Hamiltonian Monte Carlo and the No-U-Turn-Sampler

Computational Statistics Project

Jonathan Fitter (jfitter@wu.ac.at) Max Heinze (mheinze@wu.ac.at)

Department of Economics, Vienna University of Economics and Business (WU Vienna)

May 27, 2025

Intro and Recap

Hamiltonian Monte Carlo

The No-U-Turn-Sampler (NUTS)

Our Implementation

Markov Chain

A **Markov chain** is a stochastic process $\{X_t\}$ indexed by time $t \geq 0$.

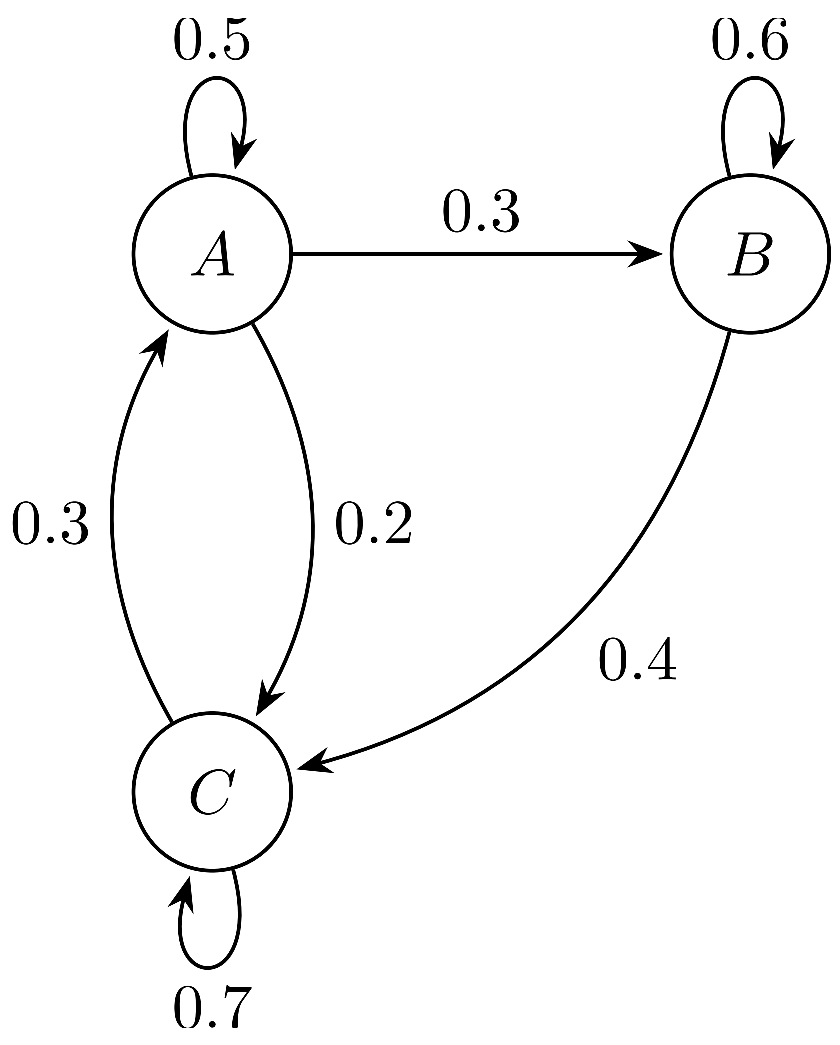
- We call the **set of all values** that $\{X_t\}$ can assume the **state space** of the Markov Chain.
- $\{X - t\}$ is called a Markov Chain if it fulfills the **Markov property** that

$$P(X_{t+1} = j \mid X_0 = i_0, \ldots, X_t = i_t) = P(X_{t+1} = j \mid X_t = i_t),$$

  i.e., that the conditional distribution of the next state depends only on the current state.

Markov Chain Monte Carlo (MCMC) methods make use of Markov chains to sample from a **target distribution**.

Markov Chain

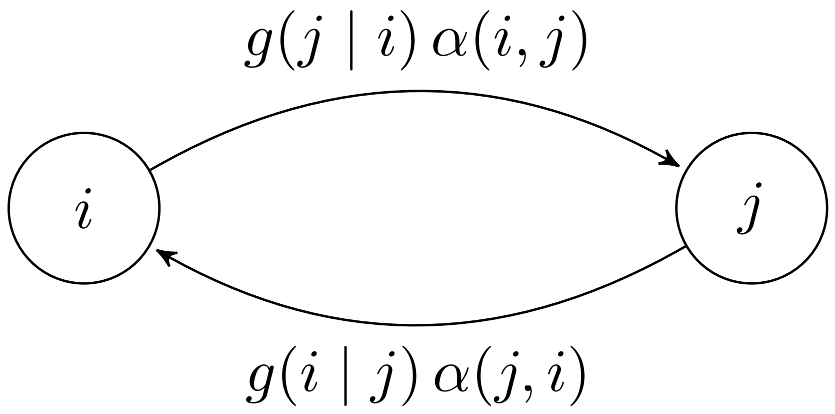


Markov chains are

- **irreducible**, i.e. every state can eventually be reached from any state;
- **aperiodic**, i.e. reverting to a state is not only possible after a multiple of a certain number of steps;
- **positive recurrent**, i.e. for all states, the expected number of transitions to return to that state is finite.

For such Markov Chains, **transition probabilities** converge to a unique stationary distribution on the state space.

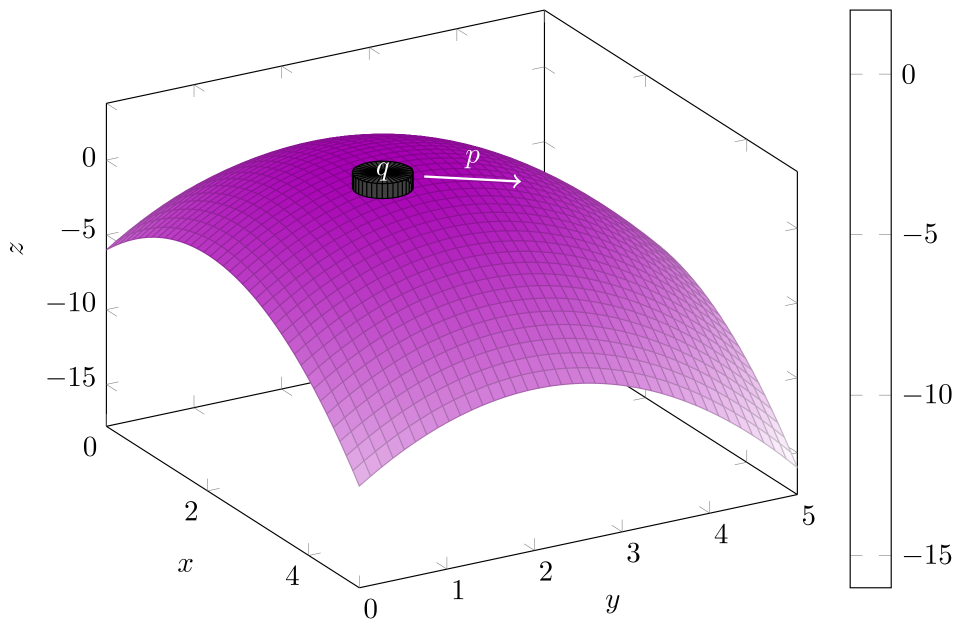
Metropolis-Hastings



Metropolis-Hastings is a class of MCMC methods. Using a Metropolis-Hastings algorithm, we follow these steps:

