# Hamiltonian Monte Carlo

Stochastic Simulations, 2022-2023


| Student Name | Student Number |
|---|---|
| Maxence Hofer | 289274 |
| Giacomo Mossinelli | 359961 |


## 1 Introduction

Hamiltonian Monte Carlo (HMC) is a powerful tool for Bayesian computation, which offers a good computational efficiency. This method, inspired by classical mechanics, and originated for the study of lattice models of quantum field theory, soon became well known in data science. One of the key aspects of this algorithm, is given by the possibility of generating moves that can be accepted with very high probability. This acceptance probability can be brought close to 1 by implementing sufficiently accurate integration methods. In this project, we analyze this method by focusing on its theoretical framework and comparing it with other MCMC methods.

## 2 Hamiltonian conservation

Given the position vector $\vec{q} = (q_1, ..., q_d)$ and the momentum vector $\vec{p} = (p_1, ..., p_d)$, the Hamiltonian dynamics is described by:

$$\frac{d\vec{q_i}}{dt} = \frac{\partial H}{\partial \vec{p_i}} \tag{1}$$

$$\frac{d\vec{p_i}}{dt} = -\frac{\partial H}{\partial \vec{q_i}} \tag{2}$$

for $i = 1, ..., d$, where $H(\vec{q}, \vec{p}) = U(\vec{q}) + K(\vec{p})$, the Hamiltonian in the phase space, denotes the total energy of the system. In vector notation, it can be written also as:

$$\frac{d}{dt}\begin{bmatrix}\vec{q}\\ \vec{p}\end{bmatrix} = J^{-1}\nabla H(\vec{q}, \vec{p}) \tag{3}$$

where $\nabla H = [\frac{\partial H}{\partial q_1}, ..., \frac{\partial H}{\partial q_d}, \frac{\partial H}{\partial p_1}, ..., \frac{\partial H}{\partial p_d}]^T$ and $J = \begin{bmatrix} 0_{dxd} & -I_{dxd} \\ I_{dxd} & 0_{dxd}\end{bmatrix}$. We can then define the trajectory $\varphi_t : \mathbb{R}^D \to \mathbb{R}^D$, where $D = 2d$, as the map which associates at the initial value $\begin{bmatrix}q(\vec{0})\\ p(\vec{0})\end{bmatrix}$ the correspondent value at time $t$ of the solution of the system of equations 1 and 2, i.e.:

$$\begin{bmatrix}q(\vec{t})\\ p(\vec{t})\end{bmatrix} = \varphi_t(\begin{bmatrix}q(\vec{0})\\ p(\vec{0})\end{bmatrix}) \tag{4}$$

Since $J^{-1}$ is skew-symmetric, we can say that, if we consider $(q(\vec{t}), p(\vec{t}))$ solution of 3, then:

$$\frac{d}{dt}H(q(\vec{t}), p(\vec{t})) = \nabla H(q(\vec{t}), p(\vec{t}))^T J^{-1}\nabla H(q(\vec{t}), p(\vec{t})) = 0 \tag{5}$$

This result defines the energy preservation of the Hamiltonian system, which allows us to define the following:

**Proposition 1.** *The value of the Hamiltonian function is conserved on the trajectory of the correspondent Hamiltonian system, i.e.* $H \circ \varphi_t = H \quad \forall t \in \mathbb{R}$

Another characteristic property of an Hamiltonian system is the symplecticness, which implies that the oriented volume is conserved in the dynamics [1]. From these properties, it is possible to state that a probability measure with density of the form:

$$\mu = \frac{1}{Z} \exp(-\beta H(\vec{q}, \vec{p})) \tag{6}$$

(where $\beta > 0$) is preserved over the trajectory $\varphi_t$. Here, $Z$ is the normalization constant obtained with the integration $Z = \int_{\mathbb{R}^{\mathbb{D}}} \exp(-\beta H(\vec{q}, \vec{p}))dqdp < \infty$. We can see this result as a consequence of the conservation properties since the component $dqdp$ is preserved as a volume element and $\exp(-\beta H(q, p))$ because it is an energy component.

## 3 Hamiltonian Monte Carlo Algorithm

Given the Hamiltonian Monte Carlo algorithm, two types of situations can be distinguished.
First, let us consider the case in which the system does not require any kind of numerical discretization. In this case, the algorithm is significantly simplified, since the acceptance rate $\alpha = \min[1, \exp(-U(\vec{q^*}) + U(\vec{q^n}) - K(\vec{p^*}) + K(\vec{p^n}))]$ is identically equal to one. More precisely, since the result given in Section 2 can be seen as a mathematical expression of the principle of conservation of energy, we can state that $-U(\vec{q^*}) + U(\vec{q^n}) - K(\vec{p^*}) + K(\vec{p^n}) = 0$ and hence the result follows directly.
This fact changes when one considers the case where the system is subject to a numerical approximation. In fact, since in most cases the exact solution $\varphi_t$ is not available, it is necessary to introduce the approximation $\Psi$ but, since it is difficult to conserve energy and the volume in phase state, $H$ can be only approximately invariant [4]. As a consequence, we are introducing an approximation error that cannot be ignored and the acceptance rate becomes:

$$\alpha(\vec{q}, \vec{p}) = \min[1, \exp(-\Delta H(\vec{q}, \vec{p}))|det\Psi'(\vec{q}, \vec{p})|] \tag{7}$$

with $\Delta H = H(\Psi(\vec{q}, \vec{p})) - H(\vec{q}, \vec{p})$. There are many methods that can be used to describe the evolution of the Hamiltonian system, such as the simple Euler method, but the Verlet method is one of the most widely used in this context which, in our case, is:

$$p_i(t + \frac{\epsilon}{2}) = p_i(t) - \frac{\epsilon}{2} \frac{\partial U(q(t))}{\partial q_i} \tag{8}$$

$$q_i(t + \epsilon) = q_i(t) + \epsilon \frac{p_i(t + \frac{\epsilon}{2})}{m_i} \tag{9}$$

$$p_i(t + \epsilon) = p_i(t + \frac{\epsilon}{2}) - \frac{\epsilon}{2} \frac{\partial U(q(t + \epsilon))}{\partial q_i} \tag{10}$$

This method presents many advantages. The global error of the method has order 2 (unlike the aforementioned Euler method, of order 1) [1]. Furthermore, being a symplectic integrator, it is volume preserving and, consequently, it allows a simpler expression to define $\alpha$, not involving the determinant of the trajectory approximation ( $\alpha(\vec{q}, \vec{p}) = \min[1, \exp(-\Delta H(\vec{q}, \vec{p}))]$ ). However, the transition from the exact form to the numerical form implies the loss of the principle of conservation of energy and, consequently, does not allow one to state that $\alpha$ is identically equal to 1.

