Multi-fidelity No-U-Turn Sampling

Kislaya Ravi, Tobias Neckel and Hans-Joachim Bungartz

Abstract Markov Chain Monte Carlo (MCMC) methods often take many iterations to converge for highly correlated or high-dimensional target density functions. Methods such as Hamiltonian Monte Carlo (HMC) or No-U-Turn Sampling (NUTS) use the first-order derivative of the density function to tackle the aforementioned issues. However, the calculation of the derivative represents a bottleneck for computationally expensive models. We propose to first build a multi-fidelity Gaussian Process (GP) surrogate. The building block of the multi-fidelity surrogate is a hierarchy of models of decreasing approximation error and increasing computational cost. Then the generated multi-fidelity surrogate is used to approximate the derivative. The majority of the computation is assigned to the cheap models thereby reducing the overall computational cost. The derivative of the multi-fidelity method is used to explore the target density function and generate proposals. We select or reject the proposals using the Metropolis Hasting criterion using the highest fidelity model which ensures that the proposed method is ergodic with respect to the highest fidelity density function. We apply the proposed method to three test cases including some well-known benchmarks to compare it with existing methods and show that multi-fidelity No-U-turn sampling outperforms other methods.

1 Introduction

Bayesian inference is a widely used method in many fields such as astrophysics, geological exploration, machine learning etc [1]. Exact inference of the posterior

Kislaya Ravi
Technical University of Munich, Garching, Germany. e-mail: kislaya@cit.tum.de

Tobias Neckel
Technical University of Munich, Garching, Germany. e-mail: neckel@cit.tum.de

Hans-Joachim Bungartz
Technical University of Munich, Garching, Germany. e-mail: bungartz@cit.tum.de

is rarely possible. The Markov Chain Monte Carlo (MCMC) method is one of the most commonly used methods in Bayesian inference to draw samples from the posterior. MCMC draws samples from the target density function that converge to the given distribution and are asymptotically unbiased. Classical methods such as the Metropolis-Hastings algorithm [2] and Gibbs sampling [3] often take a very long time to converge because of the inefficient random walk. This makes it intractable for computationally expensive models.

Hamilton Monte Carlo (HMC) [4] also known as hybrid Monte Carlo is a method that circumvents the random walk by using the derivative of the target density to propose better samples. There are two disadvantages of HMC. First, the user needs to specify some parameters of the algorithm manually. A poor choice of these parameters will have a considerable negative impact on the quality of drawn samples. Second, HMC requires the values of the derivative to be evaluated multiple times to propose one sample which means multiple evaluations of the target density function making it infeasible for computationally expensive models. The first issue can be addressed using No-U Turn sampling (NUTS) [5] which automatically determines the values of the crucial parameters. However, NUTS still requires access to the derivative.

In this work, we propose to replace the use of the actual model for derivative evaluation with a surrogate. The proposed sample is accepted/rejected based on the evaluation of the actual model. This ensures that the samples are invariant with respect to the target density function and not it's surrogate. Moreover, we also build the surrogate using multi-fidelity method.

One is often provided with different types of models solving the same phenomenon. We denote computationally cheap but inaccurate models as low-fidelity models whereas the expensive but accurate models are called high-fidelity models. We can combine these models efficiently for different applications by using the low-fidelity models more often than the high-fidelity function. Such methods fall under the category of multi-fidelity methods [6]. In this work, we use the concept of Gaussian Processes [7] to build multi-fidelity surrogates using a non-linear fusion of models [8, 9]. To the best of our knowledge, the algorithm proposed in this work is the first of its kind that applies multi-fidelity to NUTS. The algorithm is implemented in python and the implementation is publicly accessible [1]. This library can be easily coupled with solvers from various fields of application to generate multi-fidelity surrogates and solving computationally expensive problems such as forward uncertainty quantification, Bayesian optimization and MCMC sampling. In this work, we only discuss the MCMC sampling part of the implementation.

We first introduce the Bayesian inference setup in Section 2. Then, we provide a brief theoretical background for HMC and NUTS in Section 3. After that, we discuss the multi-fidelity method in Section 4 where we explain the multi-fidelity Gaussian Process surrogate, delayed acceptance algorithm and finally describe the proposed algorithm by combining all the previous methods. Finally, we test the performance of the proposed method and compare it with some of the commonly used sampling methods in Section 5 and summarise our conclusions in Section 6.

[1] https://github.com/KislayaRavi/MuDaFuGP

2 Bayesian Inference

Let $\boldsymbol{X} \in \mathbb{R}^d$ denote the parameter space and $\boldsymbol{Y} \in \mathbb{R}^m$ a separable Banach space that represents the data space. $d, m \in \mathbb{Z}^+$ are the dimensions of parameters and data respectively. Both the parameters and the data belong to the finite-dimensional spaces which allow us to work with densities with respect to the Lebesque measure. Let us consider a function $\mathcal{F} : \boldsymbol{X} \to \boldsymbol{Y}$, which is a map from the parameter space to the data space.

The noisy observations $\boldsymbol{y} \in \boldsymbol{Y}$ are typically modeled by adding some Gaussian noise $\boldsymbol{\eta} \sim \mathcal{N}(0, \Gamma)$, where Γ is a positive definite covariance matrix. For a given parameter $\boldsymbol{\theta} \in \boldsymbol{X}$, we can express observed data $\boldsymbol{y}$ as a random variable:

$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{\theta}) + \boldsymbol{\eta} \tag{1}$$

In inverse problems, the target is to solve (1) w.r.t. $\boldsymbol{\theta}$, given the observations $\boldsymbol{y}$, to infer the true parameters $\boldsymbol{\theta}^{true} \in \boldsymbol{X}$. However, this problem is ill-posed in the sense of Hadamard [10]. One can cure the ill-posedness of the problem by reformulating it as a Bayesian inference problem [1].

