Abstract

Hamiltonian Monte Carlo (HMC) are a special class of Markov Chain Monte Carlo algorithms (MCMC) which demonstrated empirical success in many foregrounds due to their ability to efficiently sample in the typical set [3], even for high dimensions problem.

The parameters space is embedded with auxiliary momentum variables to generate a fake Hamiltonian where the potential energy is proportional to some transformation of the target probability distribution. In this way, the information about the parameter space is taken into account for new efficient proposal. The Hamiltonian Dynamics are evolved at each step of the Markov chain, to ensure a transition to state space at low autocorrelation and avoiding the tedious random walk behaviour of many MCMC samplers. This result in much more efficiency and less waste of computational power.

This thesis delve to study the theoretical foundation of Hamiltonian Monte Carlo algorithm and its variations and improvements [13] [11] [14]. The implementation is carried out in python and comprehend the realization of the existing algorithms as well as the presentation of a novel version.

To ensure the validity of the implemented algorithms, statistical test have been carried out on toy problems with known results, to compare the outcomes. Kolmogorov-Smirnov and likelihood ratio tests have been applied on multivariate Gaussian distributions and Rosenbrock function. Results from automatic software validation [8] are analyzed in a pp-plot for a statistical analysis for the validity of algorithms.

Benchmarks application for simple linear and logistic regression are presented and compared to the well known Metropolis-Hastings algorithm to show the enormous advantages of HMC samplers.

Finally HMCs algorithms are tested to analyze their suitability to infer parameters from Gravitational Waves (GWs) signals captured by the Laser Interferometer Space Antenna (LISA) [**?**] for the Massive Black Holes mergers and the binaries Black Hole-Black Hole during their inspiralling phase.

Result have then been analyzed and compared displaying promising outcomes.

Contents

Chapter 1

Introduction

1.1 Introduction

This thesis focuses on the study of Hamiltonian Monte Carlo (HMC) samplers, the aim is to burrow into a detailed discussion of HMC and its implementation. To ensure clarity and provide a solid foundation for understanding the upcoming content and notation, the introduction will cover two essential topics: Markov Chain Monte Carlo (MCMC) and Hamiltonian dynamics.
MCMC methods play a critical role in statistics and physics by facilitating the exploration of complex probability distributions. These methods, including HMC, utilize iterative sampling to generate samples from these distributions, aiding in tasks such as statistical inference and model fitting.
Hamiltonian dynamics, rooted in classical mechanics, offers a framework for comprehending the evolution of physical systems over time. By integrating principles from Hamiltonian mechanics, HMC presents a sophisticated approach to sampling distributions by drawing analogies between statistical mechanics and Hamiltonian dynamics.
By thoroughly examining these foundational concepts, the thesis sets the stage for an in-depth exploration of Hamiltonian Monte Carlo. This approach establishes a strong basis for understanding how HMC is applied in physics and computational sciences.

1.2 Markov Chain Monte-Carlo

1.2.1 MC

?? Markov chain Monte-Carlo algorithms are a class of computational methods to obtain numerical solutions through iterative sampling. Those algorithms have been

used in many research fields like optimization, numerical integration and regression. In general their power is regarded the ability to solve a problem of the form:

$$\bar{f}(q) = \int dq\pi(q)f(q) \tag{1.1}$$

With $\pi()$ a general probability distribution, dq the metric space and $f()$ a generic function.
Monte-Carlo approach consists on sampling the stochastic process q with a particular algorithm and substitute the integral with a sum over the collected sample.

$$\bar{f}(q) \sim \mu_f(q) = \frac{1}{N}\sum_i^N f(q_i) \tag{1.2}$$

The central limit theorem assure us that whatever the starting distribution is (as long as it has finite variance), this estimator will be distributed accordingly to a Gaussian with mean and variance related to the original values.

$$\mu_f \sim G(\bar{f}, \sigma_f^2/N) \tag{1.3}$$

In this view for an uncorrelated sample the estimators 1.2 for the mean is asymptotically consistent and unbiased, meaning that for a high enough number of samples, the estimator converge in probability to the true distribution. While is easy to demonstrate that the mean of the data represent a unbiased estimator for the mean. Due the Gaussian asymptotic distribution of the mean, its natural maximum likelihood estimator (MLE) for its variance is the sum squared of the differences with the mean. Since the true mean is unknown the estimated mean is used, which results in an almost unbiased estimator and for which bias correction is $N/(N-1)$.

$$\sigma_{\mu_f}^2 = \sigma_f^2/N = \frac{1}{N(N-1)}\sum_i^N (f_i - \mu_f)^2 \tag{1.4}$$

1.2.2 MCMC

Markov Chain MC are particular Monte-Carlo methods. The stochastic sample of a continuous variable in $\mathbb{R}^D$ is generated according to a Markov chain. This means that in the given configuration space the next state visited will depend only on the current state ($\pi(q_{t+1}|q_t, q_{t-1}...) = \pi(q_{t+1}|q_t)$) and occurs with a time-independent transition kernel K which is defined as:

- Given a support $A \subset \mathbb{R}$, $K(q, A)$ is defined as the probability of reaching the measurable set A from state q.
- Given q, it becomes a pdf, $\int K(q, q')dq' = 1$ which means that probability to get q starting from all possible configuration space is 1.

The Kernel proposes the next candidate at each iteration, and the goal of the algorithm is to approximate the target distribution using the collection of all proposed states through this method. This implies that if the algorithm accurately reproduces the target distribution, the produced sample must eventually recover its form after a certain number of iterations, and further iterations of the kernel should not alter the results. This fundamental requirement states that the algorithm must ultimately converge to a stationary distribution, reaching an equilibrium state that leaves the target distribution invariant.
In the discrete case, this is known as the detailed balance principle and can be shown to be equivalent to requiring symmetry of the kernel starting from two different points. In fact this equilibrium condition can be expressed in the discrete case as $lim_{j\to\infty} \sum_b K^j_{ab}\pi^b = \pi^a$. To hold, the following relation must be verified: $K^{ab}\pi^a = K^{ba}\pi^b$, hence imposing a symmetry on the proposal. The continuous counterpart can be much more cumbersome, and in general, there are no simple formulas like the detailed balance itself. However, it can be schematized in a general form based on the definition.:

$$\pi(q) \sim \pi(q)_{t+1} = \int K(q, q')\pi(q')_t dq' \tag{1.5}$$

This to remark the property that after a sufficient number of iterations, subsequent re-application of the kernel should not change the obtained distribution, hence it has reached equilibrium.

Another essential property the kernel must possess to ensure correct convergence to the target distribution is ergodicity. Formally, this require that for every measurable set A of the state space, for each $q \in A$ exists n such that $K(q, A)^n > 0$.. In other words, a Markov chain kernel is ergodic if it can eventually transition from any initial state to any other state in the system.
Pseudocode describing a MCMC algorithm with a to-be specified kernel is as follow:

1.2.3 Autocorrelation

The aim of Monte Carlo algorithms is to produce a sample which follows the probability distribution of interest. In the previous consideration, the estimator in eq.

Algorithm 1 MCMC
given *iteration*, *Kernel*

```
procedure MAIN: MCMC
    q has to be initialized
    while i < iteration do
        collect q ← Kernel(q) #
    end while
return collected sample
end procedure
```

(**??**) is calculated assuming independent samples. However, Markov Chain Monte Carlo samplers update based on the current position, thereby inevitably introducing autocorrelation between subsequent points. For NN independent draws, the extraction probability is simply the product of NN independent probability distributions. Conversely, if the sample is drawn as a sequence in a Markov chain, the sample extraction probability is governed by the Markov process itself, resulting in a biased variance estimate.

$$\sigma^2_{\mu_f} =< (\mu_f - \bar{f})^2 > \tag{1.6}$$

This equation, that reduces to eq.(1.4) in case of independent samples, can be rearranged to include such phenomena.
After some algebra one can arrive at a general definition

$$\sigma^2_{\mu_f} = \frac{\sigma^2_f}{N^2} \sum_i \sum_{j-i} C(i,j) \tag{1.7}$$

with

$$C(i,j) = \frac{< \delta f_i \delta f_j >}{\sigma^2_f} \tag{1.8}$$

$$\delta f_i = f_i - \bar{f} \tag{1.9}$$

Where $C(i,j)$ is the so-called (sample) autocorrelation function and it is usually not trivial to compute. However when the Markov-chain reaches its equilibrium, some important properties can be stated which significantly simplify the problem. In fact $C(i,j)$ is, by definition, symmetric under exchange of i and j and, for a time-independent Markov process, can only depend on the distance between i and j, i.e. $C(i,j) = C(|i-j|)$, so that $\sigma^2_f = C(0)$, hence at stationarity it becomes:

$$\sigma^2_{\mu_f} = C(0)\frac{1}{N} \sum_{|j-i|=-\infty}^{\infty} C(|i-j|) = \frac{C(0)}{N}(1 + 2\tau_{int}) \tag{1.10}$$

with

$$\tau_{int} = \sum_{k=1}^{\infty} \frac{C(k)}{C(0)} \tag{1.11}$$

Thus, even if one is interested only in the static quantity μ, it is necessary to estimate the dynamic quantity τ_{int} in order to determine valid error bars for μ [20].

The natural estimator for $C(k)$ is:

$$\hat{C}(k) = \frac{1}{N-k} \sum_{1}^{N-k} (f_i - \mu_f)(f_{i+k} - \mu_f) \tag{1.12}$$

Since the autocorrelation for $k >> \tau_{int}$ contains much noise but little signal [20], the easiest estimator which comes at hand is wrong, as the variance does not go to 0 as the sample increases.
The solution is to apply a cutoff which minimizes both variance and bias of the estimator τ_{int}.

$$\hat{\tau}_{int} = \sum_{k=1}^{M} \frac{\hat{C}(k)}{\hat{C}(0)} \tag{1.13}$$

with $N >> M >> \tau_{int}$ to have an optimal trade off between bias and variance.

This motivation shows the effect of a Markov-Chain sampler and the precautions one should take when making estimates through this kind of algorithms. In fact when the obtained sample is characterised by a relevant auto-correlation, the variance of the estimator is a factor $2\tau_{int}$ higher than it would be in a independent sample.
Hence running a MCMC simulation for N points, produce only $\frac{N}{2\tau_{int}}$ effectively independent data points.
So this plays a crucial role and in general when comparing 2 Monte Carlo algorithms, the better one is the one that has smaller autocorrelation time when time is measured of units of computing time to run such algorithms.

1.2.4 Metropolis-Hastings Algorithm

The Metropolis–Hastings algorithm is the workhorse of MCMC methods, both for its simplicity and its versatility, and hence the first solution to consider in intractable situations.
It operates by constructing a progressive picture of the target distribution, proceed-

ing by local exploration of the state space A until all the regions of interest have been uncovered [19]. It leverages the proportionality to the target distribution by proposing a move that will be accepted with a probability ratio, thereby canceling out the intractable constant. The move is proposed from a probability distribution that depends on the current state of course and usually is taken to be a Gaussian centred on the current position in the chain.
Let $\pi(q)$ be the target distribution of interest and $G(\cdot|\cdot)$ the proposal distribution. The algorithm proceeds as follows:

1. Choose an initial state q_0.
2. For each iteration $t = 1, 2, \ldots$:
   (a) Propose a candidate q' from a proposal distribution $G(q'|q)$.
   (b) Compute the acceptance probability $\alpha(q', q)$ as:

$$\alpha(q', q) = \min\left(1, \frac{\pi(q')}{\pi(q)}\frac{G(q'|q)}{G(q|q')}\right)$$

   (c) Update q as: $q_{new} = \begin{cases} q' & \text{with probability } \alpha(q', q) \\ q & \text{with probability } 1 - \alpha(q', q) \end{cases}$

Hence the Kernel is defined as:

$$K(\cdot, q) = \alpha(\cdot, q)G(\cdot|q) + \delta(\cdot - q)\int(1 - \alpha(\cdot, q))G(\cdot|q)dq \tag{1.14}$$

The kernel function $\alpha(x_{t-1}, x')$ determines whether a proposed state is accepted or rejected based on the target distribution $\pi(x)$ and the proposal distribution $G(x'|x_{t-1})$.

In summary, the Metropolis-Hastings algorithm generates a Markov chain whose stationary distribution is the target distribution $\pi(x)$ by iteratively proposing and either accepting or rejecting candidate states according to an acceptance probability. In 6 is presented a general pseudocode, given the distribution to pick from $G(\cdot, \cdot)$ and its hyperparameters ϵ.

Example

A first application to illustrate both the power and limitations of the methods discussed is the two-dimensional Gaussian distribution with zero mean and a unit covariance matrix. This simple and iconic example helps investigate the properties and address issues associated with these types of algorithms. A total of 10000 points are sampled after the rejection of the first burn-in period of 1000 steps. From the

Algorithm 3 Metropolis-Hastings Kernel: given the current point q_0, a proposal distribution G, and the hyper-parameter ϵ which define the average magnitude of the proposed jump from q_0, it proceeds by computing the acceptance probability α and either reject or accept the new proposed state.

1: **procedure** KERNEL
2: $q_{new} \leftarrow G(q_0, \epsilon)$
3: **if** $\alpha <$ random number $[0, 1]$ **then return** q_{new}
4: **elsereturn** q_{old}
5: **end if**
6: **end procedure**

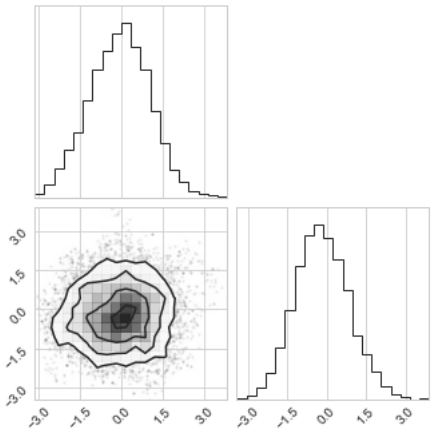

Figure 1.1: Posterior histogram [10] for a 2D Gaussian distribution: The sample is obtained with a Metropolis-Hastings algorithm for 10000 iterations.

histogram of the posterior in fig. 1.1 one could be satisfied with the results since the algorithm correctly reproduce the Gaussian form of the target distribution. However the esteem for the mean and its variance in absence of auto-correlation yield:

$$\mu_0 = -0.0967 \pm 0.0001$$
$$\mu_0 = -0.1885 \pm 0.0001$$

which are not compatible with the known values.
As previously mentioned it is clear that the auto-correlation plays a crucial role in this setting even for such a simple problem. In fact, even by naked eye, one could spot the high auto-correlation of the sample just by looking at the iteration plot in fig.1.2. In fact due to this issue, the state space display a random walk behaviour around the true value. For instance plotting the auto-correlation function and its integral (fig. 1.3) shows the properties previously mentioned. In fact for lag much greater than the auto-correlation time, the signal contain too much noise to be informative.

The trade off for the maximum lag [20] is set to be $lag_{max} = \frac{N}{100} + 1$. In this setting, fig. 1.4, the exponential decay of the auto-correlation can be appreciated as well as the convergence to a fixed Integrated Autocorrelation Time (IAT) at around

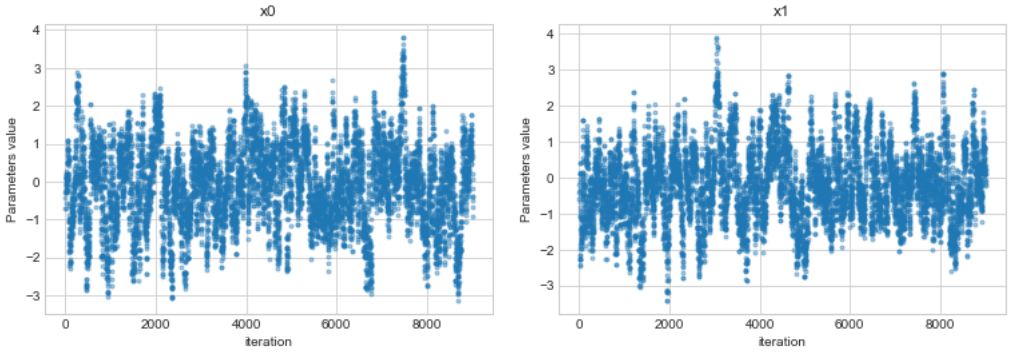

Figure 1.2: Iteration plot for a 2D Gaussian distribution

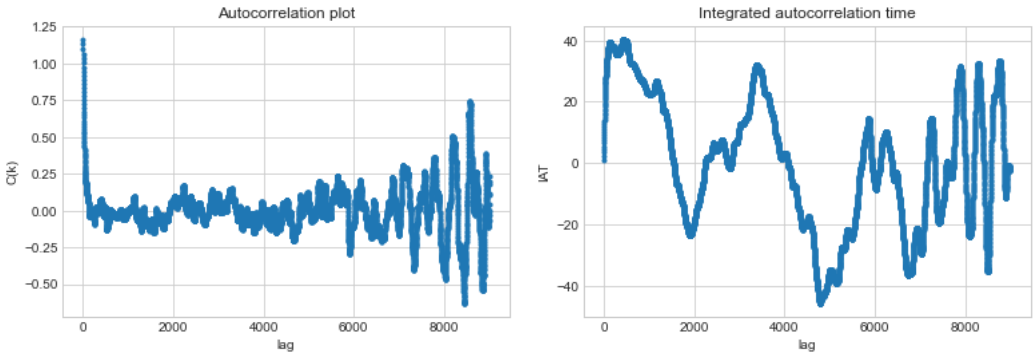

Figure 1.3: Left: autocorrelation function. Right: integrated autocorrelation function
When every value of the lag is considered, is clear the major contribution about the noise as the lag increase

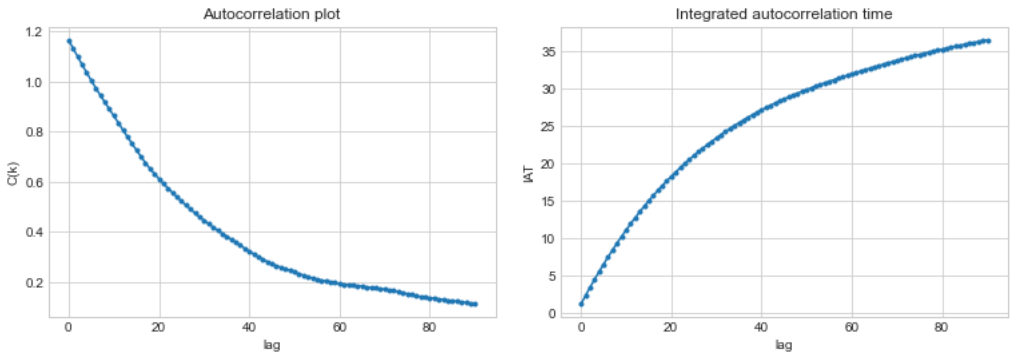

Figure 1.4: Left: autocorrelation function. Right: integrated autocorrelation function
A stopping criterion for the lag allows correct estimation of the IAT and appreciation of its exponential decay.

~ 28 .

