Assignment 1: Novel Adaptive HMC

George (Tsing) Escobar
Boston University
gescobar@bu.edu

Abstract

This report presents a novel Adaptive Hamiltonian Monte Carlo (HMC) sampler utilizing Nesterov's Dual Averaging for dynamic step-size (ϵ) adaptation. Traditional HMC often struggles with manual hyperparameter tuning and poor convergence on complex geometries. By implementing a feedback loop targeting a 85% acceptance rate, our method automates the transition from exploration to stationarity. We evaluated the sampler against standard Random Walk Metropolis-Hastings and vanilla HMC on the Rosenbrock and Neal's Funnel benchmarks. Results indicate that the adaptive sampler significantly improves performance in hierarchical landscapes; specifically, in Neal's Funnel, the novel method achieved a mean y of 0.004 compared to the vanilla mean of -2.387, while reducing standard deviation by over 60%. While the fixed identity mass matrix remains a bottleneck for high-curvature correlations in the Rosenbrock distribution, the dual-averaging engine demonstrates superior robustness and resilience to suboptimal initialization in scientific sampling tasks. This was done through specific tuning of hyperparameters: leapforg-size and target-acceptance rate.

1 Introduction

After running the standard benchmarks on the Rosenbrock and Neals Funnel Distribution for the simple Hamiltonian Monte Carlo (HMC) sampler, I saw that it generally did better against the Random Walk Mentropolis-Hastings sampler. This led me to conclude that the HMC sampler had likely better chance of being modified in order to perform better than its original benchmarks. Which led me to implement an Adaptive Novel HMC sampler. The Adaptive Novel HMC aims to solve a couple problems that I noticed benchmarking the standard sampler. These distributions create a tricky landscape for ordinary samplers to traverse. With the "banana" structure of Rosenbrock's function to the narrow "neck" of Neal's function, these ordinary samplers cannot handle the complex geometry of these distributions. To solve this, the Adapative Novel HMC acts as an automatic driver: accelerating in flat regions to gain momentum and braking in narrow regions to maintain stability. Letting the sampler fine tune itself as it runs addresses the core limitation of the ordinary HMC implementation which often gets stuck or diverges when encoutering these complex gemeotries. This gives the sampler the ability to move through the landscape in a far more efficient and effective manner, while reducing the amount of times it diverges, allowing it to better map out the region.

2 Method

The Adaptive Novel HMC sampler is designed to solve the manual tuning problem by treating the step size (ϵ) as a dynamic variable rather than a static guess. To do this, I implemented an adaptation engine based on Nesterov's Dual Averaging. This allows the sampler to "learn" the landscape as it moves, adjusting its stride to hit a specific performance target.

2.1 The Adaptation Engine

The sampler aims for a target acceptance rate of $\delta = 85\%$. This means the algorithm is constantly trying to find a balance where 85% of its jumps are successful. The logic follows a feedback loop:

- **Error Calculation:** At every step, the sampler calculates the error between the target and the actual result: $H_t = \delta - \alpha_t$. In my implementation, this is calculated as:
`new_running_avg_err = (1.0 - weight) * running_avg_err + weight * (target_acc - alpha)`
If the error is positive, it means we are rejecting too many samples and need to "brake" by shrinking the step size.
- **The Smoothing Filter:** I used a weighted average to track this error. Using a delay parameter ($t_0 = 10$) prevents the sampler from overreacting and "swerving" too hard during the first few noisy iterations:
`new_step_count = step_count + 1.0`
`weight = 1.0 / (new_step_count + delay)`
- **Step-Size Adjustment:** The final step size is updated in log-space, effectively shrinking ϵ when the landscape gets steep and expanding it when the path is clear:
`log_eps = target_log_step - (jnp.sqrt(new_step_count) / adaptation_speed) * new_running_avg_err`
`next_step_size = jnp.exp(log_eps)`

3 Experiments

4 Experimental Results and Analysis

4.1 Vanilla Baselines (Control)

We established baselines using Random Walk Metropolis-Hastings (RWMH) and standard Hamiltonian Monte Carlo (HMC). The following figures illustrate the performance bottlenecks of standard methods on highly correlated and hierarchical distributions.