- We draw the initial value from some density.
- We selet a proposal density $g(\cdot \mid i)$
- We compute $\alpha(i,j) = \min\left(1, \frac{f(j)\,g(i|j)}{f(i)\,g(j|i)}\right)$.
- We draw U from $U(0,1)$.
- If $U \leq \alpha(i,j)$, we move from i to j, otherwise we stay at i.

Origins in Physics



Imagine a **frictionless puck** on a **non-even surface**. The **state of this system** is given by

- $\boldsymbol{q}$, the **position** of the puck;
- $\boldsymbol{p}$, the **momentum** of the puck, given by $\text{mass} \times \text{velocity}$.
- If the puck encounters a **rising slope**, momentum will decrease; if it encounters a **falling slope**, momentum will increase.
- The higher the surface at a position, the higher the puck's **potential energy**, and the lower its **kinetic energy**.

From Physics to Statistics

In **non-physical** applications,

- the **position** $\boldsymbol{q}$ corresponds to the **variables of interest**,
- the **potential energy** $\boldsymbol{p}$ corresponds to the **negative log of the probability density** for these variables, and
- the **momentum** is an auxiliary variable that facilitates exploration of the target distribution space.

Hamiltonian Dynamics

The system is described by a so-called **Hamiltonian equation**, $H(q_p)$. Its partial derivatives, the **equations of motion**, determine how $\boldsymbol{q}$ and $\boldsymbol{p}$ change over time:

$$\begin{aligned}\frac{\mathrm{d}q_i}{\mathrm{d}t} &= \frac{\partial H}{\partial p_i},\\ \frac{\mathrm{d}p_i}{\mathrm{d}t} &= -\frac{\partial H}{\partial q_i},\end{aligned}$$

for $i = 1, \ldots, d$, and d being the length of the vectors $\boldsymbol{q}$ and $\boldsymbol{p}$.

Potential and Kinetic Energy

For **Hamiltonian Monte Carlo**, we usually use the following kind of Hamiltonian functions:

$$H(\boldsymbol{q}, \boldsymbol{p}) = U(\boldsymbol{q}) + K(\boldsymbol{p}),$$

where the **potential energy** $U(\boldsymbol{q})$ is defined as minus the log probability density of the distribution for $\boldsymbol{q}$ that we wish to sample, plus a constant;

and the **kinetic energy** $K(\boldsymbol{q})$ is given by

$$K(\boldsymbol{p}) = \boldsymbol{p}'\boldsymbol{M}^{-1}\boldsymbol{p}/2,$$

where $\boldsymbol{M}$ is a symmetric, p.s.d., and often diagonal mass matrix.

Hamiltonian Dynamics

For **Hamiltonian Monte Carlo**, we usually use the following kind of Hamiltonian functions:

$$H(\boldsymbol{q}, \boldsymbol{p}) = U(\boldsymbol{q}) + K(\boldsymbol{p}),$$

Using this specification, the **Hamiltonian functions** can be written as:

$$\begin{aligned} \frac{\mathrm{d}q_i}{\mathrm{d}t} &= [\boldsymbol{M}^{-1}\boldsymbol{p}]_i, \\ \frac{\mathrm{d}p_i}{\mathrm{d}t} &= -\frac{\partial U}{\partial q_i}. \end{aligned}$$

Properties

Hamiltonian dynamics fulfill a set of **properties** that make them suitable for use in MCMC updating:

- **Reversibility.** The mapping from the state at time t to the state at time $t + s$ is one-to-one and hence has an inverse. This means that Markov chain transitions are reversible.
- **Conservation of the Hamiltonian.** With dynamics as specified, the Hamiltonian H itself is kept invariant. For Metropolis updates, the acceptance probability is one if H is kept invariant (only approximatively achievable).
- **Volume preservation.** Hamiltonian dynamics preserves volume in $(\boldsymbol{q}, \boldsymbol{p})$ space, meaning that we do npt need to account for a change in volume in the acceptance probability for Metropolis updates.

Reversibility and **preservation of volume** can be maintained even when the Hamiltonian is approximated.

Discretization

For implementation, we need to **discretize** the Hamiltonian equations using a small step size ε: time is then discrete with $t = 0, \varepsilon, 2\varepsilon, 3\varepsilon, \ldots$

The simplest way to approximate the solution is **Euler's method**:

$$p_i(t+\varepsilon) = p_i(t) - \varepsilon \frac{\partial U}{\partial q_i}(q(t)), \qquad q_i(t+\varepsilon) = q_i(t) + \varepsilon \frac{p_i(t)}{m_i}$$

We can obtain better results by slightly **modifying Euler's method**:

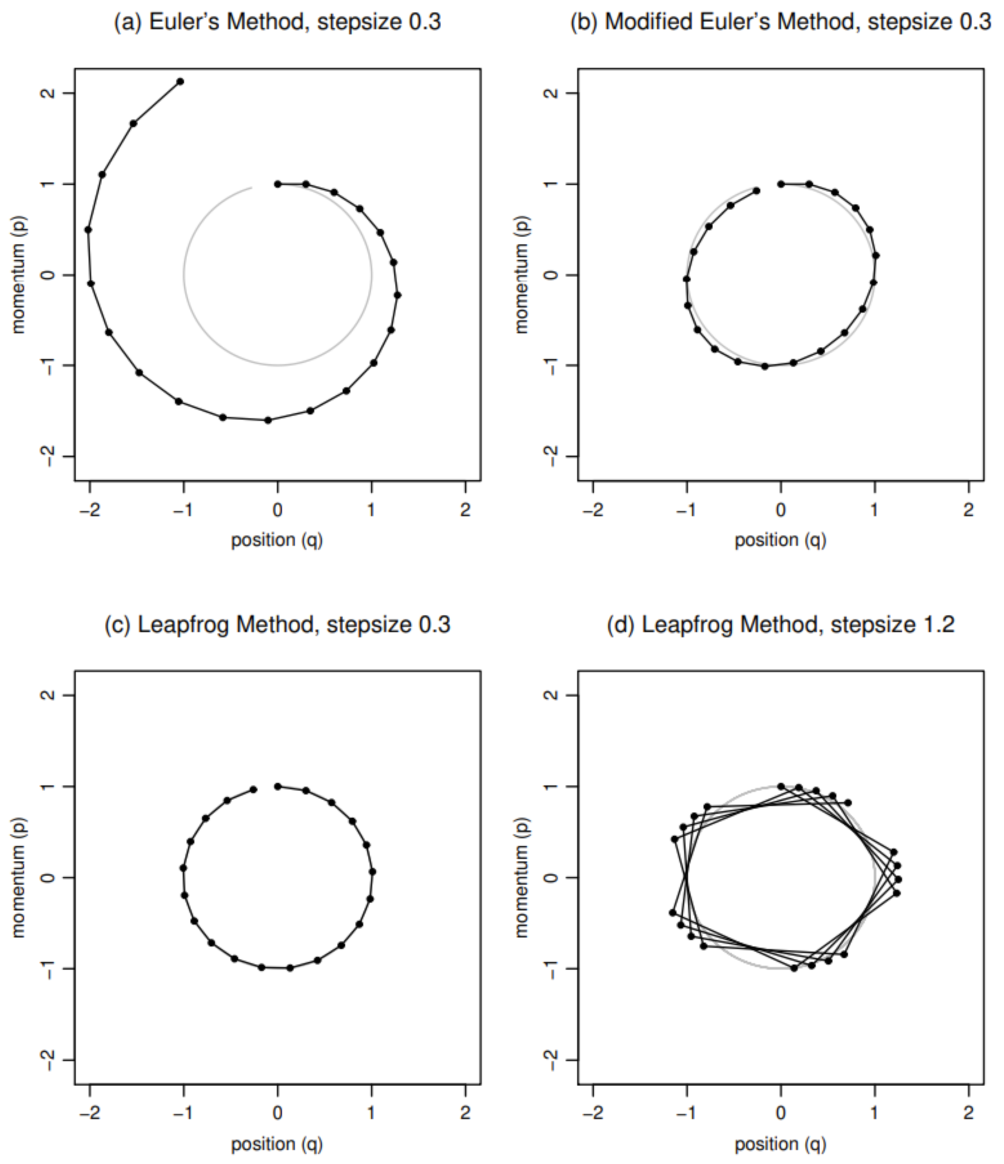
$$p_i(t+\varepsilon) = p_i(t) - \varepsilon \frac{\partial U}{\partial q_i}(q(t)), \qquad q_i(t+\varepsilon) = q_i(t) + \varepsilon \frac{p_i(t+\varepsilon)}{m_i}$$

For even better results, we can use the **Leapfrog method**:

$$p_i(t+\varepsilon/2) = p_i(t) - (\varepsilon/2) \frac{\partial U}{\partial q_i}(q(t)), \qquad q_i(t+\varepsilon) = q_i(t) + \varepsilon \frac{p_i(t+\varepsilon/2)}{m_i},$$

$$p_i(t+\varepsilon) = p_i(t+\varepsilon/2) - (\varepsilon/2) \frac{\partial U}{\partial q_i}(q(t+e))$$

Discretization



In this example, $H(q,p) = q^2/2 + p^2 + 2$. The initial state was $q = 0, p = 1$. We can see that the leapfrog method preserves volume exactly.

Hamiltonian Monte Carlo

Using **Hamiltonian dynamics** to sample from a distribution requires translating the density to a **potential energy** function and introducing **momentum** variables. We can use the concept of a canonical distribution from statistical mechanics. Given an energy function $E(x)$ for a state x, the canonical distribution over states has density $P(x) = (1/Z)\exp(-E(x)/T)$, where T is temperature and Z is a normalizing constant. Using the Hamiltonian $H(\boldsymbol{q}, \boldsymbol{p}) = U(\boldsymbol{q}) + K(\boldsymbol{p})$ as an energy function, we get

$$P(\boldsymbol{q}, \boldsymbol{p}) = \frac{1}{Z}\exp(-U(\boldsymbol{q})/T)\exp(-K(\boldsymbol{p})/T).$$

We can see that $\boldsymbol{q}$ and $\boldsymbol{p}$ are independent, and both have canonical distributions. The former will be used to represent our variables of interest, and the latter for momentum.

The Two Steps of HMC

Each **iteration** has **two steps**:

(1) In the **first step**, new values for the **momentum variables** are drawn from their Gaussian distribution.

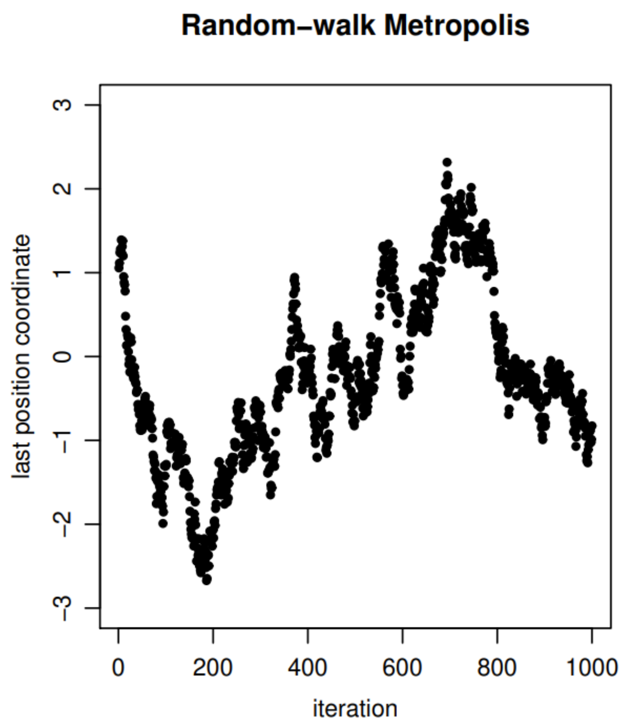
(2) In the **second step**, a **Metropolis** update is performed using Hamiltonian dynamics as explained before. We get a proposed new state $(\boldsymbol{q}^*, \boldsymbol{p}^*)$. This state is accepted as the next state of the Markov chain with probability $\min\Big(1, \quad \exp(-H(\boldsymbol{q}^*, \boldsymbol{p}^*) + H(\boldsymbol{q}, \boldsymbol{p}))\Big) \quad = \quad \min\Big(1, \quad \exp\big(-U(\boldsymbol{q}^*) + U(\boldsymbol{q}) - K(\boldsymbol{p}^*) + K(\boldsymbol{p})\big)\Big)$. If the proposed state is not accepted, the next state is set equal to the current state.