As a consequence the esteem after updating the variance changes to more consistent result:

$$\mu_0 = -0.0967 \pm 0.0079$$
$$\mu_0 = -0.1885 \pm 0.0071$$

1.3 MCMC for Bayesian purpose

Bayesian inference is a statistical framework used for updating the probability of a hypothesis as new evidence becomes available. At the core of Bayesian inference is Bayes' theorem, which states:

$$\pi(q|D) = \frac{\pi(D|q) \cdot \pi(q)}{\pi(D)} \tag{1.15}$$

where:

- $\pi(q|D)$ is the posterior distribution of the parameters q given the data D.
- $\pi(D|q)$ is the likelihood of observing the data D given the parameters q.
- $\pi(q)$ is the prior distribution of the parameters q.
- $\pi(D)$ is the marginal likelihood, also known as the evidence.

Bayesian inference involves updating our prior beliefs ($\pi(q)$) about the parameters of interest based on observed data ($\pi(D|q)$). The posterior distribution ($\pi(q|D)$) encapsulates our updated beliefs after observing the data.

In many Bayesian inference problems, the posterior distribution cannot be calculated analytically due to its complexity or unavailability in closed form. Markov Chain Monte Carlo (MCMC) methods provide a solution by generating samples from the posterior distribution.

The proportionality constant $\pi(D)$ in Bayes' theorem, known as the evidence, is often difficult to compute and requires computational methods. MCMC algorithms, such as the Metropolis-Hastings algorithm, exploits the known proportionality of the system. This means that one can work with the unnormalized posterior distribution, often denoted as $\pi(q|D) \propto \pi(D|q) \cdot \pi(q)$. By sampling from the unnormalized posterior distribution using MCMC methods, one can approximate the posterior distribution and make Bayesian inference about the parameters of interest.

1.4 Hamiltonian Dynamics

A physical system can be described by D number of variables q, whose evolution depends on another set of D variables $\dot{q}$ relative to their velocity. Through a Legendre transformation the Lagrangian of the system can be re-expressed in term of the conjugate momenta p. This has the important consequence of describing the evolution of the system with $2D$ first order differential equation rather than D second order differential equations.
This is the Hamiltonian description and the so called phase of space, which is the collection of the possible state for the generalized position and momenta, has $2D$ degree of freedom.
The resulting Hamiltonian is, in fact, the total energy of the system. For the aim of this work, the functional form of the Hamiltonian will be restricted to time independent Hamiltonians (hence the total Energy must be preserved during its evolution) with a quadratic Kinetic term as in eq. 1.16.

$$H(q,p) = U(q) + K(p) = U(q) + \frac{1}{2}p^T M^{-1} p \tag{1.16}$$

where $U(q)$ is the potential energy, $K(p)$ is the kinetic energy and M the mass matrix.

1.4.1 Dynamics

The dynamics of a system described by Hamiltonian mechanics is governed by Hamilton's equations, which relate the time derivatives of the generalized coordinates q and momenta p to the partial derivatives of the Hamiltonian function $H(q,p)$ in a system of $2D$ first order differential equations. Hamilton's equations are given by:

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} \tag{1.17}$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \tag{1.18}$$

where i ranges from 1 to D. Considering only the form of energy mentioned in eq. 1.16, this is equivalent to:

$$M\frac{d^2}{dt^2} = F(q) \tag{1.19}$$

Where $F(q)$ in this case is equivalent to $-\nabla U(q)$. This because when forces depends only on the positions, Newton's second law for mechanical system give rise to the differential equation in eq. 1.19 [4].
This can also be represented in a much more compact and direct representation in a matrix form. Let $q = (q_1, q_2, \ldots, q_d)$ and $p = (p_1, p_2, \ldots, p_d)$ be vectors of generalized coordinates and momenta, respectively. Then, the equations can be written as:

$$\frac{d}{dt}z = J\nabla H = \frac{d}{dt}\begin{pmatrix} q \\ p \end{pmatrix} = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}\begin{pmatrix} \nabla_q H \\ \nabla_p H \end{pmatrix}$$

where J is a anti-symmetrix matrix composed of I, the identity, off diagonal and zeros elsewhere. While $\nabla_q H$ and $\nabla_p H$ are the gradients of the Hamiltonian function with respect to q and p, respectively.
This express a flow in the a $2D$ phase space that in general terms can be express by a map φ which follows the group property of composition and inversion. Moreover it must leave the phase space invariant, hence the Jacobian determinant of the flow must be equal to 1 to respect the symplecticness [1] of the map.

$$\varphi_l \circ \varphi_m = \varphi_{l+m}, \qquad (\varphi_l)^{-1} = \varphi_{-l} \qquad det(J(\varphi)) = 1$$

Moreover those system of differential equation satisfy the reversibility under the momentum flip involution S:

$$S \circ \varphi_l = \varphi^{-1} \circ S$$

Which express the expected property of a dynamical system for which if (q_i, p_i) is the initial state of a system and (q_f, p_f) the final state after f units of time have elapsed, then the state $(q_f, -p_f)$ evolves in f units of time to the state $(q_l, -p_l)$.

[1] A map $\varphi : \mathbf{R}^{2D} \to \mathbf{R}^{2D}$ is said to be symplectic if at each point (q, p), $\varphi' J \varphi' = J$

Harmonic oscillator Example

The most canonical example is the harmonic, its energy is defined as:

$$H(q,p) = \frac{q^2}{2} + \frac{k}{m}p^2$$

For which the Hamiltonian dynamics:

$$\frac{d}{dt}q = wp \qquad \frac{d}{dt}p = -wq$$

Can be easily solved to get the flow which is

$$(q,p) = (Asinwt + Bcoswt, -Bsinwt + Acoswt)$$

.

1.4.2 Numerical methods for Hamiltonian Dynamics

The dynamics cannot always be calculated exactly, so numerical methods are often employed. These methods can be viewed as a map that, using a step size parameter dt and the function derivatives, iteratively determines the next step of the integration.

$$z_{n+1} = \psi(z_n) \tag{1.20}$$

With $z = (q,p)$ and ψ is the approximated map to integrate the Hamiltonian dynamics. Starting at time 0 and wait for a time T which correspond to f application of the map for a step of order dt, then ψ_{dt}^f must be an approximation of φ_T.
If $\varphi_{dt} - \varphi_{dt}^n = O(dt^{\nu+1})$ then the method is consistent of order $\nu \geq 1$. This mean that the difference between a single application and its flow on every point must be an big O of at least order 1.
If an integrator is consistent of order ν then for any starting point $q_0 = q(0)$, $|q_t - q(t)| = O(dt^{\nu})$.
Those techniques, of course, arise many problems like the error propagating through the integration and the stability of the orbit itself. Moreover it has the direct effect to increment the number of hyperparameters of the model since an optimal choice of the stepsize has to be taken and, as explained later, this is not a trivial problem.

Euler's Integrator

The Euler method is a simple numerical technique used to approximate solutions to ordinary differential equations (ODEs). It is based on the principle of approximating

the derivative of a function with a finite difference. Consider the first-order ODE:

$$\frac{dq}{dt} = f(q,t) \tag{1.21}$$

where q is the dependent variable and $f(q,t)$ is some given function.

The Euler method proceeds by discretizing the time interval into small steps of size dt. At each step, it approximates the derivative $\frac{dq}{dt}$ using the following forward difference formula:

$$\frac{dq}{dt} \approx \frac{q_{n+1} - q_n}{dt} \tag{1.22}$$

where q_n is the value of q at time t_n, and q_{n+1} is the value of q at time $t_{n+1} = t_n + h$. By rearranging the equation, we can solve for q_{n+1} to get the updating formula used in the Euler method:

$$q_{n+1} = \psi_{dt}(q_n) = q_n + dt \cdot f(q_n, t_n) \tag{1.23}$$

This updating formula allows us to iteratively compute q at each time step using the previous value of q and the derivative function $f(q,t)$.

The Euler method is a first-order method, meaning its local truncation error is of the order of $O(h^2)$. Consequently, for each step of size dt, the error in the approximation is proportional to dt^2. However, over many steps, the cumulative error can accumulate and lead to inaccuracies, particularly noticeable for stiff ODEs or when using large step sizes.
Moreover, the Euler method fails to preserve the properties of Hamiltonian flows. Its first-order nature leads to drifts in the Hamiltonian energy over time [4], and the integration is not reversible, as can be readily observed.

Stormer-Verlet method

In order to correctly simulate the flow of Hamilton's equations one must ensure that the map respect the property of the group and the space invariance as φ as described in 1.4.1.
Recalling the functional form of the integration as:

$$\frac{d}{dt}z = J\nabla(U(q) + K(p)) \tag{1.24}$$

Thus the Hamiltonian is separable and so are the integration:

$$\frac{d}{dt}q = \nabla K(p) = M^{-1}p \qquad \frac{d}{dt}p = -\nabla U(q)$$

Taking advantage of the independence [2] the integration can be split into $\psi_{dt}(q,p) = \psi^Q_{dt} \circ \psi^P_{dt}$.
Exploiting this mechanisms, the Stormer-Verlet method construct a symplectic integrator using only the first derivative of the system. In fact this method, also known as the Velocity-Verlet integrator, is a splitting numerical technique used to approximate solutions to ordinary differential equations (ODEs) that arise in classical mechanics. It is particularly useful for simulating the dynamics of particles in molecular dynamics simulations and other systems governed by Newton's equations of motion.

Consider the second-order ODE for a particle's position q and momentum q:

$$\frac{dq}{dt} = M^{-1}p \qquad \frac{dp}{dt} = -f(q)$$

where $f(q)$ represents the acceleration of the particle as a function of its position q, in our case the gradient of the potential energy $\nabla U(q)$.

The Stormer-Verlet method proceeds by discretizing the time interval into small steps of size dt. It dived the update steps in 3 performing $\psi^Q_{dt/2} \circ \psi^P_{dt/2} \circ \psi^Q_{dt/2}$:

$$q_{n+1/2} = q_n - \frac{dt}{2} M^{-1} p_n \tag{1.25}$$

$$p_{n+1} = p_n + dt \cdot \nabla U(q_{n+1/2}) \tag{1.26}$$

$$q_{n+1} = q_{n+1/2} - \frac{dt}{2} M^{-1} p_{n+1} \tag{1.27}$$

where $q_{n+1/2}$ is the position at the midpoint of the time step, p_{n+1} is the momentum at the next time step, and q_{n+1} is the position at the next time step.
The Stormer-Verlet method is a second-order method, meaning its local truncation error is on the order of $O(dt^3)$. The method is known for its stability and accuracy in simulating conservative systems over long time periods.

Stability study

If two numerical schemes are candidates to integrate an initial value problem for a given system, then the scheme that leads to smaller global errors for a given computational cost may seem more desirable.
Even though global errors may be bound as a $bigO(dt^\nu)$, in practice, it is almost always impossible, for the problem at hand, to estimate realistically the error constant.

[2] Independent in the sense that they do not compare explicitly together in the equation so they can be split in more parts, however their Lie brackets are not 0 otherwise no computational methods were needed [4]

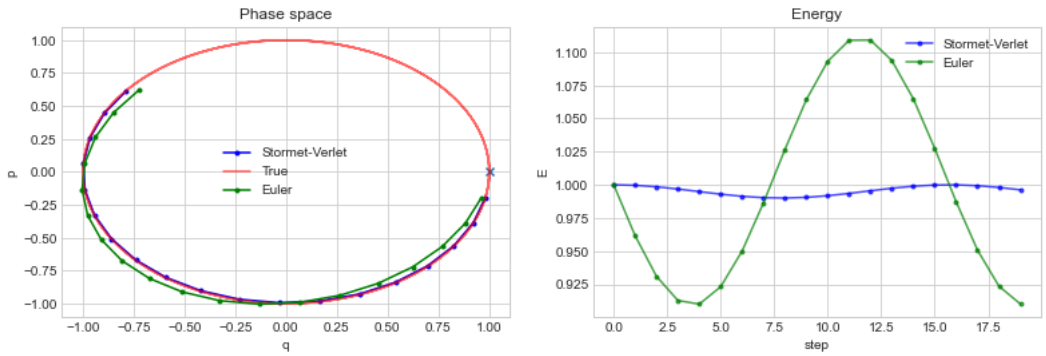

Figure 1.5: Left: Phase space integration compared to the true orbit. Right: numerical error during integration. The Stormer-Verlet method results are much more robust, as expected.

For this reason, the literature on numerical integrators has traditionally resorted to well-chosen problems where both the numerical and true solutions, q_n and $q(t_n)$, may be written down in closed form. The performance of the various integrators on the model may then be investigated analytically and it is taken as an indication of their performance when applied to realistic problems.
For this purpose the model of choice is the simple one dimensional Gaussian.
For a general one step integrator the map can be written in the form:

$$\frac{d}{dt}\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

.

In the Euler's case $A = D = 1$ and $B = -C = dt$ and the eigenvalues of the matrix can be easily found to be, in modulus, equal to $(1 + dt^2)^{1/2}$, hence, for every $dt > 0$, reapplication of the map will cause an increase of the phase space volume as can be appreciated in fig.1.5.

It is clear Euler's integrator is not suitable to integrate even the harmonic oscillator.
In the case of Stormer-Verlet the coefficients: $A = D = 1 - dt^2/2$, $B = dt - dt^3/4$ and $C = -dt$, for which the instability study of the eigenvalues fix a limit for the dt value to keep the orbit stable.
Symplectic integrators are powerful because the numerical trajectories they generate exactly preserve phase space volume, just like the Hamiltonian trajectories they are approximating. This incompressibility limits how much the error in the numerical trajectory can deviate from the energy of the exact trajectory. Consequently, the numerical trajectories cannot drift away from the exact energy level set, instead oscillating near it even for long integration times.

Chapter 2

Hamiltonian Monte Carlo

2.1 The Algorithm

Hamiltonian Monte Carlo (HMC) is a powerful Markov chain Monte Carlo (MCMC) method for sampling from probability distributions and have been extensively studied in literature [7], [5], [2], [3]. It relies on the principles of Hamiltonian dynamics described in the previous section 1.4.1.

In Hamiltonian dynamics, the state of a physical system is described by a set of canonical coordinates q and momenta p, collectively denoted as (q, p). The Hamiltonian function $H(q, p)$, also known as the total energy, characterizes the system's dynamics and is defined as the sum of the kinetic energy $K(p)$ and the potential energy $U(q)$:

$$H(q, p) = U(q) + K(p) \tag{2.1}$$

In our context, the potential energy is related to the target distribution of interest and auxiliary momentum variables p are introduced to explore the state space more efficiently. Those play the role of "propelling" the system through the state space, guided by the potential energy landscape defined by the target distribution. The mass matrix M controls the dynamics of the momenta and influences the behavior of the algorithm allowing the exploration of different energy levels.

In Bayesian inference, HMC is used to sample from the posterior distribution $\pi(q|D)$, where D represents observed data. In this context the posterior distribution is related to the potential energy by taking its negative logarithm:

$$U(q) = -log\pi(D|q) - log\pi(q) \tag{2.2}$$

At the same time, the kinetic energy $K(p)$ is typically picked to be quadratic and

the corresponding momenta p are sampled from a multivariate Gaussian distribution with 0 mean and covariance matrix M.

$$K(p) = \frac{1}{2}p^T M^{-1} p \tag{2.3}$$

With the Hamiltonian at hand, a new state is proposed by picking from a generic orbit generated by the system dynamics. However, the integration introduces unavoidable discretization errors which have to be taken into consideration. This is to ensure a correct convergence to the right distribution. In fact since error is unavoidable, when evaluating the orbits and their probabilities, the deviation from the correct orbit causes to virtually sample from the real distribution plus an error: $\pi(\tilde{q}, \tilde{p}) = \pi(q + \delta, p + \epsilon) = e^{-H(q+\delta, p+\epsilon)}$.

To summarize, the steps of a classical HMC are the following:

- Draw auxiliary momentum variable from a known distribution;
- Evolve the Hamiltonian to create an orbit J;
- Pick the next state accounting for the integration error;
- Set the updated position state to be next state in the Markov chain.

2.2 Kernel

Although the existing theory has demonstrated commendable results, advancing our understanding demands the exploration of alternative theoretical constructs or the integration of more sophisticated theoretical frameworks as presented in the novel work of [9].
The method described at the beginning of chapter 2 can be formalized as a general Kernel that acts in four steps:
given the starting point, (i) the first step is to sample a new kinetic energy, (ii) select an orbit J which of course contains the starting point and (iii) a particular state along the orbit has to be chosen as far as possible from the starting point to ensure minimum correlation, (iv) project the state in the phase space onto the coordinates. Those steps are to be taken taking care of maintaining the target distribution invariant and respecting ergodicity.
In this view, given $z_0 = (q_0, p_0)$, the orbit as the ordered collection and j indexing the orbit, a general Kernel can be defied as:

$$\tilde{K}(., q_0) = \int dp_0 \pi(p_0) K(., q_0, p_0) \tag{2.4}$$

with K as:

$$K(.,z_0) = \sum_{J\subset\mathbb{Z}} \sum_{j\in J} O(J|z_0) I(j|J,z_0) \delta_{projq} \tag{2.5}$$

Where $O(J|z_0)$ and $I(j|J,z_0)$ have been introduced. $O(J|z_0)$ represents the probability of visiting the orbit J, containing the starting point z_0 and whose elements are represented by ordered indices around the starting point $J = \{-k,..,0,..,l\}$:

$$O = O(.|z) : z \in \mathbb{R}^2 \,. \tag{2.6}$$

$I(j|J,z_0)$ represents the transition probability from the state z_0 to every other state within the orbit J, corresponding to points in the phase space $j = (q_j, p_j)$:

$$I = I(.|J,z) : z \in \mathbb{R}^2 \,. \tag{2.7}$$

The general algorithm can then be deconstructed in:
1 - select p with probability $G(0,M)$
2 - Select an orbit J with probability $O(J|z_0)$
3 - select a point z_1 from the orbit J with probability $I(z_1|J,z_0)$
4 - keep q_1 as the projection of z_1

2.2.1 Invariance

To prove the invariance of a transition kernel K, we must prove that

$$\pi(q') = \int dq\pi(q)\tilde{K}(q',q) = \int dq\pi(q) \int dp\pi(p) K(q',p'|q,p) \,. \tag{2.8}$$

Hence,

$$\pi(q') = \int dqdp\pi(q,p) \sum_{J\subset\mathbb{Z}} \sum_{j\in J} O(J|q,p) I(j=q',p'|J,q,p) \delta_{proj-p'} \,, \tag{2.9}$$

where $\pi(q,p) = \pi(z) = e^{-H(z)}$.
This is equivalent to saying that reapplication of the kernel do not change the final target distribution, thus the system is stationary.
In [9] a comprehensive formal demonstration has been conducted, imposing a universal rule for invariance on the kernel. In fact if O and I satisfy:

$$\pi(z_0) O(J|z_0) = \sum_{j\in J} \mathbb{I}_{J-j}(j) \pi(z_j) O(J-j|z_j) I(j|J-j,z_j) \,, \tag{2.10}$$

then the transition kernel K leaves the target distribution $\pi(q)$ invariant. It is worth noting, as depicted in fig.2.1, that $O(J|z_0)$ and $O(J-j|z_j)$ represent the same orbit.

