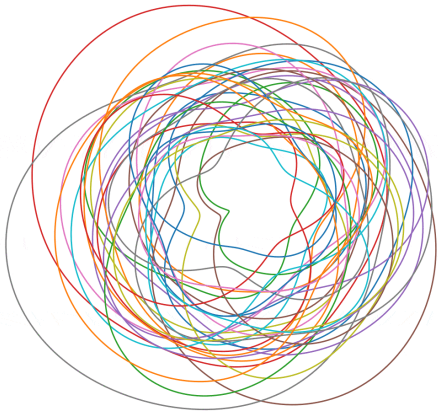


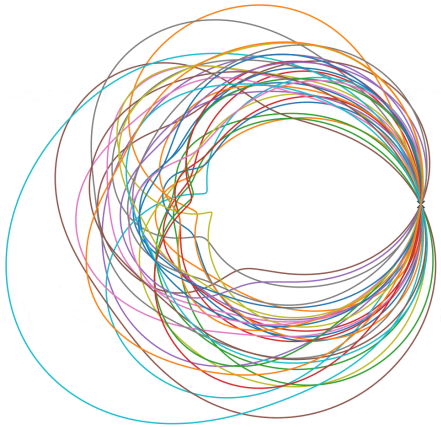


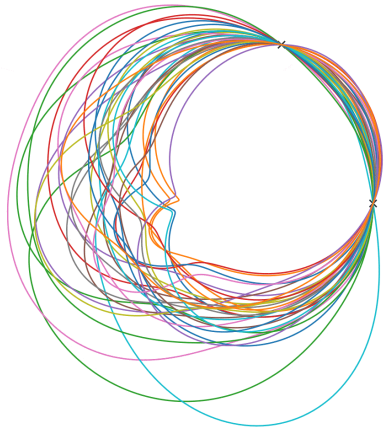


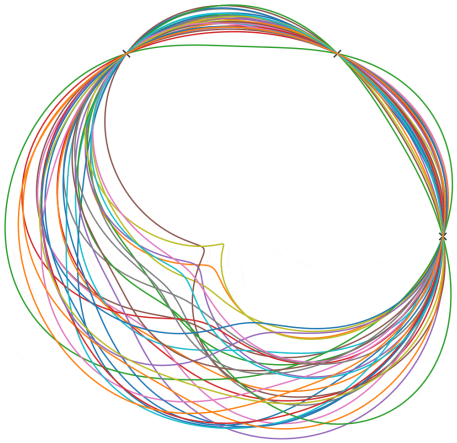


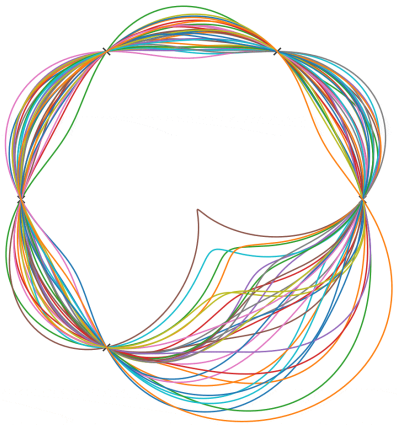
Bayesian Learning



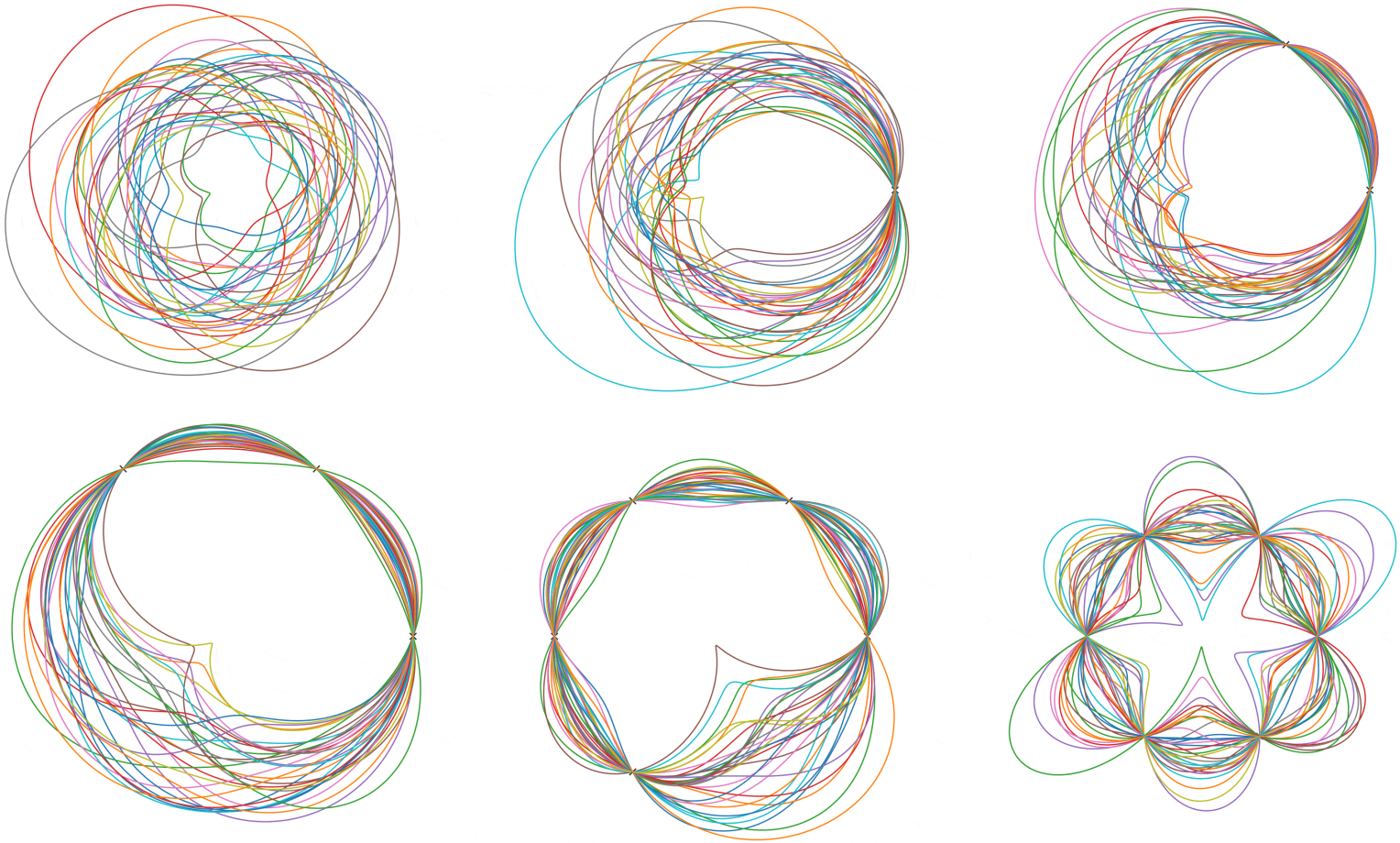








Bayesian Learning



Probabilistic Models

How to represent a probabilstic model?

Probabilistic Models

How to represent a probabilstic model?

- A table of all possible events!

Probabilistic Models

How to represent a probabilstic model?

- A table of all possible events!
- Probabilistic Graphical Models

Probabilistic Models

How to represent a probabilstic model?

- A table of all possible events!
- Probabilistic Graphical Models
- Probabilistic Programs

Probabilistic Models

Can we represent a joint density with an algorithm?

Beta-Binomial Example


```
@model function coin_toss(N, X)
    θ  ~ Beta(1.0, 1.0)

    for i in 1:N
        X[i]  ~ Bernoulli(θ)
    end
end
```

Beta-Binomial Example


```
@model function coin_toss(N, X)
    θ  ~ Beta(1.0, 1.0)

    for i in 1:N
        X[i]  ~ Bernoulli(θ)
    end
end
```

$$p(\theta, X)$$

Beta-Binomial Example


```
X = [1, 1, 1, 1, 0]
```

Beta-Binomial Example


```
X = [1, 1, 1, 1, 0]

@model function coin_toss(N, X)
    θ  ~ Beta(1.0, 1.0)

    for i in 1:N
        X[i]  ~ Bernoulli(θ)
    end
end
```

Beta-Binomial Example


```
X = [1, 1, 1, 1, 0]

@model function coin_toss(N, X)
    θ  ~ Beta(1.0, 1.0)

    for i in 1:N
        X[i]  ~ Bernoulli(θ)
    end
end
```

$$p(\theta|X)$$

Probabilistic Programs

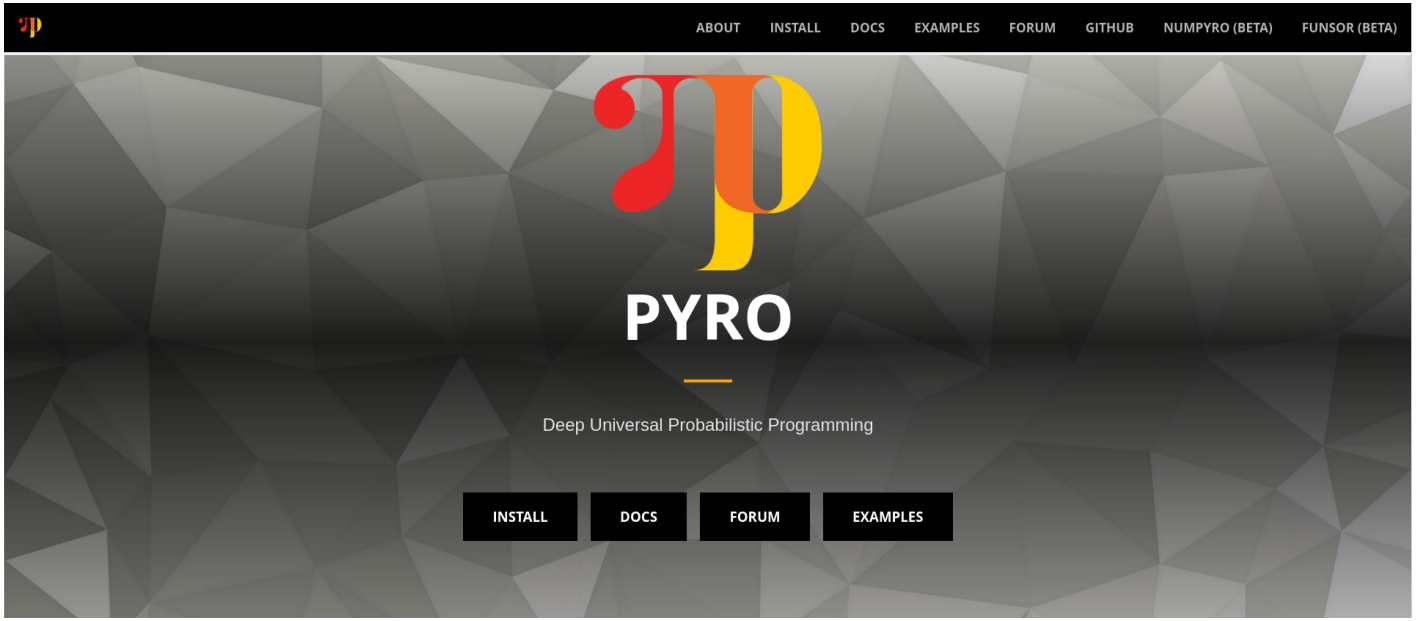
Probabilistic Programs offer more representation power than PGMs!

