The NUTS algorithm and its efficiency

Celia Caballero Cárdenas
Tutor: Jesús María Sanz Serna

Universidad Carlos III de Madrid

Máster en Ingeniería Matemática

26th March 2021

Índex

1. Markov Chain Monte Carlo methods
2. The Hamiltonian Monte Carlo method
3. The NUTS algorithm
4. Efficiency of the NUTS algorithm

Why are MCMC methods useful?

- **Problem:** having to sample from a random variable with pdf μ.
- **One way to solve it:** using Markov Chain Monte Carlo methods.

Importance of MCMC

- Bayesian statistics could not take off until MCMC methods were developed.
- Very important in other disciplines: bio-statistics, machine learning, artificial intelligence, population modelling, etc.

MCMC: Motivation

When working with univariate probability distributions it is necessary to know certain values associated to them such as:

$$\mu = \int_{\mathbb{R}} x\rho(x)dx,$$
$$\sigma^2 = \int_{\mathbb{R}} x^2\rho(x)dx - \mu^2,$$
$$P((-\infty, a)) = \int_{-\infty}^{a} \rho(x)dx = \int_{\mathbb{R}} 1_{(-\infty,a)}(x)\rho(x)dx.$$

In general, for random variables in $\mathbb{R}^d$ with probability density function ρ we need to compute

$$\mathbf{E}(F(X)) = \int_{\mathbb{R}^d} F(x)\rho(x)dx.$$

MCMC: Some important definitions

A **Markov chain** in the state space $\mathbb{R}^d$ is a sequence of random vectors of dimension d, $\{X_i\}_{i\in\mathbb{N}}$, that satisfy the Markov property:

$$\mathbf{P}(X_{i+1} \in A | X_1, \ldots, X_i) = \mathbf{P}(X_{i+1} \in A | X_i) \tag{1}$$

for every measurable set A and for every $i \in \mathbb{N}$.

MCMC: Some important definitions

A **Markov chain** in the state space $\mathbb{R}^d$ is a sequence of random vectors of dimension d, $\{X_i\}_{i\in\mathbb{N}}$, that satisfy the Markov property:

$$\mathbf{P}(X_{i+1} \in A | X_1, \ldots, X_i) = \mathbf{P}(X_{i+1} \in A | X_i) \tag{1}$$

for every measurable set A and for every $i \in \mathbb{N}$.

Markov Chain Monte Carlo (MCMC) methods produce a Markov chain that has the target distribution as an invariant distribution.
This means that if X_i has pdf μ, so does X_{i+1}.

MCMC: Some important definitions

A **Markov chain** in the state space $\mathbb{R}^d$ is a sequence of random vectors of dimension d, $\{X_i\}_{i\in\mathbb{N}}$, that satisfy the Markov property:

$$\mathbf{P}(X_{i+1} \in A | X_1, \ldots, X_i) = \mathbf{P}(X_{i+1} \in A | X_i) \tag{1}$$

for every measurable set A and for every $i \in \mathbb{N}$.

Markov Chain Monte Carlo (MCMC) methods produce a Markov chain that has the target distribution as an invariant distribution.
This means that if X_i has pdf μ, so does X_{i+1}.

Then the expectation $\mathbf{E}(F(X))$ is approximated by an average

$$\frac{1}{N}\left(F(x_1) + \cdots + F(x_N)\right),$$

where $x_1, \ldots, x_N$ is a realization of the chain.

MCMC: detailed balance condition

Detailed balance condition: the probability of jumping from a set A to a set B in the next step of the chain is the same as the one of jumping from B to A.

MCMC: detailed balance condition

Detailed balance condition: the probability of jumping from a set A to a set B in the next step of the chain is the same as the one of jumping from B to A.

The detailed balance condition ensures the invariance of the probability distribution.

Índex

1. Markov Chain Monte Carlo methods
2. The Hamiltonian Monte Carlo method
3. The NUTS algorithm
4. Efficiency of the NUTS algorithm

HMC: a fictitious dynamical system

The **Hamiltonian Monte Carlo** (HMC) method is an MCMC method in (Duane et al., 1987).

Let $\mu : \mathbb{R}^d \to \mathbb{R}$ be the pdf we want to sample from (no need to normalize). A fictitious dynamical system is created:

- **Coordinate:** $\theta \in \mathbb{R}^d$.
- **Potential energy:** $\mathcal{L}(\theta) = -\log(\mu(\theta))$.
- **Force:** $F(\theta) = -\nabla\mathcal{L}(\theta)$
- **Momentum:** $r \in \mathbb{R}^d$.
- **Kinetic energy:** $\mathcal{T}(r) = \frac{1}{2} r^T M^{-1} r$, $(M = I)$.
- **Total energy:** $\mathcal{H} = \mathcal{T} + \mathcal{L}$
- **Equations of motion:** Hamilton's equations.

$$\frac{d}{dt}\theta = r, \qquad \frac{d}{dt} r = F(\theta).$$

HMC: Boltzmann-Gibbs distribution

The **Boltzmann-Gibbs distribution**, Π_{BG}, with pdf proportional to

$$\exp\left(-\frac{1}{2}r^T r\right)\exp\left(-\mathcal{L}(\theta)\right) \tag{2}$$

is invariant by the flow of Hamilton's equations.

- θ-marginal of $\Pi_{BG} \longrightarrow$ our target distribution
- r-marginal of $\Pi_{BG} \longrightarrow \mathcal{N}(0, I)$

HMC methods generate a Markov chain with elements $(\theta_i, r_i) \in \mathbb{R}^{2d}$ which has Π_{BG} as its invariant distribution and therefore, the chain with elements $\{\theta_i\}_{i\in\mathbf{N}}$ will have the target distribution as invariant distribution.

HMC: the algorithm

1. Let $\lambda > 0$ and Ψ_λ the numerical approximation of the solution of Hamilton's equations at time λ.
2. Let the current state of the chain be $(\theta_0, r_0) \in \mathbb{R}^{2d}$.
3. Generate $\xi_0 \sim \mathcal{N}(0, I)$.
4. Compute $\Psi_\lambda(\theta_0, \xi_0)$.
5. The next element of the chain will be

$$(\theta_1, r_1) = \gamma \Psi_\lambda(\theta_0, \xi_0) + (1 - \gamma)(\theta_0, -\xi_0), \tag{3}$$

with γ a Bernoulli random variable of parameter $\alpha(\theta_0, \xi_0)$, where

$$\alpha(\theta_0, \xi_0) = \min\{1, \exp\left(-(H(\Psi_\lambda(\theta_0, \xi_0)) - H(\theta_0, \xi_0))\right)\}. \tag{4}$$

This acceptance-rejection mechanism is necessary to suppress the bias introduced by the numerical integration.

HMC: the integrator

The numerical integrator of choice for this algorithm is the **Leapfrog integrator**. This integrator takes a step size, ε and a number of steps, L.

$$\Psi_\varepsilon^L = \varphi_{\varepsilon/2}^{(B)} \circ \overbrace{\left(\varphi_\varepsilon^{(A)} \circ \varphi_\varepsilon^{(B)}\right) \circ \ldots \left(\varphi_\varepsilon^{(A)} \circ \varphi_\varepsilon^{(B)}\right)}^{L-1 \text{ times}} \circ \varphi_\varepsilon^{(A)} \circ \varphi_{\varepsilon/2}^{(B)}, \tag{5}$$

where

$$\varphi_t^{(A)}(\theta, r) = (\theta + tr, r) \tag{6}$$

and

$$\varphi_t^{(B)}(\theta, r) = (\theta, r + tF(\theta)). \tag{7}$$

$\Longrightarrow$ To use HMC two parameters have to be set: the number of steps, L, and the step size, ε.

Índex

1. Markov Chain Monte Carlo methods
2. The Hamiltonian Monte Carlo method
3. The NUTS algorithm
4. Efficiency of the NUTS algorithm

The NUTS algorithm

The **No-U-Turn Sampler** (NUTS) is a modification of the original HMC that is proposed in (Hoffman and Gelman, 2014) whose aim is to avoid the need to set the parameter L.

Importance of NUTS

- More than 2300 citations in Google Scholar.
- STAN software: thousands of users rely on it for statistical modeling, data analysis, and prediction in the social, biological, and physical sciences, engineering, and business.

▸ Link

NUTS: criterion to stop the numerical integration

Criterion to stop the numerical integration

If running additional time steps would not result in a higher distance between the initial state, θ, and the proposal, $\tilde{\theta}$, then performing additional Leapfrog steps would lead to higher computational cost without providing a better proposal.