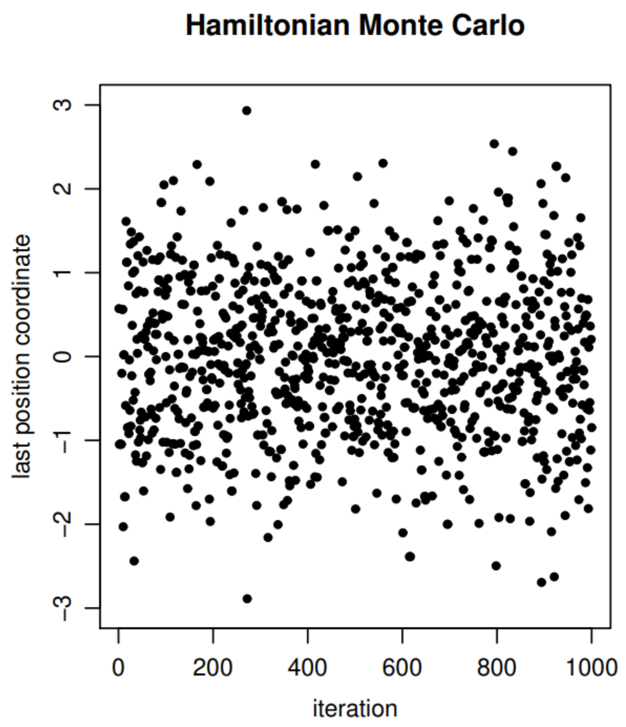
Only in the very first step of the chain does the **probability density** for $(\boldsymbol{q}, \boldsymbol{p})$ change from one step to the next. Also, the algorithm leaves the canonical distribution invariant.

Benefits of HMC

We can use HMC to sample only from **continuous** distributions on $\mathbb{R}^d$ for which the **density** function can be evaluated and the **partial derivatives of its log** can be computed.

It allows us to **sample more efficiently** from such distributions **than simpler methods** such as random-walk Metropolis.





Tuning HMC

However, in order to actually **attain the benefits** of HMC, we have to **properly tune** the **step size** ε and the **trajectory length** L, i.e., the number of leapfrog steps. Tuning an **HMC algorithm** is also more difficult than tuning a **simple Metropolis** method.

- If the **step size** ε is **too large**, acceptance rates will be low.
- If the **step size** ε is **too small**, we will waste computation time.

- If the **trajectory length** L is **too large**, trajectories may be long and still end up back at the starting point.
- If the **trajectory length** L is **too small**, it may not reach far enough into unconstrained directions. In connection with a **small** **step size** ε, it may lead to slow exploration by a random walk.

Illustration

Why NUTS?

We use **MCMC methods** usually to sample from **complicated distributions**.

- **Random-walk Metropolis** may take extremely long to converge to a target distribution.
- **Hamiltonian Monte Carlo** suppresses this random walk behavior.
  - This yields a much lower computational cost. The cost of a sample from a d-dimensional target distribution is $O(d^{5/4})$ for HMC, but $O(d^2)$ for random-walk Metropolis.
  - However, HMC requires the researcher to specify ε and L.
  - Especially setting L is costly itself, since it frequently requires multiple costly training runs.
- The **No-U-Turn-Sampler** (**NUTS**) eliminates the need to choose L, and an extension of it even allows for automatically tuning ε.

Setting L

- An L that is **too small** will yield to successive steps being very close to each other. The sampler then exhibits a form of **random walk behavior**.
- An L that is **too large** gives rise to the risk of the trajectory looping back.
- It is **difficult** to find out what the right L in a given case is. What is usually done is to run the sampler preliminarily and check autocorrelation statistics.

Eliminating the Need to Set L

The **No-U-Turn-Sampler** (**NUTS**) eliminates the need to specify L. The **basic idea** is that it retains the random-walk suppression features of HMC, but additionally evaluates a criterion telling it when it has run for ***long enough***.

- The simplest way to think of this is to conceive a metric for whether the distance from the original value θ to the proposed value $\tilde{\theta}$ has increased:

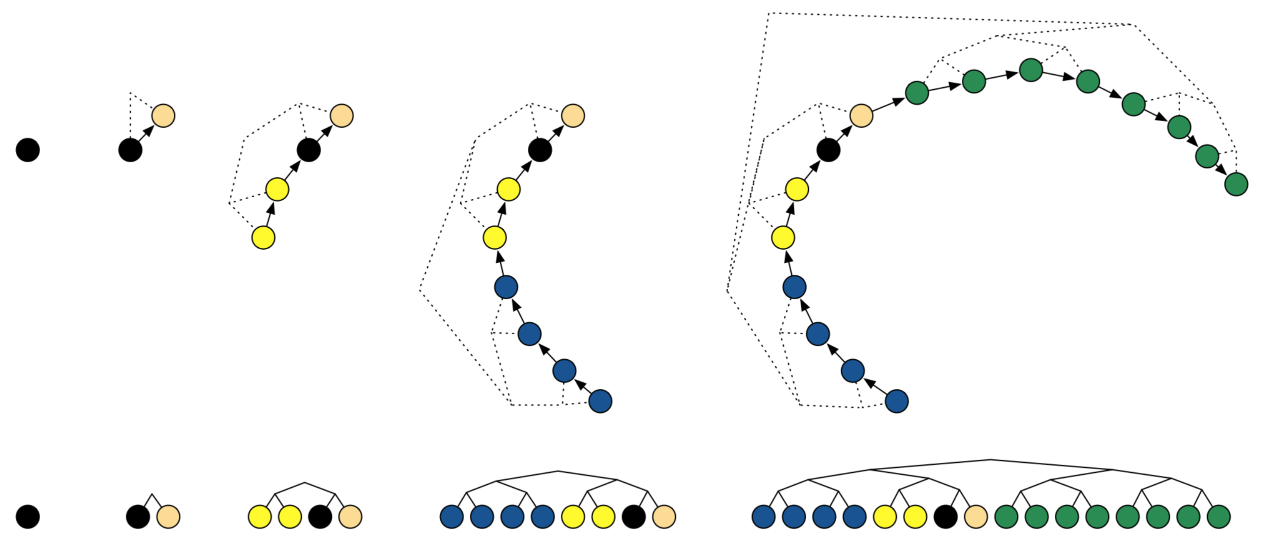
$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{\mathrm{d}}{\mathrm{d}t}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot\tilde{r},$$

  where $\tilde{r}$ is the current momentum.

- However, an algorithm that lets the leapfrog steps run until the above expression becomes negative does not guarantee **time reversibility**.