Probabilistic Programs

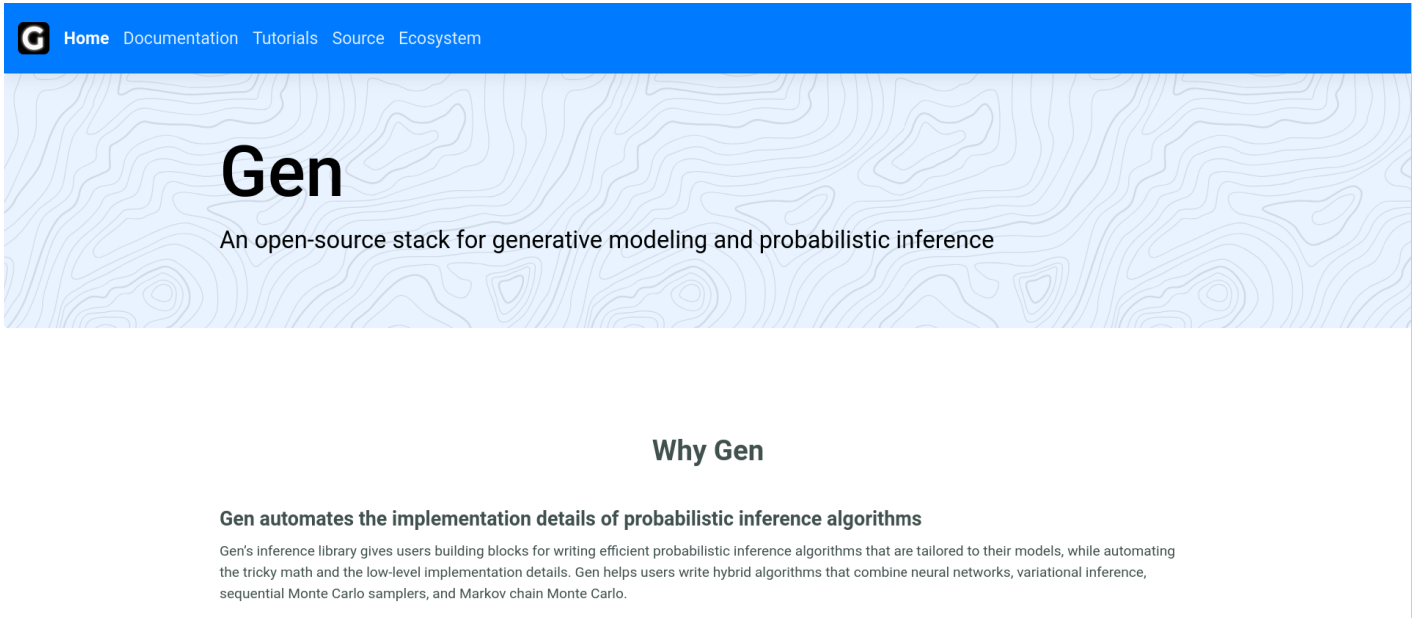
Probabilistic Programs offer more representation power than PGMs!

```
@model function program()
    T ~ Geometric(0.1)
    S = 0
    X = Vector{Any}(undef, T)
    for t ∈ 1:T
        X[t] ~ Bernoulli(0.5)
        S = S + X[t]
    end
    return S
end
```

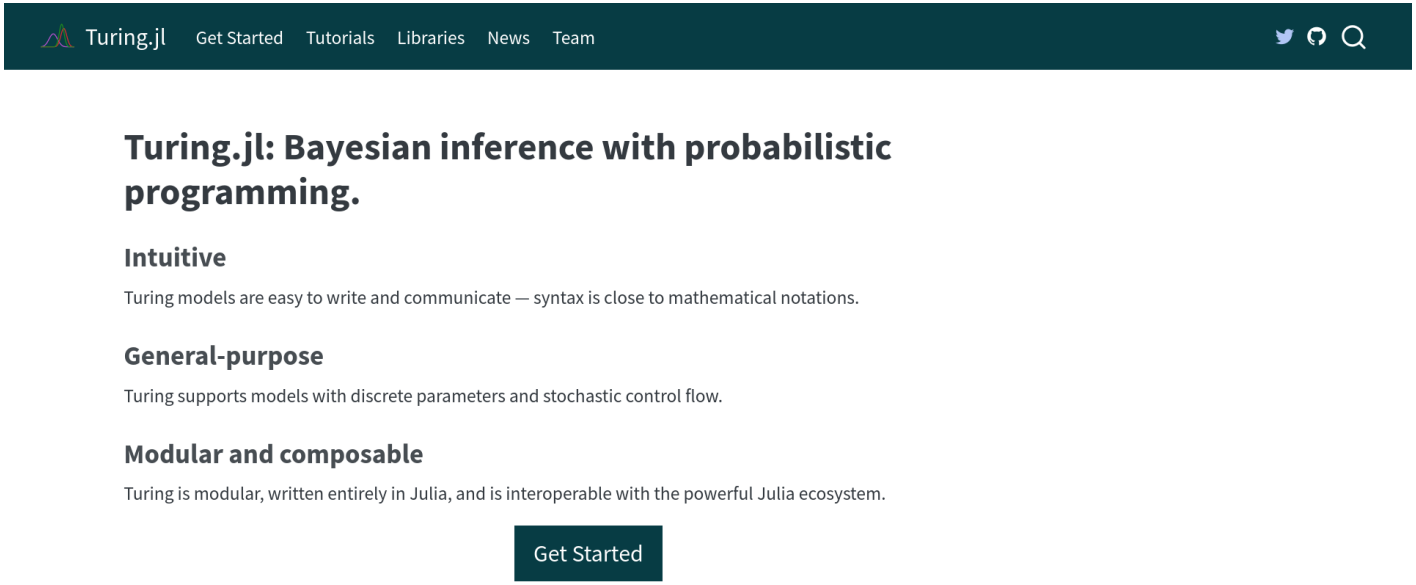
Probabilistic Programming Languages



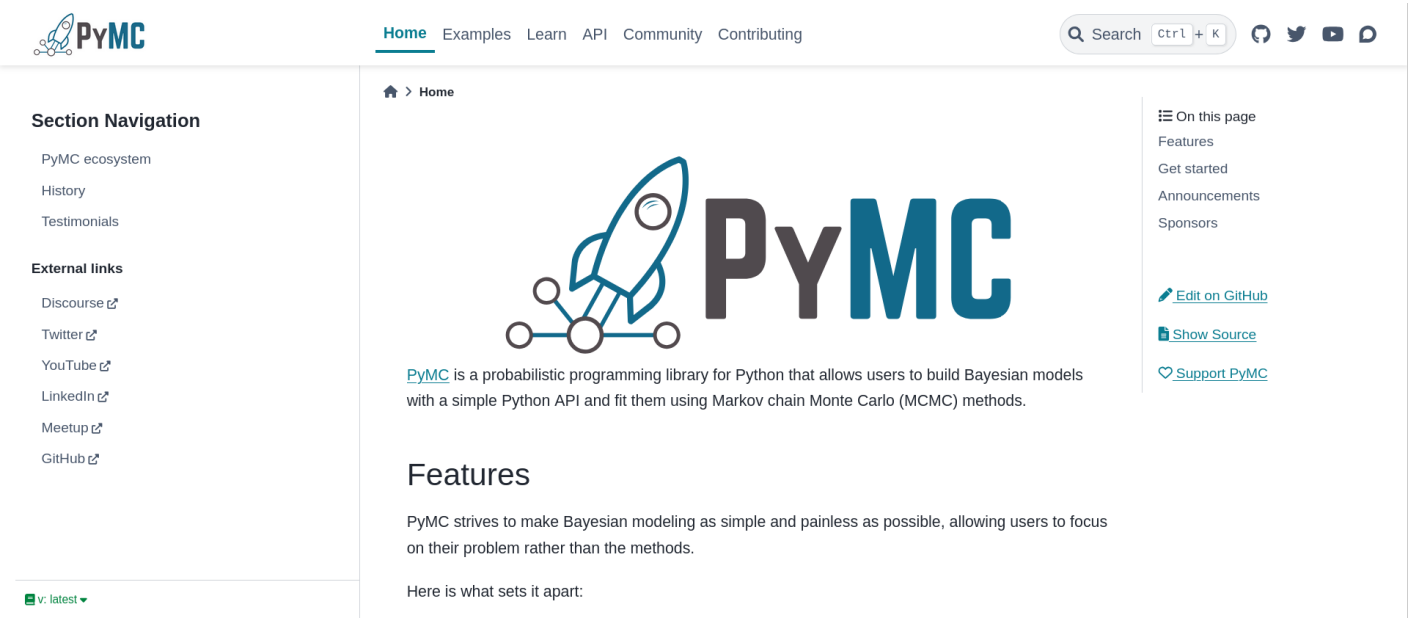
Probabilistic Programming Languages



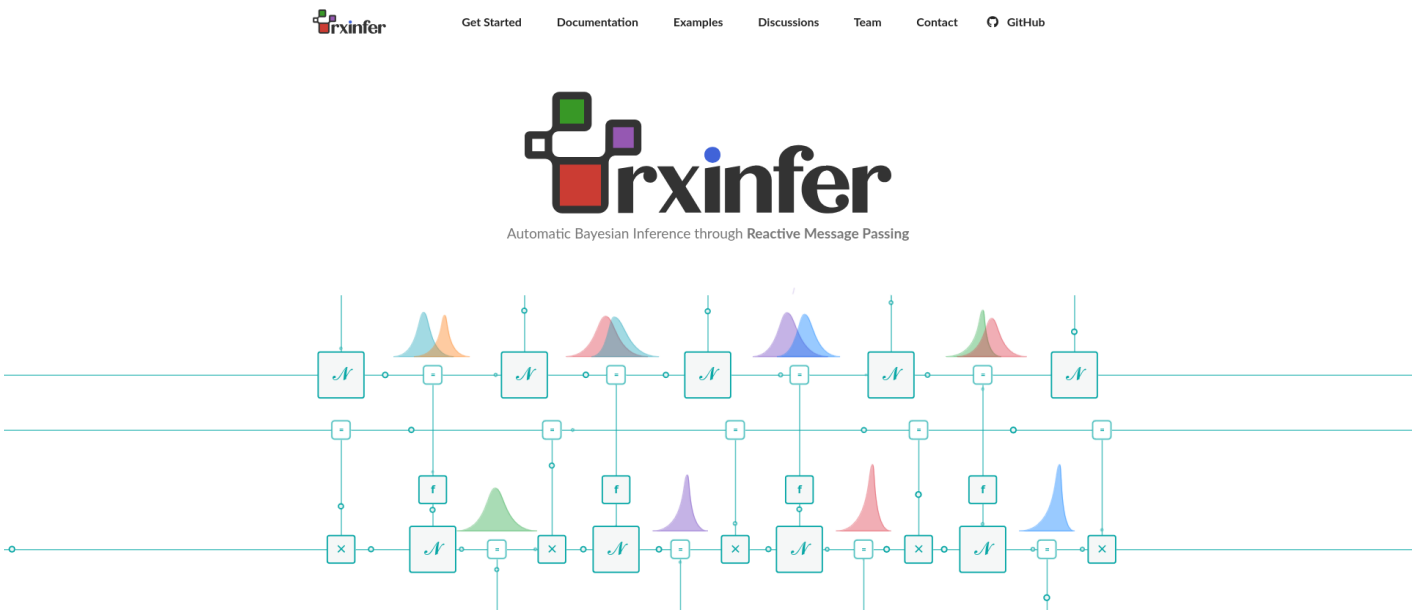
Probabilistic Programming Languages



Probabilistic Programming Languages



Probabilistic Programming Languages



Why is inference difficult?

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

Why is inference difficult?

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

To obtain $p(x)$ we have to marginalize all possible hypotheses:

Why is inference difficult?

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

To obtain $p(x)$ we have to marginalize all possible hypotheses:

$$p(x) = \int p(x, z)dz$$

Why is inference difficult?

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

To obtain $p(x)$ we have to marginalize all possible hypotheses:

$$p(x) = \int p(x, z)dz$$

Now imagine what does $p(x)$ look like if we have used something like neural networks inside the model!

Why is inference difficult?

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

To obtain $p(x)$ we have to marginalize all possible hypotheses:

$$p(x) = \int p(x, z)dz$$

Now imagine what does $p(x)$ look like if we have used something like neural networks inside the model!

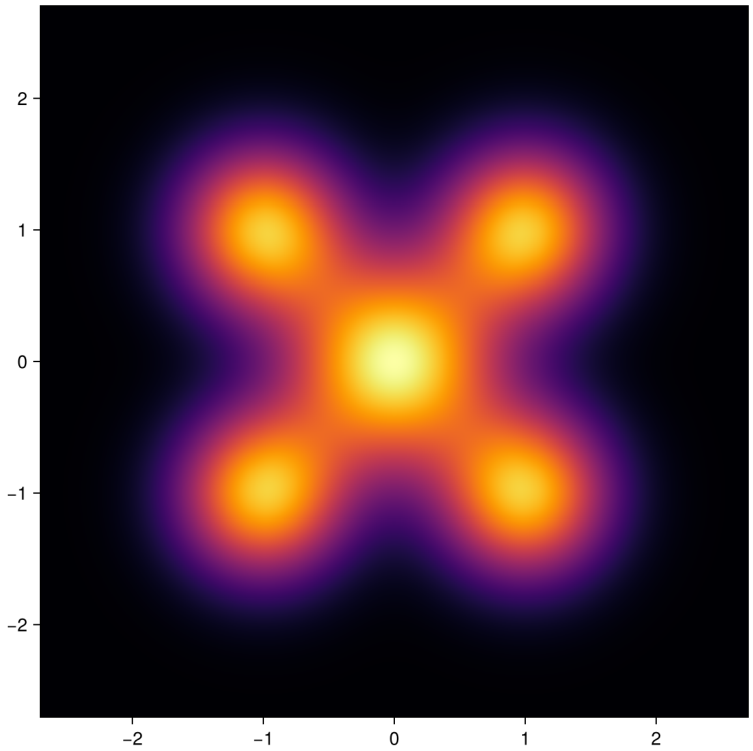
So the posterior turns out to be intractable

If we choose likelihood and prior carefully, Exact Inference is possible.