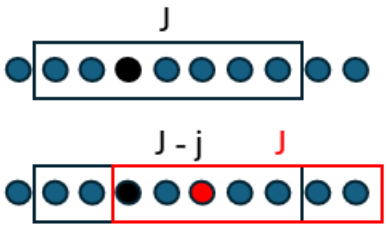

Figure 2.1: Visual representation to show the equivalence of the orbits defined by $O(J|z_0)$ and $O(J-j|z_j)$.

Hence this equilibrium condition express a property that has to be valid only on a local level since the equality refers to the same Hamiltonian flow.

2.2.2 Orbit selection

In this general approach, a the step from z_0 to z_l is characterized by:

$$K(z_l, z_0) = \sum_J I(z_l|J, z_0)O(J|z_0)$$

. which, given z_l, expresses the probability to end up in this state. This is the sum of the probabilities to be in any other position and generating the orbit J which includes z_l times the actual probability of choosing z_l among the orbit.
In case of symmetry for the orbit selection kernel O : $O(J|z_0) = O(J-j|z_j)$ a stronger condition for the invariance can be obtained. In fact, since they actually refer to the same orbit, one can fix a particular path $\hat{J}$ and this reduces to the condition:

$$\mathbb{I}(z_0 \in \hat{J})O(\hat{J}|z) = \mathbb{I}(z_l \in \hat{J})O(\hat{J}|z)\,. \tag{2.11}$$

Formally, it is required that the probability of transitions to a trajectory is the same regardless of the state in which the trajectory begun.
This implies that the algorithm's invariance is ensured solely by the invariance of the selection index:

$$\pi(z_0) = \sum_{j\in\mathbb{Z}} \mathbb{I}_{J-j}(0)\pi(z_j)I(j|J-j, z_j)\,. \tag{2.12}$$

Beside respecting condition in eq. 2.11, nothing has been said on the form the selection orbit kernel must have. However in order to respect this local requirement, one must only assure that the orbit has no favorite paths. Due to the random resampling of the momenta, there are no favourite directions that the sampler is forced

to take, hence the validity of eq. 2.11 is respected as long as the orbit generation is not biased in some region of the space.

Static HMC

The HMC algorithms in which the orbit selection kernel keeps the number of integration steps L fixed through all algorithm are called *static*.
This can be achieved in several ways. For instance, one can directly integrate the flow L times forward in time. Alternatively, one can decide after each step if the evolution should proceed forwards or backwards in time.
Another approach, given L steps, is to double the number of points to integrate at every step. This procedure generates a random ordered tree where at every step it doubles, adding a new equally sized tree ordered with the old one. Given L it will produce an orbit the size of 2^L
As it will be clear later, this helps with the implementation of more efficient algorithms.
The reversibility condition is most easily satisfied by uniformly sampling over all of trajectories that contain the initial point or sampling with some probability the last point of the orbit.

Dynamic HMC

Regardless of the level of optimization applied to a static scheme, it is inherently limited. Except for a few exceedingly simple scenarios, different energy levels will require varying integration times to achieve equivalent exploration efficacy [2]. While some sets may be adequately surveyed with short integration periods, others may demand considerably longer time. To ensure optimal performance of the HMC, dynamic implementations are essential. These implementations should have the capability to automatically adjust integration times to approximate uniform exploration across all energy level sets.
Once a criterion has been selected to determine the sufficient length of a trajectory for satisfactory exploration of the corresponding energy level set, the process unfolds as follows: an initial trajectory of a specified length is constructed, after which the termination criterion is evaluated. If the criterion is not satisfied, the trajectory is extended, and the process is repeated. However, if the criterion is met, the trajectory along with its corresponding sample is returned.
In this approach, to ensure symmetry of the orbit selection kernel, when adding a new orbit, one must carefully propose only those candidates capable of reproducing

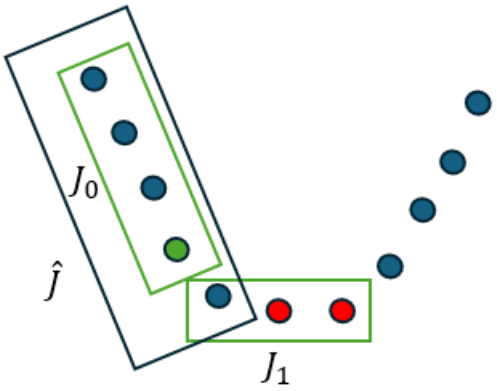

Figure 2.2: In a dynamic implementation that utilizes a termination criterion, all points along the trajectory that fail to reconstruct the original orbit, due to an early stop, must be excluded.

the orbit under consideration without meeting the stopping criteria. All the points in the new trajectory that would lead to an early stopping criteria, should be not considered as next candidates, as depicted in fig. 2.2.

This means that if an orbit J_0 in fig. 2.2 is considered and the expansion through the new orbit J_1 is made. If the point which met the stopping criterion (or subsequent points) are considered as valid proposals (red points), then starting from one of those points will end up in an early stop, failing to generate the wanted orbit, hence violating the condition: $\hat{J}$, $O(\hat{J}|z_0) \neq O(\hat{J}|z_{red})$.
In other words, in order to respect the symmetry condition in a dynamical implementation, all the points that cannot generate the orbit in consideration, must be excluded. This implies that the integration of further points is a waste of computation and the exploration can be early stopped.

The renowned method for such a stopping criterion is detailed by [13], where the exploration halts when either end of the trajectory, denoted as $z_+ = (q_+, p_+), z_- = (q_-, p_-)$ respect:

$$\begin{cases} p_+(q_+ - q_-) < 0 \\ p_-(q_- - q+) < 0\,. \end{cases}$$

This condition is fulfilled exclusively when the momentum at both ends of the trajectory are anti-aligned along the line connecting the positions at each boundary. In such instances, it is anticipated that further integration in either direction would result in the ends of the trajectory drawing closer together, a phenomenon typically observed only when the trajectory has already extended across the width of the energy level set.

In this thesis, we propose another method which takes advantage of the same consideration but whose implementation is easier and more robust against the integration noise. The method is based on the following observation: the momentum vector from the starting point will either evolve forward or backwards. Compared to the starting point, p_+ and p_- are parallel but pointing towards opposite directions. When the angle between these 2 vectors will be smaller than 90 degrees the orbit will start to fold onto itself. Hence the evolution will be continued dynamically as long as the following condition is respected:

$$p_- p_+ < 0\,.$$

2.2.3 Index selection

Once the orbit has been generated taking care on the reversibility condition, the next state has to be chosen among all the J points in the orbit.
This has to be done trying to be as far away as possible from the initial point to lower autocorrelation, but at the same time respecting the invariance of the target distribution, $\pi(z_0) = \sum_{j\in\mathbb{Z}} \mathbb{I}_{J+j}(0)\pi(z_{-j})I(j|J+j, z_{-j})$.
In this perspective, employing a Metropolis-Hastings update on the last visited state, as proposed in the simpler version of HMC as described in [5], appears suboptimal as all the computation is performed only to evaluate one state. To enhance efficiency, more effective solutions involve considering proposals across windows or the entire length of the orbit.
In general, to respect the equality one has to generate a process which yields transitions proportional to the canonical densities of the orbit ensemble:

$$I(z_l|J) = \frac{\pi(z_l)}{\sum_j \pi(z_j)}\mathbb{I}(z_l \in J)\,. \tag{2.13}$$

This can be done in several ways, as is discussed henceforth.

Slice Sampling

As discussed in [16] and used on the first implementation of the original NUTS algorithm [13], slice sampling provides a procedure to achieve the desired transition probability.
Given the the initial state z_0 and a function f proportional to the target distribution $\pi(z_0)$, the space is augmented with a supplementary variable u, for which the joint

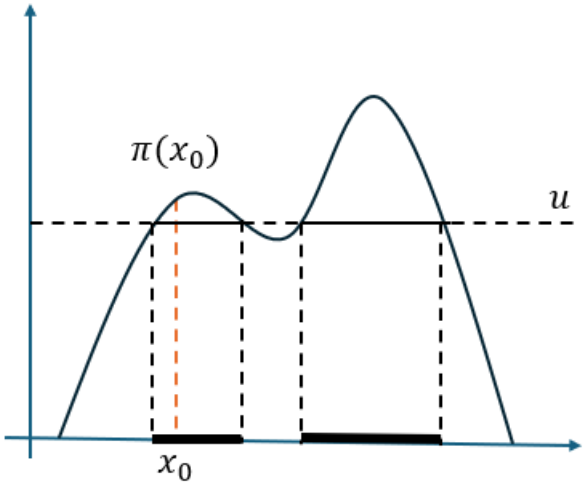

Figure 2.3: Graphical representation of the slice sampling procedure: Given the current point x_0 with probability $\pi(x_0)$. A random draw in $[0 - \pi(x_0)]$ is performed to sample the slice variable u. The slice defines a sub-space conditioned on $\pi(x) > u$, denoted as the dark area projected on the x plane. A uniform random draw from this sub-space leaves the target distribution invariant.

distribution is defined to be uniform:

$$\pi(z_0, u) = \begin{cases} 1/Z & \text{if } 0 < u < f(z_0), \\ 0 & \text{otherwise} \end{cases}$$

where $Z = \int \pi(z)dz$, so that the marginal density for z retrieves the target distribution, since $\pi(z) = \int_0^{f(z)} \frac{1}{Z} du = \frac{f(z)}{Z}$ as desired.
In order to effectively perform such sampling, given the initial state z_0, the procedure is the following:

- Draw uniformly $0 < u < f(z_0)$, hence defining a slice $S = \{z : f(z) > u\}$
- Find an interval which contain all or much of the slice, assuring that the initial point belongs to that interval.
- Draw uniformly the new point from the slice interval.

The slice defines a subspace (the dark area in the x-axis in fig. 2.3) which is uniform on the joint probability distribution with the slice by construction. So, as long as the proposed interval comprehend all the relevant space containing the subsample respecting the slice, a random draw over those state will yield the correct transition probability.
Many procedure have been applied in order to create a satisfying interval containing much of the slice. However in this context the proportionality function is $f(z) = e^{-H(z)}$ and the interval is defined by the orbit explored by the Hamiltonian evolution.

To ensure accurate calculations of the quantity of interest, it's important to take certain precautions. This is because, generally, dealing with exponentials on computers often leads to underflow or overflow, making the computation ineffective. Hence in the program is always better to work with their logarithms. It is easy to demonstrate that the uniform draw for the slice variable, is equivalent to sample from an exponential distribution with rate 1, minus a shift proportional to $f(z_0)$. This is because if $u \sim U[0, e^{-H(z_0)}]$ for which the slice is defined by $\{z : u < e^{-H(z)}\}$ then carefully transforming the probability $\hat{u} \sim exp(\hat{u}; 1) - H(z_0)$ and the slice is then defined by $\{z : H(z) < \hat{u} - H(z_0)\}$, which is much more stable to calculate on a computer.

Multinomial Sampling

A more direct approach uses a multinomial sampling proportional to the canonical densities of the orbit. Hence, defining $\pi(z_j) = e^{-H(z_j)}$, the index selection kernel is given by:

$$I(z_l|J) = \frac{e^{-H(z_l)}}{\sum_j e^{-H(z_j)}} \tag{2.14}$$

Other than its simplicity, the absence of an auxiliary variable can be shown to generate better performance [3]. It can be shown by direct calculation that this selection kernel preserve the target probability. Recalling that in case of symmetry of the Orbit kernel, the invariance of the algorithm is assured by the index selection kernel as in eq. 2.12. Fixing the orbit $\hat{J}$, containing the initial point and since the index selection does not depend on the starting point, the invariance:

$$\pi(z) = \sum_{z' \in \hat{J}} \pi(z') I(z|\hat{J}) = \sum_{z' \in \hat{J}} \pi(z') \frac{e^{-H(z)}}{\sum_{z''} e^{-H(z'')}} = e^{-H(z)} = \pi(z)$$

Since the two independent sums are equivalent.
However, this method can cause instabilities when implemented on a real computer due to underflow and overflow issues caused by the exponential calculations. In order to adjust those instabilities one can recur to the log-sum-exp trick. In fact eq. 2.14 can be re-arranged as:

$$I(z_l|J) = exp[-H(z_l) - log \sum_j exp[-H(z_j)]] \tag{2.15}$$

Where the last term can be rewritten including an arbitrary constant c:

$$log \sum_j exp[-H(z_j)] = c + log \sum_j exp[-H(z_j) - c] \tag{2.16}$$

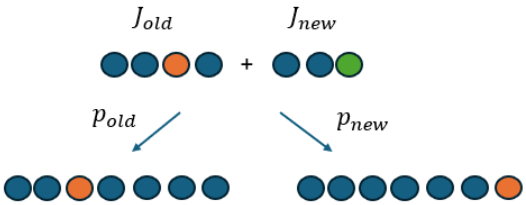

Figure 2.4: Graphical representation of the union of the current orbit with the new one. This can be interpreted as a Bernoullian event where the probability reflects the canonical density of the total trajectory. The orange color represents the currently selected state of the old trajectory, while the green color represents the proposed point for the new trajectory. The update process is balanced by the ensemble probabilities for each orbit.

The arbitrary constant is chosen to keep the exponential within a range that ensures effective computation. This means setting the value to either the maximum or minimum of the total orbit, depending on whether the problem is experiencing overflow or underflow. However, since the energy is preserved up to an error factor (which should be of the order of the integration step, $O(dt^2)$, due to the symplectic nature of the second order integrator described in Sec. 1.4.2), the constant is chosen to be the initial energy, $c = -H(z_0)$.

2.2.4 Efficient implementation

When dealing with proposals over the entire trajectory, the computational issue that arises is the necessity of storing the entire trajectory in memory to generate the transition. For static trajectories of length L, this means constantly retaining L states in memory, which can be cumbersome, particularly in high-dimensional problems. To significantly enhance efficiency, and resources one can adopt the following approach: instead of sampling an entire trajectory, generate a random trajectory sequentially. For instance, initially, a trajectory containing the initial point, called J_{old}, is considered. Subsequently, a new trajectory, J_{new}, is appended by randomly integrating either forwards or backwards in time. The combination of these two component trajectories forms the complete trajectory, $J = J_{new} \cup J_{old}$.

If for each J_{new} and J_{old} the chosen state was given by some selection kernel as described in sec. 2.2.3, namely z_{new}, z_{old}, then the final state is generated from a Bernoulli event that takes z_{new} with probability p_{new} and z_{old} otherwise.
With p_{new} defined as:

$$p_{new} = \frac{\sum_{z \in J_{new}} \pi(z)}{\sum_{z \in J_{new}} \pi(z) + \sum_{z \in J_{old}} \pi(z)} \tag{2.17}$$

Hence, this process can be described by the following proposal:

$$I(z|J) = p_{new} I(z|J_{new}) + p_{old} I(z|J_{old}) \tag{2.18}$$

By direct computation can be shown that this preserve the form of the original selection kernel $I(z|J)$, in fact:

$$p_{new} I(z|J_{new}) + p_{old} I(z|J_{old}) =$$

$$= \frac{\sum_{z \in J_{new}} \pi(z) I(z|J_{new}) \mathbb{I}(z \in J_{new})}{\sum_{z \in J_{new}} \pi(z) + \sum_{z \in J_{old}} \pi(z)} + \frac{\sum_{z \in J_{old}} \pi(z) I(z|J_{old}) \mathbb{I}(z \in J_{old})}{\sum_{z \in J_{new}} \pi(z) + \sum_{z \in J_{old}} \pi(z)} =$$

$$= \frac{\sum_{z \in J_{new} \cup J_{old}} \pi(z) I(z|J_{new} \cup J_{old})}{\sum_{z \in J_{new} \cup J_{old}} \pi(z)} \mathbb{I}(z \in J_{new} \cup J_{old}) = I(z|J)$$

This progressive sampling allows to generate a sample as the trajectory is explored, keeping only a few states in memory at any given time and its implementation can be achieved with the help of the log-sum-exp trick described in the previous section Sec. 2.2.3.

Different exploration methods can be applied, starting with the initial point as the active sample one could repeatedly expand the trajectory by integrating one step forward or backwards in time and then update the active sample using the transition above as shown in fig 2.4. Furthermore, by using the doubling procedure described in sec.2.3.3, both the old and new trajectory components at every iteration are equivalent. This allows to generate a tree recursively, propagating the state as the tree is building up as discussed later on in the NUTS algorithm.

2.2.5 Optimal Kinetic Energy

The momenta parameters are sampled each time from a known distribution, usually Gaussian. It is interesting to ask how this choice impacts the sampling of the target distribution.

One of the distinctive properties of a Hamiltonian flow is that it preserves the Hamiltonian itself. This implies that each Hamiltonian trajectory is confined to a level set. The total energy can be seen as the marginal density π_H and the microcanonical distribution over the level set $\pi(q,p)_H$ [2] which is the probability of picking particular energy level set times the distribution of the conjugate momenta on those levels.

$$\pi_{H(q,p)} = \pi_H \pi(q,p)_H \tag{2.19}$$

To quantify the efficacy of the momentum resampling, consider $\pi_{H|q}$, the distribution of energies E, induced by a momentum resampling at position q. The closer

this distribution is to the marginal energy distribution for any q, the faster the random walk will explore energies and the smaller the autocorrelations will be [3].
Optimal performance is achieved when the momentum-resampling induced energy distributions are uniformly equal to the marginal energy distribution so all relevant energy level are explored by the algorithm:

$$log(\frac{d\pi_{H_i|q}}{d\pi_{H_i}}) = 0, \quad \forall q\,.$$

Consequently, one can quantify the consistency of this two distributions and tune accordingly the momenta distribution. However, this is unfeasible in any nontrivial problem.
In practice, a criterion that is readily evaluated using the Hamiltonian Markov chain itself is needed. One theoretically appealing choice is the Bayesian fraction of missing information [3]:

$$BFMI = \frac{\mathbb{E}_{\pi_{H(q,p)}}[Var_{\pi_{H|q}}(H|q)]}{Var_H(E)} \tag{2.20}$$

which quantifies how insufficient the energy variation induced by the momentum resampling is: in the worst case BFMI $\to 0$ and the momentum resampling induces very slow exploration across the level sets, while in the best case BFMI $\to 1$ and the momentum resampling effectively generates exact draws from the marginal energy distribution.
Because the momentum resampling does not change the position, q, the BMFI can also be interpreted as a function of ΔE instead. Since $\Delta E = \Delta K$, with ΔK the change in kinetic energy that only depends on the way the momentum variables are embedded on the space i.e, the distribution it is sampled from.
Hence, we can readily estimate the BFMI using the history of energies in the Hamiltonian Markov chain:

$$\hat{BMFI} = \frac{\sum(E_n - E_{n-1})^2}{\sum(E_n - \bar{E})^2}\,. \tag{2.21}$$

However, this method does not provide information about the magnitude of error in momentum resampling, as the required variation depends on the properties of the space. A low BMFI only indicates undersampling of momenta but does not provide guidance on how to correct it accurately. Additionally, it necessitates running the algorithm itself, so it cannot be effectively used to determine the optimal value, but rather serves as a posteriori check.