Eliminating the Need to Set L

- **NUTS** overcomes the issue by simulating both forward and backward in time, thus building a balanced binary tree via repeated doubling.
- The process is halted when the tree would **double back on itself**, i.e., the proposal would make an **U-turn**.



Example of building a binary tree via repeated doubling. Each doubling proceeds by choosing a direction (forwards or backwards in time) uniformly at random, then simulating Hamiltonian dynamics for 2^j leapfrog steps in that direction, where j is the number of previous doublings. The figures at top show a trajectory in two dimensions as it evolves over four doublings, and the figures below show the evolution of the binary tree. In this example, the directions chosen were forward (light orange node), backward (yellow nodes), backward (blue nodes), and forward (green nodes).

Sampling with NUTS

Roughly speaking, we follow these steps:

(1) Sample momentum $r \sim \mathcal{N}(0, I)$.

(2) Draw the slice variable $u \sim \mathrm{Uniform}([0, \exp(\mathcal{L}(\theta^t) - \frac{1}{2} r \cdot r)$.

(3) Sample $\mathcal{B}$ and $\mathcal{C}$ from their conditional distribution $p(\mathcal{B}, \mathcal{C} \mid \theta^t, r, u, \varepsilon)$; where $\mathcal{B}$ collects every state visited by leapfrog integration during an iteration, and $\mathcal{C} \subseteq \mathcal{B}$ are the states eligible as the next state.

(4) Sample $\theta^{t+1}, r \sim T(\theta^t, r, \mathcal{C})$, where $T(\theta', r' \mid \theta, r, \mathcal{C})$ is a transition kernel that leaves the uniform distribution over $\mathcal{C}$ invariant.

When to Stop Sampling with NUTS

We then stop expanding the tree when

(1) The error in the simulation becomes extremely large, i.e., $\mathcal{L}(\theta) - \frac{1}{2} r \cdot r - \log(u) < -\Delta_{\max}$; or

(2) one end of the simulated trajectory makes an U-turn, i.e., continuing with the simulation in any direction would make the left- and rightmost nodes of any subtree, θ^{-} and θ^{+}, move closer together.

Adaptively Tuning ε

In addition to letting NUTS choose L, we can automatically determine ε.

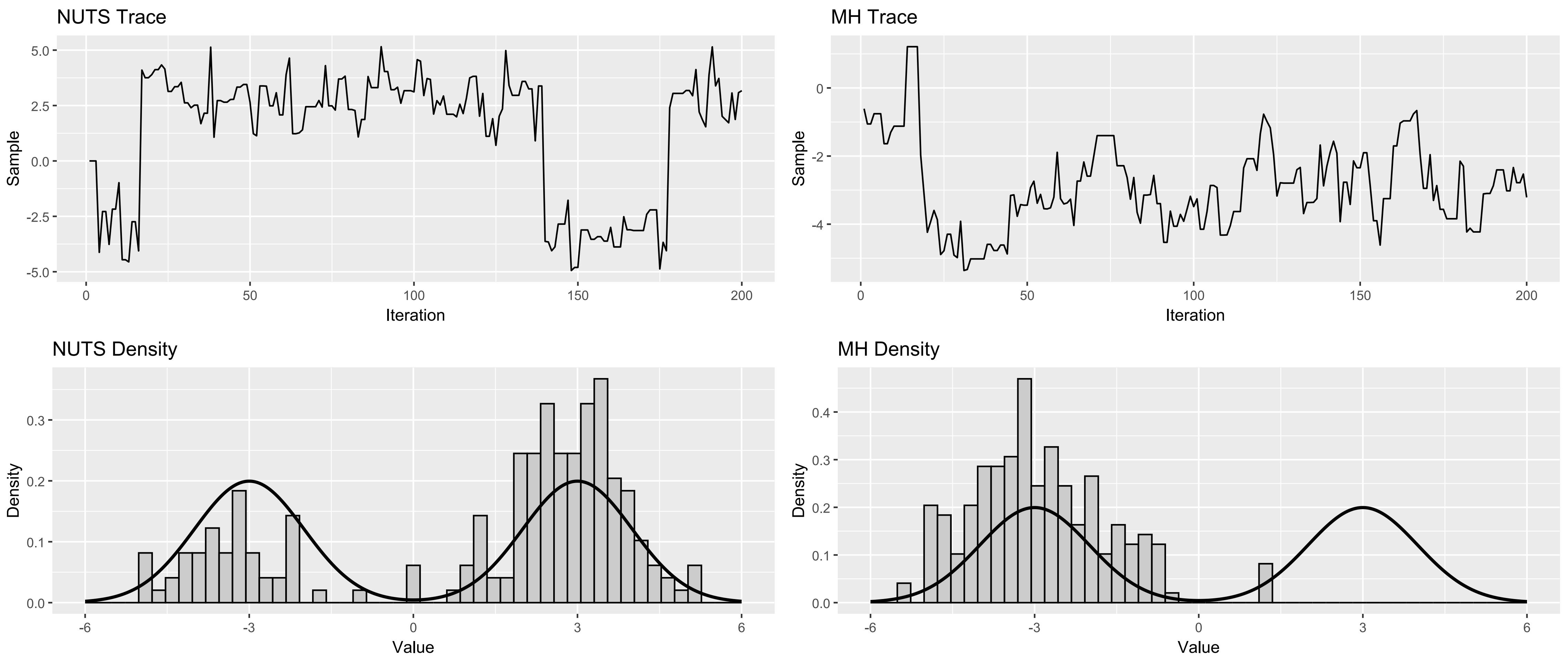
- Using **Robbins-Monro**:
  - Use a statistic H_t that describes some aspect of the behavior of an MCMC algorithm at $t \geq 1$, such as the Metropolis acceptance probability.
  - We compare that statistic to a target and adjust the step size by a function of that target, letting adjustments decay with increasing t.
- Using **Dual averaging**:
  - Robbins-Monro gives large weight to early iterations. However, we would want parameters to adapt quickly as the sampler later moves to the stationary regime.
  - Dual averaging techniques decay more slowly and keep a running average of past errors, giving less influence to very early iterations.