Likelihood	**Conjugate Prior**
Bernoulli	Beta
Binomial	Beta
Poisson	Gamma
Categorical	Dirichlet
Uniform	Pareto

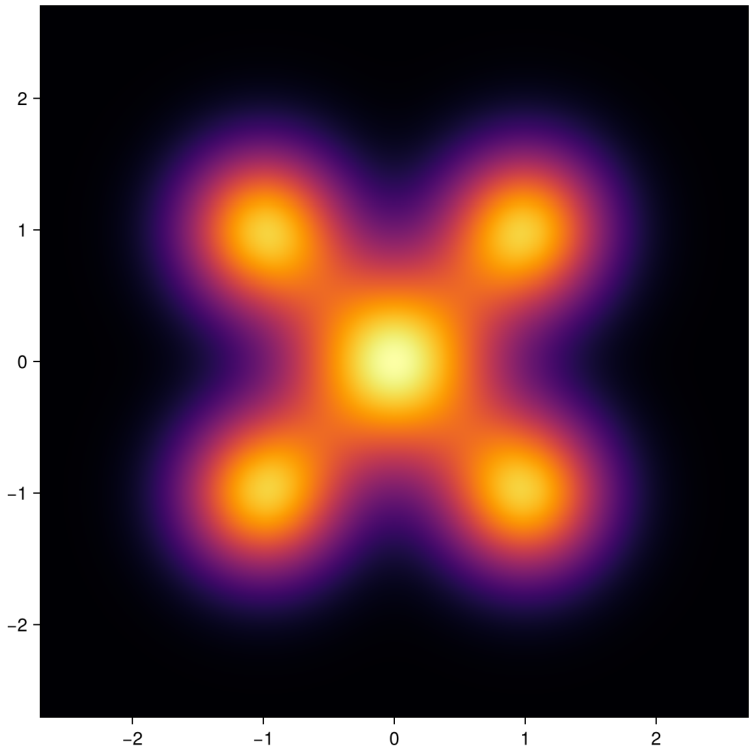
Approximate Inference Methods

Exact Inference is not possible for most of the models



Approximate Inference Methods

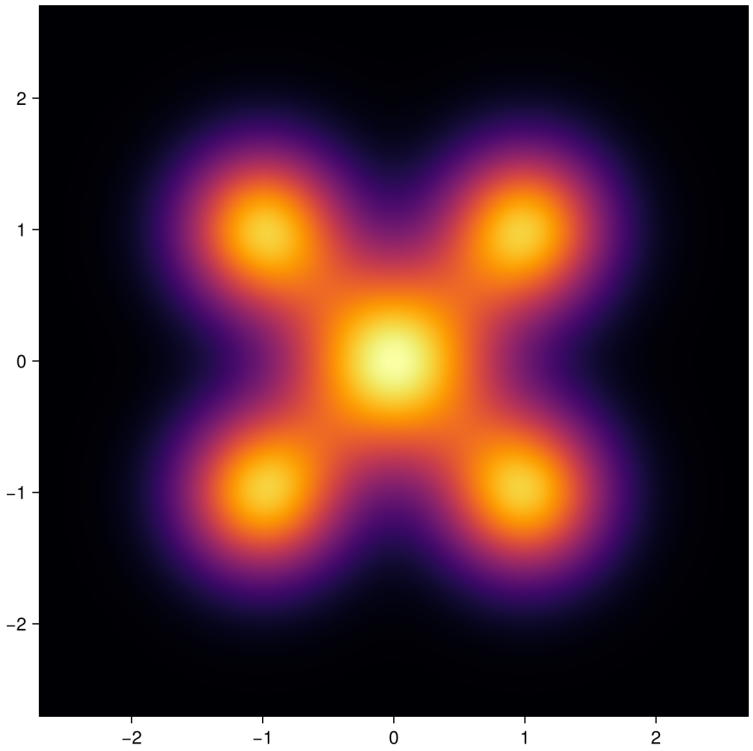
Exact Inference is not possible for most of the models



Fortunately there are some methods for approximate inference

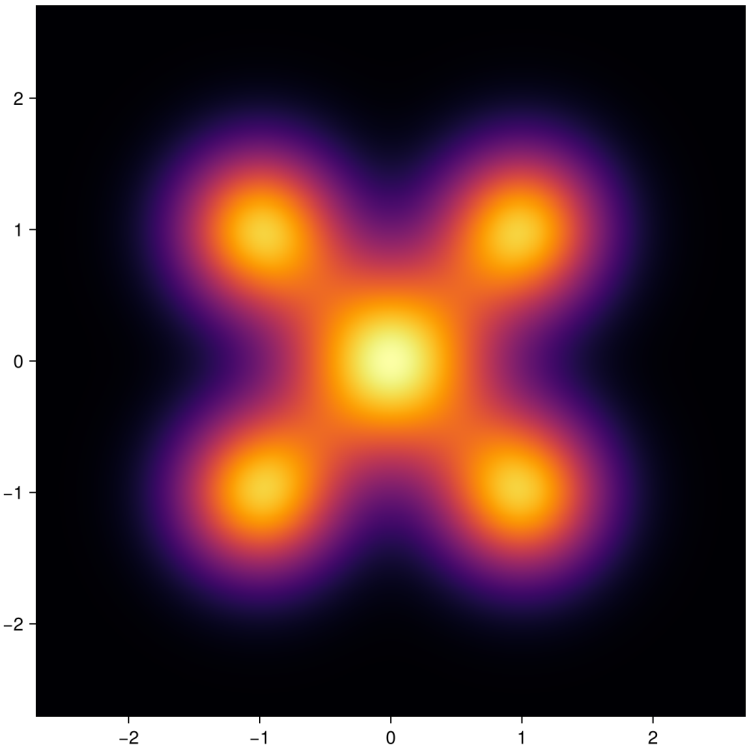
Markov Chain Monte Carlo

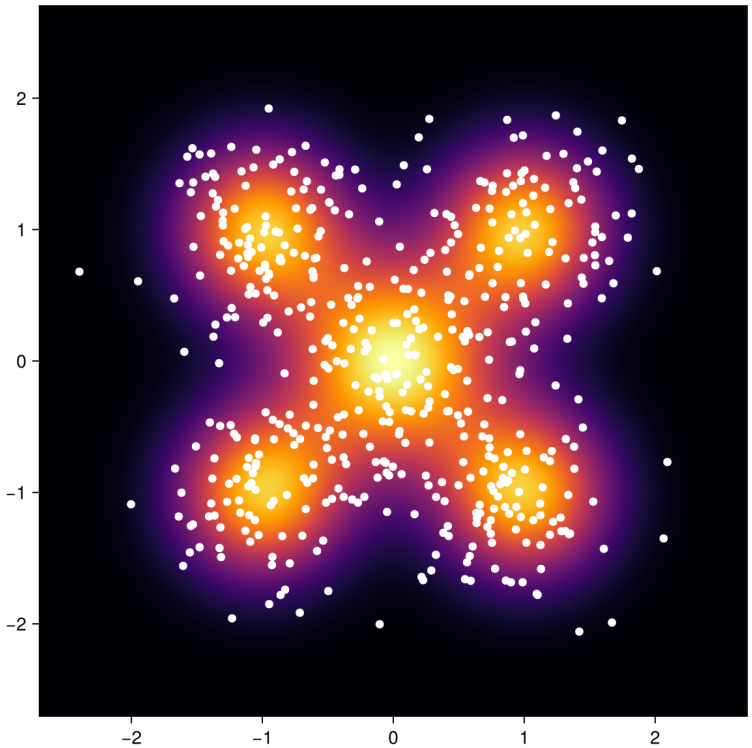
posterior $p(\boldsymbol{z}|\boldsymbol{x})$ is intractable



Markov Chain Monte Carlo

posterior $p(\boldsymbol{z}|\boldsymbol{x})$ is intractable





If we somehow generate enough number of samples from $p(\boldsymbol{z}|\boldsymbol{x})$ then we can estimate our desired quantities.

Markov Chain Monte Carlo

Consider a particle in $\mathbb{R}^n$ with an initial position (state) X_0. When the particle is in a position X_t it will move to a position X_{t+1} with probability $p(X_{t+1}|X_t)$ so we have a sequence of random variables

$$X_0, X_1, X_2, ...$$

Markov Chain Monte Carlo

Consider a particle in $\mathbb{R}^n$ with an initial position (state) X_0. When the particle is in a position X_t it will move to a position X_{t+1} with probability $p(X_{t+1}|X_t)$ so we have a sequence of random variables

$$X_0, X_1, X_2, ...$$

Such a stochastic process is called **Markov Chain**

Markov Chain Monte Carlo

Consider a particle in $\mathbb{R}^n$ with an initial position (state) X_0. When the particle is in a position X_t it will move to a position X_{t+1} with probability $p(X_{t+1}|X_t)$ so we have a sequence of random variables

$$X_0, X_1, X_2, ...$$

Such a stochastic process is called **Markov Chain**
Under some conditions after a time τ the Markov Chain will forget it's initial state and becomes **stationary**

Markov Chain Monte Carlo

Consider a particle in $\mathbb{R}^n$ with an initial position (state) X_0. When the particle is in a position X_t it will move to a position X_{t+1} with probability $p(X_{t+1}|X_t)$ so we have a sequence of random variables

$$X_0, X_1, X_2, ...$$

Such a stochastic process is called **Markov Chain**
Under some conditions after a time τ the Markov Chain will forget it's initial state and becomes **stationary**
In other words the terms in the sequence

$$X_{\tau+1}, X_{\tau+2}, X_{\tau+3}, ...$$

will be random samples from the stationary distribution of Markov Chain

Markov Chain Monte Carlo

Is it possible to construct a Markov chain that converges to a specific distribution?

Markov Chain Monte Carlo

Is it possible to construct a Markov chain that converges to a specific distribution?
We need a way to guaranty the existence of stationary distribution.

Markov Chain Monte Carlo

Definition

A Markov chain is called reversible if it satisfies the **detailed balance** equations. It means the probability of being in a state x_i then transitioning to a state x_j is equal to the probability of being in x_j and then transitioning to x_i. formally:

$$\pi(x_i)P(x_j|x_i) = \pi(x_j)P(x_i|x_j)$$

Where $\pi(x)$ is the stationary distribution

Markov Chain Monte Carlo

Definition

A Markov chain is called reversible if it satisfies the **detailed balance** equations. It means the probability of being in a state x_i then transitioning to a state x_j is equal to the probability of being in x_j and then transitioning to x_i. formally:

$$\pi(x_i)P(x_j|x_i) = \pi(x_j)P(x_i|x_j)$$

Where $\pi(x)$ is the stationary distribution

The idea is to define the transition probability $P(x'|x)$ such that it satisfies the detailed balance equations for the target distribution $\pi(x)$

Random Walk Metropolis-Hastings

So we use the detailed balance equation:

$$\pi(x)P(x'|x) = \pi(x')P(x|x')$$

Random Walk Metropolis-Hastings

So we use the detailed balance equation:

$$\pi(x)P(x'|x) = \pi(x')P(x|x')$$

$$\frac{P(x'|x)}{P(x|x')} = \frac{\pi(x')}{\pi(x)}$$

Random Walk Metropolis-Hastings

So we use the detailed balance equation:

$$\pi(x)P(x'|x) = \pi(x')P(x|x')$$

$$\frac{P(x'|x)}{P(x|x')} = \frac{\pi(x')}{\pi(x)}$$

Now let's break the transition into two steps. First, we propose a new state with probability $g(x'|x)$. Next deciding whether we move to the proposed state x' according to an acceptance distribution $A(x', x)$