## 4 Exact time integration

In this section, in order not to have excessively heavy writing, we have avoided putting the arrow to indicate that $q$ and $p$ are a vector. Given the potential energy $U(q) = -\log \pi(q)$ and the kinetic energy $K(p) = \sum_{i=1}^{d} \frac{p_i^2}{2m_i}$, we consider the Gibbs-distribution as:

$$G(q,p) = \frac{1}{\tilde{Z}} \exp(U(q)) \frac{1}{\Pi_{i=1}^{d} \sqrt{2\pi m_i}} \exp(-K(p)) \tag{11}$$

where q is the variable of interest, while p is introduced only to allow Hamiltonian dynamics to operate. In fact, Hamiltonian Monte Carlo uses the Hamiltonian dynamical system to construct the Markov Chain Monte Carlo algorithm on a given invariant density $\pi(q)$.
If we consider a case where the time integration is being carried-out exactly, it is possible to show that equation 11 is invariant [1]. To do so, it is necessary to introduce the reversibility with respect to the momentum flip involution of the Hamiltonian system, which is related to the fact that $H(q,p)$ is even w.r.t. $p$. This property, which can also be seen as the time-reversibility of mechanics, states that, if $(q^0, p^0)$ is the initial state of the system and $(q(T), p(T))$ is the final state after a timespam $T$, then $(q(T), -p(T))$ evolves in the same amount of time to the state $(q^0, -p^0)$.
Following the Hamiltonian Monte Carlo algorithm, we can see that $p^0 \sim \mathcal{N}(0, M)$, which obviously preserves Gibbs distribution. Then, we let the system evolve deterministically for a fixed time $T$, according to Hamilton's system (1,2). This evolution defines a mapping $(q^0, p^0) \to (q^*, p^*) = (q(T), p(T))$ which obviously implies that $\alpha = 1$. Since this process is deterministic, we can also say that $g(q^*, p^*, q^0, p^0) = P[(q(T), p(T)) = (q^*, p^*) | (q(0), p(0)) = (q^0, p^0)] = \delta((q(T), p(T)) - (q^*, p^*))$. The transition probability for $q$ can then be defined as:

$$P[q(T) = q^* | q(0) = q^0] = \int P[p(0) = p^0] g(q^*, p^*, q^0, p^0) \alpha(q^*, p^*)$$

and we would like to prove that it is in detailed balance [2].
Since

$$\exp(-H(q,p)) \min(1, \exp(-\Delta H)) = \exp(-H(q^*, p^*)) \min(\exp(\Delta H), 1)$$

and since the Gibbs distribution $G(q,p)$ has the same form as equation 6, and so depends on $H(q,p)$, which is reversible, we obtain:

$$G(q^0, p^0) \alpha(q^*, p^*) = G(q^*, p^*) \alpha'(q^0, p^0) = G(q^*, -p^*) \alpha'(q^0, -p^0)$$

where $\alpha'(q^0, -p^0)$ denotes the acceptance in the opposite direction. Finally, using the fact that $g(q^*, p^*, q^0, p^0) = g(q^0, -p^0, q^*, -p^*)$ because of reversibility, then we have

$$\int G(q^0, p^0) g(q^*, p^*, q^0, p^0) \alpha(q^*, p^*) = \int G(q^*, -p^*) g(q^0, -p^0, q^*, -p^*) \alpha'(q^0, -p^0) \tag{12}$$

which proves the detailed balanceness. In the case where both the energy and the volume are conserved as in paragraph 2, then $\alpha = 1$ always. This proves that the distribution will be correct for any configuration of $q$ and that the Gibbs distribution is conserved.
The same is not true when we use the Verlet method, since in this case we introduce an error in the energy conservation.

## 5 Implementation of the algorithm

Markov Chain Monte Carlo algorithm are used in order to sample from a unnormalized distribution $\pi$. The idea is to create an ergodic Markov Chain with invariant distribution $\pi$ [5]. In

the following part we will compare the performance of two different algorithms to sample from the unnormalized distribution:

$$f_1(q_1, q_2) = e^{-\alpha(q_1^2+q_2^2-1/4)^2} \tag{13}$$

More precisely, in section 6 we will implement a Hamiltonian Monte Carlo method (HMC) using a Verlet's scheme to simulate the next element of the chain and a Random Walk Metropolis-Hastings (RWMH) algorithm.

**Random Walk Metropolis-Hastings** is an Markov chain Monte Carlo algorithm using a Normal distribution to simulate the next point $\vec{q^*} \sim \mathcal{N}(\vec{q_n}, \sigma^2 I_2)$, with $\vec{q_n}$ the last element of the chain and $\sigma$ a parameter to adjust. This point $\vec{q^*}$ will then be accepted with a probability $\beta$ defined as

$$\beta = min\left\{\frac{f(\vec{q^*})}{f(\vec{q_n})}, 1\right\} \tag{14}$$

If the next element is accepted, $\vec{q_{n+1}} = \vec{q^*}$, otherwise $\vec{q_{n+1}} = \vec{q_n}$.

In order to analyse the Markov chains produced by both algorithms, we need to define two usefull functions.

First, the autocovariance of a Markov chain is a measure of the degree to which the current state of the chain is correlated with its past states. It is defined as:

$$c(k) = \mathrm{Cov}(X_j, X_{j+k}) \tag{15}$$

where $X_t$ is the state of the Markov chain at time $t$, and $k$ is the lag at which the autocovariance is being computed.

This function can be used to assess the persistence of the chain over time. If the autocovariance decays quickly as the lag increases, then the chain is said to have low persistence, which means that the current state is largely independent of the past states of the chain. On the other hand, if it remains high for large lags, then the chain is said to have high persistence, which means that the current state is strongly influenced by the past states of the chain.

The autocovariance function is often used in the study of Markov chain, and it is closely related to the autocorrelation function, which is another measure of the correlation between the current state of the chain and its past states. The autocorrelation function is defined as:

$$\rho(k) = \frac{\mathrm{Cov}(X_j, X_{j+k})}{\mathrm{Var}(X_j)} \tag{16}$$

where $\mathrm{Var}(X_t)$ is the variance of the state at time $t$.

The effective sample size (ESS) of a Markov chain represents the size of an equivalent independent sample that would lead to the same variance of the estimator. This is defined as:

$$\mathrm{ESS} = N\frac{c(0)}{\sigma^2_{mcmc}} \tag{17}$$

Where N is the length of the chain, $c(0)$ the variance of $X_j$ and $\sigma^2_{mcmc}$ is called time-average variance constant and is given by:

$$\sigma^2_{mcmc} = \mathrm{Var}(X_j) + 2\sum_{l=1}^{\infty}\mathrm{Cov}(X_j, X_{j+l}) < \infty \tag{18}$$

In our case, we have calculated an estimator of the covariance as:

$$\hat{c}(k) = \frac{1}{N-k-1}\sum_{j=B+1}^{N+B-k}(X_j - \hat{\mu}^{mcmc}_{N,B})(X_{j+k} - \hat{\mu}^{mcmc}_{N,B}) \tag{19}$$

with $B$ a burn-in lag during which the chain reaches a stationary state, and $\mu_{N,B}^{mcmc}$ the average of the chain after the burn-in lag. the correlation as

$$\hat{\rho}(k) = \frac{\hat{c}(k)}{\hat{c}(0)} \tag{20}$$

and the time-average variance constant

$$\hat{\sigma}_M^2 = \hat{c}(0) + 2\sum_{k=1}^{M} \hat{c}(k) \tag{21}$$

where we only use the first terms of the sum, because of the instability of the last terms. M is defined as $M = 2min\{k : \hat{c}(2k) + \hat{c}(2k+1) < 0\}$.
In this experiment parameters of both algorithm are the following:

1. **HMC**:
   - $\epsilon$: the step size in the Verlet's scheme
   - $T$: the final time of integration in the Verlet's scheme
   - $\vec{m}$ the vector of the mass used in the scheme
2. **RWMH**:
   - $\sigma^2$ the variance of the Normal distribution used to compute $\vec{q^*}$
3. **Both algorithms**:
   - $\alpha$: the constant in the distribution, that will be studied for $\alpha = 10$ and $\alpha = 1000$
   - $N$: the length of the chains
   - $n$: the number of chains simulated
   - $\vec{q_0}$: the distribution of the initial state of the chain, in this exercise, we will use $\vec{q_0}$ a Dirac distribution centered in $[0, 0]$

## 6 Results

In this section we present the results obtained with the implemented algorithms. First of all, we used both RWMH and HMC algorithms to simulate n=1000 chains. Figure 1 represents the final $q$ of the chains, with initial parameters $\alpha$=10, $\vec{q_0} = \delta([0, 0])$, $m_1 = 1$, $m_2 = 1$, $N = 100$, $n = 1000$, $\epsilon = 0.01$, $T = 0.1$, $\sigma^2 = 0.1$. Both algorithms use a Dirac distribution centered in