Illustration

Our Implementation

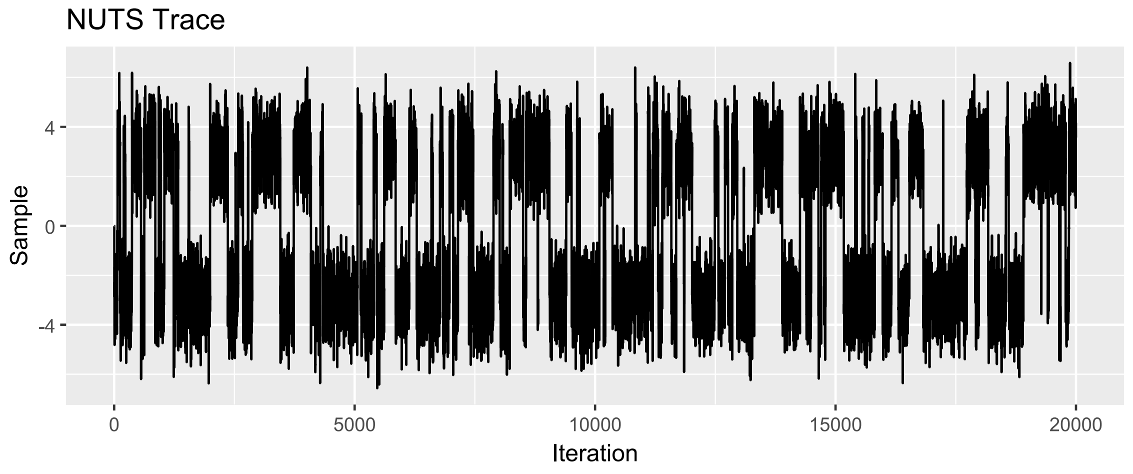
Two Peak 1D Density

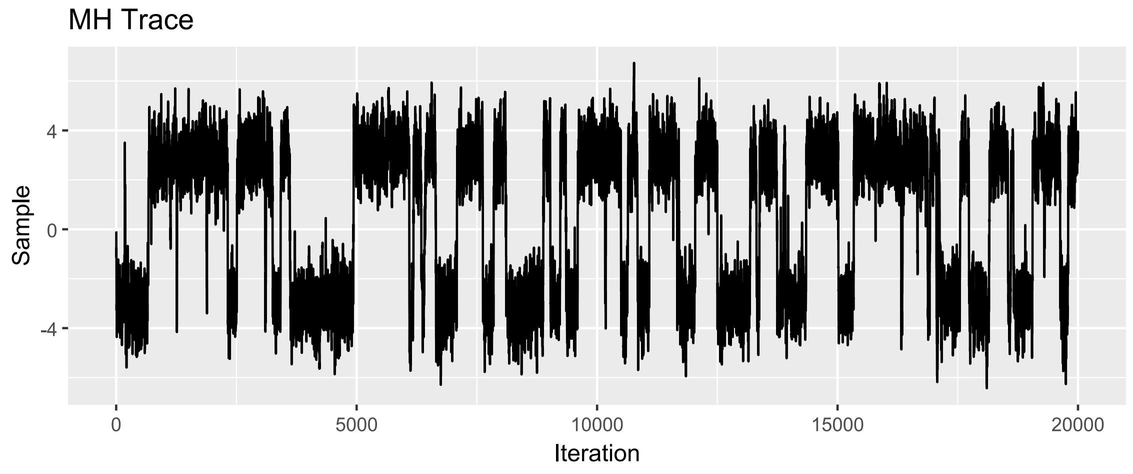
- Comparison of No-U-Turn Sampler and Metropolis-Hastings for one-dimensional mixture of Gaussians with peaks at -3 & 3:

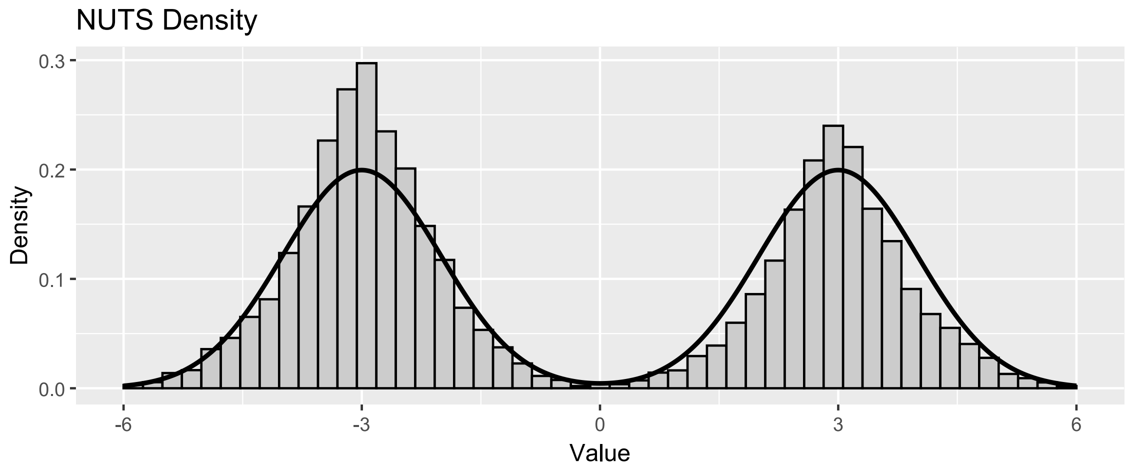


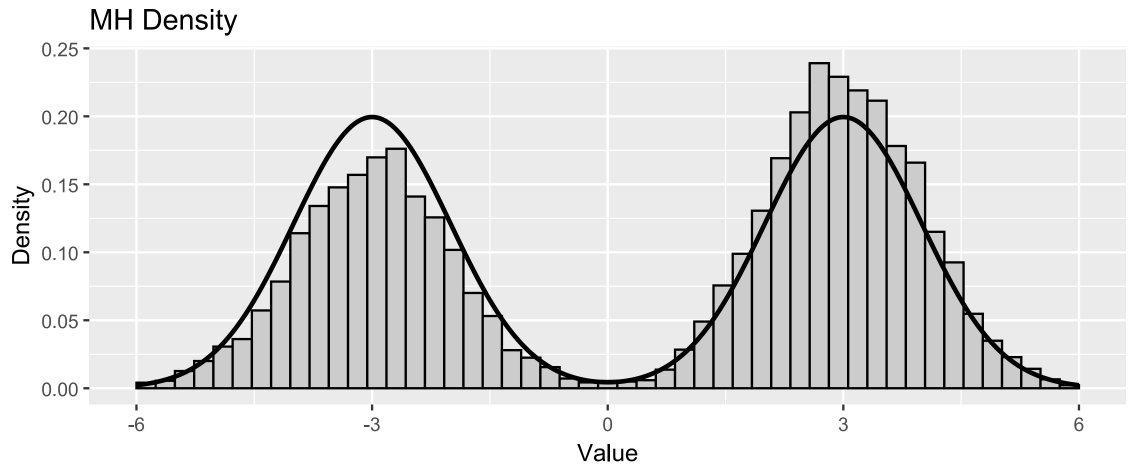
Two Peak 1D Density

- However, after 20,000 iterations, even though NUTS obtains a sample with less autocorrelation, the quality of approximation is similar to MH.







