Density function estimation using ergodic recursion

Erik Bodin University of Bristol
Zhenwen Dai Spotify Research
Neill D. F. Campbell University of Bath
Carl Henrik Ek University of Cambridge

Abstract

We present a novel approach to Bayesian inference and general Bayesian computation that is defined through a recursive partitioning of the sample space. It does not rely on gradients, nor require any problem-specific tuning, and is asymptotically exact for any density function with a bounded domain. The output is an approximation to the whole density function including the normalization constant, via partitions organized in efficient data structures. This allows for evidence estimation, as well as approximate posteriors that allow for fast sampling and fast evaluations of the density. It shows competitive performance to recent state-of-the-art methods on synthetic and real-world problem examples including parameter inference for gravitational-wave physics.

1 Introduction

Bayesian methods require the computation of posterior densities, expectations and model evidence. In all but the simplest conjugate models, these calculations are intractable and require numerical approximations to be made. In this work, we propose sample-efficient and scalable construction of discrete approximations to black-box continuous density functions as a means to provide such results with one simple-to-use algorithm. In practice, density functions are typically unnormalized joint distributions of data and parameters, where the posterior distribution or model evidence is of interest. In this work, the true density function is assumed explicitly unknown or non-differentiable, meaning it can only be point-wise evaluated. The output is a piece-wise constant approximation of the function, organized in data structures allowing efficient operations. With this, we obtain properties that are typically only associated with standard families of parametric distributions, such as fast sampling, fast density evaluations, and tractable quantities like entropy.

Motivation for use cases of this algorithm can be found in natural science research areas such as gravitational-wave physics [1, 2, 3]. In GW research, scientific knowledge is often expressed in the form of physically motivated likelihood functions and priors [4]. An increasing sophistication has brought challenges from an inference standpoint. Tractable gradients are often missing, the functions may exhibit undefined (or zero density) regions, and the induced density surfaces tend to be multi-modal, discontinuous, have mass concentrated in tiny regions, and exhibit complicated correlations. As a consequence, inference can be prohibitively slow even for problems of around ten dimensions. Simple techniques such as standard rejection sampling are typically infeasible due to small typical sets, requiring a prohibitively large number of function evaluations to obtain representative samples. Markov Chain Monte Carlo methods, generally popular for their asymptotic guarantees [5] and strengths in high dimension [6], struggle in handling the multi-modality present in these problems. Furthermore, MCMC does not provide evidence estimation, which often is of crucial importance in scientific applications and elsewhere.

A popular family of methods to deploy instead is Nested Sampling [7, 1], known for performing well on multi-modal and degenerate posteriors, as well as additionally providing evidence estimation. Still, the computation required to run an experiment can often be impractical, such as weeks on a cluster. Re-sampling, density re-weighting and local density integration have been identified as important tasks to save computation, when considering different priors [3] or estimating equations of state [8].

Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada.

Currently these tasks, as well as parameter and evidence estimation, require a portfolio of different algorithms with various tuning parameters, requiring significant expertise and effort to obtain reliable results.

In this paper, we present an efficient approach to general Bayesian computation and inference based on recursive partitioning of the domain. The algorithm is designed for black-box settings and to handle all the discussed challenges. Our approach is, by design, asymptotically exact and has no sensitive free parameters; this leads to an algorithm with a robust performance for general density functions that is easy to use. Our method defines a sequential decision process sampling the integrand, prioritizing regions of high probability mass within a recursive refinement of the approximation, filling the sample space. The decision process has similar algorithmic efficiency as MCMC and nested sampling, with a time complexity allowing for sub-millisecond decision times and high scalability to a large number of observations to obtain sufficient precision. However, differently from these approaches, our method provides a *density function approximation* defined over the whole domain, that can be sampled and re-sampled at a very low constant cost per sample, or used to produce an approximation to expectations of any function with respect to the approximated density function. Naturally, this includes constant functions, and thus evidence can be estimated as well. We show that our approach can be turned into an algorithm that performs competitively compared with recent state-of-the-art methods while retaining the above benefits.

