Hamiltonian Monte Carlo Selection

Kelsey Cherland

December 2021

Contents

1 Model Structure

1.1 Justifying the Structure

Recall that, without any adjustments, Hamiltonian Monte Carlo (*HMC*) requires a continuous parameter space because it cannot "glide" through discrete parameter space. However, even with adjustments to allow discrete sampling, *HMC* remains efficient when compared to Metropolis or Gibbs sampling. This is especially true when given multilevel models with thousands of parameters [12] because the Random Walk strategy will have a high rate of failure in high-dimensional spaces [24]. In order to begin the *HMC* process, we need the log-probability of the data and parameters, $log[p(\theta|data)]$. Then we need the gradient with respect to the parameters of the log-probability:

$$\frac{d\ell(\theta|data)}{\partial\theta} = \left[\frac{d\ell(\theta|data)}{\partial\theta_1}, \frac{d\ell(\theta|data)}{\partial\theta_2}, ..., \frac{d\ell(\theta|data)}{\partial\theta_d}\right]$$

Then, we will also need the number of leapfrog steps (L) and a choice of step size (ϵ). When the step size is small, the algorithm can turn sharply and better explore the sample space. In application, the program Stan will choose the leapfrog steps and step size for you by conducting a warm-up phase in which it explores different values for L and ϵ. Stan recognizes this as a tuning simulation. This will be further discussed in the Example section. Let's begin with the theoretical understanding of *HMC*.

The gradient defines the vector field that assigns direction at every point in parameter space, helping to guide us through the sampling of the parameter space while remaining sensitive to the structure of the target distribution. Without any adjustments, the target density, dependent on the choice of parameterization, makes the gradient dependent on the choice of parameterization as well [24]. The mathematics of physics via differential geometry allows use to utilize the information in the gradient while removing dependence on parameterization. We do this by transforming the gradient vector field into a vector field with the introduction of an auxiliary momentum parameter, ϕ. The additional parameter is often defined as $\phi \sim N_d(0, M)$:

$$p(\phi) = det(2\pi M)^{-1/2} exp\left(-\frac{1}{2}(\phi - 0)^T M^{-1}(\phi - 0)\right) \quad (1)$$

The dimension d matches the dimension d of θ and typically the mass matrix is defined as $M = [Var(\theta|data)]^{-1}$. If the ϕ's are independent, M will be a diagonal matrix with components proportional to the posterior $Var(\theta|data)$ [14].

The additional momentum parameter compliments each dimension of our target parameter space and ensures the volume of any neighborhood of the position-momentum phase space remains unchanged. When we update our joint parameters, we update the following joint distribution of our parameters: $p(\theta, \phi|data) = N_d(0, M)p(\theta|data)$. The additional momentum parameter transforms our target distribution in this new phase space into a canonical distribution [24]. This density does not depend on a particular choice of parameterization.

We need the symplectic integrator that updates both θ and ϕ so that we can explore the sample space more efficiently. Integrating along the vector field defined by Hamilton's equation, $H(\theta, \phi)$, often cannot be

done with a numerical integrator because these will cause "drift" away from the optimal path. This cumulative error pushes the trajectory away from the probabilistic set we want to explore. With more variables (larger dimensions) in the phase space, the damage of this drift increases and the most responsible adjustment is to take short Hamiltonian trajectories, invalidating our efficiency.

A symplectic integrator preserves phase space volume for the Hamiltonian trajectories. The numerical trajectories oscillate near the exact energy level for long integration times [24]. By integrating in both directions for the momentum parameter, we ensure optimal performance of the integrator with minimal error [24]. Recall, the probabilistic distribution of the momentum (ϕ) is independent of the position vector θ. Updating using the leapfrog integrator is possible because these distributions can be updated separately. By using the leapfrog integrator we can preserve the volume of our Hamiltonian trajectories and remain more accurate. Divergences occur when the modified level extends outside the boundaries of phase space. Typically, programs like Stan, used in our Example section, will notify the user of divergence issues.

The reversibility property discussed previously says that we also need to ensure that the Hamiltonian transition is reversible, allowing for the ratio of proposal densities to become non-zero. We can do this by flipping the sign of momentum, ϕ [24]. Therefore, a simple implementation of Hamiltonian Monte Carlo involves integrating for a static time, flipping the momentum, and applying a Metropolis-Hasting acceptance probability. Simple application typically uses a first-order leapfrog integrator but research suggests higher-order integrators may be more effective in higher-dimensions [11,25]. Below is an outline of the standard simple *HMC* algorithm using the leapfrog integrator.

1.2 Standard Structure

A simple explanation of the *HMC* [14] follows. The iteration begins with a sample from the posterior distribution (same as its prior) $\phi \sim N_d(0, M)$ and define ϕ_0.

