



# Parallelizing MCMC sampling via space partitioning

Vasyl Hafych<sup>1</sup> · Philipp Eller<sup>2</sup> · Oliver Schulz<sup>1</sup> · Allen Caldwell<sup>1</sup>

Received: 12 May 2021 / Accepted: 11 June 2022 / Published online: 27 June 2022  
© The Author(s) 2022

## Abstract

Efficient sampling of many-dimensional and multimodal density functions is a task of great interest in many research fields. We describe an algorithm that allows parallelizing inherently serial Markov chain Monte Carlo (MCMC) sampling by partitioning the space of the function parameters into multiple subspaces and sampling each of them independently. The samples of the different subspaces are then reweighted by their integral values and stitched back together. This approach allows reducing sampling wall-clock time by parallel operation. It also improves sampling of multimodal target densities and results in less correlated samples. Finally, the approach yields an estimate of the integral of the target density function.

**Keywords** MCMC Parallelization · Probability and statistics · Integral estimation

## 1 Introduction

Markov chain Monte Carlo (MCMC) is a technique that allows generating samples with a distribution proportional to a given target density function<sup>1</sup>. This technique is widely used in Bayesian statistics, statistical mechanics, computational biology, and many other fields or research. One of the major strengths of this technique is that it can converge to the target density even if target functions are highly multidimensional and multimodal. A major difficulty is that convergence is reached only asymptotically, and approaching the stationary distribution can require a very large number of sampling steps.

A Markov chain, by definition, consists of a series of consecutive steps that move a point or a set of points across the parameter space, which is an inherently serial process. A

proposed displacement, by definition of the Markov process, does not depend on the history that led to the current location. Determining whether to accept a displacement involves evaluating the target density at the proposed location. For many real-world applications, evaluation of a target density can be very computationally costly, and there is usually a limit to how far a single target evaluation can be parallelized efficiently; this can make MCMC sampling very costly. A further complication stems from the fact that a large number of burn-in steps (the steps necessary for the MCMC to reach the stationary distribution) need to be performed for each MCMC chain before representative samples can be generated. The burn-in duration can even exceed the sampling time, especially for target densities that have a complex shape. While separate MCMC chains can be run independently and in parallel, simply increasing their number while producing fewer samples from each chain is therefore not an effective parallelization strategy as the length of the burn-in process for each chain would not change.

Significant research has been conducted to enhance the efficiency of MCMC methods. The developments in this field can be divided into several categories (Robert 2018). The first is based on exploiting the geometry of the target density function. Hamiltonian Monte Carlo (Duane 1987) (HMC) belongs to this category and it introduces the auxiliary variable, called momentum, which is refreshed by using the gradient of the density function. Symplectic integrators of different orders of precision are developed to approximate Hamiltonian equation (Leimkuhler and Reich 2004;

---

✉ Vasyl Hafych  
hafychvasyl@gmail.com

Philipp Eller  
philipp.eller@tum.de

Oliver Schulz  
oschulz@mpp.mpg.de

Allen Caldwell  
caldwell@mpp.mpg.de

<sup>1</sup> Max Planck Institute for Physics, Munich, Germany

<sup>2</sup> Technical University of Munich, Munich, Germany

<sup>1</sup> In the following, we will use the terms ‘target density function’ also for unnormalized positive definite continuous functions.

Blanes et al. 2014). The HMC provides less correlated samples than the Metropolis-Hastings (MH) algorithm; however, the gradient of the density is not always readily available or cannot be computed in reasonable time. The second approach of accelerating MCMC is based on improving the proposal function. Techniques such as simulated tempering (Geyer 1991; Marinari and Parisi 1992), adaptive MCMC (Douc 2007), and multi-proposal MCMC (Liu et al. 2000; Bédard et al. 2012) are available and have been shown to be effective for many applications (Laloy and Vrugt 2012; Neal 1996; Xie et al. 2010; Carter and White 2013; Nampally and Ramakrishnan 2014). The third approach is based on breaking initially complicated problems into simpler pieces. For example, separate MCMC chains explore the parameter space in parallel and the resulting samples are merged together (Mykland et al. 1995; Neiswanger et al. 2013). As discussed earlier, this approach does not simplify convergence of chains to the stationary distribution.

It is also possible to partition the data space (Neiswanger et al. 2013; Scott 2016; Wang and Dun 2013) or parameter space (VanDerwerken and Schmidler 2013; Hallgren and Koski 2014; Basse et al. 2016) into simpler pieces that can be processed independently. The latter is the approach we pursue in this paper. An effective partitioning of the parameter space can change the task from sampling from a complicated target distribution to sampling from many, simpler target distributions. In our approach, any sampling algorithm can be used to generate samples in the subspaces, and all subspaces can be sampled independently and in parallel. This allows for massively parallel and distributed execution on multiple processors of multiple computer systems. After sampling, the integrated density of each subspace is calculated and the samples for each subspace are re-weighted correspondingly.

The general approach of parallelization of MCMC algorithms by partitioning parameter space is not entirely new and it has been studied in VanDerwerken and Schmidler (2013), Hallgren and Koski (2014), Basse et al. (2016), Kim and Lee (2020). The differences between our approach and those presented earlier are in the way we perform the space partitioning and in the way we reweight samples, and it will be discussed further below. But the common idea of these approaches results in increased MCMC sampling efficiency, yielding samples with reduced correlations. It also reduces the required MCMC burn-in time significantly since the possibly multiple modes of the full target density will ideally lie in separate subspaces; each chain will then only have to sample a unimodal density. In combination, these benefits result in a shorter total sampling time for each MCMC chain (including burn-in), which makes this approach of running many chains distributed over many subspaces efficient, whereas running many chains over the full density is not (due to constant burn-in overhead).

We start with a brief review of Bayesian data analysis, as this is the primary application that we aim for. In Sect. 3, the general idea of the algorithm and our implementation of it are described. In Sect. 4, we discuss differences between our approach and those presented in earlier literature. In Sect. 5, the performance of the algorithm is shown using two examples with the target functions given by a mixture of multivariate normal distributions. Finally, Sect. 6 summarizes developments presented in this paper and discusses further directions.

## 2 MCMC for bayesian data analysis

For the given model  $M$ , parameters  $\lambda$ , and the data  $\mathcal{D}$ , Bayes' theorem is defined as

$$P(\lambda|\mathcal{D}, M) = \frac{P(\mathcal{D}|\lambda, M)P_0(\lambda|M)}{P(\mathcal{D}|M)}, \quad (1)$$

where  $P(\mathcal{D}|\lambda, M)$  denotes the likelihood that is used to update the prior probability density  $P_0(\lambda|M)$  of  $\lambda$  to the posterior probability density  $P(\lambda|\mathcal{D}, M)$ . The denominator is usually called 'evidence' or 'marginal likelihood' and it is given by the Law of Total Probability

$$P(\mathcal{D}|M) = \int P(\mathcal{D}|\lambda, M)P_0(\lambda|M)d\lambda. \quad (2)$$

MCMC methods do not require knowledge of the normalization constant,  $P(\mathcal{D}|M)$ , to generate samples with the correct distribution. However, we require the normalizing constant in each subspace in order to reproduce the correct target distribution by patching together samples from different subspaces. Our approach thereby relies on being able to accurately calculate  $P_i(\mathcal{D}|M)$ , where  $i$  labels one of the subspaces. A variety of techniques that allow estimating the evidence of models exist, an overview summary can be found in Gelfand and Smith (1990), Friel and Wyse (2012). These typically require a re-sampling of the target probability density function once modes have been identified and only a few of the methods can be used in a post processing step based on existing samples. We use the recently published AHMI algorithm (Caldwell 2020) to evaluate the integrals in each subspace directly from the samples. This provides the user with the correctly weighted samples, and allows for a simple calculation of the evidence for the full target distribution. This is relevant for evaluating, amongst other things, the Bayes factor used in model comparisons.