Full paper: https://arxiv.org/abs/2010.13632

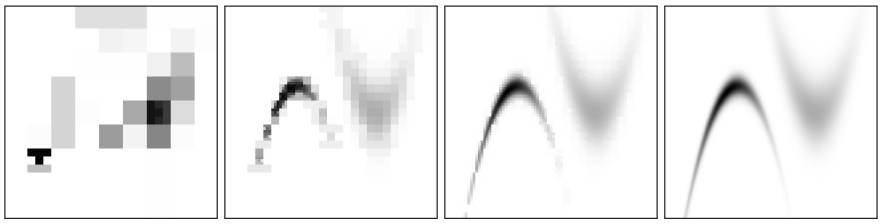


Figure 1: *Partitions produced using DEFER, shown after 53, 303, 2 101, and 100 003 density function evaluations, respectively.*

2 Methodology

Let $f : \Omega \to \mathbb{R}_+$ be an explicitly unknown, continuous density function defined on a bounded sample space $\Omega \subset \mathbb{R}^D$, which can be evaluated for any $\boldsymbol{\theta} \in \Omega$. Our goal is to construct a representation of the function that enables tractable performance of down-stream tasks such as estimation of its integral, expectations of functions, and constant time re-sampling with support over the full domain Ω. A density function implies a distribution

$$\mathcal{P}(\boldsymbol{\theta}) = \frac{f(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}\in\Omega} f(\boldsymbol{\theta}) d\boldsymbol{\theta}} = \frac{f(\boldsymbol{\theta})}{Z}, \tag{1}$$

where Z constitutes the unknown normalizing constant. If f is formed as the joint distribution of data and parameters $\boldsymbol{\theta}$ conditioned on known data $\mathcal{D}$, we refer to Z as the *evidence* and $\mathcal{P}(\boldsymbol{\theta})$ as the *posterior distribution*.

Representation of $\hat{f}$ Our approximation $\hat{f}$ will be represented by the combination of two data structures. Firstly, a non-overlapping partitioning $\Pi := \{\Omega_i\}$ of the domain Ω in an array, with an observed value of f at each respective centroid $\boldsymbol{\theta}_i$ of the corresponding partition Ω_i, that represents a Riemann sum over the domain with hyper-rectangular, non-overlapping partitions of non-constant side lengths as illustrated in Fig. 1. Secondly, a tree-structure in which these partitions are organized, forming a search tree over volumetric objects. Together, these permit integrals to be approximated by summation as in quadrature, density queries of $\boldsymbol{\theta}$ or partitions by tree search, and constant-time sampling. The latter is achieved by sampling the index of a partition, using the alias method for categorical distributions [9], followed by uniform sampling within the partition.

Quality of the approximation $\hat{f}$ The quality of the approximation over a partition in Ω_i is determined by the absolute error of mass. This minimizes errors in down-stream tasks such as obtaining

Algorithm 1: DEFER

Input: General density function f defined over Ω with unknown normalization constant Z

Output: Approximation $\hat{f}$, $\hat{Z}$, as specified by the produced partitioning Π_T

set t = 1 and initial partitioning $\Pi_1 = \{\Omega\}$;

while $N_t \leq N_{max}$ **do**

  $\{\Omega_i\} = \text{to\_divide}\,[\,\Pi_t]$;

  divide each partition Ω_i, each one resulting in $\{\Omega_j\}$ new partitions;

  add all sets of $\{\Omega_j\}$ into Π_t remove the divided partitions $\{\Omega_i\}$ from Π_t;

  set $t \to t + 1$ and update data structures;

end

return Π_T;

samples proportional to f, or estimating its integral. The same is true for expectations of functions under f where the functional form of the quantity is not specified ahead of time. In the recursive structure, this applies to each partition individually, as well as the domain as a whole.