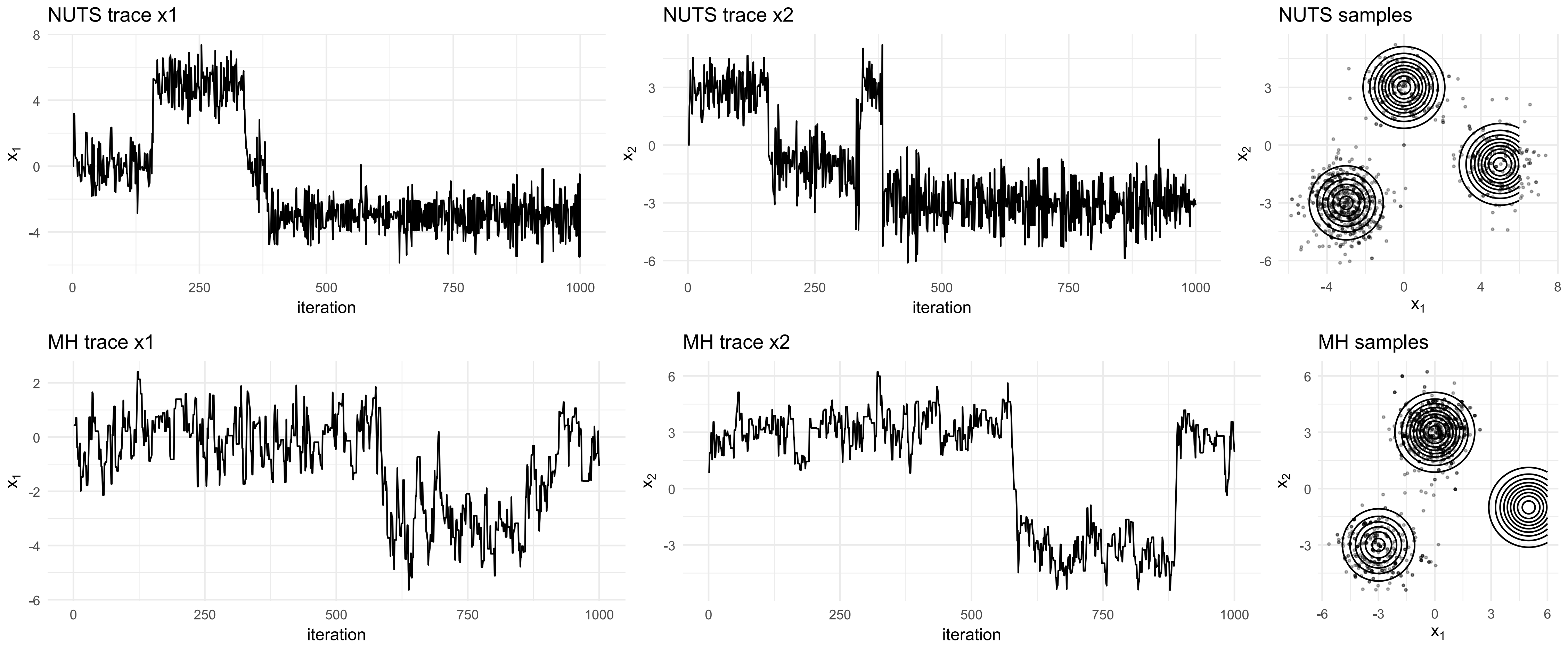


Three Peak 2D Density

- Real strength of NUTS shows for multidimensional distributions.
- Already after 100 iterations, NUTS has visited all three peaks.

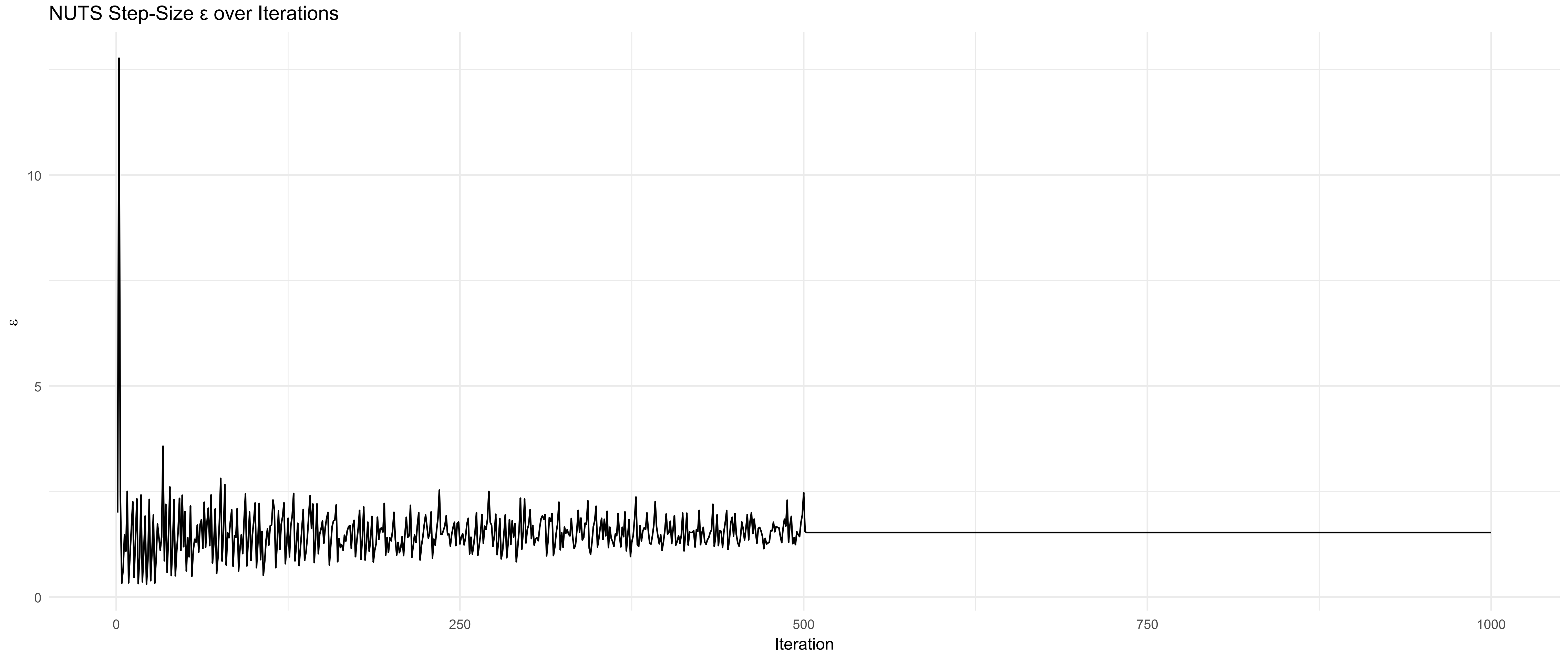
Three Peak 2D Density

- NUTS particularly good at exploring concentrations that are more distant to other peaks