We consider the target $\boldsymbol{\theta}$ as a random variable that follows a prior distribution $p(\boldsymbol{\theta})$ on $\boldsymbol{X}$ which represents our assumption on the parameters without looking at the observations. Let us assume that $\boldsymbol{\theta}$ is independent of the noise $\boldsymbol{\eta}$. The likelihood $p(\boldsymbol{y}|\boldsymbol{\theta})$ represents the quality of an assumed $\boldsymbol{\theta}$ to produce the given observations $\boldsymbol{y}$. Because of Gaussian noise $\boldsymbol{\eta}$, the likelihood can be written as:

$$p(\boldsymbol{y}|\boldsymbol{\theta}) \propto \exp\left(-\frac{1}{2}||\Gamma^{-1/2}(\boldsymbol{y} - \mathcal{F}(\boldsymbol{\theta}))||^2\right) \tag{2}$$

In Bayesian Inference, we want to find the posterior distribution $p(\boldsymbol{\theta}|y)$, which represents the probability distribution of the parameter given the observation. Using Bayes theorem, one obtains

$$p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y})} \ . \tag{3}$$

The denominator term in (3) is known as evidence which is often difficult to calculate. However, it is independent of $\boldsymbol{\theta}$. There are multiple ways to draw samples from the posterior distribution. One of the commonly used methods is the Markov Chain Monte Carlo (MCMC) algorithm [4] which also circumvents the evidence term. In this work, we deal with No-U Turns Sampling (NUTS) [5] which we explain in the next section.

3 Hamilton Monte Carlo and No-U-Turn Sampling

No-U-Turn Sampling (NUTS) is a sampling method based on the Hamilton Monte Carlo (HMC) [4] method. HMC utilizes the geometry of the density function to propose better samples. We introduce an extra momentum variable $r \in \mathbb{R}^d$. The Hamiltonian of the system is defined using the state (θ) and the momentum term (r). Let $\mathcal{L}(\theta)$ be the logarithm of the target density function $\pi(\theta)$. With $r \cdot r$ representing the inner product of $\mathbb{R}^d$, the Hamiltonian of the system is defined as

$$H(\theta, r) = -\mathcal{L}(\theta) + \frac{1}{2} r \cdot r \ . \tag{4}$$

The canonical distribution of the system is

$$p(\theta, r) \propto \exp\{-H(\theta, r)\} \ . \tag{5}$$

At each iteration, the momentum term is resampled ($r \sim \mathcal{N}(0, \mathbb{I}_d)$) to propose the next state by solving the Hamiltonian dynamics for some time-steps. The Hamiltonian dynamics are numerically simulated using a time-stepping method which is reversible and volume-preserving. The leapfrog method is one of the most commonly used methods for this purpose and can be formulated as

$$\begin{aligned} r^{i+1/2} &= r^i + (\epsilon/2)\nabla_\theta \mathcal{L}(\theta^i) \\ \theta^{i+1} &= \theta^i + \epsilon r^{i+1/2} \\ r^{i+1} &= r^{i+1/2} + (\epsilon/2)\nabla_\theta \mathcal{L}(\theta^{i+1}) \ , \end{aligned} \tag{6}$$

where ϵ is the time-step size for the leapfrog method. We perform the fictitious time integration for given T steps to obtain the proposal $(\tilde{\theta}, \tilde{r})$. The proposal is accepted/rejected based on the acceptance probability α defined as:

$$\alpha = \min\left[1, \frac{\exp\{-H(\tilde{\theta}, \tilde{r})\}}{\exp\{-H(\theta, r)\}}\right] = \min\left[0, H(\theta, r) - H(\tilde{\theta}, \tilde{r})\right] \tag{7}$$

Note that the normalization term will get canceled out. So, we can sample from density functions that are not normalized.

Lemma 1 *The HMC algorithm is ergodic with respect to the canonical density function mentioned in* (5) *provided the leapfrog integrator does not generate periodic proposals.*

Proof Interested readers can refer to [4] for more detailed proof. If the leapfrog integrator does not generate periodic proposals then the Markov chain does not get trapped in certain subsets. We can also prove the detailed balance of the algorithm by taking into account the volume-preserving property of the leapfrog method. □

The quality of the samples depends on the choice of the tuning parameters, namely the number of steps T and the step size ϵ. NUTS is an extension of HMC, where the need to specify a fixed value of T is eliminated by performing the time integration until

one observes a U-turn. After the U-turn, the time integrator is very likely to retrace the old steps or visit states that are very close to already explored states. The slice sampling method is used to sample from the canonical distribution $p(\theta, r)$. The sign of the momentum term does not affect the value of $p(\theta, r)$. So, we build a tree while running the time integration. The first step is to sample the momentum ($r \sim \mathcal{N}(0, \mathbb{I}_d)$) as in standard HMC. We set the depth of the tree to zero ($l = 0$). At every level, we perform the time integration for 2^l steps to obtain $(\tilde{\theta}, \tilde{r})$ and increase the value of l by 1. At the end of time integration, we randomly draw either 1 or -1 with equal probability. If the number is 1, then we move to the rightmost node of the tree (θ^r) and perform the next time integration steps using the corresponding momentum term. If the number is -1, then we move to the leftmost node of the tree (θ^l) and perform the next time integration steps using the negative of the corresponding momentum term. At the end of every step, we check if we observe a U-turn. A U-turn is checked by verifying the following condition

$$(\theta^r - \theta^l) \cdot \tilde{r} < 0 \,. \tag{8}$$

We cut a random slice out of the canonical distribution $u \sim \mathcal{U}[0, p(\theta, r)]$. Out of all the states explored while building the tree, we select the states that satisfy the condition $p(\tilde{\theta}, \tilde{r}) \geq u$. We uniformly select one state out of all the chosen states which serves as the next state θ^{i+1}. We repeat the aforementioned steps until the required number of samples are obtained. Interested readers can refer to [5] for the detailed algorithm. The samples drawn using NUTS are ergodic with respect to the canonical density function [5].

One of the main challenges of any gradient-based sampling method is the calculation of the gradient. In many cases, the gradients are not available but one can use the numerical approximation of the gradients. With the help of auto differentiation methods, the derivatives are easier to calculate. However, we need to compute the derivative at every point of the time integration step. This is very resource-intensive for computationally expensive functions. In this work, we suggest building a computationally cheap multi-fidelity surrogate of the function to approximate the derivative. In the next section, we explain the multi-fidelity method in more detail.