Random Walk Metropolis-Hastings

So we use the detailed balance equation:

$$\pi(x)P(x'|x) = \pi(x')P(x|x')$$

$$\frac{P(x'|x)}{P(x|x')} = \frac{\pi(x')}{\pi(x)}$$

Now let's break the transition into two steps. First, we propose a new state with probability $g(x'|x)$. Next deciding whether we move to the proposed state x' according to an acceptance distribution $A(x', x)$

$$P(x'|x) = g(x'|x)A(x', x)$$

Random Walk Metropolis-Hastings

Now we rewrite the equation

$$\frac{A(x', x)}{A(x, x')} = \frac{\pi(x')}{\pi(x)} \frac{g(x|x')}{g(x'|x)}$$

Random Walk Metropolis-Hastings

Now we rewrite the equation

$$\frac{A(x', x)}{A(x, x')} = \frac{\pi(x')}{\pi(x)} \frac{g(x|x')}{g(x'|x)}$$

Metropolis-Hastings algorithm defines an acceptance ratio that satisfies the above condition

$$A(x', x) = min(1, \frac{\pi(x')}{\pi(x)} \frac{g(x|x')}{g(x'|x)})$$

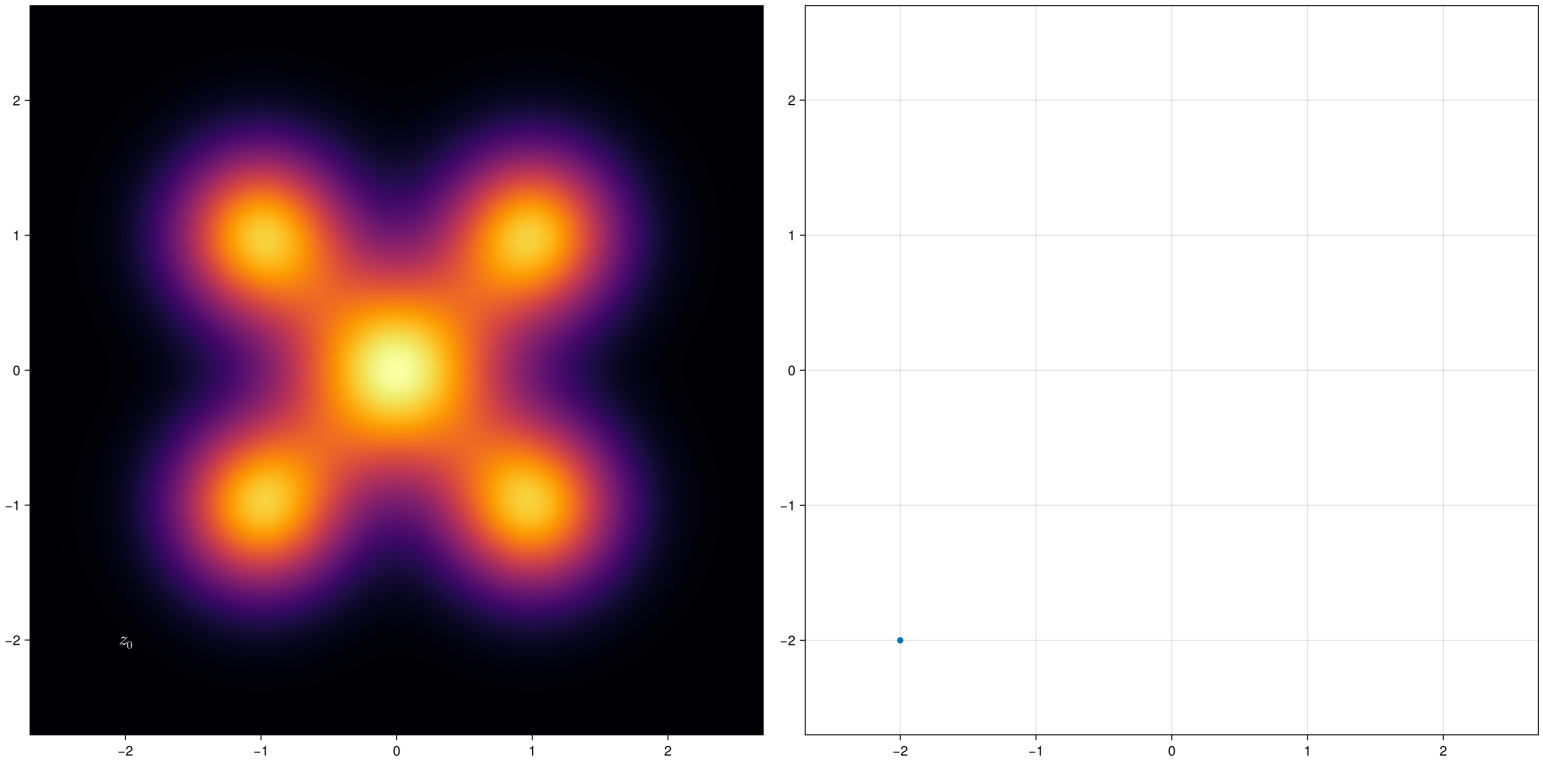
Random Walk Metropolis-Hastings

The funny fact is that for computing $\frac{\pi(x')}{\pi(x)} \frac{g(x|x')}{g(x'|x)}$ we don't need to know the normalization constant of $\pi(x)$!

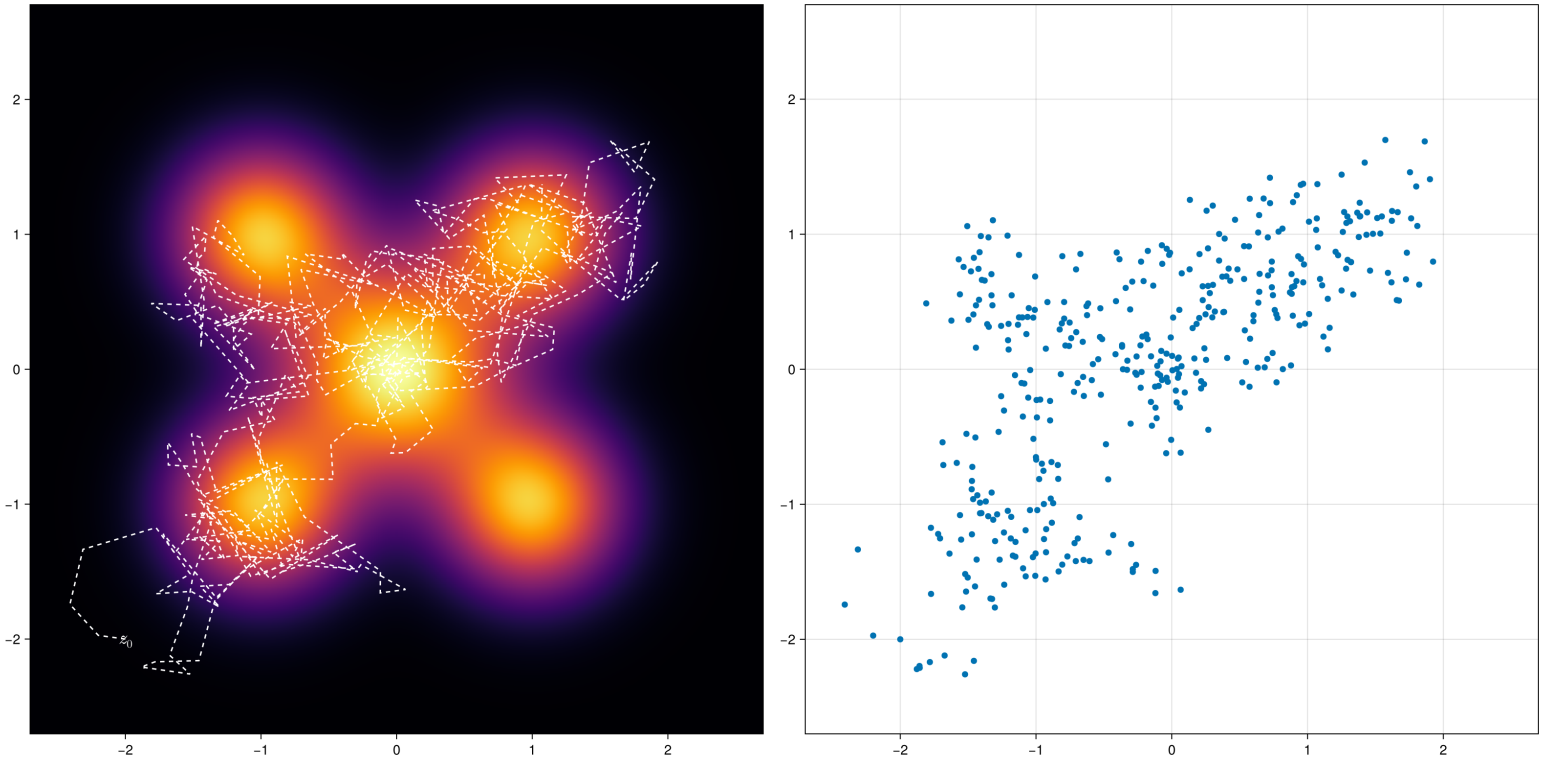
Random Walk Metropolis-Hastings


```
function random_walk_metropolis(logπ, z₀, σ, T)
    samples = [z₀]
    z = z₀
    for τ ∈ 1:T
        y = z .+ σ * randn(size(z))
        α = exp(logπ(y) - logπ(z))
        if rand() < α
            z = y
            push!(samples, z)
        end
    end
    return samples
end
```