Then, we simultaneously update (θ, ϕ) during the L "leapfrog steps", each scaled by a factor of ϵ. We repeat the following steps L times per iteration:

1. Use gradient of the log posterior to make a half step of ϕ:

   $\phi_{temp} \leftarrow \phi^{(t-1)} + \frac{1}{2}\epsilon \frac{dlogp(\theta^{(t-1)}|y)}{d\theta^{(t-1)}}$

2. Use ϕ_{temp} to update $\theta^{(t-1)}$:

   $\theta^* \leftarrow \theta^{(t-1)} + \epsilon M^{-1}\phi_{temp}$

3. Again, use gradient to half update:

   $\phi^* \leftarrow \phi_{temp} + \frac{1}{2}\epsilon \frac{dlogp(\theta^*|y)}{d\theta^*}$

Except at the first and last steps, the first and third updates can be performed together. If the leapfrog steps move into an area of low density for θ, then the momentum parameter ϕ also slows down, reducing the kinetic energy, $log[p(\phi)]$ [14,24]. Then, we definite the accept-reject step:

1. Accept-reject step $r = \frac{p(\theta^*|y)p(\phi^*)}{p(\theta^{(t-1)}|y)p(\phi^{(t-1)})}$

2. Set $\theta^t = \begin{cases} \theta^* & \text{with probability } min(r, 1) \\ \theta^{(t-1)} & \text{otherwise} \end{cases}$

We do not need to keep track of ϕ after the accept/reject step. We want approximate convergence where $\hat{R}$ is close to 1 (Figure 2-4,6). After L steps, we are done with the iteration. After all iterations, we should have a sample with an acceptance rate approximately between 65 and 85 percent.

We can tune the *HMC* be adjusting the diagonal elements of M, the scaling factor ϵ, and the number of leapfrog steps L per iteration. We typically set M to a crude estimate of the scale of the target distribution (see Example below). We set the product $\epsilon L = 1$ because it will roughly take the leapfrog steps from one side of the distribution to the other (often $\epsilon = 0.1$ and L=10 are used) [14].

If the acceptance rate is lower, then we should lower ϵ and correspondingly increase L so that their product remains 1. If the average acceptance is far greater than 65 percent, then a good place to start is to try increasing ϵ and decreasing L. Although, much of the literature suggests that a higher acceptance rate is not uncommon (we've seen 80 percent listed as a norm). Another strategy is to vary ϵ and L from iteration to iteration while the simulation is running. There might be some loss of optimality, taking short steps where long steps would be feasible and vice versa. Similarly, we can vary the mass matrix M to scale to the local curvature of the log density as the algorithm moves through the posterior density [14]. Notice that the challenges for *HMC* are different from the Metropolis Hastings and Gibbs samplers, which might require reducing auto-correlation and considering different proposals.

Locally adaptive *HMC* structures have been developed to address more challenging sampling scenarios. The no-U-turn sampler, for example, determines the number of steps adaptively at each iteration. When it reaches a negative value of the dot product between ϕ and the distance traveled at the start of the iteration, it will turn around. This sampler applied alone will not converge to the desired target distribution and requires serious adaptive changes to the mass matrix and step size [14]. This section is a basic overview of the *HMC* and its tuning parameters. Next is an example.

2 Example

2.1 Eight Schools Using R

We illustrate the Hamiltonian Monte Carlo algorithm with the eight schools example using a hierarchical normal model. The eight schools data and background information can be found in Chapter 5 of the *Bayesian Data Analysis* textbook [14]. In this example, eight high schools each use a different program to increase student SAT scores. A study is conducted to analyze the effects of these programs, and the estimated treatment effect and standard error of effect estimate is recorded for each school. The estimated treatment effect and its standard error were obtained by an analysis of covariance adjustment. The estimates are obtained by independent experiments and have approximately normal sampling distributions with sampling variances that are known.

We use the code from Appendix C in the *Bayesian Data Analysis* textbook to demonstrate *HMC* [14]. Please see Appendix I at the end of this report for the code that we will now discuss. In the algorithm code, we use the "rstan" and "arm" libraries. We import or create the eight schools dataset. Next, we create a function for the log posterior density, which takes in three arguments. The first argument is "th," a vector containing all 10 parameters for our model. This includes the eight theta values, μ, and τ. The second argument is the estimated treatment effect and the third argument is the standard error, both of which come from our data.

Because τ is restricted to be positive, we set the posterior density to be zero when τ is not a number or when $\tau \leq 0$. Next, we define our priors for the final posterior density. We have a uniform hyperprior, so the log of the hyperprior is 1. The prior is normal because it yields a proper distribution. The likelihood function is normally distributed, and the likelihood equation is:

$$L(\theta|y) = \prod_{j=1}^{J} N(y_j|\theta, \sigma) \tag{2}$$

where y_j is the data and σ is the standard deviation from the dataset. Our log posterior density function returns the sum of the log of the hyperprior, the log of the prior, and the log likelihood.