4 Multi-fidelity Methods

For different applications, we often have a hierarchy of models. These functions model the same phenomena but with different levels of accuracy and resource requirements. Low-fidelity functions are computationally inexpensive but have relatively high errors. In contrast, high-fidelity functions are computationally expensive but accurately model the phenomena. We arrange the functions in increasing order of fidelity $\{f_1, f_2, ..., f_L\}$, where $f_i : \mathbb{R}^d \to \mathbb{R}$, $i = 1, 2, ..., L$. We need to evaluate the functions multiple times in applications that require uncertainty quantification, optimization, inverse problems, etc. Computing high-fidelity frequently is practically not possible

because of resource limitations. Moreover, we cannot completely replace the high-fidelity function with the low-fidelity function because we need the final result to be reasonably accurate, too. Multi-fidelity methods combine different fidelity functions to leverage the speed of the low-fidelity function and the accuracy of the high-fidelity method. A detailed survey of the different multi-fidelity methods is given in [6]. In this work, we use Gaussian Processes (GPs) to create the multi-fidelity surrogate.

We discuss a method to use GP to build a multi-fidelity surrogate in Section 4.1. The surrogate guides NUTS to propose samples. In Section 4.2 we discuss the delayed acceptance method that ensures ergodicity of samples with respect to the high-fidelity function. Finally, in Section 4.3 we combine all the different components to explain the multi-fidelity No-U Turn sampling method.

4.1 Multi-fidelity Gaussian Process Surrogates

Let us consider a two-fidelity system where $f_l(\theta)$ and $f_h(\theta)$ represent the low and high-fidelity functions, respectively, that reside on the same parameter space with points $\theta \in \mathbb{R}^d$. We perform a non-linear fusion of the different fidelities as discussed in [8, 9]. We first discuss non-linear auto-regressive Gaussian process (NARGP) [8] where the high-fidelity model is expressed as a function of the low-fidelity model and the parameter space as:

$$f_h(\theta) = g(f_l(\theta), \theta) \,, \tag{9}$$

where g is a function that resides in a $d+1$ dimensional space. In this way, one can explore the non-linear relationship between the high-fidelity and low-fidelity function. A Gaussian process (GP) [7] is used to express the function g because it provides not just the prediction but also the confidence interval of the predicted value. The kernel of NARGP is expressed in a way to mimic the auto-regressive structure as discussed in [11] as:

$$k_{NARGP} = k_f(f_l(\theta), f_l(\theta')|\lambda_f)k_\rho(\theta, \theta'|\lambda_\rho) + k_\delta(\theta, \theta'|\lambda_\delta) \,, \tag{10}$$

where k_f, k_ρ, k_δ are positive definite kernels and $\lambda_f, \lambda_\rho, \lambda_\delta$ are the corresponding hyperparameters. We use square exponential kernels in our work. However, one can choose a tailored kernel depending on the application case [7].

One can also include the derivative of the low-fidelity function in the formulation of the composite function g in (9). However, the derivative of the low-fidelity function may not be readily available. Instead of calculating the derivative using the finite difference method that can sometimes cause rounding errors, extra parameters are added in the composite function that corresponds to the lag term $f_l(\theta - \tau)$ and $f_l(\theta + \tau)$ which will mimic the derivative of the low-fidelity function, where τ represent a small real number. Now, the structure of the surrogate is:

$$f_h(\theta) = g(f_l(\theta), f_l(\theta + \tau), f_l(\theta - \tau), \theta) \tag{11}$$

We can use any kernel of our choice to build the surrogate. It does not need to follow a structure like NARGP. This multi-fidelity Gaussian process surrogate is discussed in [9]. We call this method Gaussian Process with Derivative Fusion (GPDF).

4.2 Delayed acceptance

The Delayed Acceptance (DA) algorithm [12] was developed to sample from a distribution $\pi(\theta)$ when there exists an approximation $\pi^*(\theta)$ of the target density distribution. The distribution does not need to be normalized. Just like any Metropolis-Hastings algorithm, one needs a proposal density function $q(\tilde{\theta}|\theta)$ and an initial point θ^0. The steps of the simplified version of DA are summarized below:

1. Draw a sample from the proposal distribution $\tilde{\theta} \sim q(\tilde{\theta}|\theta)$ and accept/reject the proposal for the next step based on the following acceptance probability:

$$\alpha^*(\tilde{\theta}|\theta) = \min\left\{1, \frac{\pi^*(\tilde{\theta})q(\theta|\tilde{\theta})}{\pi^*(\theta)q(\tilde{\theta}|\theta)}\right\} \tag{12}$$

2. If the sample is accepted in the previous step, then accept/reject the proposal $\tilde{\theta}$ based on the following acceptance probability:

$$\begin{aligned} q^*(\tilde{\theta}|\theta) &= \alpha^*(\tilde{\theta}|\theta)q(\tilde{\theta}|\theta) \\ \alpha(\tilde{\theta}|\theta) &= \min\left\{1, \frac{q^*(\theta|\tilde{\theta})\pi(\tilde{\theta})}{q^*(\tilde{\theta}|\theta)\pi(\theta)}\right\} \end{aligned} \tag{13}$$

3. The next sample is taken to be the same as the previous sample if the proposal is rejected at any of the previous two steps.
4. Repeat the first three steps until the required number of samples are drawn.

Lemma 2 *The DA algorithm preserves the detailed balance with respect to the target density* $\pi(\theta)$.

Proof We need to show $q(\tilde{\theta}|\theta)\alpha^*(\tilde{\theta}|\theta)\alpha(\tilde{\theta}|\theta)\pi(\theta) = q(\theta|\tilde{\theta})\alpha^*(\theta|\tilde{\theta})\alpha(\theta|\tilde{\theta})\pi(\tilde{\theta})$ to satisfy the detailed balance. The proposed sample is either accepted or rejected. Proving the detailed balance for rejected case ($\tilde{\theta} = \theta$) is trivial. Let us analyze the case when the sample is accepted. We look at the left-hand side of the target equation.