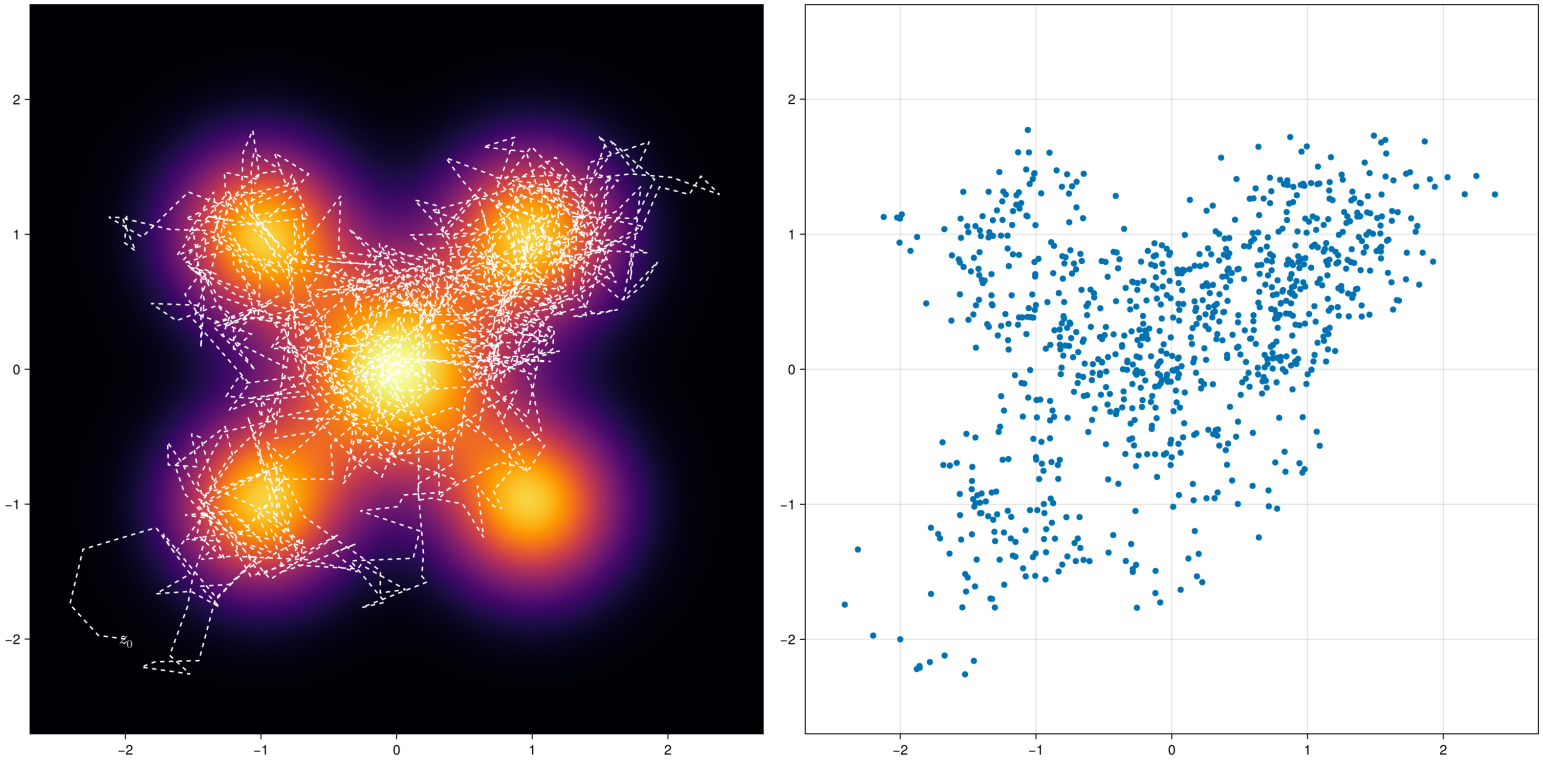
Random Walk Metropolis-Hastings



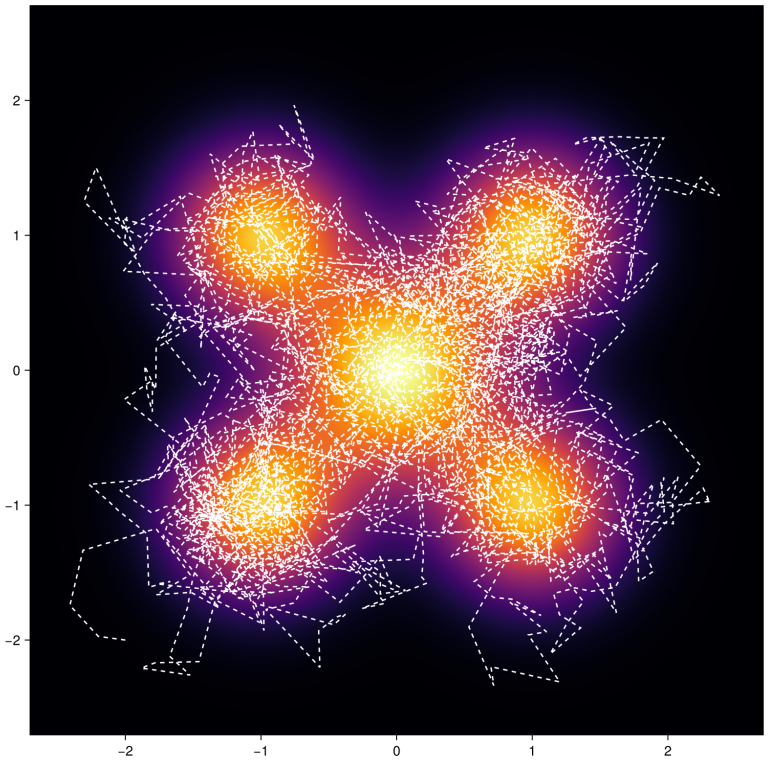
Random Walk Metropolis-Hastings

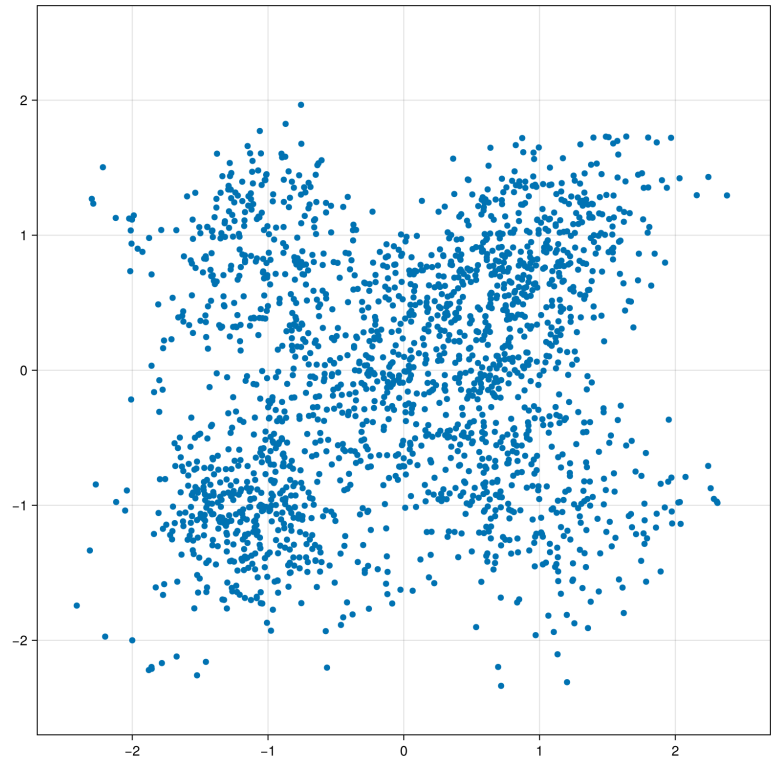


Random Walk Metropolis-Hastings



Random Walk Metropolis-Hastings





Random Walk Metropolis-Hastings

- The acceptance probability in higher dimensions is not satisfying

Random Walk Metropolis-Hastings

- The acceptance probability in higher dimensions is not satisfying
- RWMH treats the target density as blackbox

Random Walk Metropolis-Hastings

- The acceptance probability in higher dimensions is not satisfying
- RWMH treats the target density as blackbox

Advanced MCMC methods exploit properties of the target density!

Langevin Monte Carlo

$$\mathrm{d}X_t = \nabla \log \pi(X_t) + \sqrt{2}\,\mathrm{d}W_t$$

Overdamped Langevin Dynamics

Langevin Monte Carlo

$$\mathrm{d}X_t = \nabla \log \pi(X_t) + \sqrt{2}\,\mathrm{d}W_t$$

Overdamped Langevin Dynamics

```
function langevin_dynamics(logπ, z, τ)
    ζ =  sqrt(2τ) * randn(size(z))
    ∇logπ = ForwardDiff.gradient(logπ, z)
    z .+ τ * ∇logπ .+ ζ
end
```

Langevin Monte Carlo

$$\mathrm{d}X_t = \nabla \log \pi(X_t) + \sqrt{2}\,\mathrm{d}W_t$$

Overdamped Langevin Dynamics

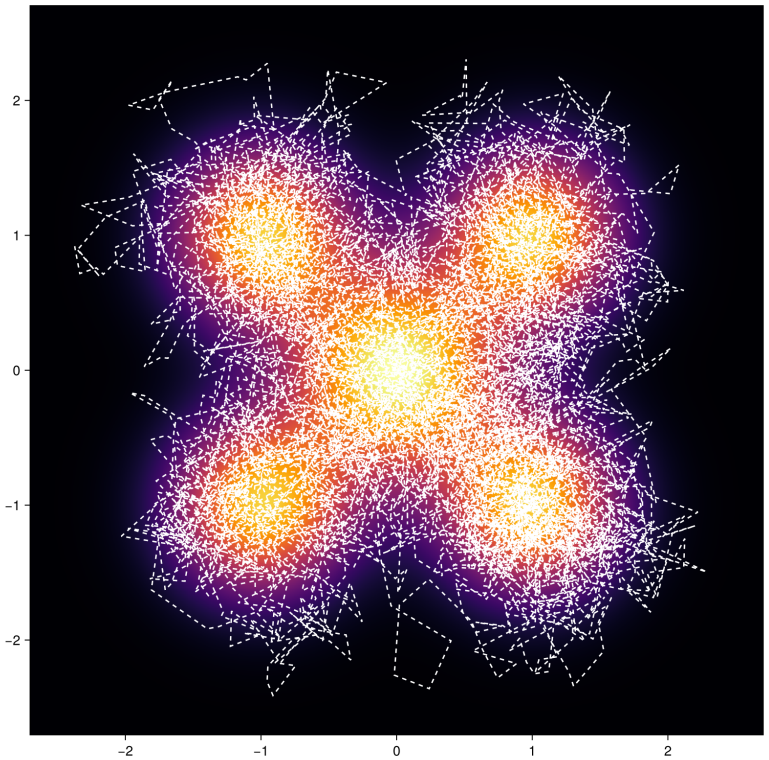
```
function langevin_dynamics(logπ, z, τ)
    ζ =  sqrt(2τ) * randn(size(z))
    ∇logπ = ForwardDiff.gradient(logπ, z)
    z .+ τ * ∇logπ .+ ζ
end
```

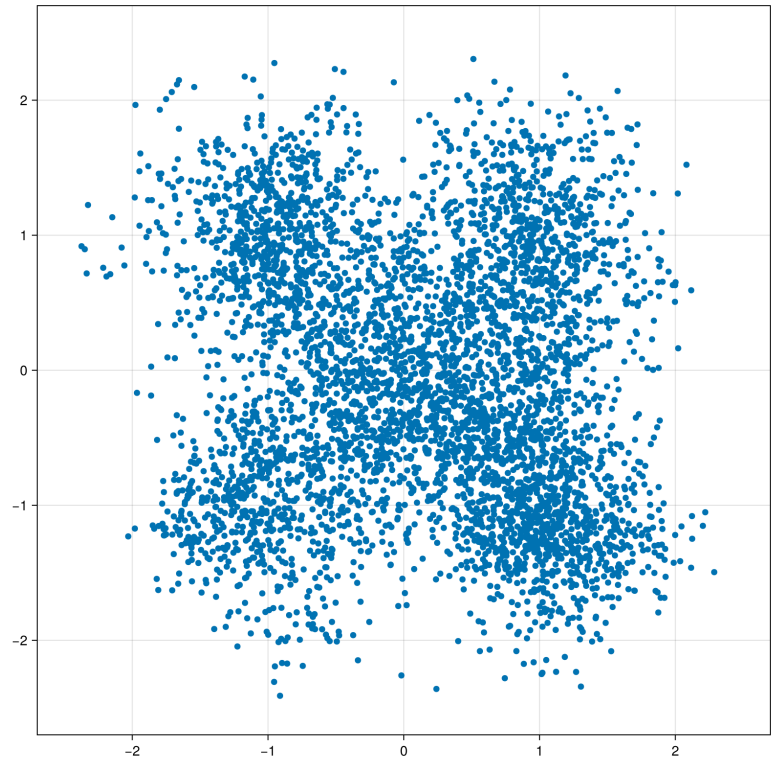
To derive **Langevin Monte Carlo** we adjust this dynamics using Metropolis-Hastings

Langevin Monte Carlo


```
function langevin_monte_carlo(logπ, z₀, τ, T)
    z = z₀
    samples = [z₀]
    for t ∈ 1:T
        ζ = sqrt(2τ) * randn(size(z))
        ∇logπ = ForwardDiff.gradient(logπ, z)
        y = z .+ τ * ∇logπ .+ ζ
        ∇logπ_y = ForwardDiff.gradient(logπ, y)
        logg_y_z = -1/(4τ) * norm(y .- z .- τ * ∇logπ)^2
        logg_z_y = -1/(4τ) * norm(z .- y .- τ * ∇logπ_y)^2
        α = logπ(y) + logg_z_y - logπ(z) - logg_y_z
        if rand() < exp(α)
            z = y
            push!(samples, z)
        end
    end
    return samples
end
```

Langevin Monte Carlo





Research Trends

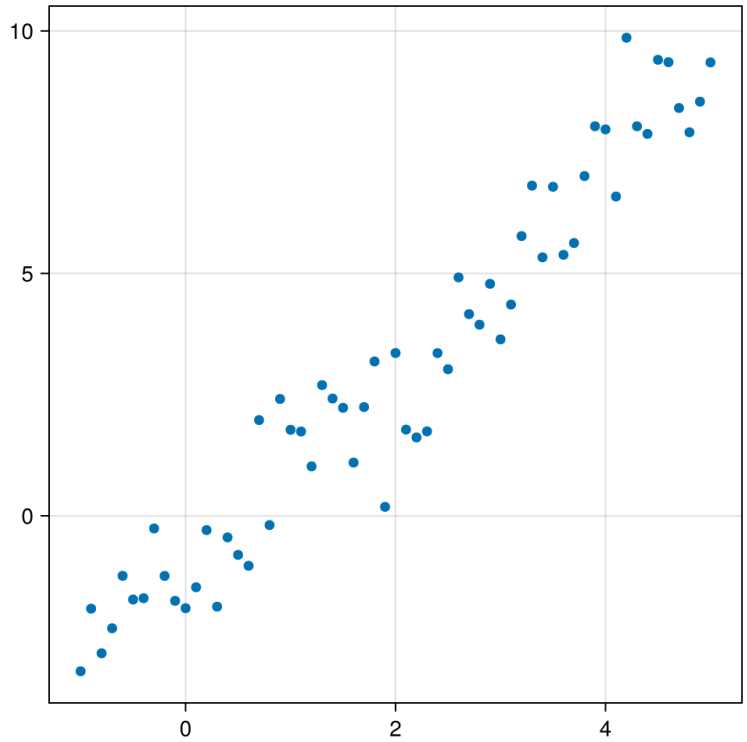
- alternatives for the base dynamics (HMC)
- adaptive samplers (NUTS)
- exploiting other properties of the target density
- compositional samplers
- SMC, particle filter

Useful Tools

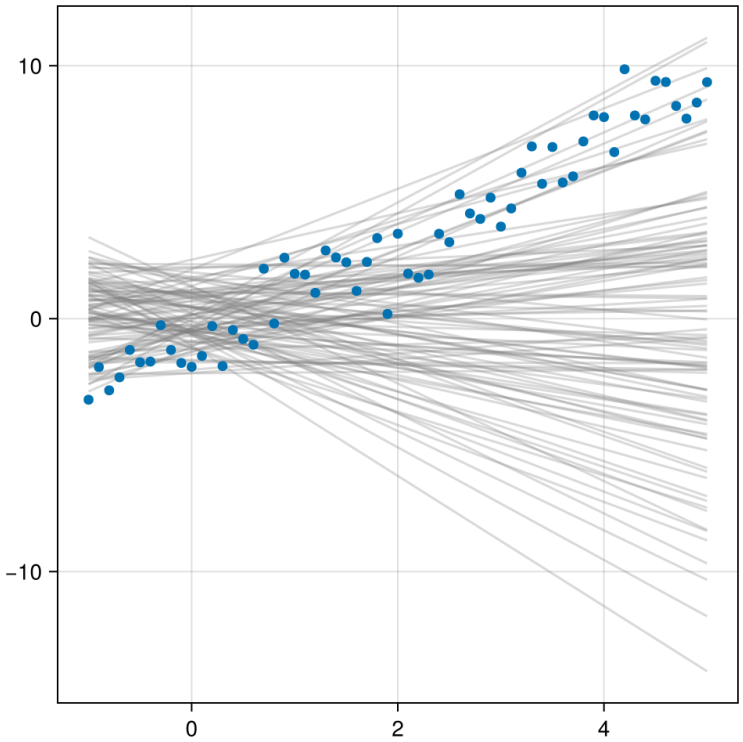
BlackJAX: Composable Bayesian Inference in JAX