Next, we write a function to find the gradient of the log posterior density. This function has the same arguments as our previous function. τ has the same restriction, so we let the gradient equal 0 when $\tau \leq 0$. We define the gradients with respect to their parameters, and we return each gradient in the output. Theoretically, the gradient of the log posterior density for a school effect, θ_j, would be:

$$\frac{dlogp(\theta|data)}{d\theta_j} = -\frac{\theta_j - data_j}{\sigma_j^2} - \frac{\theta_j - \mu}{\tau^2} \tag{3}$$

The gradient with respect to μ is:

$$\frac{dlogp(\theta|data)}{d\mu} = -\sum_{j=1}^{J} \frac{\mu - \theta_j}{\tau^2} \tag{4}$$

The gradient with respect to τ is:

$$\frac{dlogp(\theta|data)}{d\tau} = -\frac{J}{\tau} + \sum_{j=1}^{J} \frac{(\mu - \theta_j)^2}{\tau^3} \tag{5}$$

In practice, we use the analytic gradient. The textbook also shares a numerical gradient function for purposes of debugging. If both gradients return results that are exact to several decimal places, then we know that there is likely not a problem with our gradients. This alternative function for the gradient uses small finite differences to find the gradient numerically. It approximates the gradient with first differences. We define a variable called "e" to equal 0.0001, which acts as a small positive amount for stability. This function uses our log posterior density function to find and return the gradient.

The function for a single *HMC* iteration takes in three arguments, three of which are new. The new arguments include ϵ, L, and M. ϵ is the step size. L is the number of leapfrog steps per iteration. M is the diagonal mass matrix. We begin by initializing ϕ by drawing 10 values at random from the normal distribution centered at 0 with a standard deviation of the square root of M, the mass matrix. Next, we find the log posterior density of θ_{old} and ϕ_{old} and call it log_p_old to use later in the ratio for the accept/reject step. Then we complete a half step of ϕ, which is Step 2a of our algorithm. We enter the leapfrog steps and complete a full step of θ, which is Step 2b. We complete a full step of ϕ, which is Step 2c. Finally, we find the log posterior density of θ^* and ϕ^* and call it log_p_star. We use log_p_star and log_p_old to find the ratio of the log posterior densities. We define the probability of jumping to a new value as the ratio of the log posterior densities or 1, whichever is smaller. We define the new θ value. The function returns the acceptance probability and the new θ value as a list.

The function for our *HMC* algorithm with multiple iterations takes five arguments. The first is starting_values, which is an $(m \times d)$ matrix. The number of chains is represented by m. The textbook recommends to set the number of chains to 4. The number of parameters in the vector is represented by d, which is 10 in our example. The number of iterations is represented by iter. ϵ_0 is the baseline step size. L_0 is the number of steps. M is the diagonal mass matrix. We set half of our iterations to be part of the warm-up period. The function tracks and prints the parameter simulations and the average acceptance rate.

Now that we have finished writing the functions, we define a vector to make reading the results easier. We define parameter_names to include the names of the parameters in our output. We define the diagonal mass matrix to be proportional to the inverse variance matrix of the posterior distribution. The textbook recommends 15 for this example. Since 15 is the standard deviation, we square 15 to get the variance. We take the inverse to get the inverse variance, and we use this to construct our diagonal mass matrix. Then we set our initial values in an array called "starts". We assign all 10 starting points to be sampled at random from the normal distribution centered at 0 with a standard deviation of 15. Because the starting point for τ must be positive, we sample τ from the uniform distribution from 0 to 15.

In our first run, we run for 20 iterations and set $\epsilon_0 = 0.1$ and $L_0 = 10$ (See Figure 2). The program runs successfully and does not crash, so we increase the iterations to 100. From looking at the results of our second run, the acceptance rates seem low, so we tune ϵ_0 and L_0 to improve the acceptance rate (See Figure 3). We decrease ϵ_0, our step size, to 0.05 and we increase L_0, our base number of steps to 20. In our third run, our average acceptance rates have improved (See Figure 4). All that is left is to run the algorithm for thousands of iterations until we obtain stable inferences.

```
Inference for the input samples (4 chains: each with iter = 20; warmup = 10):

           Q5  Q50  Q95 Mean   SD  Rhat Bulk_ESS Tail_ESS
theta[1] -13.0 12.3 24.8 10.4 12.3  1.61       20       20
theta[2]  -6.1  4.9 23.7  4.7  8.3  2.09       20       40
theta[3]  -7.0 -0.8 15.1  0.5  7.0  1.16       20       20
theta[4]  -3.8  5.1 19.2  6.1  8.2  1.63       20       20
theta[5]  -5.3  0.7  5.5  0.5  3.7  1.72       20       20
theta[6] -14.0  1.0  7.8  1.0  7.2  1.86       20       20
theta[7]   2.1  9.4 23.1 10.6  7.0  1.40       20       20
theta[8]  -7.4  5.0 23.4  5.1  8.7  1.73       20       20
mu        -2.4  3.7 10.1  4.2  4.5  1.24       20       20
tau        4.5  9.4 21.2 11.4  5.7  1.28       20       20

For each parameter, Bulk_ESS and Tail_ESS are crude measures of
effective sample size for bulk and tail quantities respectively (an ESS > 100
per chain is considered good), and Rhat is the potential scale reduction
factor on rank normalized split chains (at convergence, Rhat <= 1.05).
Avg acceptance probs: 0.70 0.70 0.52 0.57
```

