École polytechnique fédérale de Lausanne

HMC : When is it worth over RWMC ?

Stochastic Simulation Project Report

Authors:
Aude Maier
Tara Fjellman

Professor:
Fabio Nobile

Contents

1 Introduction

Markov Chain Monte Carlo (MCMC) methods are essential tools for sampling from complex probability distributions, particularly in Bayesian inference and high-dimensional statistical modelling. Knowing which method to choose can however be challenging. This report focuses on how to make this choice based on problem characteristics between two prominent MCMC algorithms: Random Walk MC (RWMC) and Hamiltonian MC (HMC).

We first present an overview of the theoretical foundations of HMC, and prove important properties for it. We then explore and interpret how parameters, such as step size for RWMC and mass for HMC influence performance. To systematically assess and compare their efficiency, we use among others a similarity metric and effective sample size. Specifically, we consider two case studies: a 2D toy example with tunable complexity and a higher-dimensional 10D birthweight dataset.

Our findings suggest that RWMC performs better in low-complexity, low-dimensional scenarios, while HMC proves superior as complexity and dimensionality increase.

2 Hamiltonian Monte Carlo

2.1 The Algorithm

Hamiltonian Monte Carlo is a more sophisticated MCMC algorithm designed to display better exploration properties than RWMC, whose main shortfall is the slow exploration rate of the state space. It introduces fictitious variables $\{p_i\}$ to the state space, and simulates Hamiltonian dynamics to propose new states. The Hamiltonian function $H(q,p)$ treats the fictitious variables as momenta, and the variable of interest $\{q_i\}$ as positions. $H(q,p)$ is defined as the sum of the potential energy $U(q) = -\log \pi(q)$, with $\pi(q)$ the unormalized target distribution from which we want to sample, and the kinetic energy $K(p) = \sum_{i=1}^{d} \frac{p_i^2}{2m_i}$, where m_i are mass parameters.

The dynamics are governed by the Hamiltonian equations :

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \tag{1}$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}, \tag{2}$$

for $i = 1, \ldots, d$.

The dynamics is used to propose new states, which are then accepted or rejected based on the Metropolis-Hastings acceptance rule. Concretely, starting from a state vector q^n, a momentum vector p^n is sampled from a Gaussian distribution $N(0, M)$ with $M = \text{diag}\,(m_1, \ldots, m_d)$, and the Hamiltonian system is evolved for a time T to obtain the proposal (q^*, p^*). The acceptance rate for the new state q^{n+1} is then computed as:

$$\alpha = \min\left(1, \exp\left[U\left(q^n\right) + K\left(p^n\right) - U\left(q^*\right) - K\left(p^*\right)\right]\right). \tag{3}$$

In practice, analytical time integration of the Hamiltonian dynamics is often not possible, and numerical discretization is needed. A common choice is Verlet's method, which divides the time interval $[0, T]$ into N_T intervals of size $\varepsilon > 0$. The position and momentum are updated as follows:

$$p_i(t + \varepsilon/2) = p_i(t) - (\varepsilon/2)\frac{\partial U(q(t))}{\partial q_i}, \tag{4}$$

$$q_i(t + \varepsilon) = q_i(t) + \varepsilon \frac{p_i(t + \varepsilon/2)}{m_i}, \tag{5}$$

$$p_i(t + \varepsilon) = p_i(t + \varepsilon/2) - (\varepsilon/2)\frac{\partial U(q(t + \varepsilon))}{\partial q_i}. \tag{6}$$

2.2 Theoretical Properties of HMC

2.2.1 Acceptance Rate

To explore how the acceptance rate behaves in HMC, we must consider the quantity $\exp\left[U\left(q^n\right)+K\left(p^n\right)-U\left(q^*\right)-K\left(p^*\right)\right]$ which appears in the expression for α in the Metropolis-Hastings acceptance probability. Using the definition $H(q,p)=U(q)+K(p)$ we can write this quantity as:

$$\exp\left(U\left(q^n\right)+K\left(p^n\right)-U\left(q^*\right)-K\left(p^*\right)\right)=\exp\left[H\left(q^n,p^n\right)-H\left(q^*p^*\right)\right]. \tag{7}$$

Since the Hamiltonian is conserved under the Hamiltonian dynamics :

$$\begin{aligned}\frac{dH}{dt}&=\sum_i \frac{\partial H}{\partial p_i}\frac{dp_i}{dt}+\sum_i \frac{\partial H}{\partial q_i}\frac{\partial q_i}{dt} &(8)\\ &=-\sum_i \frac{\partial H}{\partial p_i}\frac{\partial H}{\partial q_i}+\sum_i \frac{\partial H}{\partial q_i}\frac{\partial H_i}{\partial p_i}=0, &(9)\end{aligned}$$

we find by using this in Eq.7 that if integration is exact, the acceptance rate is always 1.

Under the assumption that the Hamiltonian dynamics is discretised, conservation is there in the best case on average. This implies the acceptance rate will be less than 1.

2.2.2 Convergence to Target Distribution

Under the assumption that there is no numerical error, we want to prove that the Gibbs measure is invariant for the chain generated by the Hamiltonian dynamics.

This is equivalent to saying that the Gibbs measure π is the same before and after an evolution of t seconds from the Hamiltonian dynamics. To prove this we first introduce Hamiltonian dynamics operators φ, Φ acting respectively on the phase space and the Gibbs measure : $\varphi_t(q_s,p_s)=(q_{s+t},p_{s+t}); \Phi_t[\pi_s]=\pi_{t+s},\ \forall t\in\mathbb{R}$. The statement we want to prove can then be expressed as

$$\Phi_t[\pi_s](D)=\pi_{s+t}(D)\quad \forall D\in\mathcal{B}(\Omega),\forall s,t\in\mathbb{R}, \tag{10}$$

with Ω the phase space.

We therefore write the left hand side of the equation as

$$\begin{aligned}\Phi_t[\pi_s](D)&=\int_D \pi_{s+t}(q,p)\ dqdp &(11)\\ &=\int_{\varphi_{-t}(D)} \pi_s(q,p)\ dqdp &(12)\\ &=\pi_s(\varphi_{-t}(D)). &(13)\end{aligned}$$

The final result is obtained using the fact that volumes in phase space are preserved by the Hamiltonian dynamics (in conservative systems). This result is known as Liouville's theorem, but is mentioned as theorem 2.3 in [1].

This implies specifically that $q_k\sim\pi$ for all $k\in\mathbb{N}$ if $q_0\sim\pi$.

If the dynamics is discretised with Verlet's method, the volume in phase space is preserved up to a small error [2], which is why we use the Metropolis-Hastings acceptance rule.

3 Exploring a 2D example

3.1 Context

In this section we explore the performance of the considered algorithms on a 2D toy example[1]. The target distribution is taken as $f_1(q_1, q_2) = e^{-\alpha(q_1^2+q_2^2-0.25)^2}$, with $\alpha > 0$ a parameter. This target has polar symmetry, and has global maxima at a radius of $\sqrt{0.25} = 0.5$ away from the origin. It also admits a local minima at the origin. One can therefore picture it as some sort of centered disk with α controlling its thickness.