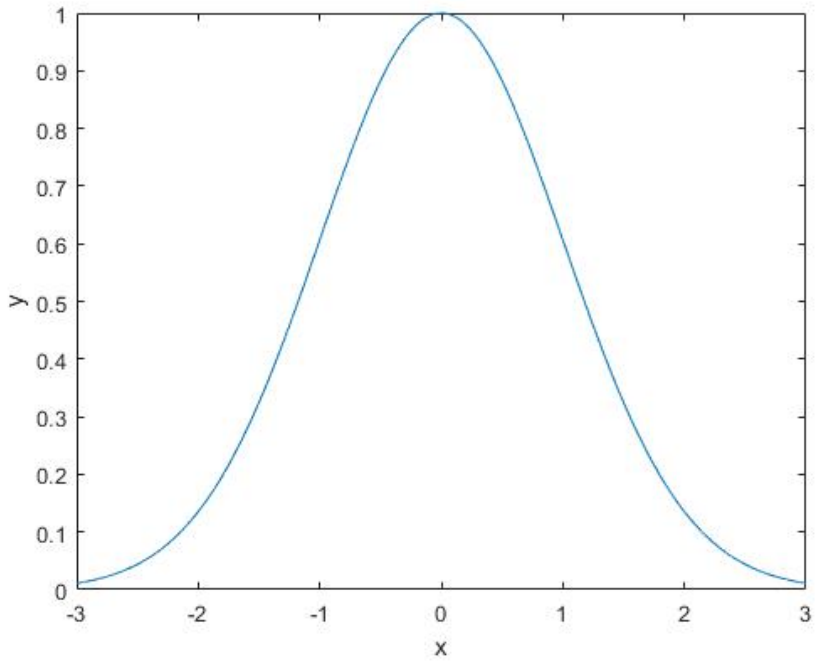
$$\frac{d}{dt}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{d}{dt}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot\tilde{r}. \tag{8}$$

- If $(\tilde{\theta}-\theta)\cdot\tilde{r} > 0 \longrightarrow$ keep on simulating the system's dynamics.
- As soon as $(\tilde{\theta}-\theta)\cdot\tilde{r} < 0 \longrightarrow$ stop the simulation.

NUTS: slice sampling

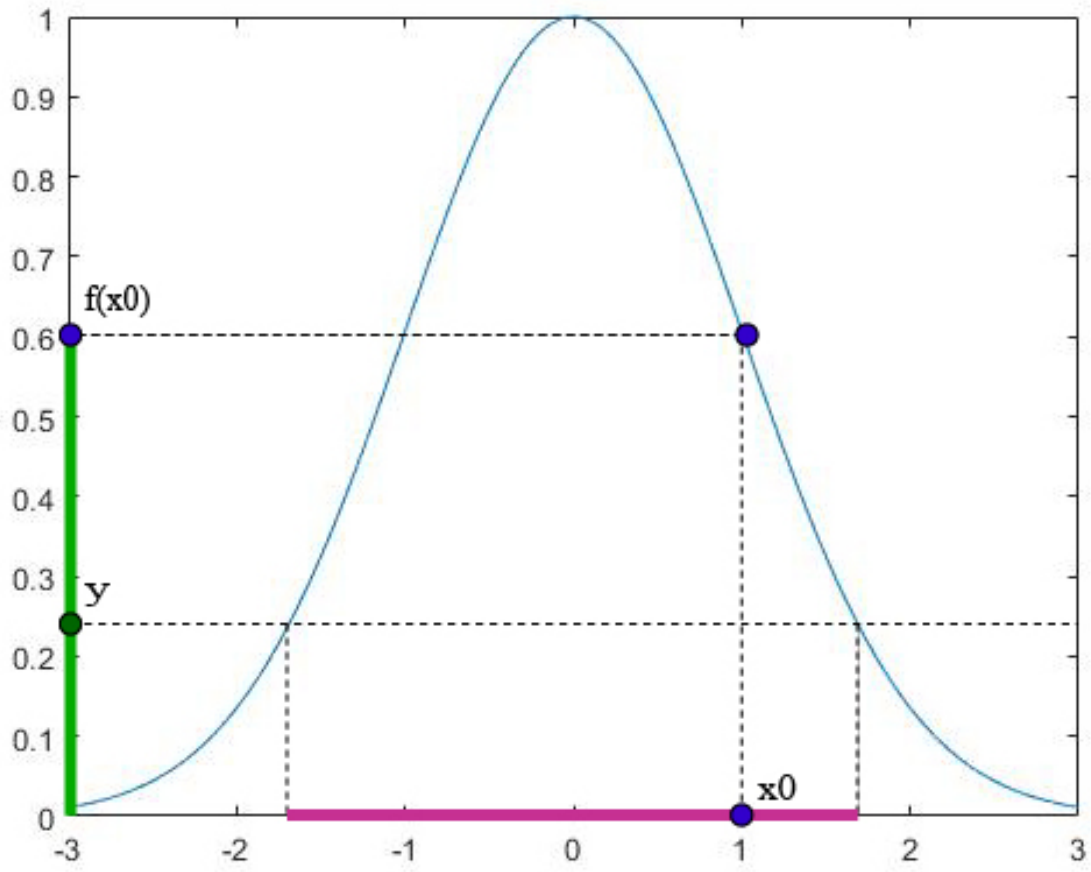
Slice sampling: sampling from a pdf $f : \mathbb{R}^d \to \mathbb{R}$ can be done by uniformly sampling points from the region under the graph of f and just keeping the variable x (Neal, 2003).

- $y \longrightarrow$ slice variable
- $y < f(x) \longrightarrow$ slice condition



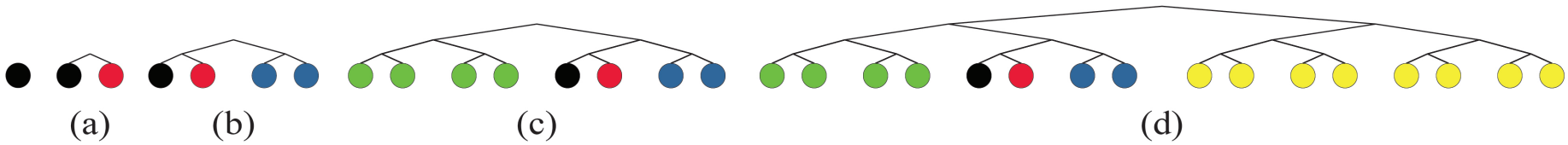
NUTS: slice sampling

How can we uniformly generate independent samples from the set $\{(x, y) \in \mathbb{R}^d \times \mathbb{R} : 0 < y < f(x)\}$?



NUTS: creation of the binary tree

NUTS sampling $\longrightarrow$ Binary tree with nodes of the form (θ, r).



NUTS stops building the tree when the leftmost (θ_-, r_-) and rightmost (θ_+, r_+) leaves of one of the subtrees satisfy either $(\theta_+ - \theta_-) \cdot r_- < 0$ or $(\theta_+ - \theta_-) \cdot r_+ < 0$.

NUTS: One additional stopping criterion

Stop when the simulation is not accurate enough.

Accuracy condition

$$-\mathcal{L}(\theta) - \frac{1}{2} r \cdot r - \log u < -\Delta_{max} \tag{9}$$

for some nonnegative Δ_{max}.

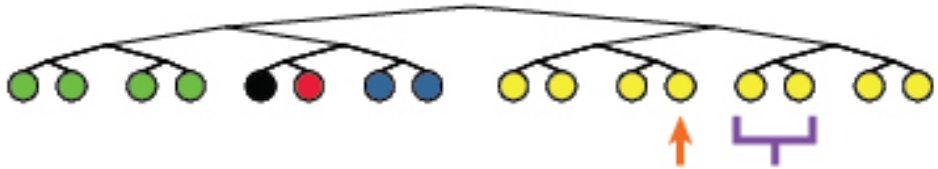
If this condition is violated we have reached a point of extremely low probability and there is no point in pursuing the integration of the Hamiltonian dynamics.

NUTS: Obtaining the final set to sample from

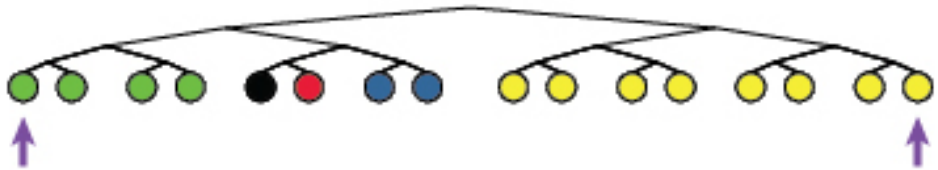
Set $\mathcal{B}$: all the nodes visited.

Set $\mathcal{C}$: nodes we can sample from.

- Every node in $\mathcal{C}$ must satisfy the **slice condition**.
- **Detailed balance**
  1. If one of these conditions was satisfied by a node or subtree added during the last doubling iteration.