Figure 1: Results of the first run. We run the algorithm with 20 iterations.

```
Inference for the input samples (4 chains: each with iter = 100; warmup = 50):

            Q5 Q50  Q95 Mean   SD  Rhat Bulk_ESS Tail_ESS
theta[1]   1.3 9.0 29.6 11.5  8.6  1.19       53       65
theta[2]   0.4 7.6 17.4  8.5  5.3  1.12       33       41
theta[3] -17.3 5.7 21.8  3.9 10.4  1.71       42       78
theta[4]  -4.6 6.4 21.5  7.8  7.1  1.67       30       64
theta[5]  -3.4 6.7 12.1  5.0  5.1  1.61       56       71
theta[6]  -5.5 8.0 16.8  6.5  6.2  1.44       32       36
theta[7]  -2.2 8.6 20.2  9.5  6.4  1.19       25       29
theta[8]  -2.5 7.3 28.3  9.5  9.0  1.57       28       43
mu        -0.5 7.3 15.3  7.9  4.7  1.67       42       38
tau        1.3 6.0 19.8  7.5  6.3  1.83        7        6

For each parameter, Bulk_ESS and Tail_ESS are crude measures of
effective sample size for bulk and tail quantities respectively (an ESS > 100
per chain is considered good), and Rhat is the potential scale reduction
factor on rank normalized split chains (at convergence, Rhat <= 1.05).
Avg acceptance probs: 0.59 0.53 0.06 0.53
```

Figure 2: Results of the second run. We increase the iterations to 100. The acceptance rates seem low, so we tune ϵ_0 and L_0.

```
Inference for the input samples (4 chains: each with iter = 100; warmup = 50):

          Q5  Q50  Q95 Mean  SD  Rhat Bulk_ESS Tail_ESS
theta[1]  1.3 12.3 27.7 13.2 9.0  1.13       24       80
theta[2] -1.4  6.8 20.8  7.6 6.8  1.16      106      102
theta[3] -7.5  4.2 19.9  5.3 8.3  1.02       84       58
theta[4] -4.5  7.3 21.0  7.7 7.2  1.09      114       40
theta[5] -4.6  4.4 14.4  4.3 5.9  1.05      103      149
theta[6] -5.7  4.9 15.9  5.6 6.8  1.07      109      155
theta[7]  1.6 10.1 26.0 12.0 7.0  1.06      104       85
theta[8] -4.8  8.2 21.5  8.8 8.1  1.22      101      116
mu        0.6  8.0 16.4  8.0 4.5  1.09       75       82
tau       2.2  7.1 16.3  7.9 4.4  1.18       17       24

For each parameter, Bulk_ESS and Tail_ESS are crude measures of
effective sample size for bulk and tail quantities respectively (an ESS > 100
per chain is considered good), and Rhat is the potential scale reduction
factor on rank normalized split chains (at convergence, Rhat <= 1.05).
Avg acceptance probs: 0.72 0.65 0.70 0.86
```

Figure 3: Results of the third run. Our average acceptance rates have improved.

This coding example illustrates how hard-coding the algorithm and writing out the functions can be costly in programming effort. In addition, the user must tune the step size and the number of steps. Stan can do this work more efficiently, saving the user time and energy.

2.2 Eight Schools Using Stan via R

Stan, which is an acronym for "Sampling through adaptive neighborhoods," was created to automatically apply *HMC* given a Bayesian model. The algorithms in Stan require data and model inputs, computation of the log posterior density and its gradients, a warm-up phase for the tuning parameters, an implementation of the no-U-turn sampler, and convergence monitoring with inferential summaries at the end. The Stan code translates a model to C++ code, compiles the C++ code to a dynamic shared object (DSO) and loads the DSO. Stan then conducts the sampling, given the user-specified data and other settings. Stan uses a stochastic process so the results will vary. We applied the Stan code so that we can compare the process of coding each step by hand in the previous section to what a robust system like Stan might be capable to doing.

