Hamiltonian Monte Carlo

220014792

April 2025

Contents

1 Introduction

1.1 The drawbacks of Metropolis-Hastings

The Metropolis-Hastings algorithm (MH) remains a widely used tool for simulating observations from complex distributions and applying Monte Carlo methods (Hitchcock 2003, Chap. 1). However, it struggles in high-dimensional or complex target distributions, where the typical set's density becomes negligible compared to the rest of the space. This leads to low acceptance rates when the proposal distribution is too wide. While reducing variance will increase acceptance rates, it slows exploration, causing the chain to move slowly through parameter space and biasing the chain if not run for a great number of samples, as in figure 1 (M. Betancourt 2018, Chap. 2).

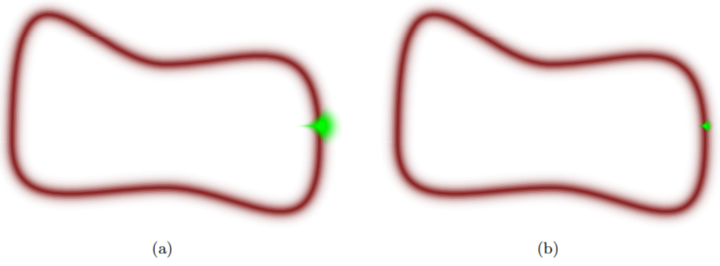

Figure 1: a) If the proposal variances are large, the majority of proposed parameter values will fall out with the typical set and be rejected. b) Decreasing the variance of the proposal distribution will increase the acceptance rate but considerably slow exploration (M. Betancourt 2018, Chap. 3).

This means that a better sampling method is necessary - one that can move more efficiently through multiple dimensions of parameter space and make more educated proposals as to explore the typical set in fewer iterations.

1.2 A Solution from Physics

Hamiltonian dynamics reformulate classical mechanics by describing a system in terms of position variables $\underline{q}(t)$ and momenta $\underline{p}(t)$. The Hamiltonian function defines the system's total energy, and Hamilton's equations govern its time evolution in *phase space* - a $2n$-dimensional space combining n positions and n momenta. These equations generate a Hamiltonian-dependent vector field that defines the system's flow through phase space (Etxebarria 2024, Chap. 6).

Definition 1. *The Hamiltonian $H(\underline{q},\underline{p})$ combines the potential energy $U(\underline{q})$ and kinetic energy $K(\underline{q},\underline{p})$:*

$$H(\underline{q},\underline{p}) = U(\underline{q}) + K(\underline{q},\underline{p}) \tag{1}$$

where: $\underline{q}$ is the position (model parameters) and $\underline{p}$ is the momentum (an auxiliary variable).

Definition 2. *The evolution of the system is governed by Hamilton's equations:*

$$\frac{d\underline{q}}{dt} = \frac{\partial H}{\partial \underline{p}} \tag{2a}$$

$$\frac{d\underline{p}}{dt} = -\frac{\partial H}{\partial \underline{q}} \tag{2b}$$

As the total energy of a closed system stays constant over time, and the Hamiltonian for that system is *just* this total energy, we can describe a Hamiltonian as invariant in time.

$$\frac{dH}{dt} = \sum_{i=1}^{d} \left(\frac{dq_i}{dt}\frac{\partial H}{\partial q_i} + \frac{dp_i}{dt}\frac{\partial H}{\partial p_i} \right) = \sum_{i=1}^{d} \left(\frac{\partial H}{\partial p_i}\frac{\partial H}{\partial q_i} - \frac{\partial H}{\partial q_i}\frac{\partial H}{\partial p_i} \right) = 0$$

This is an important property of the Hamiltonian in a closed system (Neal 2012, Chap. 2).

1.3 Applying Hamiltonian Dynamics to Statistics

By interpreting the target posterior as a potential field, we can apply Hamiltonian mechanics - using the Hamiltonian function, its equations, and phase space - to construct a Markov chain that leverages the geometry of the distribution for efficient exploration. When combined with Monte Carlo methods, this enables statistical inference much like Metropolis-Hastings. Originally developed by Duane et al. in 1987 for quantum chromodynamics and called *Hybrid Monte Carlo* (Duane et al. 1987), the method - now known as *Hamiltonian Monte Carlo* (HMC) - offers a powerful alternative to traditional samplers, especially in high-dimensional settings (LIVINGSTONE et al. 2019, Chap. 1).

2 The Hamiltonian Markov Chain

2.1 Conceptualisation

A Hamiltonian Markov chain requires the generation of a vector field across an extension of the parameter space which aligns with the typical set. Then, using the directional property of this vector field to inform Markov chain transitions, we allow for a more precise exploration. See figure 2.

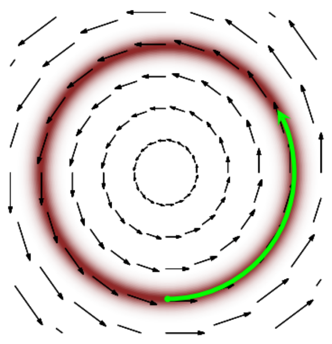

Figure 2: A vector field that aligns with the typical set would allow more informed movement from one sample to the next (M. Betancourt 2018, Chap. 2).