2.2.6 Optimal time step

The time step chosen for the integrator is an essential hyperparameter. Its value influences the behaviour of the algorithm drastically. A too large step size results in high integration errors and consequently high rejection rate, a too small step size causes the algorithm to slow the dynamics, and the integration is dominated by random fluctuation. As a result, the next proposed state will be very close to the starting point, wasting all computing power. Moreover, the optimal step size depends, in principle, on the current position: since the approximated integration is gradient-based, depending on the geometry of the local parameter space the required integration time should be adapted to the local steepness for a correct approximation. This issue is also more subtle than it seems because even by adapting the integration time based on the position could not ensure the symmetry of the orbit selection kernel, as required in eq. 2.11.
All of those aspect become really problematic when the number of dimensions rises for what is know as the *curse of dimensionality*. This issue causes the enclosure of the typical set to smaller subspaces as the dimension increase. To understand this process [3] proposed a simple example which make clear this phenomenon. Consider a 1 dimensional parameter space and divide it in 3 parts. Knowing that the mode of the distribution lies on 1 of the 3 subsets, the probability of exploring a relevant portion of the parameter space is 1/3. The same process extending the number of dimension to two, creates a square divided in 9, a cube in 27 and so on. Although HMCs have been devised to overcome this problem when the number of dimensions rises, the integration might become cumbersome. Exploiting the fact that the mean average error for the integration is strictly positive [4], there is a power law relation between the integration time step and the number of dimensions:

$$dt = lD^{-1/2\nu} \tag{2.22}$$

with ν the order of the integrator. These results rely on various approximation and assumption and so should be taken with a pinch of salt, thus as a guideline for the general behaviour that will be used in the next sections.

2.3 HMC algorithms

In the following, an overview on the most common gradient based samplers. Their theoretical foundation is discussed and the pseudocode is shown for all the algorithms in Appendix A. Moreover their `python` implementations are available at my github page: https://github.com/killua1219/HMC.

We will begin discussing what is not a HMC algorithm but rather a Brownian motion with a gradient-based drift. This algorithm has settled to be the theoretical foundations of the HMC sampler. Next simple HMC and NUTS algorithms are explored as well as some relevant variations regarding the Hamiltonian transformation of the space. Finally, we present our novel algorithm based on all the concepts previously explained.

2.3.1 Metropolis adjusted Langevin algorithm

As proposed in [12], [18], Metropolis adjusted Langevin algorithm (MALA) has been intensively studied and constitute a first approach to a HMC, since the random walk from the Metropolis-Hastings algorithm is augmented with the information from the gradient of the space coordinates, as in the HMC, to simulate for each step a diffusion Langevin equation.

$$\frac{dq}{dt} = \frac{1}{2}\nabla_q L(D|q) + \nu(t) \tag{2.23}$$

where $\nu(t)$ represents a noise term that can be assumed to be $< \nu(t) >= 0$ and $< \nu(t), \nu(t') >= \delta(t-t')M^{-1}$. Defining an integration step ϵ, the Langevin equation can be integrated to get:

$$q_1 = q_0 + \frac{\epsilon}{2}\nabla_q L(D|q) + \sqrt{\epsilon}p \tag{2.24}$$

Where $p \sim G(0, M)$ is the random step. Next, the proposal is accepted through a Metropolis-Hastings acceptance rule:

- find q_1 through: $\frac{dq}{dt} = \frac{1}{2}\nabla_q L(D|q) + \nu(t)$
- accept with probability $\alpha = \{min(1, \frac{\pi(q_1)}{\pi(q_0)}\}$

So a random drawn for the momentum, which corresponds to the noise term of the Langevin equation, is followed by a single update. Even though this kind of algorithm helps improve the pathology of random walk, it does not help much with autocorrelation.
Note that the improvement from a simple Metropolis-Hastings is the presence of drift term given by the gradient of $L(D|q)$ which is the only difference to the original algorithm.

2.3.2 Classic Hamiltonian Monte Carlo

Hamiltonian Monte Carlo algorithms as presented in [5] made their first appearance with a static implementation, in the sense defined on sec. 2.2.2, using L fixed steps for the orbit generation. Then the error for the integration is corrected though a

Metropolis-Hastings acceptance.
Given q_0, a symplectic one step integrator ψ_{dt} and the number of step L, one step of the HMC is structured as follows:

- draw p_0 from $G(0, M)$
- Evolve for L steps the Hamiltonian: $(q_L, p_L) = \psi^L(q_0, p_0)$
- accept the new state with probability $\alpha = max(1, exp(-H(q_L, -p_L)) - exp(H(q_0, p_0)))$

In this view it is clear how the MALA algorithm is improved by the integration over more steps in order to propose a state far away from the current one. However the efficacy of classical HMC are strictly related to the hyper-parameters at hand, namely the number of steps L and the mass matrix M.
By default the mass is set to be the identity matrix but, in this simplistic view, important features of the space can get lost. In principle, the mass matrix can be adjusted after a first run by looking at the BFMI. It could, however, be extremely computationally expensive and no information on how to modify those values are available.
To give an idea of the nature of this issue it is sufficient to sample from a 2D Gaussian distribution with 0 mean and known unit covariance matrix. Setting the path length at 40 and the time step at 0.01, the results are highly dependent on the choice of the mass matrix.

This is demonstrated in fig.2.5: different mass matrices lead to different results. The two mass matrices differ by 2 orders of magnitude $M_1 = I$, $M_2 = 0.01 * I$ in order to enchant the effect on the $\hat{BFMI}$. As expected resultant values are: 0.97 for the first and 0.19 for the second.
It is clear how in the second case the variation in energy, during resampling, were not wide enough to explore correctly the typical set and instead concentrating in the narrow high probability space compromising the ergodicity of the algorithm.
For what concern the path length L there are no apriori principles to set a fixed integration length that will optimally explore every given energy level.
Returning to the previous example, when the correct mass matrix is used, the Integrated Autocorrelation Time (IAT) is approximately 13. However, under the same conditions, reducing the number L of integration steps to 15 significantly increases the IAT to around 64, resulting in severely different outcomes. This can be observed in the posterior plot in Fig. 2.6, which clearly shows that a stationary distribution is not achieved in this setting.

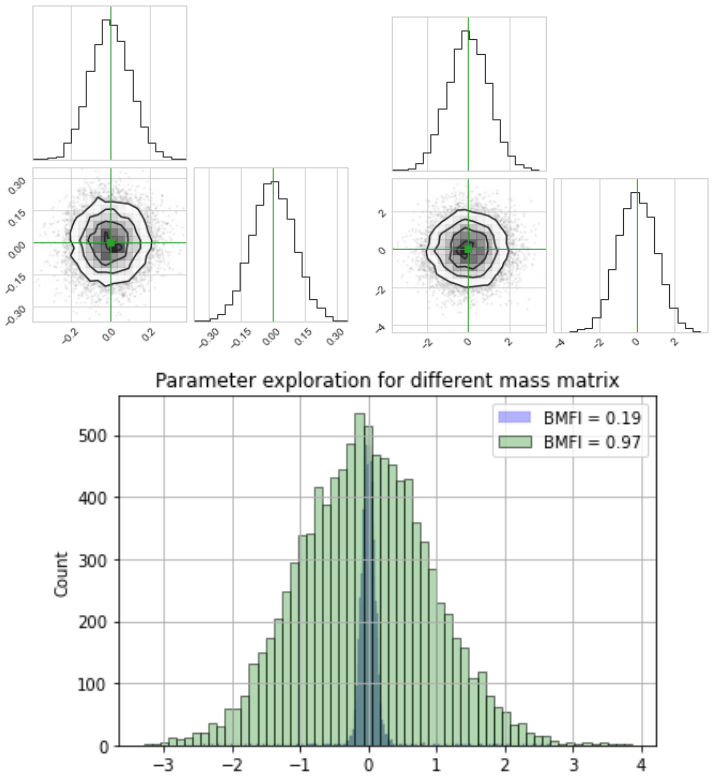

Figure 2.5: Comparing plots between different hyperparameter settings: 10000 iterations, $L = 40$, with different mass matrices M. Left (greenGreen): $M = I$. Right (blueBlue): $M = 0.01I$.

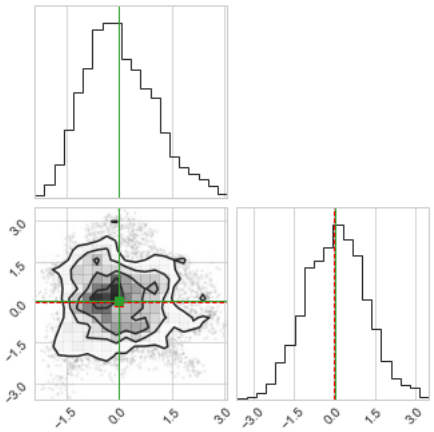

Figure 2.6: Posterior plot for the 2d Gaussian distribution using the HMC when the number of integration steps is reduced to 15.

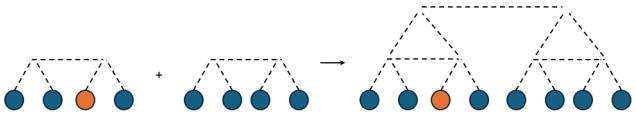

Figure 2.7: Graphical representation of the doubling procedure proposed by [13]. Given a tree, the next step is to double the iteration creating an exact new tree either forward or backwards in time. This generate a perfectly ordered tree that can be constructed by recursion.

2.3.3 NUTS

NUTS implemented in [13] overcome the problem of finding optimal integration length and at the same time take into consideration all the aspect considered so far. It is, in fact, a dynamic HMC algorithm since integrate the space until the realization of the stopping criterion explained in eq. 2.2.2. It is cleverly built to sustain a recursive and efficient implementation where the state is chosen through a slice sampling among the orbit.
After sampling the momentum at the beginning of the kernel, it proceeds by randomly choosing a direction, either backward or forward in time, and evolves the dynamics by doubling the integration steps each time. This process creates a perfectly ordered tree, with each subsequent integration forming equal trees to be concatenated..

This allow to recursively build a tree of size 2^k (depending on the current step k) proposing new states with the slice sampling procedure and updating the new propose state among the identical binary trees as defined in eq.2.17. To understand how this procedure works, consider the update depicted in Fig. 2.7. In this example, the orde $k = 2$ defines a new tree with $2^k = 4$ new points. Rather than constructing the entire tree directly, it can be viewed as a collection of smaller trees, in this case, 2 binary trees. EEach binary tree is computed sequentially, progressively using the slice sample to choose a new proposal state. They are then joined sequentially, respecting the order of integration, and only the selected state is saved in memory using the usual updating procedure described in eq. 2.17. This process is repeated until the stopping criteria in eq. 2.2.2 is not satisfied.
With this process, only $O(k)$states need to be kept in memory to update a new tree, instead of $O(2^k)$ states without the efficient implementation.
The pseudocode presented in Appendix A.4 focus to replicate the code proposed by [13] aiming to preserve its property and adjust some coding aspect from the version there presented.
It is worth noting how the inversion check has to be done always before the update to respect the Dynamic reversibility of the HMC that would provoke the early stop of the branch of the tree as discussed in Sec. 2.2.3. In fact starting from any of

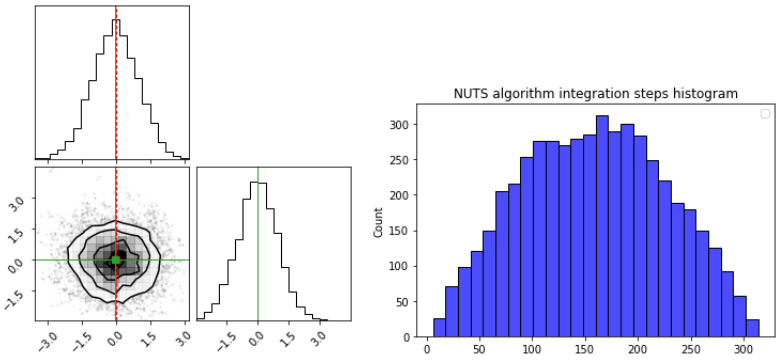

Figure 2.8: NUTS algorithm run 5000 times to sample a 2D Gaussian distribution. Left: histograms of the posteriors. Right: Histogram of number of steps performed for each iteration

those point would not reproduce the orbit in consideration.
This algorithm significantly improve the overall performances of the sampler due its ability to effectively explore most of the space adaptively for each energy set.
This can be appreciated by displaying the number of integration steps performed for each iteration of the algorithm. In fact even for a simple Gaussian distribution the optimal number of steps is wide spread as depicted in the right panel of Fig. 2.8, as depending on the current energy level, different length are needed. In fact the same example from the previous scenarios is used, i.e., the $2D$ Gaussian distribution. Maintaining the same settings for the timestep and mass matrix, results in a much lower IAT (approximately 2) compared to the HMC algorithm which was around 13.

2.4 Different energy function

Numerous methods for enhancing Hamiltonian Monte Carlo capabilities involve altering the Hamiltonian function itself. However, such modifications must be executed in a manner that maintains the desired property of accurately recovering the target distribution upon marginalizing out the induced momentum variables. Additionally, any adjustments made to the Hamiltonian needs a verification on the symplectic nature of the integrator.

2.4.1 Riemannian manifold Hamiltonian Monte Carlo

The introduced algorithm – Riemannian manifold Hamiltonian Monte Carlo (RMHMC) – can be seen as an extension of Hamiltonian Monte Carlo where the local geometry of the distribution is taken into account through the metric tensor M(q) [11], [17]

which need to be hermitian and positive definite.
The Hamiltonian becomes:

$$H(q,p) = U(q) + \frac{1}{2}p^T M(q)^{-1}p + \frac{1}{2}log(det(M(q))) \tag{2.25}$$

The complication arising from this trick is to generate an Hamiltonian which is not separable anymore since now p is distributed according to $N(p; 0, M(q))$, displaying a dependencies on the position variables.
The Stormer-Verlet integrator does not preserve the volume anymore hence a more suitable method is required.
The most common choice adopted in literature is the semi-explicit generalized leapfrog integrator which reduces to the Stormer-Verlet in case the Hamiltonian is separable:

$$L = \begin{cases} p_{1/2} = p_0 - \frac{dt}{2}\nabla_q H(q_0, p_{1/2}) \\ q_1 = q_0 + \frac{dt}{2}\nabla_p [H(q_0, p_{1/2}) + H(q_1, p_{1/2})] \\ p_1 = p_{1/2} - \frac{dt}{2}\nabla_q H(q_1, p_{1/2}) \end{cases} . \tag{2.26}$$

The first two equations are implicit, requiring interpolation to solve them. Additionally, more computational effort is needed to calculate the derivative itself. Further complications arise by the complexity of the Hamilton equations that must be solved:

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} = \{M(q)^{-1}p\}_i \tag{2.27}$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} = -\frac{\partial U}{\partial q_i} - \frac{1}{2}tr\{M(q)^{-1}\frac{\partial M(q)}{\partial q_i}\} + \frac{1}{2}p^T M(q)^{-1}\frac{\partial M(q)}{\partial q_i}M(q)^{-1}p \tag{2.28}$$

For each dimension, traces and matrix multiplications and differentiation must be performed, which can become much more computationally expensive. Faster computations can be achieved when the mass matrix in question is diagonal and the eq. 2.27 - 2.28 takes the much simpler form:

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = M(q)^{-1} \circ p \tag{2.29}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q} - \frac{1}{2}M(q)^{-1} \circ \frac{\partial M(q)}{\partial q} + \frac{1}{2}p^T \circ M(q)^{-1} \circ \frac{\partial M(q)}{\partial q} \circ M(q)^{-1} \circ p \tag{2.30}$$

Where the $\circ$ denote the much simpler and faster Hadamard product.

Given the understanding of the mechanism and how to handle this algorithm, it also raises the question of which metric would be the most suitable for any given problem. The optimal choice would theoretically be the Fisher information matrix,

but it is often impractical to compute the expected value. Additionally, any approximation would be susceptible to noise fluctuations, potentially causing divergences in the flow.
As proposed by [11] an informative matrix is the Hessian of the potential energy, allowing a navigation guided by the shape of the surroundings. However a direct application of such metric would inevitably lead to singularities due to invalid proposal for the matrix, yielding negative eigenvalues for example. To solve this issue a SoftAbs map to smooth and regularize the Hessian could be applied, however this method would require anyway all the first, second and third derivatives of the system for a huge computational need. Even though [1] proposed a sophisticated method, it still takes $O(D^3)$ at each Hamiltonian step and can make the integration process extremely long for high dimensional problems.

2.4.2 Quantum Inspired HMC

The Quantum Inspired HMC method [14] aim to overcome multimodality of the space through a mass randomization process inspired by quantum mechanics.
Knowing the energy-time uncertainty relation from quantum mechanics, QIHMC sets M at random, with a probability distribution which is a hyperparameter of the system. In the usual setting the mass matrix M is usually chosen as diagonal such that $M = mI$ and further the scalar mass m is commonly set as 1. Such a choice corresponds to the classical physics where the mass is scalar and deterministic. In the quantum physics, however, the energy and time obeys the uncertainty relationship $\Delta E \Delta t \sim h$ indicating that a quantum particle can have a random mass obeying a distribution $\pi(M)$ which is independent from position and momentum.
Recalling the form of the Stormet-Verlet integrator defined in 1.4.2, the stability of the numerical integration can be regularized by employing a large enough mass and a small step size balancing out those effects for a optimal performance. However with fixed settings, in a multimodal space with many local minima, the momentum resampling could get stuck in one of those minima. For example in Fig. 2.9, if the re-sample of the momenta allows the system to move at max to energy level H_1, this would compromise the ergodicity of the algorithm or at least its efficiency. In fact in order to escape from this minima, it is required first to end up in the low-probability edge of the well and then proposing a new energy, high enough, to overcome the potential and exploring the other peaks.

With QIHMC instead, the mass itself is stochastic hence there are chance that one uses a small mass in one path. The light particle has a higher chance to jump

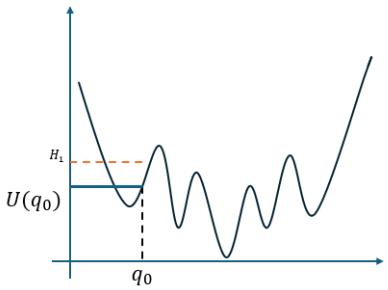

Figure 2.9: Upon a complex parameter space, jumping from local minima to other position is everything but trivial. A careful tuning of the kinetic energy proposal must be performed in order to achieve satisfactory results. The QIHMC through the randomization of the mass matrix, if carefully adapted, grants an elegant method to overcome this issue, allowing the exploration of many energy level thanks to the randomness of the mass.

to another well through “tunneling" like effect.
The optimal choice for an adequate proposal distribution for the mass matrix is still a subject of research. Considering only diagonal form, one could in general use all kind of strictly positive distributions. The most simple case would be a random draw from 2 Identity matrix with different proportionality coefficient:

$$M = I(\delta(c-1) + \delta(k-1)) \tag{2.31}$$

Other methods have been explored, employing different distributions (log-normal, exponential, etc.) where draws from those distribution select the mass for each dimension at random at the beginning of each iteration.

2.5 Proposed Algorithm

The proposed algorithm aims to improve the performance of the existing methods exploiting all the aspect discussed so far. It employs a dynamic implementation similar to the NUTS algorithm. However, instead of progressive slice sampling, it utilizes progressive biased multinomial sampling, suggested by [3], but in a new innovative manner.