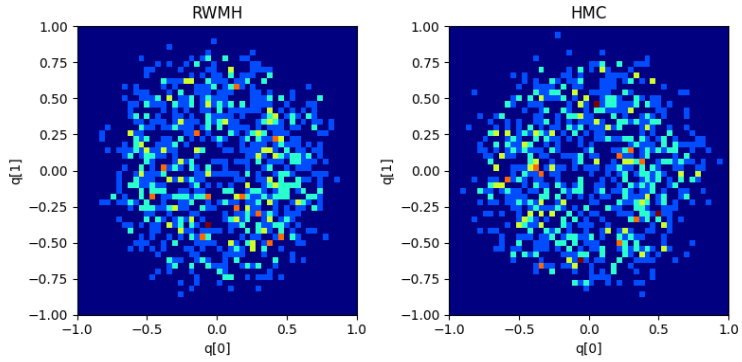



Figure 1: Final $\vec{q}$

$[0, 0]$ to produce the Markov chains. Once the stationary state is reached, the final distribution is invariant under the effect of both algorithms. Therefore, it is possible to plot the sum of every element of the Markov chain after a certain burn-in lag $B$, when the chain has reached its stationary state. Figure 2 represents the density after burn-in lag $B = 20$, using the previously mentioned parameters. The left figure represents the RWMH algorithm, the central figure the

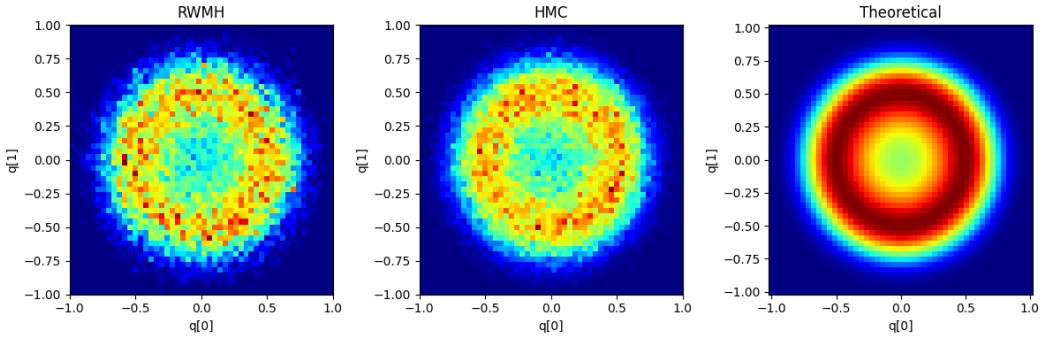



Figure 2: Sum of $\vec{q}$ starting at B=20, $\alpha = 10$

HMC method and the right one the distribution $f$. It is possible to observe that both methods seem to model correctly the distribution. For what concerns RWMH, this result was expected also from a theoretical point of view. Since the distribution is log-concave in tails, the Markov chain generated by the algorithm should be geometrycally ergodic [5]. The ratio of acceptance is for RWMH $r_{RWMH} = 0.632$ and for HMC $r_{HMC} = 0.9997$. This result for HMC is in line with what was explained in section 3. Since we are exploiting a symplectic integrator, the problem is approximated with an high accuracy, even if it was not possible to reach a perfect acceptance rate. Figure 3 represents the same process, but with $\alpha = 10^3$, $B = 80$ and $N = 200$. In this case, the two methods seem to converge to the distribution $f$.

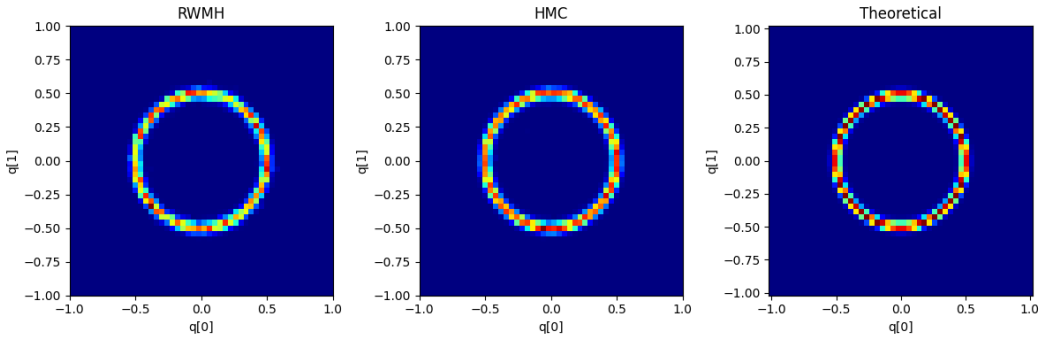
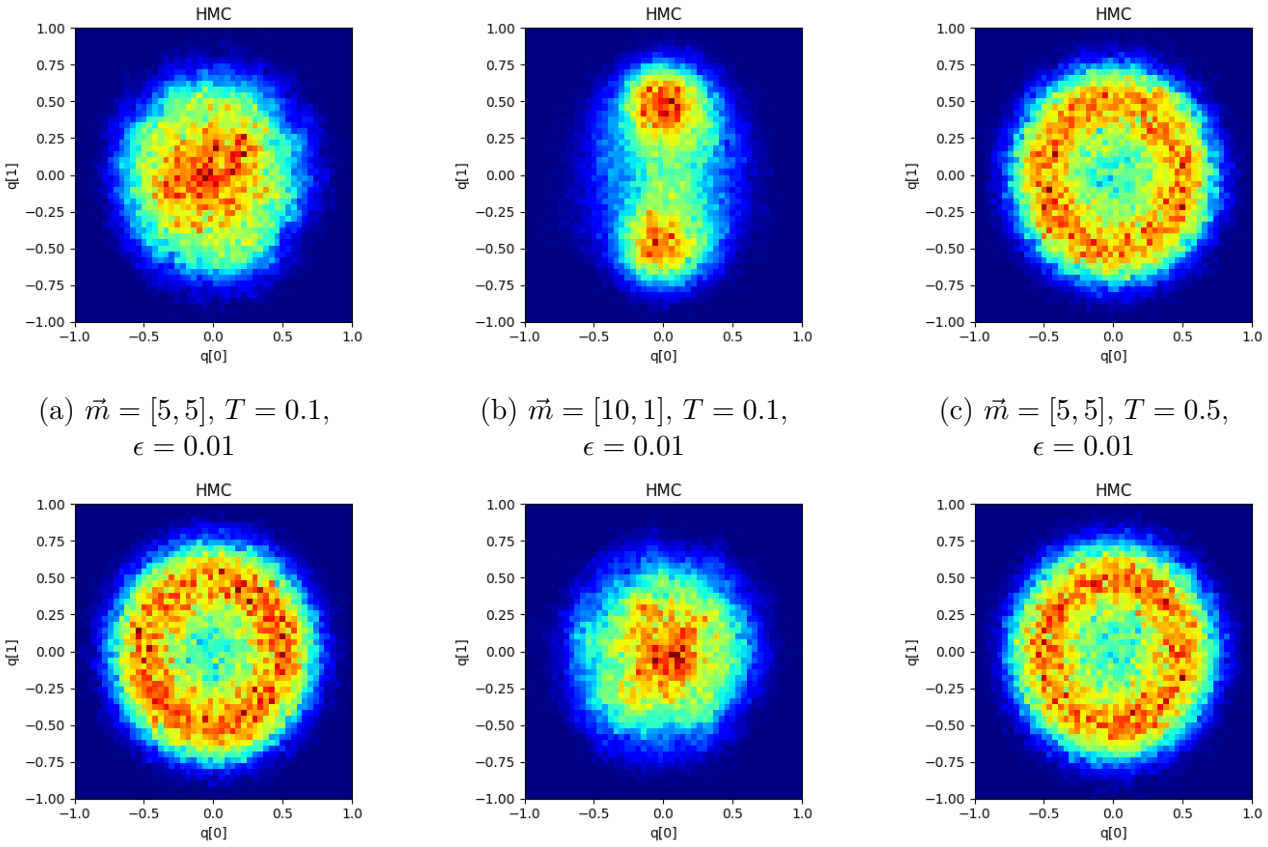