The reader can follow along for the Stan portion in Appendix 2. First, we need the "rstan" library and we need to set the appropriate options (see the Stan example). Then we need to import or create our data as list of 17 items: J=8, the values for the eight schools, and their variances. It's in this order because that is the order of the data portion of the Stan file created for our *HMC*. Then we define our parameters and transform our parameters so that they can be put into our model where the prior is a normal and the target is a normal. The exact syntax is shown on page 592 in *Bayesian Data Analysis* by Gelman et al, but is also shown below in Figure 5.

```
// saved as schools.stan
data {
  int<lower=0> J;         // number of schools
  real y[J];              // estimated treatment effects
  real<lower=0> sigma[J]; // standard error of effect estimates
}

parameters {
  real mu;                // population treatment effect
  real<lower=0> tau;      // standard deviation in treatment effects
  vector[J] eta;          // unscaled deviation from mu by school
}

transformed parameters {
  vector[J] theta;
  theta = mu + tau * eta;        // school treatment effects
}

model {
  target += normal_lpdf(eta | 0, 1);       // prior log-density
  target += normal_lpdf(y | theta, sigma); // log-likelihood
}
```

Figure 4: Stan code from the schools.stan file.

When we use the stan() function, we must refer to the separate Stan file in the argument 'file' and the listed data in the argument 'data'. Then we define the number of Markov chains ('chains'), the number of warm-up iterations ('warmup'; typically half the number of total iterations), the total iterations ('iter'), the number of cores ('cores'), and 'refresh'. The 'refresh' argument shows the progress for every 'refresh' iterations. Therefore, if we set 'refresh' to 1000 and we have 2000 iterations, we should have two progress updates. We assign the run of the stan() function to value "fit1".

After we run the *HMC* via the stan() function and store it in object "fit1", we can print the results using the print() function. Within print(), we define the printout values and credible probabilities (0.1, 0.5, 0.9) for θ, μ, τ, and the log posterior. Our printout should show the 4 chains, each with 2000 iterations (see Figure 6).

```
> print(fit1, pars=c("theta", "mu", "tau", "lp__"), probs=c(.1,.5,.9))
Inference for Stan model: schools.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

          mean se_mean   sd    10%    50%    90% n_eff Rhat
theta[1]  11.46    0.19 8.30   2.23  10.24  22.23  1862    1
theta[2]   8.02    0.10 6.33  -0.06   8.04  15.86  4430    1
theta[3]   6.01    0.15 8.01  -3.63   6.64  14.90  2688    1
theta[4]   7.87    0.10 6.51  -0.14   7.91  15.85  4099    1
theta[5]   5.29    0.10 6.37  -3.05   5.78  12.97  3828    1
theta[6]   6.27    0.11 6.82  -2.23   6.65  14.24  3990    1
theta[7]  10.74    0.11 6.84   2.83  10.04  19.71  3693    1
theta[8]   8.85    0.14 8.01   0.07   8.58  18.57  3114    1
mu         8.24    0.16 5.32   1.86   8.09  14.89  1152    1
tau        6.61    0.19 5.76   0.98   5.28  13.55   924    1
lp__     -39.57    0.07 2.69 -43.19 -39.34 -36.30  1306    1

Samples were drawn using NUTS(diag_e) at Thu Dec 16 14:37:42 2021.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
>
```

Figure 5: Results of fit1.

We can also observe the summary of our iterations by using summary() on our sampled parameters, see Figure 7. Here we can observe that our average acceptance rate is about 80%, the average $\epsilon = 0.35$, and the average number of leapfrog steps is about 13 per iteration. The figure also shows that our sampling rarely yielded divergent results (≤ 1).

```
> summary(do.call(rbind, sampler_params), digits = 2)
 accept_stat__    stepsize__     treedepth__   n_leapfrog__  divergent__       energy__
 Min.   :0.00   Min.   :0.028   Min.   :0.0   Min.   :  1   Min.   :0.0000   Min.   :35
 1st Qu.:0.73   1st Qu.:0.314   1st Qu.:3.0   1st Qu.:  7   1st Qu.:0.0000   1st Qu.:42
 Median :0.94   Median :0.347   Median :3.0   Median : 15   Median :0.0000   Median :44
 Mean   :0.81   Mean   :0.392   Mean   :3.4   Mean   : 13   Mean   :0.0095   Mean   :45
 3rd Qu.:0.99   3rd Qu.:0.402   3rd Qu.:4.0   3rd Qu.: 15   3rd Qu.:0.0000   3rd Qu.:47
 Max.   :1.00   Max.   :8.368   Max.   :7.0   Max.   :159   Max.   :1.0000   Max.   :61
```

Figure 6: Summary of our iterations.

We can also use the traceplot() function for variables like μ, τ to identify the stability of our sampling. The gray area to the left is the warm-up phase and the white area to the right is the real sampling. The 4 different chains are shown in different colors. We can see that there is good mixing (see Figure 8).

