

# Probabilistic Programming and AI: Lecture 3

## Markov Chain Monte Carlo Inference

---

Markus Böck and Jürgen Cito

Research Unit of Software Engineering

# Table of contents

1. Independent versus Dependent Sampling
2. Metropolis Hastings Algorithm
3. Metropolis Hastings Algorithm For PPLs
4. Hamiltonian Monte Carlo

# Independent versus Dependent Sampling

---

# Why sampling?

Prior  $P(\Theta)$

- chosen by modeller
- **computable**

Likelihood  $P(X|\Theta)$

- encodes generative process from latents  $\Theta$  to observes  $X$
- **computable**

Marginal / Evidence  $P(X)$

- can be **approximated** with sampling methods

Posterior  $P(\Theta|X) = \frac{P(X|\Theta) \times P(\Theta)}{P(X)}$

- what we want to know
- can be **approximated** with sampling methods

# Limitations of Independent Samples

## Rejection Sampling:

Curse of dimensionality / continuous variables

## Importance Sampling:

How to specify reference distribution?

# Rejection Sampling

Coin model:

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

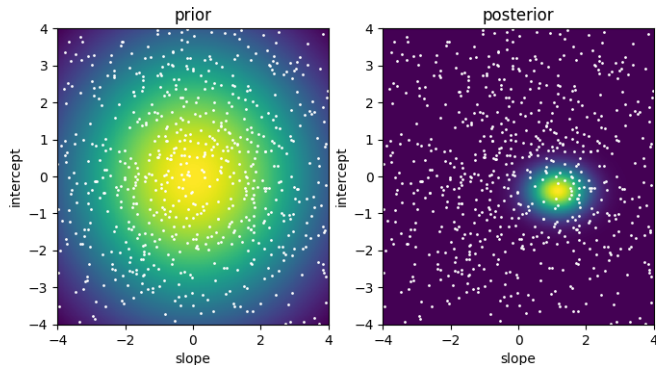
$$x_i \sim \text{Bernoulli}(p)$$

```
1 def rejection_sampling(xs_observed):  
2     while True:  
3         p = dist.Uniform(0,1).sample()  
4         xs = dist.Bernoulli(p).sample(xs_observed.shape)  
5         if (xs == xs_observed).all():  
6             return p # accept  
7         # reject
```

*Rejected 50.54% of iterations for 1 number of observations.  
Rejected 83.78% of iterations for 2 number of observations.  
Rejected 91.76% of iterations for 3 number of observations.  
Rejected 96.68% of iterations for 4 number of observations.  
Rejected 98.26% of iterations for 5 number of observations.*

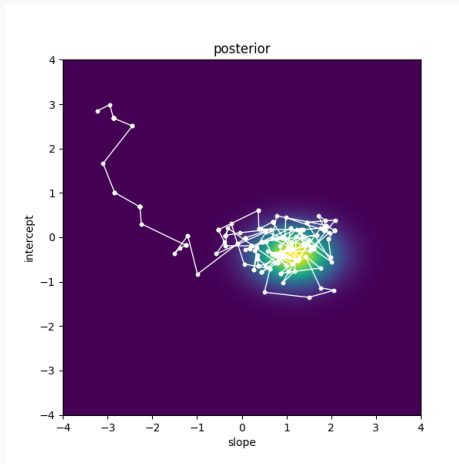
# Likelihood Weighting

Sample from prior, reweigh to approximate posterior.



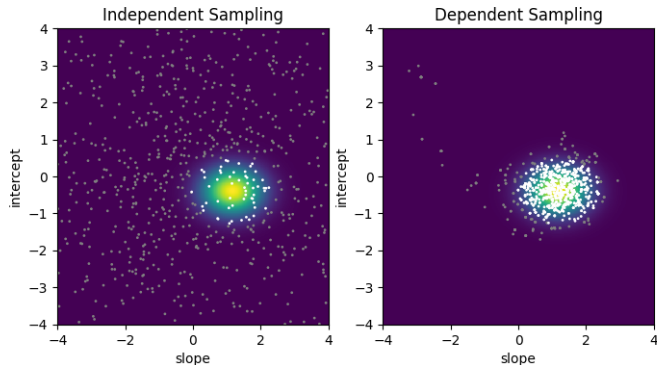
Most samples from reference distribution are of low probability.

# Dependent Sampling Idea



Perturb the current sample to generate new sample.

# Independent versus Dependent Sampling



# Dependent Sampling Idea

$$P(\Theta|X) = \frac{P(X|\Theta) \times P(\Theta)}{P(X)} \propto P(X|\Theta) \times P(\Theta)$$

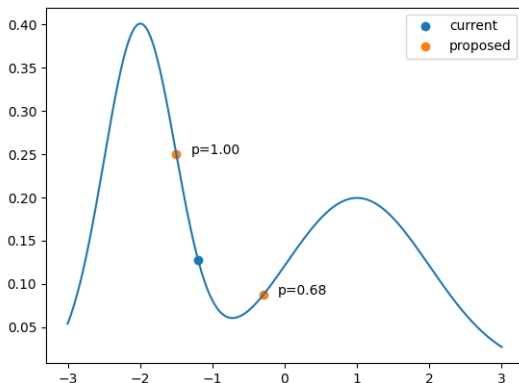
As we will see, we can generate dependent samples of the posterior by only evaluating the joint.