The geometry of a target distribution can be explored using principles from Hamiltonian dynamics. This parallels a physical system: consider a satellite near a planet. With too little momentum, it crashes; too much, and it escapes. But with just enough, it enters orbit - conserving total energy while trading between kinetic and potential forms. This stable trajectory, determined by the satellite's initial position and momentum (its phase space coordinates), traces out a *level set* - a path of constant total energy (M. Betancourt 2018, Chap. 3).

We apply the same idea to sampling: by augmenting parameters with momentum variables, we define a Hamiltonian system over phase space. The resulting dynamics trace level sets of the target distribution's geometry, guiding a Markov chain to *orbit* the typical set, leading to high acceptance rates.

2.2 Formalisation

We assume that we want to sample from the improper posterior $\pi(\underline{q}|\underline{x})$, where $\underline{\text{q}}$ is an n-dimensional vector of parameters and $\underline{\text{x}}$ is the observed data. We will introduce an auxiliary parameter vector $\underline{\text{p}}$ to match $\underline{\text{q}}$, and thus map onto a *phase space*,

$$\underline{q} \mapsto (\underline{q}, \underline{p})$$

giving us $2n$ parameters in total. We will think of our $\underline{\text{q}}$ parameter as position (in parameter space) and the $\underline{\text{p}}$ as momentum. Together, $\underline{\text{q}}$ and $\underline{\text{p}}$ define the phase space in which the vector field will be created. Now consider the joint density function of $\underline{\text{q}}$ and $\underline{\text{p}}$: $\pi(\underline{q}, \underline{p}|\underline{x})$. We will omit the conditional on $\underline{\text{x}}$ and refer to this as the canonical density. Applying Bayes' theorem to the canonical density, we can make it conditional over $\underline{\text{p}}$,

$$\pi(\underline{q}, \underline{p}) = \pi(\underline{p}|\underline{q})\pi(\underline{q}),$$

so that our desired posterior density, $\pi(\underline{q})$, can be easily calculated by marginalising out $\underline{\text{p}}$. Now choose the Hamiltonian $H(\underline{q}, \underline{p})$ to be the negative log of the canonical density:

$$\begin{aligned} H(\underline{q}, \underline{p}) &= -\log\Big(\pi(\underline{q}, \underline{p})\Big) \\ &= -\log\Big(\pi(\underline{p} \mid \underline{q})\pi(\underline{q})\Big) \\ &= -\log\Big(\pi(\underline{p} \mid \underline{q})\Big) - \log\Big(\pi(\underline{q})\Big) \end{aligned}$$

We now define the kinetic and potential energies:

$$\begin{aligned} &= \underbrace{-\log\Big(\pi(\underline{p} \mid \underline{q})\Big)}_{K(\underline{p},\underline{q})} + \underbrace{-\log\Big(\pi(\underline{q})\Big)}_{V(\underline{q})} \\ &= K(\underline{p}, \underline{q}) + V(\underline{q}) \end{aligned}$$

To derive the vector field, use Hamilton's equations 2a and 2b:

$$\frac{d\underline{q}}{dt} = \frac{\partial H}{\partial \underline{p}} = \frac{\partial K(\underline{p}, \underline{q})}{\partial \underline{p}} = \frac{\partial}{\partial \underline{p}}\Big[-\log\Big(\pi(\underline{p} \mid \underline{q})\Big)\Big] \tag{3}$$

$$\frac{d\underline{p}}{dt} = -\frac{\partial H}{\partial \underline{q}} = -\frac{\partial K(\underline{p}, \underline{q})}{\partial \underline{q}} - \frac{\partial V(\underline{q})}{\partial \underline{q}} = \frac{\partial}{\partial \underline{q}}\Big[\log\Big(\pi(\underline{p} \mid \underline{q})\Big) + \log\Big(\pi(\underline{q})\Big)\Big] \tag{4}$$

These vector equations represent the trajectory followed by any point in the phase space. Following these vectors will carry a particle along a level set corresponding to the initial conditions, as discussed earlier.

2.3 The Ideal Case

Given an initial point in the parameter space, $\underline{q}_0$, we can sample a corresponding initial momentum from $\pi(\underline{p}|\underline{q})$, the conditional distribution of momentum, given a position. If $\underline{q}_0$ is in the typical set, then $(\underline{q}_0, \underline{p}_0)$ should be in the typical set of the phase space. Now by integrating the equations 3 and 4 over some time interval t, we can construct a trajectory along a level set,

$$(\underline{q_0}) \mapsto (\underline{q_0}, \underline{p_0}) \mapsto \phi_t(\underline{q_0}, \underline{p_0}) = (\underline{q_1}, \underline{p_1}),$$

where $\phi_t(\underline{q_0}, \underline{p_0})$ describes the new position that the trajectory reaches after integrating over chosen time t. By marginalising for $\underline{\mathrm{q}}$, we can obtain the new position in the original parameter space:

$$(\underline{q_1}, \underline{p_1}) \mapsto (\underline{q_1}).$$

Iterating this process will define the Hamiltonian Markov chain. If far from the typical set, it can rapidly converge to areas of greater density, where it will efficiently propose samples from the target distribution (M. Betancourt 2018, Chap 3.). This process can be split into a deterministic step - integrating over the fixed level set - and a stochastic step, a random walk between level sets - see figure 3.