- effiecient implementation of samplers
- composable samplers
- GPU acceleration
- suitable for designing PPLs or using with exisiting ones

Example : Bayesian Linear Regression



Example : Bayesian Linear Regression



Example : Bayesian Linear Regression

Probabilistic Program:

```
@model function bayesian_regression(X, y)
    α ~ Normal(0.0, 1.0)
    β ~ Normal(0.0, 1.0)

    for i ∈ eachindex(y)
        y[i] ~ Normal(α * X[i] + β, 1.0)
    end
end
```

Example : Bayesian Linear Regression

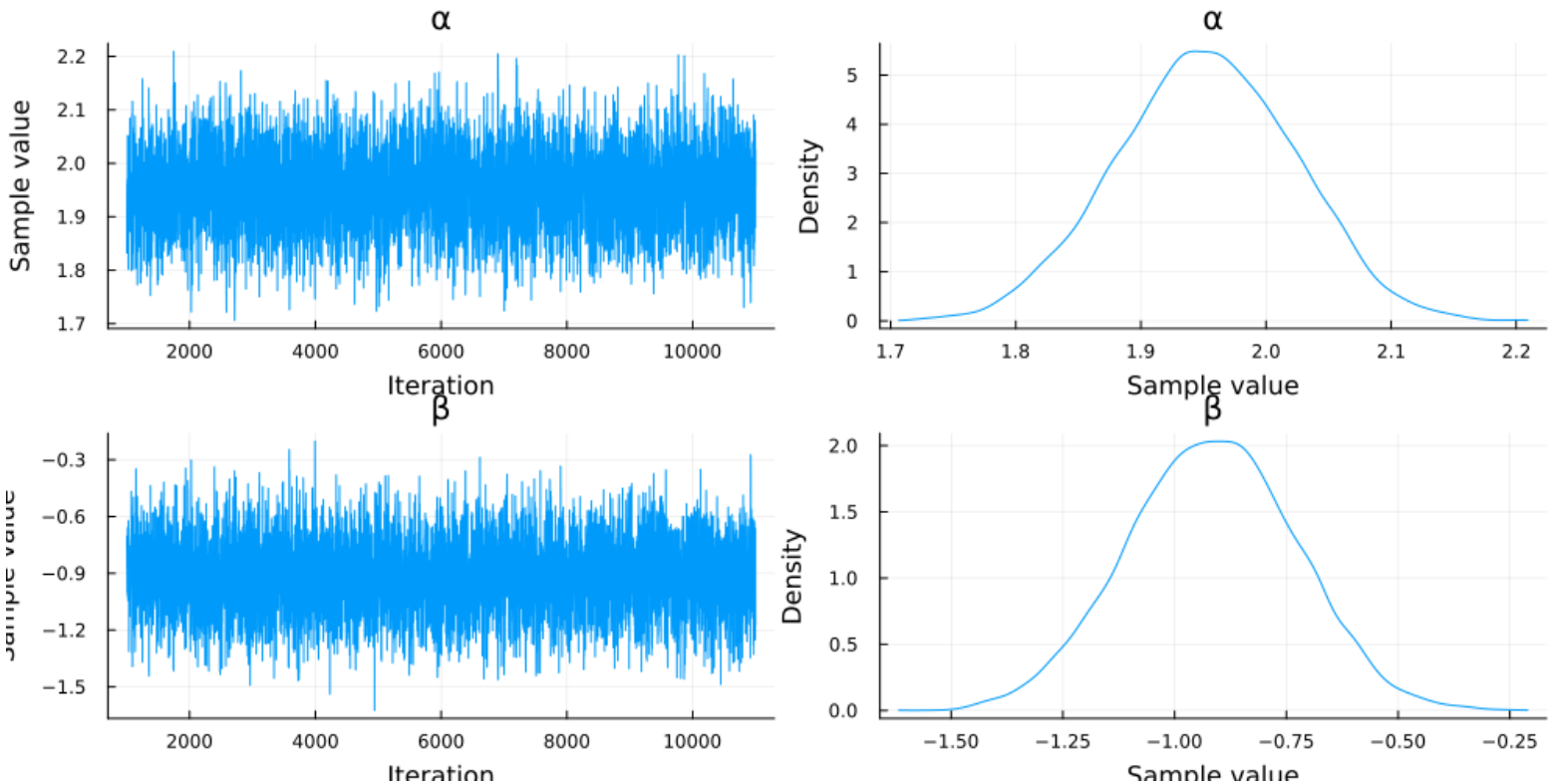
Probabilistic Program:

```
@model function bayesian_regression(X, y)
    α ~ Normal(0.0, 1.0)
    β ~ Normal(0.0, 1.0)

    for i ∈ eachindex(y)
        y[i] ~ Normal(α * X[i] + β, 1.0)
    end
end
```

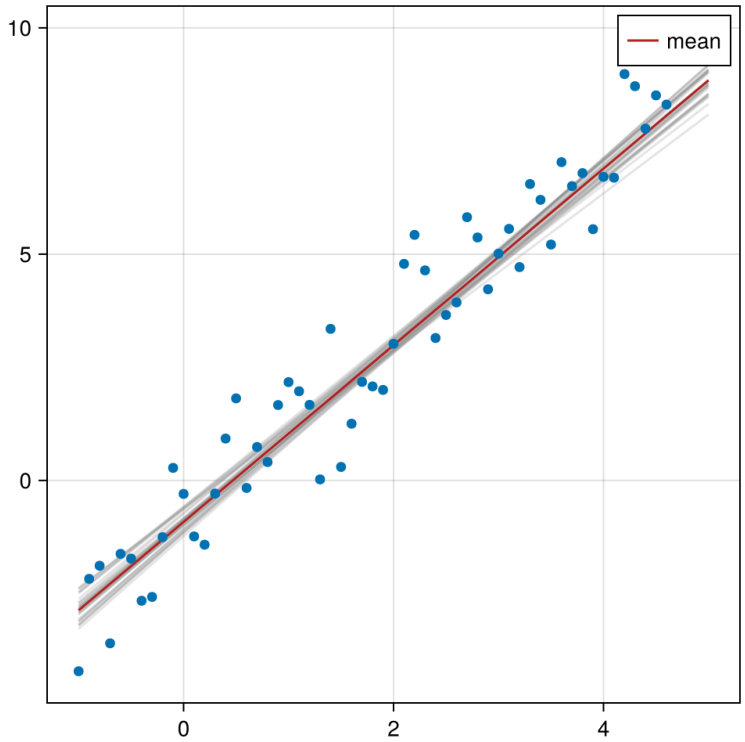
Inference:

```
ch = sample(bayesian_regression(x, y), NUTS(), 10000)
```

Example : Bayesian Linear Regression



Example : Bayesian Linear Regression



Example : Bayesian Neural Network

Let $f(x; \boldsymbol{w})$ be a neural network with three layers and sigmoid activation:

$$\boldsymbol{w} = [\boldsymbol{w}^{L1}, \boldsymbol{w}^{L2}, \boldsymbol{w}^{L3}]$$

$$f(x; \boldsymbol{w}) = \sigma(\sigma(\sigma(x\boldsymbol{w}^{L1})\boldsymbol{w}^{L2})\boldsymbol{w}^{L3})$$

Example : Bayesian Neural Network

Let $f(x; \boldsymbol{w})$ be a neural network with three layers and sigmoid activation:

$$\boldsymbol{w} = [\boldsymbol{w}^{L1}, \boldsymbol{w}^{L2}, \boldsymbol{w}^{L3}]$$

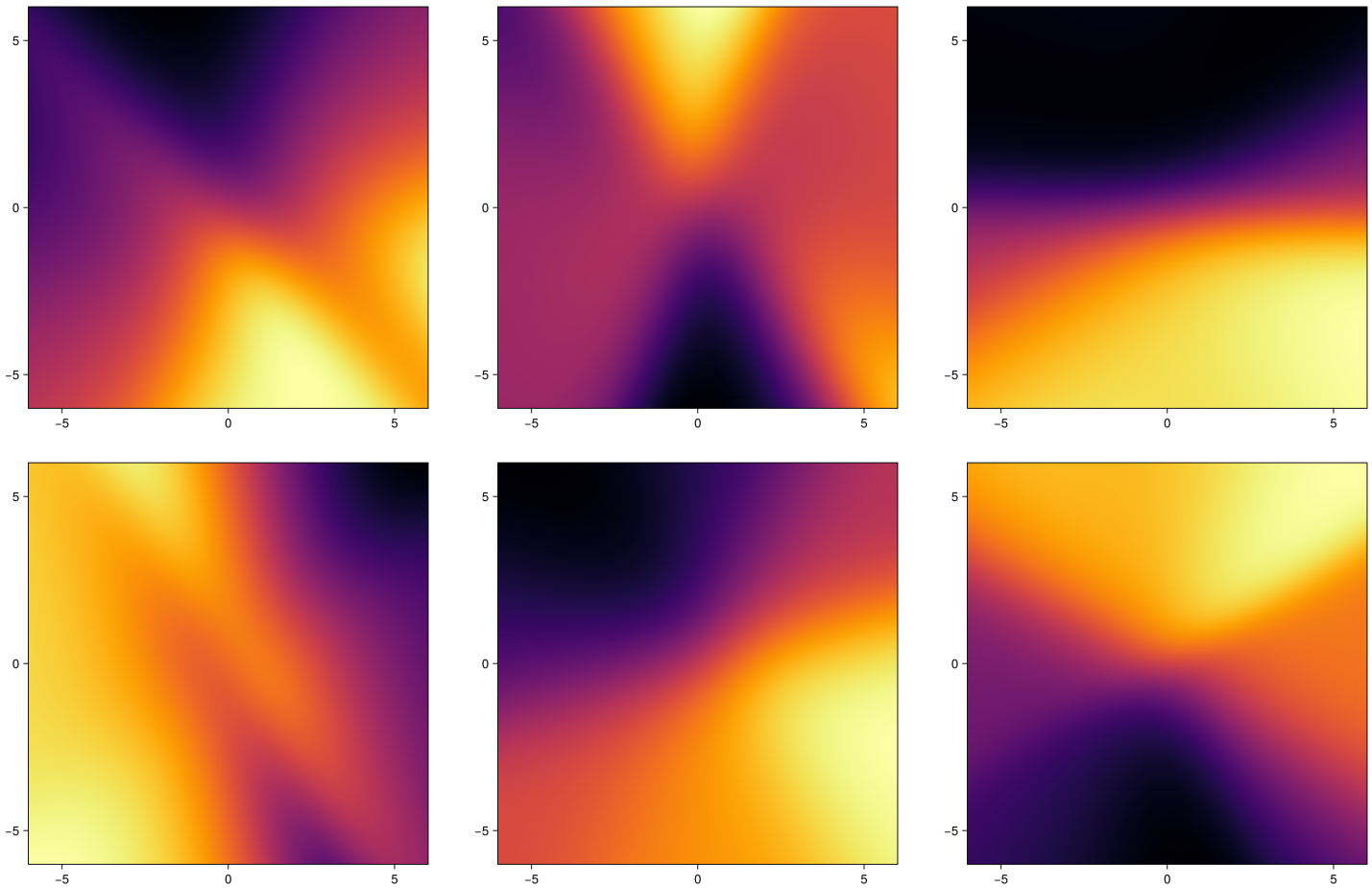
$$f(x; \boldsymbol{w}) = \sigma(\sigma(\sigma(x\boldsymbol{w}^{L1})\boldsymbol{w}^{L2})\boldsymbol{w}^{L3})$$