Figure 3: Sum of $\vec{q}$ starting at B=80, $\alpha = 10^3$

We can observe the consequences of the modification of some of the parameters on the HMC method. Fig. 4, for example, represents the effect of different $\vec{m}$, $T$, and $\epsilon$. In this case, we adopted: $\alpha = 10$, $\vec{q_0} = \delta([0, 0])$, $N = 100$, $n = 1000$. It is possible to observe that an increase of $m_i$, $\epsilon$ or a decrease of $T$ imply that the simulation does not follow the distribution $f$. This is probably due to the fact that, with these parameters, the chain does not reach a stable state. It is then possible to study different properties on these Markov chains. Figures 5 and 6 represent the average and variance on the 1000 chains, as a function of the state of the chains, respectively for $\alpha = 10$ and $\alpha = 10^3$. We can observe that, as the initial distribution is a Dirac distribution centered in $[0, 0]$, the initial value of average and variance are both zeros. Then, the average oscillates around zero while the variance stabilizes around 0.15 for $\alpha = 10$ and 0.12 when $\alpha = 10^3$. Another meaningful information we can deduce from this graphs is the instant when the stability of the variance occurs. More precisely, we can define the beginning of this




(a) $\vec{m} = [5, 5]$, $T = 0.1$, $\epsilon = 0.01$

(b) $\vec{m} = [10, 1]$, $T = 0.1$, $\epsilon = 0.01$

(c) $\vec{m} = [5, 5]$, $T = 0.5$, $\epsilon = 0.01$

(d) $\vec{m} = [5, 5]$, $T = 0.5$, $\epsilon = 0.05$

(e) $\vec{m} = [5, 5]$, $T = 0.1$, $\epsilon = 0.05$

(f) $\vec{m} = [0.1, 0.1]$, $T = 0.1$, $\epsilon = 0.01$

Figure 4: Effect of different parameters for HMC

trend before 10 elements of the Markov chain for both methods when $\alpha = 10$, and around 80 for RWMH and HMC when $\alpha = 10^3$. These values allow us to define the burn-in lag, which will be used to calculate the autocovariance of the chain.

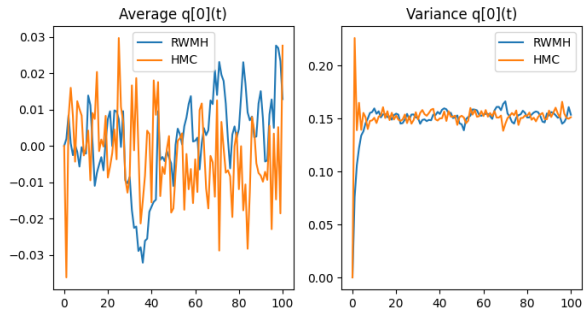



Figure 5: Average and variance of $\vec{q}$, $\alpha = 10$

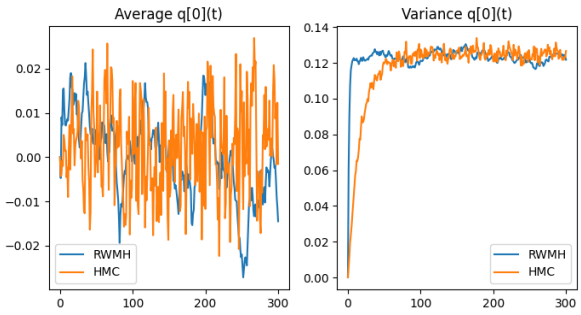



Figure 6: Average and variance of $\vec{q}$, $\alpha = 10^3$

Other key features for a detailed analysis, as more deeply explained in section 5, are the autocovariance and autocorrelation. In this case, the average correlation on n chains in the HMC chain is going quickly to zero, which means that each state is largely independent of the past states of the chain. For what concern RWMH, on the contrary, the autocorrelation is not following the exact same path. This may mean zero there is a higher relation between the states. Therefore, it is possible to conclude that the sampling from HMC has more potential, as it provides more independent samples than RWMH.
These results are represented in figures 7 and 8 while in figures 9 and 10 the averages on the

n chains are reported. Since there is a symmetry between $q[0]$ and $q[1]$, the graphs are only shown for $q[0]$. We also observe that the autocorrelation and autocovariance for one chain are not stable. This is probably due to the calculation of the covariance, since the elements $\hat{c}(k)$ are samples average of very few terms. Parameters used are: $\vec{q_0} = \delta([0,0])$, $\vec{m} = [0.1, 0.1]$, $n = 1000$, $\epsilon = 0.01$, $T = 0.2$, $\sigma^2 = 0.1$, $N = 100$, $B = 20$ for $\alpha = 10$ and $N = 500$, $B = 100$ for $\alpha = 10^3$.

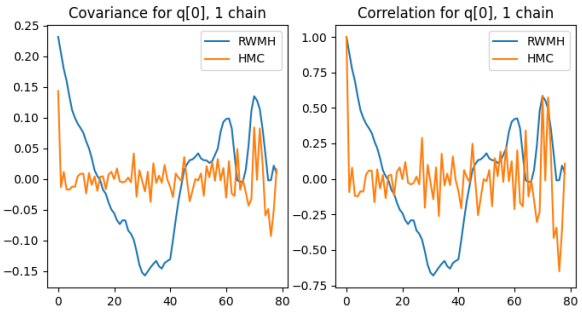



Figure 7: Covariance and correlation of 1 chain for $\vec{q}$, $\alpha = 10$

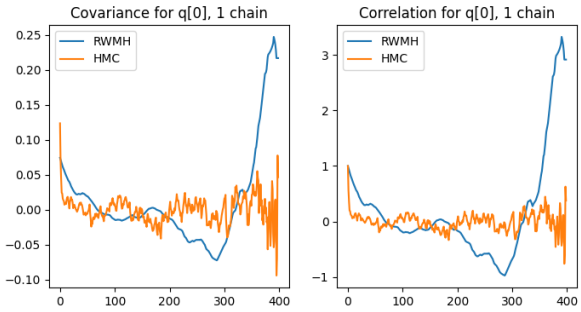



Figure 8: Covariance and correlation of 1 chain for $\vec{q}$, $\alpha = 10^3$

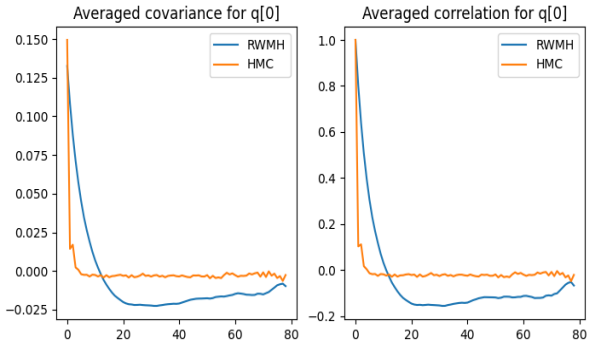



Figure 9: Averaged covariance and correlation for $\vec{q}$, $\alpha = 10$

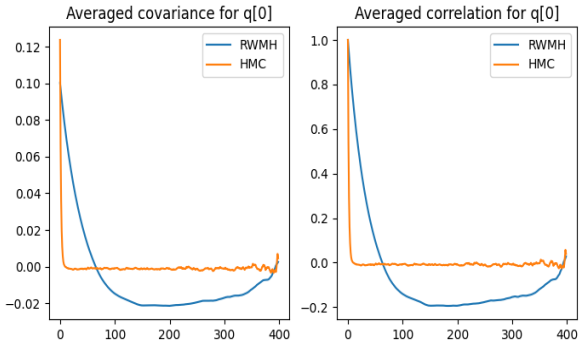



Figure 10: Averaged covariance and correlation for $\vec{q}$, $\alpha = 10^3$

The previously mentioned remark is also shown in figures 11 and 12, that represent the distribution of the effective sample size (ESS), as described in section 5 obtained using the same parameters as before. We observe, as we could expect from the autocorrelation graphs, that the effective sample size is higher for HMC than for RWMH.