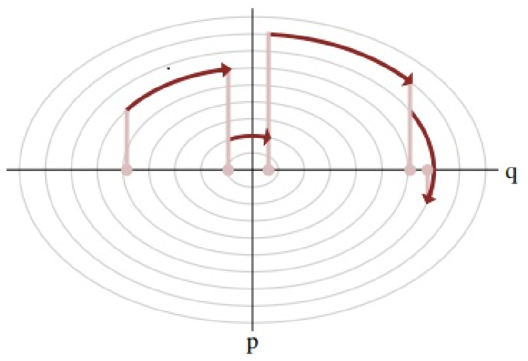

Figure 3: By mapping to phase space, adding a randomly sampled momentum, and following the level set before mapping back to the parameter space, informed proposals can be made. Note this diagram assumes a 1-D parameter space.

2.4 Choosing a Distribution for the Momentum

To optimise the stochastic step, we must choose a suitable conditional distribution for the momentum, $\pi(\underline{p}|\underline{q})$. The form of this distribution is crucial as it informs the random walk step, where momentum is resampled. The choice involves a trade-off between simplicity and accuracy. We want the resampling step to be simple to reduce computational time, but the distribution must also align with the geometric properties of the target distribution to define uniform level sets and improve movement efficiency. We will discuss two main options:

2.4.1 Euclidean Hamiltonian Monte Carlo

We employ:

$$\pi(\underline{p}|\underline{q}) = N(\underline{p}|0, M),$$

which defines a *Euclidean-Gaussian* kinetic energy;

$$K(\underline{q}, \underline{p}) = \frac{1}{2} p^T \times M^{-1} \times p + \log|M| + const.$$

Here, M is a fixed $n \times n$ covariance matrix. Ideally, M should be set to the inverse of the target covariance matrix, such that

$$M^{-1} = \mathbb{E}[(\underline{q} - \underline{u})(\underline{q} - \underline{u})^T].$$

To do this in practise, it is common to run the chain with a standard mass matrix, e.g. I_n, to begin with and use Monte Carlo integration to estimate the covariances, which can then be used to amend the mass matrix (Neal 2012; M. Betancourt 2018). This is the most common distribution for sampling momentum form due to its simplicity, however it does suffer from reduced performance as the fixed nature of the mass matrix will deal better in some areas of the target distribution than others.

2.4.2 Riemannian Hamiltonian Monte Carlo

Alternatively, should we seek more precision - at the expense of computational time - we use:

$$\pi(\underline{p}|\underline{q}) = N(\underline{p}|0, \Sigma(\underline{q})),$$

which defines a *Riemannian-Gaussian* kinetic energy:

$$K(\underline{q}, \underline{p}) = \frac{1}{2} p^T \times \Sigma^{-1}(\underline{q}) \times p + \log|\Sigma(\underline{q})| + const.$$

where ideally $\Sigma(\underline{q})$ is the Hessian of the target distribution. This result is derived from the spatially-dependent Riemannian-metric tensor, producing a mass matrix which is also spatially-dependent. This adapts the sampler to local curvature in the target, improving efficiency. Non-Gaussian forms for $\pi(\underline{q}, \underline{p})$ do exist, but it is generally accepted that the Gaussian is optimal, due to the Central Limit Theorem applying to the momentum (M. Betancourt 2018, Chap. 4).

2.5 Choosing Integration Time and Symplectic Integrators

Optimising the integration time t is crucial for effective HMC. A small t leads to slow exploration, while a large t risks inefficiently retracing the level set, or moving too far from the typical set, reducing acceptance rate. The optimal t depends on the level set, making tuning sensitive for each target distribution. Since analytical integration is impractical, we rely on numerical integrators, which suffer from *drift*. This drift causes the numerical solution to deviate from the level set over time, requiring careful selection of integrators to mitigate error.

2.5.1 Symplectic Integrators

A symplectic integrator is a numerical integration method, designed for the integration of Hamilton's equations, 2a and 2b. Symplectic integrators are useful here, because they preserve the total energy of the system. This means that the integrator *corrects itself* back onto the level set of constant energy - effectively oscillating around the level set as they move across it. The most common example is the *leapfrog integrator*. The algorithm will use a step-size ϵ and iterate $\frac{t}{\epsilon}$ times, following:

1. **Initialize**: Start with positions $\underline{q_0}$ and momenta $\underline{p_0}$
2. **Update Momentum**: $\underline{p} \leftarrow \underline{p} - \frac{\epsilon}{2}\frac{\partial V(\underline{q})}{\partial \underline{q}}$
3. **Update Position**: $\underline{q} \leftarrow \underline{q} + \epsilon M^{-1}\underline{p}$

4. **Update Momentum Again**: $\underline{p} \leftarrow \underline{p} - \frac{\epsilon}{2}\frac{\partial V(\underline{q})}{\partial \underline{q}}$

5. **Repeat**: Repeat for multiple steps to simulate phase space evolution.