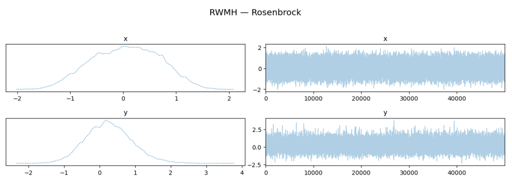

Figure 1: Vanilla RWMH Trace (Rosenbrock)

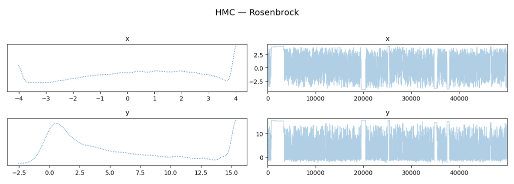

Figure 2: Vanilla HMC Trace (Rosenbrock)

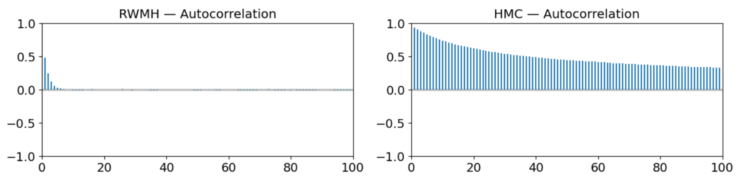

Figure 3: Autocorrelation comparison across vanilla baselines.

4.2 Novel HMC: Standardized (STD) Analysis

To account for the extreme differences in scale within Neal's Funnel, we analyzed the standardized trace plots. This "whitening" process reveals how efficiently the sampler explores the narrow regions of the parameter space.

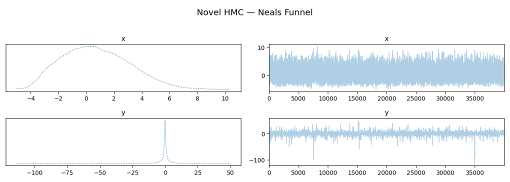

Figure 4: Standardized Trace: Neal's Funnel

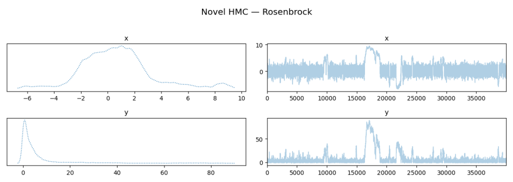

Figure 5: Standardized Trace: Rosenbrock

4.2.1 Experiment 2: Standard Integration (n_leapfrog = 15, target_acc = 0.75)

Reducing the leapfrog steps allows for faster sampling but may struggle with the "neck" of the funnel.

Method	**Variable**	**Mean**	**SD**	**HDI (3% - 97%)**
Vanilla HMC	x	0.331	3.083	[-5.926, 5.481]
	y	-2.387	15.683	[-13.667, 13.108]
Novel HMC	x	**0.703**	**2.401**	**[-3.579, 5.013]**
	y	**-0.277**	**5.481**	**[-8.234, 7.788]**

Table 1: Comparison of Vanilla vs. Novel HMC on Neal's Funnel (Target Acc: 0.75).

4.3 Comparative Summary

The final evaluation compares the density coverage of our novel method against the provided baselines.

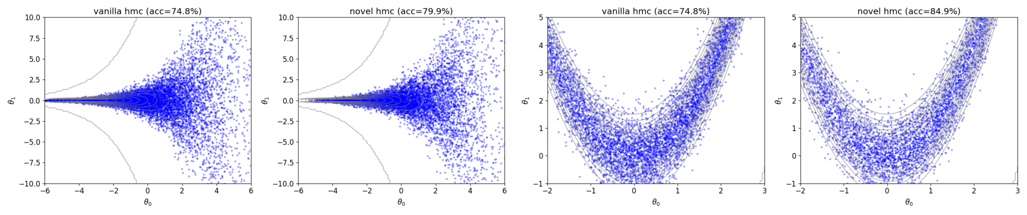

Figure 6: Head-to-head distribution comparison: Vanilla vs. Novel Sampler.