# Metropolis Hastings Algorithm

---

# Metropolis Hastings Algorithm

"Hill climbing with the possibility of stepping down."



$$p = \min \left( \frac{f(x_{proposed})}{f(x_{current})}, 1 \right)$$

# Metropolis Hastings Algorithm

Let  $P(x)$  be the target distribution and  $Q(x'|x)$  be a proposal distribution.

1. Initialise  $x_0$ .
2. For each  $i = 1, \dots, n$ 
  - Propose a new value according to the proposal  $x' \sim Q(\cdot|x_i)$
  - Calculate the acceptance probability

$$A = \min \left( 1, \frac{P(x')Q(x_i|x')}{P(x_i)Q(x'|x_i)} \right)$$

- Draw a random number  $0 \leq p \leq 1$  and let

$$x_{i+1} = \begin{cases} x' & p \leq A \quad (\text{accept}) \\ x_i & p > A \quad (\text{reject}) \end{cases}$$

The Metropolis Hastings algorithm produces a so-called **Markov Chain**, where each value only depends on its predecessor (and not multiple predecessors).

## Soundness:

It can be shown that for sensible  $Q$  and  $n \rightarrow \infty$ , the resulting chain is indistinguishable from a sample from the target distribution  $P$ .

# Metropolis Hastings Algorithm

## Proposal Distributions

- Should propose values that are accepted frequently, but also explore the entire state space.
- Random walk (Gaussian drift) proposals:

$$x' \sim \text{Normal}(x_i, \sigma)$$

- Unconditional proposals:

$$x' \sim Q(.) \quad (\text{no dependence on } x_i)$$

- Example nonsensical proposal:

$$x' \sim \text{Uniform}(x_i, x_i + 1)$$

- Best proposal distribution would be the target itself  
 $Q(x'|x_i) = P(x')$  (not available)

# Metropolis Hastings Algorithm

## Ad Soundness:

Let  $x_{i+1} \sim T(x_{i+1}|x_i)$  be the transition kernel, as defined by the algorithm ( $x_{i+1} = x'$  with probability  $A$ , ...).

This kernel satisfies the detailed-balance condition

$$P(x_i)T(x_{i+1}|x_i) = P(x_{i+1})T(x_i|x_{i+1}),$$

the probability of being in state  $x_i$  and transitioning to state  $x_{i+1}$  must be equal to the probability of being in state  $x_{i+1}$  and transitioning to state  $x_i$ .

Also, if the proposal distribution is chosen such that the resulting Markov chain is ergodic, then the stationary distribution of the Markov chain is  $P$ .

Informally, any state should be reachable from any other state in any number of steps less or equal to a number  $N$ .

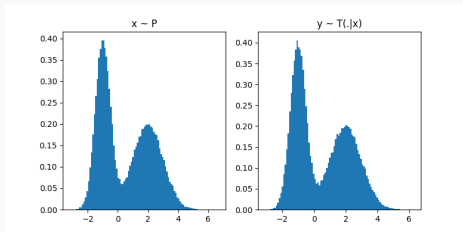
# Metropolis Hastings Algorithm

## Ad Soundness: Stationary distribution

$$\int T(y|x)P(x)dx = \int T(x|y)P(y)dx = P(y)$$

```
def T(x, P, sigma):  
    P_current = P(x)  
    proposed = dist.Normal(x, sigma).sample()  
    P_proposed = P(proposed)  
  
    A = P_proposed/P_current  
  
    if torch.rand(()) < A:  
        y = proposed  
    else:  
        y = x  
  
    return y
```

```
X = sample_P(-1,2,100000)  
Y = torch.tensor([T(x, P, 0.5) for x in X])
```



# Metropolis Hastings Algorithm

Why does it work for Bayesian Inference?

Target is  $\tilde{P}(\Theta) = P(\Theta|X)$  which we cannot evaluate.

But, with Bayes Theorem:

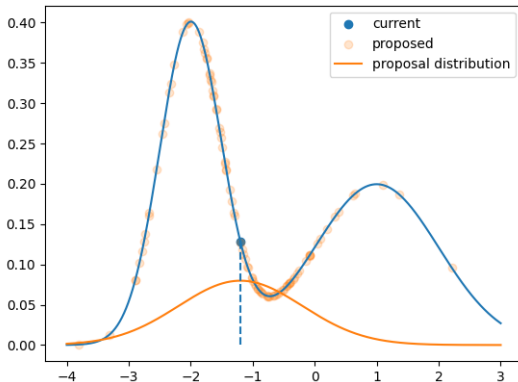
$$\begin{aligned}\frac{\tilde{P}(\theta') \times Q(\theta_i|\theta')}{\tilde{P}(\theta_i) \times Q(\theta'|\theta_i)} &= \frac{\frac{P(X|\theta')P(\theta')}{P(X)} \times Q(\theta_i|\theta')}{\frac{P(X|\theta)P(\theta)}{P(X)} \times Q(\theta'|\theta_i)} \\ &= \frac{P(X|\theta')P(\theta') \times Q(\theta_i|\theta')}{P(X|\theta)P(\theta) \times Q(\theta'|\theta_i)}\end{aligned}$$

The normalisation constant, the marginal  $P(X)$ , cancels!