The algorithm alternates between updating $\underline{p}$ and $\underline{q}$, preserving energy and ensuring reversibility. (Cook 2020). The end point will be the next *proposal.* Note that some integrators are better than others for a specific target distribution.

2.5.2 Corrections using a Metropolis Step

While symplectic integrators such as leapfrog are much better than standard integrators, they do still have small associated errors which introduce biases. To correct these, we can draw upon the concept of a *Metropolis acceptance* step. Define the probability of proposing $(\underline{q}', \underline{p}')$, given previous position $(\underline{q_0}, \underline{p_0})$, as $\mathbb{Q}(\underline{q}', \underline{p}'|\underline{q_0}, \underline{p_0})$. For the Metropolis acceptance to work in this context, we want

$$\mathbb{Q}(\underline{q}', \underline{p}'|\underline{q_0}, \underline{p_0}) = \mathbb{Q}(\underline{q_0}, \underline{p_0}|\underline{q}', \underline{p}')$$

which requires us to augment the momentum by switching its sign after conducting the integration (M. Betancourt 2018, Chap. 5).

$$(\underline{q}', \underline{p}') \mapsto (\underline{q}', \underline{-p}')$$

Now, assume that integrating along the level set, from point $(\underline{q_0}, \underline{p_0})$, for time t leaves us with a new point $(\underline{q_t}, \underline{p_t})$. Due to the deterministic nature of the integration step, we can write

$$\mathbb{Q}(\underline{q}', \underline{p}'|\underline{q_0}, \underline{p_0}) = \delta(\underline{q}' - \underline{q_t})\delta(\underline{p}' - \underline{p_t}).$$

Thus, the acceptance probability for the point we have reached via integration, $(\underline{q_t}, \underline{p_t})$, is

$$\begin{aligned} a(\underline{q_t}, -\underline{p_t} \mid \underline{q_0}, \underline{p_0}) &= \min\left(1, \frac{\mathbb{Q}(\underline{q_0}, \underline{p_0} \mid \underline{q_t}, -\underline{p_t})\,\pi(\underline{q_t}, -\underline{p_t})}{\mathbb{Q}(\underline{q_t}, -\underline{p_t} \mid \underline{q_0}, \underline{p_0})\,\pi(\underline{q_0}, \underline{p_0})}\right) \\ &= \min\left(1, \frac{\delta(\underline{q_t} - \underline{q_t})\,\delta(-\underline{p_t} + \underline{p_t})\,\pi(\underline{q_t}, -\underline{p_t})}{\delta(\underline{q_0} - \underline{q_0})\,\delta(\underline{p_0} - \underline{p_0})\,\pi(\underline{q_0}, \underline{p_0})}\right) \end{aligned}$$

$$= \min\left(1, \frac{\pi(\underline{q_t}, -\underline{p_t})}{\pi(\underline{q_0}, \underline{p_0})}\right) = \min\left(1, \frac{\exp\Big(-H(\underline{q_t}, -\underline{p_t})\Big)}{\exp\Big(-H(\underline{q_0}, \underline{p_0})\Big)}\right) = \min\Big(1, \exp\Big(-H(\underline{q_t}, -\underline{p_t}) + H(\underline{q_0}, \underline{p_0})\Big)\Big),$$

as adapted from the derivation shown in (M. Betancourt 2018, Chap. 5).

3 Implementation

We now have the conceptual and practical tools to implement HMC in practise.

1. Begin chain at position $\underline{q_0}$ in parameter space.

2. Sample a new momentum $\underline{p_0}$ from $\pi(\underline{p} \mid \underline{q})$, e.g., $N(\underline{p} \mid 0, M)$.

3. Map $\underline{q_0}$ to $(\underline{q_0}, \underline{p_0})$ in phase space.

4. Integrate Hamilton's equations using the leapfrog method to obtain $(\underline{q_1}, \underline{p_1})$.

5. Flip the momentum: $(\underline{q}_1, \underline{p}_1) \mapsto (\underline{q}_1, -\underline{p}_1)$.
6. Accept the proposal with probability: $a = \min\left(1, \exp\left(-H(\underline{q}_1, \underline{p}_1) + H(\underline{q}_0, \underline{p}_0)\right)\right)$.
7. If accepted, set $\underline{q}_{\text{next}} = \underline{q}_1$; else, set $\underline{q}_{\text{next}} = \underline{q}_0$.
8. Return to step 2 using $\underline{q}_{\text{next}}$ as the new starting point.