### 3 Sampling parallelization via space partitioning

#### 3.1 Overview

We consider generating samples according to a target density function  $f(\lambda)$  where  $\lambda \in \mathbb{R}^m$  and  $\Omega$  is the support of the function. To illustrate our method, we will use as example the sum of four bivariate normal distributions with  $\lambda = (\lambda_1, \lambda_2)$ :

$$f(\lambda) = \sum_{i=1}^4 a_i \cdot \mathcal{N}(\lambda|\mu_i, \Sigma_i), \tag{3}$$

where  $a_1 = a_2 = 0.48$ ,  $a_3 = a_4 = 0.02$ ,  $\mu_i = (\pm 3.5, \pm 3.5)$ ,  $\Sigma_1 = \Sigma_2 = (0.33, 0.17; 0.17, 0.33)$ , and  $\Sigma_3 = \Sigma_4 = (0.019, -0.003; -0.003, 0.017)$ . Each bivariate normal distribution is individually normalized. Two, in the upper-right and lower-left quadrants, have large weights (0.48) and the other two, in the other quadrants, have small weights (0.02). The covariances are relatively small compared to the separations of the modes, making this a challenging target distribution to sample from for many MCMC algorithms. Probability contours of this test function are shown in Fig. 1a.

Our approach consists of the following four steps:

1. Generate a set of  $N_{exp}$  exploration samples  $\{\lambda_i^*\}_{i=1..N_{exp}} \in \Omega$ , distributed amongst  $N_{chains}$ , where  $N_{exp}$  is a small number compared to the desired number of final MCMC samples and  $N_{chains}$  is the number of chains. The chains should have different (possibly randomly chosen) starting points and can be run in parallel. The samples are used to find regions of the parameter space with a high density and the MCMC chains are not required to converge. An initial sampling of our example function with  $N_{exp} = 500$  generated using  $N_{chains} = 25$  with 20 samples per chain is shown in Fig. 1a.
2. Partition the parameter space into  $N_{sp}$  mutually exclusive subspaces

$$\{\omega_k\}_{k=1..N_{sp}} \in \Omega \tag{4}$$

in such a way that  $\cup \omega_k = \Omega$ ,  $\omega_k \cap \omega_m = \emptyset$  if  $k \neq m$  (see also Fig. 1b). While in general, the boundaries of the subspaces could be arbitrary shapes, in the following, ‘ $N$  space partitions’ will refer to  $N$  cuts along different, single parameter axes. This splits the parameter space into  $N + 1$  rectangular subspaces.

3. Generate  $N_{samp}^k$  samples  $\{\lambda_i^k\}_{i=1..N_{samp}^k} \in \omega_k$  in each subspace  $k$  with the distribution proportional to  $f(\lambda)\mathbb{I}_{\omega_k}(\lambda)$ , where  $\mathbb{I}_{\omega_k}(\lambda)$  denotes the so-called ‘indicator function’ that returns a 1 when  $\lambda$  is in  $\omega_k$  and 0 otherwise

(see Fig. 1c). There are no particular requirements for the choice of the sampling algorithm, so any sampler that will faithfully sample the truncated target density can be used. Note that each sampler has to perform its burn-in cycle only in the reduced subspace  $\omega_k$ , which significantly reduces tuning time.

If the sampling algorithm failed for some reason in one of the subspaces, e.g.,  $\omega_j$ , generate an additional partition in that subspace using the existing samples  $N_{samp}^j$ . Afterward, generate new samples in each of two resulting subspaces,  $\omega_{j,1}, \omega_{j,2}$ . Repeat this procedure until the convergence criteria is passed in each subspace or a maximum number of recursive iterations is reached.

4. Determine the integrated density of the target distribution in each subspace by computing  $I_k = \int_{\omega_k} f(\lambda)d\lambda$  and assign the following weights to the sample of subspace  $k$

$$w_k \propto \frac{I_k}{N_{samp}^k}.$$

5. Stitch the now weighted samples together resulting in the final sampling distribution (see Fig. 1d).

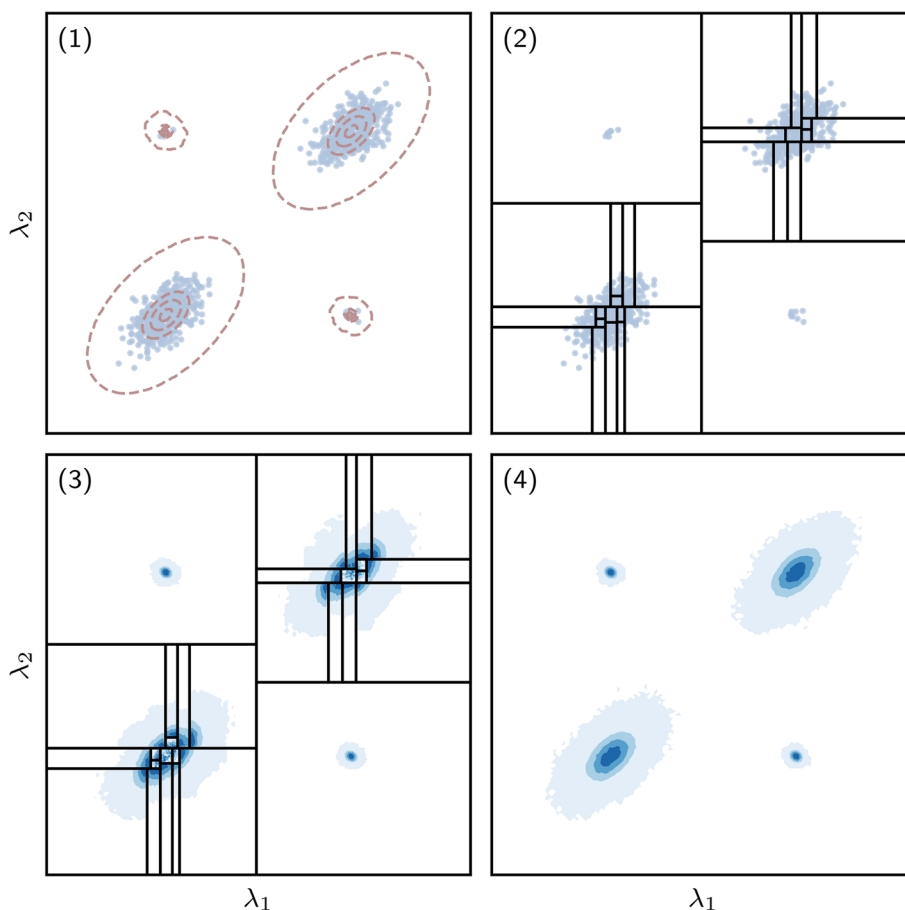
There are many ways of implementing the described idea based on choices of samplers, integrators and space partitioning strategies. In the following, we describe our implementation that is also made available in the Bayesian Analysis Toolkit (BAT.jl) package (Schulz 2021).

### 3.2 Implementation

#### 3.2.1 Exploration samples

Exploration samples play an important role in this algorithm, since the parameter space is partitioned based on them. If exploration samples represent the structure of the target density closely enough - for example indicating the presence of multiple modes by clusters of spatially neighboring points - then the space partitioning algorithm can capture these features and generate partitions in such a way to split those clusters. This simplifies the target density in each subspace and thus allows for much faster burn-in and tuning procedures. In our implementation, we generate exploration samples by running a large number of MCMC chains, where each chain generates a few hundred samples. There is no tuning or convergence requirement for these chains, but a small set of samples are initially used to set the parameters of the proposal functions for each chain. Some knowledge of the form of the target distribution is useful in determining how many chains and how many samples will be necessary. While the morphology of the resulting sample clouds should resemble that of the target density as closely as possible, this

**Fig. 1** Color maps displaying the steps in the partitioned sampling approach. (1) The 500 samples from the exploration step for the target density defined in Eq. 10. The red dashed lines demonstrate contours of the true density. (2) The result of partitioning the parameter space into 30 subspaces. The black lines demonstrate the boundaries of subspaces. (3) The  $10^4$  samples in each subspace from the individual MCMC chains. (4) The weighted samples displayed in the full space. (Color figure online)



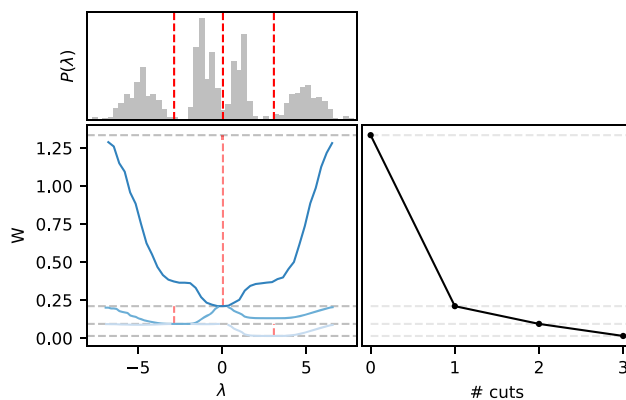
initial exploration should be fast compared to the following sampling time in the partitioned space.