We only need to be able to evaluate the joint to apply the Metropolis Hastings algorithm to the posterior!

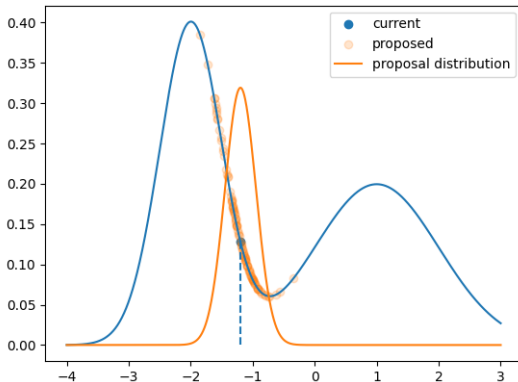
# Metropolis Hastings Algorithm

Random walk proposal  $x' \sim \text{Normal}(x_i, \sigma)$



# Metropolis Hastings Algorithm

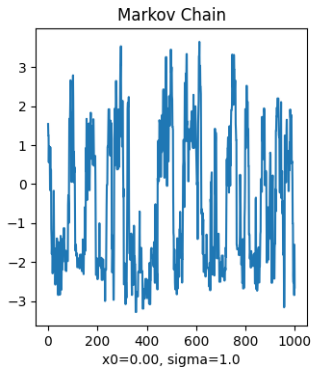
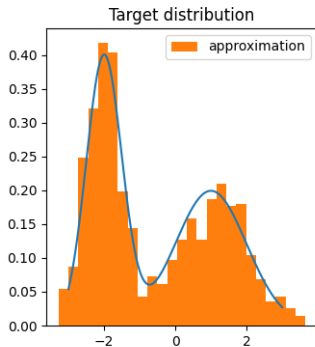
Random walk proposal  $x' \sim \text{Normal}(x_i, \sigma)$



# Metropolis Hastings Algorithm

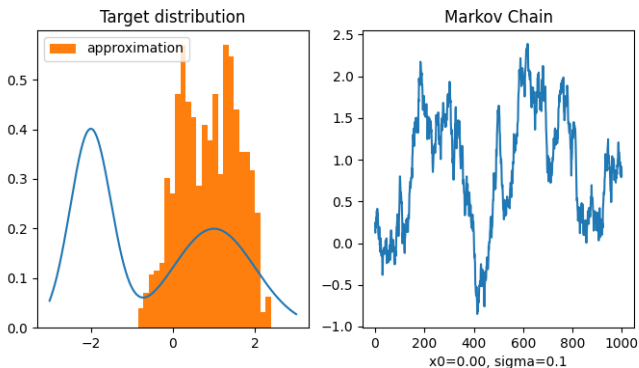
Example Markov Chain generated by random walk proposal

$$x' \sim \text{Normal}(x_i, \sigma)$$



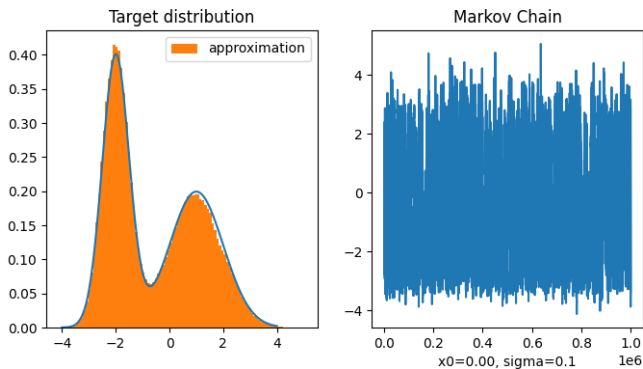
# Metropolis Hastings Algorithm

Example Markov Chain generated by random walk proposal  
 $x' \sim \text{Normal}(x_i, \sigma)$ : step-size too small