To make the study concise, we limit our analysis to two values of the parameter α. To make comparisons interesting between these two values, we take these as being $\alpha = 10$ and $\alpha = 1000$. Indeed we expect the $\alpha = 1000$ case to be more difficult to sample from than the $\alpha = 10$ case, due to the extent of localisation.

We decide to quantify performance throught the computation of a similarity measure based on the Jensen-Shannon divergence. This choice allows us to feed in a discretised version of f_1 (which we can normalise) and the empirical distribution of the samples generated by the algorithm, and return a similarity measure between the two. We decided to use a measure like this because it is still numerically tractable in 2D.

Note that in the following, unless specified otherwise, a burn-in period of 100 is used.

3.2 RWMC Solution

As seen previously, the RWMC algorithm only depends on the step size. We can therefore easily find the best RWMC sampler version by exploring the impact of the step size on performance.

The similarities associated to 3000 samples for the different step sizes are presented in Fig.1. The results are averaged over 15 and 100 chains for $\alpha = 10$ and $\alpha = 1000$ respectively. Looking

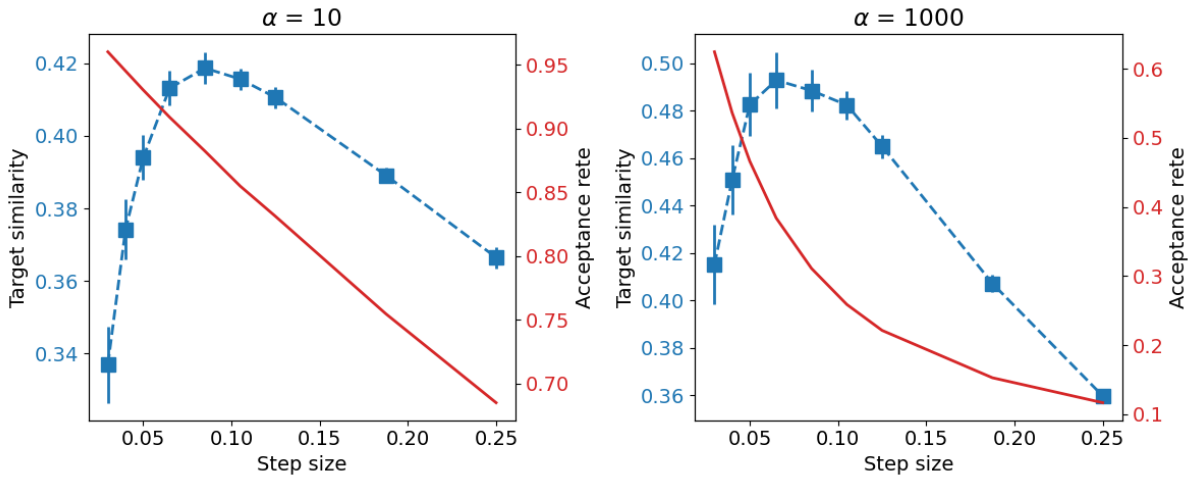


Fig 1: *Similarity as a function of RWMC step size for considered values of α.*

at the plots, it is clear they display a local maxima around 8.5×10^{-2} and 7×10^{-2} for $\alpha = 10$ and $\alpha = 1000$ respectively. Such a feature is expected as the step size is a tuning parameter that should be chosen to match the scale of the target distribution. More precisely, the peak is sharper on its right for the $\alpha = 1000$ case, which is consistent with the fact that the density is more localised (and therefore large steps are more heavily penalised).

The value of the similarity is in all cases < 0.5, which means that 3000 samples are too few to accurately estimate the target distribution. The value is higher in the $\alpha = 1000$ case,

[1]We call it a toy example because in principle one could sample from this distribution by simply writing it in polar coordinates and using the F^{-1} method.

which can at first look surprising. Indeed, this case is meant to be harder than the $\alpha = 10$ one. However, since the density is more localised for $\alpha = 1000$, there are fewer places where the estimate and the target can differ, which leads to a better similarity. It is therefore important to consider relative values of similarities for the different step sizes, rather than absolute ones.

Looking at the evolution of the acceptance rate, we see that it is of course monotonically deacreasing with step size. This means that the obtimal value corresponds with the best exploration-acceptance rate trade-off. The acceptance rate is higher and decreases slower for the $\alpha = 10$ case, which is consistent with the fact that the density is more spread out. The acceptance rate of the $\alpha = 1000$ case associated to the best step size is still around 0.4, which suggests that even this case is still quite easy to sample from (i.e. random proposals are "pretty good").

3.3 HMC Solution

Before exploring the impact of the different parameters of the HMC algorithm, we first present the potential energy landscape associated to the algorithm for this specific problem. The landscape is presented in figure Fig.2.

The only difference in the landscape for $\alpha = 1000$ w.r.t. $\alpha = 10$ is the scale of the potential. This means that the $\alpha = 1000$ will give rise to stronger potential forces, which translates the fact that the density is more localised.

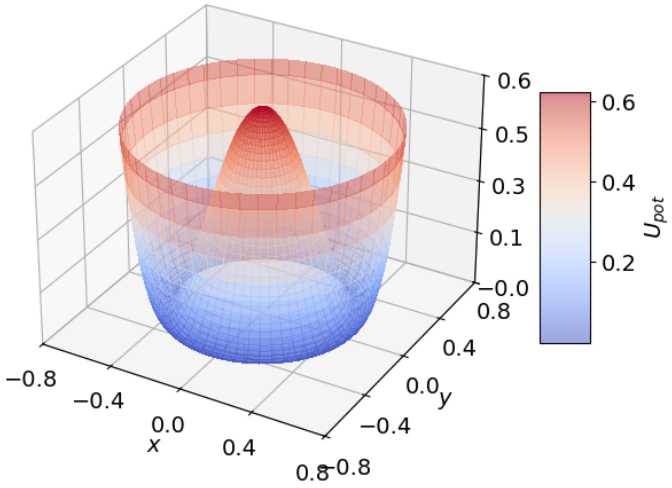


Fig 2: *Potential energy landscape associated to HMC algorithm for $\alpha = 10$. Version for $\alpha = 1000$ is identical, except the vertical scale is multiplied by a factor 100.*

This landscape gives us a good idea of how trajectories will look like. We just need to immagine trowing a bead into it with given initial momentum, and letting it slide down and up the "hills".

The first parameter we explore is the mass. We split this in terms of the mass scale and the mass anisotropy. We do not consider the impact of off-diagonal terms as this would make the report too long and was not suggested in the statement.

3.3.1 Impact of Mass Scale

The impact of mass scale is readily understood by thinking back at the formulas defining the algorithm. Firstly, we can interpret what it means for p to scale with m. Given that the potential does not depend on m, the Δp applied at each integration step are independent of m. This implies that it will take longer for the particle to react to the potential landscape. Physically, this is what we call the particle's inertia.