2.5.1 Orbit Selection

As just mentioned, the orbit selection Kernel presents the same characteristics as the one used in the NUTS algorithm. In fact it employs a doubling procedure to create copies of the tree in consideration and extend the orbit until the inversion

criteria described by eq. 2.2.2 is satisfied. After each doubling the direction of the evolution is chosen at random as usual.
This choice is justified by the fact that when resampling the momenta, the direction of evolution is randomly chosen. Moving only forward in time does not guarantee a shift through the typical set, so both forward and backward movements are allowed to improve efficiency. This raises the question of how many steps should be taken each time. Since a single step would require generating a random direction for every point, the doubling procedure is the most effective solution, leaving no room for ambiguity.

2.5.2 Index Selection

Hence maintaining the doubling trajectory procedure, given J_{old} and the current picked state $z' \in J_{old}$, each new subtree J_{new} is created by starting from an empty orbit and then merging it with the old one. The differences from the NUTS is on the index selection and on the construction as well of merging of the new subtree. In fact exploration of the new branch is performed sequentially rather than recursively, adding each new point to the new trajectory using a progressive biased multinomial sampling. For example, to build a new branch at level k has to be built, one needs to take 2^k steps. Arriving at step $j < 2^k$, the proposal will be updated as:

$$I(z|J^j_{new}) = min(1, \frac{e^{-H(z_j)}}{\sum_{z\in J^{j-1}_{new}} e^{-H(z)}})I(z_j|j) + (1 - min(1, \frac{e^{-H(z_j)}}{\sum_{z\in J^{j-1}_{new}} e^{-H(z)}}))I(z|J^{j-1}_{new})$$

Here J^j_{new} indicates the new tree built up to the position j, and $I(z_j|j)$ represent the probability of choosing z_j given only this state which, under the multinomial sampling procedure, is equal to 1. Then this transition is re-applied when merging the final trees $J_{new} \cup J_{old}$.

$$I(z|J) = min(1, \frac{\sum_{z\in J_{new}} e^{-H(z)}}{\sum_{z\in J_{old}} e^{-H(z)}})I(z|J_{new}) + (1 - min(1, \frac{\sum_{z\in J_{new}} e^{-H(z)}}{\sum_{z\in J_{old}} e^{-H(z)}}))I(z|J_{old})$$

This describes an iterative method to sample from the trajectory keeping only the currently picked state in memory.
The transition described favors samples from the new trajectory hence far away from the starting point, however averaging over all possible initial states assures convergence to the correct distribution, cancelling out the bias.

In order to show the invariance of such procedure that involve the sequential application of eq. 2.5.2 the simplest way is by induction.

Case 0

Given the initial state z_0, the first point is generated hence $J = \{z_0, z_1\}$:

$$\sum_{z' \in \{z_0, z_1\}} \pi(z') I(z|z_0, z_1) = \pi(z_0) I(z|z_0, z_1) + \pi(z_1) I(z|z_0, z_1)$$

inserting the definition of $I(z|z_0, z_1)$:

$$I(z|z_0, z_1) = min(1, \frac{e^{-H(z_1)}}{e^{-H(z_0)}}) \mathbb{I}(z = z_1) + (1 - min(1, \frac{e^{-H(z_1)}}{e^{-H(z_0)}})) \mathbb{I}(z = z_0)$$

is straightforward to show $\sum_{z' \in \{z_0, z_1\}} \pi(z') I(z|z_0, z_1) = \pi(z') \mathbb{I}(z \in z_0, z_1)$.

Suppose

$$\sum_{z' \in J} \pi(z') I(z|J) = \pi(z') \mathbb{I}(z' \in J)$$

Then adding one more point

$$\sum_{z' \in J^{+1}} \pi(z') I(z|J^{+1}) = \sum_{z' \in J} \pi(z') I(z|J) + \pi(z') \mathbb{I}(z = z^{+1}) =$$

$$= \pi(z') \mathbb{I}(z' \in J) + + \pi(z') \mathbb{I}(z = z^{+1}) = \pi(z') \mathbb{I}(z' \in J^{+1})$$

as requested.

2.5.3 Energy function

In this work, in addition to the algorithm itself, a new energy function is proposed. Drawing on the theory from Riemann Manifold HMC, but conscious about its limitations and computational demands when the mass matrix is taken to be the Hessian, the proposed energy introduces a dependence on the mass matrix based on the current position without a massive increase in the computational cost. This modification to the Hamiltonian flow speeds up the particle in low probability regions and slows it down in high probability regions.

This kind of transformation results valid only for a certain class of probability distributions. In fact the target distribution must be bounded by a known lower bound (supposed to be at 0 without loss of generality) and ∞. If the distribution respect those conditions then the mass matrix can be modified, without any singularities, in:

$$M_f(q) = \frac{1}{1 + U(q)} M \tag{2.32}$$

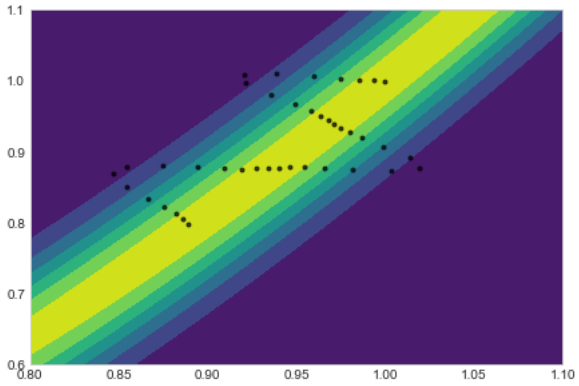

Figure 2.10: Trajectory on the parameter space for the Position Based energy function. The mass gets bigger in high probabilities spaces, increasing the mass and slowing the orbit, and vice versa for low probability spaces.

Where the +1 term in the denominator is a shift to ensure positivity of the mass and in general depends on the lower bound.
It is clear by this transformation that when the current position is far away from any relevant space, i.e. low probability, then potential energy is high, reducing the mass with the consequence of faster integration even though it might be a local minima. Contrarily when the position is close to the minima, i.e. $U(q) \sim 0$ then the mass matrix recover its original value, slowing the exploration of the high probability space as can be appreciated in fig. 2.10.

To better understand this procedure and its application the energy can be rewritten in:

$$H(q,p) = U(q) + p^T(\frac{M}{1+U(q)})^{-1}p + log(det(\frac{M}{1+U(q)})) =$$
$$= U(q) + (1+U(q))p^T M^{-1}p - Dlog(1+U(q))$$

As in the RMHMC, this modification renders the Stormet-Verlet integrator non-symplectic due to the dependace on position on the Kinetic term. However by a more careful analysis the integration of the space can be decomposed to create a more sophisticated splitting integrator of the form:

$$\psi_{dt/2}^{dQ,U,logDET} \circ \psi_{dt/2}^{dQ,T} \circ \psi_{dt/2}^{dP,T} \circ \psi_{dt/2}^{dQ,T} \circ \psi_{dt/2}^{dQ,U,logDET} \tag{2.33}$$

The evolution along the edges can be performed using the Stormer-Verlet method, while the updates at the center, where the kinetic energy depends on both position and momentum, can be handled with a generalized leapfrog integrator.

$$L = \begin{cases} p_{1/2} = p_0 - \frac{dt}{2}\nabla U(q_0) + \frac{dt}{2}\frac{\nabla U(q_0)}{1+U(q_0)} \\ p_1 = p_{1/2} - \frac{dt}{2}\nabla U(q_0) p_1 M^{-1} p_1 \\ q_1 = q_0 + \frac{dt}{2}[(1+U(q_0))M^{-1}p_1 + (1+U(q_1))M^{-1}p_1] \\ p_2 = p_1 - \frac{dt}{2}\nabla U(q_1) p_1 M^{-1} p_1 \\ p_3 = p_2 - \frac{dt}{2}\nabla U(q_1) + \frac{dt}{2}\frac{\nabla U(q_1)}{1+U(q_1)} \end{cases} . \tag{2.34}$$

However this approach may cause different problems depending on the space of the parameters. A fast rising potential or starting from a extreme low probability state, may cause divergences in the integration due to the potential multiplication in the parameter update. This has to be balanced out by the mass matrix, coherently defining a more massive point for high potentials, since almost massless point would result in instabilities and a full rejection of the candidate states.
To overcome this challenge an additional hyperparameter can be introduced to lower the effect of the potential as $U(q) \to U(q)/f$. Moreover for high correlated samples, generating a diagonal mass matrix can results in a systematically wrong direction of the momenta. This can be balanced out with the proposed method to shuffle the mass matrix after the random diagonal proposal. This process can be justified as: given the diagonal mass matrix M, this diagonal form is referred to the existing diagonalization of the space which is unknown, while our basis is on some other mixed form. Hence a random change of basis is performed with a random orthogonal matrix sampled with the Haar distribution.
However this method introduce other matrix multiplication slowing the algorithms of orders of magnitude hence is not considered on the following tests.
In general the energy function proposed can be mixed with other existing methods. In fact the values of the mass matrix can be chosen to be the identity as well as employing the QIHMC or defyining the hessian, depending on the problem at hand. In general HMCs have many variation that can be performed to improve the performancies of the algorithm, however is not trivial to find globally efficient solutions which work on the majority of the problem. This freedom to chose the a particular variation on the HMC sampler reflect the general variability of the MCMC samplers for which the Kernel can have infinite solutions.

Chapter 3

Test

3.1 Testing

Software is often tested by applying it to datasets for which the "right answer" is known or approximately known, and comparing the expected results to those obtained from the software. Such a strategy becomes more complicated with software for Bayesian analyses, whose results are inherently stochastic, but can be extended to develop statistical assessments of the correctness of Bayesian model-fitting software. The basic idea is that if parameters are drawn from their prior distribution and data from their sampling distribution given the drawn parameters, and then perform Bayesian inference correctly, the resulting posterior inferences will be, on average, correct.
Different forms of inference can be performed in order to check the validity of the null hypothesis H_0, corresponding to the known results.

Credible interval

Even though the Maximum Likelihood Estimator (MLE) provides the best value for the parameter to fit the data—represented in our case by Eq.1.2 to estimate the posterior in the central limit theorem, which is Gaussian distributed—it does not offer any insight into the error range within which the true value might lie. Additionally, the variances derived from MLE are not actual probabilities that indicate this range.
For this purpose, interval estimation is more suitable. Since the posterior represents a probability density function (pdf) for the parameter, it is straightforward to construct an interval estimator. By integrating the posterior distribution over a specific region of the parameter space, ψ, known as the credible region, one can determine the probability, or credibility, that the parameter falls within the region

ψ. Formally, this can be expressed as:

$$CI(x) = \int_{\psi} \pi(q|x)dq \tag{3.1}$$

Therefore, by setting the credible interval (CI) to a high probability, constructing the interval becomes straightforward. In our context, it is easy to approximate this interval using the posterior samples. The only ambiguity lies in choosing the credible region, ψ. However, the smallest interval can be obtained by using a probability ordering on the posterior, which is simple to implement.

Kolmogorov–Smirnov test

Another useful statistics is the p-value extracted from a Goodness of Fit (GOF) which is a statistical tool to assert the accordance of the hypothesis with the data. Many statistics can be employed to perform a GOF measure, however for continuous 1-D distribution the most reliable would be the Kolmogorov–Smirnov test. Given the null hypothesis of a one dimensional distribution with known cumulative function, the empirical cumulative is calculated and compared to the expected one.
Specifically, the statistic is of the form:

$$D_n = sup_x|F_n(x) - F(x)| \tag{3.2}$$

Since the true distribution D is independent of the original distribution it is easy to extract a $p - value = 1 - Pr(D < D_n)$ since the distribution D is known and its cumulative can be calculated.
This is a powerful test but it considers just 1D dimensional distributions, hence when more dimensions are involved, the test is taken on the marginal distribution for each dimension.

3.1.1 Gaussian

As usual, the first testing ground will be the multivariate Gaussian distribution. In this simple case, let's consider a 5-dimensional distribution with unit variance.
The mean are chosen at random from a uniform distribution $\mu \sim U(0, 10)$. Then if the data are sampled given the known distribution using a MCMC, the resulting sample can be viewed as the likelihood from which inference on the mean can be performed. Comparing these results provides insight into the performance and validity of the algorithms being used.

The Metropolis and MALA algorithms serve as benchmarks for comparing Hamiltonian-based algorithms. These include the basic Hamiltonian Monte Carlo [5], the more sophisticated NUTS [13], and the proposed MyNUTS algorithm with the PB (position-based) mass matrix. Each algorithm is executed to generate a sample of 15,000 iterations, with the first 5,000 iterations discarded to ensure the stationarity of all algorithms.

Tables tab.**??** shows an incredible lowering of the IAT and the rejection rate, meaning less waste of computation. However, the trade-off between computation time and IAT is still not favorable for HMCs due to the problem's simplicity. In this context, MALA and Metropolis perform extremely fast, and better results can be achieved with an order of magnitude more steps. However form tab.**??** clearly demonstrate the advantages of Hamiltonian-based algorithms, even for a simple problem like a multivariate Gaussian. These algorithms produce results that are more consistent with expectations, although only a few parameters are compatible with the true values. However when looking at 90% credible level in fig. 3.2 it can be appreciated how all the algorithms manage to find an appropriate interval which includes the true value.

Method	**Time (s)**	**Rejected**	**IAT**
Metropolis	1.29	899	95.7
MALA	4.29	968	71.8
HMC	20.7	0	14.0
NUTS	139	2	12.7
MyNUTS	174	0	3.79
PB	34.6	45	1.72

Table 3.1: Results for the Gaussian inference. It is noteworthy that the number of rejected samples is much lower compared to simple MCMC methods, indicating better space exploration. Although the integrated autocorrelation time (IAT) is significantly lower for the HMC sampler, given the simplicity of the problem and low dimensions, an overall assessment of performance would still favor the Metropolis and MALA algorithms due to their speed.

Method	**Dim 1**	**Dim 2**	**Dim 3**	**Dim 4**	**Dim 5**
Real Mean	9.767	3.802	9.232	2.617	3.191
Metropolis	9.606 ± 0.019	3.623 ± 0.016	9.934 ± 0.022	2.742 ± 0.034	3.186 ± 0.019
MALA	9.739 ± 0.006	3.706 ± 0.009	9.369 ± 0.006	2.423 ± 0.009	3.121 ± 0.006
HMC	9.789 ± 0.003	3.807 ± 0.003	9.260 ± 0.003	2.658 ± 0.003	3.300 ± 0.003
NUTS	9.855 ± 0.004	3.817 ± 0.004	9.181 ± 0.004	2.700 ± 0.004	3.213 ± 0.004
MyNUTS	9.797 ± 0.001	3.761 ± 0.001	9.271 ± 0.001	2.578 ± 0.001	3.170 ± 0.001
PB	9.794 ± 0.001	3.807 ± 0.001	9.230 ± 0.001	2.587 ± 0.001	3.209 ± 0.001

In order to perform the KS test, since it is a 1D test, the marginal posterior has to be taken into consideration. However, due diagonal form of covariance matrix, there are no correlation between the variables. Hence considering its marginal distribution

is equivalent to test a 1-D Gaussian. The resultant p-values are listed in tab. 3.2.

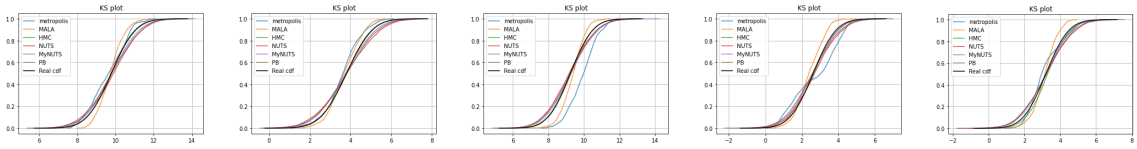

Figure 3.1: Graphical representation of the KS test. For each dimension and algorithm, the empirical and true cumulative distributions are compared. It is evident that the more complex algorithms exhibit significant advantages, with the lines almost entirely overlapping the true distribution.

Method	Dim 1	Dim 2	Dim 3	Dim 4	Dim 5
Metropolis	4.40e-75	4.35e-108	0.000	0.000	4.88e-91
MALA	1.27e-101	1.32e-50	5.39e-184	1.78e-118	1.03e-131
HMC	4.48e-03	7.07e-02	2.17e-02	3.50e-05	6.73e-21
NUTS	6.76e-52	2.94e-51	1.60e-37	3.32e-34	1.01e-37
MyNUTS	1.07e-18	6.06e-30	2.02e-20	2.77e-14	1.07e-15
PB	6.88e-40	2.10e-27	5.43e-26	3.58e-34	1.65e-26

Table 3.2: P-values for each dimension across different methods

Better analysis can be achieved with the KS test described earlier in Section 3.1. The intuitive cumulative distribution plot shown in fig. 3.1 clearly anticipates the resulting p-values displayed in tab. 3.2. Even for these few dimensions, the Metropolis and MALA algorithms fail to sample the correct mean and variance of the projected dimension, resulting in extremely low p-values and visible discrepancies with the known cumulative distribution.

To combine the information from the tests in each dimension, several statistics can be applied to the p-values. However, it is important to remember that a p-value does not represent a probability, so simple multiplication would be incorrect. Here,

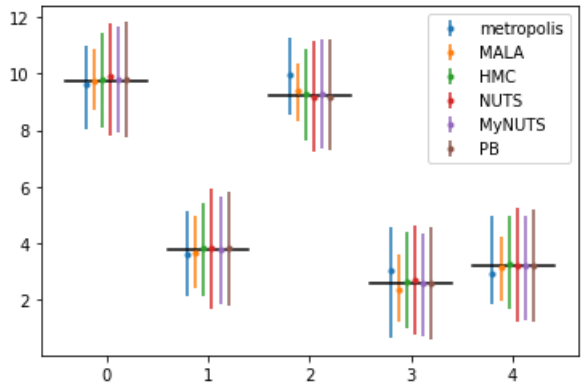

Figure 3.2: In black the real value for each dimension of the Gaussian. Comparing 90% CI between the algorithms.

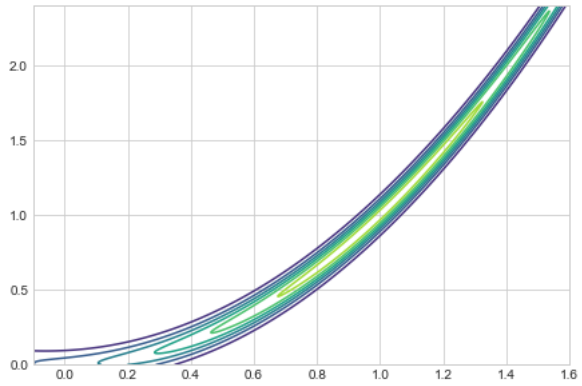

Figure 3.3: Exponential of the Rosenbrock function

we use the following statistic for the p-values: $\lambda = -2\sum_i^D \log \text{p-value}_i$, which is distributed according to a χ^2_{2D}. The results are displayed in tab. 3.3.
Even though the p-values are very low for all the algorithms, it is clear that there are statistical improvements in the quality of the results for HMC samplers. This is evident because the p-values for Metropolis and MALA were so low that they were rounded to zero.