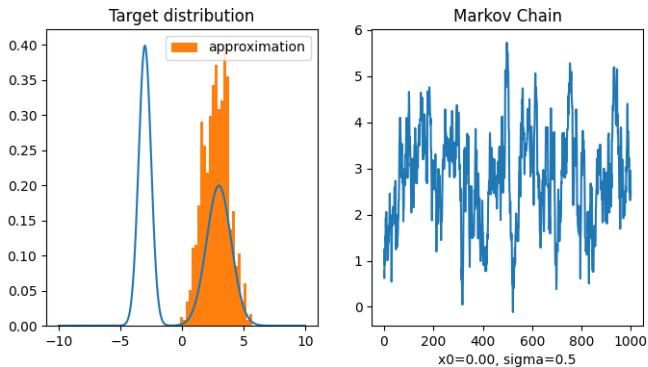
# Metropolis Hastings Algorithm

Example Markov Chain generated by random walk proposal  
 $x' \sim \text{Normal}(x_i, \sigma)$ : large number of iterations



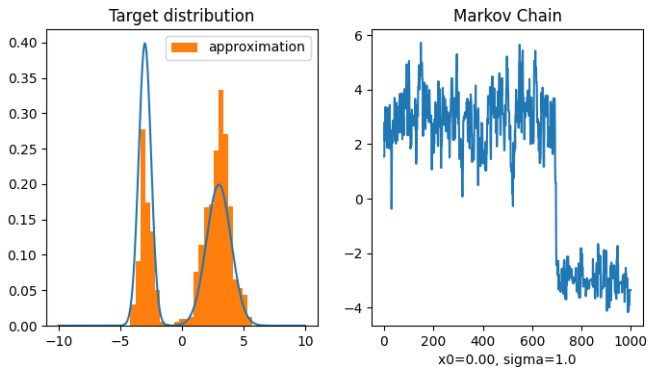
# Metropolis Hastings Algorithm

Example Markov Chain generated by random walk proposal  
 $x' \sim \text{Normal}(x_i, \sigma)$ : cannot bridge gaps in target density



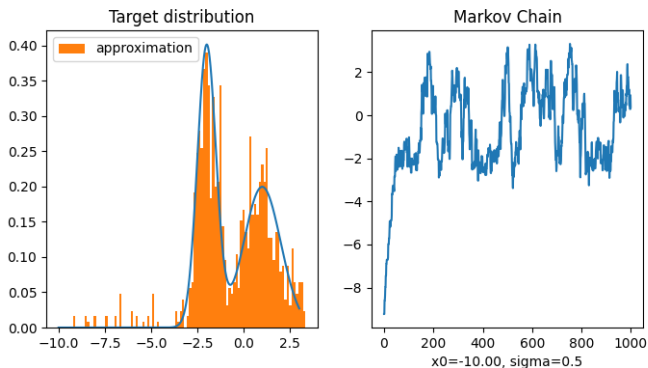
# Metropolis Hastings Algorithm

Example Markov Chain generated by random walk proposal  
 $x' \sim \text{Normal}(x_i, \sigma)$ : increased step-size



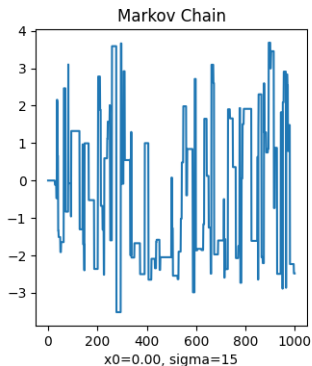
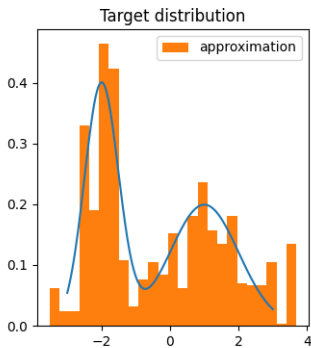
# Metropolis Hastings Algorithm

Example Markov Chain generated by random walk proposal  
 $x' \sim \text{Normal}(x_i, \sigma)$ : bad initial state, "burn-in" phase



# Metropolis Hastings Algorithm

Example Markov Chain generated by random walk proposal  
 $x' \sim \text{Normal}(x_i, \sigma)$ : step-size too large



# Metropolis Hastings Algorithm For PPLs

---

# Metropolis Hastings Algorithm For PPLs

```
def noisy_geometric(p):  
    x = 0  
    while True:  
        b = sample(f"b_{x}", dist.Bernoulli(p))  
        if b:  
            break  
        x += 1  
    y = sample("y", dist.Normal(x,1), observed=torch.tensor(3))  
    return x
```

Sample  $b_0, b_1, b_2, \dots$  until  $b_n = 1$

Then observe  $y$ .

# Metropolis Hastings Algorithm For PPLs

```
torch.manual_seed(0)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
Returns:
{'b_0': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_1': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_2': {'value': tensor(1.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-1.3863)},
'y': {'value': tensor(3),
'distribution': Normal(loc: 2.0, scale: 1.0),
'is_observed': True,
'log_prob': tensor(-1.4189)}}
```

## How to setup proposal distribution for PPL?

```
torch.manual_seed(1)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
Returns:
{'b_0': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_1': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_2': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_3': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_4': {'value': tensor(1.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-1.3863)},
'y': {'value': tensor(3),
'distribution': Normal(loc: 4.0, scale: 1.0),
'is_observed': True,
'log_prob': tensor(-1.4189)}}
```

# Metropolis Hastings Algorithm For PPLs

Idea: update one variable at a time.

