The NO-U-Turn Sampler (NUTS)

Presented by Per Joachims on 19.12.2019

Research Seminar in Statistics: Bayesian computation: state of the art and recent developments

Humboldt University of Berlin

Introduction

- *Goal*: do bayesian inference based on posterior distribution
- *Problem*: posterior distribution often only known up to a normalizing constant:

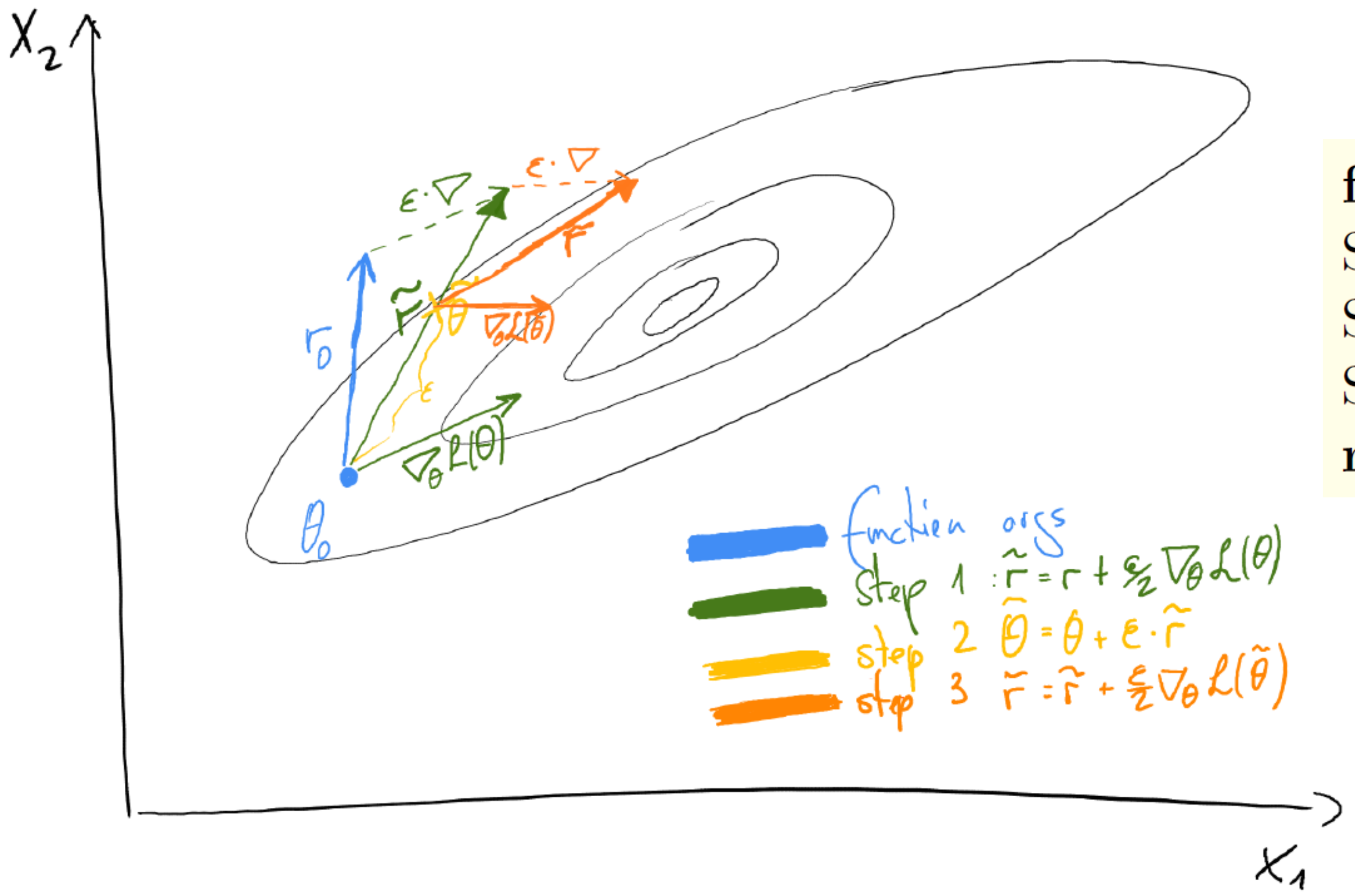
$$p(\theta|x) \propto p(x|\theta)p(\theta)$$

- *Solution*: approximate posterior by **sampling** with Markov Chain Monte Carlo (MCMC)

NUTS and HMC

- The NO-U-Turn Sampler (**NUTS**) by M.D. Hoffman and A. Gelman (2011) is an extension of Hamilton Monte Carlo (**HMC**), which is a MCMC method
- HMC generates samples by setting up and simulating Hamilton dynamics
- With **r** = momentum vector, do repeatedly:
  1. Sample r from multivariate normal
  2. Evolve θ, r by simulating L steps of the dynamics of the system
  3. Accept or reject sample (similar to Metropolis)

HMC: The Leapfrog Update



function Leapfrog(θ, r, ϵ)
Set $\tilde{r} \leftarrow r + (\epsilon/2)\nabla_\theta \mathcal{L}(\theta)$.
Set $\tilde{\theta} \leftarrow \theta + \epsilon \tilde{r}$.
Set $\tilde{r} \leftarrow \tilde{r} + (\epsilon/2)\nabla_\theta \mathcal{L}(\tilde{\theta})$.
return $\tilde{\theta}, \tilde{r}$.