Metropolis	**MALA**	**HMC**	**NUTS**	**MyNUTS**	**PB**
-	-	1.58e-24	2.38e-200	3.85e-88	2.80e-142

Table 3.3: For each algorithm, the combination of the p-values from the KS test in each dimension is displayed. The statistic in use is: $\lambda = -2\sum_i^D \log \text{p-value}_i$.

3.1.2 Rosenbrock function

The Rosenbrock function:

$$R(\bar{x}) = \sum_{i=0}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{3.3}$$

It is characteristic for its banana shape in 2 dimension which clearly present a minima in $(1, 1)$ and it is used in form of a probability as $P_R \propto e^{-R(\bar{x})}$.

Again, the Metropolis and MALA are compared to the Hamiltonian algorithms as in the Gaussian case before 3.1.1. In fig. 3.4 are displayed 100 iteration from the same starting point for each algorithm. It is clear the improvement in the sampling procedure given from the information on the space for the Hamiltonian based algorithms. In fact they manage to efficiently explore the typical set, allowing distant jumps from one point to the next. In tab. 3.4 the information about the various runs and in tab. 3.5 its relative posterior plots.
HMCs outperform Metropolis and MALA under all the aspects but the execution

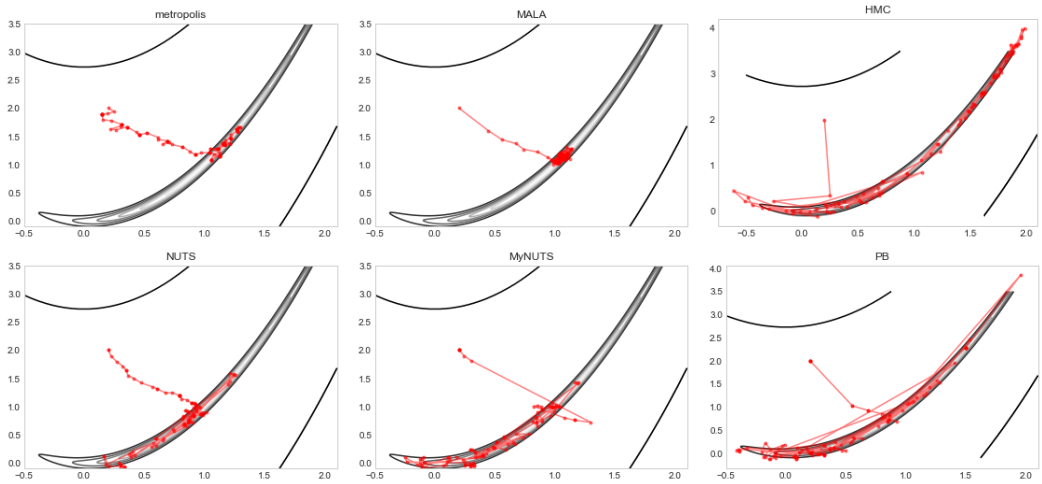

Figure 3.4: Plot for 100 iteration (Rosenbrock function). HMCs sampler display a better ability to sample from the typical set, allowing considerable jumps for subsequents points.

time which is balanced out by their poor results.
However, it is worth noting that achieving decent results with HMCs, in this example requires careful tuning of the mass matrix due to the rapidly rising potential of the Rosenbrock function, which increases with the fourth power. From the definition in eq. 1.4.2 , it is clear that the stability of the Stormet-Verlet integrator is strongly linked to the gradient of the potential. For potentials that grow quickly, especially in the tails as seen in this case, stability is compromised [4]. To maintain stability, it is necessary to use a sufficiently small integrator step and a low mass matrix, preventing the average energy from exceeding what is necessary.

Name	**Rejected**	**IAT**	**Time (s)**
Metropolis	4919	94	1
MALA	891	96	1
HMC	1	86	24
NUTS	482	31	155
MyNUTS	751	16	173
PB	324	27	526

Table 3.4: Results for the Rosenbrock inference. Once again, the performance of the algorithms must be balanced among results, time, IAT, and computational waste. As usual, the Metropolis and MALA algorithms demonstrate very fast execution times but yield much poorer results.

3.1.3 Automatic Software Validation

A more effective way to validate the performance of software is suggested by [8]. This method aims to provide a statistical verification of the algorithm.

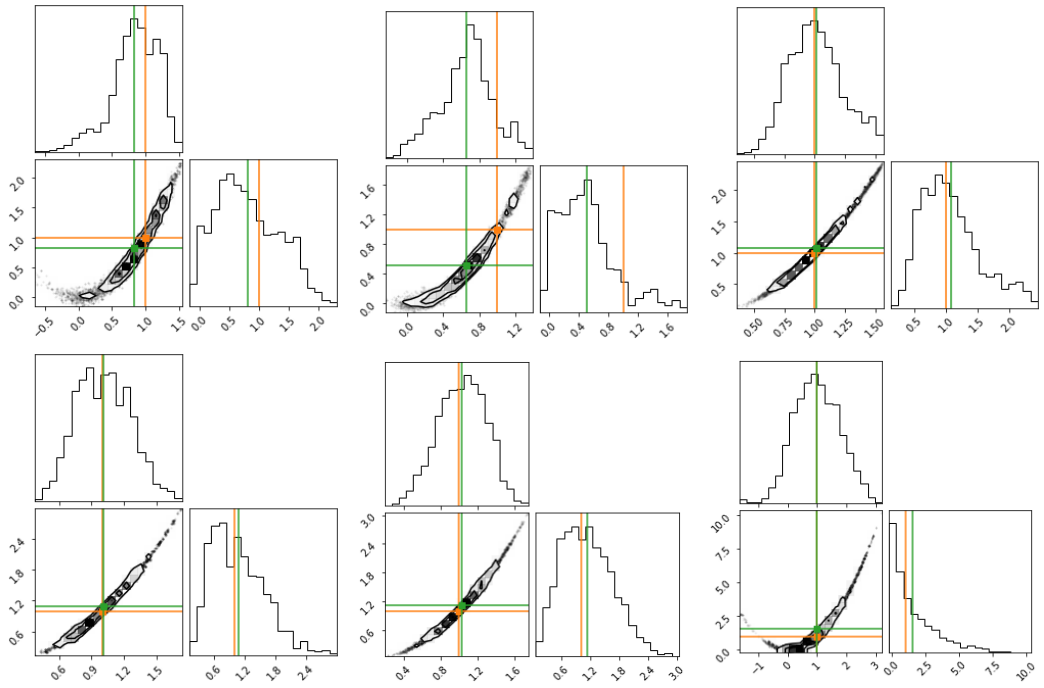

Figure 3.5: Posterior results for the Rosenbrock function.

Consider the general Bayesian model $\pi(x|q)p(q)$, where $\pi(x|q)$ represents the sampling distribution of the data, x, $\pi(q)$ represents the proper prior distribution of the parameter vector, q. Inferences are made based on the posterior distribution,$\pi(q|x)$. In sec. 3.1.1 the parameters are sampled from $\pi(q)$, then the likelihood $\pi(X|Q)$ is sampled using the algorithms to infer the parameters from this data.

This method goes further by considering a process that can generate the likelihood of a known problem. Specifically, if a q^0 is generated according to $\pi(q)$, and data x' are then generated according to $p(x|q^0)$, then (x', q^0) represents a draw from the joint distribution (x, q), which implies that q^0 represents a draw from $\pi(q|x')$. Now using the to-be-tested software, sample from $\pi(q|x')$ consider the posterior sample of size L, $q_s = (q_1, q_2, ..., q_L)$, .If the software is written correctly then marginal distribution of q^0 and each of $q_1, ..., q_L$ is the same, they are all drawn from $\pi(q|x')$. This is also true for any subvector of the components of q, hence this approach is valid considering each scalar component of q independently (and that's why we drop the vector notation).
Using this information, the software bases its validation on the posterior quantiles of the true values of scalar parameters. Let $F(q^0) = Pr(q^0 > q)$, the posterior quantile of q^0, where q represents a random draw from the distribution $(q(1), ..., q(L))$

generated by the to-be-tested software. The estimated quantile is:

$$F(q^0) = Pr(q^0 > q) = \frac{1}{L}\sum_i (q^0 > q_i) \tag{3.4}$$

If the software is written correctly, for continuous q, the distribution of $F(q^0)$ is fundamentally discrete, taking one of L + 1 evenly spaced values on $[0, 1]$ or equivalently the rank uniform in $[0, L]$ [21].
This suggest an iterative procedure to obtain ranks for the sampled parameters (independently for each dimension) and test the null hypothesis of uniformity. Specifically the steps are:

- sample q^0 from $\pi(q)$
- sample x from $\pi(x|q)$
- run the to be tested software L times to generate a sample from $\pi(q|x)$
- collect the ranks $\frac{1}{L}\sum(q^0 > q_i$
- repeat

Therefore, this algorithm allows for the evaluation of the software's performance by analyzing these statistics. Unlike previous methods, it offers a clear perspective on the software's statistical validity.

Here this software is used on a 10 dimensional normal distribution with unit covariance and mean values generated according to a uniform distribution in $[0, 10]$. Metropolis, MALA, HMC, NUTS, and the proposed variations are each run for 1000 iteration, repeating the process 300 times, for a total of 3000 recorded ranks.
Each bin population in $[0-L]$ follows a binomial distribution $\sim Bin(N, p)$ where the probability in this setting is assumed to be uniform, i.e. $p = \frac{1}{L+1}$. Hence expectation value for the mean and variance are known, given respectively by Np and $Np(1-p)$. Results are displayed in histogram in fig. 3.6.
Again to assure that the ranks follow a uniform distribution a better approach is to perform a GOF test. The null hypothesis is the uniform distribution, therefore the Kormogorow test can be applied and the resultant p-values are listed in tab. 3.5. In this statistical point of view, clear are the advantages of HMCs samplers which manage to outperform the simpler counterparts.

The cumulative of a uniform variable, is still distributed according a Binomial where the probability is exactly the cumulative function in the point, $p = \frac{x}{L+1}$. Therefore, the empirical cumulative distribution function (ECDF) used for the KS test is shown in 3.6 along with its variance. The Metropolis and MALA algorithms

Metropolis	MALA	HMC	NUTS	MyNUTS	PB
1.6e-34	7.1e-21	5.6e-07	0.144	0.175	0.003

Table 3.5: Table with six values

tend to produce an "S" shape in the ECDF more than the other algorithms due to the accumulation of evidence at the extremes. This behavior is related to the autocorrelation within the algorithms. Generating points close to each other shifts the cumulative distribution towards higher or lower values.

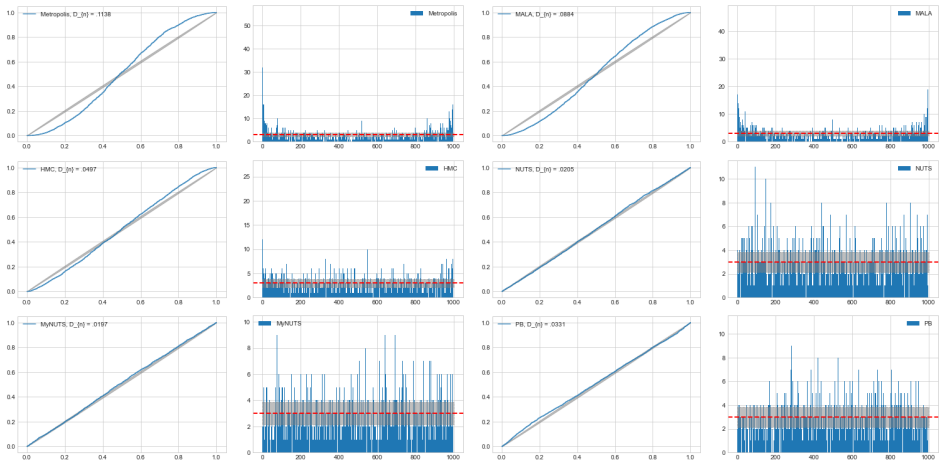

Figure 3.6: KS test for the posterior quantiles and their respective rank histograms. It is evident that HMC samplers provide solid and consistent results, even from a statistical standpoint. In contrast, the Metropolis and MALA algorithms exhibit high autocorrelation, concentrating evidence in specific regions and creating an S-shaped deviation from the theoretical straight line.

3.2 Benchmarks: Regression

The first real application on toy problems will include the analysis of a linear regression and a logistic regression problems. However the application of the algorithms have been straightforward, without an actual tuning of the hyperparameters so if not explicitly specified the mass matrix is set to the identity while the the timestep is set to $dt = 0.01$. This because in this section HMCs samplers are applied to regression problem to show their suitability and consistency in comparison to their simpler counterpart, rather than exactly solving the problem.

3.2.1 Linear Regression

In standard additive-error regression model for response data $Y = y_1$ and covariates $X = x_1, \ldots, x_n$ for model parameters q can be expressed as:

$$Y = f(X, q) + \epsilon$$

where f defines the functional relation between $E(Y)$ and X, q, and ϵ is a Gaussian variable with zero mean and covariance matrix Σ.

The likelihood function is defined as:

$$p(Y|X, q, \Sigma) = N(f(X, q), \Sigma)$$

In a Bayesian approach, one seeks to infer on the parameters and find either the posterior

$$p(q|Y, X)$$

. However, in general, X is affected by some error, and two kinds of modeling can be performed:

$$Y = f(X^*, q) + \epsilon$$

$$X = X^* + \delta$$

or

$$Y = f(X, q) + \epsilon$$

$$X = X^* + \delta$$

Hence, the true value X^* is never known for sure, and the response data depend either on the true value or the measurement. In the following the first method is applied.

The first benchmark model is a simple linear regression to compare the known real points $X^D = \{x_i^D\}$, intercept c_0 and coefficient m_0 to the esteemed ones. In this case, f take the simple form:

$$f(x, m, x) = \beta x + \beta_0 \tag{3.5}$$

Therefore data and respective response data ($X = \{x_i\}$ and $Y = \{y_i\}$) are measured, assuming to known the error on the measure (ϵ, δ).

The likelihood for the response data along with the Gaussian prior on the covariates, would yield a model with the following posterior distribution:

$$p(Y|X, m, c) \propto \prod_i G(y_i^D; \beta x_i + \beta_0, \epsilon) G(x_i^D; x_i, \delta) \tag{3.6}$$

Setting the values for the model as: slope $\beta = 50$, intercept $\beta_0 = 30$, while the errors

measures $\delta = 10$, $\epsilon = 15$. Then a fake sample for covariates and response data is generated, consisting of 10 points.
Metropolis, MALA and all the HMCs variations are run to obtain a total sample of 10000 iteration after a first burn-in period of 5000 steps. Main results from the runs are displayed in tab. 3.6.

Name	**Time (s)**	**Rejected**	**IAT**	**Slope (30)**	**Intercept(50)**
Metropolis	0.70	5882	477	30.04 ± 0.001	26.65 ± 0.06
MALA	2.21	5033	480	30.61 ± 0.002	27.29 ± 0.33
HMC	56.66	1	474	30.48 ± 0.003	55.91 ± 3.50
NUTS	499.05	68	451	28.47 ± 0.059	266.08 ± 15
MyNUTS	593.50	7	381	29.14 ± 0.227	106 ± 20
PB	759.18	8	440	29.61 ± 0.291	52 ± 79

Table 3.6: Results for the linear regression problem. Once again, the Metropolis and MALA algorithms are instantaneous, while other samplers increase computation time by two orders of magnitude. This is particularly true for the position-based samplers, which exhibit exceptionally high execution times.

The benefits of HMCs starts to be clear as the number of rejected point drastically decreases indicating a better exploration of the typical set without waste of computation on irrelevant space. Hamiltonian methods manage to retrieve a consistent parameter space with the given data as the credible region at 90% are correctly compatible all the data points and relative error, fig. 3.7. However the IAT is really high for all the algorithms. It is worth noting that it will be probably higher for metropolis and MALA since the truncation on the calculation of the autocorrelation, is valid only on the limit of samples much bigger that the integration as discussed in sec.**??**, which might not be the case in this situation. Moreover from fig. 3.7 is clear how simpler samplers manage to find reasonable points of the space, corresponding to local minima, but lack the ability to actually explore it and rather gets stuck in them with consequent high rejection rate.

3.2.2 Logistic Regression

Logistic regression is a categorical binary model which supposes linear dependencies on the odd space of a bernullian event $Bern(k; p)$.

$$\frac{p}{1-p} = \beta x + \beta_0 \tag{3.7}$$

The likelihood of the points:

$$L(x|k, p) = \prod Bern(k, p) = \prod p_i^{k_i} * (1 - p_i)^{1-k_i} \tag{3.8}$$

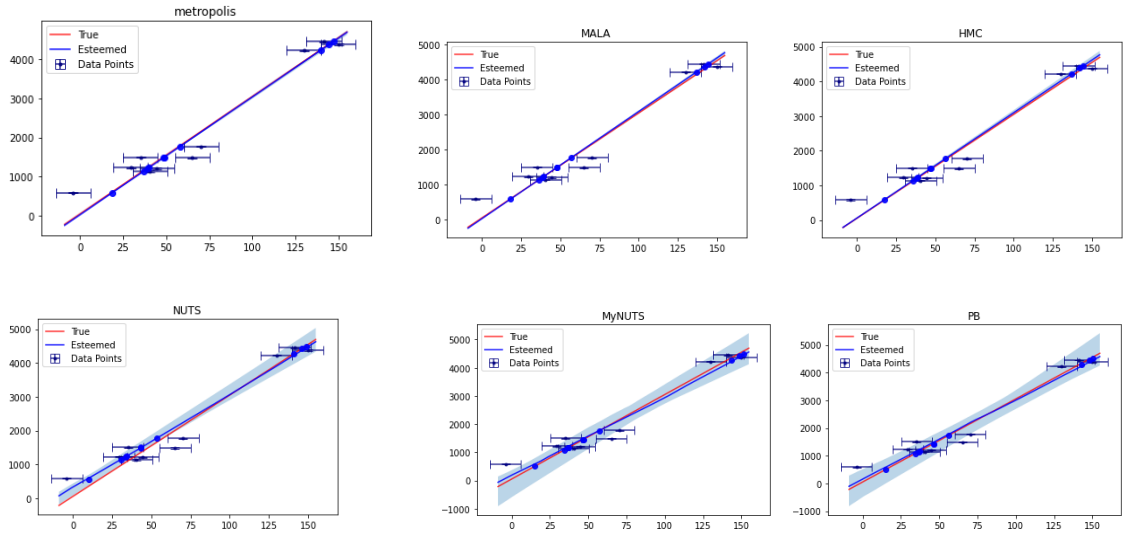

Figure 3.7: Graphical comparison between the true straight line and the estimated one within the 90% confidence interval for each presented algorithm. It is evident that Metropolis, HMC, and MALA only found the minima of the system and failed to correctly explore the space.

with:

$$p_i = \frac{exp[\beta x_i + \beta_0]}{1 + exp[\beta x_i + \beta_0]} \tag{3.9}$$

Where $q = (\beta, \beta_0)$ are the parameter for the linear dependency and x are the covariates data of the model and can be of arbitrary dimension.
Hence, given a MCMC and a initial set of known categorical data and their covariates, the obtained sample can be use for the inference of future observation. After a first phase of training given the data with known label, the class Y_{new} of a new point X_{new} can be inferred with the predictive posterior:

$$\pi(Y_{new}|X_{new}, X, Y) = \int \pi(Y_{new}|X_{new}, q)\pi(q|X, Y)dq \tag{3.10}$$

which is easy to esteem from the sample due to the discrete nature of the problem.

$$\pi(Y_{new}|X_{new}, X, Y) \sim \frac{1}{N}\sum_{i}^{N} \pi(Y_{new}|X_{new}, q^i) \tag{3.11}$$

Since $q^i \sim \pi(q|X, Y)$.