Iterative construction requirements of $\hat{f}$ We now consider how to obtain a sufficiently close approximation to f in an efficient manner. We address this in three ways. Firstly, we prioritize where in the domain Ω the approximation should be refined. Secondly, we allow for a large number of partitions to obtain high resolution in areas of high mass. Lastly, we provide guarantees that the algorithm asymptotically approaches f. In a non-overlapping space partitioning, the first requirement translates into a decision-problem over which partitions to divide at a given step in sequence. The second requirement translates into making the decisions in an efficient manner, with an algorithmic complexity that allows for fast decisions that remain fast also for a large number of partitions. And lastly, for the asymptotic behaviour, guaranteeing that all partitions will eventually be divided. The outline of the algorithm is shown in Alg. 1. The partition division rule, algorithmic complexity analysis, and details of the procedure is found in the appendix A.

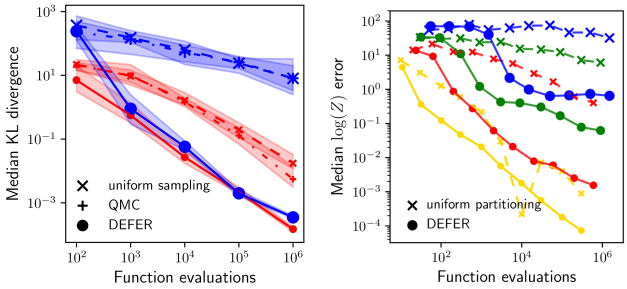


Figure 2: *Shown on the left is an illustrative comparison to using standard rejection sampling for parameter inference on a Gaussian distribution in 5D, with a true scale parameter of $1/20$ and $1/100$ of the domain sides, corresponding to small and large marker sizes, respectively. The rejection methods use proposals drawn uniformly or in a Sobol sequence (Quasi-Monte Carlo), respectively. Shown on the right is a comparison to using a uniform grid for evidence estimation on the Student's t-distribution in 2D, 4D, 6D, and 10D corresponding to the markers of increasing size. The true scale parameter corresponds to $1/100$ of the domain sides. Both experiments were run 20 times with uniformly sampled true means, and the Student's t-distribution has $2.5 + (D/2)$ degrees of freedom.*

3 Experiments

As discussed in Section 1, the focus of the paper is on obtaining an efficient algorithm to produce approximations to density functions with support over the whole sample space, which in turn can be used for a number of downstream tasks. However, it is crucial that the algorithm as a minimum is comparable to modern Bayesian methods available for individual tasks, such as posterior sampling and evidence estimation. By comparing with such methods, we can evaluate DEFER in terms of sample-efficiency, computational efficiency in practice, as well as confirm the quality of the resulting approximations.

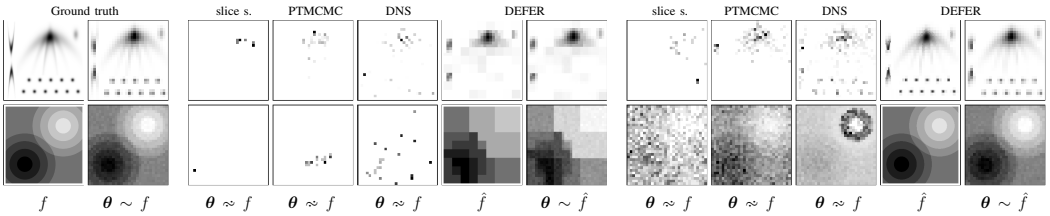


Figure 3: *Robustness to characteristics in the density surface. In the upper row, a surface with heavy correlations and multi-modality is shown, and in the lower row a surface with discontinuities and a valley. The output of the methods is shown after approx. 150 (left) and 1k (right) density evaluations for the first function, to illustrate the relative inefficiency of MCMC in capturing modes. The same is shown after approx. 30 and 100k evaluations for the second function, to illustrate the early coarse approximation by DEFER as well as its asymptotic guarantees in contrast to DNS. As different to the other methods, DEFER outputs a function $\hat{f}$.*

