Probabilistic Machine Learning with Julia

Lecture 6 | Approximate Bayesian Inference

Amirabbas Asadi

amir.asadi78@sharif.edu

2025

Bayesian Inference

$$p(z) \xrightarrow{\text{Bayesian Inference}} p(z|x)$$

Exact Inference

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

Exact Inference

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

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

Exact Inference

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

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

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

Exact Inference

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

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!

Exact Inference

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

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!

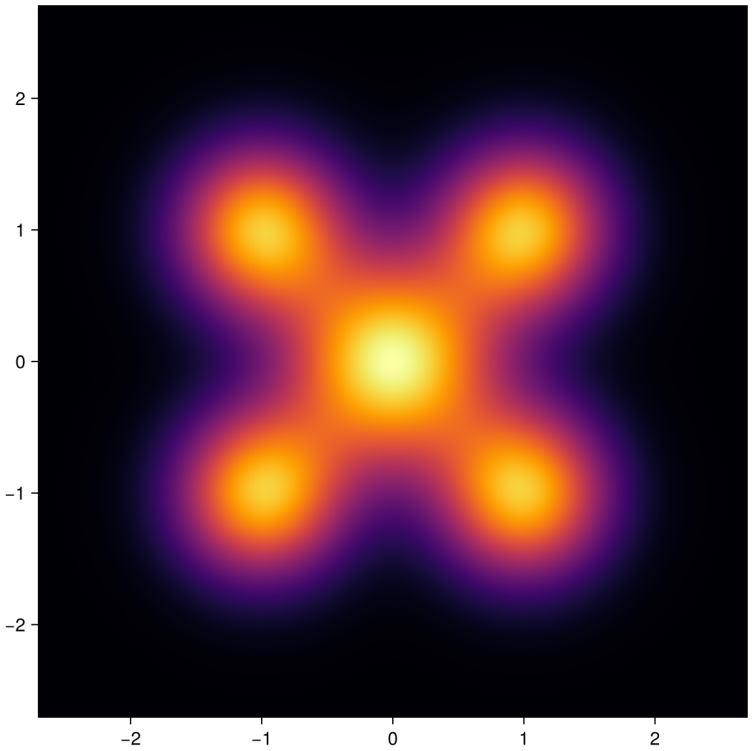
So the posterior turns out to be intractable

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

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

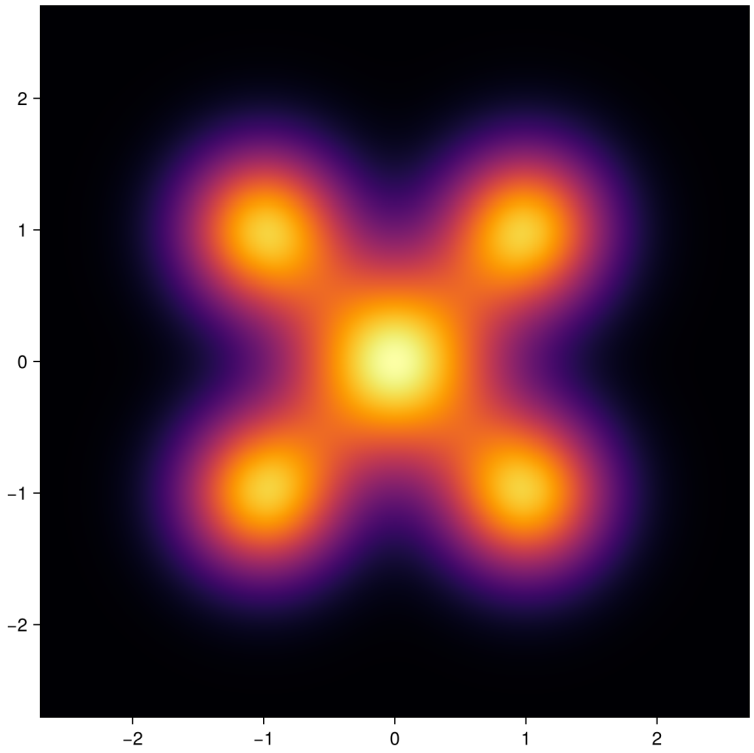
Approximate Inference Methods

Exact Inference is not possible for most of the models

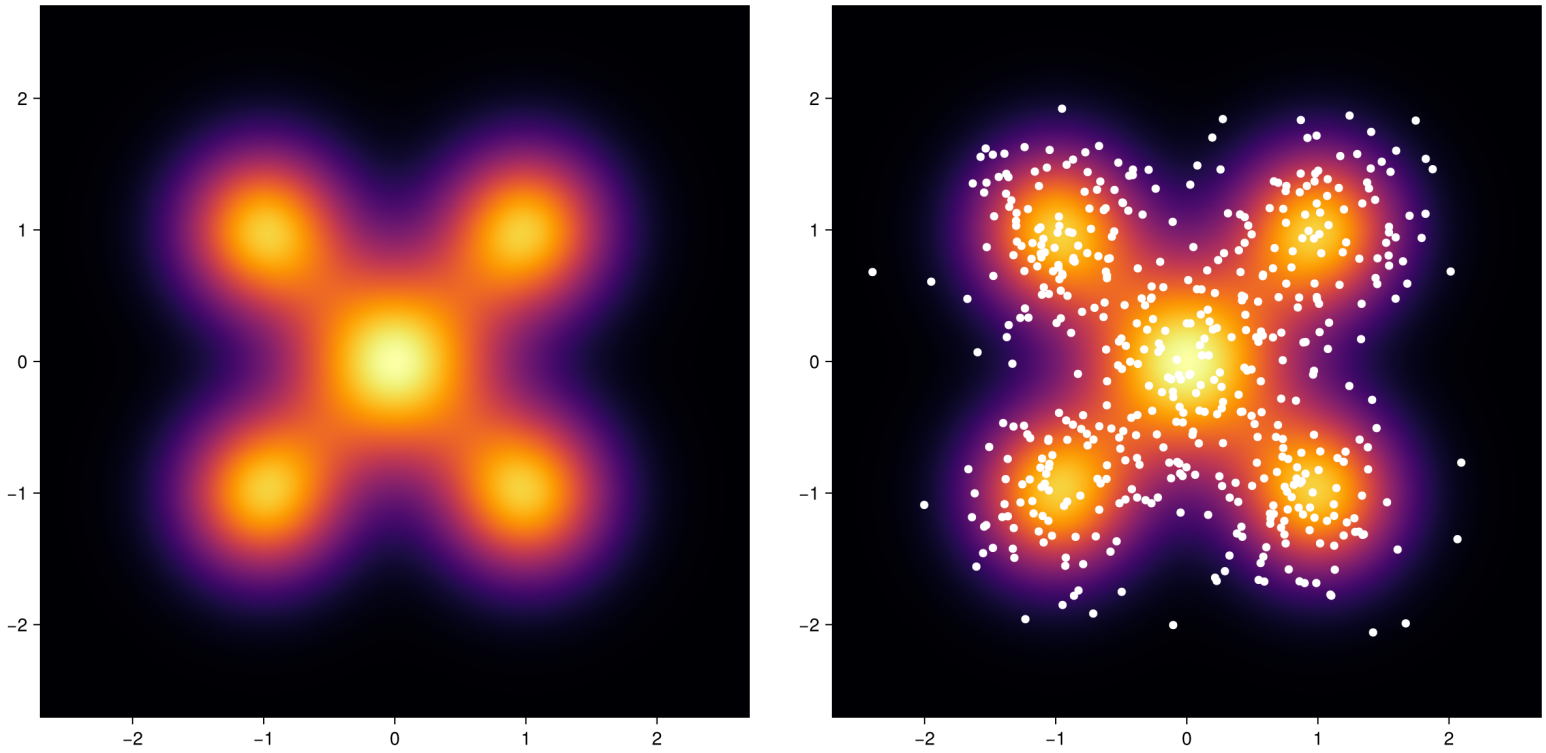


Fortunately there are some methods for approximate inference

Markov Chain Monte Carlo



Markov Chain Monte Carlo



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

$$X_0, X_1, X_2, \ldots$$

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

$$X_0, X_1, X_2, \ldots$$

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

$$X_0, X_1, X_2, \ldots$$

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

$$X_0, X_1, X_2, \dots$$

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

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

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 to satisfy this condition

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

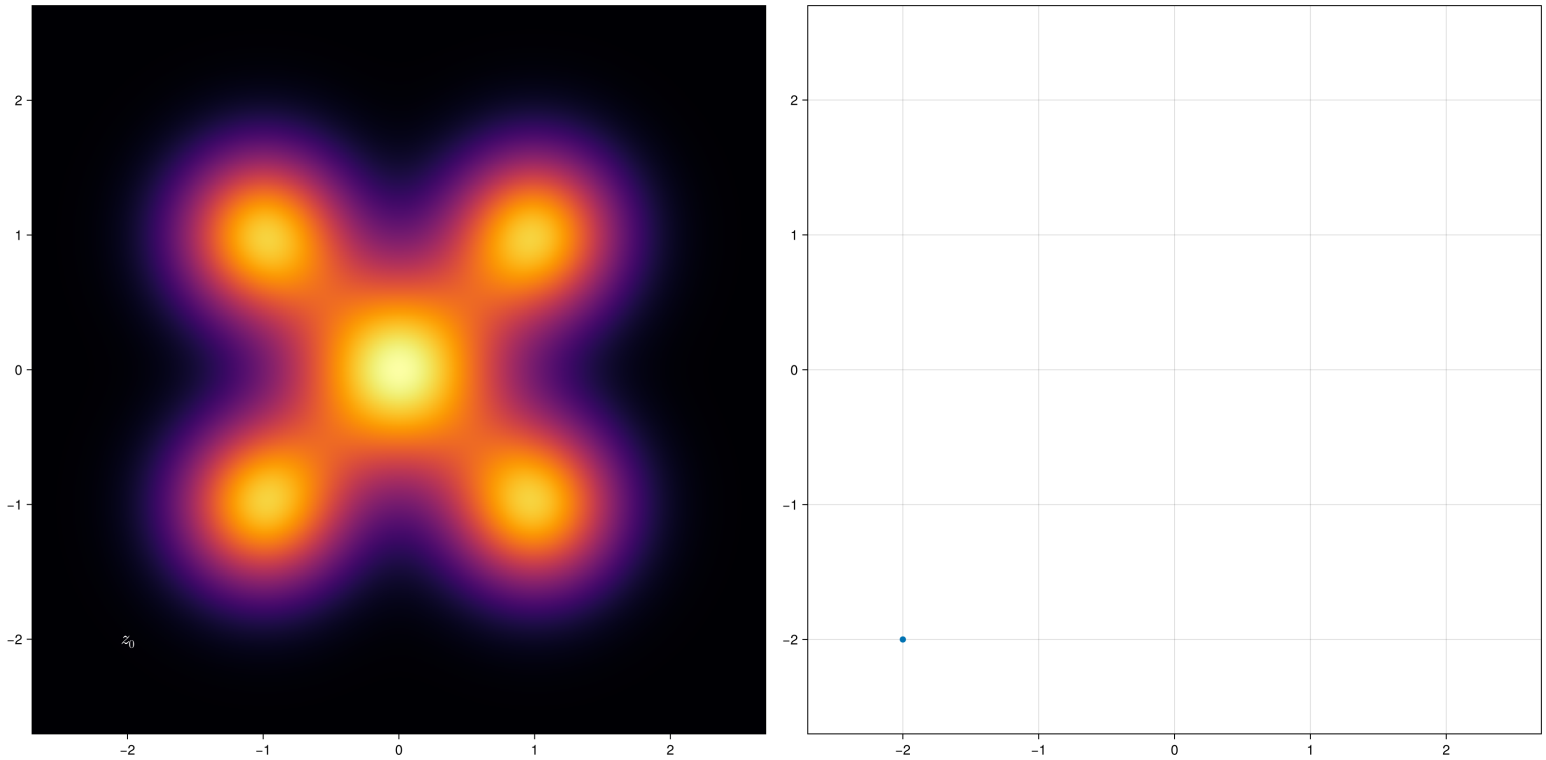
Random Walk Metropolis-Hastings

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 $\pi(x)$!

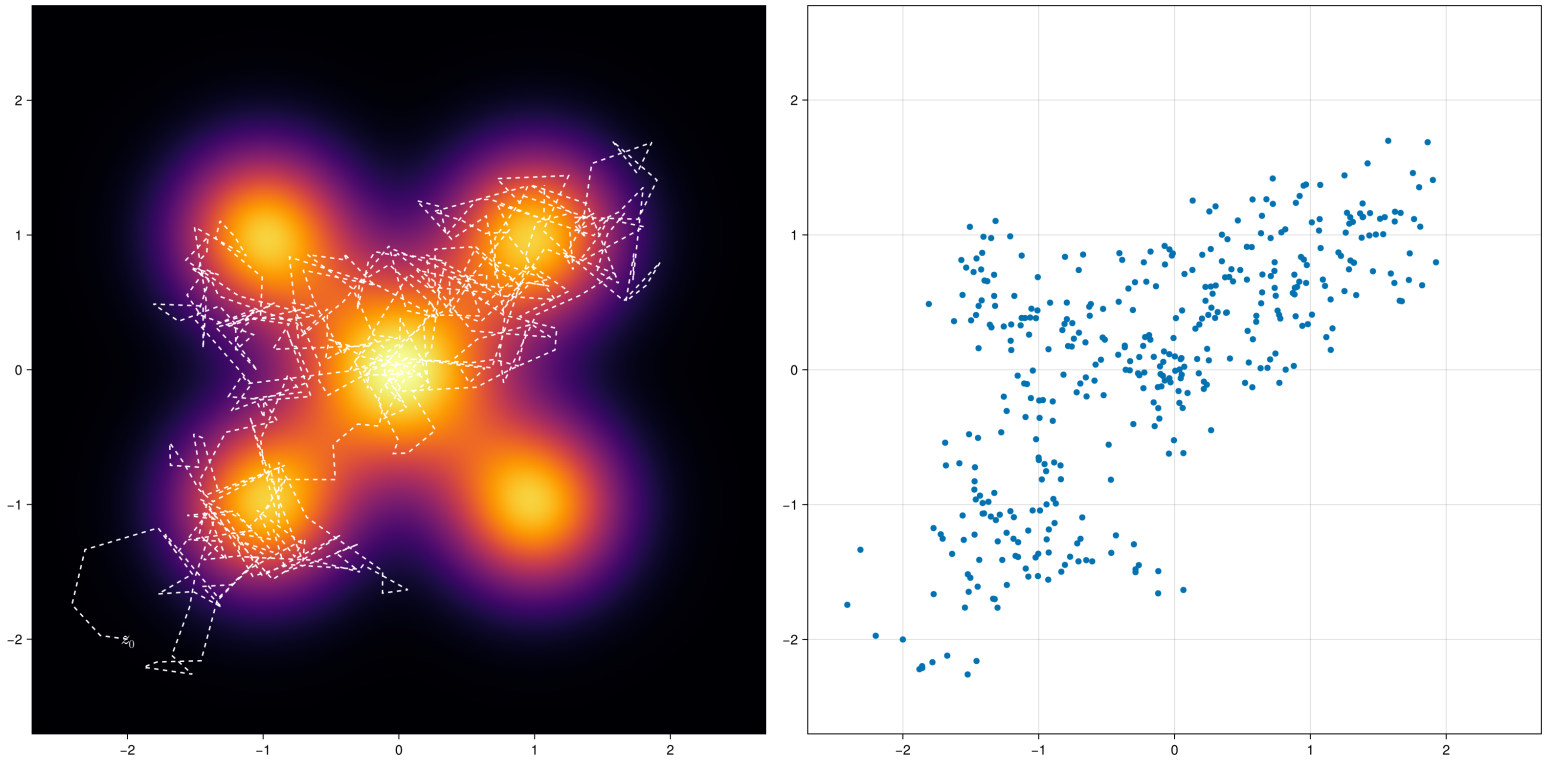
Random Walk Metropolis-Hastings

```
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
```