In the context of a classification problem the initial sample is given. In this case 200 points are generated from a 2dimensional Gaussian distribution. Half of them are sample using the means $\mu_0 = (0, 0)$ and are assigned, without loss of generality, to class 0. The other half has means equal to $\mu_1 = (1.5, 1.5)$ and belong of course to class 1.

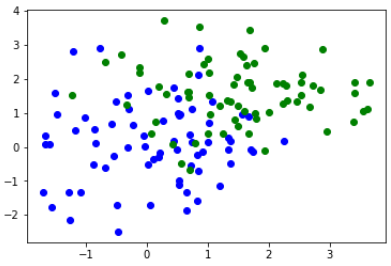

Figure 3.8: Data points extracted from two different Gaussian distributions for classification. Green represents class 1, while blue represents class 0.

The dataset is then split in a train set (140 of them, scattered with their respective class in fig. **??**) and a test set (60) on which the validation of the classification is performed.
3.8.

The procedure of splitting 70% train and 30% test set is repeated 5 times to perform a cross validation over the full dataset to lower the statistical relevance of taking particular samples that can over-fit or under-fit the model, then the full obtained posterior is used for the inference.
During test phase, the dataset is classified and compared to true class. Since the classification is binary the possible outcomes are of 4 kinds, namely TP, TN, FP, FN, the true positive, true negative and the respective false counterpart.To understand the properties of a classifier is wise to look at those 4 quantities as a whole and depending on the problem, class distribution and results, different statistics on those results can be more informative.
However, for the straightforward problem at hand, the confusion matrix, accuracy, and ROC curve provide sufficient insight into the prediction accuracy of the algorithm. The ROC curve is a statistical tool to understand the behaviour of the classifier on a overall view. In fact it is built by comparing the true positive rate and false positive rate on different threshold (defined as, $TPR = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$). In fact the classifier returns the probability of the point to belong to a certain class and the threshold is up to the user since in theory one could prefer correctly predict more positives with the drawback of missclassifiying some negatives class and vice versa. This curve gives information about the overlap of the 2 distribution defining the two groups as depicted in fig. 3.9.

Best case scenario, the distribution of the two categories do not overlap at all, hence the relative ROC curve would be a straight line always equal to 1 except when the threshold becomes 0, as the TPR must become 0 as well. When, instead the 2 distribution overlap completely, then the ROC curve must lie on the diagonal of the space due to the totally uncertainty of the classification. This give a hint to determine how good the algorithm fit the model through the area of the ROC curve.

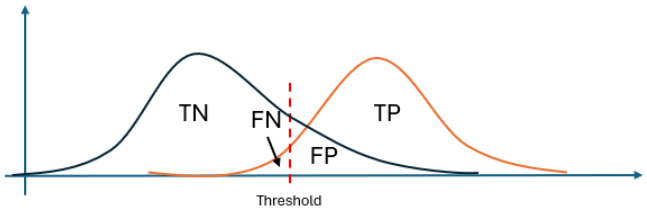

Figure 3.9: Graphical representation of the overlap between two distributions concerning a threshold value for classification. The areas crossing the threshold are the misclassified instances.

In the following the results for one split of the dataset executed for 10000 iteration along with the accuracy (calculated as the number of correct classification over the total) for each algorithm.

Name	**Time (s)**	**Rejected**	**IAT**	**Accuracy**
Metropolis	32.85	283	96.35	0.78
MALA	50.72	1872	98	0.83
HMC	354.96	3711	97	0.82
NUTS	165.20	4649	83	0.83
MyNUTS	272.20	655	87	0.720
PB	184.19	7548	90	0.78

Table 3.7: Results for the Rosenbrock inference. Once again, the performance of the algorithms must be balanced among results, time, IAT, and computational waste.

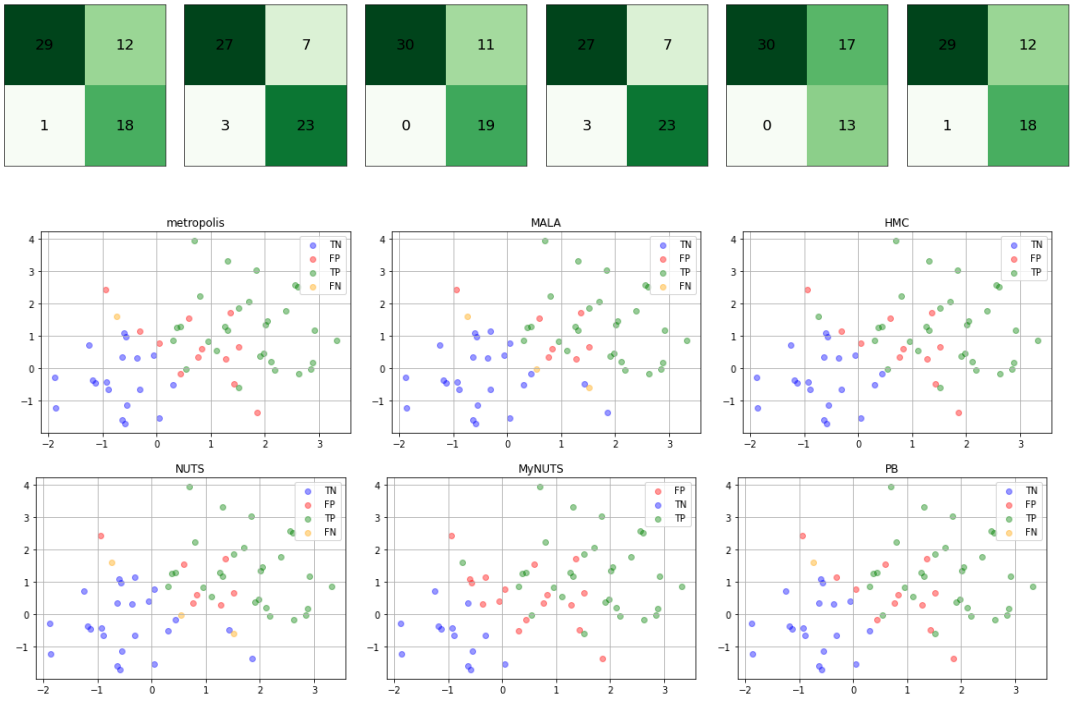

Figure 3.10: Confusion matrix and scatter plot illustrating the classification performance of various algorithms.

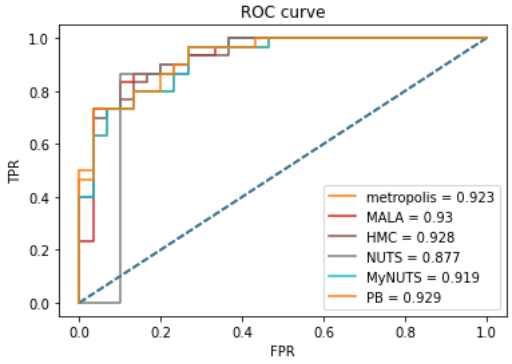

Figure 3.11: ROC curve comparison for all algorithms.

Chapter 4

Application

4.1 Laser Interferometer Space Antenna

The Laser Interferometer Space Antenna (LISA) is a planned space mission designed to detect and accurately measure gravitational waves. It aims to achieve this through the use of laser interferometry. The LISA concept involves a constellation of three spacecraft arranged in an equilateral triangle with sides 2.5 million kilometers long, orbiting the Sun in a path similar to Earth's.
To achieve its goals, current knowledge in two main source classes for LISA is reviewed: ultra-compact stellar-mass binaries and massive black hole binaries.
Unlike binaries consisting of neutron stars (NSs) and black holes (BHs), white dwarf (WD) binaries have larger radii and lower orbital frequencies at merger, making them difficult to detect with ground-based high-frequency (Hz–kHz) gravitational wave observatories like LIGO, Virgo, and KAGRA, as well as the planned third generation of these detectors. These high-frequency detectors can observe the final few to several thousand orbits of inspiral (lasting from a fraction of a second to minutes) and the merger event itself for NSs and BHs. A major advantage of LISA is that the inspiral phase (due to orbital GW damping in the compact binaries) of the vast population of tight Galactic double WDs, NSs and BHs is in the low-frequency ($\sim mHz$) GW window for up to $\sim$ 106 yr prior to their merger event. Thus a significant number of such local sources are anticipated to be detected by LISA, even though their emitted GW luminosity is relatively small compared to that of the final merger process. The possibility that LISA can measure sky locations of its sources will allow for EM follow-up observations which may result in much more precise compact object component masses, e.g. compared to high-frequency GW mergers

One of the outstanding challenges of LISA will be to analyze a datastream that consists of multiple overlapping signals from astrophysical and possibly cosmological

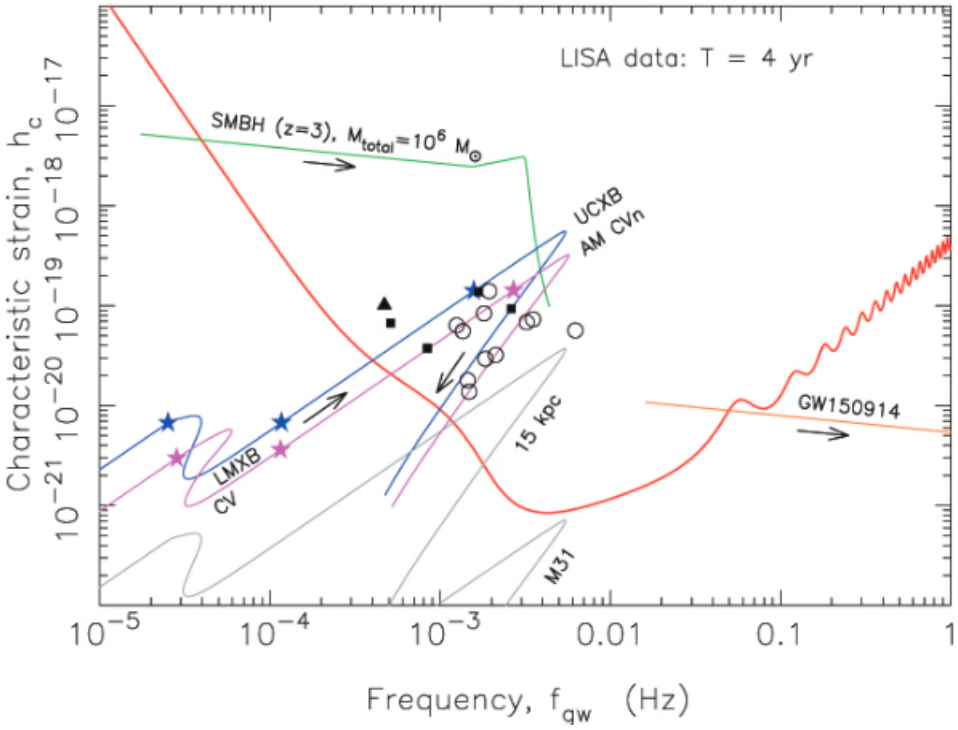

Figure 4.1: Characteristic strain amplitude vs GW frequency for LISA

sources as well as instrumental noise. Since many LISA sources will remain in band for multiple orbits, from several days or months up to the entire duration of the mission, there will be an overlap between multiple sources in any given data stretch. Hence given the data:

$$\pi(N|D) = \frac{\pi(N)\pi(D|N)}{\pi(D)} \tag{4.1}$$

Where setting the number of events fixed the form of the likelihood can be rearranged to :

$$\pi(D|N) = \int dq\pi(D,q|N) = \int dq\pi(q|N)\pi(D|q,N) \tag{4.2}$$

4.1.1 Compact binaries

The inspiral phase (due to orbital GW damping in the compact binaries) is in the low-frequency ($\sim mHz$) GW window for up to 106 yr prior to their merger event. Thus a significant number of such local sources are anticipated to be detected by LISA as a form of superposition of wave of fixed frequency, with a simpler schematization with just 3 variable per wave:
$q_\Phi = (A, w, \phi)$.

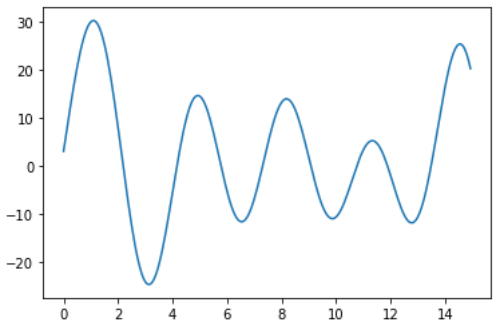

Figure 4.2: Zero noise toy example for the signal from 20 waves.

$$\Phi(A, w, \phi) = Asin(wt + \phi) \tag{4.3}$$

Therefore, given the number N of wave in the signal and the acquisition error σ, the likelihood can be expressed as:

$$-log\pi(D|q, N) \propto \sum_{i}^{T} \frac{(D_i - \sum_j^N \Phi_i(q_\Phi^j))^2}{\sigma^2} \tag{4.4}$$

Where $q = \{q_\Phi^0, .., q_\Phi^N\}$ is the collection of the parameters for each wave.
However with this likelihood form, the parameters space scale with the number of acquisition points. In order to prevent numerical instabilities in the software, it is helpful to perform a rescaling transformation.

$$\pi(G) = \frac{1}{Z} e^{-\sum G_i}, \quad \sum G_i \to \frac{1}{N} \sum G_i \tag{4.5}$$

Which simply add a additional constant to the evidence Z and allow to deal with smaller numbers. A toy example comprehending 20 waves is depicted in fig. 4.2. A direct application of Metropolis, MALA and the HMCs samplers is carried on this datastream for a total of 20000 iteration with a first burn-in period of 5000 steps. Inserting the zero noise data the posterior for the parameters should be centered in the exact values. The information about the run is showed in tab. 4.1 while the signal reconstruction with 90% CI is displayed in fig. 4.3.

Name	**Time (s)**	**Rejected**
Metropolis	9.58	9734
MALA	55.18	13613
HMC	1312.48	15025
NUTS	138.51	10424
MyNUTS	518.68	5181
PB	396.99	9242

Table 4.1: Comparison of different algorithms

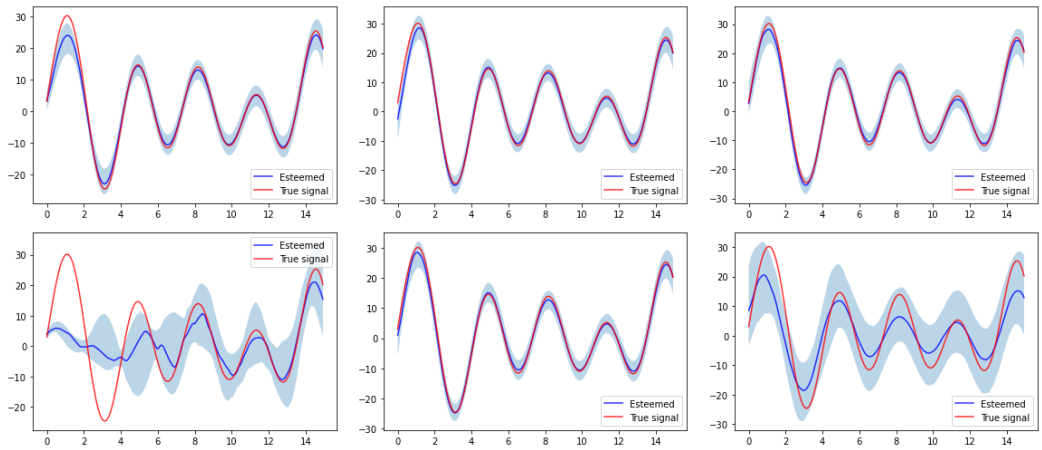

Figure 4.3: Confusion matrix and scatter plot illustrating the classification performance of various algorithms.

The Integrated Autocorrelation Time is omitted due to the fact that none of the algorihms reached the stationarity and even though they all manage to reconstruct the given signal the deviations from real parameters are more clear on the posterior plots in Appendix. **??**. The only algorithm which failed to give a good approximation of the signal is the NUTS. This problem can be addressed to the complexity of the space. The consequences are significant integration errors which increase the rejection rate in the slice sampling procedure. Recalling from [4], that the average mean error is positive, in this significant problem, this introduce a shift where most of the proposed point are below the slice parameter.
The HMC display a $> 70\%$ rejection rate due to the same problem and demand the more computation time. That is because the dynamic implementations on average performed less iterations.

4.1.2 MBHs

There is observational evidence that a significant fraction of galaxies host MBHs in their centres (Kormendy and Ho, 2013), and at least some of them harbour an MBH since the dawn of structure formation (e.g. Bañados et al., 2014; Wu et al., 2015; Bañados et al., 2018b). This, combined with the notion that galaxies aggregate via repeated mergers of smaller structures (Fakhouri et al., 2010; O'Leary et al., 2021), leads to the conclusion that a number of MBHBs should have formed across cosmic epochs, and that their ultimate coalescence phase could be observed by LISA (e.g. Klein et al., 2016; Dayal et al., 2019; Chen et al., 2020b; Barausse et al., 2020b; Valiante et al., 2021; Bonetti et al., 2019).
The GWs recorded by LISA for this event can be modelled by a wave dumped by an exponential factor for a total of 5 parameters $q_\Phi = (A, t^0, \gamma, w, \phi)$.

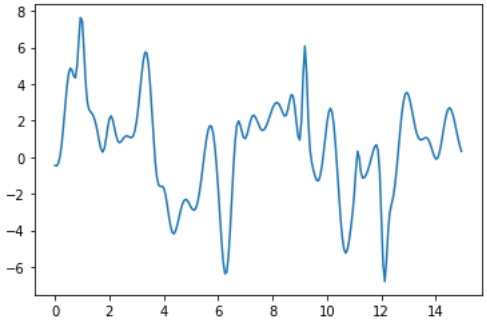

Figure 4.4: Caption

$$\Phi(A, t^0, \gamma, w, \phi) = Aexp[-(\frac{t - t^0}{\gamma})^2]sin(wt + \phi) \tag{4.6}$$

As for the compact binaries the likelihood is:

$$-log\pi(D|q, N) \propto \sum_{i}^{T} \frac{(D_i - \sum_{j}^{N} \Phi_i(q_\Phi^j))^2}{\sigma^2} \tag{4.7}$$

Again a toy problem with 30 waves (fig. **??**), for which arrival is well separated is presented and tested. The results from the raw application of the algorithms is presented in tab. 4.2 and the signal reconstruction in fig. 4.5.