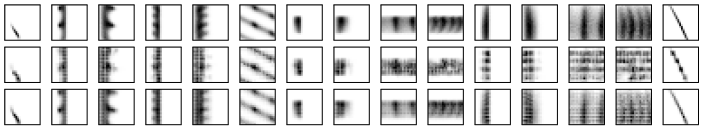


Figure 4: *Parameter estimation for gravitational-wave physics. Shown are histograms of the two-dimensional marginals of a 6D parameter inference problem [1]. Upper row: ground truth estimated by long-running nested sampling as in the paper. Second row: samples drawn from the partitioning after 200k evaluations. Bottom row: samples drawn from the partitioning after 2M evaluations.*

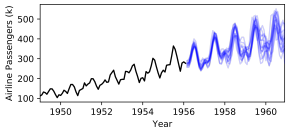


Figure 5: *Time-series forecasting using Gaussian Processes with the Spectral Mixture [10] kernel. The data is shown in black, and posterior GP samples in blue.*

Before we begin with the main comparisons, we highlight the problem with using simple methods such as standard rejection sampling for posterior sampling, and grid partitioning, which at first glance may seem practical to deploy to also construct function approximations. In typical, realistic scenarios, the likelihood function causes the typical set of the posterior to be concentrated to small regions. As illustrated in Fig. 2, uninformed proposal distributions become infeasibly inefficient already in low dimension (5D in the example) when the typical set of the posterior is even just moderately small. This is a consequence of the vast number of proposals needed on average per proposal landing in the typical set ($\approx 10^7$ in the example). In contrast, DEFER, using dynamic allocation, is significantly less affected by the size of the typical set. A similar example for evidence estimation is also shown.

We now proceed with the main comparisons. We choose dynamic nested sampling (DNS) [7], slice sampling [11], and parallel-tempering MCMC (PTMCMC) [12] as the baselines because they are easy-to-use or able to handle multi-modality and work in absence of gradient information. DNS is a state-of-the-art instantiation of nested sampling, known for performing well on multi-modal and degenerate posteriors with relatively little tuning. PTMCMC uses parallel-tempering to improve MCMC for multi-modality handling. Slice sampling is chosen for its ease-of-use, which is also a feature of DEFER. We used the following implementations: DNS [13], slice sampling [14], and PTMCMC [12]. For details, see Section D in the appendix.

Illustrative synthetic density functions We first test the robustness of the inference methods against challenging properties (see Fig. 3) including multi-modality, strong correlations, and discontinuities, on low dimensional examples that are easy to visualize. We illustrate the asymptotic guarantees of DEFER in the presence of such properties in the density surface. We also note its sample-efficiency in representing all modes when compared to MCMC, and in capturing detail when compared to both DNS and the MCMC methods. In one of the shown examples DNS fail to recover the ground truth, which may be related to the algorithm's determination of likelihood contours [7].

Real-world problems We continue with a problem from gravitional-wave (GW) physics. The goal of GW research is to understand events such as mergers of compact objects [2], for example binary black-holes [1]. As these events are rare and not repeatable, Bayesian statistics is widely