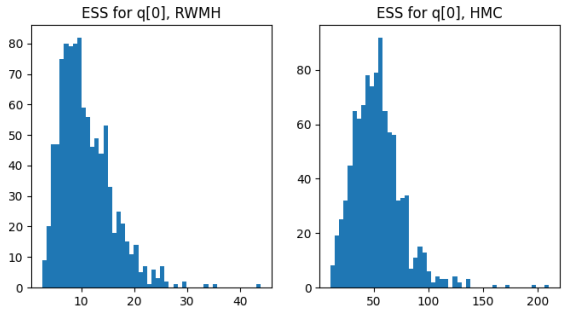



Figure 11: Distribution of effective sample size, $\alpha = 10$

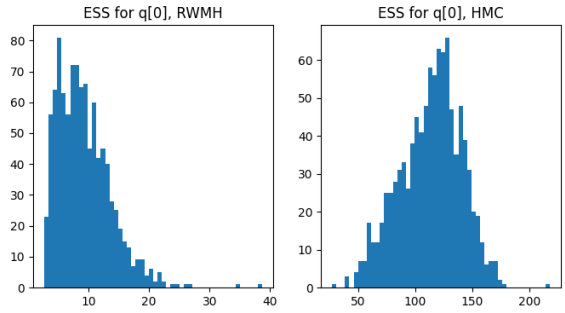



Figure 12: Distribution of effective sample size, $\alpha = 10^3$

Finally, one last useful thing to analyse is the effective sample size as a function of the number of evaluations of $f$ and $\partial f/\partial q_i$. The results are shown in figures 13 and 14. Parameters are given by: $\vec{q_0} = \delta([0,0])$, $\vec{m} = [0.1, 0.1]$, $n = 50$, $\epsilon = 0.01$, $T = 0.2$, $\sigma^2 = 0.1$, $N = 50 - 500$, $B = 20$ for $\alpha = 10$ and $N = 100 - 500$, $B = 80$ for $\alpha = 10^3$. The length of the chain $N$ has been varied between the two reported values to get different number of evaluations of $f$ and $\partial f/\partial q_i$. The number of evaluations $n_{ev}$ for HMC is equal to $n_{ev} = 2 + 4\frac{T}{\epsilon}N$ while, for RWMH, is defined as $n_{ev} = 2N$. One can see that the effective sample size is linearly increasing as a function of $n_{ev}$ of $f$ and $\partial f/\partial q_i$. For both algorithms, the effective sample size is higher for $\alpha = 10$ than for $\alpha = 10^3$. The number of evaluations of $f$ and $\partial f/\partial q_i$ is higher for HMC than for RWMH, due to the fact that the Verlet's scheme needs each time $O(4 * \frac{T}{\epsilon})$ evaluations when the RWMH only needs two computation of $f$ in order to calculate the coefficient of acceptance. But as the Verlet's scheme does not require a high computational time, it would not make sense to compare the performance of both algorithms for the same number of evaluations of $f$ and $\partial f/\partial q_i$.

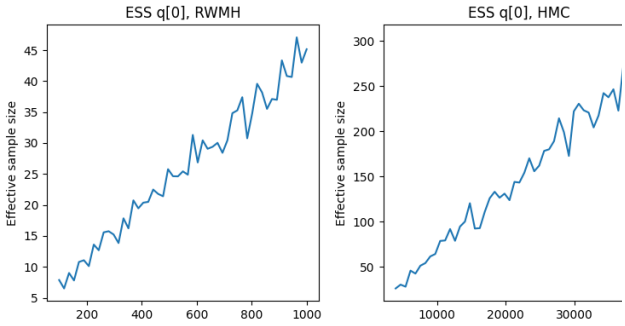



Figure 13: ESS as a function of the number of evaluation of $f$ and $\partial f/\partial q_i$, $\alpha = 10$

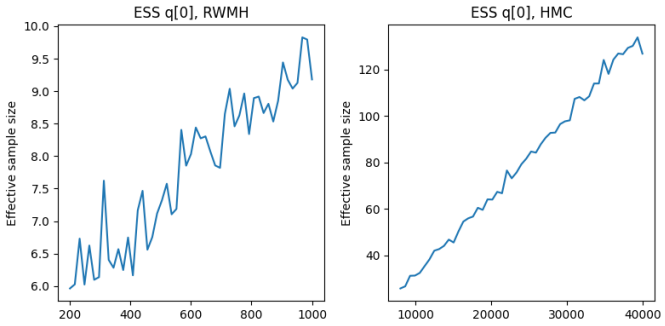



Figure 14: ESS as a function of the number of evaluation of $f$ and $\partial f/\partial q_i$, $\alpha = 10^3$

## 7 Application

In this section, we applied our Hamilonian Monte-Carlo to sample from a distribution reported in [3]. The dataset, contains data of 189 births, collected at Baystate Medical Center in Springfield, Massachusetts. More precisely, there are information about the weights of newborns and indications about characteristics of the mother which can be correlated to this. It was asked to perform a logistic regression with the HMC sampler on this dataset.
First of all, the proposed dataset has been slightly modified to meet the requests. For example, a column for the intersept has been added and the initial covariate "RACE" (which had value 1 for white, 2 for black and 3 for other) has been splitted in 2 categorical binary features, "race2black" and "race2other". Moreover, the column "LOW" as been used as binary response of our model. The features of the dataset are:

- **intersept**: column of ones added to the original dataset
- **age**: integer values which denotes the age of the mother
- **lwt**: mother's weight in pounds at last mestrual period
- **race2black**: categorical variable: 0 if white, 1 if black
- **race2other**: categorical variable: 0 if white or black, 1 if other
- **smoke**: categorical variable: 1 if smoking during pregnancy

- **ptd**: categorical variable: 1 if premature birth
- **ht**: categorical variable: 1 if hypertension
- **ui**: categorical variable: 1 if uterine irritability
- **ftv21**: categorical variable: 1 if one physician visit during the first trimester
- **ftv22+**: categorical variable: 1 if at least two physician visits during the first trimester