Name	**Time (s)**	**Rejected**
Metropolis	25.17	9001
MALA	187.51	13511
HMC	1810.71	20477
NUTS	461.26	10369
MyNUTS	1653.83	6010
PB	542.47	17450

Table 4.2: Comparison of different algorithms

4.2 Setting

So far, we have discussed basic algorithms such as Metropolis and MALA, as well as Hamiltonian-based samplers, ranging from their classic forms to more sophisticated implementations. The straightforward application aligns with the theoretical background, which suggests that simpler algorithms struggle to explore high-dimensional parameter spaces effectively. These simpler methods often get stuck near local minima and reject many proposals. In contrast, more advanced HMC methods can explore the given space more efficiently and reduce the autocorrelation problem, although at a higher computational cost. For complex applications, this computa-

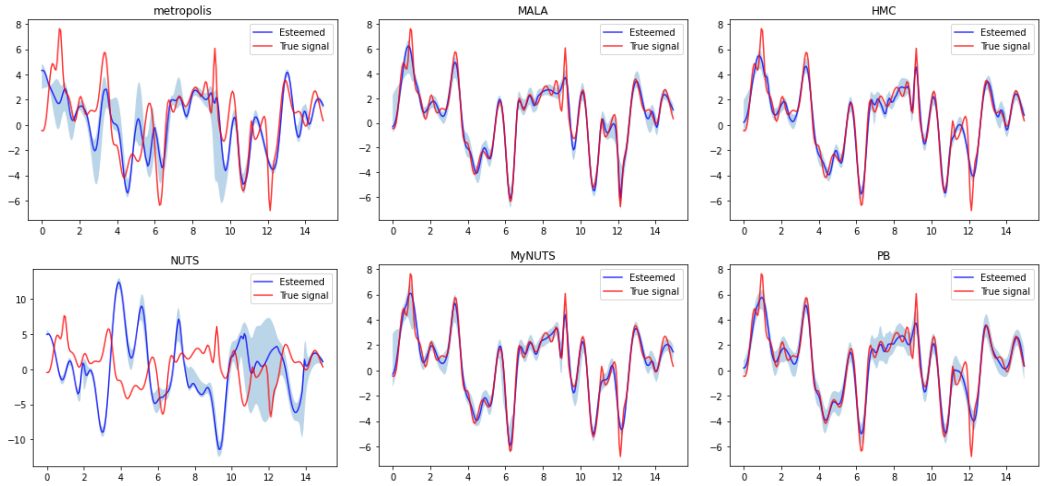

Figure 4.5: Confusion matrix and scatter plot illustrating the classification performance of various algorithms.

tional demand is necessary, and every effort must be made to fasten up the process. In the following section, we evaluate the performance of the NUTS, MyNUTS, and PB algorithms using a more realistic dataset that includes both MBH and compact binary wave signals. For such datastream the likelihood function has the same functional form as eq.**??** and eq. **??** but comprehend the superposition of each waves, for each class of wave.

$$-log(D|q, N_{CB}, N_{BHs}) = \sum_{i}^{T} \frac{(D_i - \sum_{cl}^{N_{CB},N_{BHs}} \sum_{j}^{N_{cl}} \Phi_i^{cl}(q_\Phi^j))^2}{\sigma^2} \tag{4.8}$$

As discussed previously in Sections X and Y, the integration time step and the mass matrix are the only hyperparameters of the system, and they are closely linked to each other to ensure accurate integration. The optimal values for these hyperparameters depend on the current position. Therefore, the first step in the procedure is to apply the Metropolis algorithm for a burn-in period, with the results serving as the starting point. From this optimal starting point, an integration time step is determined and fixed, with the requirement that no more than 30% of proposals are rejected over a window of steps. Finally, an optimal mass matrix must be chosen. By sequentially checking a window of steps, the mass matrix is adjusted to achieve a reasonable BMFI. **??**. This procedure can be performed in parallel runs, to generate a bigger and more accurate sample. This allow the system to better explore the space, given different starting points. The parallelization in carried out with the python library Ray [15].

4.2.1 Hypertriangularization

To improve the algorithm and its ability to retrieve the parameters space, other action, apart from changing the algorithm itself, can be made.
In fact the functional form of the likelihood can give insights of the composition of the parameters space which can be exploited to allow better performance. In fact, the two likelihood proposed in eq. 4.7 and eq. 4.4, present a strong symmetry under the permutation of the parameters referring to each wave. This means that if there are N wave in the signal, the algorithm has to infer Nk parameters where k is the number of variable to describe each wave. In doing so, if at the current iteration the samplers live point is at some configuration nothing stop the algorithm from proposing the same configuration but switched. This because since the likelihood contain N indistinguishable components, so for each peak there will be $N!$ copies related to the symmetry.
This characteristic rises many challenges to correctly sample the posterior without wasting time in spaces that are non-informative as well as the possibility to use this symmetry to speed up the sampler.
[6] proposed a method to overcome this problems by enforcing an order on 1 of the variables, describing each indistinguishable components, among all components.
This method restrict the space from the hyper-cube where the parameters lie with all the multimodality, to the hyper-triangle with the map $\phi : C \to T$. This can be done treating the sampler as a black-box and using a 1 to 1 transformation for the ordering:

$$q_i' = 1 - \prod_{j=1}^{i} (1 - x_j)^{\frac{1}{N+1-j}} \tag{4.9}$$

This map has the properties to be 1 on 1, hence preserving the correct prior space since its determinant is a constant.
However another method to solve this problem is proposed that aim to modify the HMC samplers. This is done with the same logic imposing an ordering among one of the representative value as before. The change is that this ordering is done on the first iteration of the algorithm itself and is preserved during the evolution of the Hamiltonian by imposing an hard constrain on those variables. For instance, when two of the ordered variable between 2 indistinguishable components reach a threshold value, i.e. they become too close to each other, they are reflected in opposite direction in the same view of a total elastic collision with a wall as depicted in fig. 4.6. It is necessary to reflects the variable instead of terminating the iteration since terminating upon reflection would reject all the sample close to that point which could instead be the real parameter. So in order to ensure the symmetry of the orbit a reflection mechanism is required.

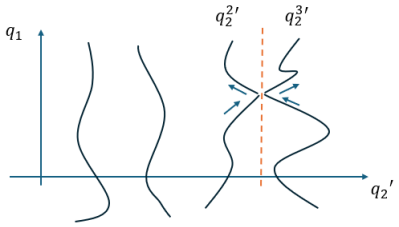

Figure 4.6: Graphical representation of the reflection mechanism to impose an ordering on the identical components of the likelihood to sample from the Hypertriangle instead

However this technique can rises some problem which the hypertriangularization of [6] is free of. The first problem is the introduction of another hyperparameter that identifies the limit for which two variable can come close togheter before being reflected. Moreover the initial ordering become important since for optimal performance is achieved when the initial configuration reflect the position on the true data space. Given the first inizialization, it can become crumbersome if the real parameters lie close togheter between other variable of the proposed first state. However due to the lack of studies about the validity of the hypertriangularization for a gradient base method, the proposed method is applied.

4.3 Results

In order to investigate the suitability of the presented algorithms with the proposed problem, a known signal is generated for both of the phenomena that have been described.

Chapter 5

Conclusion

Bibliography

[1] Michael Betancourt. *A General Metric for Riemannian Manifold Hamiltonian Monte Carlo*, page 327–334. Springer Berlin Heidelberg, 2013.

[2] Michael Betancourt. Diagnosing suboptimal cotangent disintegrations in hamiltonian monte carlo, 2016.

[3] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo, 2018.

[4] Nawaf Bou-Rabee and J. M. Sanz-Serna. Geometric integrators and the hamiltonian monte carlo method. *Acta Numerica*, 27:113–206, May 2018.

[5] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, May 2011.

[6] Riccardo Buscicchio, Elinore Roebber, Janna M. Goldstein, and Christopher J. Moore. Label switching problem in bayesian analysis for gravitational wave astronomy. *Physical Review D*, 100(8), October 2019.

[7] Kiam Choo. Learning hyperparameters for neural network models using hamiltonian dynamics. 10 2000.

[8] Samantha R. Cook, Andrew Gelman, and Donald B. Rubin. Validation of software for bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*, 15(3):675–692, 2006.

[9] Alain Durmus, Samuel Gruffaz, Miika Kailas, Eero Saksman, and Matti Vihola. On the convergence of dynamic implementations of hamiltonian monte carlo and no u-turn samplers, 2023.

[10] Daniel Foreman-Mackey. corner.py: Scatterplot matrices in python. *The Journal of Open Source Software*, 1(2):24, jun 2016.

[11] Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 73(2):123–214, 2011.

[12] Mark Girolami and Ben Calderhead. Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 73(2):123–214, 03 2011.

[13] Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo, 2011.

[14] Ziming Liu and Zheng Zhang. Quantum-inspired hamiltonian monte carlo for bayesian sampling, 2020.

[15] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications, 2018.

[16] Radford M Neal. Slice sampling. *The annals of statistics*, 31(3):705–767, 2003.

[17] Ulrich Paquet and Marco Fraccaro. An efficient implementation of riemannian manifold hamiltonian monte carlo for gaussian process models, 2018.

[18] Natesh S. Pillai. Optimal scaling for the proximal langevin algorithm in high dimensions, 2022.

[19] Christian P. Robert. The metropolis-hastings algorithm, 2016.

[20] A. Sokal. *Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms*, pages 131–192. Springer US, Boston, MA, 1997.

[21] Sean Talts, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. Validating bayesian inference algorithms with simulation-based calibration, 2020.

Appendices

Appendix A

Pseudocode

A.1 MALA

Algorithm 4 MALA kernel
given dt, q_i, M

procedure KERNEL
    select random $p \sim G(0, M)$
    $q \leftarrow \psi_{dt}(q, p)$ # resolve Langevin Equation
    $q_{new} \leftarrow acceptance(\alpha)$ decide new state with a Metropolis-Hastings update
end procedure

A.2 HMC

Algorithm 5 HMC kernel
given dt, L, q_i, M

procedure KERNEL
    select random $p_i \sim G(0, M)$
    while $j < L$ **do**
        $q, p \leftarrow \psi_{dt}(q, p)$ #apply symplettic one step integrator L times
    end while
    if $\alpha <$ random number $[0, 1]$ **then return** new q
    elsereturn old q
    end if
end procedure

Algorithm 6 Kernel(q)

$q_{\text{old}} \leftarrow \text{copy}(q)$
$p \leftarrow \text{momentum\_update}(q, \text{model})$
$u \leftarrow \text{exponential\_random}() - \text{Hamiltonian.E\_old}$
$(q_r, p_r, q_l, p_l, E_t) \leftarrow \text{first\_step}(q, p, u)$
$picked\_q \leftarrow \text{copy}(q)$
$j \leftarrow 1$
$stop \leftarrow 1$
while $stop = 1$ **do**
    $v \leftarrow \text{coin\_flip}() \,?\, 1 : -1$
    $dt \leftarrow dt \times v$
    if $v = 1$ **then**
        $(picked\_q, q_r, p_r, n, stop) \leftarrow \text{binary\_tree}(q_r, p_r, q_l, p_l, dt, j, u, picked\_q)$
    else
        $(picked\_q, q_l, p_l, n, stop) \leftarrow \text{binary\_tree}(q_l, p_l, q_r, p_r, dt, j, u, picked\_q)$
    end if
    $E_t \leftarrow E_t + E_{t\text{new}}$
    if $stop = 1$ **then**
        if $\text{accept}(E_{t\text{new}}, E_t)$ **then**
            $q \leftarrow \text{copy}(picked\_q)$
        end if
    end if
    $j \leftarrow j + 1$
end while
$rj \leftarrow (q_{\text{old}} = q) \,?\, 1 : 0$
return q, rj

Algorithm 7 binary_tree(q, p, q_{dir}, p_{dir}, dt, j, u, $picked\_q$)

$(q, p) \leftarrow$ Hamiltonian.integrator$(q, p, dt,$ model$)$
$E_1 \leftarrow$ Hamiltonian.Energy$(q, p,$ model$)$
$n_1 \leftarrow \text{int}(u \leq E_1)$
$stop \leftarrow$ Hamiltonian.inversion$(q, q_{\text{dir}}, p, p_{\text{dir}}) \times \text{int}\left(\frac{(E_1 - \text{Hamiltonian.E_old})}{\text{Hamiltonian.E_old}} \leq \text{err}\right)$
if $stop = 1$ **then**
    $(q_1, p_1) \leftarrow$ Hamiltonian.integrator$(q$.copy$(), p$.copy$(), dt,$ model$)$
    $E_2 \leftarrow$ Hamiltonian.Energy$(q, p,$ model$)$
    $n_2 \leftarrow \text{int}(u \leq E_1)$
    $stop \leftarrow$ Hamiltonian.inversion$(q, q_{\text{dir}}, p, p_{\text{dir}}) \times \text{int}\left(\frac{(E_2 - \text{Hamiltonian.E_old})}{\text{Hamiltonian.E_old}} \leq \text{err}\right)$
    if $stop = 1$ **then**
        $n \leftarrow n_1 + n_2$
        if $n = 0$ **then**
            return $picked\_q, q_1, p_1, n, stop$
        else
            if $\frac{n_1}{n} >$ random.uniform$(0, 1)$ **then**
                return $q, q_1, p_1, n, stop$
            else
                return $q_1, q_1, p_1, n, stop$
            end if
        end if
    end if
end if

Algorithm 8 tree(q, p, q_{dir}, p_{dir}, dt, j, u, $picked\_q$)

if $j = 1$ **then**
  return binary_tree(q, p, q_{dir}, p_{dir}, dt, j, u, $picked\_q$)
else
  $(picked\_q_1, q, p, n_1, stop) \leftarrow \text{binary\_tree}(q, p, q_{\text{dir}}, p_{\text{dir}}, dt, j-1, u, picked\_q)$
  if $stop = 1$ **then**
    $(picked\_q_2, q, p, n_2, stop) \leftarrow \text{binary\_tree}(q, p, q_{\text{dir}}, p_{\text{dir}}, dt, j-1, u, picked\_q)$
    if $stop = 1$ **then**
      $n \leftarrow n_1 + n_2$
      if $n = 0$ **then**
        return $picked\_q, q, p, n, stop$
      else
        if $\frac{n_1}{n} > \text{random.uniform}(0, 1)$ **then**
          return $picked\_q_1, q, p, n, stop$
        else
          return $picked\_q_2, q, p, n, stop$
        end if
      end if
    end if
  end if
end if
return $0, 0, 0, 0, 0$

Algorithm 9 first_step(q, p, u)

$v \leftarrow \text{int}(\text{random.uniform}() < 0.5) \times 2 - 1$
$dt \leftarrow \text{self.dt} \times v$
if $v = 1$ **then**
  $(q_1, p_1) \leftarrow \text{Hamiltonian.integrator}(q.\text{copy}(), p.\text{copy}(), dt, \text{self.model})$
  $E_1 \leftarrow \text{Hamiltonian.Energy}(q_1, p_1, \text{self.model})$
  return $q_1, p_1, q, p, 1 + \text{int}(u \leq E_1)$
else
  $(q_1, p_1) \leftarrow \text{Hamiltonian.integrator}(q.\text{copy}(), p.\text{copy}(), dt, \text{self.model})$
  $E_1 \leftarrow \text{Hamiltonian.Energy}(q_1, p_1, \text{self.model})$
  return $q, p, q_1, p_1, 1 + \text{int}(u \leq E_1)$
end if

A.3 NUTS

A.4 MyNUTS

Algorithm 10 Kernel(q)

$q_{\text{old}} \leftarrow q$.copy()
$p \leftarrow$ self.Hamiltonian.momentum_update(q, self.model)
$q_l \leftarrow q$.copy()
$q_r \leftarrow q$.copy()
$p_l \leftarrow p$.copy()
$p_r \leftarrow p$.copy()
$j \leftarrow 1$
$n \leftarrow 1$
$stop \leftarrow 1$
while $stop = 1$ **do**
    $v \leftarrow$ int(random.uniform() $< 0.5) \times 2 - 1$
    if $v = 1$ **then**
        $(q_r, p_r,$ new_q$, n2, stop2) \leftarrow$ build_tree$(q_r, p_r, q_l, p_l, q, v, j)$
    else
        $(q_l, p_l,$ new_q$, n2, stop2) \leftarrow$ build_tree$(q_r, p_r, q_l, p_l, q, v, j)$
    end if
    $n \leftarrow n + n2$
    log_sum_exp $\leftarrow -$self.Hamiltonian.E_old $+ \log(n + n2)$
    stop $\leftarrow stop2 \times$ self.Hamiltonian.inversion(q_r, q_l, p_r, p_l)
    log_sum_exp1 $\leftarrow -$self.Hamiltonian.E_old $+ \log(n)$
    if stop $= 1$ and random.uniform() $<$ exp(log_sum_exp1 $-$ log_sum_exp) **then**
        $q \leftarrow$ new_q.copy()
    end if
    $j \leftarrow j + 1$
end while
if $(q_{\text{old}} == q)$.all() **then**
    $rj \leftarrow 1$
else
    $rj \leftarrow 0$
end if
return q, rj

Algorithm 11 build_tree(q_r, p_r, q_l, p_l, $picked\_q$, v, j)

$i \leftarrow 0$
$stop \leftarrow 1$
$n \leftarrow 0$
while $i < 2^{(j-1)}$ and $stop = 1$ **do**
    self.dt $\leftarrow v \times$ abs(self.dt)
    if $v = 1$ **then**
        $q_r, p_r, n, picked\_q, stop \leftarrow$ step_direction(self.dt, $q_r, p_r, q_l, p_l, n, picked\_q$)
    else
        $q_l, p_l, n, picked\_q, stop \leftarrow$ step_direction(self.dt, $q_l, p_l, q_r, p_r, n, picked\_q$)
    end if
    $i \leftarrow i + 1$
end while
if $v = 1$ **then**
    return $q_r, p_r, picked\_q, n, stop$
else
    return $q_l, p_l, picked\_q, n, stop$
end if

Algorithm 12 step_direction(dt, q, p, q_2, p_2, n, $picked\_q$)

$(q, p) \leftarrow$ self.Hamiltonian.integrator$(q, p, dt,$ self.model$)$
$E_{\text{new}} \leftarrow$ self.Hamiltonian.Energy$(q, p,$ self.model$)$
$stop \leftarrow$ self.Hamiltonian.inversion$(q, q_2, p, p_2) \times \text{int}\left(\frac{(E_{\text{new}} - \text{self.Hamiltonian.E_old})}{\text{self.Hamiltonian.E_old}} \leq \text{self.err}\right)$
if $stop$ **then**
    $(q, p) \leftarrow$ self.model.reflection(q, p)
    $(picked\_q, n) \leftarrow$ check$(q, p, n, picked_q, E_{\text{new}})$
end if
return $q, p, n, picked\_q, stop$

Algorithm 13 check(q, p, n, $picked\_q$, E_{new})

$n2 \leftarrow \exp(-E_{\text{new}} +$ self.Hamiltonian.E_old$)$
log_sum_exp $\leftarrow -$self.Hamiltonian.E_old $+ \log(n + n2)$
if $\exp(-E_{\text{new}} -$ log_sum_exp$) >$ random.uniform() **then**
    $picked\_q \leftarrow q$.copy()
end if
return $picked\_q, n + n2$