$$
\begin{aligned}
& q(\tilde{\theta}|\theta)\alpha^*(\tilde{\theta}|\theta)\alpha(\tilde{\theta}|\theta)\pi(\theta) \\
&= q(\tilde{\theta}|\theta)\min\left\{1, \frac{\pi^*(\tilde{\theta})q(\theta|\tilde{\theta})}{\pi^*(\theta)q(\tilde{\theta}|\theta)}\right\}\min\left\{1, \frac{q(\theta|\tilde{\theta})\min\left\{1, \frac{\pi^*(\theta)q(\tilde{\theta}|\theta)}{\pi^*(\tilde{\theta})q(\theta|\tilde{\theta})}\right\}\pi(\tilde{\theta})}{q(\tilde{\theta}|\theta)\min\left\{1, \frac{\pi^*(\tilde{\theta})q(\theta|\tilde{\theta})}{\pi^*(\theta)q(\tilde{\theta}|\theta)}\right\}\pi(\theta)}\right\}\pi(\theta) \\
&= q(\tilde{\theta}|\theta)\min\left\{1, \frac{\pi^*(\tilde{\theta})q(\theta|\tilde{\theta})}{\pi^*(\theta)q(\tilde{\theta}|\theta)}\right\}\min\left\{1, \frac{q(\theta|\tilde{\theta})\min\left\{1, \frac{\pi^*(\theta)q(\tilde{\theta}|\theta)}{\pi^*(\tilde{\theta})q(\theta|\tilde{\theta})}\right\}\pi(\tilde{\theta})\pi^*(\tilde{\theta})}{q(\tilde{\theta}|\theta)\min\left\{1, \frac{\pi^*(\tilde{\theta})q(\theta|\tilde{\theta})}{\pi^*(\theta)q(\tilde{\theta}|\theta)}\right\}\pi(\theta)\pi^*(\tilde{\theta})}\right\}\pi(\theta)\frac{\pi^*(\theta)}{\pi^*(\theta)} \\
&= \min\left\{\pi^*(\theta)q(\tilde{\theta}|\theta), \pi^*(\tilde{\theta})q(\theta|\tilde{\theta})\right\}\min\left\{\frac{\pi(\theta)}{\pi^*(\theta)}, \frac{\min\left\{q(\theta|\tilde{\theta})\pi^*(\tilde{\theta}), q(\tilde{\theta}|\theta)\pi^*(\theta)\right\}\pi(\tilde{\theta})}{\min\left\{q(\tilde{\theta}|\theta)\pi^*(\theta), q(\theta|\tilde{\theta})\pi^*(\theta)\right\}\pi^*(\tilde{\theta})}\right\} \\
&= \min\left\{\pi^*(\theta)q(\tilde{\theta}|\theta), \pi^*(\tilde{\theta})q(\theta|\tilde{\theta})\right\}\min\left\{\frac{\pi(\theta)}{\pi^*(\theta)}, \frac{\pi(\tilde{\theta})}{\pi^*(\tilde{\theta})}\right\}
\end{aligned}
$$

We observe that the final expression is symmetric about θ and $\tilde{\theta}$. So, we will arrive at the same expression if we simplified $q(\theta|\tilde{\theta})\alpha^*(\theta|\tilde{\theta})\alpha(\theta|\tilde{\theta})\pi(\tilde{\theta})$ which is the right-hand side of the detailed balance equation. This completes the proof. □

4.3 Multi-fidelity No-U Turn sampling

The steps of the Multi-fidelity No-U Turn sampling (MFNUTS) are shown in Algorithm 1. We can broadly divide the algorithm into two parts, namely the offline stage and the sampling stage. The offline stage includes building the surrogate and the sampling stage involves using the surrogate and the high-fidelity function to obtain samples. The first part of the algorithm is to build a Multi-fidelity Gaussian Process (MFGP) surrogate as described in Section 4.1. We build the surrogate of the logarithm of the target density function and randomly sample some points within the design space to build all the surrogates using NARGP and GPDF. Then, we select the model with the least mean squared error with respect to some test points. The surrogate is represented by $\mathcal{L}_s(\theta)$ and the corresponding canonical distribution as shown in (5) by $p_s(\theta, r)$.

The second step is to determine the step size (ϵ) of the leapfrog method. On the one hand, a big value of ϵ will cause the leapfrog method to visit a few states before a U-turn is observed. This corresponds to an improper exploration of states and may lead to many rejected samples. On the other hand, a very small value of ϵ results in the exploration of many nearby states before reaching the stopping criterion which is inefficient and leads to a too high acceptance ratio. So, we optimize the value of ϵ such that the expected value of the acceptance ratio reaches a target value (δ). In this work we use $\delta = 0.65$ following the argument from [4, 14]. To achieve this, the dual averaging technique as described in [5, 13] is used. Only the surrogate canonical distribution $p_s(\theta, r)$ is used to determine the step size. We are given a starting point θ_0 and we randomly sample a momentum term ($r_0 \sim \mathcal{N}(0, \mathbb{I}_d)$). We start with some assumed value of ϵ, perform one step of the leapfrog method, and

Algorithm 1: Multi-fidelity No-U Turn Sampling(MFNUTS)

Input: $\mathcal{F} := \{f_1, f_2, ..., f_L\}$; $\pi_L(\theta)$; θ^{upper}; θ^{lower}; θ^0; M^{adapt}; M^{samples}
Output: $\{\theta^1, \theta^2, ..., \theta^{M^{\text{samples}}}\}$
/* Build MFGP surrogate */
$f_s(\theta) \leftarrow$ BuildMFGP($\mathcal{F}$, θ^{upper}, θ^{lower})
Define log likelihood for surrogate ($\mathcal{L}_s(\theta)$) and corresponding density function ($\pi_s(\theta) := \exp\{\mathcal{L}_s(\theta)\}$)
Define the canonical distribution of surrogate $p_s(\theta, r) := \exp\{-\mathcal{L}_s(\theta) + \frac{1}{2} r \cdot r\}$, where r is the momentum term
/* Perform Adaptation steps using the surrogate to find step size */
$\epsilon \leftarrow$ FindStepSize($\mathcal{L}_s$, θ^0, M^{adapt})
/* Sampling step */
for $i = 1$ *to* $M^{samples}$ **do**
  /* Propose one sample by running one iteration of NUTS using the surrogate */
  $(\tilde{\theta}, \tilde{r}) \leftarrow$ NUTS (θ^{i-1}, ϵ)
  /* Accept or reject the proposal using DA algorithm */
  if $\tilde{\theta} \neq \theta^{i-1}$ **then**
    $\alpha_{\text{MFNUTS}}(\tilde{\theta}|\theta) = \min\left\{1, \frac{\min\left\{1, \frac{p_s(\theta,r)}{p_s(\tilde{\theta},\tilde{r})}\right\}\pi_L(\tilde{\theta})}{\min\left\{1, \frac{p_s(\tilde{\theta},\tilde{r})}{p_s(\theta,r)}\right\}\pi(_L\theta)}\right\}$
    if $\alpha_{MFNUTS}(\tilde{\theta}|\theta) > \mathcal{U}[0, 1]$ **then**
      $\theta^i = \tilde{\theta}$
    else
      $\theta^i = \theta^{i-1}$
  else
    $\theta^i = \theta^{i-1}$

check if the acceptance probability as shown in (7) is greater than 0.5. The value of ϵ is halved until the criterion is satisfied. This is considered as the starting value of the step size (ϵ_0) for the dual averaging algorithm which we run for some predefined steps. We call the part of finding the optimal value of the step size (ϵ) the adaptation step.