More surprisingly, we can see that altough the mass appears explicitely in the expression for kinetical energy due to $p \sim \mathcal{N}(0, M)$, we have $K(p) = \frac{1}{2}\sum_{i=1}^{d}\left(\frac{p_i}{\sqrt{m_i}}\right)^2 = 2\chi_d^2$. From this we deduce (using energy conservation) that mass does not affect the hights reached in the potential well (if we integrate sufficiently long). This makes sense in probability terms, since we want accessible domains to be the same no matter the value of m It does however affect the size of the steps that are taken given that $M^{-1}p \sim \mathcal{N}(0, M^{-1})$, i.e. it scales (up or down) the initial velocity.

Given we can correct for such changes by adapting the integration time and step, it is more

interesting to move to analysing how the algorith is affected by anisotropy of the mass, which is the goal of the next section.

3.3.2 Impact of Mass Anisotropy

The mass isotropy/anisotropy specifies relative velocities and inertias among the directions. Anisotropic mass therefore allows a particle to move quicker and react more quickly to forces in some directions, while doing so slower in others. We expect this to be useful to improve exploration for example when the potential does not have radial symmetry around a minimum for example (immagine the case of an ellipse).

In our case, the target distribution is radially symmetric around the origin, but not around its minima. We could therefore expect that better results could be obtained by playing with anysotropic masses. One should however take into account that the preculiar shape of our distribution makes it hard to be confident in this statement. Indeed, due to the specific shape of the potential, much of the momentum is actually efficiently converted between directions.

We tried gaining insight with help of Fig.3 and similar figures, in which the similarities associated to 2000 samples for both isotropic and anisotropic masses are represented. Note we limited this analysis to the $\alpha = 1000$ case since the simplicity of the $\alpha = 10$ makes it harder to interpret subtle differences in performance. Here we took $t = 0.21, dt = 0.01$, and averaged over 10 chains. Looking at the plot, we see that the similarity is not impacted by the anysotropy.

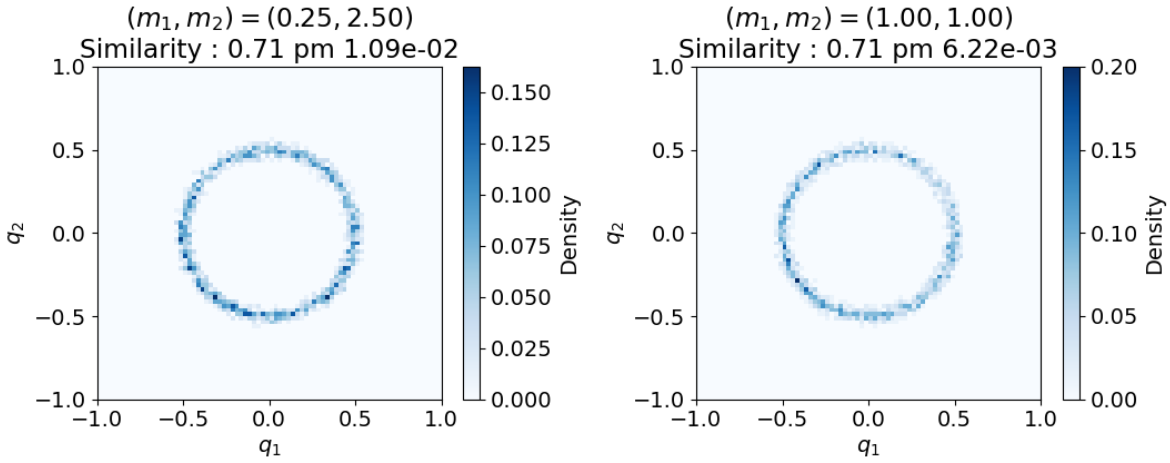


Fig 3: *Similarity for HMC samplers with asymmetric and symmetric masses for* $\alpha = 1000$ *when* $t = 0.21$ *and* $dt = 0.01$.

One can wonder if the result would change by making the anisotropy more severe. In this case however, it would be hard to disentangle the contributions since for really different masses, t and Δt would be of different quality.

3.3.3 Impact of Integration Time

The next parameter we explore is integration time. For this parameter, we expect the optimal value to be the one that allows the sampler to explore the whole space, without being excessively long (as it would slow down sampling). The similarities associated to 3000 samples for the different integration times are presented in figure Fig.4. Here we took $dt = 0.01$, and averaged over 10 chains. In both $\alpha = 10$ and $\alpha = 1000$ cases similarity increases as a function of t untill a plateau around $t = 9 \times 10^{-2}$ is reached. This weak dependence on α can be surprising at first given that the caracteristic time of radial oscillation clearly depends on α (throught the

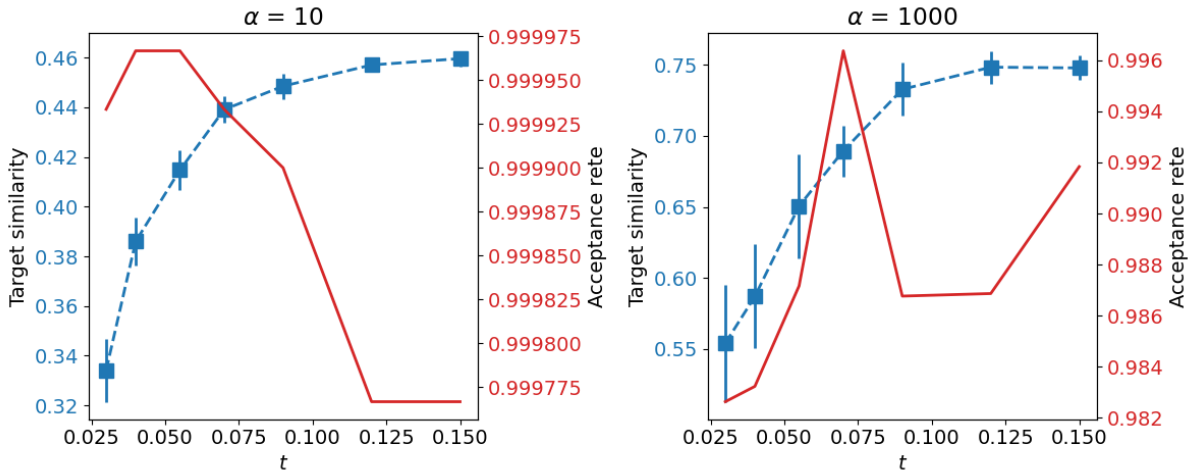


Fig 4: *Similarity as a function of HMC integration time for considered values of α with $dt = 0.01$.*

curvature of the potential). In both α cases however, we can imagine the bottle neck for two samples to be distant is actually in the direction ortogonal to the radial one. Indeed, in both cases the radius of the disk is larger than its width.

3.3.4 Impact of Δt

We now turn to analysing the impact of the time step. This parameter is important as it controls the accuracy of the integration. The goal is to find the largest time step that allows for accurate integration, as this will speed up the sampler while allowing it to accept proposals frequently.