We define a Multivariavte Normal prior over $\boldsymbol{w}$

$$\boldsymbol{w} \sim \mathcal{N}(0, I)$$

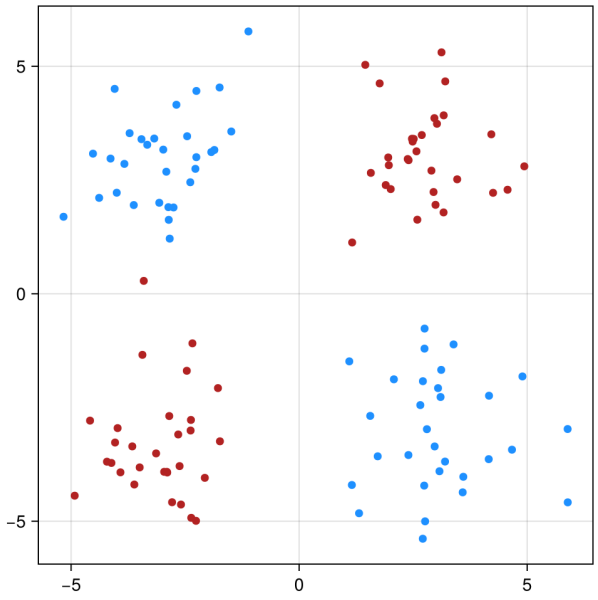
Example : Bayesian Neural Network

Some samples from $p(\boldsymbol{w})$



Example : Bayesian Neural Network

Now we update our belief about network weights using the below dataset



$$y \sim \text{Bernoulli}(f(x; \boldsymbol{w}))$$

Example : Bayesian Neural Network

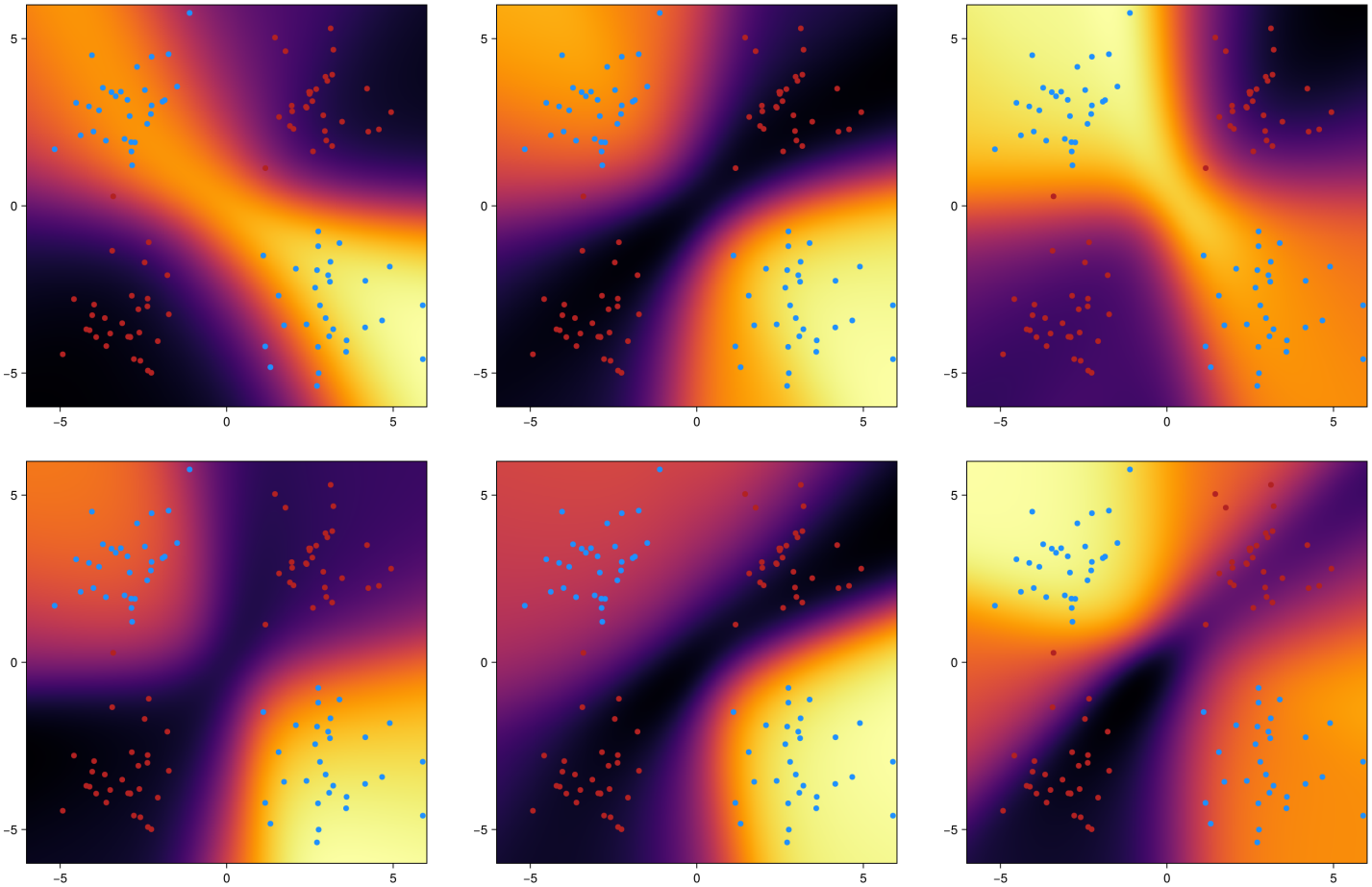

```
function neural_network(X, L1, L2, L3, H_dim)
    H = X * reshape(L1, (2, H_dim))
    H = sigmoid.(H)

    H = H * reshape(L2, (H_dim, H_dim))
    H = sigmoid.(H)

    O = H * reshape(L3, (H_dim, 1))
    O = sigmoid.(O)

    return O
end
```

Example : Bayesian Neural Network


```
@model function bayesian_neural_network(X, y, σ, H_dim)
    n_L1 = 2 * H_dim
    n_L2 = H_dim * H_dim
    n_L3 = H_dim

    Σ(n) = Diagonal(abs2.(σ .* ones(n)))

    L1 ~ MvNormal(zeros(n_L1), Σ(n_L1))
    L2 ~ MvNormal(zeros(n_L2), Σ(n_L2))
    L3 ~ MvNormal(zeros(n_L3), Σ(n_L3))

    O = neural_network(X, L1, L2, L3, H_dim)

    for i in eachindex(y)
        y[i] ~ Bernoulli(O[:][i])
    end
end
```

Example : Bayesian Neural Network

Some samples from $p(\boldsymbol{w}|y)$

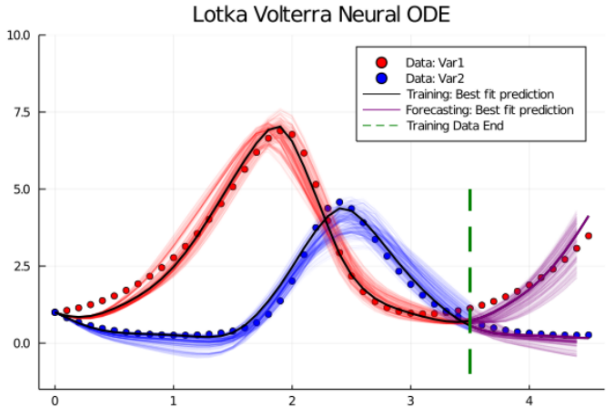


Bayesian Neural Ordinary Differential Equations

Imagine we have a differential equation with unknown parameters.

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -\alpha u_1 - \beta u_1 u_2$$
$$\frac{\mathrm{d}u_2}{\mathrm{d}t} = -\delta u_2 + \gamma u_1 u_2$$

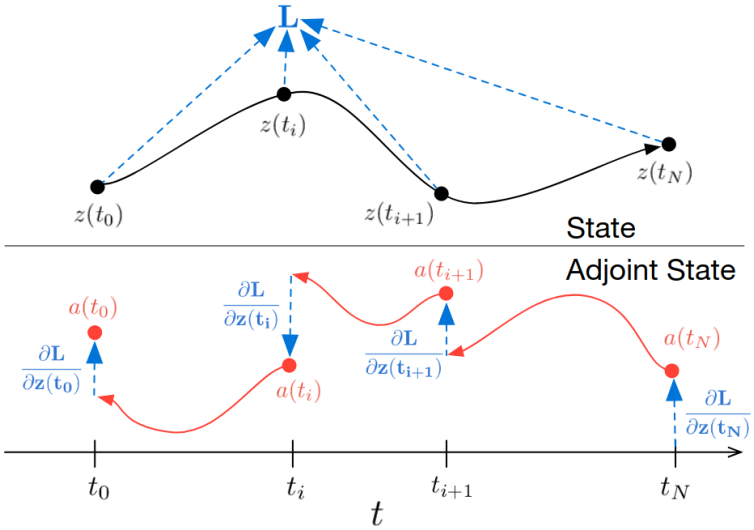
1

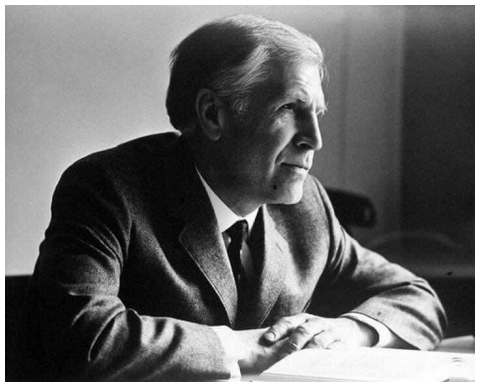


[1] Bayesian Neural Ordinary Differential Equations, arXiv:2012.07244

Bayesian Neural Ordinary Differential Equations

How to differentiate through an ODE solver !?[2]





Lev Pontryagin

(1908-1988)

[2] Neural Ordinary Differential Equations, arXiv:1806.07366

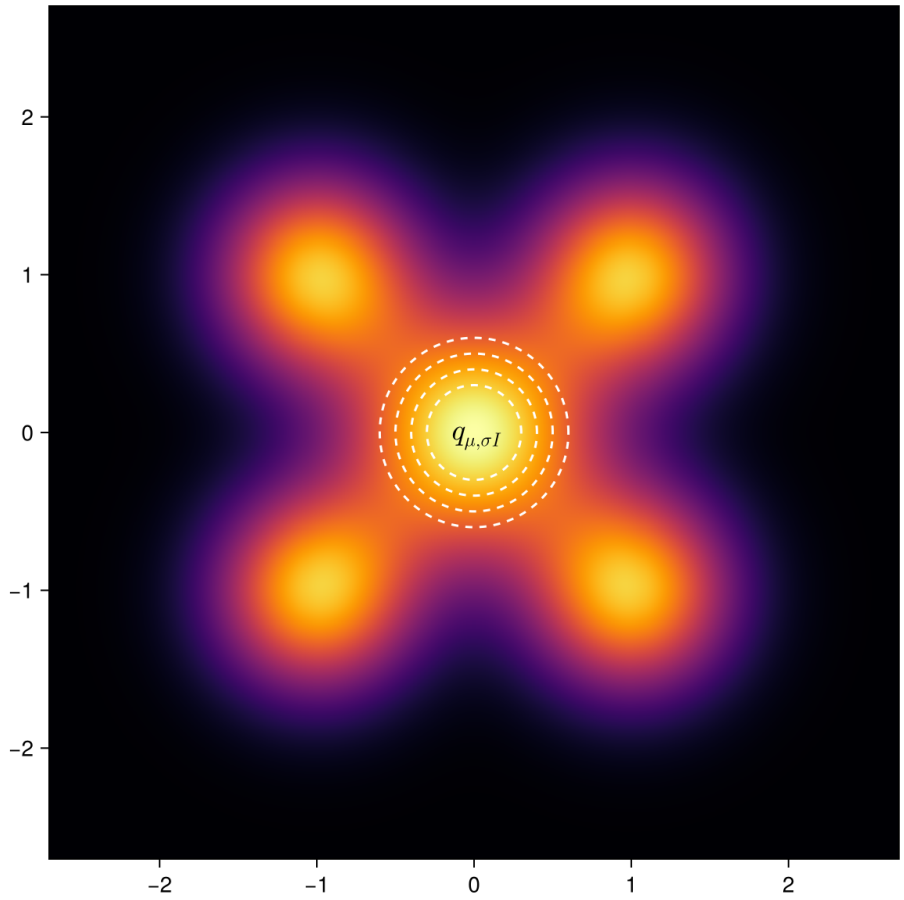
Useful Tools

SciML: Open Source Software for Scientific Machine Learning

- Advanced differential equation solvers
- Differentiable ODE solvers compatible with Turing

Variational Inference

Another idea is trying to find a tractable density $q(z; \lambda)$ as close as possible to the posterior.