HMC: Pro / Con

- *PRO*:
  - **no** random walk behaviour which makes it (1) more **efficient** and (2) allows for better **dimension-scaling** compared to Metropolis Hastings and Gibbs Samling
- *CON:*
  - HMC needs to compute **gradients** -> not possible for discrecte variables
  - Step size ε and the number of steps L must be **tuned** well

HMC vs. NUTS

Modified version of: https://twitter.com/sebvettelnews/status/589693881509253160

NUTS in a Nutshell

- Eliminates the need to hand-tune L (and ε) and makes it **available** to more people
- Efficiency (NUTS) larger or equal the efficiency of (a well tuned) HMC
- *Core Idea:* stop when the trajectory (path of hamilton states) starts to **turn back**
- *Why?* we do not want to “ruin“ our progress of exploring the state space made so far

Source: http://www.fivestaryork.com/wp-content/uploads/2014/04/Nuts01.jpg

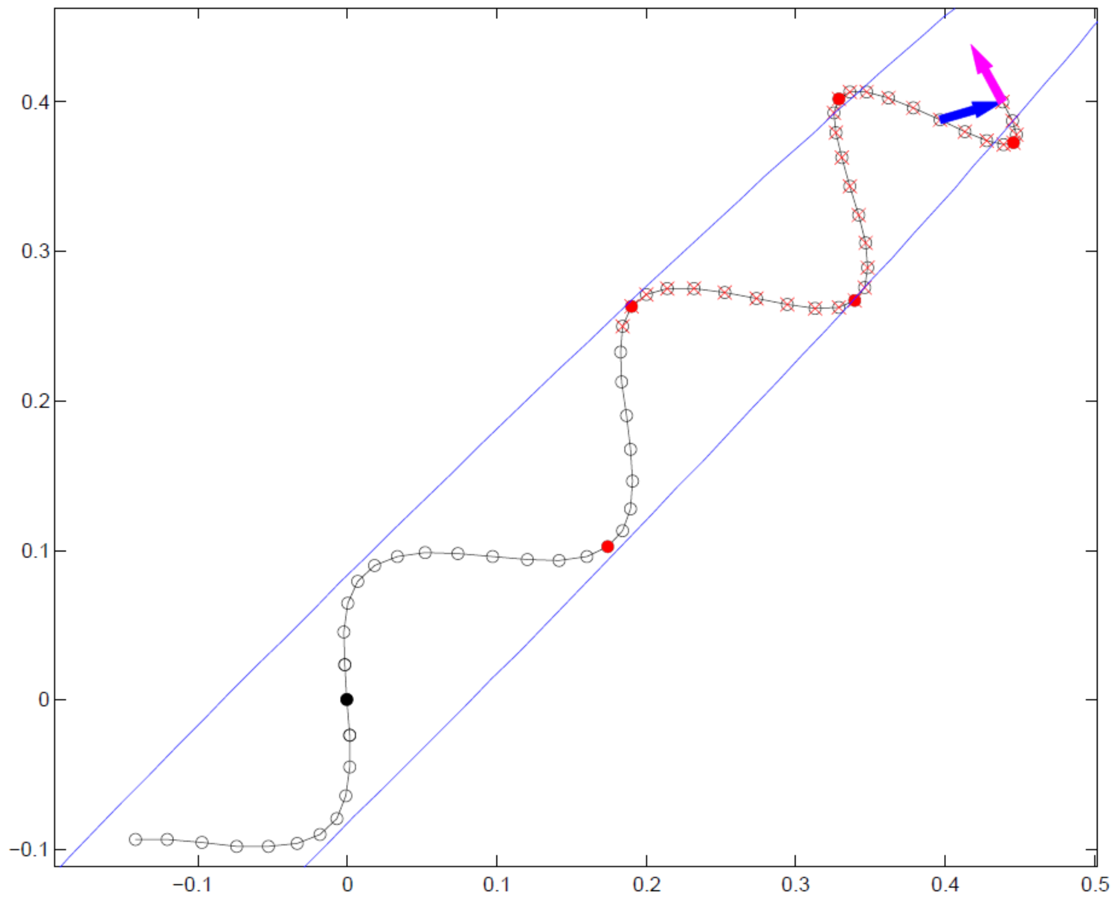
Definition of a U-Turn

- *Idea*: stop the sampling iteration if the trajectory starts to turn back
- *Measure*: **dot product!**

$$\frac{d}{dt}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{d}{dt}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot\tilde{r}$$