The similarities associated to 3000 samples for the different time steps are presented in figure Fig.5. Here we took $t = 0.12$, and averaged over 10 and 5 chains for $\alpha = 10$ and $\alpha = 1000$ respectively. The first characteristic feature the two plots share is the monotonous behaviour of

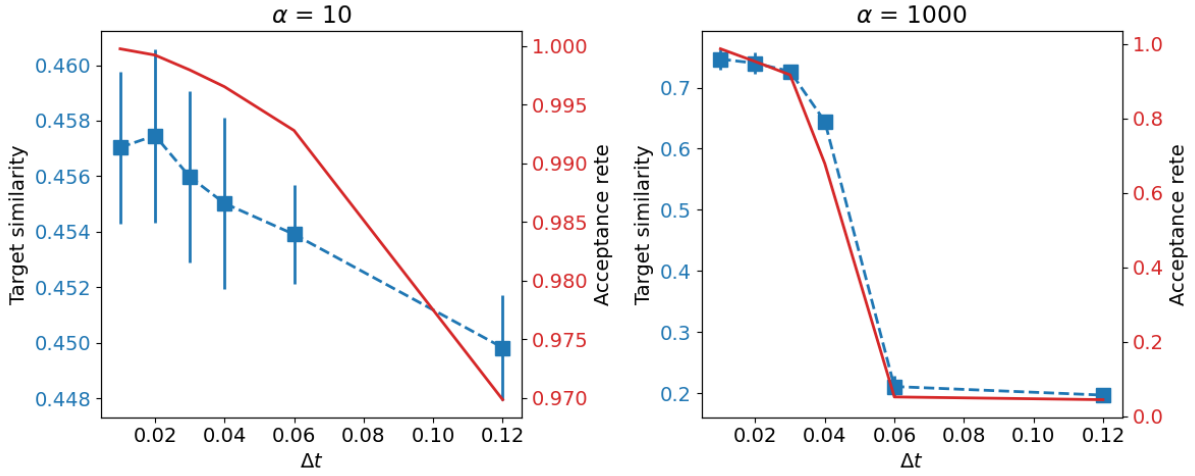


Fig 5: *Similarity as a function of HMC time step for considered values of α with $t = 0.12$.*

the acceptance rate. Indeed, as mentionned before, by making Δt grow, the simulation does not exactly leave the Hamiltonian invariant, and therefore makes the acceptance rate decrease. This in turn can affect the similarity, as fewer samples are obtained and therefore these represent the target distribution less precisely. This happens particularly noticepbly in the $\alpha = 1000$

case, where we reach a really low acceptance rate at $\Delta t = 6 \times 10^{-2}$. This is probably due to the radial characteristic time being shorter than in the $\alpha = 10$, and therefore requiring finer resolution. Indeed, for $\alpha = 10$ good acceptance rate is mantained in term of accpetance rate and target similarity even when a single timestep of size t is taken. This makes sense since the obtained algorithm is still some sort of upgraded version of RWMC where the general direction of the step is choosen smartly and RWMC already performed well over quite a broad domain of step sizes.

3.3.5 Comment on interractions between parameters

In the previous sections, the analysis of HMC's parameters was limited to a one-factor-at-a-time study. This was motivated by the non-exhaustive character of the report and the fact that meaningfull insights were associated to these longitudinal analyses.

In reality however, each parameter of course does not act in a vacum. For example, increasing mass, requires longer integration times for good space-exploration to occur, and allows for larger Δt to be used. This has practical implications in finding global optimal choice of parameters. Indeed, when grid search for the parameters is untractable (i.e. in high dimensions), these interractions between the parameters make it hard to find globally optimal values. Our intuition is that in these cases one can proceed in the following way to find "good enough" parameters "easily" :

1. estimate characteristic times for the oscillations from the expression for the potential (analytically or numerically if not tractable),

2. use assumptions on the potential to fix the structure for the mass (isotropic vs anisotropic) and select the locally optimal mass structure after having set t and Δt generously : t larger than optimal (guaranteeing good mixing) and Δt smaller than optimal (granting precise integration),

3. with the new mass matrix, select the locally optimal integration time (shortest one preserving good enough mixing),

4. finally select the locally optimal Δt (the biggest for which the acceptance rate is good enough).

We do not expect this approximate tuning of the parameters to be a problem given that alternatives like RWMC breakdown in high dimensions. A locally optimal selection of HMC parameters will probably still lead to a much more efficient sampler.

Following the procedure described above, the locally optimal HMC parameters for the $\alpha = 10$ case were found to be $m_1 = m_2 = 1$, $t = dt = 0.09$, and for the $\alpha = 1000$ case $m_1 = m_2 = 1$, $t = 0.09$, $dt = 0.03$. Refering to Fig.1, the optimal step size for the RWMC algorithm is around 8.5×10^{-2} when $\alpha = 10$ and 7.5×10^{-2} when $\alpha = 1000$. In the next section, these parameters will be used to compare the performance of the optimised HMC and RWMC algorithms.

3.4 RWMC and HMC Comparison

3.4.1 Target Similarity

To determine which algorithm is best to use for the studied 2D example, it is interesting to compare the results obtained with optimised versions of the RWMC and HMC algorithms as a function of the number of function evaluations. We do this in Fig.6. The results are averaged over 5 chains. In general, the same qualitative behaviour is observed for all algorithms : the similarity quickly increases at the start but then sees deminishing returns characteristic of concave functions. This is a characteristic inherited from Monte Carlo methods.