  2. If the U-turn stopping condition was satisfied by the leftmost and rightmost leaves of the full tree.



A simple NUTS algorithm

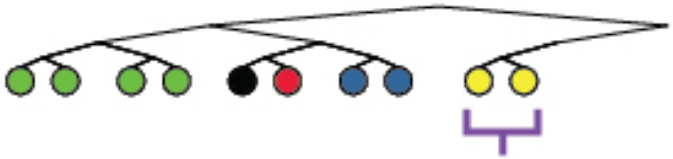
Inputs: $\theta^0,\ \varepsilon,\ \mathcal{L},\ \nabla\mathcal{L},\ M.$

For $m = 1, \ldots, M$:

1. **Initialization:**
   - $r^{m-1} \sim \mathcal{N}(0, I)$
   - Slice variable: $u \sim \text{Unif}\left(\left[0, \exp\left(-\mathcal{L}(\theta^{m-1}) - \frac{1}{2} r^{m-1} \cdot r^{m-1}\right)\right]\right)$
   - Tree height: $j = 0$
   - $\mathcal{C} = \{(\theta^{m-1}, r^{m-1})\}$
   - Indicator variable: $s = 1$
2. **While** $s = 1$ (recursive *Buildtree* function):
   - $v_j \sim \text{Unif}(\{-1, 1\})$ is sampled
   - 2^j steps of the Leapfrog integrator are computed
   - j is increased in 1 unit
   - The slice condition is always checked
3. **Once** $s = 0$: (θ^m, r^m) is sampled uniformly at random from $\mathcal{C}$.

A simple NUTS algorithm: some drawbacks

1. Large memory requirements
2. If one of the stopping criteria is satisfied during the last doubling iteration then all the states added during this iteration will not be included in the set $\mathcal{C}$ $\longrightarrow$ optimized use of the indicator variable



3. The way of choosing the element from $\mathcal{C}$ can be improved

Drawbacks 1 and 3 $\longrightarrow$ new kernel based on 2 mechanisms

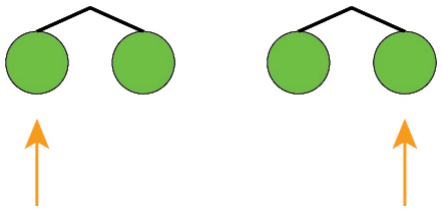
Efficient NUTS algorithm: new kernel

Mechanism 1: Inside the *BuildTree* function

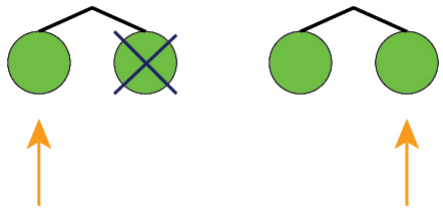
- θ', θ'': candidates of each of the subtrees
- n', n'': number of nodes of each of the subtrees that can be in $\mathcal{C}$

The candidate of the merged tree is θ'' instead of θ' with probability

$$P = \frac{n''}{n' + n''}. \tag{10}$$



(a) $n' = 2$, $n'' = 2$, $P = 1/2$



(b) $n' = 2$, $n'' = 1$, $P = 1/3$

Efficient NUTS algorithm: new kernel

Mechanism 2: Inside the while loop of the main function

- θ^m: candidate of the tree built in the previous iteration of the loop
- n: number of elements of this tree that can be in $\mathcal{C}$
- θ': candidate of the new subtree
- n': number of elements of this tree that can be in $\mathcal{C}$

The algorithm chooses θ' instead of θ^m with probability

$$P = \min\{1, n'/n\}. \tag{11}$$

(a) $n = 8,\ n' = 8,\ P = 1.$

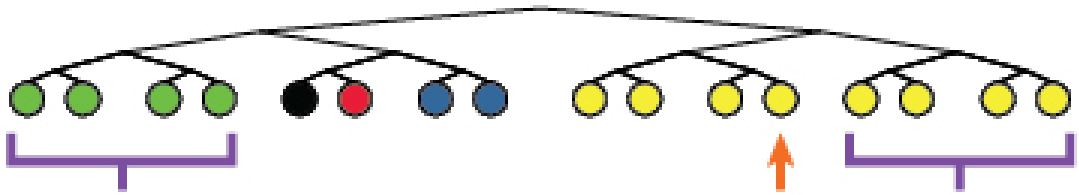
(b) $n = 7,\ n' = 6,\ P = 6/7.$

Índex

1. Markov Chain Monte Carlo methods
2. The Hamiltonian Monte Carlo method
3. The NUTS algorithm
4. Efficiency of the NUTS algorithm

Efficiency of the NUTS algorithm

How do we define the concept of efficiency?

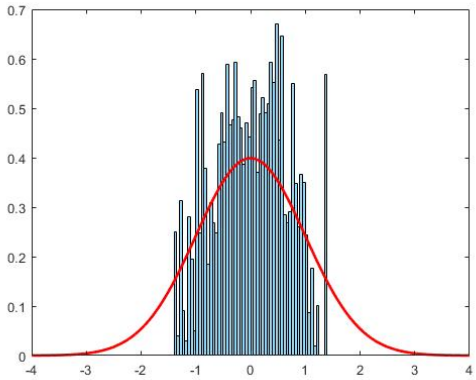


$\Longrightarrow$ Efficiency $= 7/15$ and Inefficiency $= 8/15$

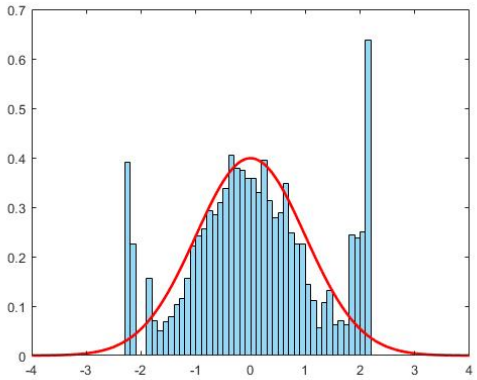
We will compute this value each time a new sample has been generated and obtain the **mean inefficiency value** for a realization of the Markov chain.

- This waste of computational effort is the price paid for not having to previously set the value L.
- The efficiency estimation is not available in the existing literature.

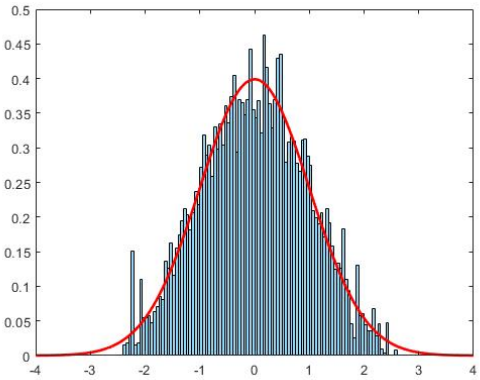
Simple NUTS: 1D standard normal distribution



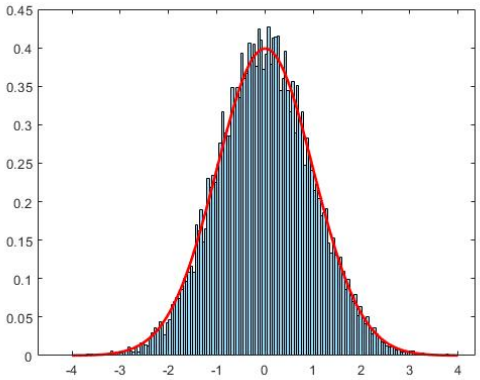
(a) $\varepsilon = 5$



(b) $\varepsilon = 4$

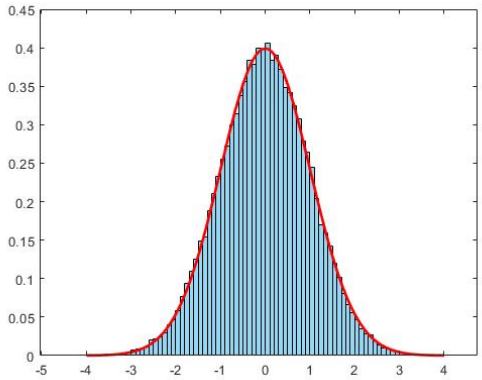


(c) $\varepsilon = 3$

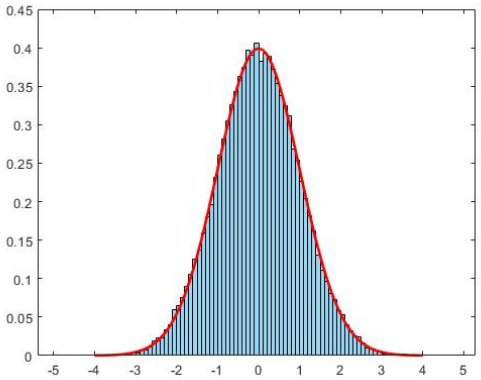


(d) $\varepsilon = 2$

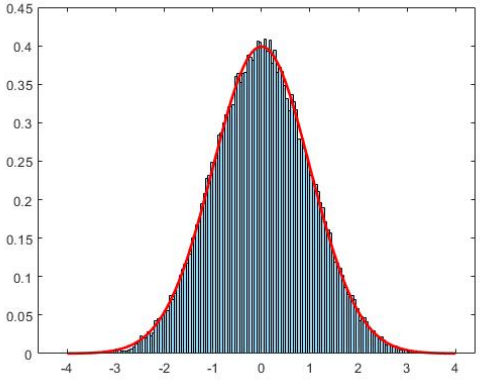
Simple NUTS: 1D standard normal distribution