- Sample backwards and forwards in time to fullfill reversibility condition

A Projectory Example



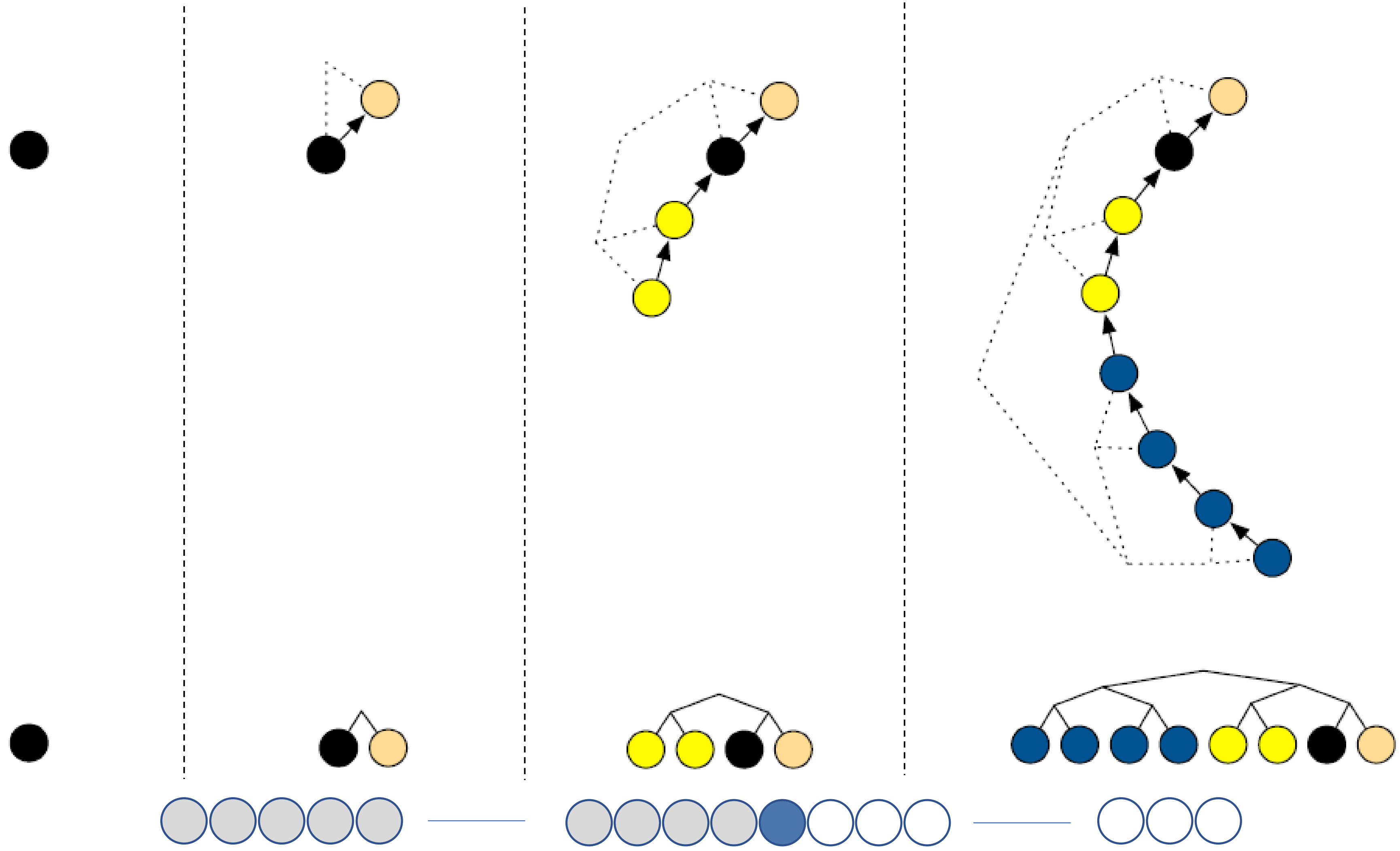
Legend

- Positions on trajectory
- Starting position
- Excluded positions because of slicing
- Excluded positions to satisfy detailed balance
- Momentum vector
- Vector between states of subtree

The Naive NUTS (High Level)

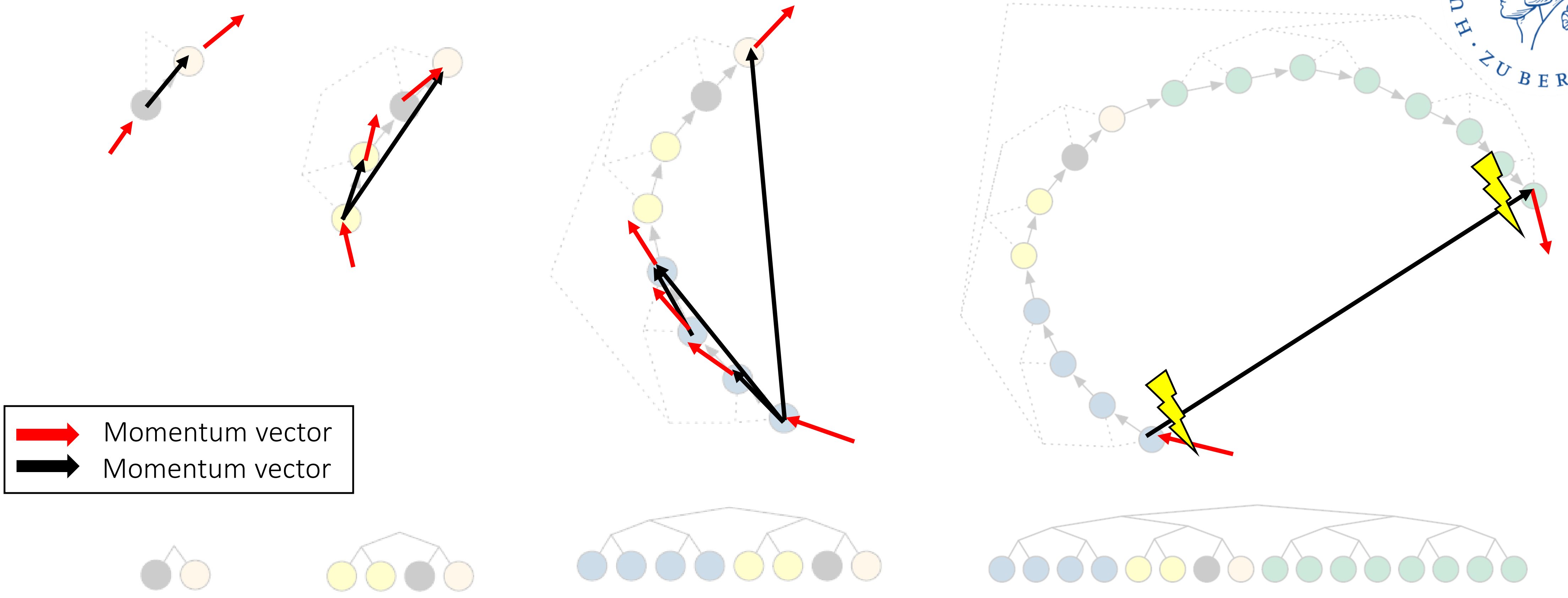
- Sample **iteration**:
  1. Sample momentum vector r
  2. Sample slicing variable u
  3. **Build trajectory:** trace out dynamics of θ, r forwards and backwards in time until stopping criteria (u-turn or/and low joint probability) is matched
  4. **Sample** uniformly from the points on trajectory