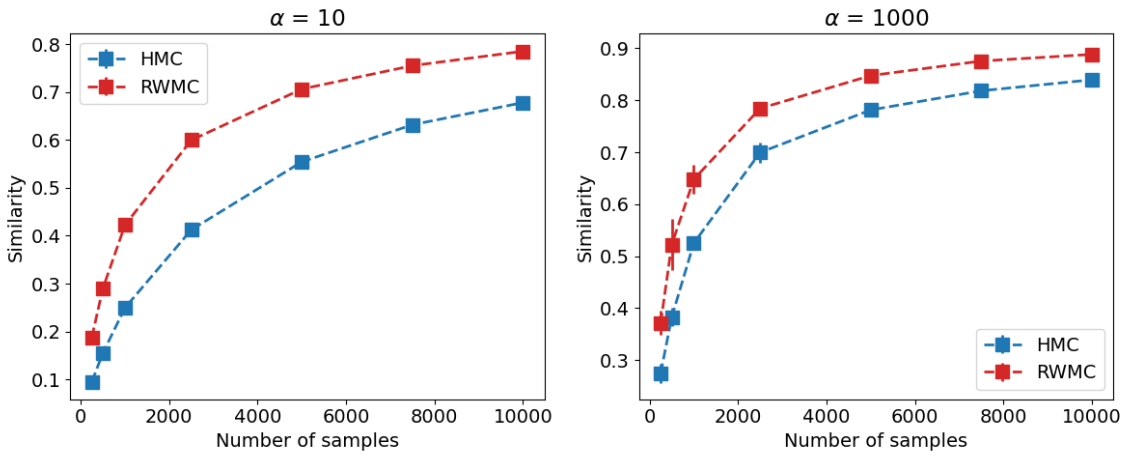


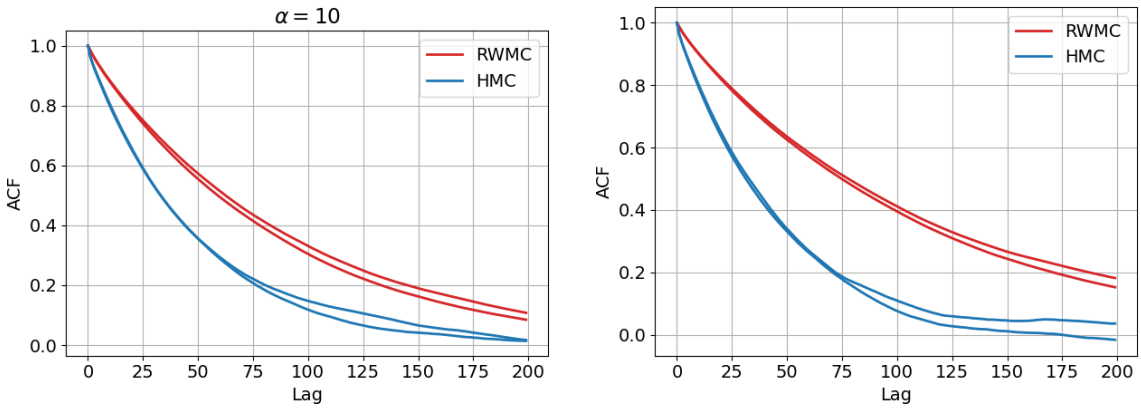
Fig 6: *Similarity as a function of the number of function evaluations for RWMC and HMC samplers and considered values of* α. $\delta = 8.5 \times 10^{-2}, m_1 = m_2 = 1, t = dt = 0.09$ *were used for* $\alpha = 10$ *while* $\delta = 7.5 \times 10^{-2}, m_1 = m_2 = 1, t = 0.09, \Delta t = 0.02$ *were used for the* $\alpha = 1000$ *one.*

In both cases RWMC outperforms HMC in terms of similarity. The difference is larger in the $\alpha = 10$ case, again, due to how simple the target function is. For $\alpha = 1000$, the computational overhead (of a factor 7) is for a big part compensated. We can easily imagine this trend continuing as a function of distribution complexity. We also expect it to accentuate when the dimension of the distribution is increased, due to the curse of dimensionality.

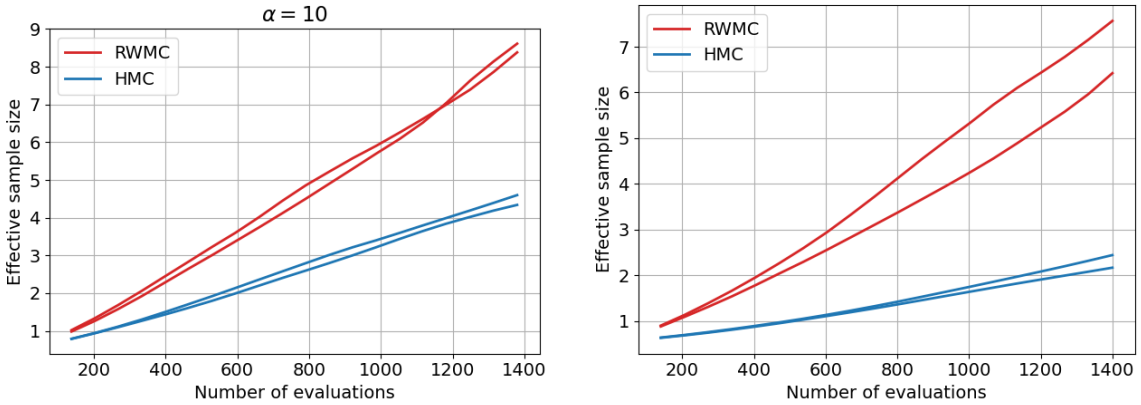
3.4.2 Effective Sample-Size

The main motivation behind HMC is to decrease the autocorrelation of the samples by making step proposals that are more distant, whilst keeping a high probability of acceptance. Indeed, as shown in 2.2.1, the theoretical acceptance rate of HMC (i.e. without time discretization) stays equal to 1, no matter how large the integration time is. This effect is expected to become important when the target distribution is difficult to sample from, and a random walk is unable to explore the space efficiently. This can be observed in the autocorrelation plots of Fig.7a. In both cases $\alpha = 10$ and $\alpha = 1000$, the autocorrelation of the HMC samples decreses faster than that of the RWMC samples. As expected, the effect is subtle for $\alpha = 10$, as the RWMC samples are already quite decorrelated, and more pronounced for $\alpha = 1000$.

On the other hand, HMC requires more evaluations of the target function to generate a sample, as it needs to compute the gradient of the potential energy twice at each step of Verlet's method, in addition to the evaluation of the proposal step. This computation time could be used to generate more samples with RWMC, which would increase the effective sample size (number of independent samples that would give the same variance as the autocorrelated samples) and therefore the similarity of the samples to the target distribution. To determine whether the decrease in autocorrelation is worth the increase in computation time, we present the effective sample size of the RWMC and HMC samples in Fig.7b as a function of the number of evaluations of the target function (and its gradient). Consistently with the similarity results presented in Fig.6, the effective sample size of the RWMC samples grows faster than that of the HMC samples, for both values of α. Therefore in this case, the RWMC algorithm is more efficient than the HMC algorithm.



(a)



(b)

Fig 7: *Autocorrelation plots and effective sample-size for RWMC and HMC samplers and considered values of α. Both q_1 and q_2 are shown on each plot, and the results are averaged over 5 chains.*

4 US Birthweight Data

In this section, we use HMC to perform a logistic regression on the US Birthweight data [3]. The data consists of $n = 189$ observations of 8 variables: the mother's age, weight, race (white, African-American, other), smoking status, hypertension status, presence of uterine irritability, the number of physician visits during the first trimester, the existence of past premature births, and the low birth weight status of the child. We are interested in predicting whether a child will have low birth weight based on the following covariates (refering to the mother):

- Numerical covariates:
  - Age
  - Weigh
- Boolean covariates:
  - Whether the mother is african american
  - Other race
  - Smoking status
  - Existence of past premature births

- Hypertension status
- Uterine irritability status
- Whether there was at least one physician visit during the first trimester
- Whether there were additional physician visits during the first trimester

The numerical covariates are normalized, such that they have mean 0 and standard deviation 1. The boolean covariates are encoded as -1 or 1. The target variable is the low birth weight status of the child, which is a boolean variable encoded as 0 or 1. We use a logistic regression model to predict the target variable based on the covariates with a Gaussian prior on the regression coefficients and intercept $\mathcal{N}(0, \text{diag}(\sigma_0^2, \ldots, \sigma_p^2))$, with $p = 10$ the number of covariates. We choose to take $\sigma_i = 1$ for all i, as the target variable and all covariates are of the same order of magnitude and there is no reason to believe a priori that some covariates are significantly more important than others. We use the following log-likelihood function, with $y \in \mathbb{R}^n$ the target variable, $X \in \mathbb{R}^{n\times(p+1)}$ the matrix of covariates, and $q \in \mathbb{R}^{p+1}$ the vector of regression coefficients and intercept:

$$\log f(q|y, X, \{\sigma_0^2, \ldots, \sigma_p^2\}) = q^T X^T (y - \mathbf{1}_n) - \mathbf{1}_n^T [\log(1 + \exp(-x_i^T q))]_{n\times 1} - \frac{1}{2}\sum_{i=0}^{p} \frac{q_i^2}{\sigma_i^2}. \quad (14)$$

This distribution will be used as an unormalized version of the target distribution $\pi(q)$ to perform HMC.