We start the actual sampling step after the adaptation step. In each sampling step, we run one step of NUTS as described in Section 3 using the surrogate canonical density function $p_s(\theta, r)$. Let the proposed sample be $(\tilde{\theta}, \tilde{r})$. The sample is accepted or rejected using the DA algorithm as described in Section 4.2. Let us represent the density function corresponding to the highest fidelity model as $\pi_L(\theta)$. We know that the leapfrog method is time-reversible. So, the proposal distribution become symmetric $q((\tilde{\theta}, \tilde{r})|(\theta, r)) = q((\theta, r)|(\tilde{\theta}, \tilde{r}))$. After substituting the previous assumption in (13), we obtain the acceptance ratio of the MFNUTS algorithm:

$$\alpha_{\text{MFNUTS}}(\tilde{\theta}|\theta) = \min\left\{1, \frac{\min\left\{1, \frac{p_s(\theta,r)}{p_s(\tilde{\theta},\tilde{r})}\right\}\pi_L(\tilde{\theta})}{\min\left\{1, \frac{p_s(\tilde{\theta},\tilde{r})}{p_s(\theta,r)}\right\}\pi_L(\theta)}\right\}. \tag{14}$$

Using Lemma 2, we can show that the samples generated conserve the detailed balance with respect to $\pi_L(\theta)$. If the leapfrog steps do not get stuck in periodic cycles, then the samples are ergodic with respect to the density function of the highest-fidelity model.

5 Numerical Results

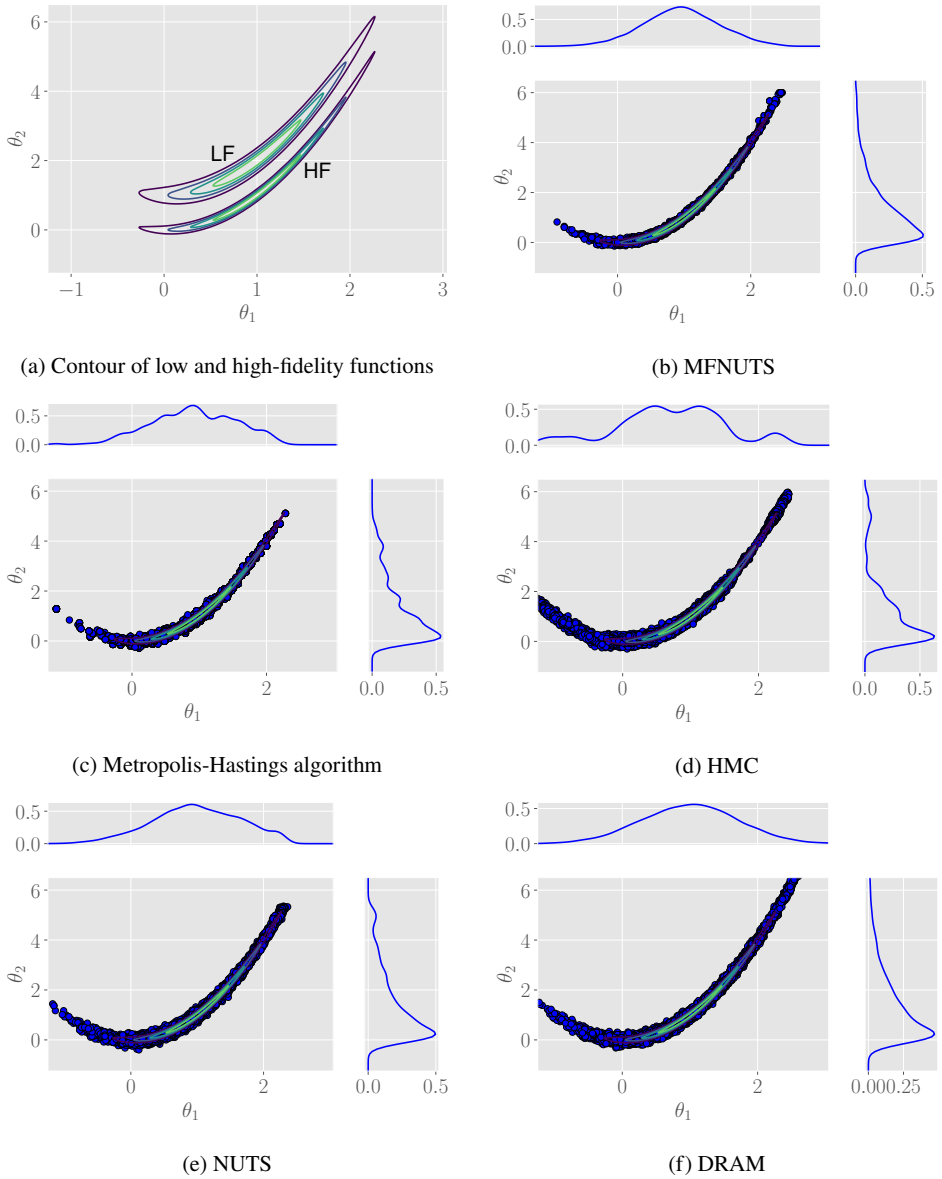
We are going to compare the MFNUTS algorithm with the Metropolis-Hastings algorithm, HMC, NUTS, and the Delayed Rejection Adaptive Metropolis (DRAM) [15] algorithm. The implementation of MFNUTS is available in the Github repository[2]. We use Paramonte [16] to run DRAM. Tensorflow Probability [17] is used to run the other three algorithms. We use three cases to compare the different methods and draw 10,000 samples for each case with 2,000 steps of adaptive or burn-in steps. The first case is drawing samples from the logarithm of the Rosenbrock function. This case checks the ability of the method to draw samples from a non-linear density function where a naive algorithm can result in the rejection of a lot of samples. The second case is drawing samples from an 8-dimensional correlated Gaussian distribution to check the performance for higher dimensions. Finally, we test the methods for calculating the intensity of multiple source terms in a steady-state groundwater flow problem. The Multivariate Effective Sample Size (mESS, see [18]) is used to compare the quality of samples drawn from each method. In many real-world applications, the evaluation of a surrogate is infinitely cheap as compared to the computation of the high-fidelity function. So, we can ignore the evaluation of the surrogate in calculating the computational cost. We compare mESS with respect to the number of high-fidelity evaluations. A higher value of mESS tantamounts to a better method. We also consider the high-fidelity function evaluations done in the offline and burn-in phases in the plots. To create the multi-fidelity surrogate, we start with some random points. We keep on adding points to the training set at the locations of highest variance until the mean squared error of the surrogate is of the order of 10^{-3}.