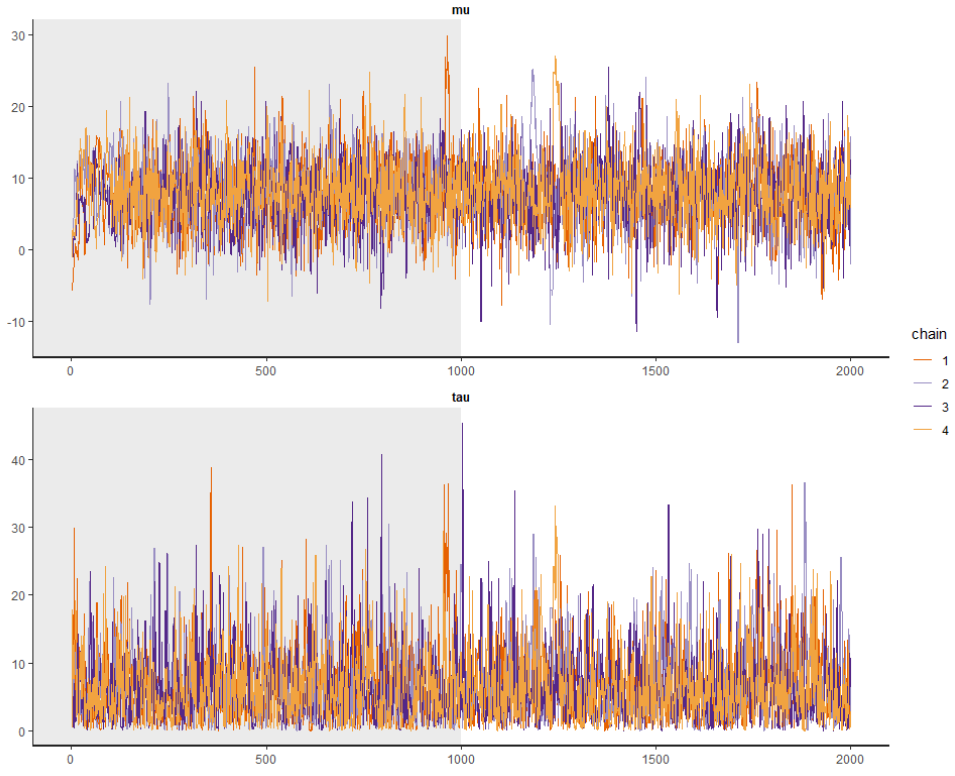

Figure 7: Trace plots for μ and τ.

Using plot() on "fit1", we can observe the confidence intervals in an image (See Figure 9). The confidence intervals show minor variations among the "eta" variables, which are the un-scaled deviations from μ by θ_j. Centered near zero, each "eta" tells us how different the school effect was from the estimated μ. If we just look at the first three "eta"s, or the first three differences between θ and μ, the first one is consistently (with many runs of an *HMC*) larger than μ and the second and third are comparatively smaller than the first difference and sometimes less than μ. The τ is centered between 5 and 7 with a large range that's skewed right, and the μ is centered between 6 and 8 with an approximately symmetrical distribution and large variance. The τ

likely has a center that is large because we used a uniform hyperprior. Using an Inverse Gamma hyperprior would yield a much smaller center for τ.

Recall, that θ_j measures the mean effect from coaching for each of the eight schools (J=8, where y_j is the data and σ_j is the sample standard deviation), and the θ's come from a population that is normally distributed with μ as the center and τ^2 as the variance ($\theta_j \sim N(\mu, \tau^2)$). We found that the estimated difference between each θ and μ is small, but it does include a range of both positive and negative values, which means that for each mean effect per school, a confidence interval includes estimates both above and below the population mean for all schools. The population mean for all schools has a mean equal to about 8, based on Figure 6, with an 80% CI that includes only positive values and a 95% CI that includes 0 and some negative values, as can be seen in Figure 10. An increase of eight points on the SAT corresponds to about one more test question correct [14]. It would seem, based on these results, that a coaching program will have an average result that, at best, will improve scores by 1-2 questions or 8-17 points. And at worst, the coaching program will make no difference at all in SAT scores.