4.1 HMC Sampling

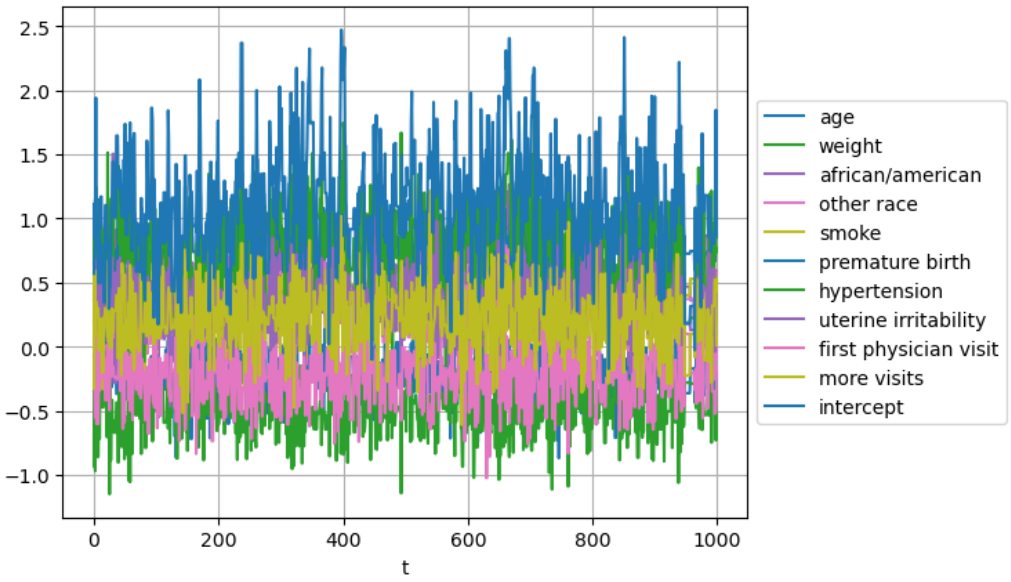


Fig 8: *Traceplot of the regression coefficients and intercept for the US Birthweight data, obtained using HMC with parameters* $dt = 0.02$, $T = 0.6$, *and* $m = 0.05$.

Fig.8 shows the traceplot of the HMC sampling from Eq.14 for each covariate of the model. We observe that the chains converge to the stationary distribution after a burn-in period of less than 100 steps. The histograms of the regression coefficients and intercept are shown in Fig.9. All covariates look to be approximately gaussian distributed, with the past premature births and hypertension status being the most influential factors.

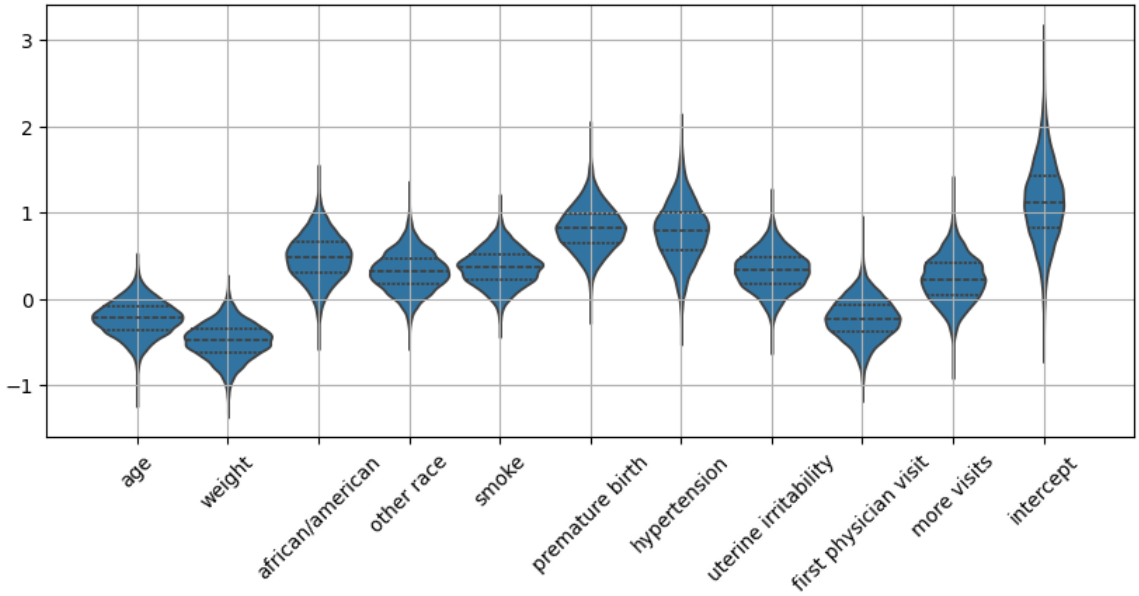


Fig 9: *Violin plot of the regression coefficients and intercept for the US Birthweight data, obtained using HMC with parameters* $dt = 0.02$, $t = 0.08$, *and* $m = 0.02$. *The histograms are taken over 10000 samples of the chains, after a burn-in period of 200 steps.*

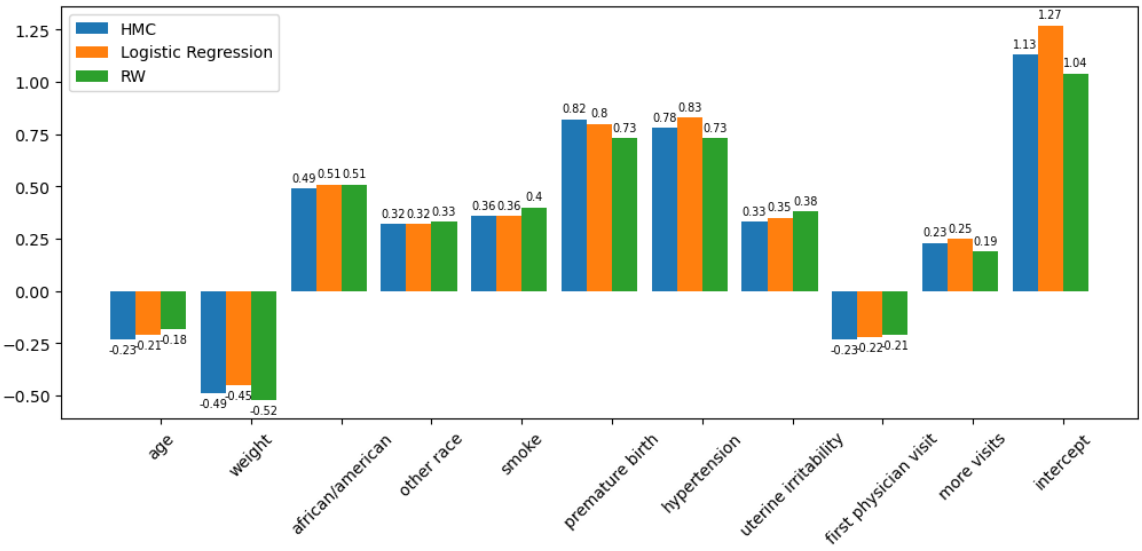


Fig 10: *Comparison of the regression coefficients and intercept for the US Birthweight data, obtained using HMC and RWMC.*