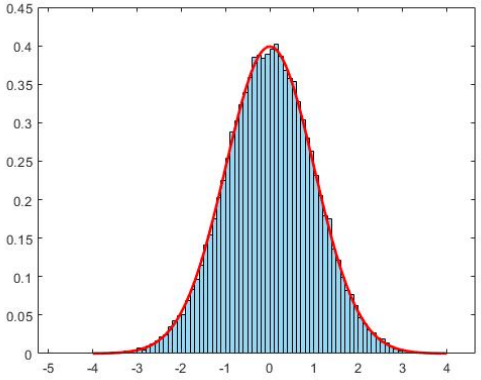
(a) $\varepsilon = 1$



(b) $\varepsilon = 0.1$



(c) $\varepsilon = 0.01$



(d) $\varepsilon = 0.001$

Simple NUTS: 1D standard normal distribution

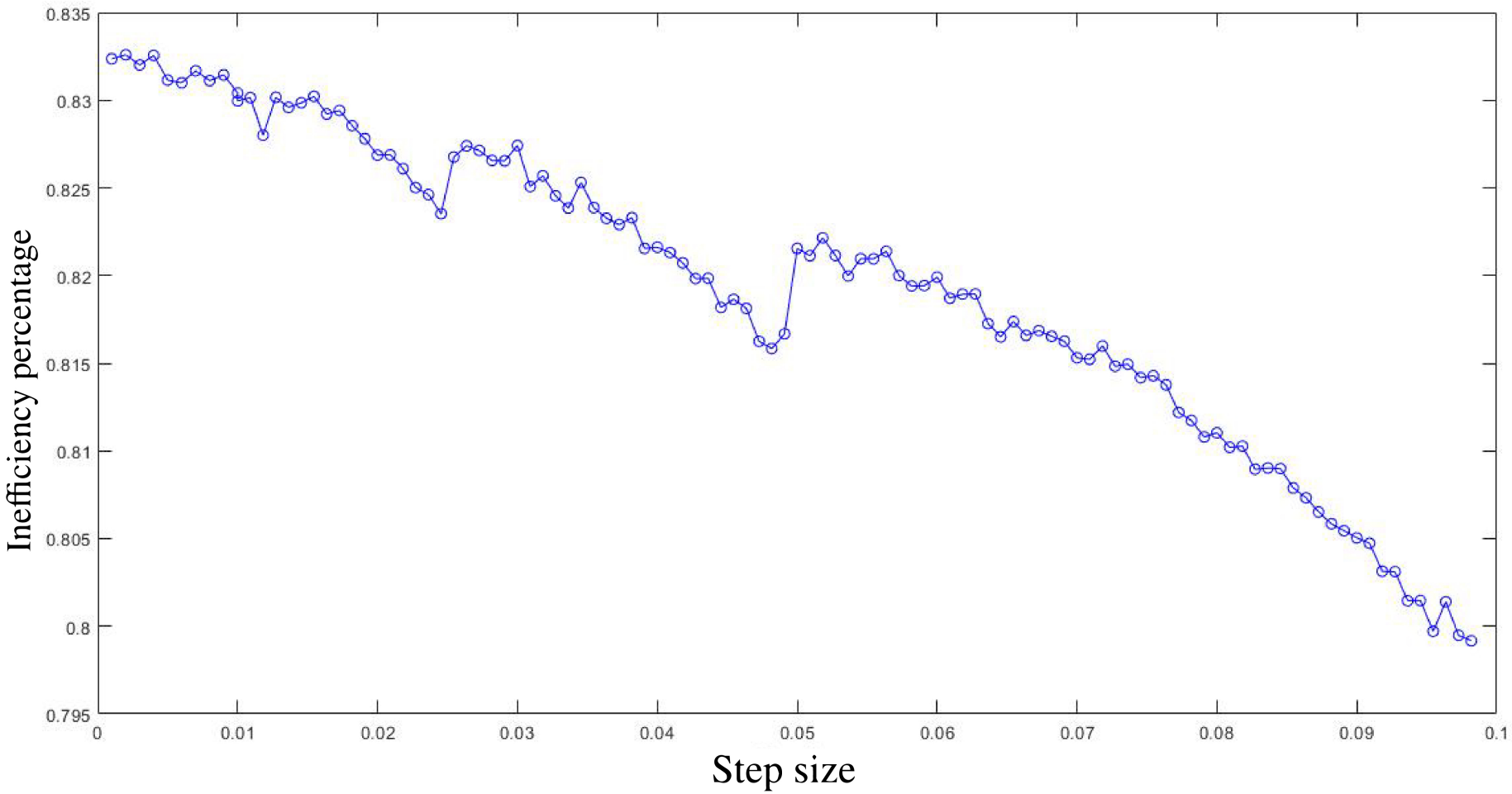


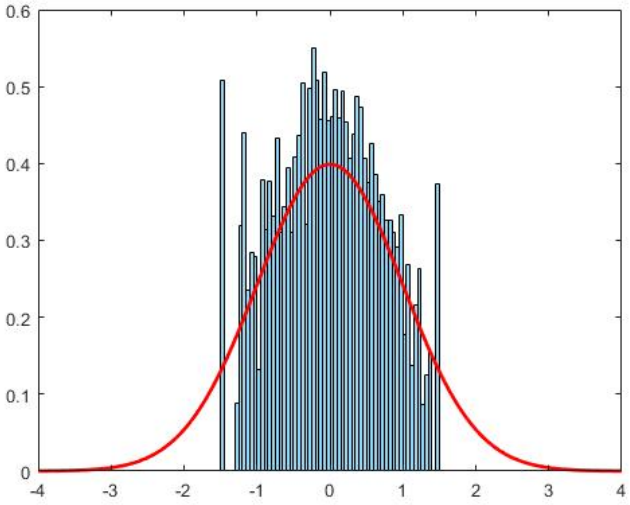
Figure 1: Inefficiency percentage for different values of the step size between 0.001 and 0.1 using the simple NUTS algorithm for a univariate normal distribution.

Simple NUTS: 1D standard normal distribution

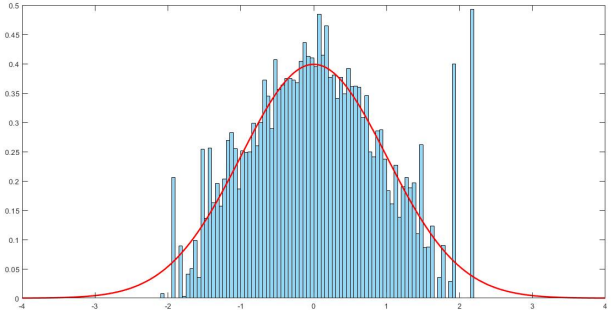
RESULTS

- The average number of steps is multiplied by 10 as the step size is divided by 10.
- The inefficiency percentage increases as the step size tends to 0.