The Doubling Process



NUTS

A NUTS Iteration: Applying the Stopping Criteria



U-Turn

$$(\theta^{+}-\theta^{-})\cdot r^{-}<0 \quad \text{or} \quad (\theta^{+}-\theta^{-})\cdot r^{+}<0$$

low probabilty

$$\mathcal{L}(\theta)-\frac{1}{2}r\cdot r-\log u<-\Delta_{\max}$$

NUTS

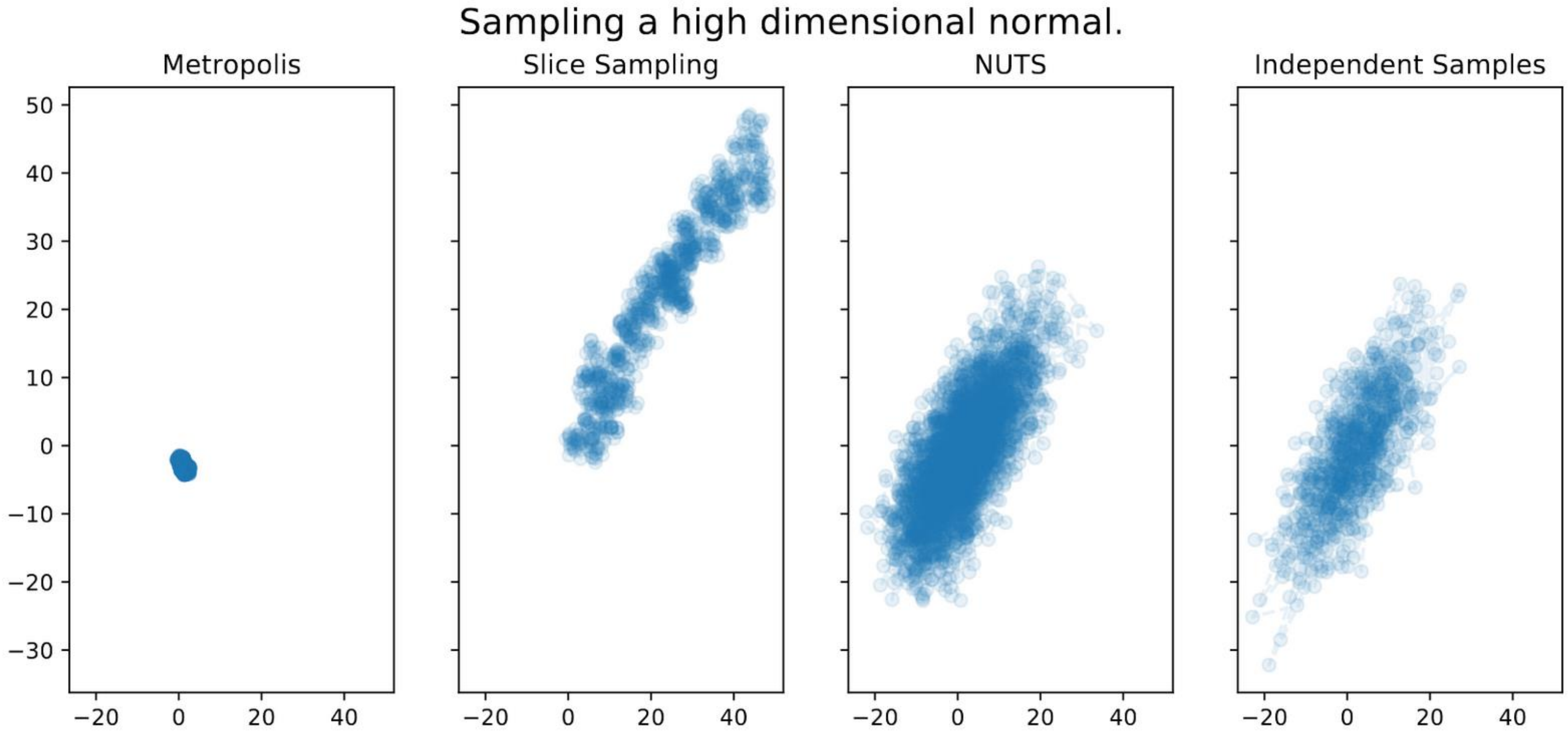
A NUTS Iteration

https://chi-feng.github.io/mcmc-demo/app.html

The Efficient NUTS

- (1) Stop building the trajectory if **one** leaf of the tree matches low-probability-criterion -> save *computation*
- (2) Try to sample at the end of the trajectory -> more *efficient* exploration of the state space
- (3) **sample θ, r while building** the tree to reduce the size of *memory* used: $O(j) < O(2^j)$
- (4) Set step size ε with dual averaging