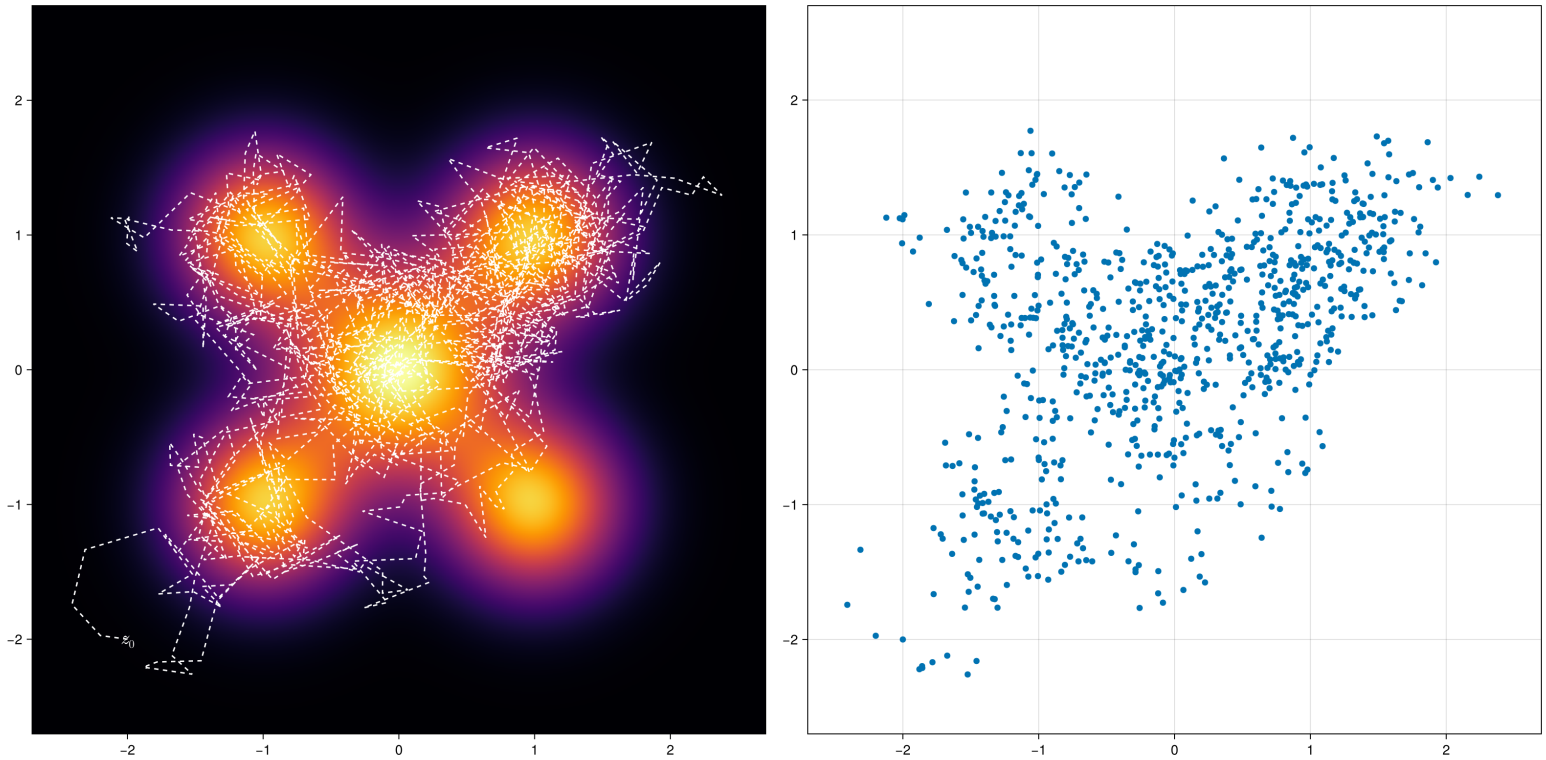
Random Walk Metropolis-Hastings



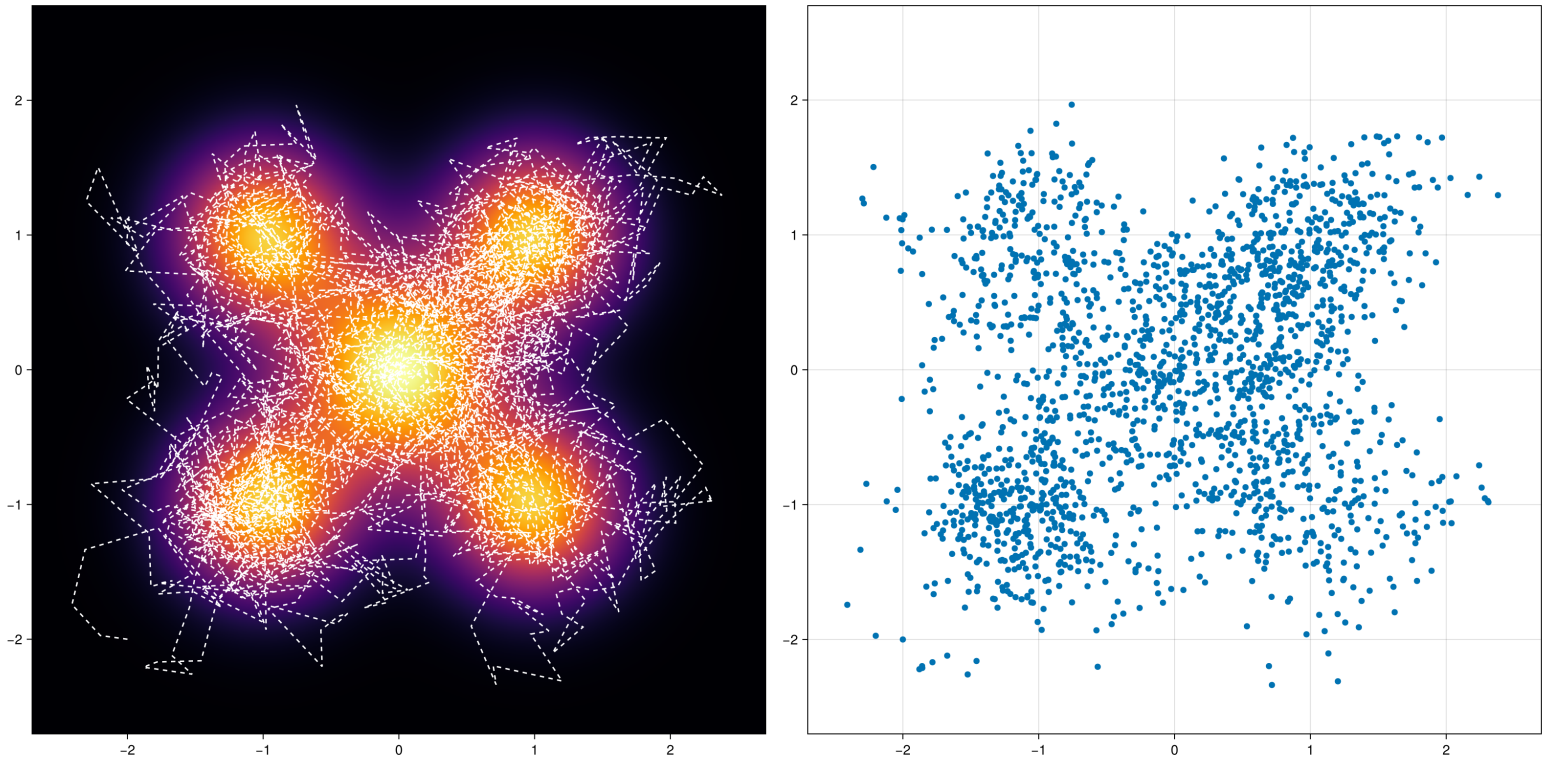
Random Walk Metropolis-Hastings



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!

Accelerating MCMC

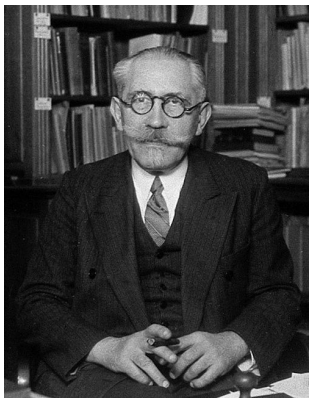
Instead of a random walk we can use more suitable dynamics designed to explore the target more efficiently and use Metropolis-Hastings rejection to achieve the desired stationary distribution. we will consider two examples:

- Langevin Monte Carlo
- Hamiltonian Monte Carlo (NUTS)

Langevin Dynamics

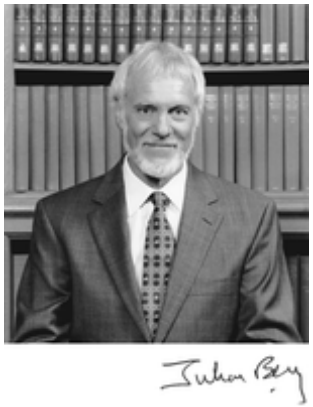
$$d\theta_t = \nabla \log \pi(\theta_t) dt + \sqrt{2} dW_t$$

```
function langevin_dynamics(logπ, θ, ϵ)
  ∇logπ = gradient(logπ, θ)
  ζ = sqrt(2ϵ) * randn(size(θ))
  ϵ*∇logπ .+ ζ
end
```



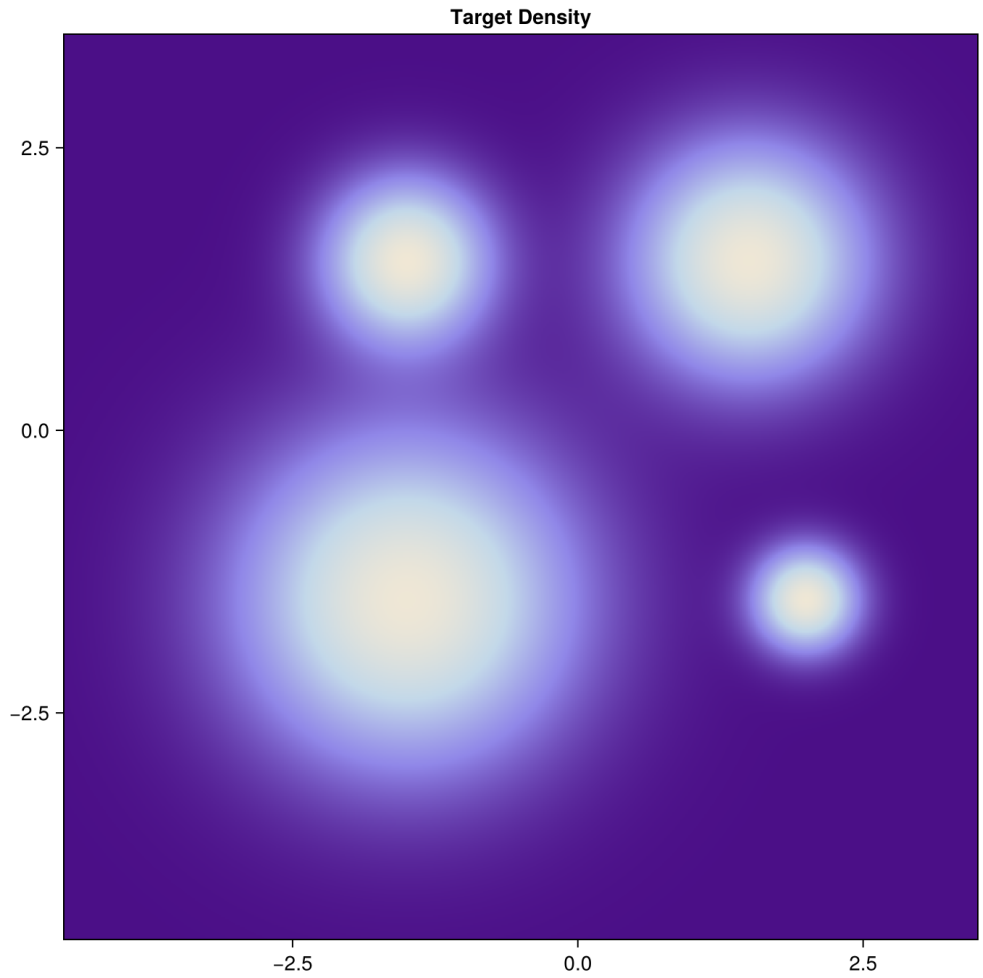
Paul Langevin

(1872-1946)



Julian Besag

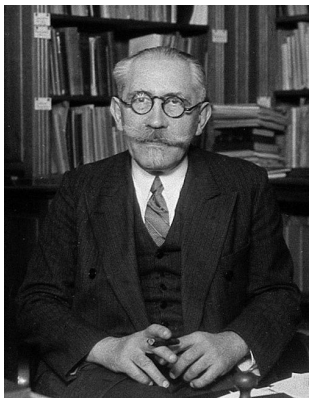
(1945-2010)



Langevin Dynamics

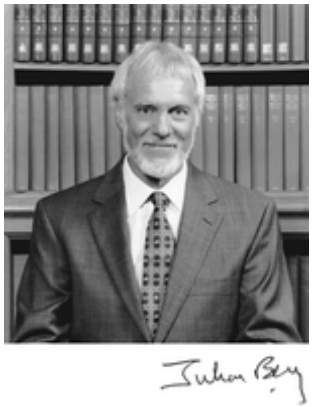
$$d\theta_t = \nabla \log \pi(\theta_t) dt + \sqrt{2} dW_t$$

```
function langevin_dynamics(logπ, θ, ϵ)
  ∇logπ = gradient(logπ, θ)
  ζ = sqrt(2ϵ) * randn(size(θ))
  ϵ*∇logπ .+ ζ
end
```



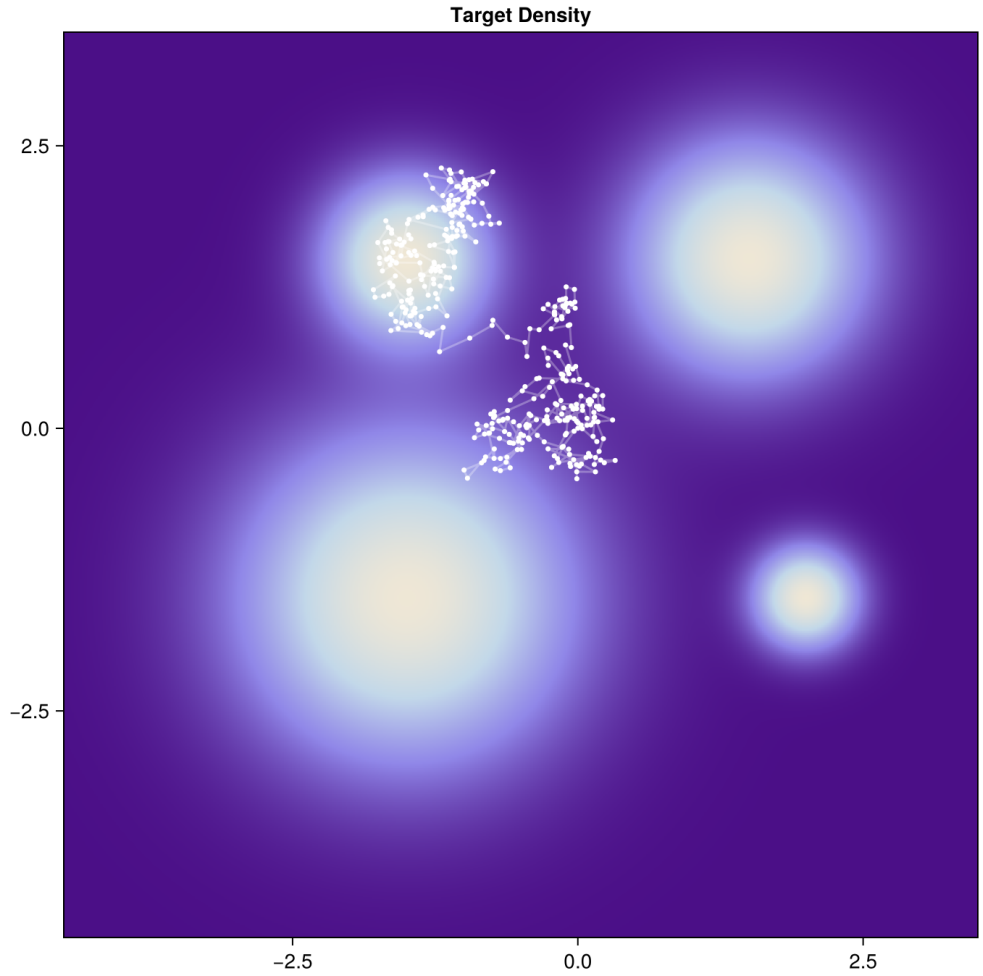
Paul Langevin

(1872-1946)



Julian Besag

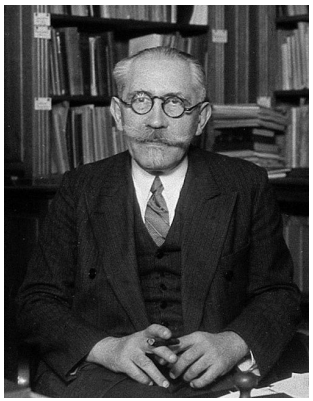
(1945-2010)



Langevin Dynamics

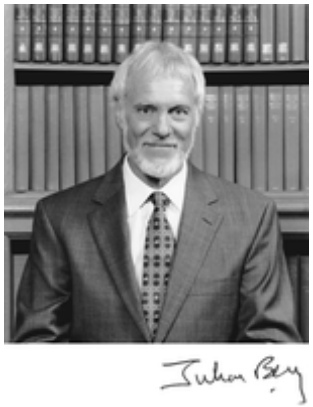
$$d\theta_t = \nabla \log \pi(\theta_t) dt + \sqrt{2} dW_t$$

```
function langevin_dynamics(logπ, θ, ϵ)
  ∇logπ = gradient(logπ, θ)
  ζ = sqrt(2ϵ) * randn(size(θ))
  ϵ*∇logπ .+ ζ
end
```