# Metropolis Hastings Algorithm For PPLs

```
def linear_regression(x, y):  
    slope = sample("slope", dist.Normal(0,3))  
    intercept = sample("intercept", dist.Normal(0,3))  
    for i in range(len(x)):  
        sample(f"y_{i}", dist.Normal(slope*x[i]+intercept, 1.), observed=y[i])
```

## Chosen address: `intercept`

$$Q(prop.|curr.) = \frac{1}{2}Q(intercept'|intercept), Q(curr.|prop.) = \frac{1}{2}Q(inter.|inter.)$$

Current:

```
{'slope': {'value': tensor(4.6230),  
  'distribution': Normal(loc: 0.0, scale: 3.0),  
  'is_observed': False,  
  'log_prob': tensor(-3.2049)},  
'intercept': {'value': tensor(-0.8803), # changes  
  'distribution': Normal(loc: 0.0, scale: 3.0),  
  'is_observed': False,  
  'log_prob': tensor(-2.0606)}, # changes  
'y_0': {'value': tensor(-1.2000),  
  'distribution': Normal(loc:-5.5032,scale:1.0), # changes  
  'is_observed': True,  
  'log_prob': tensor(-10.1780)}, # changes  
  ...  
}
```

Proposed:

```
{'slope': {'value': tensor(4.6230),  
  'distribution': Normal(loc: 0.0, scale: 3.0),  
  'is_observed': False,  
  'log_prob': tensor(-3.2049)},  
'intercept': {'value': tensor(-0.4119), # changes  
  'distribution': Normal(loc: 0.0, scale: 3.0),  
  'is_observed': False,  
  'log_prob': tensor(-2.0270)}, # changes  
'y_0': {'value': tensor(-1.2000),  
  'distribution': Normal(loc:-5.0349,scale:1.0), # changes  
  'is_observed': True,  
  'log_prob': tensor(-8.2722)}, # changes  
  ...  
}
```

# Metropolis Hastings Algorithm For PPLs

```
def linear_regression(x, y):  
    slope = sample("slope", dist.Normal(0,3))  
    intercept = sample("intercept", dist.Normal(0,3))  
    for i in range(len(x)):  
        sample(f"y_{i}", dist.Normal(slope*x[i]+intercept, 1.), observed=y[i])
```

## Chosen address: slope

$$Q(prop.|curr.) = \frac{1}{2}Q(slope'|slope), \quad Q(curr.|prop.) = \frac{1}{2}Q(slope|slope')$$

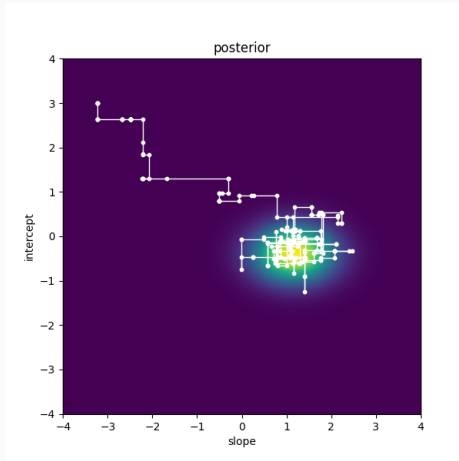
Current:

```
{'slope': {'value': tensor(4.6230), # changes  
'distribution': Normal(loc: 0.0, scale: 3.0),  
'is_observed': False,  
'log_prob': tensor(-3.2049)}, # changes  
'intercept': {'value': tensor(-0.8803),  
'distribution': Normal(loc: 0.0, scale: 3.0),  
'is_observed': False,  
'log_prob': tensor(-2.0606)},  
'y_0': {'value': tensor(-1.2000),  
'distribution': Normal(loc:-5.5032,scale:1.0), # changes  
'is_observed': True,  
'log_prob': tensor(-10.1780)}, # changes  
...  
}
```

Proposed:

```
{'slope': {'value': tensor(1.9841), # changes  
'distribution': Normal(loc: 0.0, scale: 3.0),  
'is_observed': False,  
'log_prob': tensor(-2.2362)}, # changes  
'intercept': {'value': tensor(-0.8803),  
'distribution': Normal(loc: 0.0, scale: 3.0),  
'is_observed': False,  
'log_prob': tensor(-2.0606)},  
'y_0': {'value': tensor(-1.2000),  
'distribution': Normal(loc:-2.8643,scale:1.0), # changes  
'is_observed': True,  
'log_prob': tensor(-2.3041)}, # changes  
...  
}
```

# Metropolis Hastings Algorithm For PPLs



# Metropolis Hastings Algorithm For PPLs

```
def noisy_geometric(p):  
    x = 0  
    while True:  
        b = sample(f"b_{x}", dist.Bernoulli(p))  
        if b:  
            break  
        x += 1  
    y = sample("y", dist.Normal(x,1), observed=torch.tensor(3))  
    return x
```

Sample  $b_0, b_1, b_2, \dots$  until  $b_n = 1$

Then observe  $y$ .