4.2 Coefficients Comparison

We consider the task of estimating the expectation of the regression coefficients and intercept from HMC sampling of the posterior distribution, and we compare it to the same results obtained using RWMC. To evaluate the accuracy of the estimates, we use the results of a standard logistic regression procedure performed using limited-memory BFGS optimization (this was done using the `LogisticRegression` function from the `scikit-learn` library). This optimization process aims to find the argmax of the log-likelihood function, not the mean. However, it is shown on Fig.9 that the distributions of the coefficients are approximately gaussian, in which case the mean and the argmax are the same. The results are shown in Fig.10. The mean over the HMC and RWMC distributions are taken over chains lengths matching the number of evaluations of the log-likelihood function and its gradient (500 steps

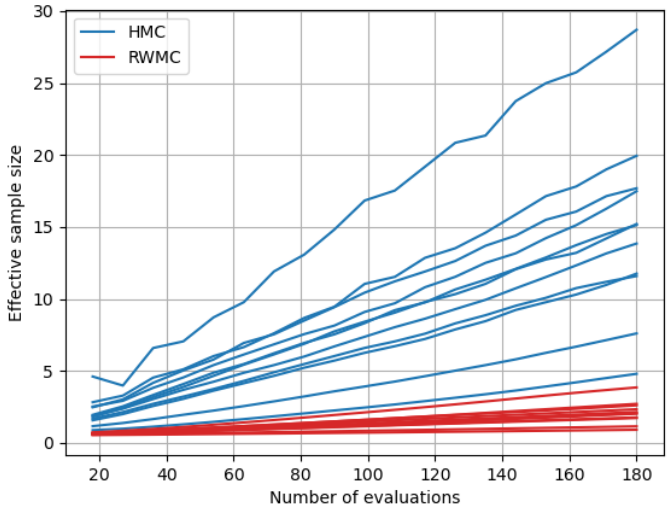


Fig 11: *Effective sample size for the covariates of the regression model Eq.14, sampled using HMC and RWMC. Each covariate is shown separately, and the results are averaged over 10 chains.*

for HMC and 4500 steps for RWMC). Both the HMC and RWMC estimates appear to be close to the BFGS estimates, showing that both algorithms are able to accurately estimate the regression coefficients and intercept. Nonetheless, we observe that the HMC estimates are overall slightly closer to the BFGS than the RWMC ones, hence HMC outperforms RWMC in this case. This is consistent with the results obtained in section 3.4.1, where HMC was shown to perform comparatively better in terms of target similarity for more complex target distributions.

To investigate this observation further, we compute the effective sample size of the regression coefficients and intercept for both HMC and RWMC. The results are shown in Fig.11. We observe that the effective sample size of the HMC samples consistently grow at a faster rate than that of the RWMC samples. Therefore in this setting, the high dimensionality of the target distribution makes it difficult for RWMC to explore the state space efficiently. This is also illustrated by the low acceptance rate of the RWMC sampler, which is around 0.4, compared to 0.7 for the HMC sampler. The additionnal computational cost of the HMC sampler is in this case overcome by its quicker autocorrelation decay, making it more efficient than RWMC.

5 Conclusion

In this report we provided insight on how to choose between RWMC and HMC based on problem characteristics.

We first presented an overview of the theoretical foundations of HMC, and proved important properties for it. Then, we explored and interpreted how parameters, such as step size for RWMC and mass for HMC influence performance. To systematically assess and compare their efficiency, we used among others a similarity metric and effective sample size. Specifically, we considered two case studies: a 2D toy example with tunable complexity and a higher-dimensional 10D birthweight dataset.

RWMC was found to perform much better in the 2D toy example. The difference in efficiency however largely decreased when the complexity parameter was increased. For the 10D birthweight dataset, HMC vastly outperformed RWMC. This suggests that RWMC performs better in low-complexity, low-dimensional scenarios, while HMC proves superior as complexity and dimensionality increase.

Acknowledgements

Aude: I used the autocompletion feature of Github Copilot to gain time, both when coding and writing the report. I only kept the suggestions that were finishing my sentences or lines of code in the way I intended them to be.

Tara: I use Github Copilot for coding purposes, as it can gain me some time by auto-completing lines. I however never keep lines I don't understand or find not relevant. I sometimes used Chat GPT to brainstorm ideas related to some problems I faced and did not have the right resources to tackle. In these cases I still kept a critical mindset and dug deeper in the directions that seemed promising. I also used Chat GPT to help with certain wordings in the introduction.

References

[1] N. Bou-Rabee and J. M. Sanz-Serna, 'Geometric integrators and the Hamiltonian Monte Carlo method', Acta Numerica, vol. 27, pp. 113-206, May 2018, doi: 10.1017/S0962492917000101.
[2] 'Leapfrog integration - Wikiwand'. Accessed: Dec. 14, 2024. [Online]. Available: https://www.wikiwand.com/en/articles/Leapfrog%20integration
[3] 'Introduction to the Logistic Regression Model', in Applied Logistic Regression, John Wiley & Sons, Ltd, 2013, pp. 1-33. doi: 10.1002/9781118548387.ch1.
[4] 'Answer to "How to present a Python code snippet efficiently in LaTeX?"', TeX - LaTeX Stack Exchange. Accessed: Dec. 23, 2024. [Online]. Available: https://tex.stackexchange.com/a/475828