3.1 A Comparison of HMC and MH in Python

Here, we implement HMC and MH in Python in order to sample from the posterior of a simple density function. In this case, we will choose a bivariate normal distribution, given by

$$\pi\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}\right).$$

where the covariance, and correlation, between x and y is 0.8. We choose:

- Initial points at $x = y = 0$ for both samplers.
- Both samplers generate 20,000 samples, with a burn-in of 5,000.
- The proposal distribution for MH is $\mathcal{N}(\underline{0}, k\Sigma)$ where Σ is the covariance matrix defined in the posterior and k is a constant equal to 5.7.
- For HMC, $\pi(\underline{p}|\underline{q}) = N(\underline{0}, \Sigma^{-1})$ (Euclidian-Gaussian with optimised covariance).
- The integration time for HMC is 50 with a leapfrog step-size of 0.5 (so 100 steps).

The scaling factor, k, for the MH proposal is selected to be 5.7 to produce an acceptance rate of around 0.234, which is considered optimal (Roberts et al. 1997, Chap. 1). Similarly, the integration and step sizes for HMC are chosen to produce a higher acceptance rate, which is better for HMC (M. J. Betancourt et al. 2014, Chap. 1). Note that the MH proposal and the HHC momentum distribution are using the posterior covariance. In reality, this would have to be estimated in order to be used. See the full code in appendix A.

3.2 Results

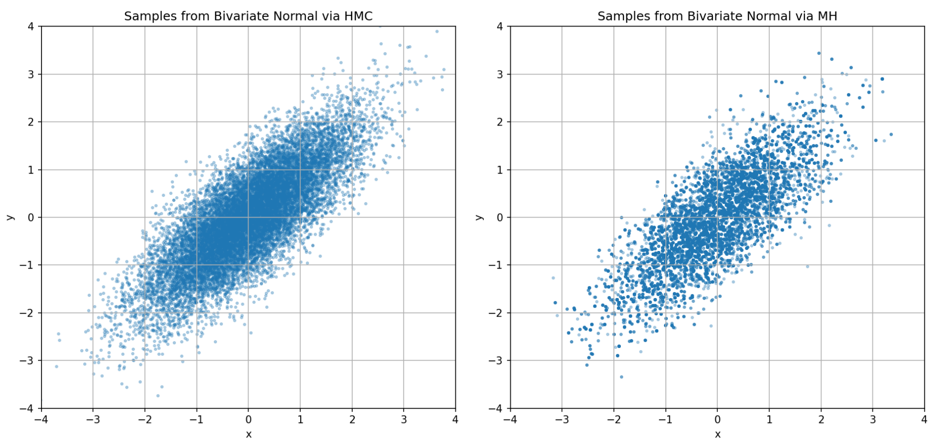

Figure 4: HMC produces a similar cloud to MH. The more *dense* look is due to the greater acceptance rate.

Using Monte Carlo methods, we estimate the means and variances for x and y.

	HMC	**MH**
Acceptance rate	0.970	0.233
Estimated mean	$[-0.0017,\ 0.0034]$	$[-0.0042,\ -0.018]$
Estimated variance	$[0.9851,\ 0.9885]$	$[0.9635,\ 0.9883]$
Effective sample size (ESS)	$[7895,\ 7951]$	$[1603,\ 1566]$

Table 1: Comparison of HMC and MH performance metrics.

The estimations for mean and variance from both HMC and MH are very close to the true values of [0,0] and [1,1], although the estimations from HMC are better. The geometry-informed proposal strategy of HMC permits it to move smoothly through parameter space, despite operating at a very high acceptance rate, while MH optimises at a very low acceptance rate. HMC can propose larger jumps to new positions without compromising acceptance rate, thus leading to a much lesser autocorrelation - as seen in figure 5 in appendix A. In contrast, MH is forced to use a lower acceptance rate as exploration would be too slow otherwise. As a result, the effective sample size from HMC is much higher than that of MH - as in table 1. In this simple case, with both HMC and MH being optimised to the target covariance, HMC is clearly more effective.

4 Drawbacks of Hamiltonian Monte Carlo

- HMC struggles with targets that have sharp curvature or discontinuous gradients.
- The method assumes continuity and needs modification for discrete parameters.
- Knowledge of the gradient is required, so HMC cannot be used if gradients are unavailable or too difficult to compute. This is also computationally intensive.
- Poor choices for the momentum distribution, $\pi(\underline{p}|\underline{q})$, can degrade performance.
- Tuning the integration time t is crucial, as performance is highly sensitive to it. See No-U-turn Sampling in appendix A.

5 Conclusion

Generally speaking, Hamiltonian Monte Carlo is a considerable improvement on other samplers such as Metropolis-Hastings or Gibbs. Due to its utilisation of the geometric properties of the target distribution, the Hamiltonian Markov chain proposes new positions that are much more likely to be accepted - allowing the chain to move through the parameter space much faster than its counterparts. This use of differential geometry in informing transitions makes it particularly powerful in higher dimensions, where other sampling methods become extremely weak. Depending on the application, HMC can be augmented in order to better adapt to more complex target distributions, for example by employing a Riemannian-Gaussian form for $\pi(\underline{p}|\underline{q})$. As shown in the implementation section, HMC can prove far superior to Metropolis-Hastings even in simple applications. Extrapolating this, should the computational power be available, it is a very strong choice for a sampler in Bayesian inference.