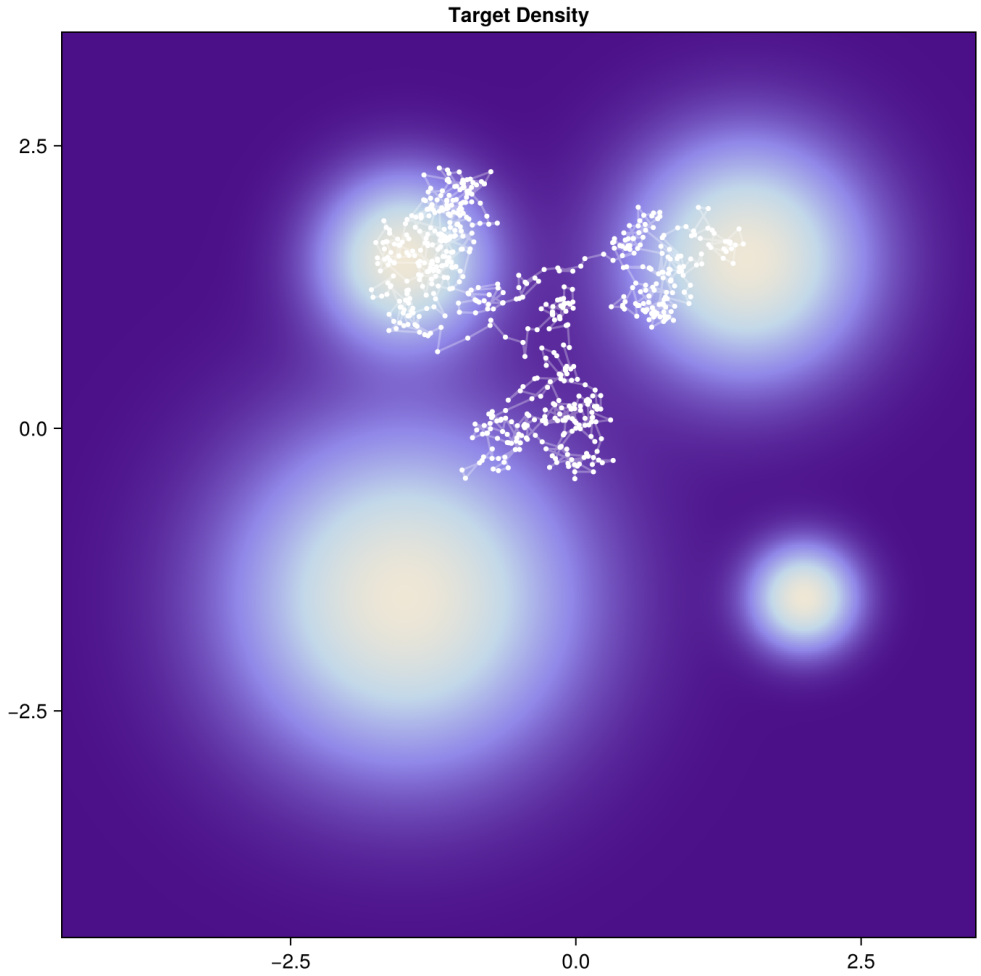
Paul Langevin

(1872-1946)



Julian Besag

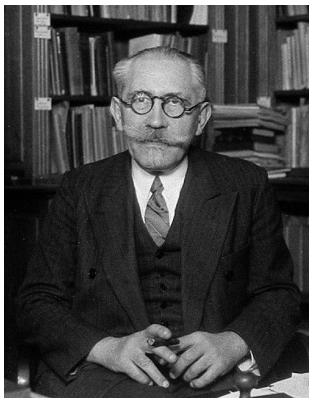
(1945-2010)



Langevin Dynamics

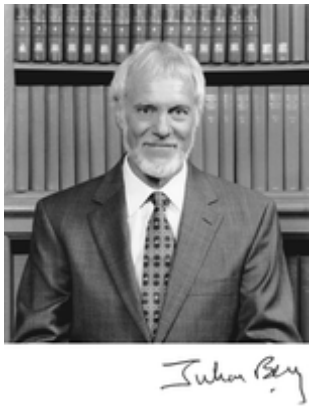
$$d\theta_t = \nabla \log \pi(\theta_t) dt + \sqrt{2} dW_t$$

```
function langevin_dynamics(logπ, θ, ϵ)
  ∇logπ = gradient(logπ, θ)
  ζ = sqrt(2ϵ) * randn(size(θ))
  ϵ*∇logπ .+ ζ
end
```



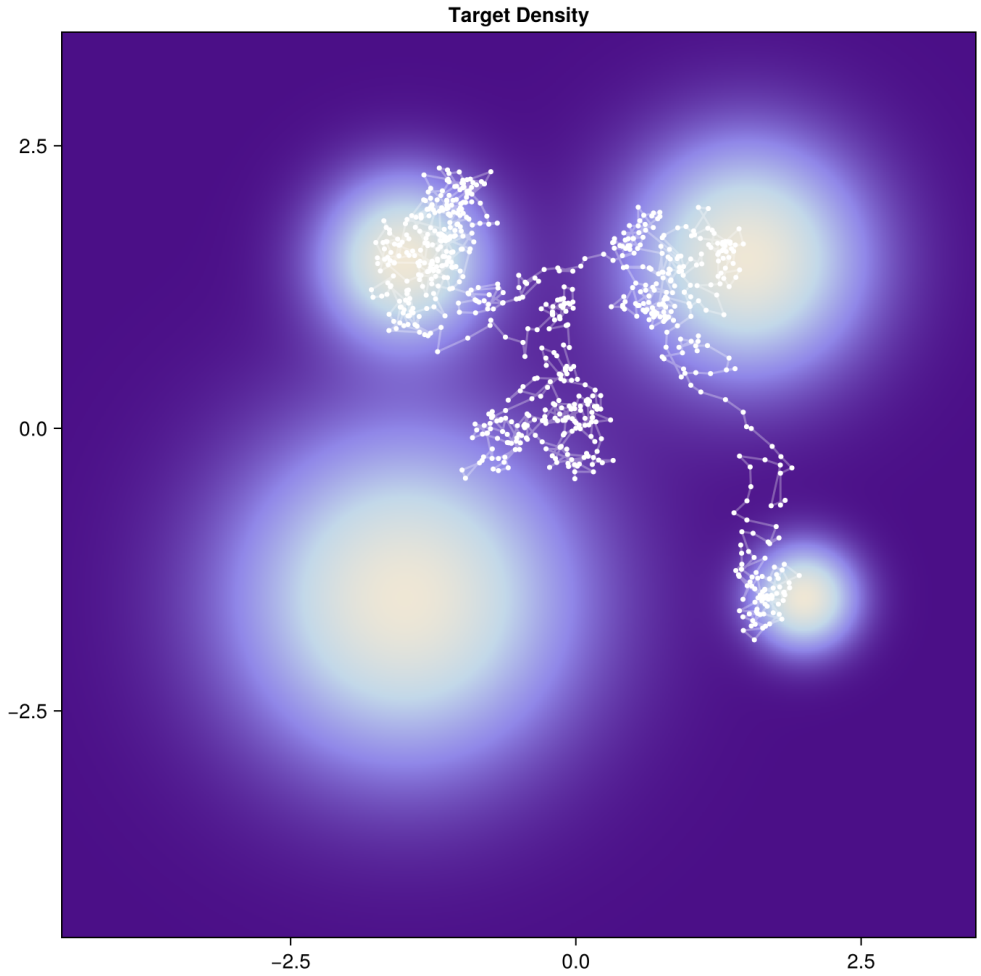
Paul Langevin

(1872-1946)



Julian Besag

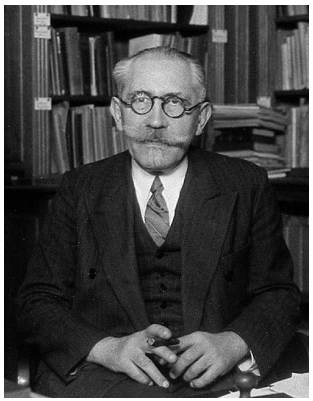
(1945-2010)



Langevin Dynamics

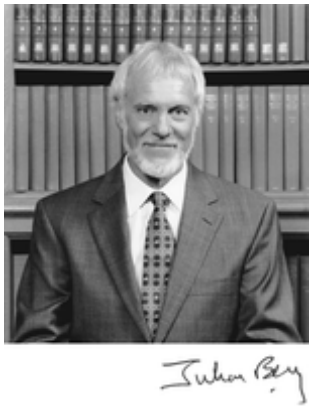
$$d\theta_t = \nabla \log \pi(\theta_t) dt + \sqrt{2} dW_t$$

```
function langevin_dynamics(logπ, θ, ϵ)
  ∇logπ = gradient(logπ, θ)
  ζ = sqrt(2ϵ) * randn(size(θ))
  ϵ*∇logπ .+ ζ
end
```



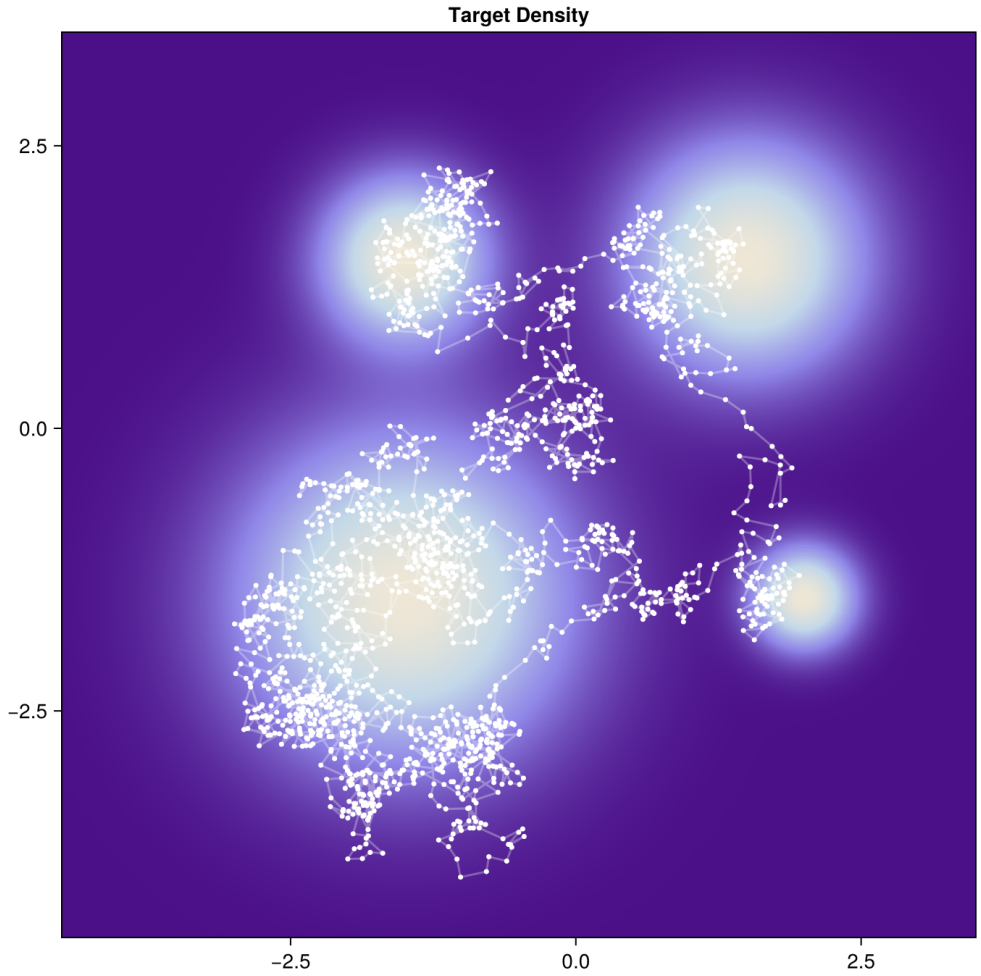
Paul Langevin

(1872-1946)



Julian Besag

(1945-2010)



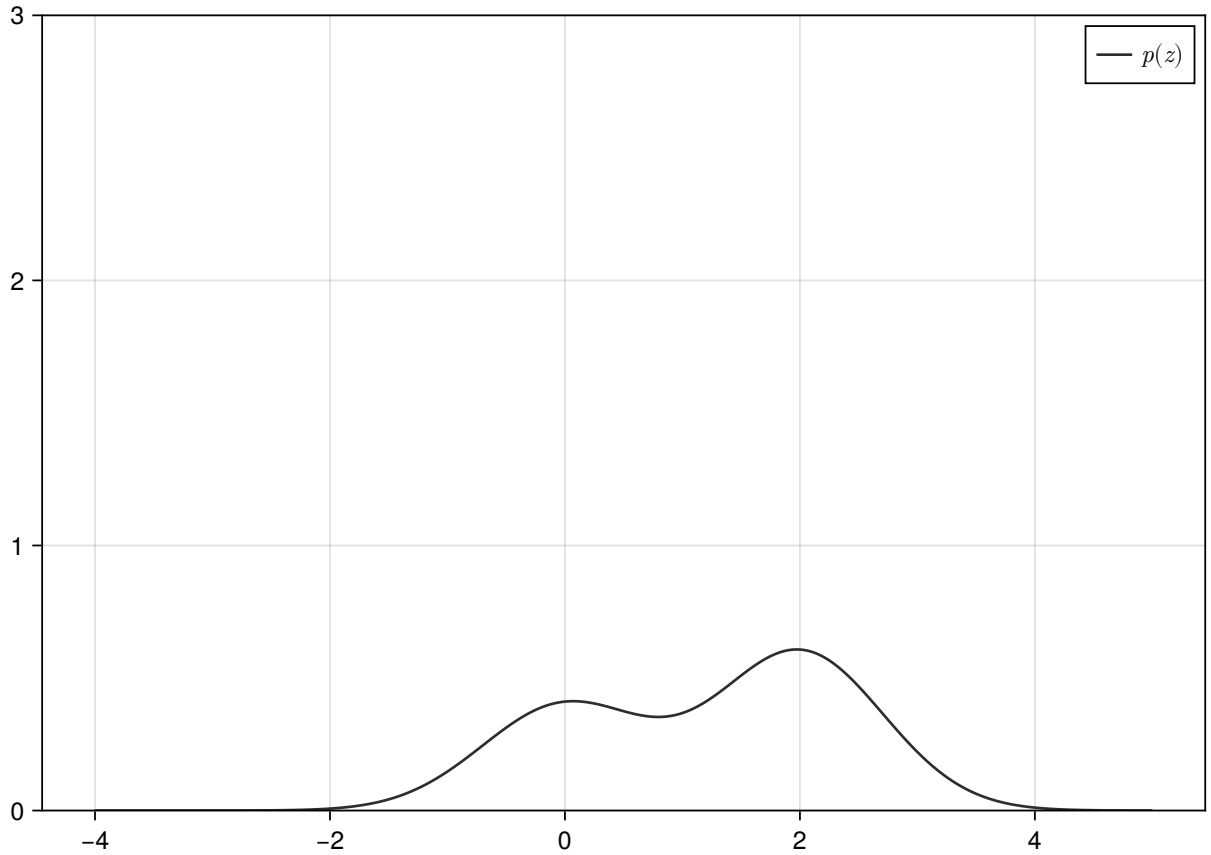
Langevin Monte Carlo

We adjust Langevin dynamics using Metropolis-Hastings rejection.

```
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
```

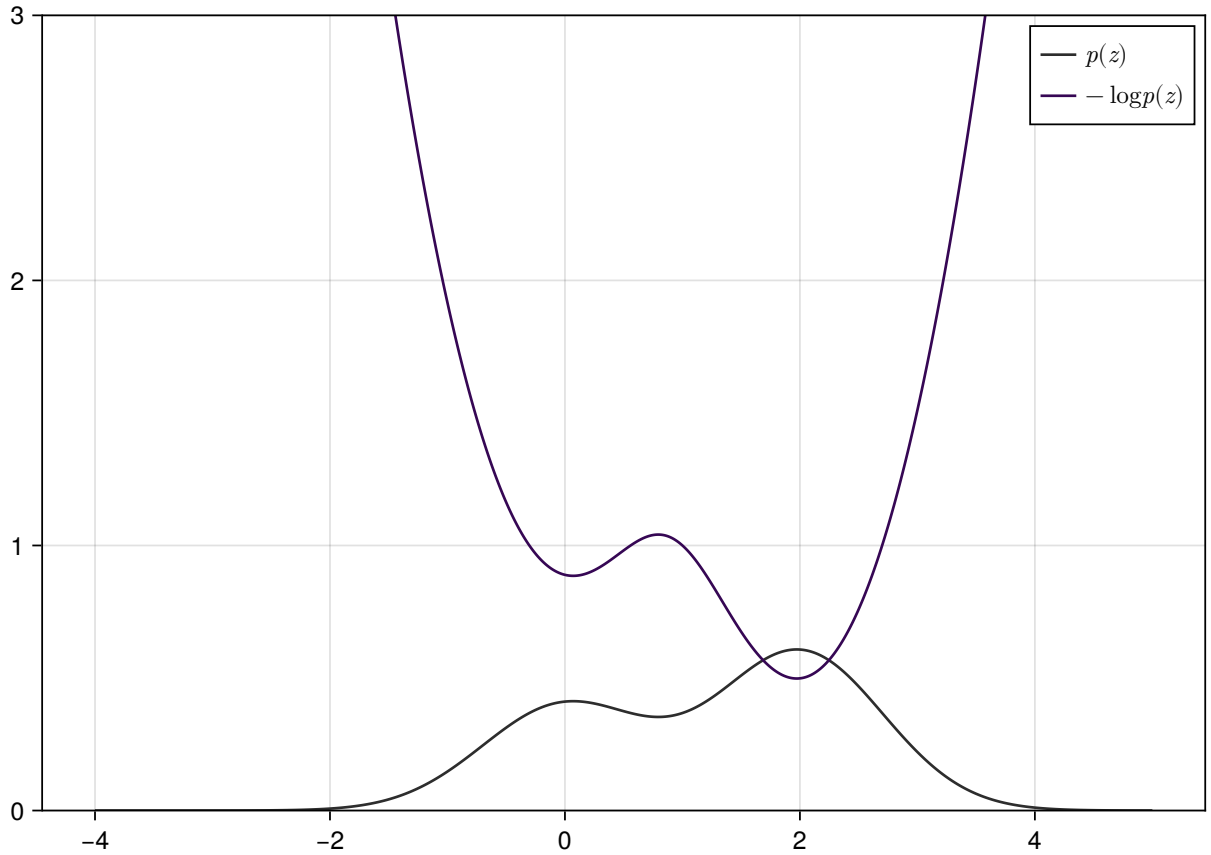
Hamiltonian Dynamics

Consider the dynamics of a ball fallen in the landspace formed by $-\log p$ moving with no friction.