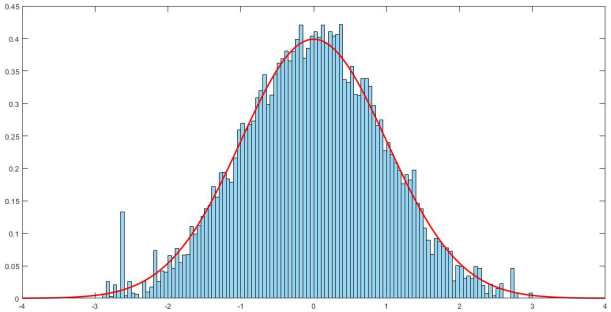
Efficient NUTS: 1D standard normal distribution



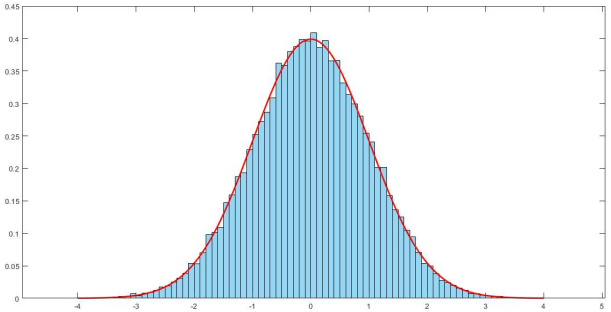
(a) $\varepsilon = 5$



(b) $\varepsilon = 4$

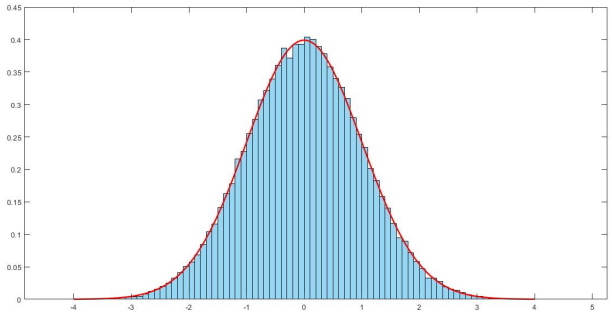


(c) $\varepsilon = 3$

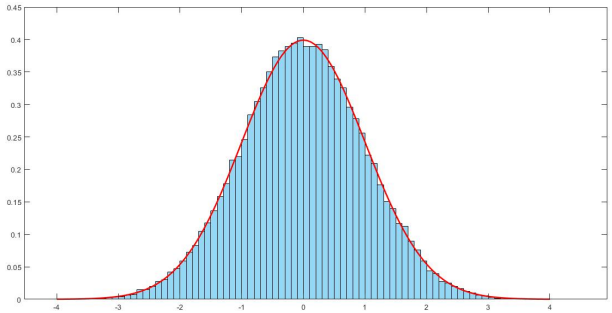


(d) $\varepsilon = 2$

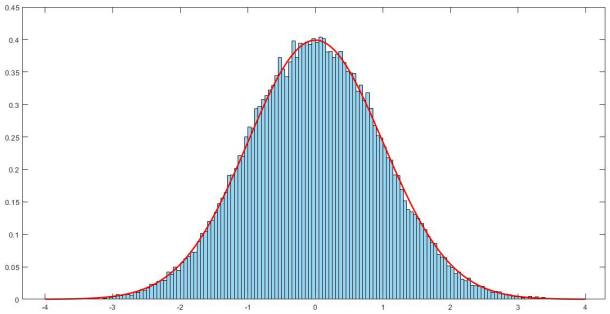
Efficient NUTS: 1D standard normal distribution



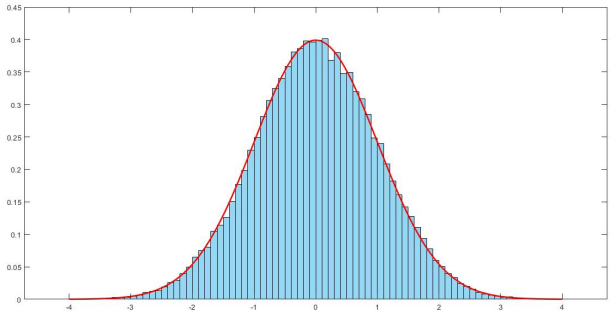
(a) $\varepsilon = 1$



(b) $\varepsilon = 0.1$



(c) $\varepsilon = 0.01$



(d) $\varepsilon = 0.001$

Efficient NUTS: 1D standard normal distribution

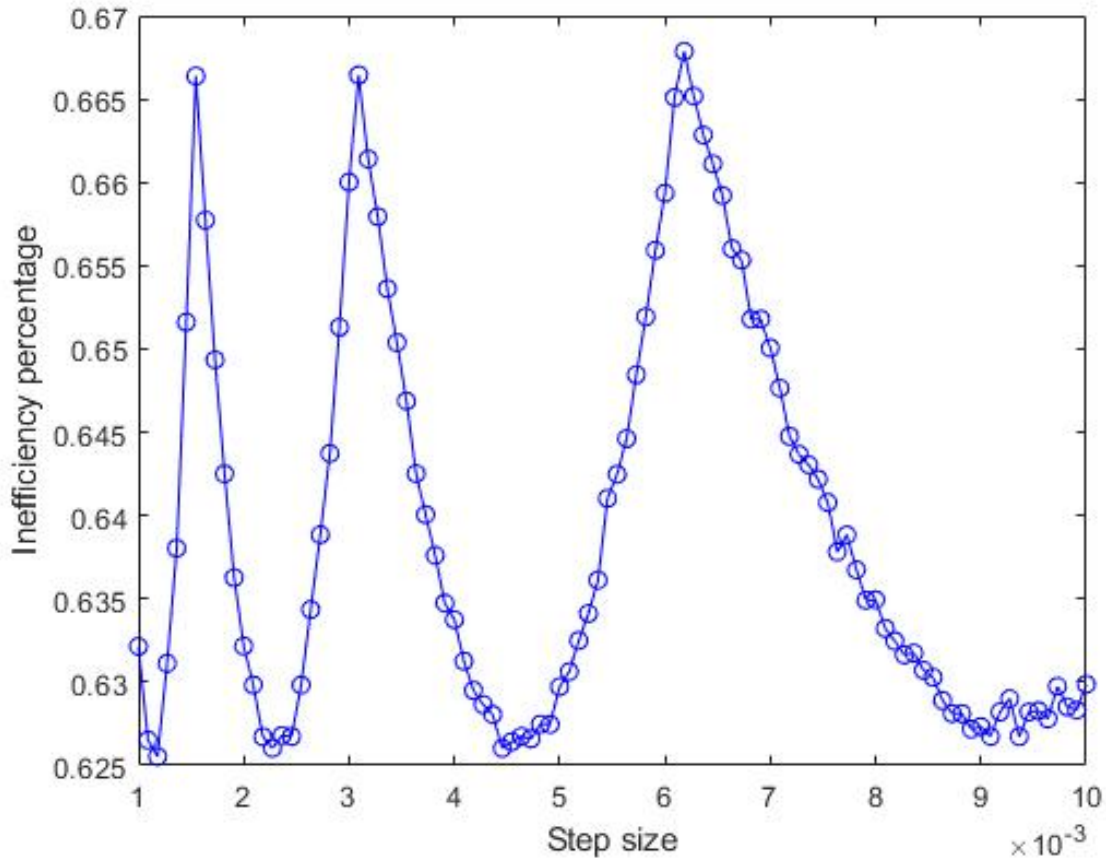


Figure 2: Inefficiency percentage for different values of the step size between 0.001 and 0.1 using the efficient NUTS algorithm for a univariate normal distribution.