### 3.2.2 Space Partitioning

Given the discussed exploration samples, we partition our parameter space into rectangular subspaces in such a way as to split clusters of spatially neighboring samples. To do so, a binary tree is used where each node is determined by a cut that is orthogonal to parameter axes. For the sake of illustration, we consider a one-dimensional problem and exploration samples  $\{\lambda\}$  (see upper histogram in Fig. 2). The cut position perpendicular to the  $\lambda$  axis is denoted as  $\tilde{\lambda}$  and it is selected by finding the minimum of the following cost function:

$$\tilde{\lambda} = \inf_a [W(a, \lambda)] = \inf_a \left[ \sum_{\lambda_i < a} |\lambda_i - \langle \lambda \rangle_{\lambda < a}|^2 + \sum_{\lambda_i > a} |\lambda_i - \langle \lambda \rangle_{\lambda > a}|^2 \right], \quad (5)$$

where  $\langle \lambda \rangle$  denote the mean of samples. This cost function represents second moments weighted by the number of samples on each side of the cut position, and it corresponds to the well-known  $k$ -means cost function (Lloyd 1982) for  $k = 2$ .



**Fig. 2** Illustration of the space partitioning algorithm using  $5 \cdot 10^3$  one-dimensional exploration samples  $\lambda^*$  with a distribution demonstrated in the upper left histogram. The red dashed lines demonstrate the first three cut positions  $\tilde{\lambda}$ . The blue lines show the value of  $W(a, \lambda)$  as a function of the cut position for 3 iterations of space partitioning. The gray dashed lines show the value of the cost function at its minimum for each iteration. The bottom right subplot illustrates the dependence of  $W$  as a function of the number of cuts. (Color figure online)

It was empirically observed that it allows obtaining a reasonable division of the parameter space, i.e., cutting multimodal densities into subspaces of unimodal densities, and prioritiz-

ing those dimensions of the multivariate distribution where multimodality is present.

The partitioning process is repeated iteratively until the desired number of partitions is reached. The blue lines in Fig. 2 demonstrate how this cost function depends on the cut positions for 3 partitioning steps. The partitioning procedure is ended when a minimal change in the cost function results from further partitioning or a maximum number of subspaces is reached. The evolution of the cost function for our example is also shown in Fig. 2.

The partitioning procedure is analogous for higher dimensional space. If, for instance, a sample vector is  $M$ -dimensional, then we evaluate Eq. 5 for every dimension which results in proposed cut positions  $\tilde{\lambda}_i$  with corresponding cost values  $W_i(a_i, \tilde{\lambda}_i)$  for dimension  $i = (1, \dots, M)$ . The minimum cost value is selected and the cut along the corresponding dimension is accepted. Additionally, if preliminary knowledge about the structure of the target density is present, the user can specify manually along which parameters partitioning of the parameter space should be performed.

### 3.2.3 Sampling

Sampling in the subspaces is performed independently and does not require communication between MCMC processes. It can therefore be trivially divided into tasks and executed in parallel on multiple processors using distributed computing. In the following, we define a ‘worker’ as a computing unit (this can either be a node of a cluster, networked machine, or a single machine) that consists of multiple CPU cores used to perform one task; i.e., sample the target density in one subspace. All the cores that belong to one worker are called threads and are used to run multiple MCMC chains in parallel within one subspace. Running multiple chains within one subspace is necessary for determining convergence of the MCMC process, which we test by using the Brooks–Gelman criteria (Gelman and Rubin 1992). Our implementation allows running subspaces on multiple remote hosts using Julia’s support for compute clusters, and MPI/TCP/IP protocols for communication between workers can be used. By default, we use the MH algorithm to generate samples on each subspace because this sampler is more general and does not require gradient information. However, one can use many other sampling algorithm, and in the following, we showcase an example that uses the HMC sampler.

It is possible that one or more modes of the target distribution were missed in the exploration step, or that the space partitioning algorithm did not choose the optimal mode separation locations. If the convergence criteria is not met in one or more of the subspaces, then these subspaces are further subdivided. The cut position is defined by using the already generated samples. These samples contain much more information about the target density structure than the

initial exploration samples and thus can be used as a more accurate approximation of the target density. This procedure is repeated until samplers in all subspaces report successful convergence tests, or a maximum number of partitioning cycles is reached.

We want to emphasize that by increasing the number of MCMC chains, we are increasing the chance to detect those modes that were missed during the initial exploration phase. If one of such new modes is detected by at least one chain, the convergence criteria will not be met and the algorithm will perform additional partition. Therefore, it is generally required to use a sufficient number of chains per subspace to make sure that all modes are discovered.

### 3.2.4 Reweighting

Samples that originate from different subspaces have different, and a priori unknown, normalizations with respect to each other. In order to correctly stitch those together, a weight proportional to the integral of the target density within the subspace needs to be applied. Given that samples are drawn from the target function  $\{\lambda^k\} \sim f(\lambda)$  in each subspace  $k$ , we approximate the following integrals

$$I_k = \int_{\omega_k} f(\lambda) d\lambda \tag{6}$$

using a recently developed AHMI algorithm (Caldwell 2020). In the following, we provide a brief description of the AHMI algorithm and we refer the reader to the original publication for more detailed information.

The central idea of the AHMI algorithm lies in constructing special rectangular subspaces<sup>2</sup>  $\Delta \in \omega$  and computing reduced harmonic mean estimates within these subspaces, i.e.

$$\begin{aligned} E \left[ \frac{1}{f(\lambda)} \right]_{f_{\Delta}(\lambda)} &= \int_{\Delta} \frac{1}{f(\lambda)} \cdot f_{\Delta}(\lambda) d\lambda \\ &= \int_{\Delta} \frac{1}{f(\lambda)} \cdot \frac{f(\lambda)}{I_{\Delta}} d\lambda = \frac{V_{\Delta}}{I_{\Delta}}, \end{aligned} \tag{7}$$

where we denote  $f_{\Delta}(\lambda) = \frac{f(\lambda)}{I_{\Delta}}$ , and  $V_{\Delta}$  defines a volume of the subspace. The estimator of this expectation value can be written as

$$\hat{X} = \frac{1}{N_{\Delta}} \sum_{\lambda_i \in \Delta} \frac{1}{f(\lambda_i)} = \frac{V_{\Delta}}{\hat{I}_{\Delta}}. \tag{8}$$

We scale the integral from the reduced subspace  $I_{\Delta}$  to the integral  $I$  over  $\omega$  by saying that  $r = I/I_{\Delta}$  and estimate it

<sup>2</sup> For notational convenience, we avoid subscript  $k$  in this paragraph and consider integration within one subspace

as  $\hat{r} = N/N_\Delta$ , where  $N$  denotes the number of samples per subspace  $\omega$ . Given this, the integral estimator can be written as

$$\hat{I} = \frac{\hat{I}_\Delta}{\hat{r}} = \frac{N_\Omega V_\Delta}{\sum_{\lambda_i \in \Delta} \frac{1}{f(\lambda_i)}}. \quad (9)$$

The crucial point of the AHMI algorithm is to define subspaces  $\Delta$  in such a way as to prevent the estimator  $\hat{X}$  — which requires evaluation of a possibly unstable  $1/f(\lambda)$  — from divergence. This adaptive procedure of finding optimal  $\Delta$  is performed by changing the size of  $\Delta$  and controlling that the ratio  $f_{max}/f_{min}$  is smaller than the given threshold. After multiple  $\Delta$  are determined, integral estimates associated with each region are computed and their values are combined to get the final estimate and its uncertainty. The AHMI algorithm converges to the truth if the samples generated on each subspace  $\omega$  has converged to the target distribution restricted to  $\omega$ .

We want to stress that there exist alternative approaches to evaluate normalization constants of the target density (e.g., bridge sampling Gelman and Meng 1998; Meng and Wong 1996; CUBA Hahn 2005). The main advantage of AHMI is that it estimates the integral of the function acting on the existing samples as a postprocessing step, with no requirements to reevaluate the target density. This can be especially beneficial if the evaluation of the target density is computationally challenging or not possible.