Hamiltonian Dynamics

Consider the dynamics of a ball fallen in the landspace formed by $-\log p$ moving with no friction.



Hamiltonian Dynamics

$$\theta \in \mathbb{R}^d, r \in \mathbb{R}^d \quad p(\theta, r) \propto e^{-H(\theta, r)}$$

$$H(\theta, r) = U(\theta) + K(r)$$

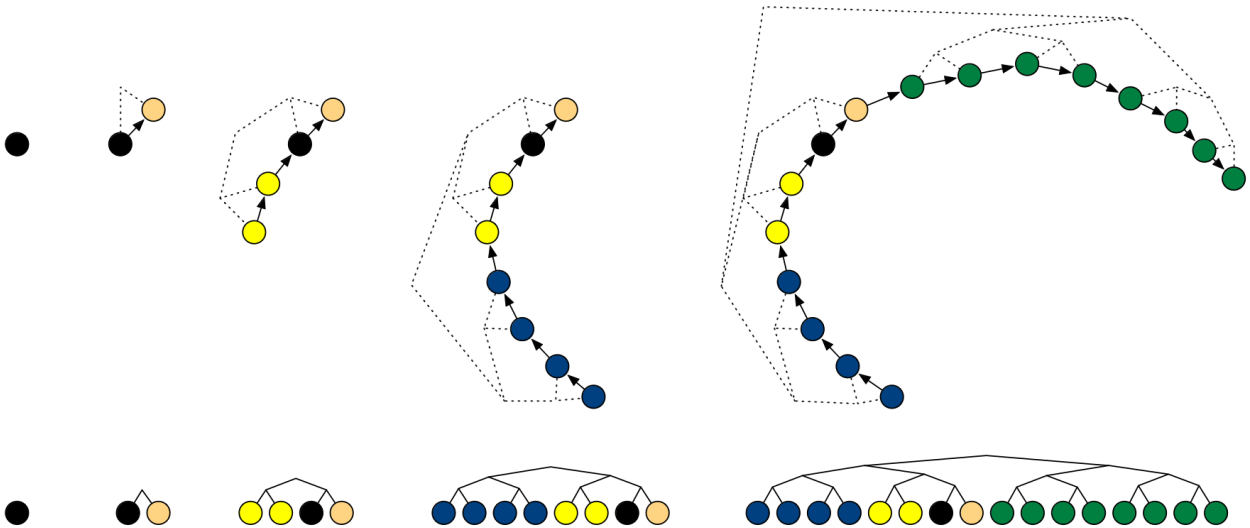
$$U(\theta) = -\log \pi(\theta) \quad K(r) = \frac{1}{2} r^\top r$$

$$\frac{d\theta_i}{dt} = \frac{\partial H}{\partial r_i}$$

$$\frac{dr_i}{dt} = -\frac{\partial H}{\partial \theta_i}$$

Hamiltonian Monte Carlo and NUTS

Hamiltonian Monte Carlo simulates the Hamiltonian dynamics with a numerical integrator[1] and applies Metropolis-Hastings. The performnace is sensitive to the choice of two parameters, the size and the number of steps of integrator. NUTS was proposed to tune these parameters. The number of steps is chosen as large as possible while preventing particle from getting back to the starting point (stopping U-turns).[2]



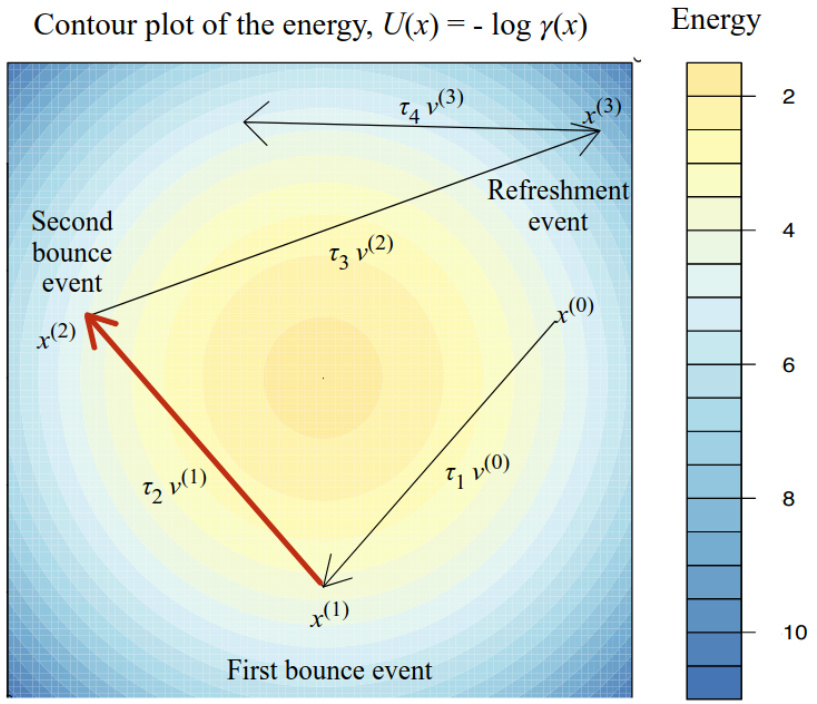
[1] A popular choice is leapfrog

[2] The No-U-Turn Sampler, Matthew D. Hoffman, Andrew Gelman, `https://arxiv.org/abs/1111.4246`

Rejection-free MCMC

In addition to Metropolis, there are other ideas for achieving the desired stationary distirbution. For instance, in *The Bouncy Particle Sampler*, particles can bounce to avoid entering areas with low probability. Unlike typical MCMC methods, we end up with a continuous trajectory![a]

[a] The Bouncy Particle Sampler, Alexandre Bouchard-Côté, Sebastian J. Vollmer, Arnaud Doucet, `https://arxiv.org/abs/1510.02451`

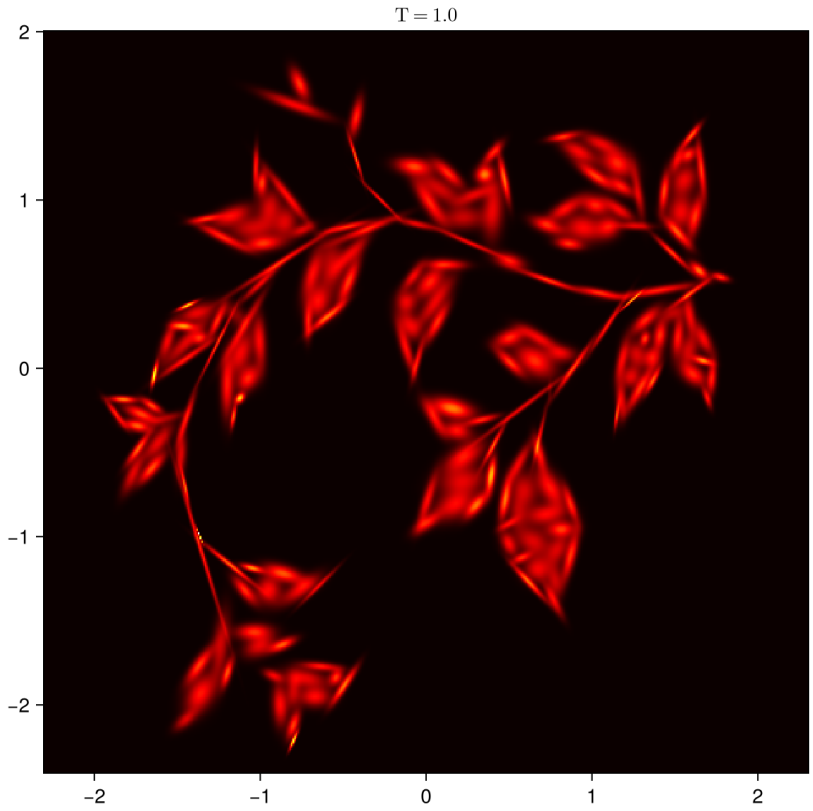


Tempering

A useful trick for sampling from densities with large number of modes is tempering.

$$p_T(z|x) \propto p(x|z)^T p(z)$$

Sampling is easier in low temperature. So we can sample from target density in low tempreature and gradually increase the temprature with a suitable scheduling. The idea can be used in combination with most of the approximate inference methods.

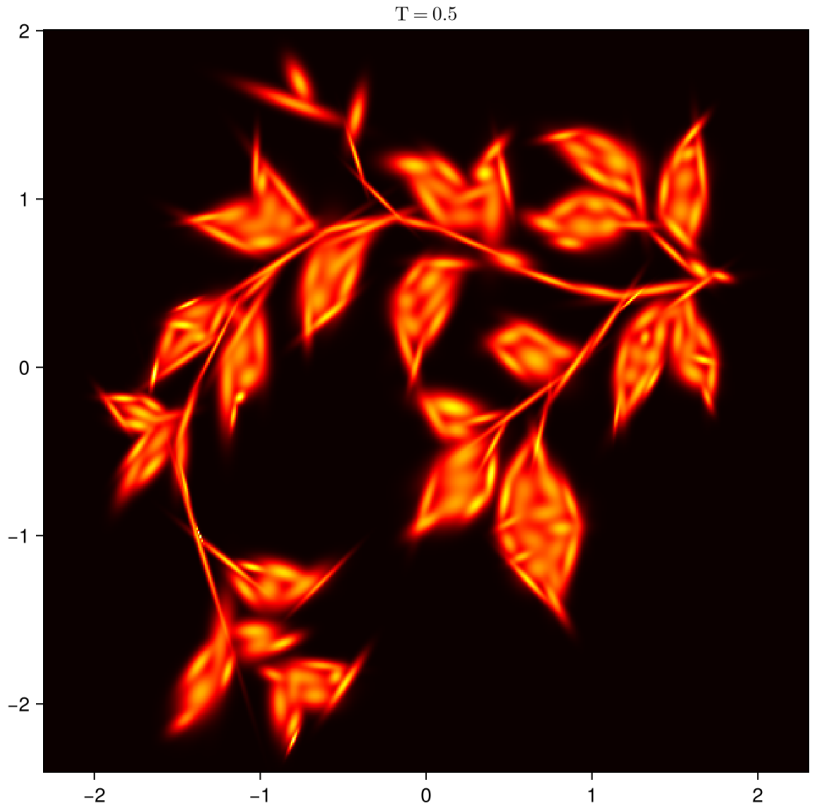


Tempering

A useful trick for sampling from densities with large number of modes is tempering.

$$p_T(z|x) \propto p(x|z)^T p(z)$$

Sampling is easier in low temperature. So we can sample from target density in low tempreature and gradually increase the temprature with a suitable scheduling. The idea can be used in combination with most of the approximate inference methods.

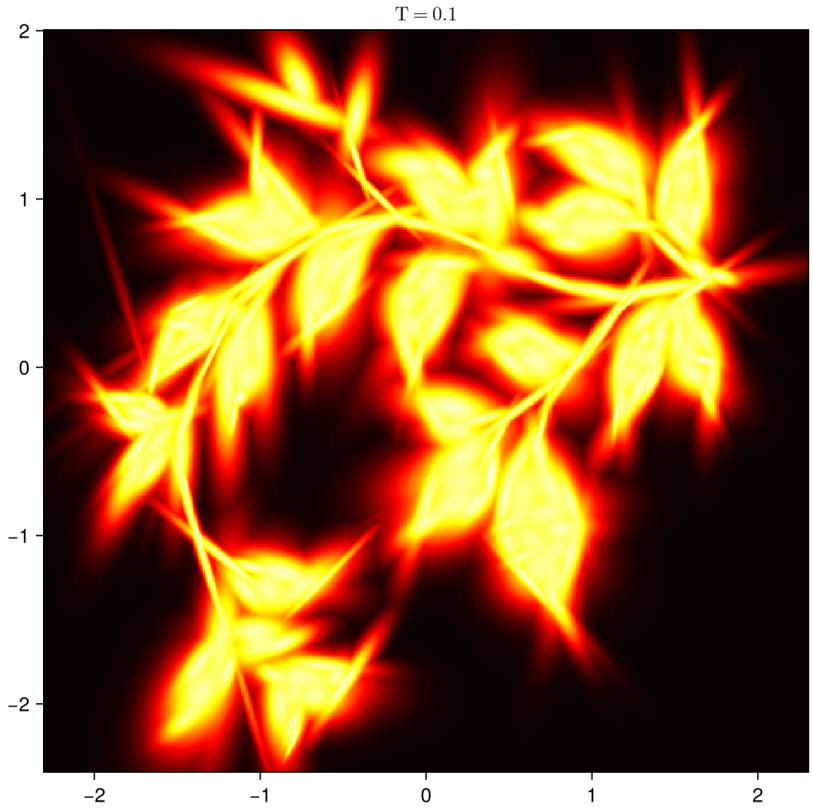


Tempering

A useful trick for sampling from densities with large number of modes is tempering.

$$p_T(z|x) \propto p(x|z)^T p(z)$$

Sampling is easier in low temperature. So we can sample from target density in low tempreature and gradually increase the temprature with a suitable scheduling. The idea can be used in combination with most of the approximate inference methods.

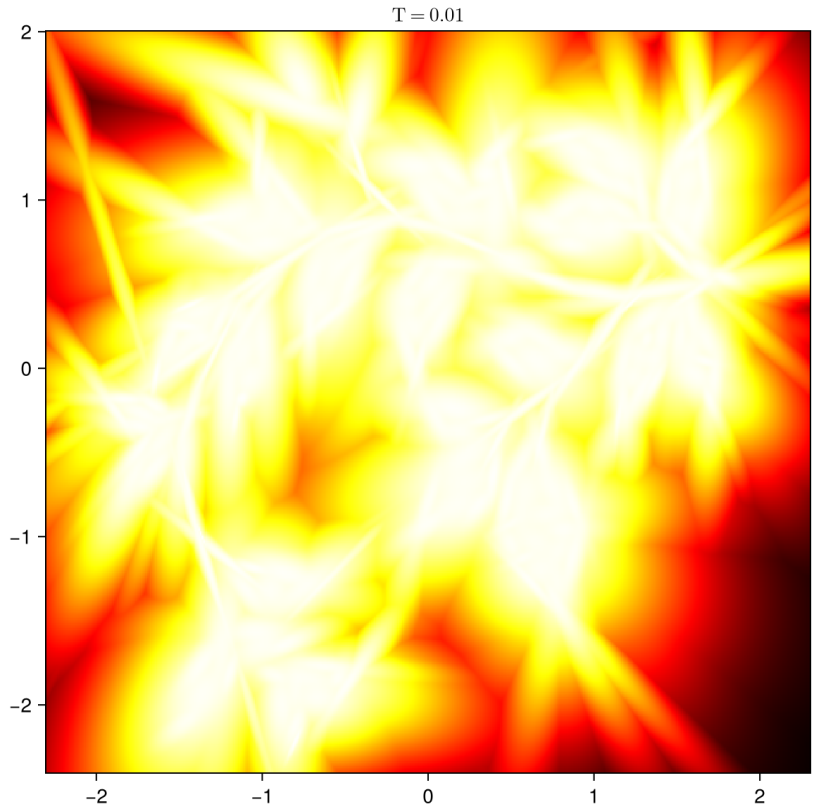


Tempering

A useful trick for sampling from densities with large number of modes is tempering.

$$p_T(z|x) \propto p(x|z)^T p(z)$$

Sampling is easier in low temperature. So we can sample from target density in low tempreature and gradually increase the temprature with a suitable scheduling. The idea can be used in combination with most of the approximate inference methods.