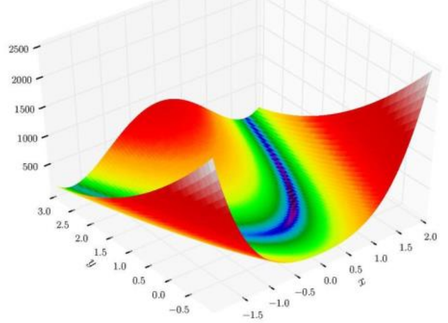
NUTS: Sampling a mvn with D=50



NUTS: Perks and Limitations

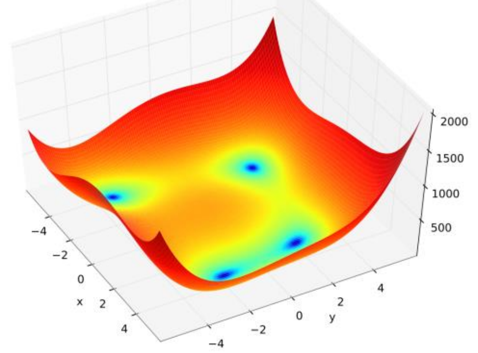
Rosenbrock's banana function

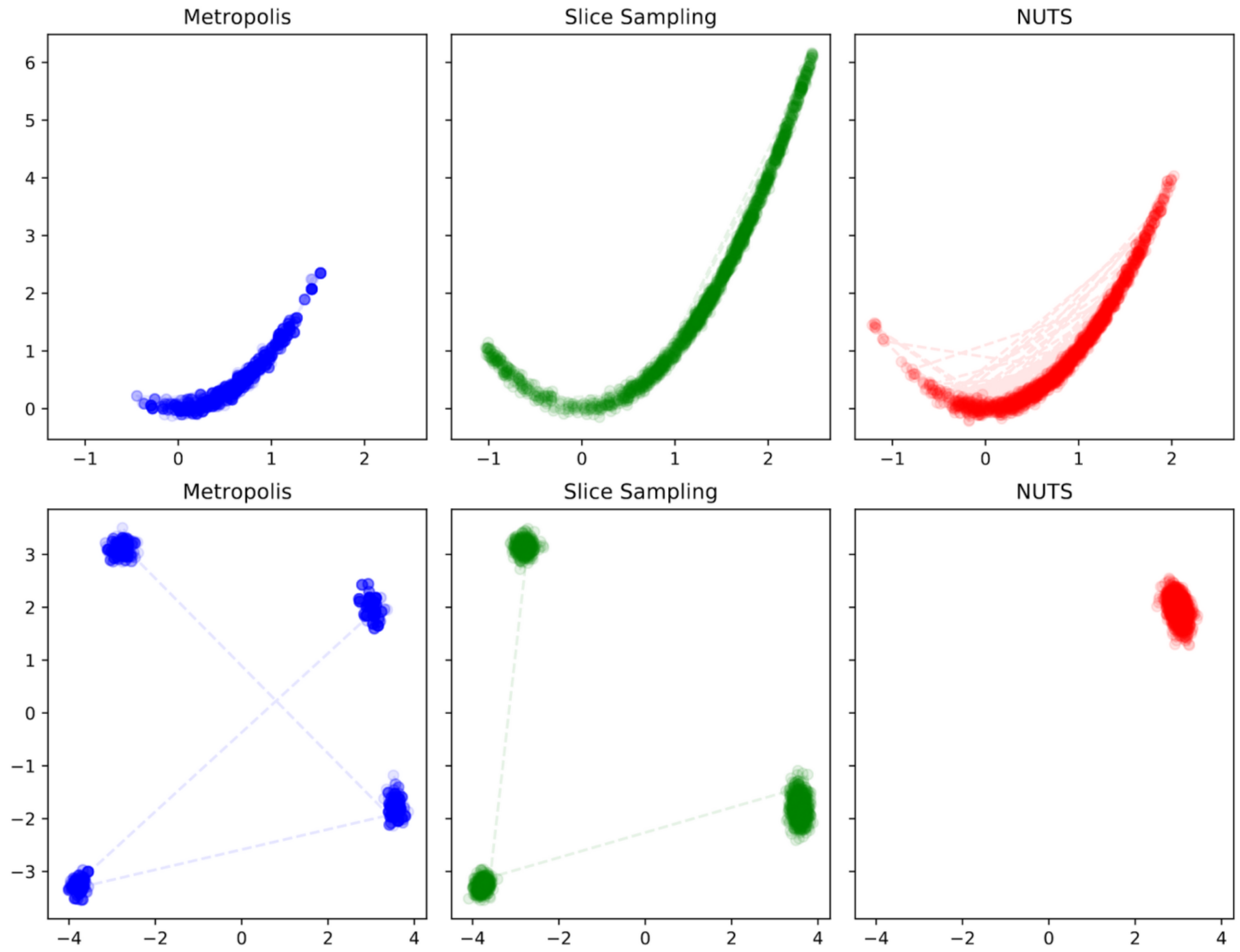
$$f(x,y) = (a-x)^2 + b(y-x^2)^2$$



Himmelblau's function

$$f(x,y) = (x^2+y-11)^2 + (x+y^2-7)^2.$$





NUTS: a Summary

- *PRO*:
  - no random walk behaviour which makes it (1) more **efficient** and (2) allows for better **dimension-scaling** compared to Metropolis Hastings and Gibbs Samling
  - no need for handtuning
- *CON:*
  - as in HMC, NUTS needs to compute **gradients** -> not possible for discrecte variables

➡ NUTS is the default sampling method in STAN / pymc3

THANKS !