In particular, the two continuous variables (namely, age and lwt) have been normalized to assume values more comparable to the other features. The binary response which divides babies into underweight and normal-weight categories is connected to the other elements of the dataset through a logistic regression model $P(y_i = 1|x_i^T q) = [1 + \exp(-x_i^T q)]^{-1}$ with $i = 1, ..., n$ where $n$ is the number of data. This problem can be modelled implementing the HMC algorithm. Even if this method has a physic background, it can be easily generalized to be suitable in a lot of different contexts. A description of this fact is, for example, reported in [6]. In this case, we have 11 covariates instead of the 2 we have considered in the previous analysis. Moreover, we have to specify a log-posterior for the model, which in this case is:

$$\log f(q|y, X, \sigma_q^2) = q^T X^T (y - \mathbf{1}_n) - \mathbf{1}_n^T [\log(1 + \exp(-x_i^T q))]_{n+1} - \frac{q^T q}{2\sigma_q^2} \tag{22}$$

with gaussian prior $\mathcal{N}(0, \sigma^2 I_{p+1})$ where $p+1$ is the number of the covariates. In the Hamiltonian Monte Carlo algorithm, instead of the potential energy, we have then

$$H(q, p) = log(q|y, X, \sigma_q^2) + \frac{1}{2} p^T M^{-1} p \tag{23}$$

The Verlet method is obviously modified too, and it becomes

$$p_i(t + \frac{\epsilon}{2}) = p_i(t) + \frac{\epsilon}{2} \nabla_q log f(q(t)|y, X, \sigma_q^2) \tag{24}$$

$$q_i(t + \epsilon) = q_i(t) + \epsilon \frac{p_i(t + \frac{\epsilon}{2})}{m_i} \tag{25}$$

$$p_i(t + \epsilon) = p_i(t + \frac{\epsilon}{2}) + \frac{\epsilon}{2} \nabla_q log f(q(t + \epsilon)|y, X, \sigma_q^2) \tag{26}$$

The parameters used to model the problem are the following: $\vec{q_0} = \delta(\vec{0})$, $\vec{m} = [\vec{1}]$, $n = 10$, $\epsilon = 0.01$, $T = 0.4$, $\sigma^2 = 1000$, $N = 400 - 6000$, $B = 200$. These parameters were chosen because the results provided using them were satisfying.In order to study the convergence of the HMC method, we have analysed the effective sample size according to the length of the chain. Since the ESS is a measure of the number of independent samples that are being drawn from the target distribution, an increasing value of this parameter would imply a convergence of the method, In fact this phenomenon means that the samples are becoming less correlated and more representative of the target distribution. Fig. 15 represents the ESS for each feature. We observe that the effective sample size is globally increasing as the length of the Markov chain grows with no exception, therefore implying a convergence of the HMC method.
We present then, in figure 17 the histograms of the simulated posteriors obtained using the previously mentioned parameters.
Finally, we compare the performance of Hamiltonian Monte Carlo with a one variable at a time Metropolis-Hastings algorithm (1MH) to compute the posterior expectation of $q$. One Variable at a time Metropolis-Hastings is similar to RWMH presented in sec. 5 but modifying only one feature at each step, using variance of the normal distribution $\sigma^2 = 0.5$. Figure 16 represents the average on $n = 50$ chains of $q(t)$ for 4 different features. The parameters were: $\vec{q_0} = \delta(\vec{0})$,

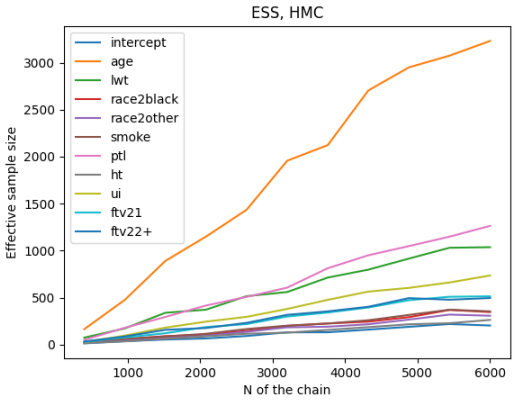



Figure 15: Effective sample size for every feature, as a function of N

$\vec{m} = [\vec{1}]$, $n = 10$, $\epsilon = 0.01$, $T = 0.4$, $\sigma^2 = 1000$, $N = 400$. We observe that the chains produced by HMC algorithm stabilize earlier than 1MH, for any feature and regardless of the value of the variance of the 1MH algorithm. This was expected because the HMC method calculate the next point using the gradient of the log-posterior, while the proposition of next element of the chain using MH method is sampled according to a normal distribution centered in the previous point. Moreover, it is important to observe that the 1MH algorithm validates the results obtained with HMC since both methods seems to reach convergence in correspondence of the same values.

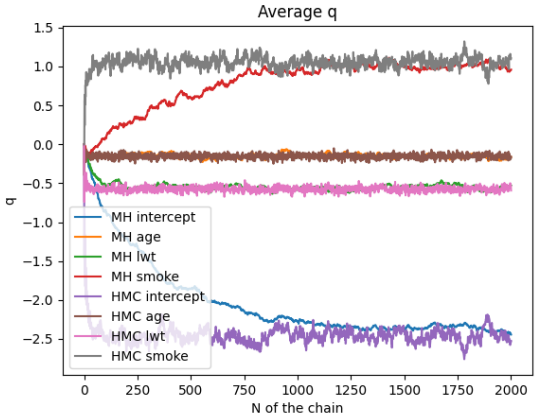



Figure 16: Average on n chains of different features

## 8 Conclusion

In this project, we presented an implementation of a Hamiltonian Monte Carlo method. After describing some theoretical results in sections 2, 3, 4 and methodology to implement the algorithm in section 5, we applied our algorithm to sample from an unnormalized distribution. Comparing HMC with Random Walk Metropolis-Hastings, we have shown that the HMC performed better to sample from the given distribution. Then, we applied the HMC to perform a logistic regression task in section 7, and we observed that the method was convergent. Finally, we compared the HMC for this problem with a One Variable at a time Metropolis-Hastings and we also observed that the HMC performed better than the other method.