Variational Inference

$$
\begin{aligned}
\log p(x) &= \log \int p(x, z) dz \\
&= \log \int \frac{p(x, z) q(z; \lambda)}{q(z; \lambda)} dz \\
&= \log \mathbb{E}_{q(z;\lambda)} \frac{p(x, z)}{q(z; \lambda)} \\
&\geq \mathbb{E}_{q(z;\lambda)} \log \frac{p(x, z)}{q(z; \lambda)}
\end{aligned}
$$

Variational Inference

$$\begin{aligned}
\log p(x) &= \log \int p(x,z)dz \\
&= \log \int \frac{p(x,z)q(z;\lambda)}{q(z;\lambda)}dz \\
&= \log \mathbb{E}_{q(z;\lambda)} \frac{p(x,z)}{q(z;\lambda)} \\
&\geq \mathbb{E}_{q(z;\lambda)} \log \frac{p(x,z)}{q(z;\lambda)}
\end{aligned}$$

The last term above is called **E**vidence **L**ower **BO**und:

$$\mathcal{L}(\lambda) = \mathbb{E}_{q(z;\lambda)} \log \frac{p(x,z)}{q(z;\lambda)}$$

Variational Inference

$$\log p(x) = \mathcal{L}(\lambda) + D_{KL}(q||p)$$

Variational Inference

$$\log p(x) = \mathcal{L}(\lambda) + D_{KL}(q||p)$$

Where D_{KL} is the Kullback–Leibler divergence

Variational Inference

$$\log p(x) = \mathcal{L}(\lambda) + D_{KL}(q||p)$$

Where D_{KL} is the Kullback–Leibler divergence

Maximizing $\mathcal{L}(\lambda)$ is equivalent to minimizing $D_{KL}(q||p)$

Research Trends

- How to pick a variational distribution?
- Is minimizing D_{KL} a good idea ?!
- Can we somehow combine sampling and optimization?

Differentiable Programming

The moral of the story!

Advanced inference methods require our program to be differentiable

Differentiable Programming

The moral of the story!

Advanced inference methods require our program to be differentiable

How to write a probabilistic program in a differentiable way?

Differentiable Programming

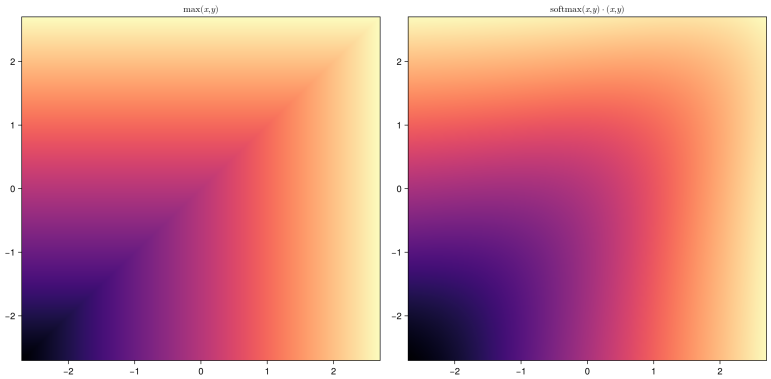
Differentiable Programming

Softmax as a differentiable approximation of argmax

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

approximating the maximum is straightforward:

$$\text{maximum element} \approx \sum_i^n \text{softmax}(x_i) x_i$$



Differentiable Programming

Now can we come up with a soft approximation of sort?[3]

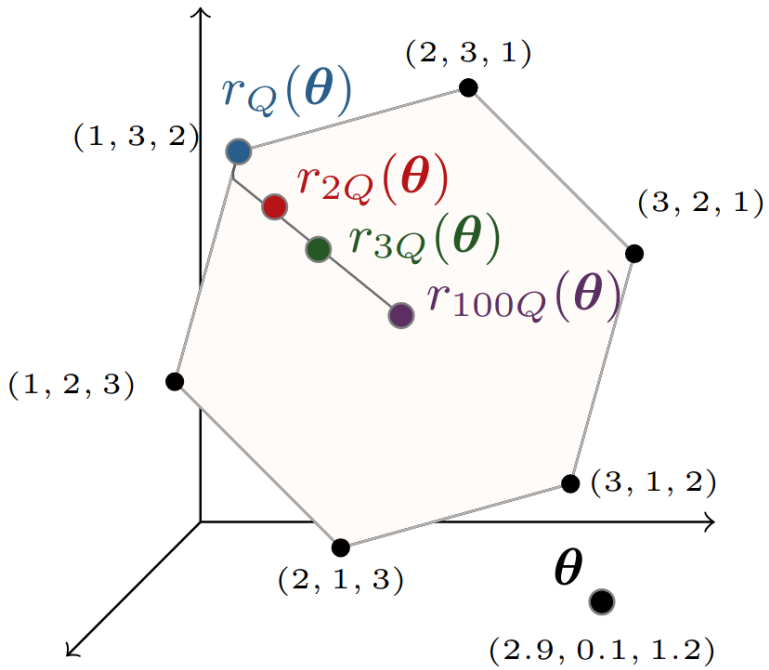
$$[10, 5, 20, 8, 40, 0]$$

[3] Image : Fast Differentiable Sorting and Ranking, arXiv:2002.08871

Differentiable Programming

Now can we come up with a soft approximation of sort?[3]

$$[10, 5, 20, 8, 40, 0]$$



[3] Image : Fast Differentiable Sorting and Ranking, arXiv:2002.08871

Differentiable Programming

How about other algorithms?

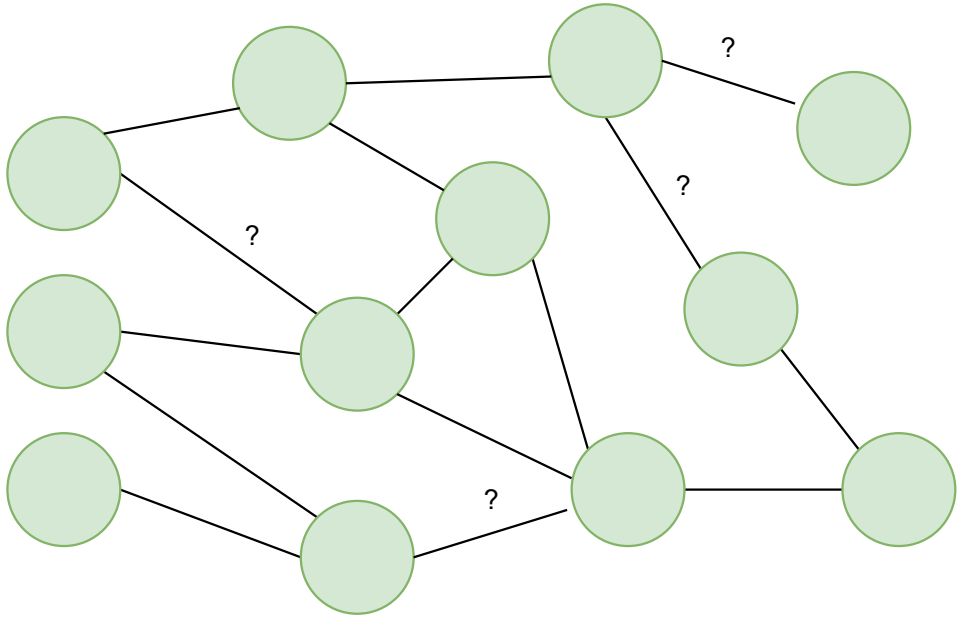
Differentiable Programming

How about other algorithms?

For example, a soft shortest path algorithm?

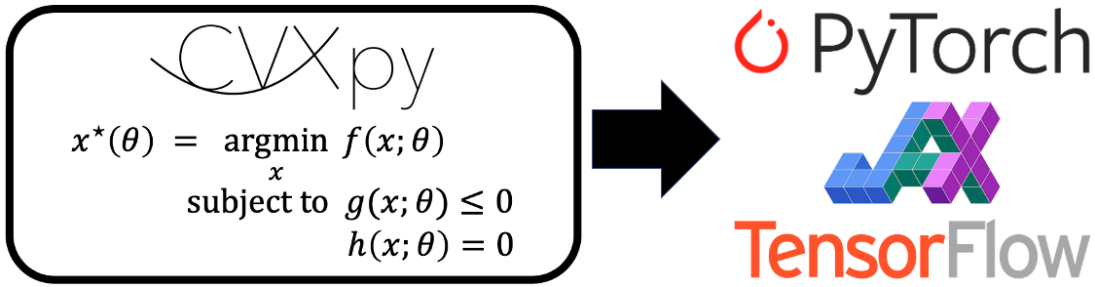
Example: Inverse Optimization

Suppose we have a graph with unknown weights



if we observe some agents traveling through this graph and assuming they tend to choose shortest paths, Can we infer the weights?

Useful Tools



CVXpyLayers : Differentiable Convex Optimization Layers[4]

[4] Differentiable Convex Optimization Layers, arXiv:1910.12430

Useful Tools

AlgoVision : Differentiable Algorithms and Algorithmic Supervision[5]

[5] Learning with Algorithmic Supervision via Continuous Relaxations, arXiv:2110.05651

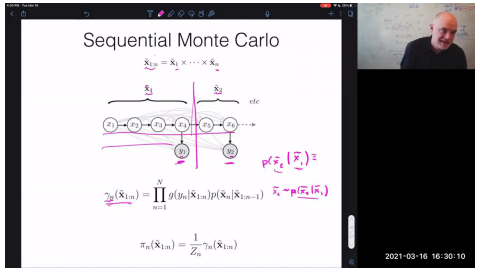
Summary

Probabilistic Programming + Differentiable Programming

Learning Resources

course A graduate course on Probabilistic Programming (25 lectures) by Dr. Frank Wood at UBC.

book An Introduction to Probabilistic Programming



An Introduction to Probabilistic Programming

Jan-Willem van de Meent
Institute of Informatics
University of Amsterdam
j.w.vandemeent@uva.nl

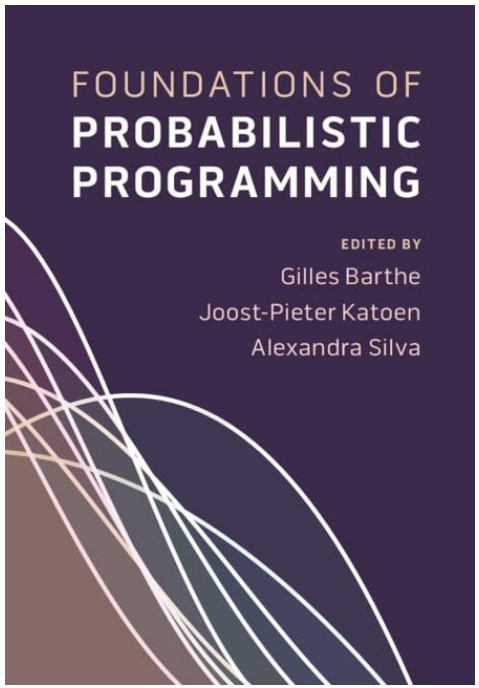
Brooks Paige
University College London
Alan Turing Institute
b.paige@ucl.ac.uk

Hongseok Yang
School of Computing
KAIST
hongseok.yang@kaist.ac.kr

Frank Wood
Department of Computer Science
University of British Columbia
fwood@cs.ubc.ca

Learning Resources

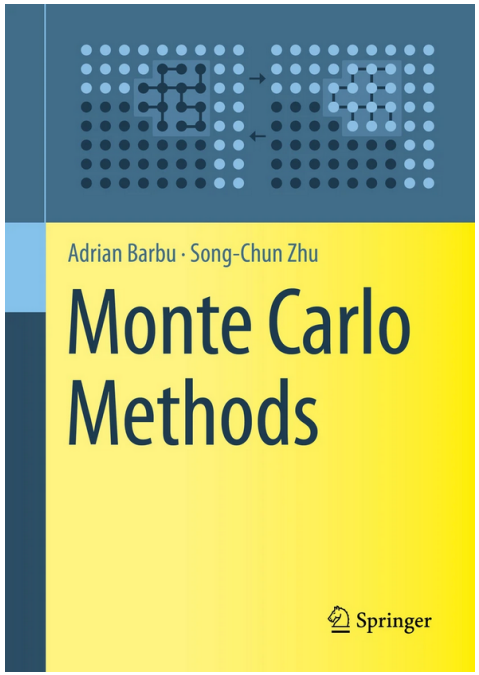
Foundations of Probabilistic Programming



Learning Resources

Monte Carlo Methods

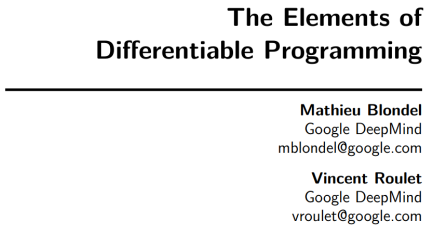
Adrian Barbu , Song-Chun Zhu



Learning Resources

The Elements of Differentiable Programming

Mathieu Blondel, Vincent Roulet



Learning Resources

Neural Algorithmic Reasoning

Homepage : `algo-reasoning.github.io`

Petar Velickovic, Andreea Deac, and Andrew Dudzik