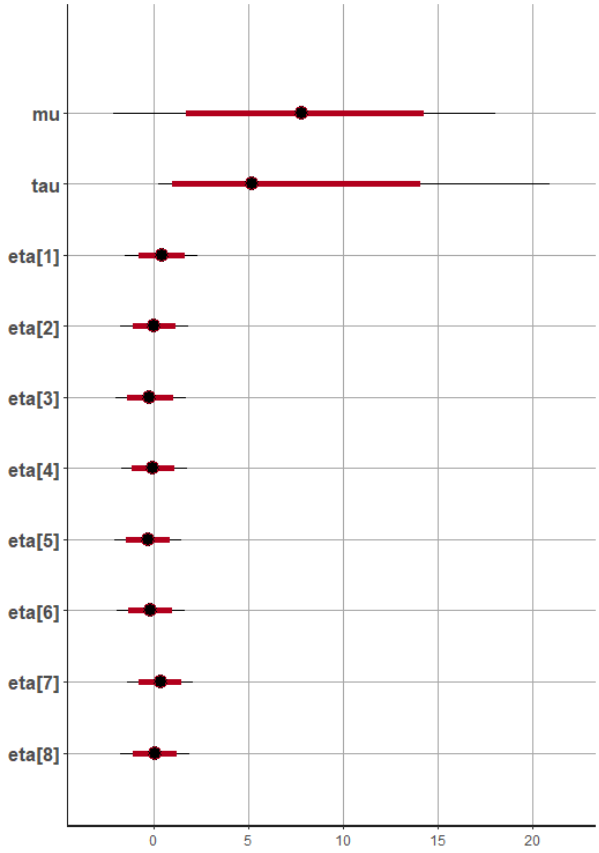

Figure 8: Confidence intervals for the 10 parameters. The red lines represent the 80% confidence intervals. The black lines represent the 95% confidence intervals.

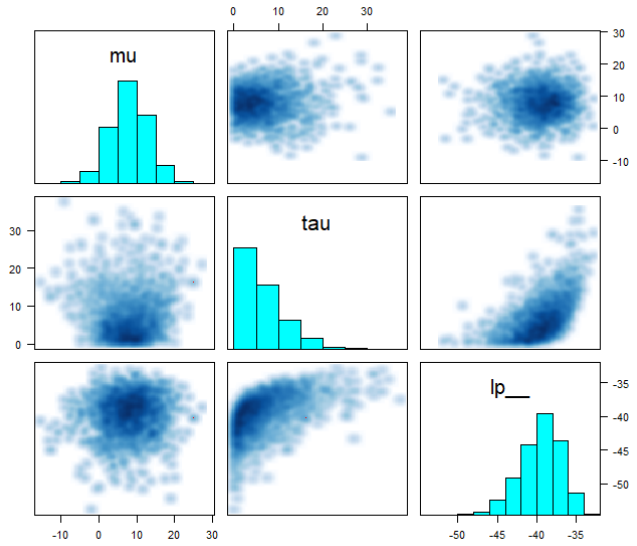

Figure 9: Distributions for μ, τ, and the log of the un-normalized joint posterior density.

3 Conclusion

This paper has introduced the relatively new (1987) *Hamiltonian Monte Carlo* sampling approach, which produces distant proposals for the Metropolis algorithm. The applications of *HMC* demonstrate robust sampling in high-dimensions and efficient proposal states compare to other methods such as Metropolis. This paper introduced the advantages, disadvantages, properties, applications, motivation, and structure with a complimentary example. The *HMC* algorithm is very efficient at sampling the posterior parameter space and produces samples with low auto-correlation, resulting in a high efficiency rate (80-90% acceptance rate) due to length of its steps. Since the distances between successive generated points are typically large, we require fewer iterations to get representative sampling [20].

We found that *HMC* has some fundamental limitations since it is designed to work with continuous target densities, and it needs some modifications by utilizing other methods such as Gibbs in case of using it with discrete cases. However, computer programs like Stan can smooth out the complexity of a basic HMC. Potential areas for future research include developing more efficient *HMC*s that can handle large data sets and discrete parameters without any modifications. The future development of HMC will likely be inspired by its application in scientific research fields.

4 References

[1] Neal, Radford M. "MCMC using Hamiltonian dynamics," in Chapter 5, *Handbook of Markov Chain Monte Carlo*, eds. Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng (Chapman and Hall/CRC: 2011). http://dx.doi.org/10.1201/b10905.

[2] Shinji, Nakaoka in "Integrated Population Biology and Modeling, Part A", 1st Edition, eds. Arni Srinivasa Rao and C.R. Rao in *Handbook of Statistics*, 2018.

[3] Murthy, Katabathula Ramachandra, Salendra Singh, David Tuck, Vinay Varadan. "Integrated Population Biology and Modeling, Part B.", in *Handbook of Statistics*, 2019.

[4] Jalaian, Brian and Stephen Russell, "Uncertainty Quantification in Internet of Battlefield Things" in *Artificial Intelligence for the Internet of Everything*, 2019.

[5] Seiler, Chistof. "Bayesian Statistics in Computational Anatomy" in *Statistical Shape and Deformation Analysis: Methods, Implementations & Applications*, eds. G. Zheng, S. Li, and G. Szekely., 2017.

[6] Radcliffe, Andrew J. and Gintaras V. Reklaitis. "31st European Symposium on Computer Aided Process Engineering" in *Computer Aided Chemical Engineering* Vol 50, 2021.

[7] Zhang, Xin, Muhammad Atif Nawaz, Xuebin Zhao, and Andrew Curtis,"Chapter Two - An introduction to variational inference in geophysical inverse problems" in *Advances in Geophysics*, Vol. 62, 2021, Pages 73-140.