Efficient NUTS: standard normal distribution

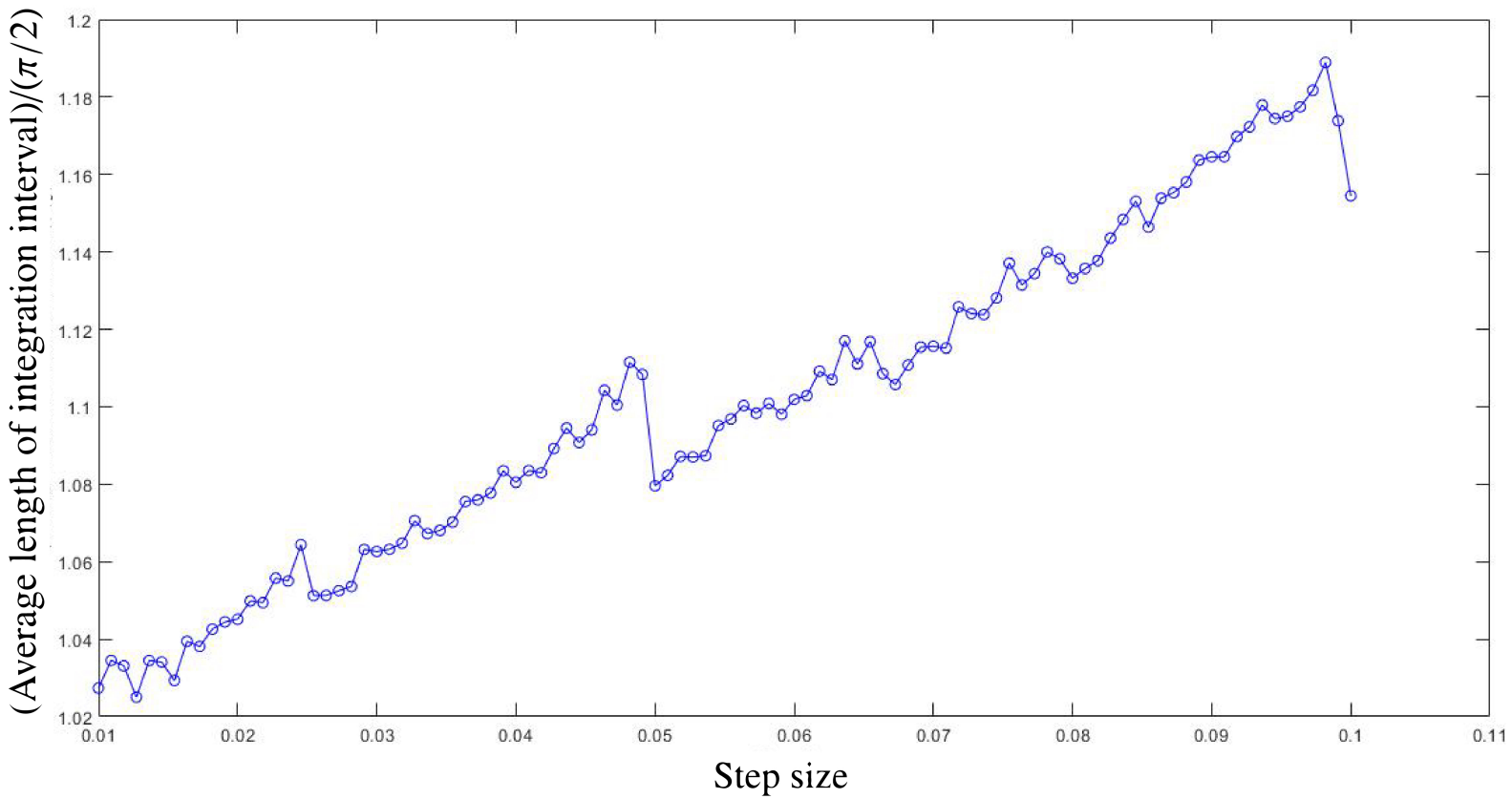


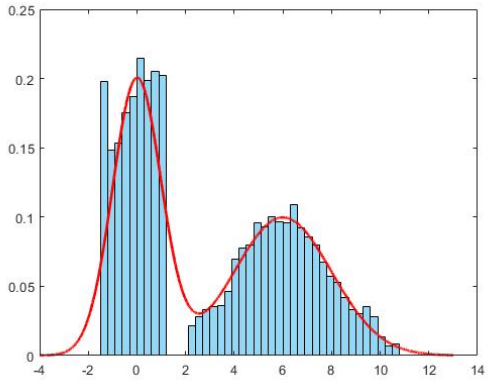
Figure 3: The average length of the integration interval divided by $\pi/2$ for different step sizes using efficient NUTS algorithm for a univariate normal distribution.

Efficient NUTS: 1D standard normal distribution

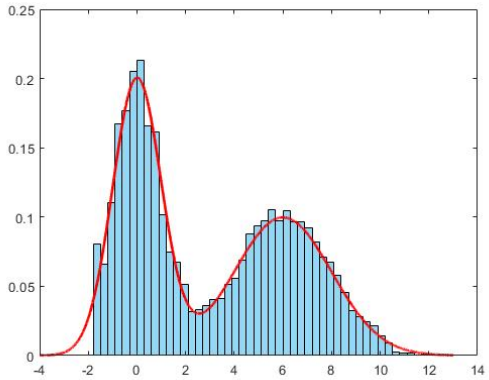
RESULTS

- Smaller average number of steps than simple NUTS.
- Reduction in the running time compared to simple NUTS.
- Lower inefficiency (between 62.5% and 67%).
- Periodic behaviour of the inefficiency percentage.
- The average length of the integration interval tends to $\pi/2$ as the step size tends to 0.
- Similar quality of the sample to simple NUTS.

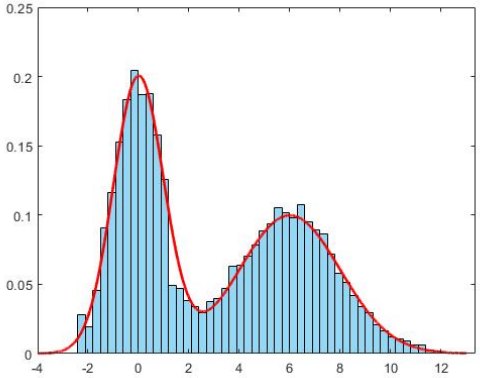
Efficient NUTS: mixture of $\mathcal{N}(0,1)$ and $\mathcal{N}(6,4)$



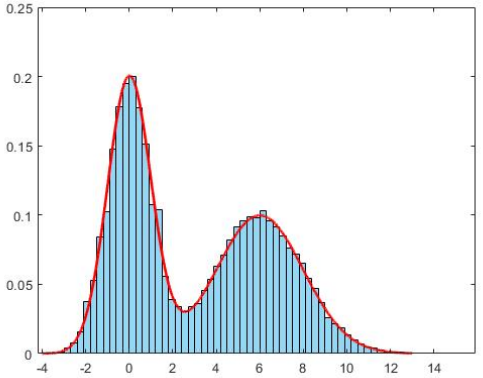
(a) $\varepsilon = 7$



(b) $\varepsilon = 6$

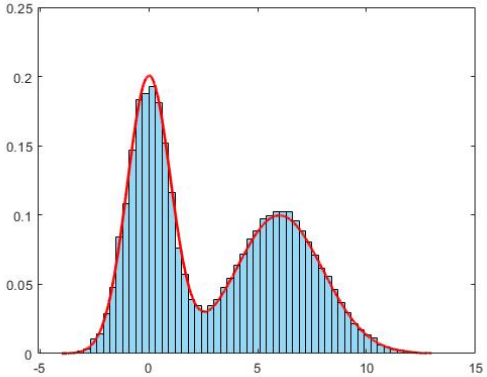


(c) $\varepsilon = 5$

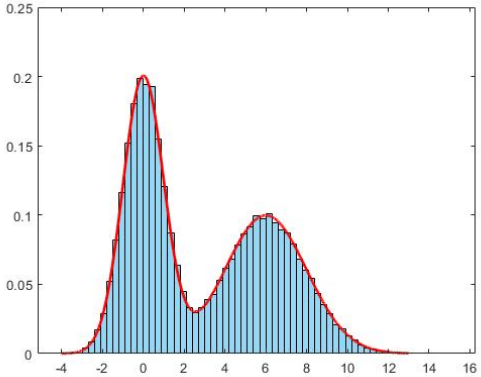


(d) $\varepsilon = 4$

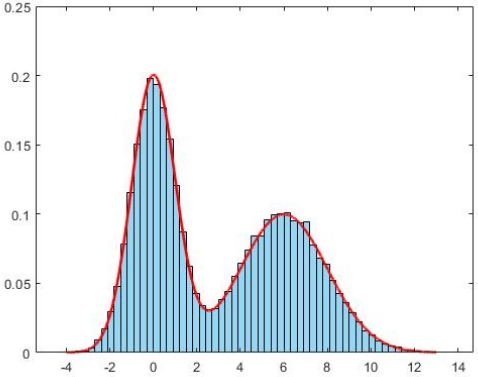
Efficient NUTS: mixture of $\mathcal{N}(0,1)$ and $\mathcal{N}(6,4)$