A Appendix

A.1 No-U-turn Sampling

Tuning the integration time t is one of the main practical challenges of HMC, as no single value is optimal across the entire target space. Instead, efficiency improves when t is adapted per trajectory. No-U-turn Sampling (NUTS) automates this by dynamically selecting both step size and integration time, halting each trajectory just before it begins to re-trace its path (Hoffman et al. 2014, Chap. 3). When Hamiltonian Monte Carlo is used in practise, it is implemented in Stan - a *C++* library for Bayesian modelling which employs NUTS (Stan Development Team 2025). In R, Stan can be used via the package RStan which works very much like Nimble. NUTS is a very valuable tool, as it prevents the need to sensitively tune the integration time and step size used by the leapfrog integrator in the HMC algorithm - which can be a time-consuming and exhaustive process.

A.2 Python Code for Comparison Between HMC and MH

A.2.1 HMC Function

```
import numpy as np
import matplotlib.pyplot as plt

def HMC(
neg_log_prob, grad_neg_log_prob,
initial_position,
M,
path_len=1.0, step_size=0.1,
n_samples=1000,

):

position = initial_position.copy()
dim = len(position)
samples = [position.copy()]
steps = int(path_len / step_size)
momentum_dist = lambda: np.random.multivariate_normal(mean=np.zeros(dim), cov=M)
M_inv = np.linalg.inv(M)

accepted = 0

for _ in range(n_samples):
    q0 = position.copy()
    p0 = momentum_dist()
    q = q0.copy()
    p = p0.copy()

    # Initial half step for momentum
    p -= 0.5 * step_size * grad_neg_log_prob(q)

    # Leapfrog steps
    for _ in range(steps):
        q += step_size * M_inv @ p
        if _ != steps - 1:
            p -= step_size * grad_neg_log_prob(q)

    # Final half step for momentum
    p -= 0.5 * step_size * grad_neg_log_prob(q)

    # Flip momentum to make proposal symmetric
    p = -p

    current_H = neg_log_prob(q0) + 0.5 * p0 @ M_inv @ p0
    proposed_H = neg_log_prob(q) + 0.5 * p @ M_inv @ p


    acceptance_prob = min(1, np.exp(current_H - proposed_H))

    if np.random.rand() < acceptance_prob:
        position = q.copy()
        accepted += 1
    samples.append(position.copy())

print("HMC acceptance rate:", accepted/n_samples)
return np.array(samples)
```

A.2.2 MH Function - for comparison

```
def MH(
    neg_log_prob,
    initial_position,
    proposal_cov,
    scaling_factor,
    n_samples=1000
):
    position = initial_position.copy()
    samples = [position.copy()]

    accepted = 0

    for _ in range(n_samples):
        proposal = position + np.random.multivariate_normal(np.array([0, 0]),
                                                scaling_factor * proposal_cov)
        acceptance_prob = min(1, np.exp(-neg_log_prob(proposal) + neg_log_prob(position))
        if np.random.rand() < acceptance_prob:
            position = proposal
            accepted += 1
        samples.append(position.copy())

    print("MH acceptance rate:", accepted/n_samples)
    return np.array(samples)
```

A.2.3 Defining the Potential and Kinetic Energies

```
def make_multivariate_normal(mu, cov):
cov_inv = np.linalg.inv(cov)

def neg_log_prob(x):
    diff = x - mu
    return 0.5 * diff @ cov_inv @ diff.T

def grad_neg_log_prob(x):
    return cov_inv @ (x - mu)

return neg_log_prob, grad_neg_log_prob
```

A.2.4 Defining the Parameters for the Bivariate Gaussian Example

```
# 2D Gaussian with correlation
mu = np.array([0.0, 0.0])
cov = np.array([[1.0, 0.8],
                [0.8, 1.0]])
neg_log_prob, grad_neg_log_prob = make_multivariate_normal(mu, cov)
```

A.2.5 Generating the Samples (Running Functions)

```
HMC_samples = HMC(
neg_log_prob=neg_log_prob,
grad_neg_log_prob=grad_neg_log_prob,
initial_position=np.array([0.0, 0.0]),
step_size=0.5,
path_len=5,
n_samples=20000,
M = np.linalg.inv(cov)
)
# Burn-in of 5000
HMC_samples = HMC_samples[5000:, :]

MH_samples = MH(
    neg_log_prob=neg_log_prob,
    initial_position=np.array([0.0, 0.0]),
    proposal_cov = cov,
    scaling_factor = 5.7,
    n_samples=20000
)
# Burn-in of 5000
MH_samples = MH_samples[5000:, :]
```