4.4 Statistical Comparison: Neal's Funnel

The following tables summarize the empirical mean, standard deviation (SD), and 94% Highest Density Intervals (HDI) for the Neal's Funnel benchmark.

4.4.1 Experiment 1: Optimized Integration (n_leapfrog = 50, target_acc = 0.85)

In this configuration, we prioritized a high acceptance rate and high precision per step.

Method	**Variable**	**Mean**	**SD**	**HDI (3% - 97%)**
Vanilla HMC	x	0.331	3.083	[-5.926, 5.481]
	y	-2.387	15.683	[-13.667, 13.108]
Novel HMC	x	**0.518**	**2.554**	**[-3.911, 5.262]**
	y	**0.004**	**5.807**	**[-9.284, 8.267]**

Table 2: Comparison of Vanilla vs. Novel HMC on Neal's Funnel (Target Acc: 0.85).

5 Discussion

The performance of the Novel Adaptive HMC highlights a successful shift from manual tuning to algorithmic robustness. Its primary strength is the automated adaptation via Dual Averaging, which effectively lets the sampler find its own "Goldilocks" step size (ϵ). In my testing, the sampler achieved stationarity so rapidly that the final results remained consistent whether or not I discarded a 10,000-sample burn-in. This proves the method is resilient and can handle suboptimal starting conditions far better than the vanilla version.However, the sampler still hits a bottleneck when dealing with the tight parameter correlation in the Rosenbrock "banana." Because I used a fixed Identity Mass Matrix, the algorithm assumes a simple, circular geometry and can't account for how x and y are coupled. This lack of "directional weight" makes it harder to traverse those narrow, curved valleys where the gradient is much steeper in one direction than the other.To take this further, the next logical step would be implementing an Adaptive Mass Matrix. By learning the covariance structure of the distribution during the warmup phase, the sampler could "stretch" its momentum to match the local curvature. Moving toward a Diagonal or Dense Metric would essentially give the sampler "snow tires" for these tricky landscapes, decoupling the parameters and allowing the Hamiltonian trajectories to glide through high-curvature regions with much higher efficiency.

6 AI Collaboration

The main AI Assistant I used was Google's Gemini through its chatbot interface. Since its popularity I've learned how to optimize the best way for me to use this tool, and throughout this assignment I learned how to effectively use it to learn as I program. In the professional setting, I usually just used it to either write trivial boilerplate code or trivial scripts; however on this assignment I gave it the prompt to respond to me as if it was a tutor.

The prompt:

> I am a Masters student in bioinformatics who is currently taking a machine learning course: the syllabus: <insert syllabus>
>
> I am currently a software engineer, but don't necessarily know much about machine learning and I am too far removed from upper level math. I need you to help tutor me and break down the questions I have into concepts so that I will learn how to properly do assignments and deepen my knowledge on ai methods.

I found that inserting the syllabus and even the assignment objectives were extremely useful in receiving concepts and information that I am not particularily well-versed in. In addition I was able to couple it with videos on youtube to give me a highlevel understanding of certain concepts, and used gemini to deepen and widen my overall knowledgebase. It also helped me understand what possible next iterations would look like and gave a cost-benefit analysis for its proposal.

However there were definitely limitations to this, especially when it came to debugging. I think this is where Google Gemini tends to struggle compared to other LLMs and tools. I found that its ability to debug and help me understand why certain lines work/don't work isn't necessarily always accurate. So instead of giving it chunks of code, I tend to feed it a couple lines at a time to fully understand what each line is doing and its down stream effects.