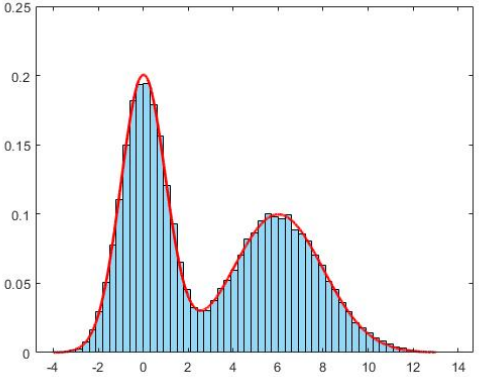
(a) $\varepsilon = 3$



(b) $\varepsilon = 2$



(c) $\varepsilon = 1$



(d) $\varepsilon = 0.1$

Efficient NUTS: mixture of $\mathcal{N}(0,1)$ and $\mathcal{N}(6,4)$

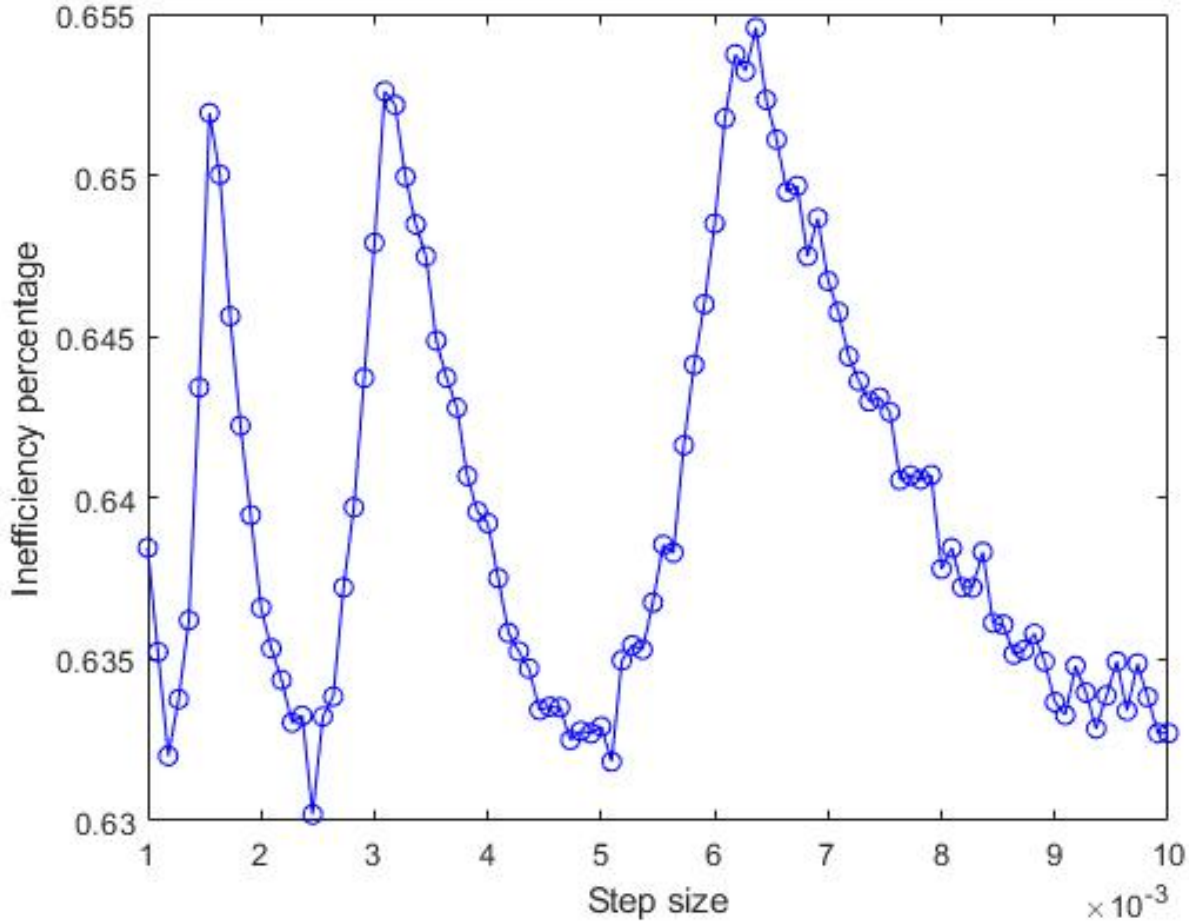


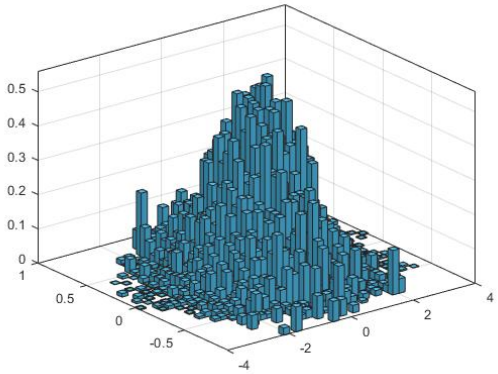
Figure 4: Inefficiency percentage for values of the step size between 0.001 and 0.01 using the efficient NUTS algorithm for a mixture of univariate normal distributions.

Efficient NUTS: mixture of $\mathcal{N}(0,1)$ and $\mathcal{N}(6,4)$

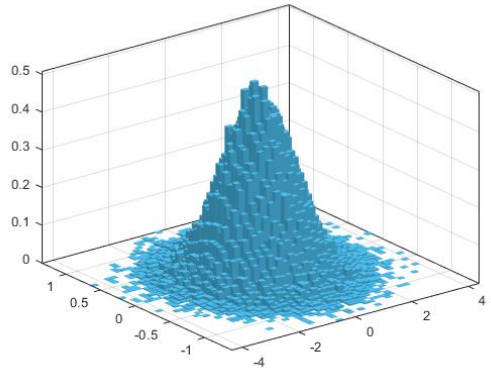
RESULTS

- Bigger average number of steps and elapsed time than in the standard normal distribution case.
- Elapsed time and average number of steps are multiplied by 10 as the step size is divided by 10.
- Similar inefficiency percentages than in the standard normal distribution.
- Periodic inefficiency percentage behaviour.

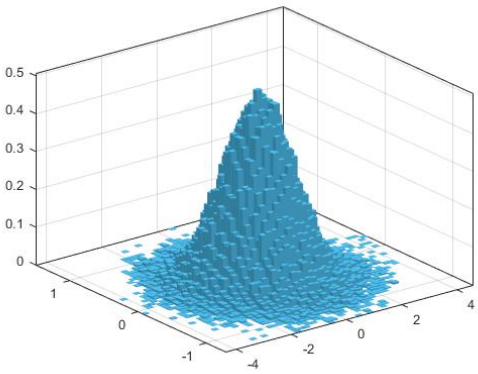
Efficient NUTS: Two-dimensional normal distribution