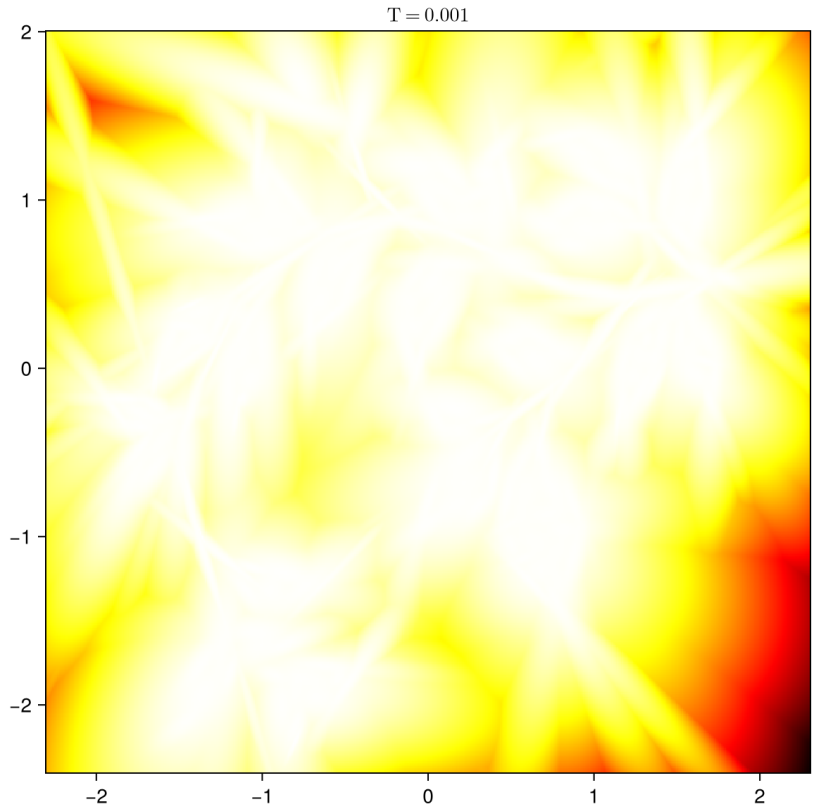


Tempering

A useful trick for sampling from densities with large number of modes is tempering.

$$p_T(z|x) \propto p(x|z)^T p(z)$$

Sampling is easier in low temperature. So we can sample from target density in low tempreature and gradually increase the temprature with a suitable scheduling. The idea can be used in combination with most of the approximate inference methods.

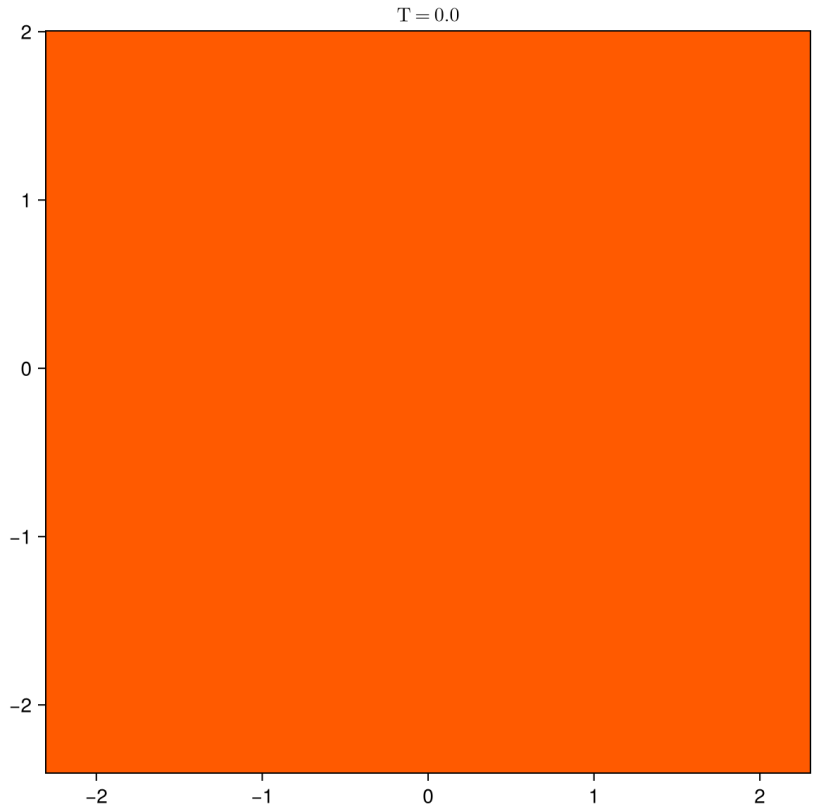


Tempering

A useful trick for sampling from densities with large number of modes is tempering.

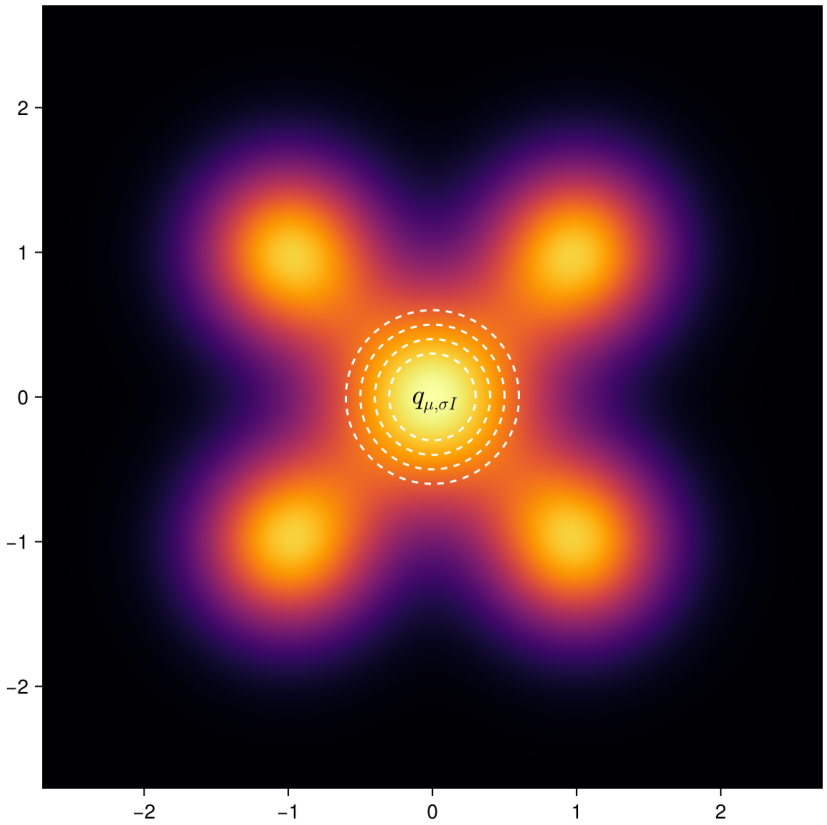
$$p_T(z|x) \propto p(x|z)^T p(z)$$

Sampling is easier in low temperature. So we can sample from target density in low tempreature and gradually increase the temprature with a suitable scheduling. The idea can be used in combination with most of the approximate inference methods.



Variational Inference

Another idea for approximate inference 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:

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

Variational Inference

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

Variational Inference

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

Where D_{KL} is the Kullback–Leibler divergence

Variational Inference

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

Where D_{KL} is the Kullback–Leibler divergence

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

Variational Inference

In parametric variational inference, The accuracy of inference is restricted by the representation power of $q(z; \lambda)$, the variational distribution.

How to choose the variational distribution?

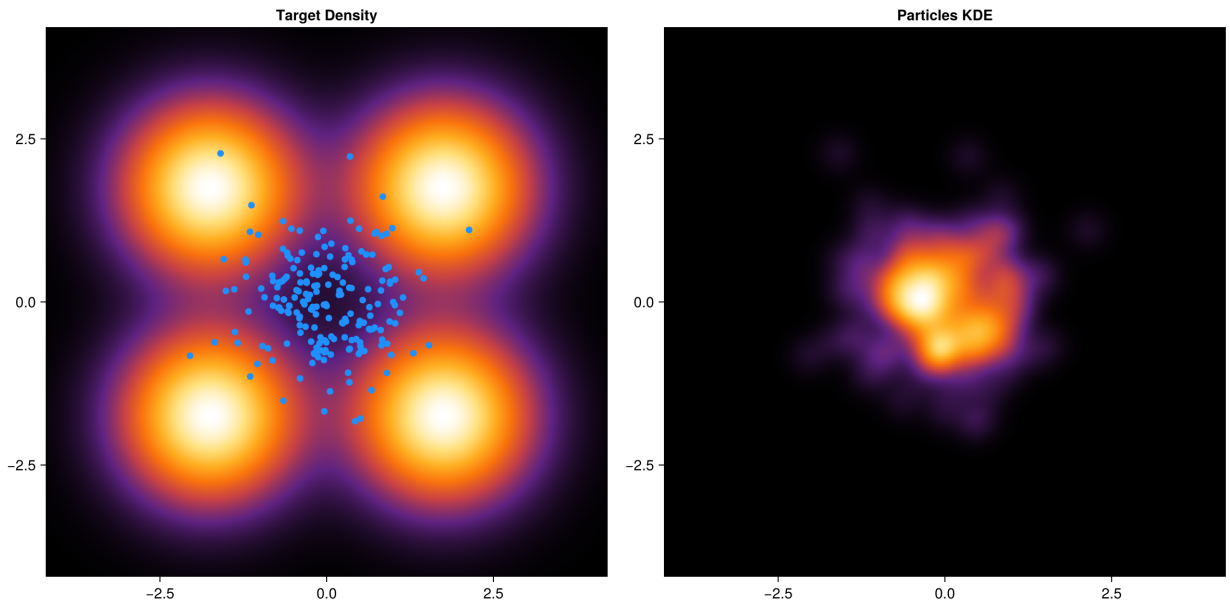
Variational Inference

One possible idea is to use tractable yet expressive enough distributions like a normalizing flow or a diffusion model as variational distribution to approximate posterior.

- Denoising Diffusion Variational Inference: Diffusion Models as Expressive Variational Posteriors, Wasu Top Piriyakulkij, Yingheng Wang, Volodymyr Kuleshov, `https://arxiv.org/abs/2401.02739`
- Variational Inference with Normalizing Flows, Danilo Jimenez Rezende, Shakir Mohamed, `https://arxiv.org/abs/1505.05770`

Nonparametric Variational Inference

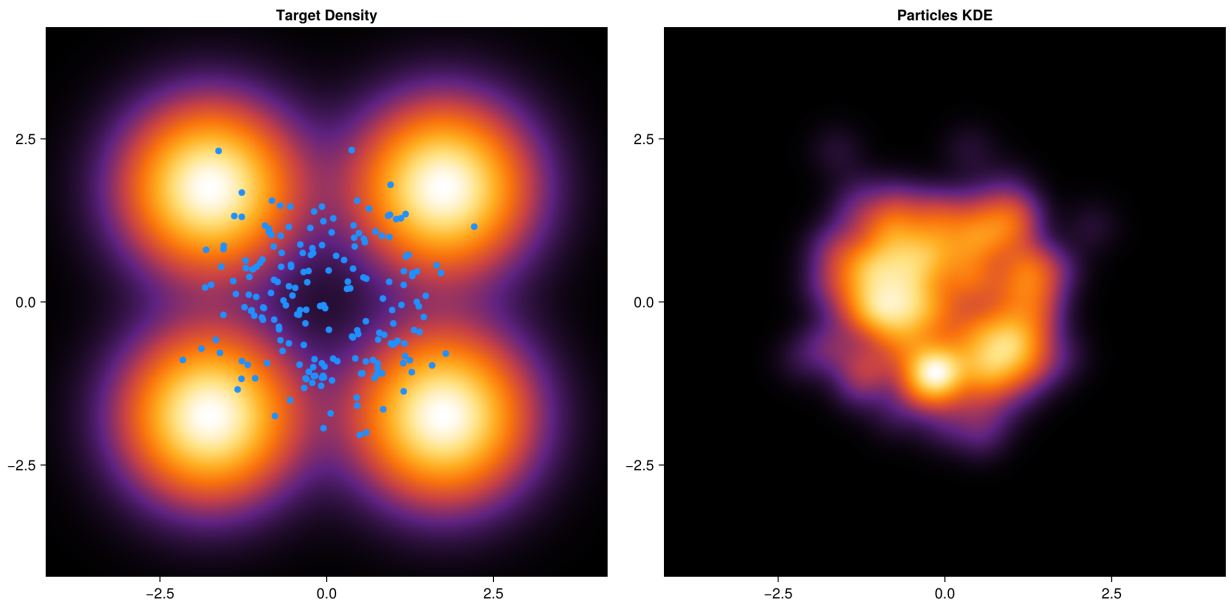
We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

Nonparametric Variational Inference

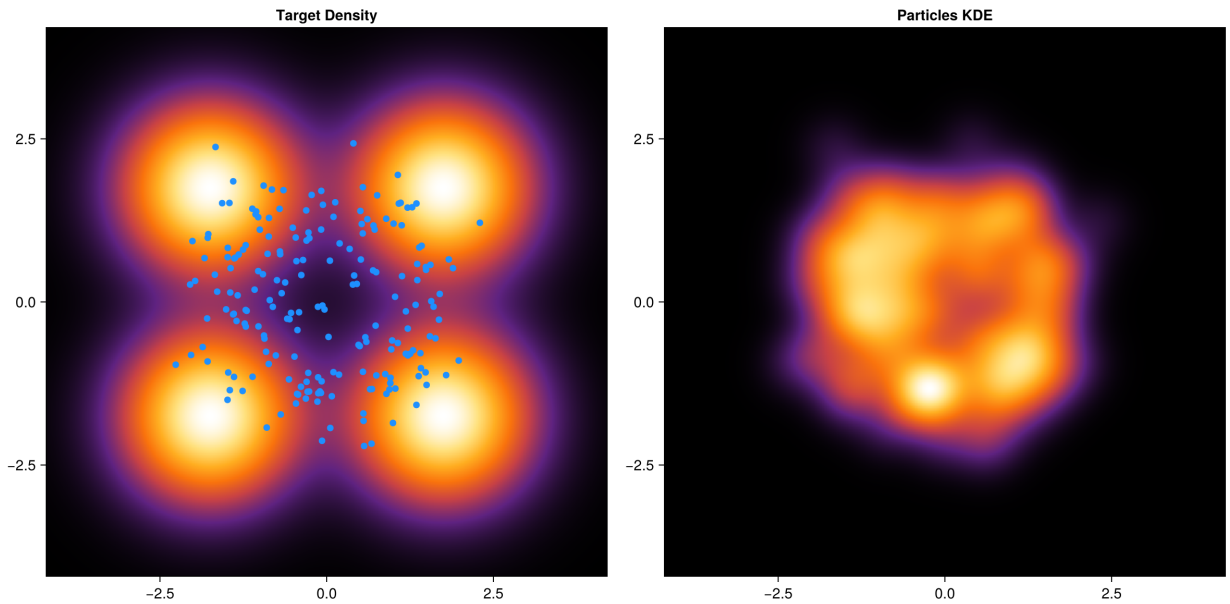
We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