See this + code https://github.com/pjoachims/rssbayes

References / Further Readings

[1] Hoffman, M.D. and Gelman, A., 2014. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, *15*(1), pp.1593-1623.

[2] Betancourt, M., 2017. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*.

[3] Neal, R.M., 1994. An improved acceptance procedure for the hybrid Monte Carlo algorithm. *Journal of Computational Physics*, *111*(1), pp.194-203.

[4] https://chi-feng.github.io/mcmc-demo/

[5] https://docs.pymc.io/

[6] https://en.wikipedia.org/wiki/Test_functions_for_optimization

Appendix: HMC

Algorithm 1 Hamiltonian Monte Carlo

Given θ^0, ϵ, L, $\mathcal{L}$, M:
for $m = 1$ to M **do**
  Sample $r^0 \sim \mathcal{N}(0, I)$.
  Set $\theta^m \leftarrow \theta^{m-1}, \tilde{\theta} \leftarrow \theta^{m-1}, \tilde{r} \leftarrow r^0$.
  for $i = 1$ to L **do**
    Set $\tilde{\theta}, \tilde{r} \leftarrow \text{Leapfrog}(\tilde{\theta}, \tilde{r}, \epsilon)$.
  end for
  With probability $\alpha = \min\left\{1, \frac{\exp\{\mathcal{L}(\tilde{\theta}) - \frac{1}{2}\tilde{r}\cdot\tilde{r}\}}{\exp\{\mathcal{L}(\theta^{m-1}) - \frac{1}{2}r^0 \cdot r^0\}}\right\}$, set $\theta^m \leftarrow \tilde{\theta}$, $r^m \leftarrow -\tilde{r}$.
end for

function Leapfrog(θ, r, ϵ)
Set $\tilde{r} \leftarrow r + (\epsilon/2)\nabla_\theta \mathcal{L}(\theta)$.
Set $\tilde{\theta} \leftarrow \theta + \epsilon\tilde{r}$.
Set $\tilde{r} \leftarrow \tilde{r} + (\epsilon/2)\nabla_\theta \mathcal{L}(\tilde{\theta})$.
return $\tilde{\theta}, \tilde{r}$.

Appendix: Algorithm -> Naive

Algorithm 2 Naive No-U-Turn Sampler

```
Given θ^0, ε, L, M:
for m = 1 to M do
  Resample r^0 ~ N(0, I).
  Resample u ~ Uniform([0, exp{L(θ^{m-1} − ½ r^0 · r^0}])
  Initialize θ^- = θ^{m-1}, θ^+ = θ^{m-1}, r^- = r^0, r^+ = r^0, j = 0, C = {(θ^{m-1}, r^0)}, s = 1.
  while s = 1 do
    Choose a direction v_j ~ Uniform({−1, 1}).
    if v_j = −1 then
      θ^-, r^-, −, −, C', s' ← BuildTree(θ^-, r^-, u, v_j, j, ε).
    else
      −, −, θ^+, r^+, C', s' ← BuildTree(θ^+, r^+, u, v_j, j, ε).
    end if
    if s' = 1 then
      C ← C ∪ C'.
    end if
    s ← s' I[(θ^+ − θ^-) · r^- ≥ 0] I[(θ^+ − θ^-) · r^+ ≥ 0].
    j ← j + 1.
  end while
  Sample θ^m, r uniformly at random from C.
end for
```

```
function BuildTree(θ, r, u, v, j, ε)
if j = 0 then
  Base case—take one leapfrog step in the direction v.
  θ', r' ← Leapfrog(θ, r, vε).
  C' ← { {(θ', r')}  if u ≤ exp{L(θ') − ½ r' · r'}
       { ∅           else
  s' ← I[L(θ') − ½ r' · r' > log u − Δ_max].
  return θ', r', θ', r', C', s'.
else
  Recursion—build the left and right subtrees.
  θ^-, r^-, θ^+, r^+, C', s' ← BuildTree(θ, r, u, v, j − 1, ε).
  if v = −1 then
    θ^-, r^-, −, −, C'', s'' ← BuildTree(θ^-, r^-, u, v, j − 1, ε).
  else
    −, −, θ^+, r^+, C'', s'' ← BuildTree(θ^+, r^+, u, v, j − 1, ε).
  end if
  s' ← s' s'' I[(θ^+ − θ^-) · r^- ≥ 0] I[(θ^+ − θ^-) · r^+ ≥ 0].
  C' ← C' ∪ C''.
  return θ^-, r^-, θ^+, r^+, C', s'.
end if
```