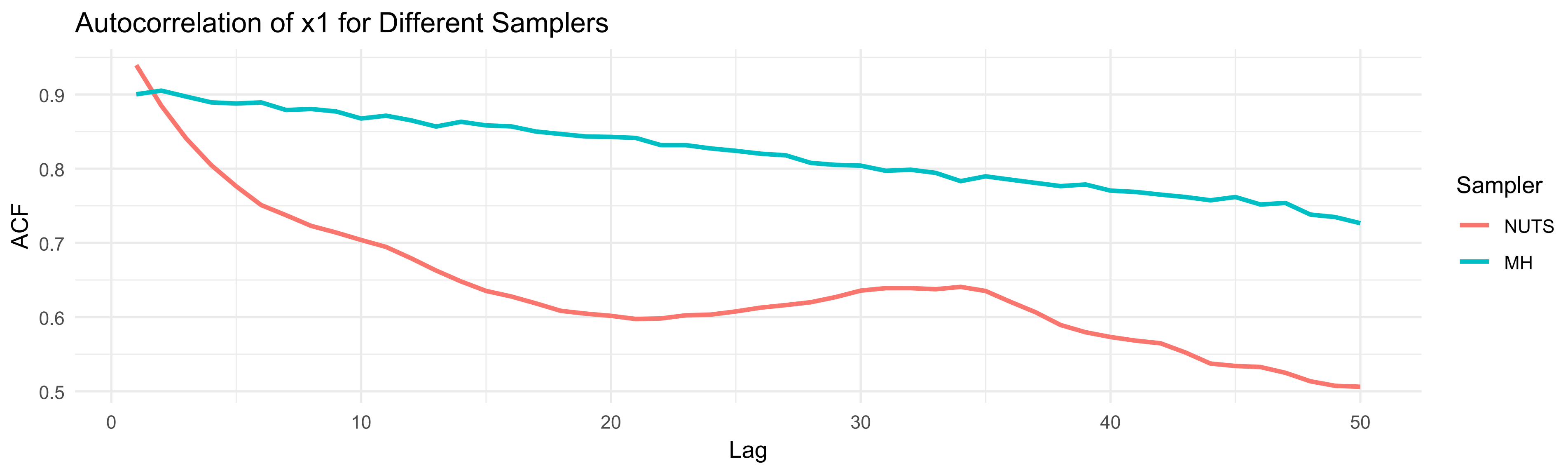


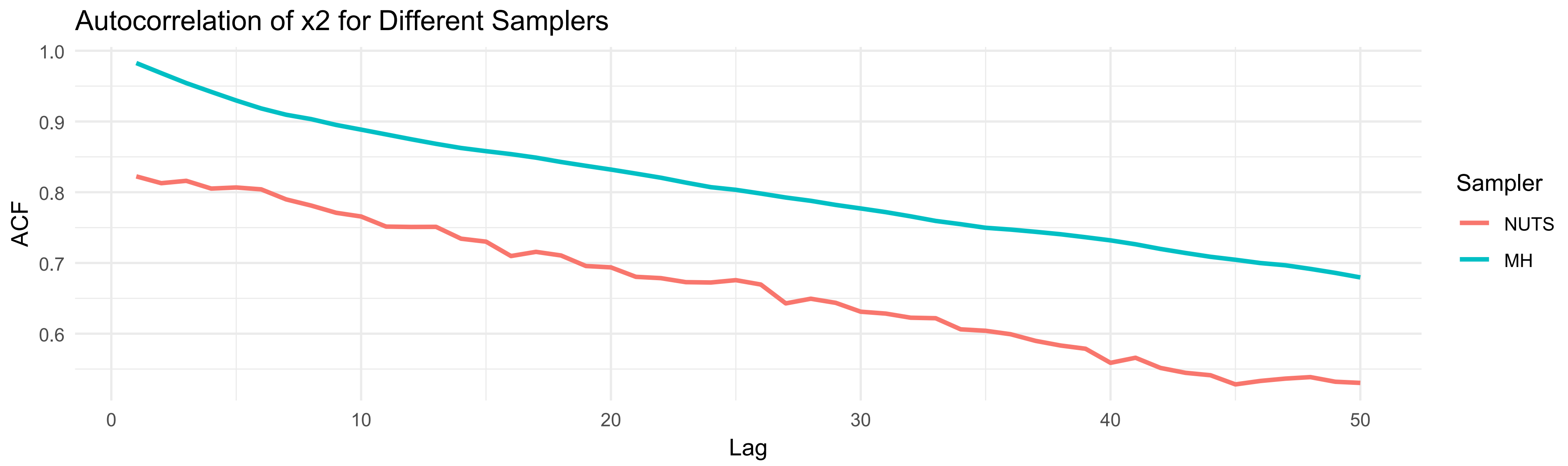
Three Peak 2D Density

- Step size adpatively set in Dual-Averaging procedure.
- Standard setting: fix epsilon after first half of iterations.



Three Peak 2D Density





References

Feng, C. (2017). *The markov-chain monte carlo interactive gallery* (Version c4723db). https://github.com/chi-feng/mcmc-demo.

Hoffman, M. D., Gelman, A., et al. (2014). The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, *15*(1), 1593–1623.

Neal, R. M. (2012). *MCMC using hamiltonian dynamics*. https://doi.org/10.48550/ARXIV.1206.1901

Hamiltonian Monte Carlo

The No-U-Turn-Sampler (NUTS)

Our Implementation

Appendix

NUTS algorithm in R

```r
# No-U-Turn sampler
leapfrog <- function(theta, r, grad_log_prob, eps) {
  r_half   <- r + 0.5 * eps * grad_log_prob(theta)
  theta_new<- theta + eps * r_half
  r_new    <- r_half + 0.5 * eps * grad_log_prob(theta_new)
  list(theta = theta_new, r = r_new)
}

find_reasonable_epsilon <- function(theta, grad_log_prob, log_prob) {
  eps <- 1
  r   <- rnorm(length(theta))
  lp0 <- log_prob(theta) - 0.5 * sum(r^2)
  lf  <- leapfrog(theta, r, grad_log_prob, eps)
  lp1 <- log_prob(lf$theta) - 0.5 * sum(lf$r^2)
  a   <- exp(lp1 - lp0)
  dir <- if (a > 0.5) 1 else -1
  while (a^dir > 2^(-dir)) {
    eps <- eps * 2^dir
    lf  <- leapfrog(theta, r, grad_log_prob, eps)
    lp1 <- log_prob(lf$theta) - 0.5 * sum(lf$r^2)
```