A.2.6 Print Statistics and Graphs

```
print("HMC estimates the mean to be:", np.mean(HMC_samples, axis=0))
print("HMC estimates the variance to be:", np.var(HMC_samples, axis=0))

print("MH estimates the mean to be:", np.mean(MH_samples, axis=0))
print("MH estimates the variance to be:", np.var(MH_samples, axis=0))

# Create a figure with two subplots side by side
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# HMC plot
axes[0].scatter(HMC_samples[500:, 0], HMC_samples[500:, 1], alpha=0.3, s=5)
axes[0].set_title("Samples from Bivariate Normal via HMC")
axes[0].set_xlabel("x")
axes[0].set_ylabel("y")
axes[0].set_xlim(-4,4)
axes[0].set_ylim(-4,4)
#axes[0].axis('equal')
axes[0].grid(True)

# MH plot
axes[1].scatter(MH_samples[500:, 0], MH_samples[500:, 1], alpha=0.3, s=5)
axes[1].set_title("Samples from Bivariate Normal via MH")
axes[1].set_xlabel("x")
axes[1].set_ylabel("y")
axes[1].set_xlim(-4,4)
axes[1].set_ylim(-4,4)
#axes[1].axis('equal')
axes[1].grid(True)

# Show the plot
plt.tight_layout()
#plt.show()

import arviz as az
```

```
HMC_array = np.expand_dims(np.array(HMC_samples), axis=0)
MH_array = np.expand_dims(np.array(MH_samples), axis=0)

# Wrap into InferenceData
HMC_idata = az.from_dict(posterior={"x": HMC_array})
MH_idata = az.from_dict(posterior={"x": MH_array})

# Compute ESS
HMC_ess = az.ess(HMC_idata)
MH_ess = az.ess(MH_idata)

print("HMC ESS:", HMC_ess)
print("MH ESS:", MH_ess)

from statsmodels.graphics.tsaplots import plot_acf

from statsmodels.graphics.tsaplots import plot_acf

fig, axs = plt.subplots(2, 2, figsize=(12, 8))

# HMC - Dimension 1
plot_acf(HMC_samples[:, 0], lags=50, ax=axs[0, 0])
axs[0, 0].set_title("HMC ACF - Dimension 1")

# HMC - Dimension 2
plot_acf(HMC_samples[:, 1], lags=50, ax=axs[0, 1])
axs[0, 1].set_title("HMC ACF - Dimension 2")

# MH - Dimension 1
plot_acf(MH_samples[:, 0], lags=50, ax=axs[1, 0])
axs[1, 0].set_title("MH ACF - Dimension 1")

# MH - Dimension 2
plot_acf(MH_samples[:, 1], lags=50, ax=axs[1, 1])
axs[1, 1].set_title("MH ACF - Dimension 2")

plt.tight_layout()
plt.suptitle("Autocorrelation Comparison: HMC vs MH", fontsize=16, y=1.03)
plt.show()
```

This code has been based on a program by (Jake 2020) which itself cites (Carrol 2019) for the generalised program. It requires the explicit definition of a target distribution's log-density (log_prob(x)), as well as its derivative (grad_log_prob(x)). Once this is complete, a Euclidian-Gaussian kinetic energy can be selected and input into the HMC function by changing M. As discussed previously, the example discussed in this report uses the inverse of the covariance matrix as M, which is ideal (short of utilising Riemmanian-Gaussian kinetic energies). Note that this covariance matrix is also used in the MH implementation in order to guide the proposals. This ensures fairness in the comparison. Once the HMC and MH functions have created their samples, the program removes the first 5000 from each as burn-in. Analysis can then be performed, with plots of both samples being printed, accompanied by acceptance rates and mean/variance Monte Carlo estimations. Beyond that, *arviz* is used to calculate the ESS for both samples and *statsmodels* is used to calculate the autocorrelation.