Nonparametric Variational Inference

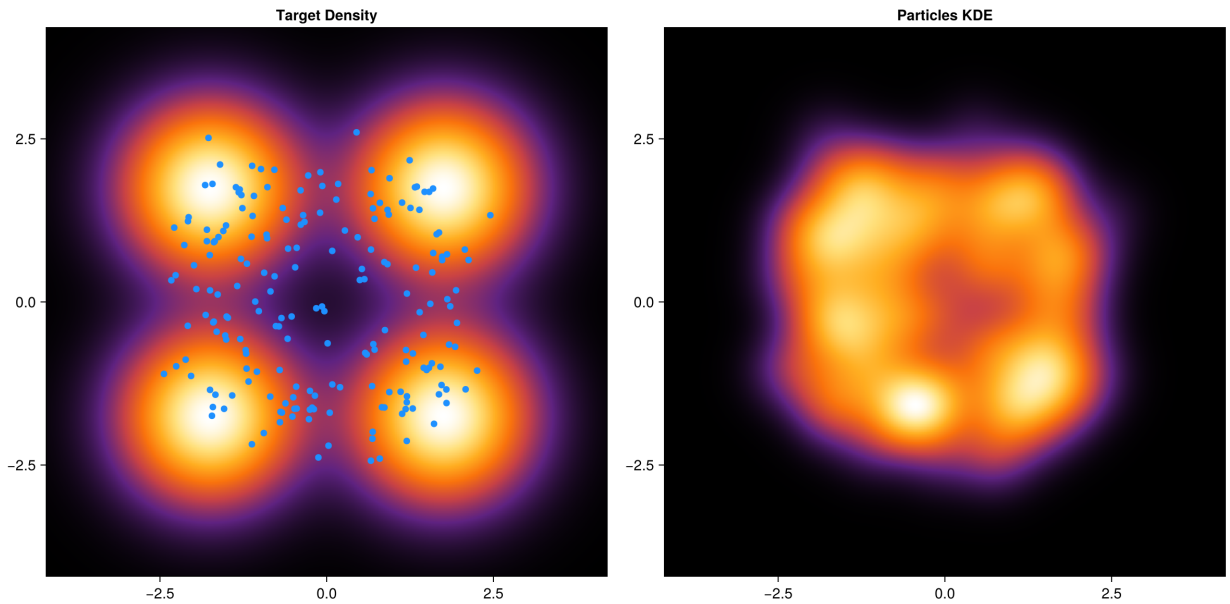
We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

Nonparametric Variational Inference

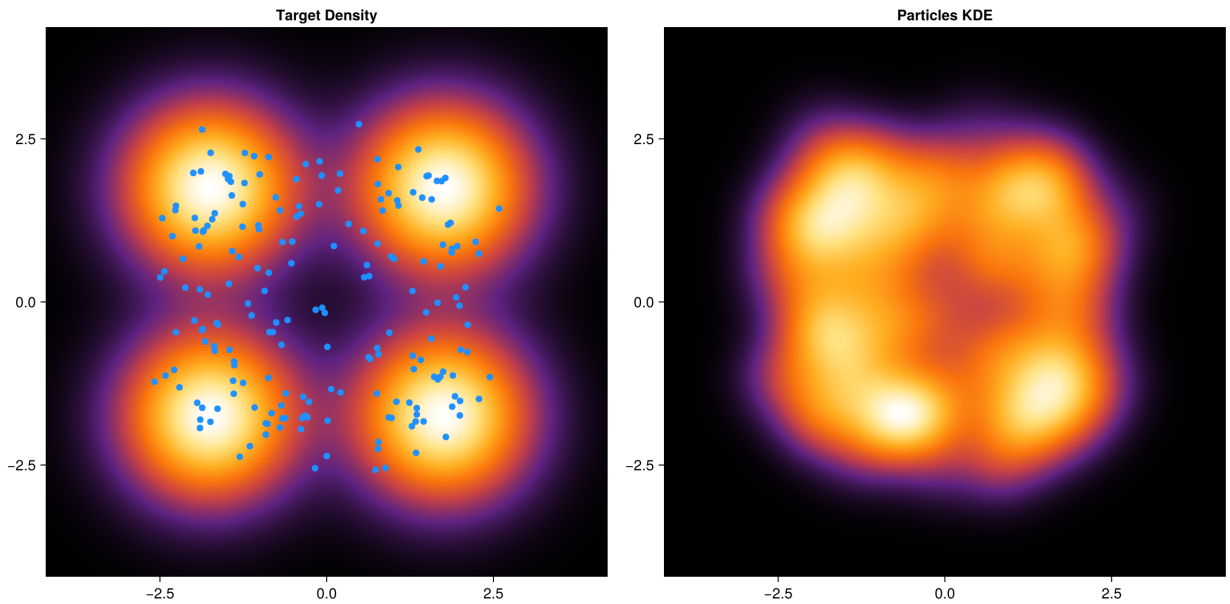
We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

Nonparametric Variational Inference

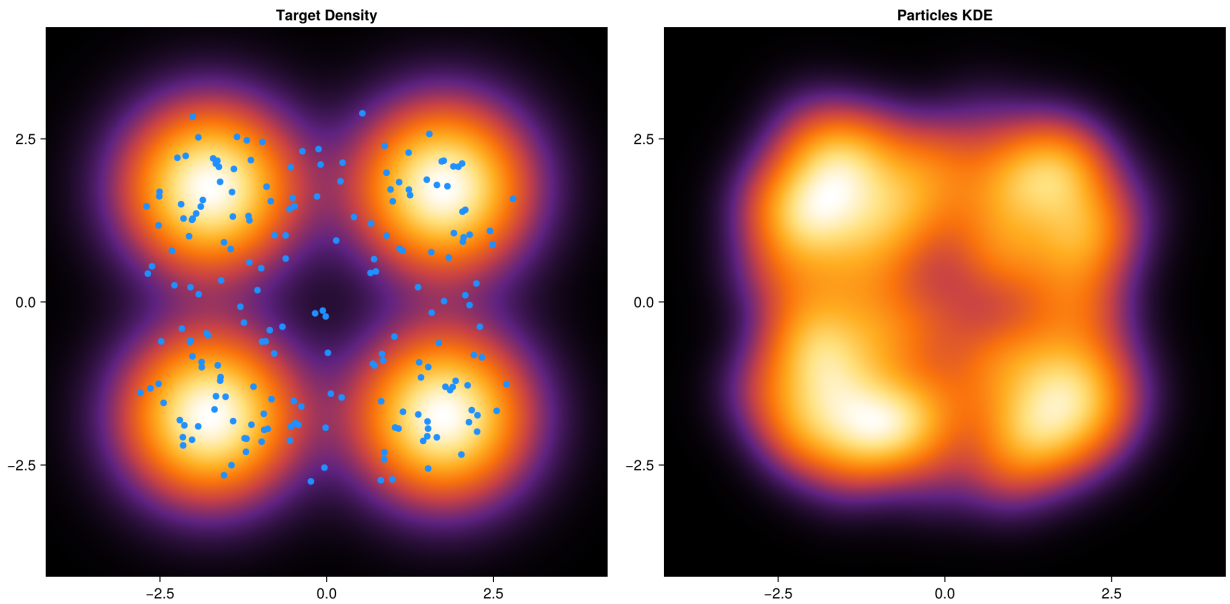
We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

Nonparametric Variational Inference

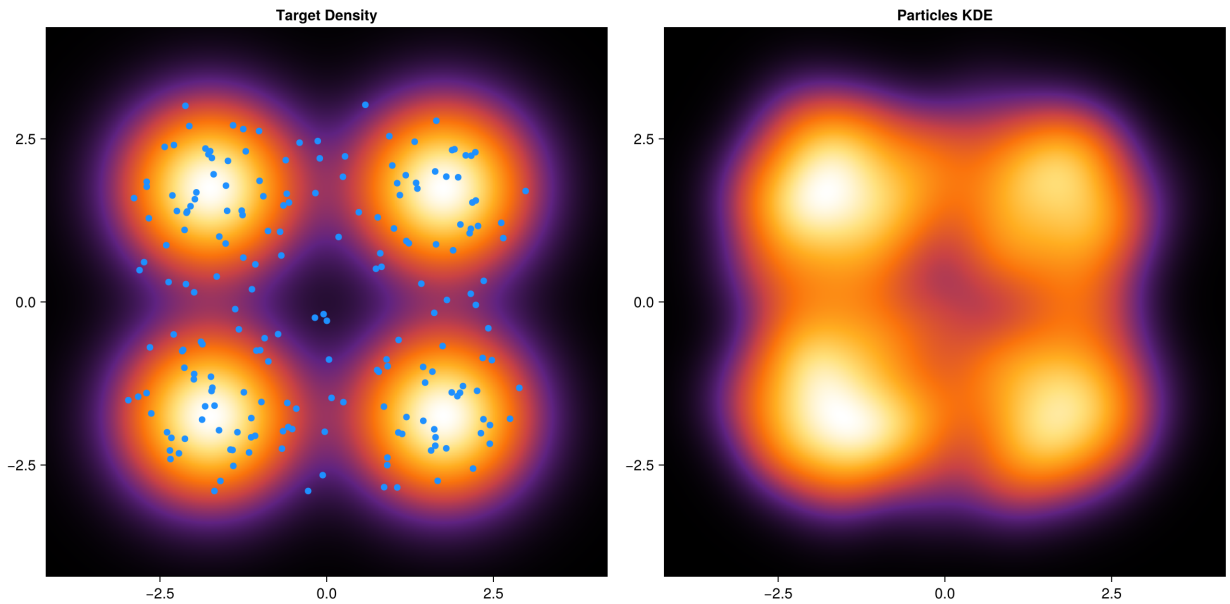
We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

Nonparametric Variational Inference

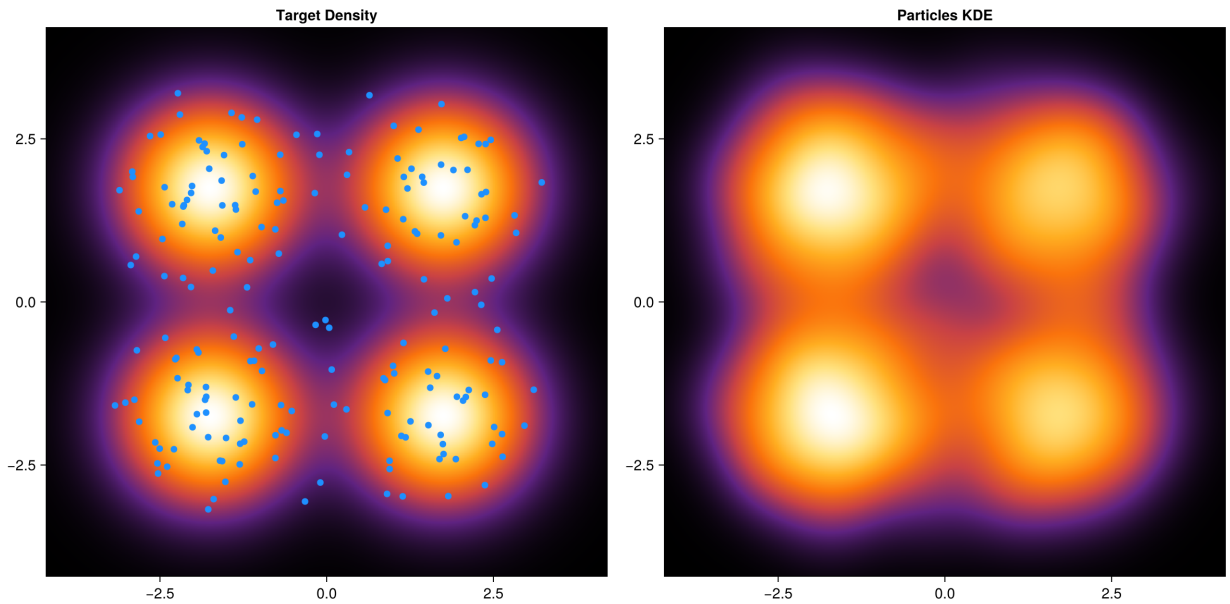
We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

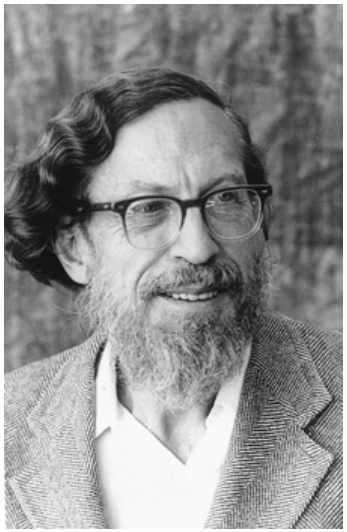
Nonparametric Variational Inference

We can instead use a nonparametric family of variational distributions to approximate the posterior. The idea is to represent the density with a set of particles then we can tweak the position of particles to make them look like samples drawn from the posterior.



To do so, we need a criterion to check whether the particles represent the posteior.

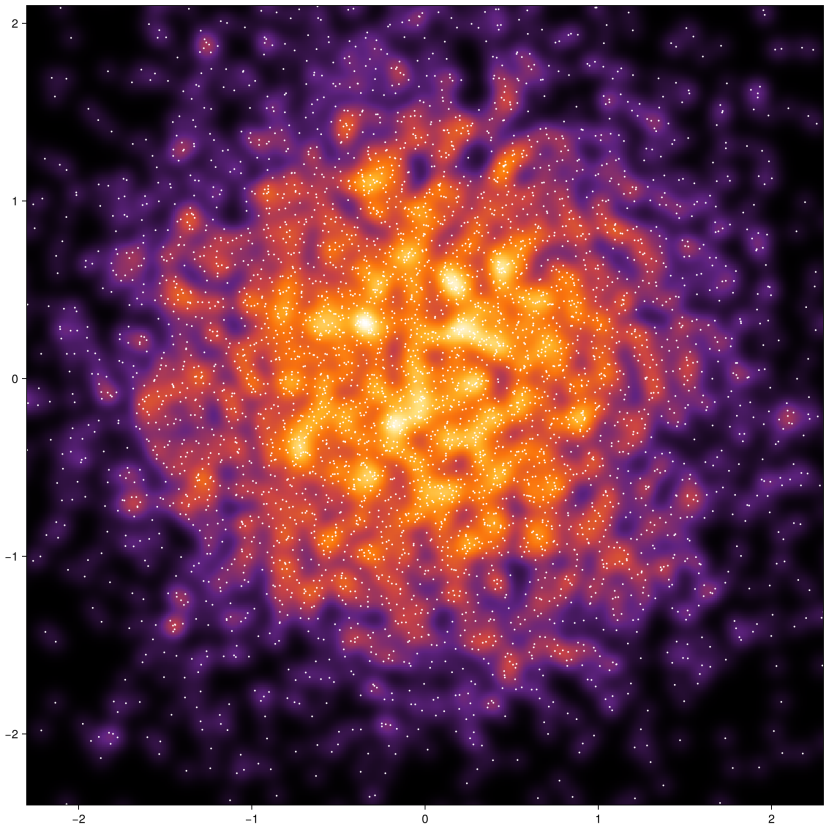
Stein Variational Gradient Descent



Charles Stein

(1920-2016)

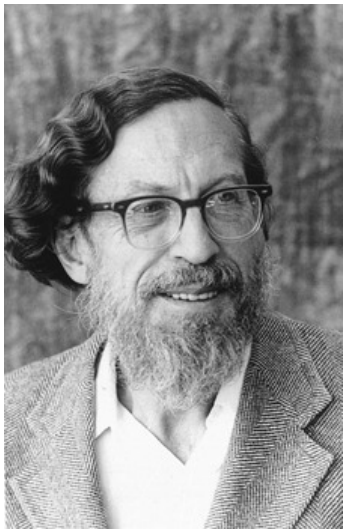
Stein's method, a technique for bounding the distance of two distributions, was discovered by Charles Stein when he was working on a proof for CLT.



$$\mathrm{L}_\pi \phi(x) = \phi(x) \nabla_x \log \pi(x)^T + \nabla_x \phi(x)$$

$$\mathbb{E}_{x \sim \pi}[\mathrm{L}_\pi \phi(x)] = 0$$

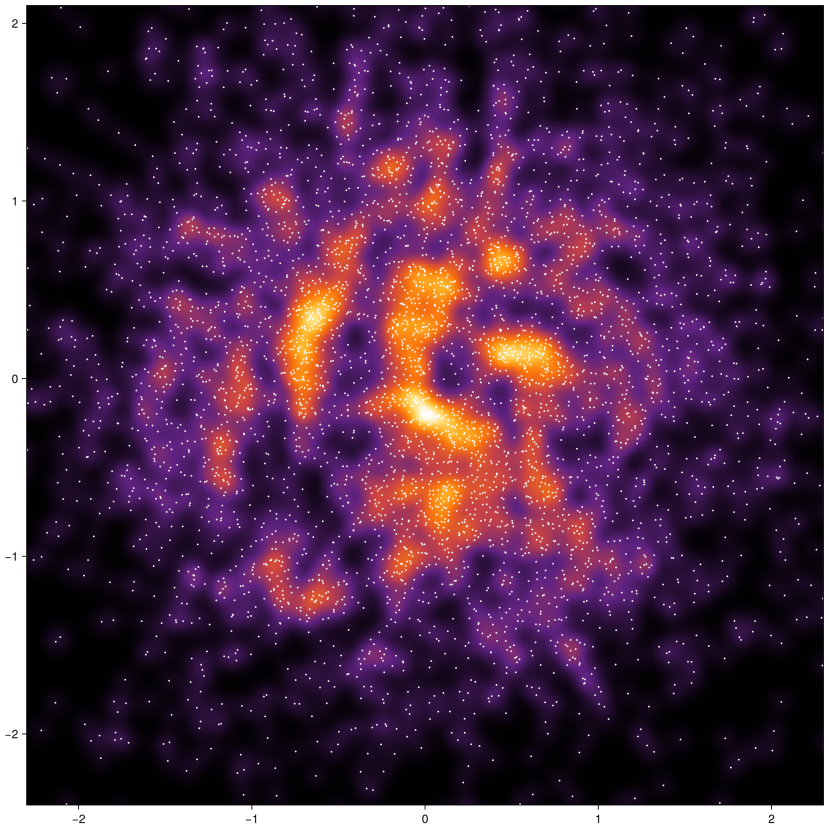
Stein Variational Gradient Descent



Charles Stein

(1920-2016)