### 3.2.5 Final sample

Once the sampling and weighting are performed, the weighted samples from multiple subspaces are concatenated and returned to the user. The total integral of the function  $f(\lambda)$  is then estimated by summing the weights of the subspaces  $I = \sum_{k=1..N_{sp}} I_k$ .

Our implementation was used to generate the results shown in Fig. 1.

## 4 Previous work on sampling with space partitioning

As mentioned above, the idea to parallelize MCMC algorithms using space partitioning was already studied in a few earlier publications. For example, VanDerwerken and Schmidler (2013) proposed using adaptive Voronoi partitioning to divide the space into multiple non-overlapping subspaces, sample these subspaces using MH algorithm, and estimate weights of each subspace using bridge sampling or adaptive importance sampling. Another study, by Hallgren and Koski (2014), suggests dividing parameter space into overlapping regions and using the intersections to evaluate integrals.

In this study, the sampling was performed using a Particle Marginal MH sampler (Andrieu et al. 2010), and the main challenge of this approach was in finding a good space partition. A further improvement was presented by Basse, Smith and Pillai in Basse et al. (2016), where the authors proposed a general way to construct space partition using spectral clustering. As suggested in the paper, this space partitioning can work in some cases where the Voronoi algorithm fails; the reweighting of the samples in this approach was performed using bridge sampling and individual subspaces were sampled using the MH algorithm. A recent study by Kim and Lee (2020) demonstrated that sampling can also be significantly sped up if one uses a random rectangular partition scheme and explores each subspace using a constrained HMC algorithm. Similar to other techniques, this approach uses bridge sampling to reweight samples from different subspaces.

Our algorithm differs from the aforementioned approaches in a way (a) we partition parameter space and (b) reweight samples from different subspaces. A summary of key differences between our approach and those presented in the literature are highlighted in Table 1. Namely, we divide the space into non-overlapping rectangular subspaces that are constructed based on the initial exploration of the parameter space. Our space partitioning approach divides multimodal space into unimodal subspaces (sharing similarities with VanDerwerken et al. and Basse et al. approaches); but also we uses rectangular partitions (sharing similarities with an approach by Kim and Lee) because their complexity scale linearly with the number of dimensions, they allow for the trivial definition of the subspace boundaries and easy recursive partitioning. The rectangular subspaces are also well suited for algorithms like HMC since by appropriate parameters transformations they can be transformed into infinite spaces that are desirable for gradient-based samplers.

Another difference in our approach is that we do not require additional computation of the target density to reweight samples from different subspaces. The corresponding weights can be estimated acting on the existing samples within the subspace as a post-processing step. In addition, our approach allows recursive repartitioning of those subspaces that did not pass the convergence test. It is implemented as an open-source easy-to-use algorithm that utilizes the capabilities of Julia's distributed computing, allowing to run parallel sampling on a computing cluster with no additional code.

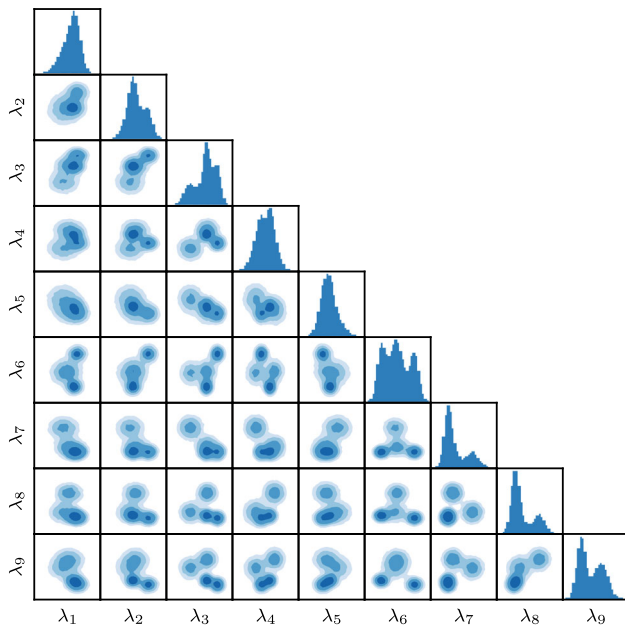
## 5 Performance

### 5.1 Example 1

In this example, we evaluate the performance of our algorithm on a more complicated test density function. The function was chosen in such a way to (a) have a known

**Table 1** Differences between previously illustrated approaches for sampling with space partitioning. The first three studies proposed to use any convenient sampling/integration methods, but we present here the algorithms they use in their showcase examples

Method	Space partition	Sampling	Reweighting
VanDerwerken et al.	Voronoi partition	MH	Importance/bridge sampling
Hallgren et al.	Overlapping partition	PMMH	From partition overlaps
Basse et al.	Spectral clustering	MH	Bridge sampling
Kim et al.	Random rectangular	HMC	Bridge sampling
Current proposal	Adaptive rectangular	MH/HMC	AHMI



**Fig. 3** One and two dimensional distributions of the density function given by Eq. 10. Histograms are constructed using  $10^5$  *i.i.d* samples

analytic integral, (b) allow generating independent and identically distributed (*i.i.d*) samples, and (c) have multiple modes in many dimensions and thus be challenging for a classical MH algorithm. With this aim, we have chosen a mixture of four multivariate normal distributions in 9-dimensional space:

$$f(\lambda) = \sum_{i=1}^4 a_i \mathcal{N}(\lambda | \mu_i, \Sigma_i), \tag{10}$$

where all  $a_i = 1/4$  and  $\mu$  and  $\Sigma$  are randomly assigned mean vector and a covariance matrix. The full parameter set used for this performance test are given in the Appendix. Figure 3 illustrates one and two dimensional distributions of  $10^5$  *i.i.d* samples drawn from this density.

There are two primary points that we demonstrate in this section. The first one is the ability to improve the wall-clock time spent on sampling by utilizing efficiently computational resources. The second one is the ability to improve the quality of samples once we increase the number of space partitions. Measurements of the performance were evaluated as follows:

- We use a varying number of subspaces  $S = (1, 2, 4, 8, 16, 32)$ . Sampling and integration in different subspaces are executed in parallel using 1 worker per subspace with 10 CPU cores per worker. All the CPU cores that are available for the worker are used for multithreaded chain execution.
- In addition, we also vary the wall-clock time that workers can spend on generating samples, considering time intervals of 3, 7, 11, and 15 seconds.
- For every combination of space partitions and wall-clock times, we repeat the sampling process 3 times to evaluate statistical fluctuations.

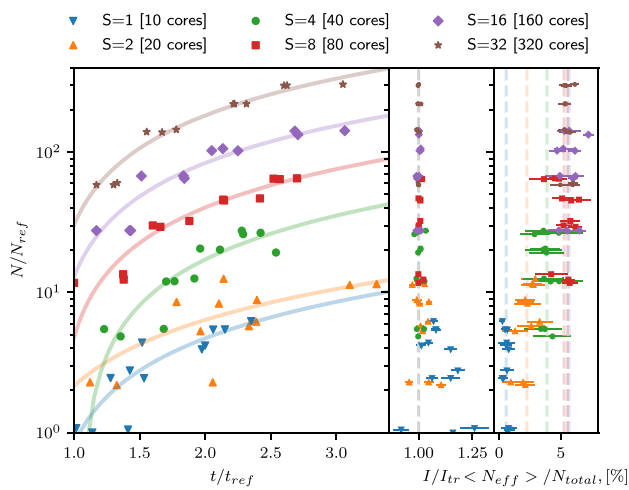
The overall number of MCMC runs is 72. An example run is: 8 subspaces and 8 workers with 10 CPU cores each are used to sample 10 chains for 11 seconds of wall-clock time, after which samples are integrated and returned; sampling with this setting is repeated 3 times.

### 5.1.1 Sampling Rate

A summary of the benchmark run is demonstrated in Fig. 4. We used the MPCDF HPC system DRACO<sup>3</sup> with Intel ‘Haswell’ Xeon E5-2698 processors (880 nodes with 32 cores @ 2.3 GHz each) to perform parallel MCMC executions. While sampling with space partitions, each subspace requires a different amount of time on sampling and integration, depending on the complexity of the underlying density region. The time of the slowest one is reported in Fig. 4. It can be seen that, by changing the number of subspaces from 1 to 32 (and the number of total CPU cores from 10 to 320), the number of generated samples increases almost two orders of magnitude while the wall-clock time remains constant.