# Metropolis Hastings Algorithm For PPLs

```
torch.manual_seed(0)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
Returns:
{'b_0': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_1': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_2': {'value': tensor(1.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-1.3863)},
'y': {'value': tensor(3),
'distribution': Normal(loc: 2.0, scale: 1.0),
'is_observed': True,
'log_prob': tensor(-1.4189)}}
```

How to propose from left to right  
and vice-versa?

```
torch.manual_seed(1)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
Returns:
{'b_0': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_1': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_2': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_3': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_4': {'value': tensor(1.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-1.3863)},
'y': {'value': tensor(3),
'distribution': Normal(loc: 4.0, scale: 1.0),
'is_observed': True,
'log_prob': tensor(-1.4189)}}
```

# Single-Site Metropolis Hastings Algorithm For PPLs

- Choose a single variable  $X_0$  at random, for which we make a conditional proposal.
- For all new variables, propose from prior.
- For all other variables, reuse value from current trace.
- The proposal probability can be calculated as follows

$$Q(\text{prop.}|\text{curr.}) = \underbrace{\frac{1}{|\text{curr.}|}}_{\text{Pr. of choosing } X_0} \times \underbrace{Q(X'_0|X_0)}_{\text{Conditional proposal for } X_0} \times \underbrace{\prod_{X \in \text{sampler}} P(X)}_{\text{Pr. of proposing new variables from priors}}$$

- New variables can be determined with

$$\text{sampler} = \text{prop.} \setminus \text{curr.},$$

i.e. all variables that are in proposed trace but not in current trace.

# Metropolis Hastings Algorithm For PPLs

```
torch.manual_seed(0)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
Returns:
{'b_0': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_1': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_2': {'value': tensor(1.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-1.3863)},
'y': {'value': tensor(3),
'distribution': Normal(loc: 2.0, scale: 1.0),
'is_observed': True,
'log_prob': tensor(-1.4189)}}
```

- Chosen address is  $b_2$ .
- $b_0$  and  $b_1$  are reused.
- $b_3$  and  $b_4$  are sampled from prior (left to right) or forgotten (right to left).
- $y$  is observed.

```
torch.manual_seed(1)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
Returns:
{'b_0': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_1': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_2': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_3': {'value': tensor(0.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-0.2877)},
'b_4': {'value': tensor(1.),
'distribution': Bernoulli(probs: 0.25),
'is_observed': False,
'log_prob': tensor(-1.3863)},
'y': {'value': tensor(3),
'distribution': Normal(loc: 4.0, scale: 1.0),
'is_observed': True,
'log_prob': tensor(-1.4189)}}
```

# Metropolis Hastings Algorithm For PPLs

```
def model():
    X = sample("X", dist.Normal(0,1))
    Y = sample("Y", dist.Normal(X,1))
    if Y < 0:
        sample("A", dist.Normal(0,1), observed=torch.tensor(1.))
    else:
        sample("B", dist.Normal(0,1))
```

Chosen address: X

$$Q(prop.|curr.) = \frac{1}{3}Q(X'|X), \quad Q(curr.|prop.) = \frac{1}{3}Q(X|X')$$

Current:

```
{'X': {'value': tensor(1.5410), # changes
'distribution': Normal(loc: 0.0, scale: 1.0),
'is_observed': False,
'log_prob': tensor(-2.1063)}, # changes
'Y': {'value': tensor(1.2476),
'distribution': Normal(loc: 1.5410, scale: 1.0),
'is_observed': False,
'log_prob': tensor(-0.9620)}, # changes
'B': {'value': tensor(-2.1788),
'distribution': Normal(loc: 0.0, scale: 1.0),
'is_observed': False,
'log_prob': tensor(-3.2925)}}
```

Proposed:

```
{'X': {'value': tensor(1.3281), # changes
'distribution': Normal(loc: 0.0, scale: 1.0),
'is_observed': False,
'log_prob': tensor(-1.8009)}, # changes
'Y': {'value': tensor(1.2476),
'distribution': Normal(loc: 1.3281, scale: 1.0),
'is_observed': False,
'log_prob': tensor(-0.9222)}, # changes
'B': {'value': tensor(-2.1788),
'distribution': Normal(loc: 0.0, scale: 1.0),
'is_observed': False,
'log_prob': tensor(-3.2925)}}
```

# Metropolis Hastings Algorithm For PPLs

```
def model():
    X = sample("X", dist.Normal(0,1))
    Y = sample("Y", dist.Normal(X,1))
    if Y < 0:
        sample("A", dist.Normal(0,1), observed=torch.tensor(1.))
    else:
        sample("B", dist.Normal(0,1))
```

Chosen address: Y

$$Q(prop.|curr.) = \frac{1}{3}Q(Y'|Y), \quad Q(curr.|prop.) = \frac{1}{2}Q(Y|Y')P(B),$$