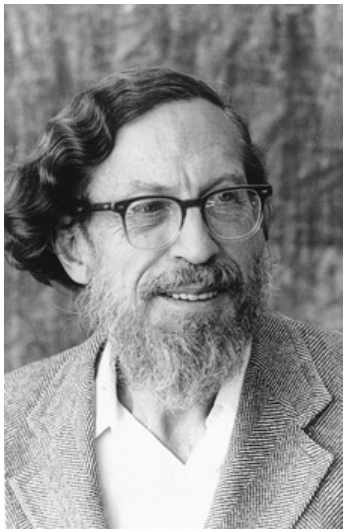
Stein's method, a technique for bounding the distance of two distributions, was discovered by Charles Stein when he was working on a proof for CLT.



$$\mathrm{L}_\pi \phi(x) = \phi(x) \nabla_x \log \pi(x)^T + \nabla_x \phi(x)$$

$$\mathbb{E}_{x \sim \pi}[\mathrm{L}_\pi \phi(x)] = 0$$

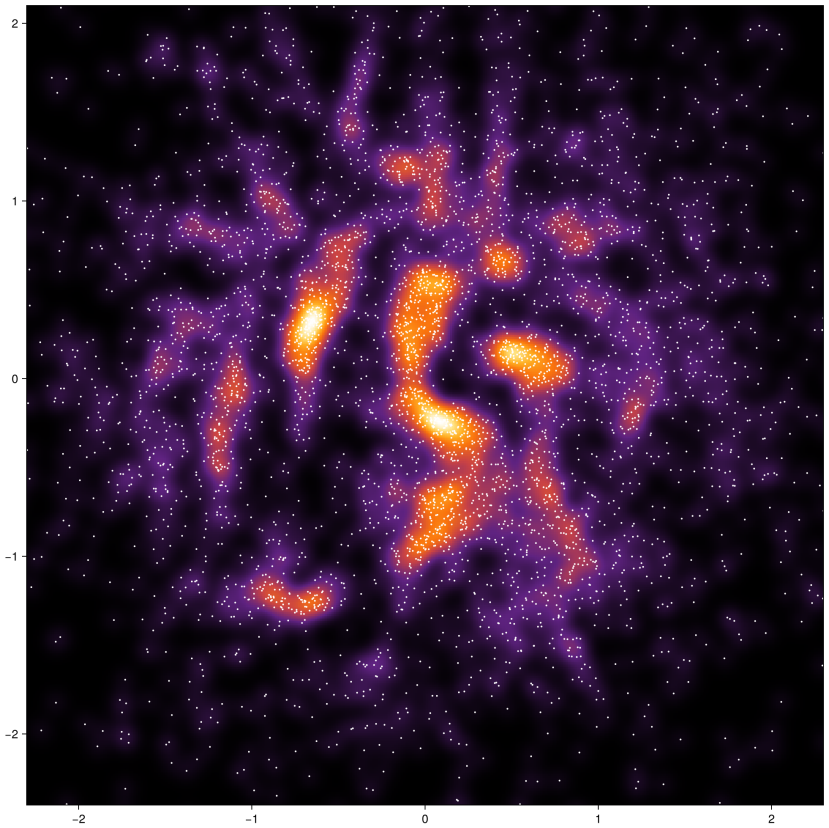
Stein Variational Gradient Descent



Charles Stein

(1920-2016)

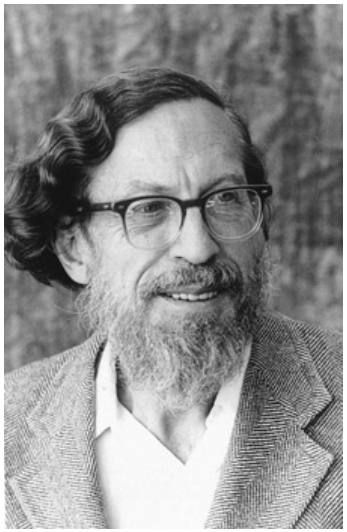
Stein's method, a technique for bounding the distance of two distributions, was discovered by Charles Stein when he was working on a proof for CLT.



$$\mathrm{L}_\pi \phi(x) = \phi(x) \nabla_x \log \pi(x)^T + \nabla_x \phi(x)$$

$$\mathbb{E}_{x \sim \pi}[\mathrm{L}_\pi \phi(x)] = 0$$

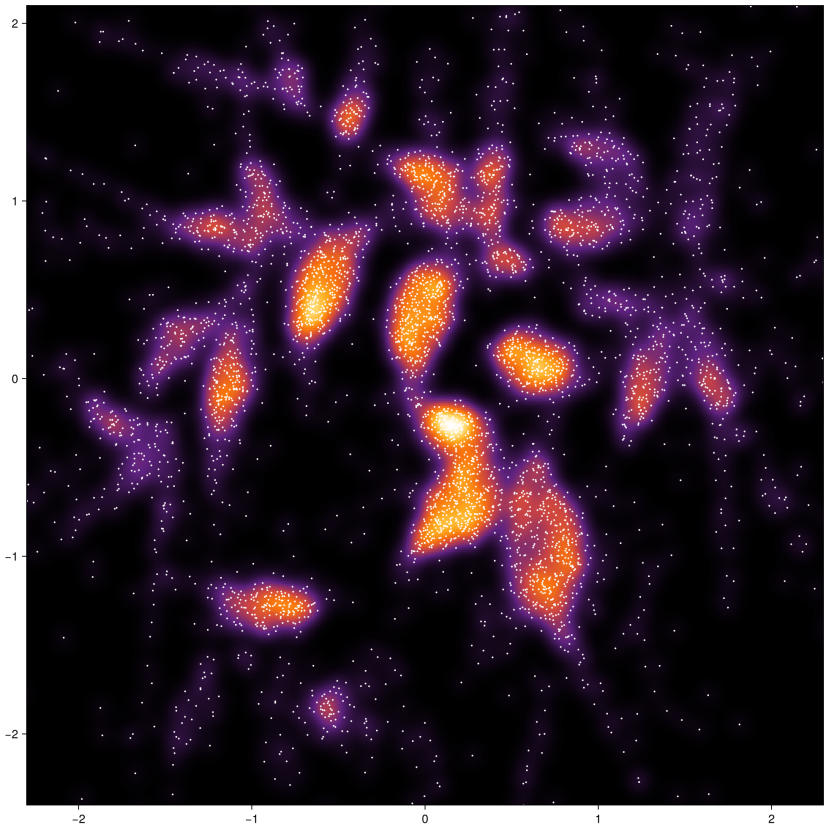
Stein Variational Gradient Descent



Charles Stein

(1920-2016)

Stein's method, a technique for bounding the distance of two distributions, was discovered by Charles Stein when he was working on a proof for CLT.

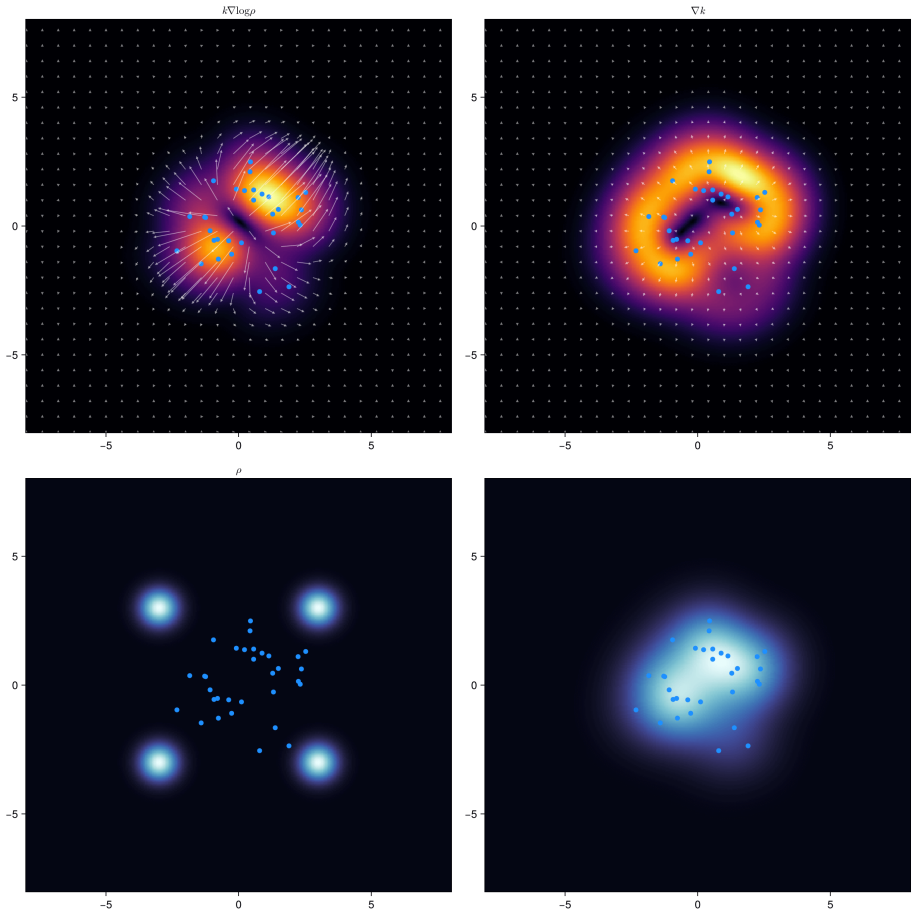


$$\mathrm{L}_\pi \phi(x) = \phi(x) \nabla_x \log \pi(x)^T + \nabla_x \phi(x)$$

$$\mathbb{E}_{x \sim \pi}[\mathrm{L}_\pi \phi(x)] = 0$$

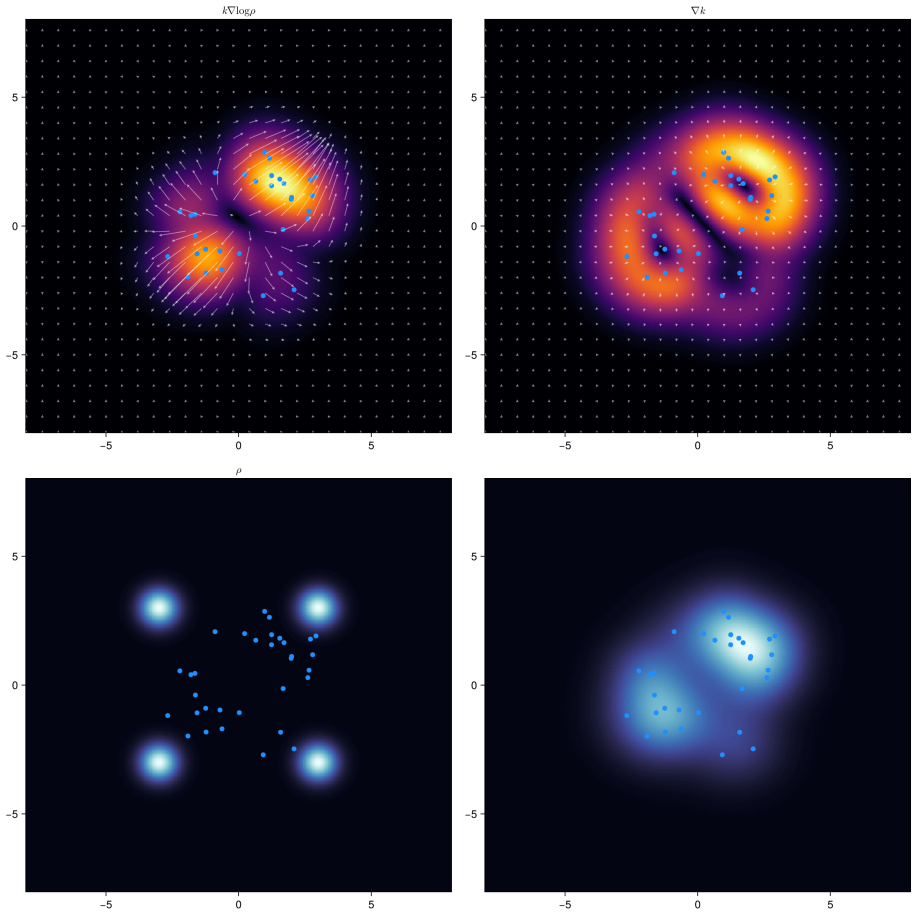
Stein Variational Gradient Descent

```
function stein_∇(P, x)
    ∇ = zeros(d)
    for i ∈ 1:N
        ∇ += k(P[:, i]) * ∇logπ(P[:, i]) + ∇k(P[:, i], x)
    end
    return ∇ / N
end

function stein_variational_gradient_descent!(P, η)
    for i ∈ 1:N
        P[:, i] += η * stein_∇(P, P[:, i])
    end
end
```



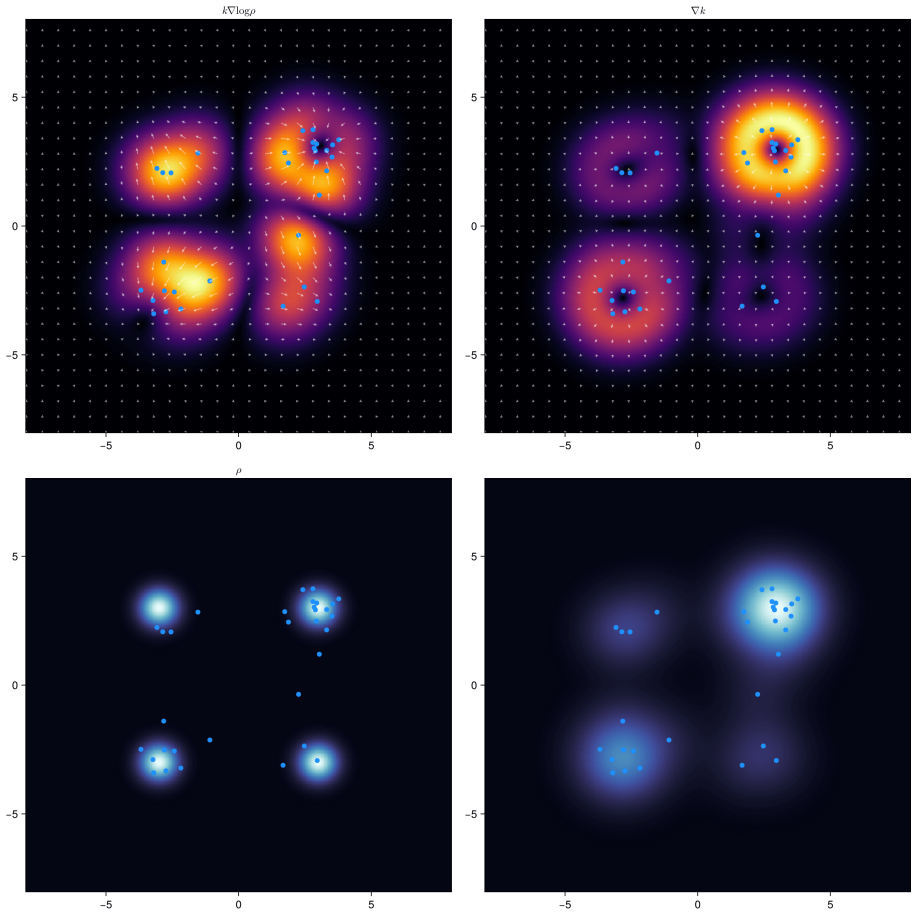
Stein Variational Gradient Descent

```julia
function stein_∇(P, x)
    ∇ = zeros(d)
    for i ∈ 1:N
        ∇ += k(P[:, i]) * ∇logπ(P[:, i]) + ∇k(P[:, i], x)
    end
    return ∇ / N
end

function stein_variational_gradient_descent!(P, η)
    for i ∈ 1:N
        P[:, i] += η * stein_∇(P, P[:, i])
    end
end
```