Appendix: Algorithm -> Efficient

Algorithm 3 Efficient No-U-Turn Sampler

Given θ^0, ϵ, $\mathcal{L}$, M:
for $m = 1$ to M **do**
Resample $r^0 \sim \mathcal{N}(0, I)$.
Resample $u \sim \text{Uniform}([0, \exp\{\mathcal{L}(\theta^{m-1} - \frac{1}{2} r^0 \cdot r^0\}])$
Initialize $\theta^- = \theta^{m-1}$, $\theta^+ = \theta^{m-1}$, $r^- = r^0, r^+ = r^0, j = 0, \theta^m = \theta^{m-1}, n = 1, s = 1$.
while $s = 1$ **do**
Choose a direction $v_j \sim \text{Uniform}(\{-1, 1\})$.
if $v_j = -1$ **then**
$\theta^-, r^-, -, -, \theta', n', s' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v_j, j, \epsilon)$.
else
$-, -, \theta^+, r^+, \theta', n', s' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v_j, j, \epsilon)$.
end if
if $s' = 1$ **then**
With probability $\min\{1, \frac{n'}{n}\}$, set $\theta^m \leftarrow \theta'$.
end if
$n \leftarrow n + n'$.
$s \leftarrow s' \mathbb{I}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{I}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$.
$j \leftarrow j + 1$.
end while
end for

function BuildTree$(\theta, r, u, v, j, \epsilon)$
if $j = 0$ **then**
Base case—take one leapfrog step in the direction v.
$\theta', r' \leftarrow \text{Leapfrog}(\theta, r, v\epsilon)$.
$n' \leftarrow \mathbb{I}[u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r'\}]$.
$s' \leftarrow \mathbb{I}[\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r' > \log u - \Delta_{\max}]$
return $\theta', r', \theta', r', \theta', n', s'$.
else
Recursion—implicitly build the left and right subtrees.
$\theta^-, r^-, \theta^+, r^+, \theta', n', s' \leftarrow \text{BuildTree}(\theta, r, u, v, j-1, \epsilon)$.
if $s' = 1$ **then**
if $v = -1$ **then**
$\theta^-, r^-, -, -, \theta'', n'', s'' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v, j-1, \epsilon)$.
else
$-, -, \theta^+, r^+, \theta'', n'', s'' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v, j-1, \epsilon)$.
end if
With probability $\frac{n''}{n' + n''}$, set $\theta' \leftarrow \theta''$.
$s' \leftarrow s'' \mathbb{I}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{I}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
$n' \leftarrow n' + n''$
end if
return $\theta^-, r^-, \theta^+, r^+, \theta', n', s'$.
end if

Appendix: Satisfying Detailed-Balance $p(\mathcal{B}, \mathcal{C} | \theta, r, u, \epsilon)$

C.1: All elements of $\mathcal{C}$ must be chosen in a way that preserves volume. That is, any deterministic transformations of θ, r used to add a state θ', r' to $\mathcal{C}$ must have a Jacobian with unit determinant.

Why?: threat unnormalized prob. density of element as unconditional probaility mass

How?: leapfrog step are volume preserving

C.2: $p((\theta, r) \in \mathcal{C} | \theta, r, u, \epsilon) = 1.$

Why?: ensure current state is a valid sample

How?: satisfied if inital state is in proposal states

C.3: $p(u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r'\} | (\theta', r') \in \mathcal{C}) = 1.$

Why?: any state in C must be in the slice defined by u -> all states have equal and positve cond. prob. density

How?: satisfied due to the slicing variable

C.4: If $(\theta, r) \in \mathcal{C}$ and $(\theta', r') \in \mathcal{C}$ then for any $\mathcal{B}$, $p(\mathcal{B}, \mathcal{C} | \theta, r, u, \epsilon) = p(\mathcal{B}, \mathcal{C} | \theta', r', u, \epsilon)$.

Why?: B and C must have equal prob. to be selected regardless of the state

How?: exclude from C any state that could not have generated B

Appendix: Transiton Kernel of the Efficient NUTS

$$T(w'|w,\mathcal{C}) = \begin{cases} \frac{\mathbb{I}[w' \in \mathcal{C}^{\text{new}}]}{|\mathcal{C}^{\text{new}}|} & \text{if } |\mathcal{C}^{\text{new}}| > |\mathcal{C}^{\text{old}}|, \\ \frac{|\mathcal{C}^{\text{new}}|}{|\mathcal{C}^{\text{old}}|} \frac{\mathbb{I}[w' \in \mathcal{C}^{\text{new}}]}{|\mathcal{C}^{\text{new}}|} + \left(1 - \frac{|\mathcal{C}^{\text{new}}|}{|\mathcal{C}^{\text{old}}|}\right) \mathbb{I}[w' = w] & \text{if } |\mathcal{C}^{\text{new}}| \leq |\mathcal{C}^{\text{old}}| \end{cases}$$