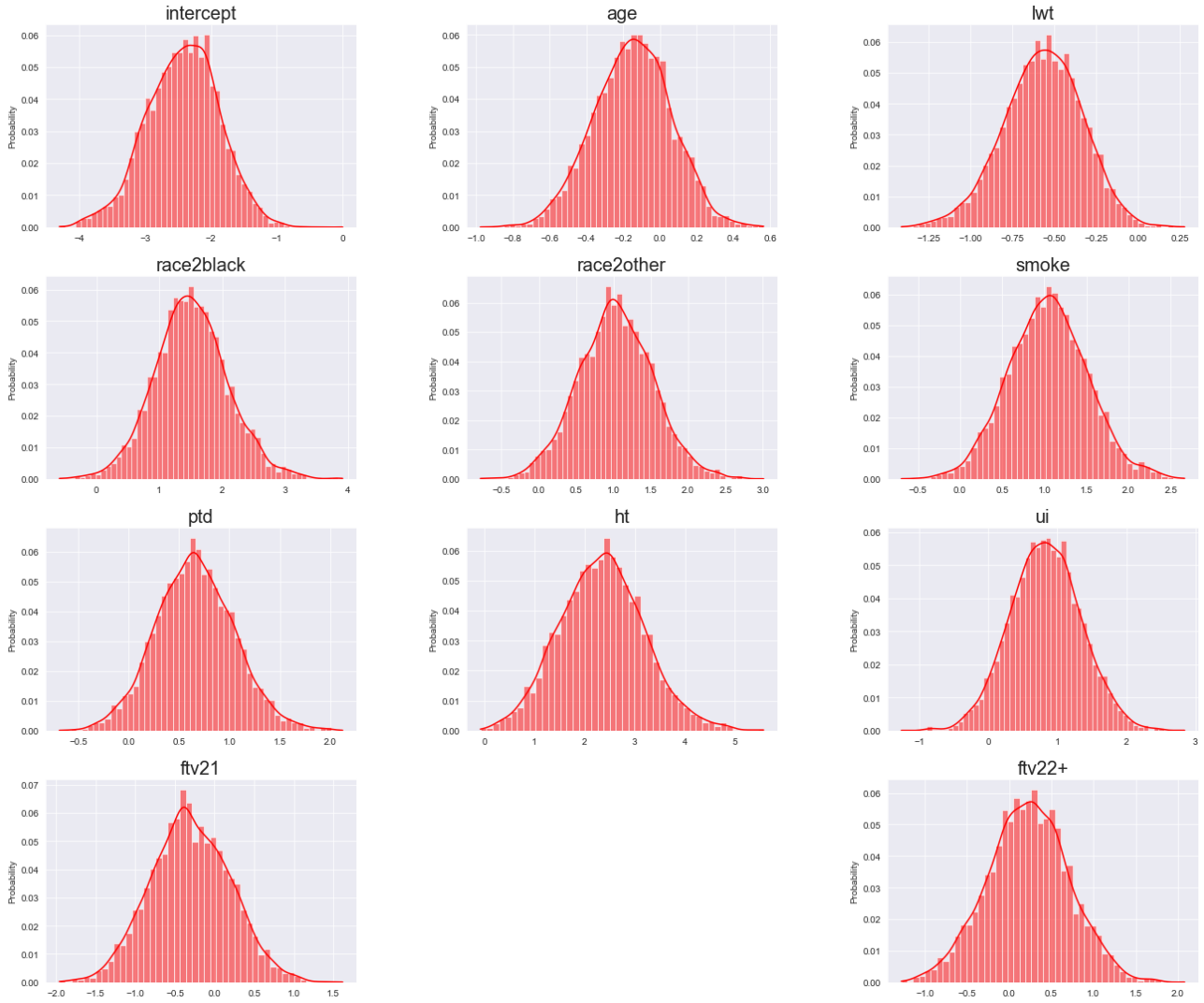



Figure 17: Histograms of the obtained coefficients

## References

[1] Nawaf Bou-Rabee and Jesus Marıa Sanz-Sern. *"Geometric integrators and the Hamiltonian Monte Carlo method"*. Acta Numerica 27, 2018.

[2] Simon Duane, A.B. Kennedy, Bryan Pendleton, and Duncan Roweth. "hybrid monte carlo". pages 216–222, 1987.

[3] Stanley Lemeshow Hosmer, David W and Rodney X Sturdivant. *"Applied Logistic Regression"*. John Wiley Sons, Inc., 1989.

[4] Radford M. Neal. *"MCMC using Hamiltonian dynamics "*. Chapman Hall/ CRC Press, 2011.

[5] Fabio Nobile. *"Lecture notes, Stochastic Simulation"*. 2022.

[6] Samuel Thomas and Wanzhu Tu. "learning hamiltonian monte carlo in r". 2020.

## 9 Code

The following script presents the main functions used to get results in section 6.

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt


def f(q, alpha): # unnormalized distribution
    return np.exp(-alpha*(q@q-1/4)**2)

def U(q, alpha): #-log(f) (potential energy)
    return alpha*(q@q-1/4)**2

def K(p, m): #kinetic energy
    return np.sum(1/2*np.divide(p*p, m))

def dU(q, alpha): #gradient of U
    dU1 = 4*q[0]*alpha*(q@q-1/4)
    dU2 = 4*q[1]*alpha*(q@q-1/4)
    return np.asarray([dU1, dU2])


def Verlet(q0, p0, eps, T, alpha, m): #Verlet's scheme
    t=0
    q=q0
    p=p0
    while t<T:
        p_tmp = p - eps/2*dU(q, alpha)
        q = q + eps*np.divide(p_tmp, m)
        p = p_tmp - eps/2*dU(q, alpha)
        t=t+eps
    return q, p

def Hamiltonian_Monte_Carlo(q0, m, N, T, eps, alpha):
    q = np.zeros([N+1, 2])
    q[0, :] = q0
    accepted = 0
    rejected = 0
    for i in range(N):
        p = st.norm.rvs(loc=0, scale=np.sqrt(m))
        q_star, p_star = Verlet(q[i, :], p, eps, T, alpha, m)
        u = np.random.uniform()
        if (u<np.exp(-U(q_star, alpha)+U(q[i, :], alpha)-K(p_star, m)+K(p, m
    ))):
            q[i+1, :] = q_star
            accepted = accepted+1
        else:
            q[i+1, :] = q[i, :]
            rejected = rejected+1
    ratio = accepted/(accepted+rejected)
    return q, ratio

def Metropolis_Hastings(q0, N, alpha, sigma):
    q = np.zeros([N+1, 2])
    q[0, :] = q0
    accepted = 0
    rejected = 0
    for i in range(N):
        q_star = np.random.multivariate_normal(mean=q[i, :], cov=sigma*np.
    eye(2))
```

```
        a = f(q_star, alpha)/np.maximum(f(q[i, :], alpha), np.finfo(np.float64).eps)
        u = np.random.uniform()
        if (u<a):
            q[i+1, :] = q_star
            accepted = accepted+1
        else:
            q[i+1, :] = q[i, :]
            rejected = rejected+1
    ratio = accepted/(accepted+rejected)
    return q, ratio


def autocovariance(X): # Autocovariance of a Markov chain X, X tail of q after burn-in lag B
    mu = np.mean(X, axis=0)
    N = len(X[:, 0])
    cov = np.zeros([N-1, 2])

    for k in range(N-1):
        for i in range(N-k-1):
            cov[k, :] += ((X[i+k, :])-mu)*(X[i, :]-mu)
        cov[k, :] = (1/(N-k-1))*cov[k, :]

    return cov

def autocorrelation(autocov): # Autocorrelation of a Markov chain, calculated from its autocovariance
    corr = np.zeros(np.shape(autocov))
    for i in range(int(len(autocov[0, :, 0]))):
        corr[:, i, 0] = np.divide(autocov[:, i, 0], np.maximum(np.abs(autocov[:, 0, 0]), np.finfo(np.float64).eps))
        corr[:, i, 1] = np.divide(autocov[:, i, 1], np.maximum(np.abs(autocov[:, 0, 1]), np.finfo(np.float64).eps))

    return corr

def get_M(autocov): # Last element of the chain used to calculate sigma^2_mcmc
    M1 = None
    for k in range(int(len(autocov[:, 0])/2)):
        if ((autocov[2*k, 0] + autocov[2*k+1, 0])<=0):
            M1 = 2*k
            break
    if M1 is None:
        M1 = int(len(autocov[:, 0])-2)
    M2 = None
    for j in range(int(len(autocov[:, 1])/2)):
        if ((autocov[2*j, 1] + autocov[2*j+1, 1])<=0) :
            M2 = 2*j
            break
    if M2 is None:
        M2 = int(len(autocov[:, 1])-2)

    return M1, M2

def get_sigma(autocov): # Time-average variance constant of a Markov chain, calculated from its autocovariance
    [M1, M2] = get_M(autocov)
    id1 = range(1, M1)
    S1 = autocov[0, 0] + 2*np.sum(autocov[id1, 0])
    id2 = range(1, M2)
```

```
    S2 = autocov[0, 1] + 2*np.sum(autocov[id2, 1])

    return [S1, S2]

def effective_sample_size(autocov): # effective sample size of a Markov chain, calculated from its autocovariance
    [n, N] = np.shape(autocov[:, :, 0])
    ess = np.zeros([n, 2])
    for i in range(n):
        sigma = get_sigma(autocov[i, :, :])
        ess[i, 0] = N*np.divide(autocov[i, 0, 0], np.maximum(sigma[0],  np.finfo(np.float64).eps))
        ess[i, 1] = N*np.divide(autocov[i, 0, 1], np.maximum(sigma[1],  np.finfo(np.float64).eps))

    return ess
```

The following script presents the main functions used to get results in section 7