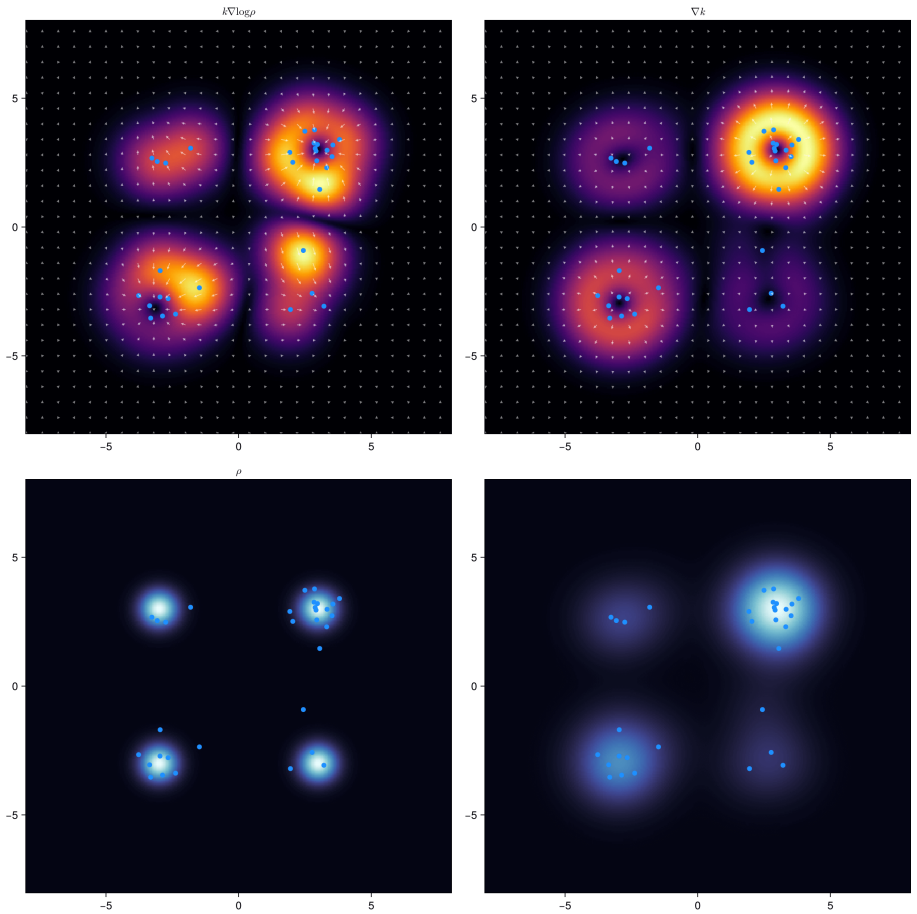
Stein Variational Gradient Descent

```julia
function stein_∇(P, x)
    ∇ = zeros(d)
    for i ∈ 1:N
        ∇ += k(P[:, i]) * ∇logπ(P[:, i]) + ∇k(P[:, i], x)
    end
    return ∇ / N
end

function stein_variational_gradient_descent!(P, η)
    for i ∈ 1:N
        P[:, i] += η * stein_∇(P, P[:, i])
    end
end
```

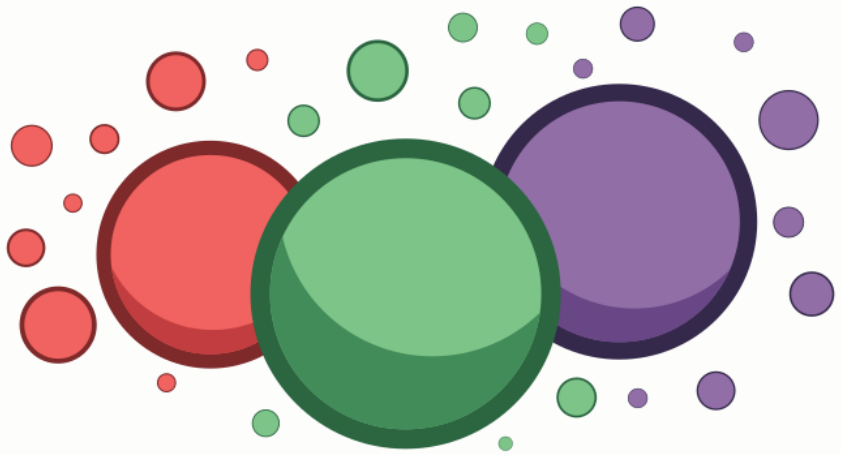


Stein Variational Gradient Descent

```
function stein_∇(P, x)
    ∇ = zeros(d)
    for i ∈ 1:N
        ∇ += k(P[:, i]) * ∇logπ(P[:, i]) + ∇k(P[:, i], x)
    end
    return ∇ / N
end

function stein_variational_gradient_descent!(P, η)
    for i ∈ 1:N
        P[:, i] += η * stein_∇(P, P[:, i])
    end
end
```



Nonparametric Variational Inference in Julia



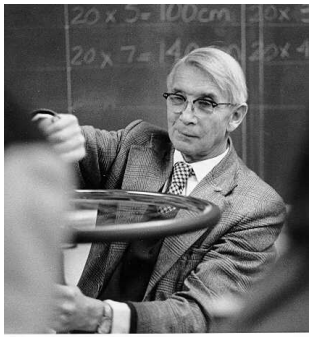
NonparametricVI.jl

Nonparametric Variational Inference for Probabilistic Programming

The Ising model and Bethe approximation

$$H(\sigma) = -\sum_{i \sim j} \sigma_i \sigma_j$$

$$p_\beta(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z(\beta)}$$

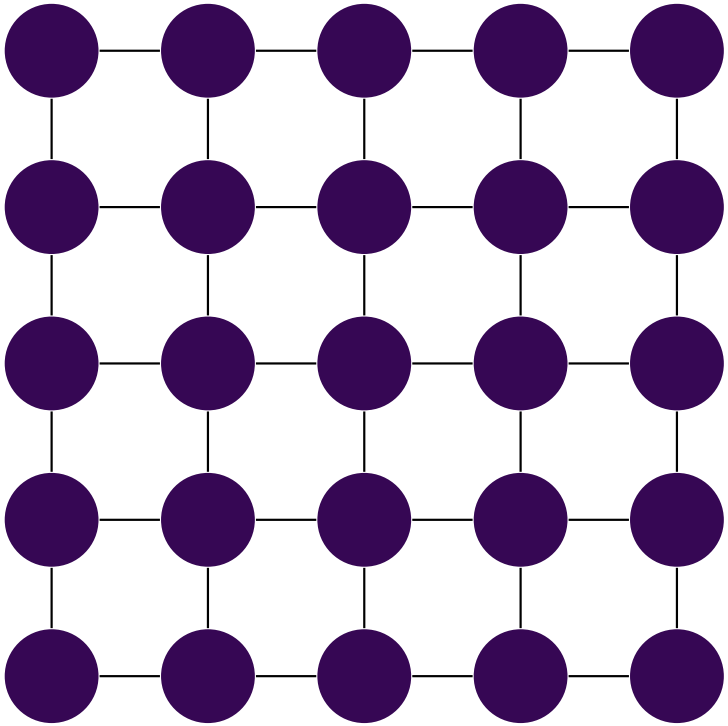


Ernst Ising

(1900-1998)

Hans Bethe

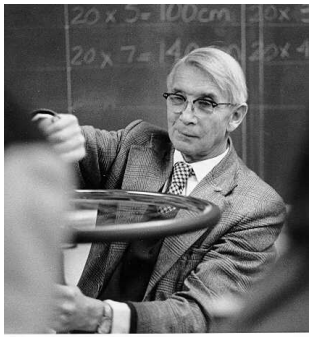
(1906-2005)



The Ising model and Bethe approximation

$$H(\sigma) = -\sum_{i \sim j} \sigma_i \sigma_j$$

$$p_\beta(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z(\beta)}$$

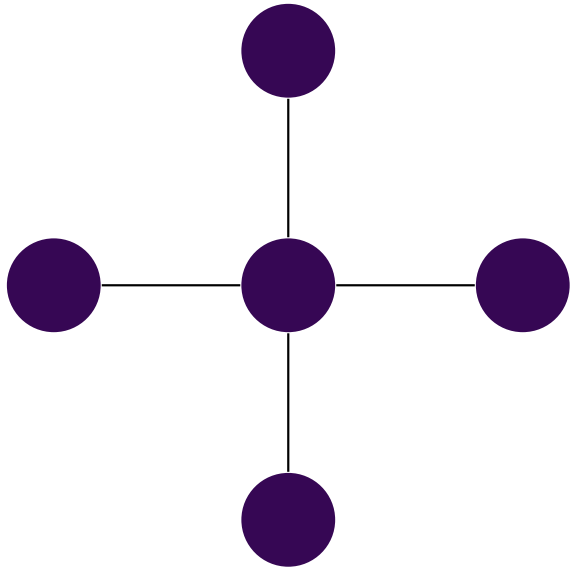


Ernst Ising

(1900-1998)

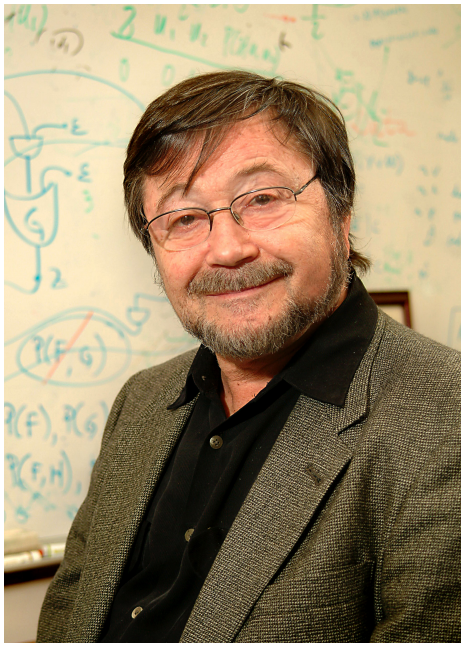
Hans Bethe

(1906-2005)



Can we approximate with interaction of a node and the rest of the lattice by the above graph?

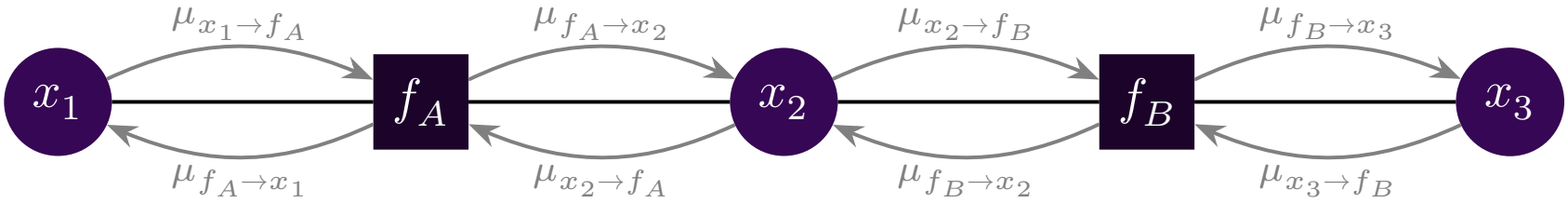
Belief Propagation



Judea Pearl (1936-Present)

If we translate probabilistic programs to *factor graphs*, with an idea similar to Bethe approximation, we can localize the inference (marginalization) problem.

$$p_{x_v}(x_v) = \prod_{a \in N(v)} \mu_{a \to v}(x_v)$$



Variational Message Passing

- BP is exact for factor graphs with no cycle

Variational Message Passing

- BP is exact for factor graphs with no cycle
- BP is still useful for cyclic graphs, of course as an approximation (Loopy Belief Propagation).

Variational Message Passing

- BP is exact for factor graphs with no cycle
- BP is still useful for cyclic graphs, of course as an approximation (Loopy Belief Propagation).
- BP breaks the inference problem into smaller local problems.

Variational Message Passing

- BP is exact for factor graphs with no cycle
- BP is still useful for cyclic graphs, of course as an approximation (Loopy Belief Propagation).
- BP breaks the inference problem into smaller local problems.
- We can solve the smaller problems exactly if possible or by hiring previously discussed methods like variational inference.

Variational Message Passing

Variational Message Passing in Julia

Bayesian Inference with Reactive Message Passing[3]

[3] Reactive Message Passing for Scalable Bayesian Inference, Dmitry Bagaev, Bert de Vries, `https://arxiv.org/abs/2112.13251`

Variational Message Passing in Julia

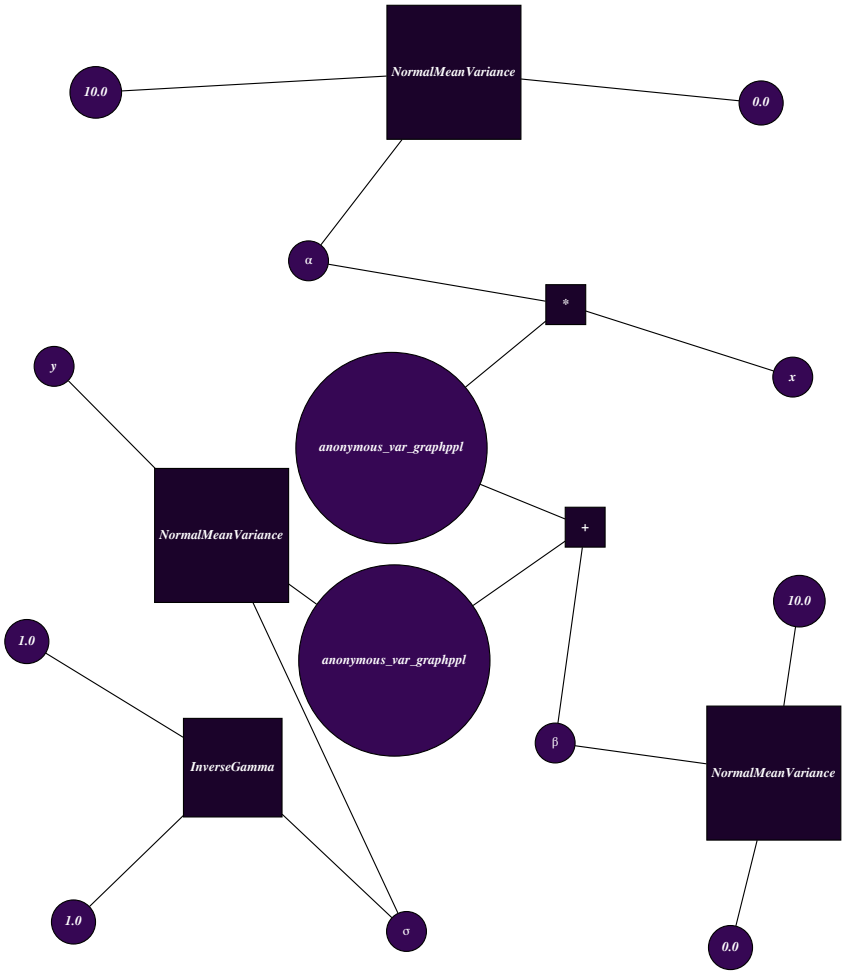
As an example, consider a Bayesian linear regression model with latent variance. RxInfer uses `GraphPPL.jl` with a syntax similar to `DynamicPPL.jl` used by Turing.

```
@model function linear_regression(x, y)
    α ~ Normal(mean = 0.0, variance = 10.0)
    β ~ Normal(mean = 0.0, variance = 10.0)
    σ ~ InverseGamma(1.0, 1.0)

    y .~ Normal(mean = α .* x .+ β, variance = σ)
end
```

Variational Message Passing in Julia

```
@model function linear_regression(x, y)
    α ~ Normal(mean=0.0, variance=10.0)
    β ~ Normal(mean=0.0, variance=10.0)
    σ ~ InverseGamma(1.0, 1.0)

    y .~ Normal(mean=α.*x.+β, variance=σ)
end
```

Unlike Turing, RxInfer converts the probabilistic programs to factor graphs. For the Bayesian regression example, the factor graph for a single pair of observations is shown here. Note that for more than one observations, the factor graphs becomes cyclic.



Variational Message Passing in Julia

Before inference, we need to specify initial messages (for Loopy BP) and independence assumptions (for variational inference).

```
@initialization function inference_init()
    μ(β) = NormalMeanVariance(0.0, 10.0)
    q(σ) = vague(InverseGamma)
end

result = infer(
    model = linear_regression(),
    data = (x=x, y=y),
    constraints = MeanField(),
    initialization = inference_init(),
    iterations = 50
)
```

Approximate Bayesian Inference in Julia

- AdvancedMH.jl : Metropolis-Hastings, Langevin Monte Carlo
- AdvancedHMC.jl : Hamiltonian Monte Carlo, NUTS
- AdvancedVI.jl : Variational Inference
- NonparametricVI.jl : Nonparametric Variational Inference
- ReactiveMP.jl : Variational Message Passing (used by RxInfer.jl)
- Pigeons.jl : Parallel tempering
- ThermodynamicIntegration.jl : Approximating evidence using Thermodynamic Integration

Next session

In the next lecture, we will see how to use probabilistic models for some decision-making problems.