5.1 Rosenbrock function

We first test MFNUTS on a well-known benchmark test case. We take a two-fidelity scenario where the likelihood of the high-fidelity term $\mathcal{L}_2$ is the Rosenbrock function and the likelihood of the low-fidelity term $\mathcal{L}_1$ is a slightly modified Rosenbrock function:

$$\begin{aligned} \mathcal{L}_1(\theta_1, \theta_2) &= -12(\theta_2 - \theta_1^2 - 1)^2 + (\theta_1 - 1)^2 \,, \\ \mathcal{L}_2(\theta_1, \theta_2) &= -50(\theta_2 - \theta_1^2)^2 + (\theta_1 - 1)^2 \,. \end{aligned} \tag{15}$$

[2] https://github.com/KislayaRavi/MuDaFuGP



(a) Contour of low and high-fidelity functions

(b) MFNUTS

(c) Metropolis-Hastings algorithm

(d) HMC

(e) NUTS

(f) DRAM

Fig. 1 Contour of low and high-fidelity functions and the samples drawn from Rosenbrock function using different algorithms

Figure 1 shows that the samples drawn from all four methods represent the target density function. However, HMC and NUTS show a heavy tail as compared to the other methods because the tail regions are generally flat and a small value of the momentum term is not enough for the system to escape the region. MFNUTS has a relatively low number of samples in the tail region as compared to the other gradient-based methods because of the additional acceptance term (α_{MFNUTS}) which mimics the acceptance ratio of the Metropolis-Hastings algorithm.

We can observe from Figure 2 that MFNUTS outperforms all other methods. The mESS of the samples generated by the Metropolis-Hasting algorithm is worse than the one of NUTS and MFNUTS. This is because the Gaussian density function which is used to propose samples is not representative enough to model the complicated banana-shaped target density. Thus, a lot of samples are rejected leading to a repetition of states which lower the value of mESS. HMC has the worst performance amongst all the methods because we used the default value of the number of time steps for the leapfrog integration as provided by Tensorflow Probability instead of manually tuning it. NUTS generates a higher mESS value than HMC. This example shows the importance of the automatic selection of the number of steps. However, NUTS evaluated the model more frequently than HMC. Moreover, a lot of high-fidelity evaluations was also done during the adaptivity steps. DRAM has the highest mESS as compared to other methods but it requires a lot of high-fidelity function evaluations. Furthermore, DRAM needs extra function evaluations to learn the adaptive transition probability and propose another sample when the previously proposed sample is rejected. MFNUTS outperforms all methods by taking the good side of NUTS and replacing the high-fidelity function evaluation for derivative-evaluation with the surrogate. In most real-world scenarios, the cost of evaluating the surrogate is very small as compared to the one of the high-fidelity function. So, we circumvent the computationally expensive part by using the surrogate.

5.2 8-d correlated Gaussian distribution

In this section, we compare the performance of MFNUTS in a higher dimensional space. The low-fidelity function is a Gaussian distribution with zero mean and identity as the covariance matrix. The high-fidelity density function is also a Gaussian distribution with zero mean but now with a tridiagonal matrix as covariance. The log-likelihood of the low and high-fidelity density will be a sphere and an ellipsoid in 8-d space respectively. We use NARGP to learn the transformation between the log-likelihoods using 100 high-fidelity and 500 low-fidelity evaluations. The evolution of mESS with respect to the number of high-fidelity evaluations is shown in Figure 3. We again observe that MFNUTS outperforms the other methods. We also observe that the gradient-based algorithms (HMC and NUTS) and DRAM outperform the Metropolis-Hastings algorithm which is the general trend observed in high-dimensional sampling problems, since they provide a better proposal than the Metropolis-Hastings algorithm. We also observe that HMC has higher mESS for the

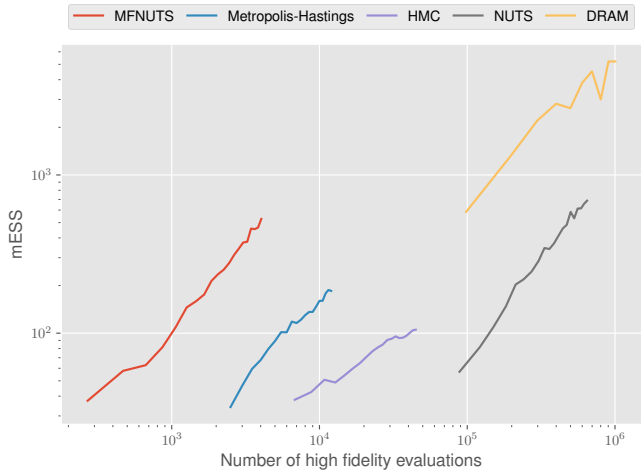


Fig. 2 mESS over the number of high-fidelity evaluations for Rosenbrock function

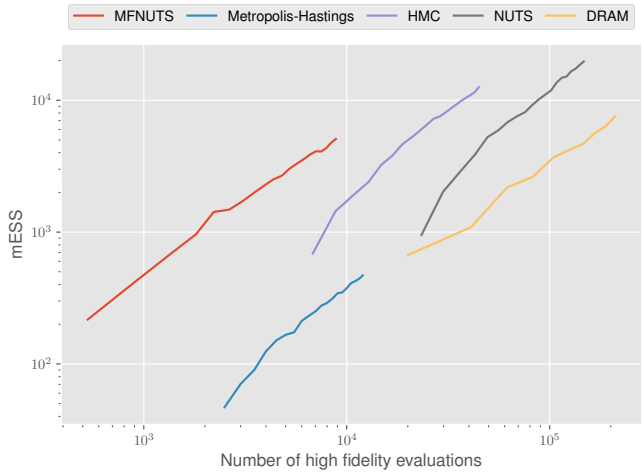


Fig. 3 mESS over the number of high-fidelity evaluations for 8-d Gaussian test case.

same number of high-fidelity evaluations. We assume that the default value of the number of integration steps was suitable for the target distribution. As in the previous example, MFNUTS is better than other methods by delegating the computationally expensive part of NUTS to the surrogate.