Current:

```
{'X': {'value': tensor(1.5410),
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-2.1063)},
 'Y': {'value': tensor(1.2476), # changes
      'distribution': Normal(loc: 1.5410, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-0.9620)}, # changes
 'B': {'value': tensor(-2.1788), # changes
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': False, # changes
      'log_prob': tensor(-3.2925)}} # changes
```

Proposed:

```
{'X': {'value': tensor(1.5410),
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-2.1063)},
 'Y': {'value': tensor(-0.1596), # changes
      'distribution': Normal(loc: 1.5410, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-2.3650)}, # changes
 'A': {'value': tensor(1.0000), # changes
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': True, # changes
      'log_prob': tensor(-1.4189)}} # changes
```

# Metropolis Hastings Algorithm For PPLs

```
def model():
    X = sample("X", dist.Normal(0,1))
    Y = sample("Y", dist.Normal(X,1))
    if Y < 0:
        sample("A", dist.Normal(0,1), observed=torch.tensor(1.))
    else:
        sample("B", dist.Normal(0,1))
```

Chosen address: B

$$Q(prop.|curr.) = \frac{1}{3}Q(B'|B), \quad Q(curr.|prop.) = \frac{1}{3}Q(B|B'),$$

Current:

```
{'X': {'value': tensor(1.5410),
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-2.1063)},
 'Y': {'value': tensor(1.2476),
      'distribution': Normal(loc: 1.5410, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-0.9620)},
 'B': {'value': tensor(-2.1788), # changes
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-3.2925)}} # changes
```

Proposed:

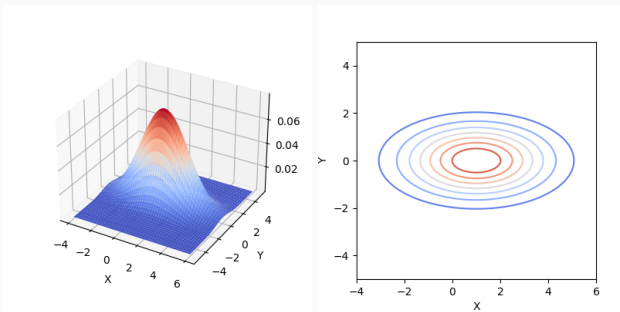
```
{'X': {'value': tensor(1.5410),
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-2.1063)},
 'Y': {'value': tensor(1.2476),
      'distribution': Normal(loc: 1.5410, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-0.9620)},
 'B': {'value': tensor(-1.2213), # changes
      'distribution': Normal(loc: 0.0, scale: 1.0),
      'is_observed': False,
      'log_prob': tensor(-1.6647)}} # changes
```

# Hamiltonian Monte Carlo

---

# Hamiltonian Monte Carlo

- Assumptions: Finite number of continuous (unconstrained) random variables
- Makes the program density essentially differentiable
- Idea: Interpret density as energy potential and samples as particles in the potential
- The improved version of HMC called NUTS is state-of-the-art inference algorithm



# Hamiltonian Monte Carlo

## Hamiltonian Mechanics:

- position vector  $x$ , momentum vector  $p$
- Hamiltonian

$$H(x, p) = \underbrace{U(x)}_{\text{potential energy}} + \underbrace{K(p)}_{\text{kinetic energy}}$$

- Mechanics
  - Particle's velocity equals the derivative of kinetic energy with respect to momentum:

$$\frac{dx_i}{dt} = \frac{\partial H}{\partial p_i} = \frac{\partial K}{\partial p_i}$$

- Force on the particle equals the negative gradient of the potential energy:

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial x_i} = -\frac{\partial U}{\partial x_i}$$

# Hamiltonian Monte Carlo

$$H(x, p) = \underbrace{U(x)}_{\text{potential energy}} + \underbrace{K(p)}_{\text{kinetic energy}}$$

$$U(x) := -\log P(x)$$

$$K(p) := \frac{p^T p}{2}$$

Joint distribution of position and momentum:

$$P(x, p) \propto \exp(-H(x, p)) = \underbrace{P(x)}_{\text{target distribution}} \underbrace{\exp\left(-\frac{p^T p}{2}\right)}_{\text{Normal distribution}}$$

# Hamiltonian Monte Carlo

## Idea:

- Sample random momentum.
- Simulate Hamiltonian mechanics.

If simulation is reversible and preserves volume, then the resulting Markov chain satisfies detailed balance and produces the correct samples.

This is true because, such a simulation makes the proposals symmetric. The acceptance probability simplifies to

$$\min \left( 1, \frac{P(x', p')}{P(x, p)} \right) = \min (1, \exp(-H(x', p') + H(x, p)))$$

A perfect simulation preserves the Hamiltonian and we would accept with probability 1!