A.3 ACF Plot for Simple Implementation

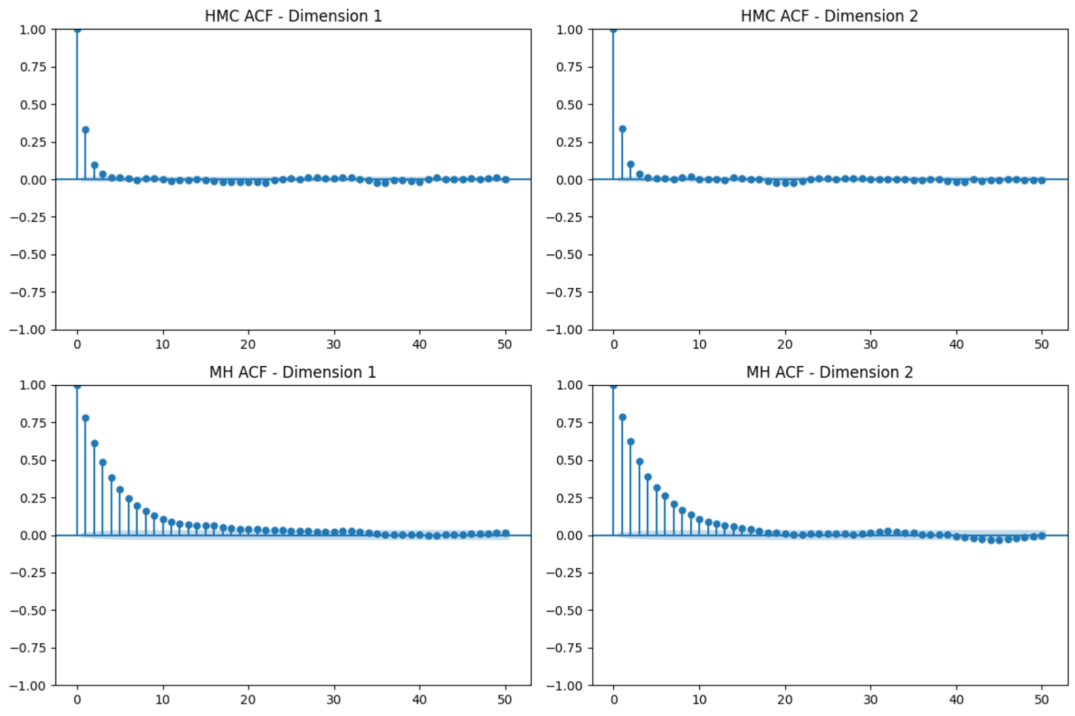

Figure 5: The autocorrelation in the samples produced by HMC decays considerably faster than that produced in the MH sample. This is due to the ability of HMC to make longer jumps to new areas of the typical set, without compromising acceptance rate. MH is forced to take smaller steps in order to retain effective acceptance, and so autocorrelation is greater.

References

Betancourt, Michael (2018). *A Conceptual Introduction to Hamiltonian Monte Carlo*. Accessed: 2025-04-08. arXiv: 1701.02434 [stat.ME]. URL: https://arxiv.org/abs/1701.02434.

Betancourt, Michael J., Simon Byrne, and Mark Girolami (2014). "Optimizing The Integrator Step Size for Hamiltonian Monte Carlo". In: *arXiv preprint arXiv:1411.6669*. URL: https://arxiv.org/abs/1411.6669.

Carrol, Colin D. (2019). *Hamiltonian Monte Carlo from scratch*. Accessed: 2025-04-13. ColinDCarrol.com. URL: https://colindcarroll.com/2019/04/11/hamiltonian-monte-carlo-from-scratch/.

Cook, John D. (2020). *Leapfrog Integrator*. Accessed: 2025-04-12. URL: https://www.johndcook.com/blog/2020/07/13/leapfrog-integrator/.

Duane, S. et al. (1987). "Hybrid Monte Carlo". In: *Physics Letters B* 195.2, pp. 216–222. DOI: 10.1016/0370-2693(87)91197-X. URL: https://www.sciencedirect.com/science/article/pii/037026938791197X.

Etxebarria, Inaki Garcia (2024). *Lagrangian and Hamiltonian Mechanics*. Accessed April 11, 2025. URL: https://www.maths.dur.ac.uk/users/inaki.garcia-etxebarria/MPII/Lectures.pdf.

Hitchcock, David B. (2003). "A History of the Metropolis-Hastings Algorithm". In: *The American Statistician* 57.4, pp. 254–257. ISSN: 00031305. URL: http://www.jstor.org/stable/30037292 (visited on 04/10/2025).

Hoffman, Matthew D. and Andrew Gelman (2014). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: *Journal of Machine Learning Research* 15. Submitted 11/11; Revised 10/13; Published 4/14, pp. 1593–1623.

Jake (2020). *Python: Hamiltonian Monte Carlo from Scratch*. Accessed: 2025-04-13. Towards Data Science. URL: https://towardsdatascience.com/python-hamiltonian-monte-carlo-from-scratch-955dba96a42d.

LIVINGSTONE, SAMUEL et al. (2019). "On the geometric ergodicity of Hamiltonian Monte Carlo". In: *Bernoulli* 25.4A, pp. 3109–3138. ISSN: 13507265, 15739759. URL: https://www.jstor.org/stable/48586025 (visited on 04/10/2025).

Neal, Radford M. (June 9, 2012). *MCMC using Hamiltonian dynamics*. arXiv:1206.1901 [stat.CO]. DOI: 10.48550/arXiv.1206.1901. arXiv: 1206.1901 [stat.CO]. URL: https://arxiv.org/abs/1206.1901.

Roberts, Gareth O., Andrew Gelman, and Walter R. Gilks (Feb. 1997). "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms". In: *The Annals of Applied Probability* 7.1, pp. 110–120. DOI: 10.1214/aoap/1034625254. URL: https://doi.org/10.1214/aoap/1034625254.

Stan Development Team (Mar. 10, 2025). *RStan: The R Interface to Stan*. Accessed: 2025-04-13. Stan Development Team. URL: https://cran.r-project.org/web/packages/rstan/vignettes/rstan.html.