5.3 Steady-state groundwater flow

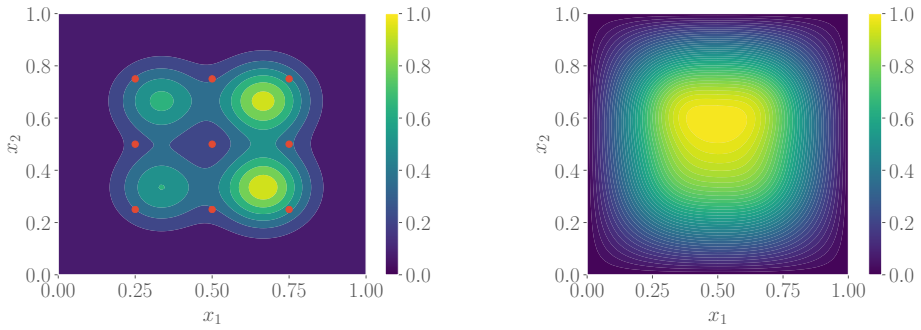
Let us consider a two-dimensional steady-state groundwater flow problem with source terms. The governing equation is:

$$\frac{\partial}{\partial X}\left(\kappa(X)\frac{\partial u}{\partial X}\right) = S(X) \qquad X \in \Omega \tag{16}$$

where, $\Omega := [0,1]^2$ represents the spatial domain, $\kappa(X)$ represents the diffusion coefficient and $S(X)$ represents the source term. We consider zero Dirichlet boundary conditions. For the given problem, we assume that the diffusion coefficient is constant ($\kappa(X) = 1$) and the source term is the summation of $N \in \mathbb{Z}$ Gaussian sources:

$$S(X) = \sum_{i=1}^{N} S_i(X) = \sum_{i=1}^{N} \theta_i \mathcal{N}(\mu_i, \sigma_i^2)\,, \tag{17}$$

where the i^{th} Gaussian source is defined by its location μ_i, variance σ_i^2 and intensity θ_i. We consider a case with four sources as shown in Figure 5.3, at locations $[(0.33, 0.33), (0.33, 0.67), (0.67, 0.33), (0.67, 0.67)]$ and each with variance 0.01. We put nine probes marked by red dots in the Figure 5.3. Our goal is to infer the source intensities for some given measurements at the probes. We use the open-source finite element solver FEniCS [19] to solve the differential equation. Measurement data is generated by solving (16) using a mesh size 64×64 with source intensity $\theta = [0.75, 1.25, 0.8, 1.2]$ and adding Gaussian noise with variance 0.005.



(a) Source term and the probe location

(b) Solution of groundwater flow equation

Fig. 4 Setup of the groundwater flow test case with $\theta = [0.75, 1.25, 0.8, 1.2]$

The unnormalized density functions $(\pi_1(\theta), \pi_2(\theta))$ are exponential of the corresponding likelihood function. The contour lines of the low- and high-fidelity density function is visualized in Figure 5.1 and are considerably different. If one directly uses the low-fidelity function as a guide to drawing proposals for the high-fidelity function

then it will lead to a lot of rejections. The transformation from the low-fidelity function to the high-fidelity function involves a translation and a non-linear shape modification. A linear function will not be sufficient to learn this transformation. Therefore, we use the non-linear transformation described in Section 4.1. The surrogate is created using 50 high-fidelity function evaluations and 200 low-fidelity function evaluations. In this case, GPDF generates the smallest mean squared error.

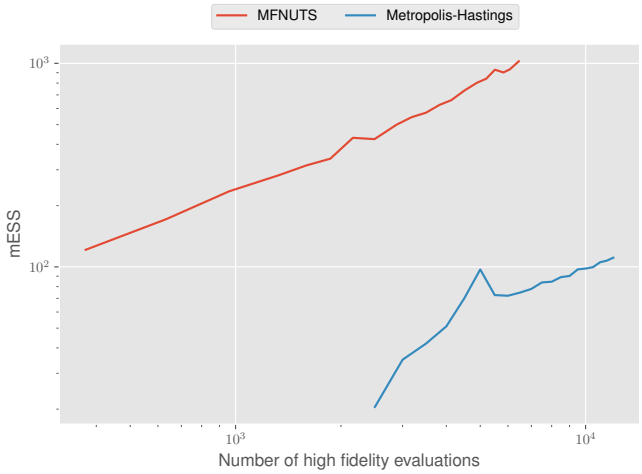


Fig. 5 mESS over the number of high-fidelity evaluations for steady-state groundwater flow case.

We model the likelihood function as Gaussian with variance 0.005 (approximately 1% of the mean value of the range u) and the prior as a Gaussian with mean 1.0 and variance 1.0. Then, we multiply the likelihood with the prior to get the unnormalized posterior distribution which is the target density function. For this test case, we only compare MFNUTS with the Metropolis-Hasting algorithm. The solver does not have support for auto-differentiation. So, the calculation of the derivative can only be done using the finite difference method which is computationally very demanding. So, NUTS and HMC have a clear disadvantage in this test case. We consider two fidelities of solvers for the multi-fidelity sampler. The low and high-fidelity solvers have a mesh size of 8×8 and 64×64, respectively. We create separate multi-fidelity surrogates for observations at all the probe locations using NARGP. 70 high-fidelity and 450 low-fidelity function evaluations were used for generating the surrogate. Then, we use all the surrogates to compute the posterior which is used as the final multi-fidelity surrogate for the MFNUTS sampler.