[8] Fletcher, Tom, Xavier Pennec, and Stefan Sommer. "Statistics on manifolds", in *Riemannian Geometric Statistics in Medical Image Analysis*, 2020.

[9] Zhang, Miaomiao."Low-dimensional shape analysis in the space of diffeomorphisms" in *Riemannian Geometric Statistics in Medical Image Analysis*, eds. Xavier Pennec, Stefan Sommer, and Tom Fletcher, 2020.

[10] Fletcher, Tom, Xavier Pennec, and Stefan Sommer. "Statistics on manifolds and shape spaces" in *Riemannian Geometric Statistics in Medical Image Analysis* 2020.

[11] Fernández-Pendás, M., Akhmatskaya, E. and J.M. Sanz-Serna. 2016. "Adaptive multi-stage integrators for optimal energy conservation in molecular simulations". https://doi.org/10.1016/j.jcp.2016.09.035.

[12] McElreath, R. 2020. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*, Second Edition. (Chapman and Hall/CRC: Taylor & Francis Group, LLC). ISBN 9780429029608.

[13] Xing, Eric. "Markov Chain Monte Carlo", Probabilistic Graphical Models (course notes), Spring 2015. https://www.cs.cmu.edu/~epxing/Class/10708-15/notes/10708_scribe_lecture17.pdf

[14] Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., and D.B. Rubin. 2013. *Bayesian Data Analysis*, Third Edition. CRC Press (Taylor & Francis Group, LLC). ISBN 9781439840955.

[15] van Biezen, Michel (Director of Ilectureonline). Physics - Adv. Mechanics: Hamiltonian Mech. (1 of 18) "What Is Hamiltonian Mechanics?" YouTube, 14 Apr. 2016, https://www.youtube.com/watch?v=R_N_2Q4v9jA&ab_channel=MichelvanBiezen. Accessed 17 Dec. 2021.

[16] Lambert, Ben (Director of SpartacanUsuals). "The Intuition behind the Hamiltonian Monte Carlo Algorithm." YouTube, 15 May 2018, https://www.youtube.com/watch?v=a-wydhEuAm0&t=672s&ab_channel=BenLambert. Accessed 17 Dec. 2021.

[17] Betancourt, Michael. "Michael Betancourt: Scalable Bayesian Inference with Hamiltonian Monte Carlo." Youtube, London Machine Learning Meetup, 18 July 2018, https://www.youtube.com/watch?v=jUSZboSq1zg&ab_channel=LondonMachineLearningMeetup.

[18] Hirvonen, Ville. "Hamiltonian Mechanics For Dummies: An Intuitive Introduction." Profound Physics (blog), 2021, https://profoundphysics.com/hamiltonian-mechanics-for-dummies/#:~:text=Hamiltonian%20mechanics%20is%20a%20formulation%20of%20classical%20mechanics%20that%20is,the%20energy%20of%20a%20system.

[19] Young, Hugh D., Roger A. Freedman, A. Lewis Ford, and Francis Weston Sears. *Sears and Zemansky's University Physics with Modern Physics (Addison-Wesley Series in Physics)*, 11th ed. (San Francisco: Pearson Addison Wesley 2004).

[20] Rogozhnikov, Alex. "Hamiltonian Monte Carlo explained", Brilliantly wrong thoughts on science and programming (blog), Dec 19, 2016, https://arogozhnikov.github.io/2016/12/19/markov_chain_

monte_carlo.html.

[21] Kramer, A., Calderhead, B. Radde, N. "Hamiltonian Monte Carlo methods for efficient parameter estimation in steady state dynamical systems.", *BMC Bioinformatics* 15, 253 (2014). https://doi.org/10.1186/1471-2105-15-253

[22] Chen, Yen-Chi. "Lecture 9: Hamiltonian Monte Carlo", STAT 535: Statistical Machine Learning (course notes), University of Washington Faculty Web Server, 2019. http://faculty.washington.edu/yenchic/19A_stat535/Lec9_HMC.pdf.

[23] Cobb, Adam D. and Brian Jalaian. 2021. "Scaling Hamiltonian Monte Carlo Inference for Bayesian Neural Networks with Symmetric Splitting." Conference paper accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021). https://auai.org/uai2021/pdf/uai2021.274.preliminary.pdf.

[24] Betancourt, Michael. "A Conceptual Introduction to Hamiltonian Monte Carlo" via arxiv (2017). https://arxiv.org/pdf/1701.02434.pdf.

[25] Blanes, S., Casas, F. and J.M. Sanz-Serna. 2014. "Numerical integrators for the Hybrid Monte Carlo method". *SIAM Journal on Scientific Computing* Vol 36 No. 4 (2014). https://arxiv.org/abs/1405.3153.