Figure 4 can be rearranged into a slightly different form in order to demonstrate the sampling rate. We define  $N_0$  as the number of samples that the sampler with no space partitions has generated during the time interval  $\Delta t_0 = t_{stop} - t_{start}$  ( $t_{stop}$  is the wall-clock time when integration has finished and  $t_{start}$  is the wall-clock time when sampling on subspace has started). We further denote as  $N$  the total number of

<sup>3</sup> <https://www.mpcdf.mpg.de/services/computing/draco/about-the-system>



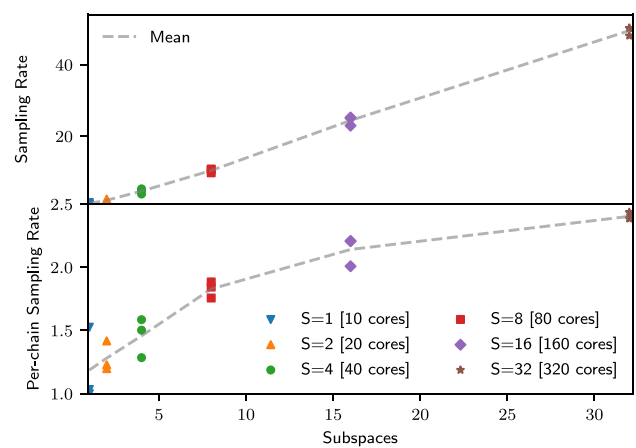
**Fig. 4** Summary of the benchmark runs for the target density function given by Eq. 10. The different colors represent runs with different numbers of partitions; the number of subspaces is denoted by  $S$ . The vertical axis is common for all subplots and gives the ratio of the total number of samples generated per single run to the number of samples that are generated if no space partition is performed ( $N_{ref} = 3.3 \cdot 10^4$ ). *Left subplot:* The horizontal axis shows the ratio of the time spent on sampling and integration to the time that a single worker spent if no space partition is performed ( $t_{ref} = 14.5$  s). The lines are from linear fits of measurements. *Middle subplot:* The horizontal axis shows the ratio of the integral to the true value; error bars are obtained from the integration algorithm. *Right subplot:* The horizontal axis illustrates the ratio of the effective number of samples (separately for each dimension) to the total number of samples. An effective number of samples is estimated per dimension and the error bars represent the standard deviation across dimensions. Dashed colored lines represent the average fraction over the runs with the same number of space partitions. (Color figure online)

samples generated on  $k$  subspaces, and the time spent on each subspace as  $\Delta t_k$ . The sampling rate is defined as

$$S = \frac{N}{\Delta t_{k,max}} \cdot \frac{\Delta t_0}{N_0}, \tag{11}$$

where we denote by  $\Delta t_{k,max}$  the longest run time among all subspaces. Intuitively, this equation compares how much wall-clock time was needed to generate one MCMC sample with and without space partitioning. We, however, do not refer to it as a speedup because during the fixed time we were tracking the number of generated samples, and not the time for the generation of a certain number of samples.

In addition to the wall-clock time, we also measure a CPU time spent on sampling and integration on each subspace<sup>4</sup>. We denote it as  $\tau_i$ , where  $i$  is the subspace index, and  $\tau_0$  is a CPU time when no space partitions are used. The per-chain



**Fig. 5** The figure illustrates sampling rate (upper subplot) and per-chain sampling rate (lower subplot) versus the number of space partitions. The gray lines represent average over 3 runs. (Color figure online)

sampling rate is defined as

$$S_{per-chain} = \frac{N}{\sum_{i=1..k} \tau_i} \cdot \frac{\tau_0}{N_0}, \tag{12}$$

and it intuitively measures how much CPU time was needed to generate one MCMC sample with space partitioning compared to the sampling without space partitioning. Equations 11 and 12 do not include time spent on the generation of exploration samples and construction of the partition tree. A time spent on the generation of exploration samples depends primarily on the complexity of a likelihood evaluation, and for our problem, it is equal to 4 seconds. The time required to generate the space partition tree primarily depends on the number of exploration samples, it is about 2 seconds for our problem.

The sampling rate and the per-chain sampling rate versus the number of space partitions are presented in Fig. 5. The figure shows that the sampling rates are improved for both cases. Improvements in the per-chain sampling rate indicates that by partitioning the parameter space we simplify the target density function resulting in faster tuning and convergence (tuning and convergence are occurring in every subspace). While improvement in the sampling rate is expected due to the scaling of the number of CPU cores, its faster-than-linear behavior can be explained by a superposition of the improved per-chain sampling rate and the linear sampling rate.

### 5.1.2 Density Integration and Effective Sample Size

Another important characteristic to track is the integral estimate of the target density function. If, for example, samples are not correctly representing the target function, then the integral will deviate from the truth. By partitioning the parameter space we are simplifying tasks for both the sam-

<sup>4</sup> Measurement of the CPU time is performed using the *CPUTime.jl* package. A detailed information about how the CPU time is defined and computed can be found in the package documentation.



pler and integrator; the complicated problem has been split into a number of simpler ones. This results in better integral estimates, which can be seen in Fig. 4 (middle subplot).

Samples that originated from one MCMC chain are correlated. A degree of sample correlation depends on the acceptance probability, number of chains, complexity of the target density function, etc. The effective number of samples can be estimated as

$$N_{eff} = \frac{N}{\hat{\tau}}, \tag{13}$$

where  $N$  is the number of samples and  $\hat{\tau}$  is the integrated autocorrelation time, estimated via the normalized autocorrelation function  $\hat{\rho}(\tau)$ :

$$\hat{\tau}_k = 1 + 2 \sum_{\tau=1}^{\infty} \hat{\rho}_k(\tau) \tag{14}$$

$$\hat{\rho}_k(\tau) = \frac{\hat{c}_k(\tau)}{\hat{c}_k(0)} \tag{15}$$

$$\hat{c}_k(\tau) = \frac{1}{N - \tau} \sum_{n=1}^{N-\tau} (\lambda_{k,i} - \hat{\lambda}_k) (\lambda_{k,i+\tau} - \hat{\lambda}_k) \tag{16}$$

where  $k$  refers to the one of the  $M$  dimensions of the multivariate sample  $\lambda_i = \{\lambda_{1,i}, \dots, \lambda_{M,i}\}$  and  $\hat{\lambda}_k$  is the average of the  $k$ -th component of all the  $N$  samples. In order to compute the sum given by Eq. 14, we use a heuristic cut-off given by Geyer’s initial monotone sequence estimator (Geyer 1992). This technique allows us to calculate an effective sample size for each dimension  $N_{eff,k} = \frac{N}{\hat{\tau}_k}$ . As it is shown in Fig. 4 (right subplot), the effective number of post-burn-in samples increases with the number of space partitions. It can also be seen that in our example there is no increase of the fraction of effective samples when the number of subspaces exceeds 8.

### 5.1.3 Summary of Scaling Performance

A summary of the performance enhancement from partitioning and sampling in parallel is presented in Fig. 6. The accuracy of the integrals of the function for different numbers of subspaces as a function of sampling wall-clock time is shown in the left panel. There, we see that even with the longest running times tested, the integral estimate without partitioning deviates considerably from the correct value. Good results are seen already for the shortest running times with 4 subspaces, and running on 32 subspaces gives excellent results in all running times tested.

The right panel in Fig. 6 shows the average number of effective samples for different combinations of numbers of subspaces and running times. We find a dramatic increase in the number of effective samples: the factor achieved in

Number of Subspaces	$I/I_{truth}$				$\langle N_{eff} \rangle / N_{ref}$			
	3	7	11	15	3	7	11	15
32	1.00	0.99	1.01	1.00	424	992	1509	2158
16	1.01	0.99	1.01	1.00	197	485	706	1072
8	1.02	1.01	1.01	1.02	81.9	223	340	358
4	1.03	1.03	1.01	1.03	25.4	65.3	94.7	128
2	1.10	1.04	1.05	0.97	4.8	17.9	23.6	41.8
1	1.26	1.18	1.15	1.08	1.0	1.4	3.5	3.4

**Fig. 6** The ratio of the integral to the true value (left panel) and the average number of effective MCMC samples (right panel) for the different numbers of subspaces and sampling times. In the right panel, the values are referenced to the average effective number of samples generated for one subspace and a running time of 3 seconds:  $N_{ref} = 267$

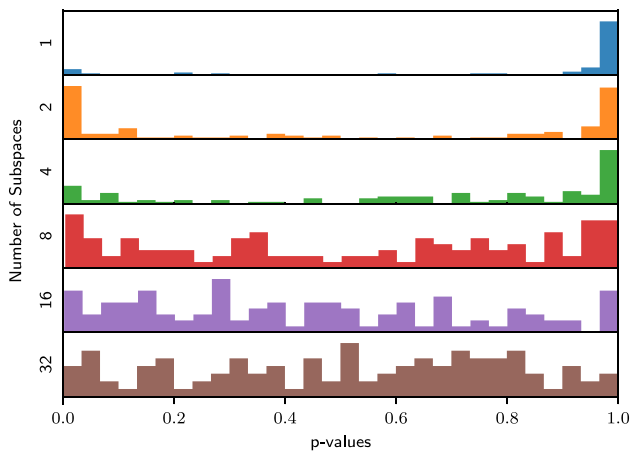
the effective number of samples is an order of magnitude larger than the increase in the computing resources (number of processors). This is due to the much simpler forms of the distributions sampled in the subspaces.