We observe from Figure 5 that the MFNUTS results in a considerably higher mESS value than the Metropolis-Hastings algorithm for a similar number of high-fidelity function evaluations, as observed in the previous two test cases. We also plot the samples drawn from the MFNUTS in Figure 6. The samples are mostly concentrated

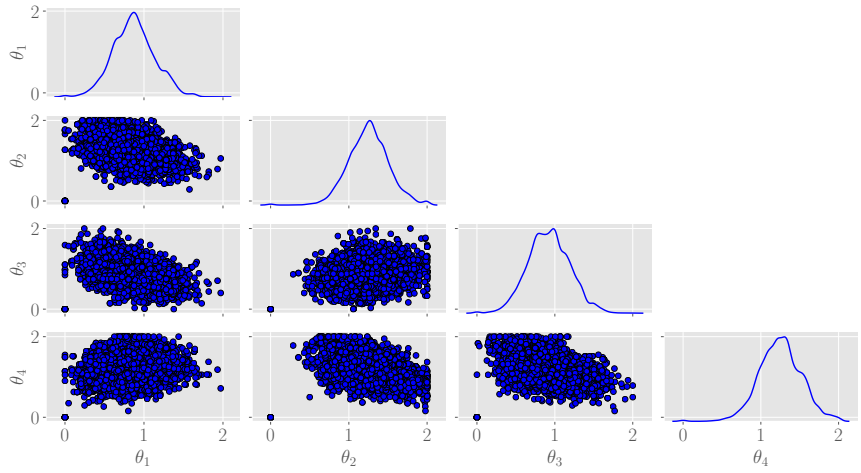


Fig. 6 Pairwise plot of the samples of source intensities from MFNUTS for groundwater flow test case

in elliptical blobs. The mean of the samples is $[0.87, 1.25, 0.92, 1.25]$ which is close to the source intensity that we used to generate the data.

6 Conclusion and future work

In this paper, we compare our proposed MFNUTS algorithm with existing single-fidelity sampling methods. In all three cases, MFNUTS outperforms the single-fidelity methods. This was achieved by taking advantage of NUTS and delegating the computationally expensive part to the surrogate. The importance of having higher mESS values and a proper exploration of the domain becomes very important when we deal with computationally expensive models. Bad proposals will lead to a waste of computational resources and low mESS will cause a high mean squared error. Our method will be particularly useful in those cases. Moreover, our method also generates samples that are invariant with respect to the high-fidelity model. However, the quality of the proposal depends upon the surrogate. If the surrogate itself has high error then the proposals will lead to a lot of rejections, thereby decreasing the effective sample size.

It is not essential to use the Gaussian process to build the surrogate. One can use any other method such as a neural network or a sparse grid approximation to build the surrogate. To further improve the algorithm, we can also add the Delayed Rejection [20] feature.

Acknowledgement

The present contribution is supported by the Helmholtz Association under the research school Munich School for Data Science - MUDS.

References

1. Kaipio J, Somersalo E. Statistical and computational inverse problems. Springer Science and Business Media; 2006 Mar 30.
2. Metropolis N, Rosenbluth A, Rosenbluth, M. Teller, and Teller E. Equations of state calculations by fast computing machines. Journal of Chemical Physics, 21:1087-1092, 1953.
3. Geman S, Geman D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Transactions on pattern analysis and machine intelligence. 1984 Nov(6):721-41.
4. Neal RM. MCMC using Hamiltonian dynamics. Handbook of markov chain monte carlo. 2011 Mar 2;2(11):2.
5. Hoffman MD, Gelman A. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res.. 2014 Apr 1;15(1):1593-623.
6. Peherstorfer B, Willcox K, Gunzburger M. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. Siam Review. 2018;60(3):550-91.
7. Rasmussen CE. Gaussian processes in machine learning. InSummer school on machine learning 2003 Feb 2 (pp. 63-71). Springer, Berlin, Heidelberg.
8. Perdikaris P, Raissi M, Damianou A, Lawrence ND, Karniadakis GE. Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2017 Feb 28;473(2198):20160751.
9. Lee S, Dietrich F, Karniadakis GE, Kevrekidis IG. Linking Gaussian process regression with data-driven manifold embeddings for nonlinear data fusion. Interface focus. 2019 Jun 6;9(3):20180083.
10. Hadamard J. Sur les problèmes aux dérivées partielles et leur signification physique. Princeton university bulletin. 1902:49-52.
11. Kennedy MC, O'Hagan A. Predicting the output from a complex computer code when fast approximations are available. Biometrika. 2000 Mar 1;87(1):1-3.
12. Christen JA, Fox C. Markov chain Monte Carlo using an approximation. Journal of Computational and Graphical statistics. 2005 Dec 1;14(4):795-810.
13. Nesterov Y. Primal-dual subgradient methods for convex problems. Mathematical programming. 2009 Aug;120(1):221-59.
14. Beskos A, Pillai N, Roberts G, Sanz-Serna JM, Stuart A. Optimal tuning of the hybrid Monte Carlo algorithm. Bernoulli. 2013 Nov;19(5A):1501-34.
15. Haario, H., Laine, M., Mira, A. and Saksman, E., 2006. DRAM: efficient adaptive MCMC. Statistics and computing, 16, pp.339-354.
16. Shahmoradi, A., Bagheri, F. and Kumbhare, S., 2020. Paramonte: Plain powerful parallel monte carlo library. Bulletin of the American Physical Society, 65.
17. Dillon J. V, Langmore I, Tran D, Brevdo E, Vasudevan S, Moore D, Patton B, Alex Alemi, Matt Hoffman, Rif A. Saurous. TensorFlow Distributions. arXiv preprint arXiv:1711.10604, 2017.
18. Vats D, Flegal JM, Jones GL. Multivariate output analysis for Markov chain Monte Carlo. Biometrika. 2019 Jun 1;106(2):321-37.
19. Scroggs M. W, Baratta I. A, Richardson C. N, and Wells G. N. Basix: a runtime finite element basis evaluation library, Journal of Open Source Software 7(73) (2022) 3982.
20. Tierney L, and Mira A, 1999. Some adaptive Monte Carlo methods for Bayesian inference. Statistics in medicine, 18(17-18), pp.2507-2515.