# Hamiltonian Monte Carlo

## Leap-Frog Integration

$$p_i\left(t + \frac{\epsilon}{2}\right) = p_i(t) - \frac{\epsilon}{2} \cdot \frac{\partial U}{\partial x_i}(x(t)) \quad \text{half-step}$$

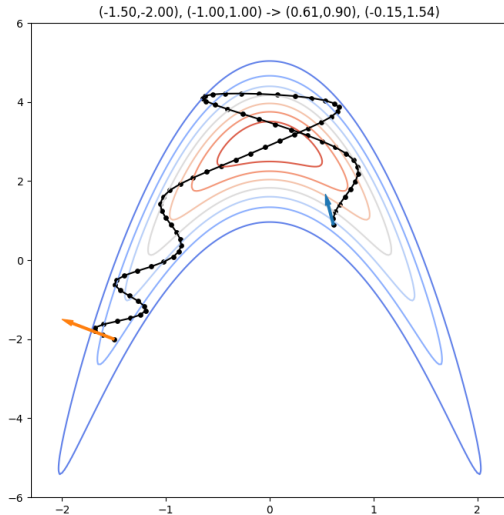
$$x_i(t + \epsilon) = x_i(t) + \epsilon \cdot p_i\left(t + \frac{\epsilon}{2}\right) \quad \text{full-step}$$

$$p_i(t + \epsilon) = p_i\left(t + \frac{\epsilon}{2}\right) - \frac{\epsilon}{2} \cdot \frac{\partial U}{\partial x_i}(x(t + \epsilon)) \quad \text{half-step}$$

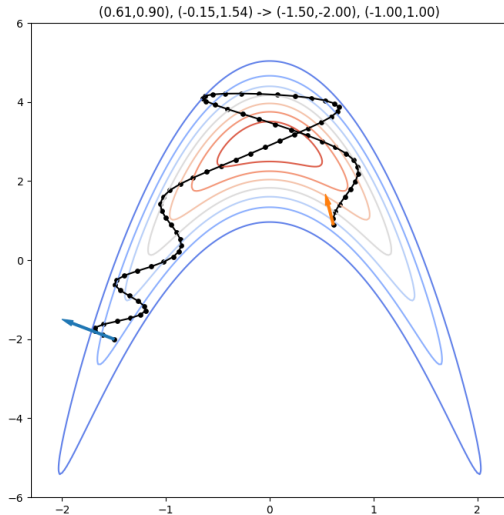
This procedure gets more accurate with  $\epsilon \rightarrow 0$ .

```
1 P = P - eps/2 * grad_U      # half-step
2
3 repeat:
4     X = X + eps * P          # full-step
5     P = P - eps * grad_U(X)  # full-step
6
7 X = X + eps * P              # full-step
8 P = P - eps/2 * grad_U(X)    # half-step
```

# Hamiltonian Monte Carlo: Leap-frog trajectory



# Hamiltonian Monte Carlo: Leap-frog trajectory



# Hamiltonian Monte Carlo

Let  $P(x)$  be the target distribution, set  $U(x) = -\log P(x)$ ,  $K(p) = p^T p/2$ .

1. Initialise  $x_0$
2. For each  $i = 1, \dots, n$ 
  - Sample the momentum,  $p \sim \text{Normal}(0, 1)$
  - Simulate mechanics

$$x', p' = \text{leapfrog}(x, p)$$

- Calculate the acceptance probability

$$A = \min(1, \exp(-H(x', p') + H(x, p)))$$

- Draw a random number  $0 \leq p \leq 1$  and let

$$x_{i+1} = \begin{cases} x' & p \leq A \quad (\text{accept}) \\ x_i & p > A \quad (\text{reject}) \end{cases}$$

## Resources

Why we use dependent sampling to sample from the posterior

[\*https://www.youtube.com/watch?v=CfpRdmddVPM\*](https://www.youtube.com/watch?v=CfpRdmddVPM)

An introduction to the Random Walk Metropolis algorithm

[\*https://www.youtube.com/watch?v=U561HGMWjcw\*](https://www.youtube.com/watch?v=U561HGMWjcw)

Paper: Single-Site MH for PPL [\*http://proceedings.mlr.press/v15/wingate11a/wingate11a.pdf\*](http://proceedings.mlr.press/v15/wingate11a/wingate11a.pdf)

Handbook of MCMC: Chapter 5: MCMC Using Hamiltonian Dynamics

[\*http://www.mcmchandbook.net/HandbookChapter5.pdf\*](http://www.mcmchandbook.net/HandbookChapter5.pdf)

The intuition behind the Hamiltonian Monte Carlo algorithm

[\*https://www.youtube.com/watch?v=a-wydhEuAm0\*](https://www.youtube.com/watch?v=a-wydhEuAm0)

MCMC Interactive Gallery

[\*https://chi-feng.github.io/mcmc-demo/app.html\*](https://chi-feng.github.io/mcmc-demo/app.html)

Paper: No-U-Turn Sampler

[\*https://arxiv.org/pdf/1111.4246.pdf\*](https://arxiv.org/pdf/1111.4246.pdf)