Both of these results show that a strong scaling of the performance is achieved using the partitioning scheme that we have outlined.

### 5.1.4 Sampling Accuracy

Since our algorithm requires the weighting of samples from different subspaces and stitching them together, we test whether this results in a smooth posterior approximating the true target density function. For comparison, we also approximate the true density by directly generating *i.i.d* samples. In the following, we will use the two-sample Kolmogorov-Smirnov test (Klotz 1967) and a two-sample classifier test (Lopez-Paz and Oquab 2016) as a quantitative assessment of how close our MCMC samples are to the *i.i.d* ones.

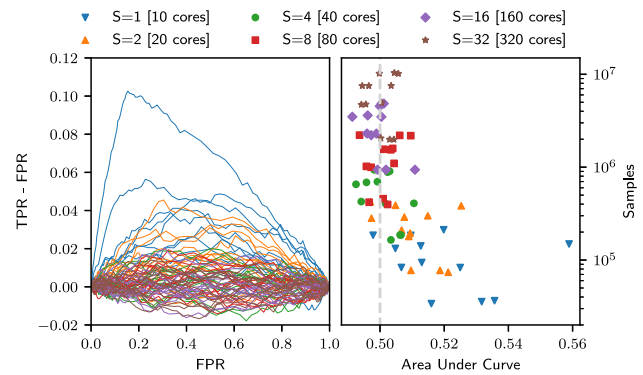
**Kolmogorov-Smirnov Test** The two-sample Kolmogorov-Smirnov test is used to test whether two one-dimensional marginalized samples come from the same distribution. P-values from this test for every marginal and different number of space partitions are shown in Fig. 7. We use the effective number of samples to calculate p-values for the Kolmogorov-Smirnov test. If two sets of samples stem from the same distribution, then the Kolmogorov-Smirnov p-values should be uniformly distributed. It can be seen that for a small number of space partitions p-values are peaking around 0 and 1. Peaks close to 1 indicate that the effective number of MCMC samples is likely underestimated, demonstrating that samples are very correlated. In contrast to this, the peaks near



**Fig. 7** The plot illustrates histograms of the p-values from the two-sample Kolmogorov-Smirnov test to verify whether MCMC and *i.i.d* samples come from the same distribution. Different colors represent different numbers of space partitions. One set of MCMC samples results in 9 p-values for every dimension. (Color figure online)

p-values of zero indicate that marginals of *i.i.d* and MCMC distributions deviate from each other. When using more than 4 partitions, p-values are uniformly covering the range from 0 to 1, indicating that marginals of *i.i.d* and MCMC samples are in a good agreement.

**Classification Test** A second approach to determining whether two samples are similar to one another uses a binary classifier aiming at distinguishing them. A training dataset is constructed by pairing MCMC and *i.i.d* samples with opposite labels. If samples are indistinguishable, the classification accuracy on the test dataset should be close to that obtained from a random guess. To train a classifier, we use a simple neural network model with two dense layers with sizes  $9 \times 20$  and  $20 \times 2$ , and the sigmoid activation function. To construct training and testing datasets, we generate MCMC and *i.i.d* samples with equal weights. By default, *i.i.d* samples come with weight one. For our weighted MCMC however, we generate  $3 \cdot 10^4$  samples with unit weights by using ordered resampling implemented in Schulz (2021). In total,  $4 \cdot 10^4$  samples are used for training and  $2 \cdot 10^4$  for testing with an equal fraction of MCMC and *i.i.d* samples. Training is performed for every MCMC run that is described in Fig. 4 individually and results are presented in Fig. 8. The left subplot in Fig. 8 shows the modified receiver operating characteristic (ROC), where the vertical axis is a difference between true positive rate (TPR) and false positive rate (FPR) for different MCMC runs. If the classifier cannot distinguish two samples, then the line will fluctuate around zero. The right subplot in Fig. 8 shows the integral under the ROC curve (expected to be close to 0.5 for indistinguishable distributions) versus the number of MCMC samples (before resampling was performed). It can be seen that even though the training datasets consist of the same number of MCMC



**Fig. 8** The figure illustrates the results of the classifier two-sample test performed to distinguish *i.i.d* samples and MCMC samples with space partitioning. The left subplot shows a modified receiver operating characteristic (ROC) where TPR stands for true positive rate and FPR for false positive rate. Different lines correspond to a different number of subspaces (*S*). The right subplot shows area under the ROC curve versus the number of samples. (Color figure online)

samples, there is a difference in their distinguishability. Samples obtained from the runs with a large number of space partitions have ROC curve integrals much closer to 0.5. It was not possible to detect this difference in the 1d Kolmogorov-Smirnov tests.

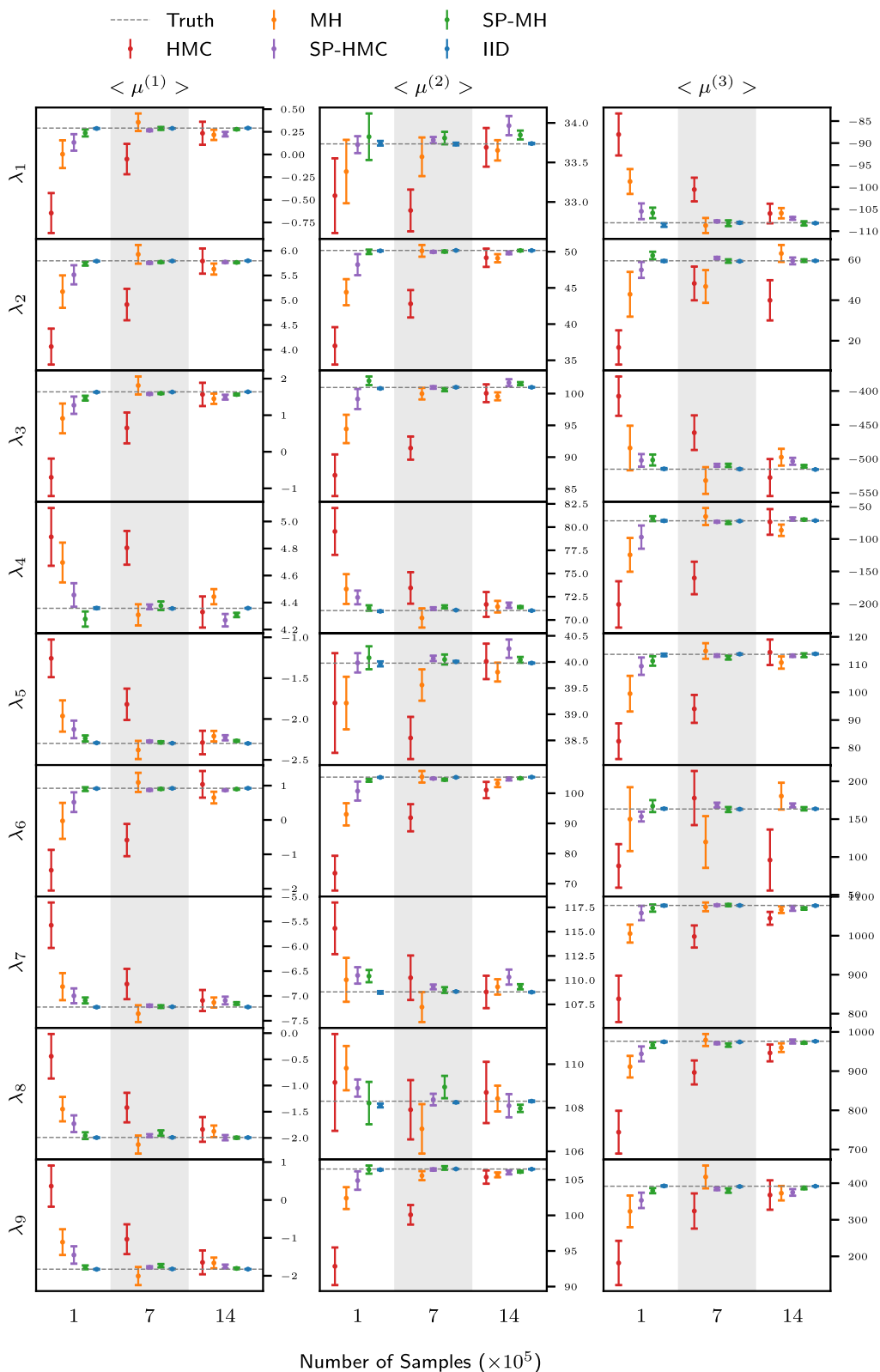
### 5.1.5 Comparison to Other MCMC Methods