```
import seaborn as sns
import pandas as pd
import os
import scipy.stats as st
import matplotlib.pyplot as plt
import numpy as np


def log_posterior(q, X, y, sigma=1000):  # log posterior

    n, p = (X.shape[0], X.shape[1] - 1)

    return q @ X.T @ (y.T - np.ones(n)) - np.ones(n) @ (
        np.log1p(np.exp(-X @ q)) - q @ q.T / (2 * sigma)
    )


def grad_log(X, y, q, sigma):  # gradient of the log posterior
    value = np.exp(-X @ q)
    place = np.where(value == np.inf)[0]

    value = value / (1 + value)
    value[place] = 1.0

    grad = X.T @ (y - np.ones(y.shape[0]) + value) - q / sigma

    return grad


def K(p, m):  # ""kinetic energy"
    return np.sum(1 / 2 * np.divide(p * p, m))


def Verlet(q0, m, p0, eps, T, X, y, sigma):  # Verlet method
    t = 0
    q = q0
    p = p0
    while t < T:
        p_tmp = p + eps / 2 * grad_log(X, y, q, sigma)

        q = q + eps * np.divide(p_tmp, m)

        p = p_tmp + eps / 2 * grad_log(X, y, q, sigma)

```

```
        t = t + eps
    return q, p


def Hamiltonian_Monte_Carlo(
    q0, m, N, T, eps, X, y, sigma
):  # HMC scheme adapted for the low-weight problem

    q = np.zeros([N + 1, q0.shape[1]])
    q[0, :] = q0
    accepted = 0
    rejected = 0
    for i in range(N):
        p = st.norm.rvs(loc=0, scale=np.sqrt(m))

        q_star, p_star = Verlet(
            q0=q[i, :], m=m, p0=p, eps=eps, T=T, X=X, y=y, sigma=sigma
        )

        u = np.random.uniform()
        if u < np.exp(
            log_posterior(q_star, X, y, sigma)
            - log_posterior(q[i, :], X, y, sigma)
            - K(p_star, m)
            + K(p, m)
        ):
            q[i + 1, :] = q_star
            accepted = accepted + 1
        else:
            q[i + 1, :] = q[i, :]
            rejected = rejected + 1
            print(f"Problem:{i+1}")
    ratio = accepted / (accepted + rejected)
    return q, ratio



def MH_one_at_a_time(q0, N, var, X, y, sigma):
    n = len(q0[0, :])
    print(n)
    q = np.zeros([N + 1, n])

    q[0, :] = q0
    accepted = 0
    rejected = 0

    for i in range(N):
        j = np.random.randint(0, n)
        q_star = np.copy(q[i, :])
        # define posterior wrt other variates (or.....use the prior, like MH
    ?)
        q_star[j] = np.random.normal(loc=q[i, j], scale=var)
        a = np.exp(log_posterior(q_star, X, y, sigma)-log_posterior(q[i, :],
     X, y, sigma))
        u = np.random.uniform()
        if (u<a):
            q[i+1, :] = q_star
            accepted = accepted+1
        else:
            q[i+1, :] = q[i, :]
            rejected = rejected+1

```

```
    ratio = accepted/(accepted+rejected)
    return q, ratio


def dataset_import():
    this_dir = os.path.dirname(os.getcwd())

    data_dir = os.path.join(this_dir, "src\\birthwt.csv")
    df = pd.read_csv(data_dir)
    y = df["low"].copy()
    df = df.drop("Unnamed: 0", axis=1)
    df = df.drop("low", axis=1)
    df = df.drop("bwt", axis=1)
    df.rename(columns={"race": "af_am"}, inplace=True)
    df.insert(3, "Other", df["af_am"], True)
    df.rename(columns={"ftv": "visit"}, inplace=True)
    df.insert(9, "visits", df["visit"], True)

    # transform to array
    X = df.to_numpy()
    y = y.to_numpy()
    # visits last months
    X[np.where(X[:, -1] < 2)[0], -1] = 0
    X[np.where(X[:, -1] >= 2)[0], -1] = 1
    # only one visit
    X[np.where(X[:, -2] != 1)[0], -2] = 0
    # afro american
    X[np.where(X[:, 2] != 2)[0], 2] = 0
    X[np.where(X[:, 2] == 2)[0], 2] = 1
    # other
    X[np.where(X[:, 3] != 3)[0], 3] = 0
    X[np.where(X[:, 3] == 3)[0], 3] = 1

    # intersept
    X = np.c_[np.ones((X.shape[0], 1)), X]
    # normalization of age and lwt, only non categorical variables
    X[:, 1] = (X[:, 1] - np.mean(X[:, 1])) / np.std(X[:, 1])
    X[:, 2] = (X[:, 2] - np.mean(X[:, 2])) / np.std(X[:, 2])
    return X, y


def autocovariance(X): # X is only the tail of q after the burn-in
    mu = np.mean(X, axis=0)
    N = len(X[:, 0])
    D = len(X[0, :])
    cov = np.zeros([N-1, D])

    for k in range(N-1):
        for i in range(N-k-1):
            cov[k, :] += ((X[i+k, :])-mu)*(X[i, :]-mu)
        cov[k, :] = (1/(N-k-1))*cov[k, :]

    return cov

def autocorrelation(autocov):
    corr = np.zeros(np.shape(autocov))
    D = len(autocov[0, 0, :])
    for i in range(int(len(autocov[0, :, 0]))):
        for k in range(D):
            corr[:, i, k] = np.divide(autocov[:, i, k], np.maximum(np.abs(
   autocov[:, 0, k]), np.finfo(np.float64).eps))

```

```
    return corr

def get_M(autocov): # check adaptation of function for  2 dimensions
    D = len(autocov[0, :])
    M = np.zeros(D)
    for d in range(D):
        M[d] = None
        for k in range(int(len(autocov[:, d])/2)):
            if ((autocov[2*k, d] + autocov[2*k+1, d])<=0):
                M[d] = 2*k
                break
        if M[d] is None:
            M[d] = int(len(autocov[:, d])-2)


    return M

def get_sigma(autocov): # To check (difference between serie 13 and lecture notes)
    M = np.ndarray.astype(get_M(autocov), int)
    D = len(autocov[0, :])
    S = np.zeros(D)
    for d in range(D):
        id = range(1, M[d])
        S[d] = autocov[0, d] + 2*np.sum(autocov[id, d])

    return S


def effective_sample_size(autocov):
    [n, N, D] = np.shape(autocov[:, :, :])
    ess = np.zeros([n, D])
    for i in range(n):
        sigma = get_sigma(autocov[i, :, :])
        for d in range(D):
        #print("Sigma=", sigma, ", c0=", autocov[i, 0, :], ", N=", N)
            ess[i, d] = N*np.divide(autocov[i, 0, d], np.maximum(sigma[d], np.finfo(np.float64).eps))

    return ess


if __name__ == "__main__":

    X, y = dataset_import()

    q0 = np.zeros((1, X.shape[1]))
    eps = 0.01
    m = np.ones(X.shape[1])
    T = 0.40
    N = 6000
    sigma = 1000
    B = 4800

    (q, ratio) = Hamiltonian_Monte_Carlo(q0, m, N, T, eps, X, y, sigma)

    # histograms
    sns.set_style("darkgrid")
    rel = sns.histplot(
        data=q[:, 0],
        kde=True,
        stat="probability",
```

```
        color="r",
        label="probabilities",
    )
    plt.title("intercept", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 1],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("age", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 2],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("lwt", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 3],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("race2black", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 4],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("race2other", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 5],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("smoke", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 6],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
```

```
    )
    plt.title("ptd", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 7],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("ht", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 8],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("ui", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 9],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("ftv21", fontsize=20)
    plt.show()

    rel = sns.histplot(
        data=q[:, 10],
        kde=True,
        stat="probability",
        color="r",
        label="probabilities",
    )
    plt.title("ftv22+", fontsize=20)
    plt.show()
```