A Commented Code Snippet

As asked in the project rules, we attach here a well commented extract of the code. We do this by using a command that we found on a Stack exchange post [4]. We decided to include the code defining the classes for our samplers as we felt if reflected best the essence of the project.

```
import numpy as np
from abc import ABC, abstractmethod

class MCMCSampler(ABC):
  '''
  Abstract base class for MCMC Samplers. Provides a common interface
                                for different MCMC algorithms.
  '''
  def __init__(self, seed, initial_condition, unnorm_logdensity,
                                burn_in):
    self.seed = seed
    self.initial_condition = np.array(initial_condition)
    self.unnorm_logdensity = unnorm_logdensity
    # we use log densities here to avoid underflow

    self.burn_in = burn_in
    self.rng = np.random.default_rng(seed)
    self.state = self.initial_condition.copy() # current state
    self.n_evaluations = 0 # number of function evaluations

  def reset_state(self, only_state=False):
    '''
    Resets the RNG and state to their initial values.

    Parameters:
    only_state (bool): If True, only resets the state, not the RNG.
                                This is useful for creating
                                independent chains starting with the
```

```
                                same initial condition.
    '''
    self.state = self.initial_condition.copy()
    if not only_state:
      self.rng = np.random.default_rng(self.seed)

  def MH_acceptance_rule(self, current_state, proposed_state,
                                current_logdensity,
                                proposed_logdensity):
    '''
    Accepts or rejects a proposed state based on the Metropolis-
                                Hastings acceptance rule.

    Returns:
    np.ndarray: The new state.
    bool: Whether the proposed state was accepted.
    '''
    prob = np.exp( proposed_logdensity - current_logdensity )
    acceptance_prob = min(1, prob)
    if self.rng.uniform() < acceptance_prob:
      return proposed_state, True
    else:
      return current_state, False

  def sample(self, n_chains, n_samples, return_info=False):
    '''
    Generates samples using the Metropolis-Hastings algorithm.

    Parameters:
    n_chains (int): Number of independent chains.
    n_samples (int): Number of samples per chain.
    return_info (bool): If True, returns acceptance_rate and
                                n_evaluations associated to the
                                sampling process.

    Returns:
    np.ndarray: Samples of shape (n_chains, n_samples, dim).
    float: Acceptance rate.
    int: Number of function evaluations.
    '''
    assert self.burn_in < n_samples, 'Burn-in period must be less than
                                 number of samples.'

    dim = len(self.initial_condition)
    samples = np.zeros((n_chains, n_samples, dim))
    acceptance_rate = 0

    # This could be made more efficient, especially for RWMC, but
    # will only do this if runtime is too long.
    for chain_idx in range(n_chains):
      self.reset_state(only_state=True) # make chains start at the
                                same place
      current_state = self.state.copy()
      for sample_idx in range(n_samples):
        current_state, accepted = self.proposal_step(current_state)
        acceptance_rate += float(accepted)
        samples[chain_idx, sample_idx, :] = current_state

    acceptance_rate /= (n_chains * n_samples)
    if return_info:
```

```
        return samples[:, self.burn_in:, :], acceptance_rate, self.
                                    n_evaluations

      print(f'Acceptance Rate: {acceptance_rate:.2f}')
      print(f'Number of function evaluations: {self.n_evaluations}')
      return samples[:, self.burn_in:, :]

    @abstractmethod
    def proposal_step(self, current_state):
      '''
      Generate a proposed state based on the current state, and accept
                                    it according to associated MH rule.
      Must be implemented in subclasses.
      '''
      pass

  class RandomWalkMCMC(MCMCSampler):
    '''
    Random Walk MCMC sampler.
    '''
    def __init__(self, seed, initial_condition, unnorm_logdensity,
                                    burn_in, step_size):
      super().__init__(seed, initial_condition, unnorm_logdensity,
                                    burn_in)
      # to allow for step_size to be a scalar or an array
      try:
        iterator = iter(step_size)
        self.step_size = np.array(step_size)
      except TypeError:
        self.step_size = step_size

    def proposal_step(self, current_state):
      '''
      Proposes a new state using a random walk.
      '''
      proposed_state = current_state + self.step_size * self.rng.normal(
                                    0, 1, size=current_state.shape)
      current_logdensity = self.unnorm_logdensity(current_state)
      proposed_logdensity = self.unnorm_logdensity(proposed_state)
      self.n_evaluations += 1
      # we only add one since in principle could evaluate density of
                                    energy difference

      return self.MH_acceptance_rule(current_state, proposed_state,
                                    current_logdensity,
                                    proposed_logdensity)

  class HamiltonianMCMC(MCMCSampler):
    '''
    Hamiltonian Monte Carlo sampler.
    '''
    def __init__(self, seed, initial_condition, unnorm_logdensity,
                                    burn_in, unnorm_logdensity_grad, mass,
                                     leapfrog_time, dt):
      super().__init__(seed, initial_condition, unnorm_logdensity,
                                    burn_in)
      assert (np.array(mass) != 0).all(), 'Mass must be non-zero.'
      # to allow for mass to be a scalar or an array
      try:
        iterator = iter(mass)
```

```
            self.mass = np.array(mass)
        except TypeError:
            self.mass = mass
        self.dt = dt
        self.num_steps = int(leapfrog_time / dt)
        self.unnorm_logdensity_grad = unnorm_logdensity_grad

    def proposal_step(self, current_state):
        '''
        Proposes a new state using Hamiltonian dynamics.
        '''
        momentum = np.sqrt(self.mass) * self.rng.normal(0, 1, size=
                                    current_state.shape)
        proposed_state = current_state.copy()
        proposed_momentum = momentum.copy()

        # integrate Hamiltonian dynamics using leapfrog
        for _ in range(self.num_steps):
            # we use : grad U = - grad log p
            proposed_momentum += 0.5 * self.dt * self.unnorm_logdensity_grad
                                    (proposed_state)
            proposed_state += self.dt * proposed_momentum / self.mass
            proposed_momentum += 0.5 * self.dt * self.unnorm_logdensity_grad
                                    (proposed_state)
            self.n_evaluations += 2

        current_energy = -self.unnorm_logdensity(current_state) + 0.5 * np
                                    .dot(momentum/self.mass,momentum)
        proposed_energy = -self.unnorm_logdensity(proposed_state) + 0.5 *
                                    np.dot(proposed_momentum/self.mass,
                                    proposed_momentum)
        self.n_evaluations += 1
        # we only add one since in principle could evaluate density of
                                    energy difference

        return self.MH_acceptance_rule(current_state, proposed_state, -
                                    current_energy, -proposed_energy)
```

B Rejection Sampling Attempt

Before resulting to doing the (f) part of the statement with RWMC, we thought that rejection sampling could be a good idea.

We therefore wanted to find a function $g(q)$ and a constant C such that the following inequality holds for all q:

$$\tilde{f}(q) = e^{q^T X^T (y - 1_n)} e^{-1_n^T \log[1+\exp(-x_i^T q)]_{n\times 1}} e^{-\frac{1}{2} q^T \Sigma^{-1} q} \leq C g(q), \tag{15}$$

where we have denoted $\Sigma = \text{Diag}(\sigma_1^2, ..., \sigma_p^2)$.

The first $g(q)$ we used relied on the bound $1_n^T \log[1 + \exp(-x_i^T q)]_{n\times 1} > 0$. The associated math looked good, as the only manipulation needed was the to complete the square at the exponent of the remaining exponentials, leading to a multivatiate gaussian $g(q)$. The implementation however did not run due to the bound beeing too loose (we would have had to wait an extremely long time to get samples).

We then tought of using the much tighter bound

$$\log[1 + \exp(-x_i^T q)] \leq \log(2) - x_i^T q. \tag{16}$$

This bound however only works for $x_i^T q \leq 0$, which is not the case for the data we have.

It is at this point we finally abandonned the idea of using rejection sampling, after having checked with the TA that the problem was indeed hard to solve with this method.