Now, we would like to demonstrate how our sampling approach improves well-established MCMC algorithms, such as MH and HMC, by calculating the expectation values of the first three moments. We first sample the target density given by Eq. 10 using MH and HMC algorithms without space partitioning<sup>5</sup>. We then repeat the sampling using 10 space partitions, where the sampling within each subspace is performed using MH or HMC algorithm, and the exploration sampling is performed using the MH algorithm. The number of final samples generated by each sampling approach is kept constant, and we use the same convergence criteria for the four approaches. For the sake of illustration, we also present the expectation values estimated using *i.i.d* samples, which show the upper limit of the possible estimate accuracy.

We consider evaluation of the first three moments of the target probability density  $f(\lambda)$  given by

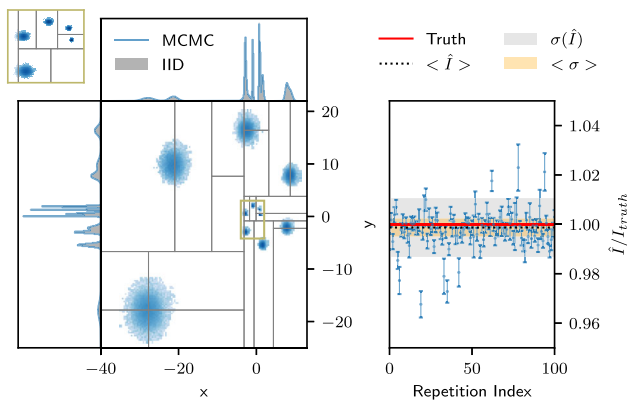
$$\begin{aligned} \mu^{(1)} &= \langle \lambda \rangle, \\ \mu^{(2)} &= \langle (\lambda - \langle \lambda \rangle)^2 \rangle, \\ \mu^{(3)} &= \langle (\lambda - \langle \lambda \rangle)^3 \rangle. \end{aligned} \tag{19}$$

<sup>5</sup> Detailed information about the implementation of these algorithms can be found in BAT.jl's documentation (Schulz 2021).



**Fig. 9** Estimation of the expectation values of the first three moments using five sampling algorithms: (HMC) Hamiltonian Monte Carlo with no space partition, (MH) Metropolis-Hastings with no space partition, (SP-HMC) Hamiltonian Monte Carlo with 10 space partitions, (SP-

MH) Metropolis-Hastings with 10 space partitions, (IID) *i.i.d* sampling. Error bars represent the standard error of the mean estimated from 20 repetitions



**Fig. 10** The left subplot shows the samples of the target density given by Eq. 18 obtained using the partitioned sampling. The gray lines show the boundaries of the subspaces. The horizontal and vertical histograms compare *i.i.d* and space partitioned samples. The area enclosed by a green color is zoomed in the upper left corner. The right subplot shows the density integral normalized by truth for 100 runs, and the error-bars show one standard deviation (that should satisfy the normality assumption due to the central limit theorem). The orange region shows the mean error, and the gray region shows the standard deviation of the means. (Color figure online)

Their true values are known analytically, and they can be used to compare estimates and the truth. Figure 9 demonstrates the results of the benchmark runs. We consider three sample sizes,  $10^5$ ,  $7 \times 10^5$ ,  $14 \times 10^5$ , and we repeat each run 20 times to account for statistical fluctuations. As shown in Fig. 9, the accuracy and the precision of the estimates generated by four methods increases with the number of samples. In our implementation, MH slightly outperforms the HMC in terms of accuracy. By performing 10 space partitions while keeping the same number of final samples, we were able to achieve a significantly better accuracy and precision of the estimates for both MH and HMC samplers. Namely, the algorithms without space partition needs to generate approximately 14 times more samples to reach the same accuracy as with 10 space partitions.

### 5.2 Example 2

In this example, we consider a problem in which the algorithm needs to detect subspaces where the convergence test was not passed and simplify and resample these subspaces until convergence is reached. We construct a target density function as a mixture of eleven bivariate normal distributions, in which covariances scale exponentially. The density function is defined as

$$f(x, y) = \sum_{i=0}^{10} a_i \mathcal{N}(x, y | \mu_{x,i}, \mu_{y,i}, \Sigma_i), \tag{18}$$

where

$$\begin{aligned} \mu_{x,i} &= e^{0.35i} \cos i, \\ \mu_{y,i} &= e^{0.35i} \sin i, \\ \Sigma_i &= \text{diag}(0.45\sqrt{\mu_{x,i}^2 + \mu_{y,i}^2}, 0.45\sqrt{\mu_{x,i}^2 + \mu_{y,i}^2}), \end{aligned} \tag{19}$$

and weights  $a_i$  are assigned randomly such that they are non-negative and their sum is equal to one. As in the previous example, this target density allows for *i.i.d* sampling, it is challenging for the MH algorithm, and the true value of the density integral is known.

We test our algorithm by first drawing exploration samples with 30 chains and 700 samples per chain. The chains typically get trapped in one of the modes of the target density, and their samples do not represent the entire space correctly. Our algorithm generates 17 initial space partitions using the exploration samples. Due to imperfect exploration sampling, some of these subspaces contain regions with multiple modes. We find that in repeated testing, typically 3 additional subspaces are generated due to a convergence failure. An example of posterior samples from one run and a summary of 100 runs are shown in Fig. 10. It can be seen that the marginals of the MCMC and *i.i.d* samples overlap, indicating that the target density was sampled accurately. Also, the density integral for 100 runs is close to the true value, with a small average underestimation of 0.1%, and the error bars have coverage of 48%.

## 6 Conclusions

We have presented an approach to both improve and accelerate MCMC sampling by partitioning the parameter space of the target density function into multiple subspaces and sampling independently in each subspace. These subspaces can be sampled in parallel and the resulting samples then stitched together with appropriate weighting. The scheme relies on a good space partitioning, which we achieve using a binary partitioning algorithm, that can recursively generate new space partitions in those subspaces where convergence tests of the samplers were unsuccessful; and a good integrator for determining the weights assigned to the samples in the different subspaces. The integrations in our examples were performed using the AHMI (Caldwell 2020) algorithm. The parallelized MCMC sampling algorithm described in the paper has been implemented in the Bayesian Analysis Toolkit (Schulz 2021). This approach provides the user with a normalization constant of the target density function - the Bayesian evidence is provided at no extra cost.

We have benchmarked this technique by evaluating the quality of samples and the sampling rate for a mixture of four multivariate normal distributions in 9-dimensional space. We

demonstrate that the space partitioning allows us to obtain a 50-fold increase in the sampling rate while increasing the number of CPUs by a factor 32. This increase is a superposition of two effects: a linear scaling with the number of CPU cores, and a CPU-time reduction due to the simplification of the target density function. In addition to the increase in the sampling rate, sampling with space partitioning also resulted in an increased quality of MCMC samples by reducing their correlations. This was evidenced in particular by more accurate integral values of the target density, and more accurate estimates of the moments of the distribution.

We have evaluated the correctness of the resulting sampling distributions by comparing the MCMC samples with *i.i.d* samples using a two-sample Kolmogorov-Smirnov test and with a two-sample classifier test. Both show that increasing the number of space partitions leads to a better agreement between MCMC and *i.i.d* samples.