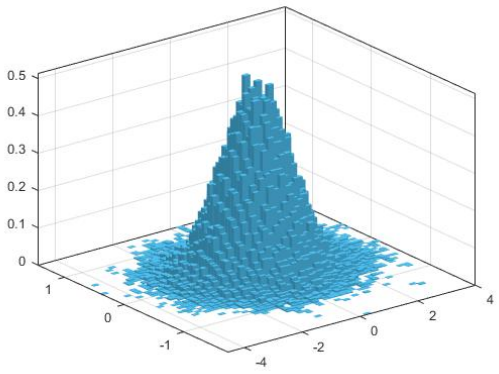
(a) $\varepsilon = 1$



(b) $\varepsilon = 0.1$



(c) $\varepsilon = 0.01$



(d) $\varepsilon = 0.001$

Efficient NUTS: Two-dimensional normal distribution

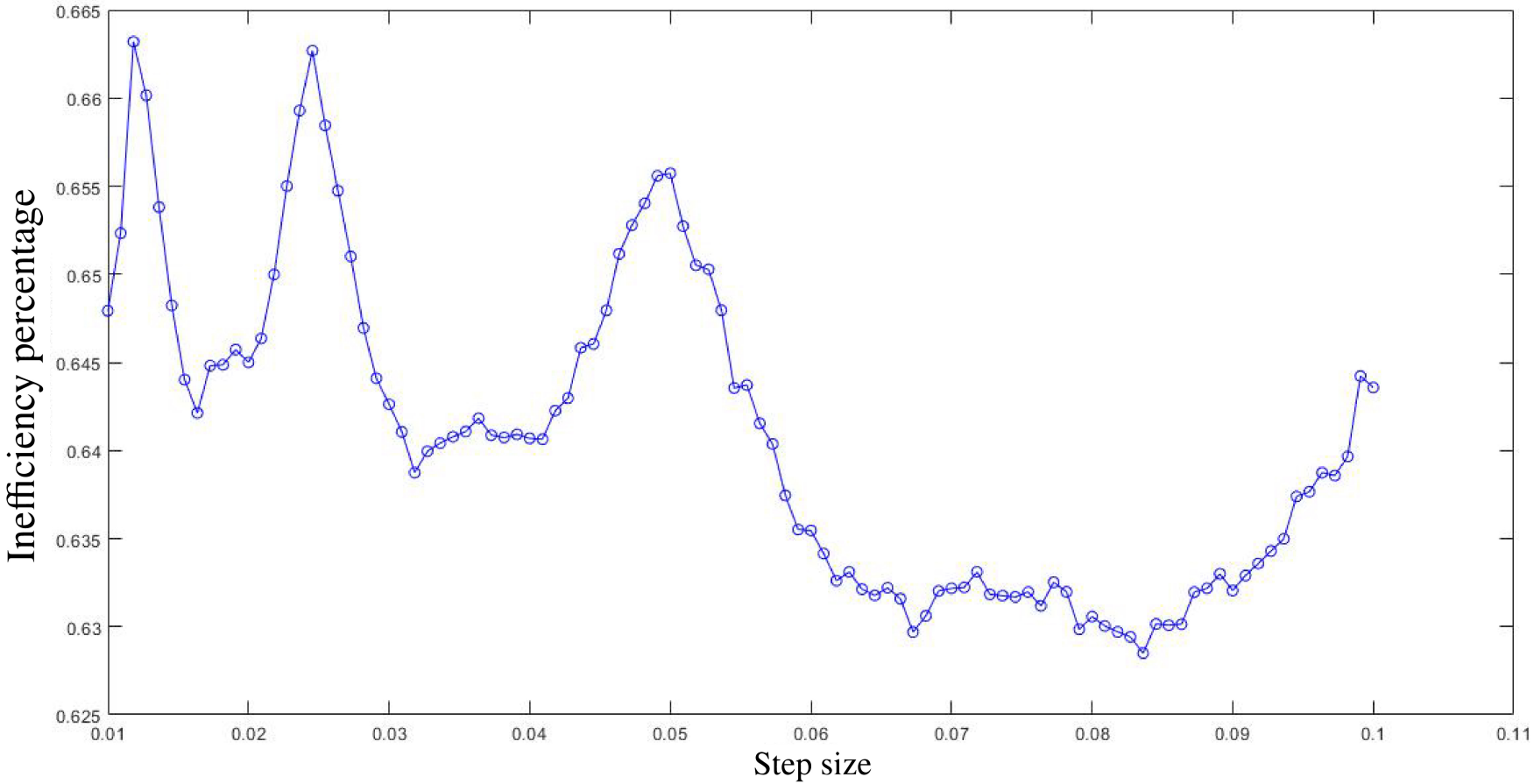


Figure 5: Inefficiency percentage for different values of the step size between 0.001 and 0.01 using the efficient NUTS algorithm for a bivariate normal distribution.

Efficient NUTS: Two-dimensional normal distribution

RESULTS

- Similar average number of steps than in the 1D standard normal distribution case.
- Similar inefficiency percentage range of values.
- Periodic inefficiency percentage behaviour.

Conclusions

- Good sampling results have been obtained when a sufficiently small step size was used without the need to previously set the number of steps.
- Similar values for the inefficiency percentage were obtained: between 62% and 67%.
- We cannot infer from these examples that the inefficiency percentages will be the same for every probability distribution.

Future lines of research

How does the NUTS algorithm perform when using **integrators different to Leapfrog**?
It would be interesting to consider the integrators proposed for HMC in (Bou-Rabee and Sanz-Serna, 2018) and (Blanes et al., 2014): **splitting integrators with more stages**.

References I


Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan.
An Introduction to MCMC for Machine Learning.
Machine Learning, 50:5–43, 2003.


Alexandros Beskos, Natesh Pillai, Gareth Roberts, Jesus Maria Sanz-Serna, and Andrew Stuart.
Optimal tuning of the hybrid Monte Carlo algorithm.
Bernoulli, 19(5A):1501–1534, 11 2013.


Sergio Blanes, Fernando Casas, and Jesús María Sanz-Serna.
Numerical Integrators for the Hybrid Monte Carlo Method.
SIAM Journal on Scientific Computing, 36(4):A1556–A1580, Jan 2014.


Nawaf Bou-Rabee and Jesús María Sanz-Serna.
Geometric Integrators and the Hamiltonian Monte Carlo method.
Acta Numerica, 27:113–206, 2018.

References II


M. P. Calvo, D. Sanz-Alonso, and J. M. Sanz-Serna.
HMC: avoiding rejections by not using leapfrog and some results on the acceptance rate.
2019.


Simon Duane, A. D. Kennedy, Brian J. Pendleton, and Duncan Roweth.
Hybrid Monte Carlo.
Physics Letters B, 195(2):216 – 222, 1987.


Z Ghahramani.
Probabilistic machine learning and artificial intelligence.
Nature, 521:452–459, May 2015.

References III


Matthew D. Hoffman and Andrew Gelman.
The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo.
Journal of Machine Learning Research, 15:1593–1623, January 2014.


Shane T. Jensen, X. Shirley Liu, Qing Zhou, and Jun S. Liu.
Computational discovery of gene regulatory binding motifs: A bayesian perspective.
Statistical Science, 19(1):188–204, 02 2004.


William A. Link and Richard J. Barker.
Bayesian Inference: With Ecological Applications.
Academic Press, Oxford, 2009.

References IV


Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller.
Equation of State Calculations by Fast Computing Machines.
The Journal of Chemical Physics, 21:1087–1092, 1953.


Radford M. Neal.
Slice sampling.
The Annals of Statistics, 31(3):705–767, 06 2003.


G. O. Roberts, A. Gelman, and W. R. Gilks.
Weak convergence and optimal scaling of Random Walk Metropolis algorithms.
The Annals of Applied Probability, 7(1):110–120, 02 1997.

References V


Jesús María Sanz-Serna.
Markov Chain Monte Carlo and Numerical Differential Equations, pages 39–88.
Springer, Switzerland, 2014.

The NUTS algorithm and its efficiency

Celia Caballero Cárdenas
Tutor: Jesús María Sanz Serna

Universidad Carlos III de Madrid

Máster en Ingeniería Matemática

26th March 2021

Simple NUTS: 1D standard normal distribution

Step size	**0.001**	**0.01**	**0.1**	**1**
Elapsed time	508.97	53.55	7.83	2.83
Average steps	$2.3{\cdot}10^3$	235.02	21.57	2.37
% inefficiency	0.8339	0.8303	0.8115	0.6897
% slice failure	0	$3.4{\cdot}10^{-6}$	$6.5 \cdot 10^{-4}$	0.08
Average	0.0023	-0.0168	0.0105	0.0113
Variance	1.0130	1.0080	0.9918	1.0046

Step size	**2**	**3**	**4**	**5**
Elapsed time	2.20	2.22	1.66	1.45
Average steps	1	1	1	1
% inefficiency	0.7485	0.9110	0.9646	0.9746
% slice failure	0.50	0.82	0.93	0.95
Average	-0.0098	0.0099	0.0885	0.0052
Variance	1.0106	1.0606	1.4017	0.4458

Efficient NUTS: 1D standard normal distribution

Step size	**0.001**	**0.01**	**0.1**	**1**
Elapsed time	135.82	14.84	3.18	1.29
Average steps	$1.57 \cdot 10^3$	161.54	18.21	2.37
% inefficiency	0.6331	0.6292	0.6618	0.4710
% slice failure	0	0	$6.92 \cdot 10^{-4}$	0.08
Average	-0.0017	0.0170	0.0068	-0.0025
Variance	1.0036	1.0043	0.9953	0.9934

Step size	**2**	**3**	**4**	**5**
Elapsed time	1.56	1.69	1.39	1.44
Average steps	1	1	1	1
% inefficiency	0.5008	0.8208	0.9178	0.9516
% slice failure	0.5008	0.8208	0.9178	0.9516
Average	-0.0015	0.0117	-0.1024	-0.0139
Variance	0.9974	1.1401	0.8473	0.5340

Efficient NUTS: mixture of $\mathcal{N}(0,1)$ and $\mathcal{N}(6,4)$

Step size	**0.001**	**0.01**	**0.1**	**1**	**2**
Elapsed time	439.25	53.13	6.54	2.27	2.06
Average steps	$2.72 \cdot 10^3$	275.83	30.16	3.97	2.16
% inefficiency	0.6368	0.6325	0.6470	0.5489	0.5668
% slice failure	0	0	$3.86 \cdot 10^{-4}$	0.0518	0.2954
Average	2.9684	2.9127	3.0346	3.0188	2.9665
Variance	11.5021	11.4234	11.5697	11.4106	11.4467

Step size	**3**	**4**	**5**	**6**	**7**
Elapsed time	2.15	1.95	1.9	1.82	1.65
Average steps	1.80	1.44	1.25	1.13	1.09
% inefficiency	0.6201	0.7304	0.7908	0.8483	0.8954
% slice failure	0.4860	0.6721	0.7696	0.8389	0.8913
Average	3.0728	3.0298	3.0564	3.0791	2.9670
Variance	11.5438	11.5579	11.4930	11.1341	11.52

Efficient NUTS: Two-dimensional normal distribution

Step size	**0.001**	**0.01**	**0.1**	**1**
Elapsed time	462.18	47.04	8.01	3.15
Average steps	$1.6 \cdot 10^3$	159.95	16.81	1.33
% inefficiency	0.6465	0.6473	0.6433	0.8640
% slice failure	$1.73 \cdot 10^{-5}$	$3.78 \cdot 10^{-5}$	0.0045	0.8507
Average X	$9.16 \cdot 10^{-4}$	-0.0037	-0.0114	-0.0076
Average Y	-0.0015	$-3.07 \cdot 10^{-4}$	-0.0011	-0.0019
Variance X	0.9978	1.0268	0.9975	0.9673
Variance Y	0.1101	0.1113	0.1119	0.1065