Appendix: Algorithm -> Differences in Main Loop

Algorithm 2 Naive No-U-Turn Sampler

Given θ^0, ϵ, $\mathcal{L}$, M:
for $m = 1$ to M **do**
  Resample $r^0 \sim \mathcal{N}(0, I)$.
  Resample $u \sim \text{Uniform}([0, \exp\{\mathcal{L}(\theta^{m-1} - \frac{1}{2}r^0 \cdot r^0\}])$
  Initialize $\theta^- = \theta^{m-1}$, $\theta^+ = \theta^{m-1}$, $r^- = r^0$, $r^+ = r^0$, $j = 0$, $\mathcal{C} = \{(\theta^{m-1}, r^0)\}, s = 1$.
  while $s = 1$ **do**
    Choose a direction $v_j \sim \text{Uniform}(\{-1, 1\})$.
    if $v_j = -1$ **then**
      $\theta^-, r^-, -, -, \mathcal{C}', s' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v_j, j, \epsilon)$.
    else
      $-, -, \theta^+, r^+, \mathcal{C}', s' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v_j, j, \epsilon)$.
    end if
    if $s' = 1$ **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$.
    end if
    $s \leftarrow s'\mathbb{I}[(\theta^+ - \theta^-) \cdot r^- \geq 0]\mathbb{I}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$.
    $j \leftarrow j + 1$.
  end while
  Sample θ^m, r uniformly at random from $\mathcal{C}$.
end for

Algorithm 3 Efficient No-U-Turn Sampler

Given θ^0, ϵ, $\mathcal{L}$, M:
for $m = 1$ to M **do**
  Resample $r^0 \sim \mathcal{N}(0, I)$.
  Resample $u \sim \text{Uniform}([0, \exp\{\mathcal{L}(\theta^{m-1} - \frac{1}{2}r^0 \cdot r^0\}])$
  Initialize $\theta^- = \theta^{m-1}$, $\theta^+ = \theta^{m-1}$, $r^- = r^0, r^+ = r^0, j = 0, \theta^m = \theta^{m-1}, n = 1, s = 1$.
  while $s = 1$ **do**
    Choose a direction $v_j \sim \text{Uniform}(\{-1, 1\})$.
    if $v_j = -1$ **then**
      $\theta^-, r^-, -, -, \theta', n', s' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v_j, j, \epsilon)$.
    else
      $-, -, \theta^+, r^+, \theta', n', s' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v_j, j, \epsilon)$.
    end if
    if $s' = 1$ **then**
      With probability $\min\{1, \frac{n'}{n}\}$, set $\theta^m \leftarrow \theta'$.
    end if
    $n \leftarrow n + n'$.
    $s \leftarrow s'\mathbb{I}[(\theta^+ - \theta^-) \cdot r^- \geq 0]\mathbb{I}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$.
    $j \leftarrow j + 1$.
  end while
end for

Appendix: Algorithm -> Differences in Tree Growing

Naive

function BuildTree$(\theta, r, u, v, j, \epsilon)$
if $j = 0$ **then**
  Base case—take one leapfrog step in the direction v.
  $\theta', r' \leftarrow \text{Leapfrog}(\theta, r, v\epsilon)$.
  $\mathcal{C}' \leftarrow \begin{cases} \{(\theta', r')\} & \text{if } u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r'\} \\ \emptyset & \text{else} \end{cases}$
  $s' \leftarrow \mathbb{I}[\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r' > \log u - \Delta_{\max}]$.
  return $\theta', r', \theta', r', \mathcal{C}', s'$.
else
  Recursion—build the left and right subtrees.
  $\theta^-, r^-, \theta^+, r^+, \mathcal{C}', s' \leftarrow \text{BuildTree}(\theta, r, u, v, j-1, \epsilon)$.
  if $v = -1$ **then**
    $\theta^-, r^-, -, -, \mathcal{C}'', s'' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v, j-1, \epsilon)$.
  else
    $-, -, \theta^+, r^+, \mathcal{C}'', s'' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v, j-1, \epsilon)$.
  end if
  $s' \leftarrow s' s'' \mathbb{I}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{I}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$.
  $\mathcal{C}' \leftarrow \mathcal{C}' \cup \mathcal{C}''$.
  return $\theta^-, r^-, \theta^+, r^+, \mathcal{C}', s'$.
end if

Efficient

function BuildTree$(\theta, r, u, v, j, \epsilon)$
if $j = 0$ **then**
  Base case—take one leapfrog step in the direction v.
  $\theta', r' \leftarrow \text{Leapfrog}(\theta, r, v\epsilon)$.
  $n' \leftarrow \mathbb{I}[u \leq \exp\{\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r'\}]$.
  $s' \leftarrow \mathbb{I}[\mathcal{L}(\theta') - \frac{1}{2} r' \cdot r' > \log u - \Delta_{\max}]$
  return $\theta', r', \theta', r', \theta', n', s'$.
else
  Recursion—implicitly build the left and right subtrees.
  $\theta^-, r^-, \theta^+, r^+, \theta', n', s' \leftarrow \text{BuildTree}(\theta, r, u, v, j-1, \epsilon)$.
  if $s' = 1$ **then**
    if $v = -1$ **then**
      $\theta^-, r^-, -, -, \theta'', n'', s'' \leftarrow \text{BuildTree}(\theta^-, r^-, u, v, j-1, \epsilon)$.
    else
      $-, -, \theta^+, r^+, \theta'', n'', s'' \leftarrow \text{BuildTree}(\theta^+, r^+, u, v, j-1, \epsilon)$.
    end if
    With probability $\frac{n''}{n'+n''}$, set $\theta' \leftarrow \theta''$.
    $s' \leftarrow s'' \mathbb{I}[(\theta^+ - \theta^-) \cdot r^- \geq 0] \mathbb{I}[(\theta^+ - \theta^-) \cdot r^+ \geq 0]$
    $n' \leftarrow n' + n''$
  end if
  return $\theta^-, r^-, \theta^+, r^+, \theta', n', s'$.
end if