We have also demonstrated an example of a recursive partitioning scheme using a target density with exponentially scaled covariances. In this problem, the algorithm was repartitioning recursively those subspaces where the MCMC chains failed convergence tests. The estimated evidence averaged over 100 runs is consistent with the true value.

**Funding** Open Access funding enabled and organized by Projekt DEAL. Open Access funding enabled and organized by Projekt DEAL.

### Declarations

**Funding** This research was supported by the European Union’s Framework Programme for Research and Innovation Horizon 2020 (2014–2020) under the Marie Skłodowska-Curie Grant Agreement No.765710 and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy - EXC-2094 - 390783311.

**Conflicts of interest** The authors declare that they have no conflict of interest.

**Code Availability** The source code of BAT.jl is available at <https://github.com/bat/BAT.jl> and under DOI 10.5281/zenodo.2587213, as well as via the Julia package management system. BAT.jl is published under the MIT open-source license (The MIT License 2020).

**Data Availability** The datasets synthetically generated by the the method described in the paper was used.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the

permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## Appendix 1

Covariance matrices used in Eq. 10 are diagonal with the size  $9 \times 9$  and are equal to

$$\begin{aligned} \Sigma_1 &= \text{diag}(12.64, 12.64, \dots, 12.64), \\ \Sigma_2 &= \text{diag}(10.48, 10.48, \dots, 10.48), \\ \Sigma_3 &= \text{diag}(33.03, 33.03, \dots, 33.03), \\ \Sigma_4 &= \text{diag}(27.45, 27.45, \dots, 27.45). \end{aligned} \tag{20}$$

Mean vectors are

$$\begin{aligned} \mu_1 &= (4.6, 14.8, 12.7, 0.4, -7.3, 14.5, -14.0, -9.8, -12.3), \\ \mu_2 &= (2.5, 2.9, 2.7, 8.7, -1.6, -11.0, -14.0, -7.5, -8.7), \\ \mu_3 &= (-4.8, 0.68, -12.0, -5.0, 4.4, -0.45, 8.7, -4.5, 2.8), \\ \mu_4 &= (-1.1, 4.8, 3.3, 13.0, -4.6, 0.99, -9.5, 14.0, 11.0). \end{aligned} \tag{21}$$

## References

Andrieu, C., Doucet, A., Holenstein, R.: Particle markov chain monte carlo methods. *J. of the Royal Stat. Society: Ser. B (Statistical Methodology)* **72**(3), 269–342 (2010)

Basse, G., Smith, A., Pillai, N.: Parallel Markov chain Monte Carlo via spectral clustering. *Artificial intelligence and statistics*. 1318–1327 (2016)

Bédard, M., Douc, R., Moulines, E.: Scaling analysis of multiple-try MCMC methods. *Stochastic Process. and their Appl.* **122**(3), 758–786 (2012)

Blanes, S., Casas, F., Sanz-Serna, J.M.: Numerical integrators for the Hybrid Monte Carlo method. *SIAM J. on Scientific Comput.* **36**(4), A1556–A1580 (2014)

Caldwell, A., et al.: Integration with an adaptive harmonic mean algorithm. *International J. of Modern Phys. A* **35**(24), 2050142 (2020)

Carter, J.N., White, D.A.: History matching on the Imperial College fault model using parallel tempering. *Comput. Geosciences* **17**(1), 43–65 (2013)

Douc, R., et al.: Convergence of adaptive mixtures of importance sampling schemes. *The Annals of Stat.* **35**(1), 420–448 (2007)

Duane, S., et al.: Hybrid monte carlo. *Phys. letters B* **195**(2), 216–222 (1987)

Friel, N., Wyse, J.: Estimating the evidence—a review. *Statistica Neerlandica* **66**(3), 288–308 (2012)

Gelfand, A.E., Smith, A.F.M.: Sampling-based approaches to calculating marginal densities. *J. of the Am. stat. assoc.* **85**(410), 398–409 (1990)

Gelman, A., Meng, X.-L.: Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. *Statistical science* 163–185 (1998)

Gelman, A., Rubin, D.B.: Inference from iterative simulation using multiple sequences. *Stat. sci.* **7**(4), 457–472 (1992)

Geyer, C.J.: Markov chain Monte Carlo maximum likelihood. In: (1991)

- Geyer, C.J.: Practical markov chain monte carlo. In: *Statistical science* 473–483 (1992)
- Hahn, T.: Cuba—a library for multidimensional numerical integration. *Computer Phys. Commun.* **168**(2), 78–95 (2005)
- Hallgren, J., Koski, T.: Decomposition sampling applied to parallelization of Metropolis-Hastings. In: (2014). arXiv preprint [arXiv:1402.2828](https://arxiv.org/abs/1402.2828)
- Kim, M., Lee, J.: Hamiltonian Markov chain Monte Carlo for partitioned sample spaces with application to Bayesian deep neural nets. *J. of the Korean Stat. Soc.* **49**(1), 139–160 (2020)
- Klotz, J.: Asymptotic efficiency of the two sample Kolmogorov-Smirnov test. *J. of the Am. Stat. Assoc.* **62**(319), 932–938 (1967)
- Laloy, E., Vrugt, J.A.: High-dimensional posterior exploration of hydrologic models using multiple-try DREAM (ZS) and high-performance computing. *Water Resources Research* **48**(1), (2012)
- Leimkuhler, B., Reich, S.: *Simulating hamiltonian dynamics*, vol. 14. Cambridge university press (2004)
- Liu, J.S., Liang, F., Wong, W.H.: The multiple-try method and local optimization in Metropolis sampling. *J. of the Am. Stat. Assoc.* **95**(449), 121–134 (2000)
- Lloyd, S.: Least squares quantization in PCM. *IEEE trans. on information theory* **28**(2), 129–137 (1982)
- Lopez-Paz, D., Oquab, M.: Revisiting classifier two-sample tests. In: (2016). arXiv preprint [arXiv:1610.06545](https://arxiv.org/abs/1610.06545)
- Marinari, E., Parisi, G.: Simulated tempering: A new Monte Carlo scheme. *EPL (Europhysics Letters)* **19**(6), 451 (1992)
- Meng, X.-L., Wong, W.H.: Simulating ratios of normalizing constants via a simple identity: A theoretical exploration. *Statistica Sinica* 831–860 (1996)
- Mykland, P., Tierney, L., Yu, B.: Regeneration in Markov chain samplers. *J. of the Am. Stat. Assoc.* **90**(429), 233–241 (1995)
- Nampally, A., Ramakrishnan, C.R.: Adaptive MCMC-based inference in probabilistic logic programs. In: (2014). arXiv preprint [arXiv:1403.6036](https://arxiv.org/abs/1403.6036)
- Neal, R.M.: Sampling from multimodal distributions using tempered transitions. *Stat. and comput.* **6**(4), 353–366 (1996)
- Neiswanger, W., Wang, C., Xing, E.: Asymptotically exact, embarrassingly parallel MCMC. In: (2013). arXiv preprint [arXiv:1311.4780](https://arxiv.org/abs/1311.4780)
- Robert, C.P., et al.: *Accelerating MCMC algorithms*. Wiley Interdisciplinary Reviews: Comput. Stat. **10**(5), e1435 (2018)
- Schulz, O., et al.: BAT.jl: A Julia-Based Tool for Bayesian Inference. *SN Computer Sci.* **2**(3), 1–17 (2021)
- Scott, S.L., et al.: Bayes and big data: The consensus Monte Carlo algorithm. *International J. of Management Sci. and Eng. Mana* **11**(2), 78–88 (2016)
- The MIT License. <https://opensource.org/licenses/MIT>. Accessed: 2020-07-23
- VanDerwerken, D.N., Schmidler, S.C.: Parallel markov chain monte carlo. In: (2013). arXiv preprint [arXiv:1312.7479](https://arxiv.org/abs/1312.7479)
- Wang, X., Dunson, D.B.: Parallelizing MCMC via Weierstrass sampler. In: (2013). arXiv preprint [arXiv:1312.4605](https://arxiv.org/abs/1312.4605)
- Xie, Y., Zhou, J., Jiang, S.: Parallel tempering Monte Carlo simulations of lysozyme orientation on charged surfaces. *The J. of chemical phys.* **132**(6), 02B602 (2010)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.