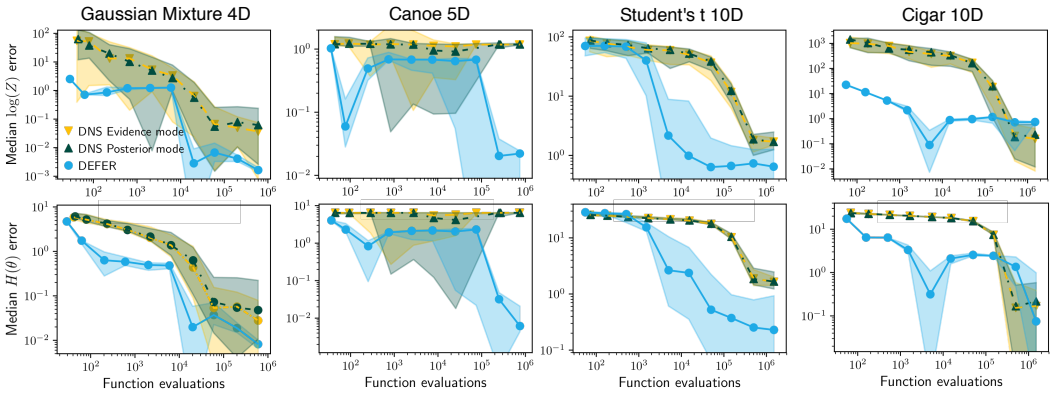


Figure 6: *Upper row: Median absolute error of* $\log Z$*. Lower row: Median absolute error of the entropy* $H(\boldsymbol{\theta})$*. For both metrics, shaded areas are 95 the % CI of the median.*

used for analysis. Models describing these events are expressed as physically motivated likelihood functions and priors [4], often without available gradients, and the density surfaces are typically complicated, multi-modal, or discontinuous. As discussed in Section 1, inference on these problems be prohibitively slow, warranting tasks such as re-sampling, density re-weighting and local density integration. DEFER outputs a density function approximation with support for all these tasks; the latter via making use of the domain-indexed search tree over partitions. We apply DEFER to a simulated signal example from [15] similar to [1]. Shown in Figure 4 are all the 2D marginals of a six-dimensional problem using the 'IMRPhenomPv2' waveform approximant. Inferred parameters are, for example, the luminosity distance, and the spin magnitudes of binary black-holes. We note the complicated interactions between parameters, showing the importance of handling multi-modality and strong correlations. Importantly, DEFER is able to handle the surface well without any tuning parameters. Full corner plots, and results using PTMCMC and DNS, are found in the appendix.

Apart from inferring a complex posterior surface, we also compare the quality of the posterior samples in terms of prediction accuracy. We consider a time-series forecasting problem because the posterior is often complex and multi-modal. We use a Gaussian process (GP) time-series regression model with a spectral mixture kernel (SMK) [10]. SMK can approximate any stationary kernel including periodic ones by learning the parameters of a Gaussian mixture to represent the kernel spectral density [16]. However, inference of the parameters of the mixture can be difficultfas the induced density surface is multi-modal. We use the airline passengers data [10] and use the first 60% of months for training and the rest for test. The GP model has three mixture components plus a linear slope, which translates into a 10D parameter inference task. With a budget of 50k function evaluations, the negative log-likelihoods on the test data are 377.66, 365.97, 236.89 and $\mathbf{205.50}$, for slice sampling, PTMCMC, DNS and DEFER, respectively. DEFER performs significantly better than other methods (see the Fig. 5 for the prediction by DEFER). Details and more plots of predictions using all methods are found in Section D.3 in the appendix.

Sample-efficiency and quantitative comparisons We use evidence and parameter estimation to give quantitative comparisons. For evidence estimation, we measure the median absolute error in log evidence (or $\log Z$) over 20 runs with various budgets. For parameter estimation, we measure the error in estimated differential entropy, due to lack of summary statistics for multi-modal distributions. Results for a few functions with various challenging characteristics are shown in Fig. 6, with more examples available in Section D.4 in the appendix. We remind the reader that although the surface of the approximation always tends towards the true surface, integral estimates can oscillate slightly as local overestimations and underestimations can cancel out to some degree. DEFER is able to match and typically significantly surpass the performance of DNS. Especially in getting close to the solution already at low N_T budgets, sometimes needing order of magnitudes fewer function evaluations, but it also performs better using the higher N_T budgets. The gaps between the methods are noticeably larger for estimating differential entropy. Estimation of entropy is more sensitive to the matching of the shape of a distribution, and not only its integral. On Canoe, DNS completely misses the concentrated mass, but DEFER finds it after 100k evaluations, which leads to a sharp drop in the error of both evidence and differential entropy.

4 Broader Impact

Machine learning methods are integral components for a fast-growing number of applications. For this development to continue and for machine learning methods to be applicable in critical scenarios, our methods need to be interpretable. Bayesian methods are designed with interpretability at its core through the principled formulation assumptions from which predictions can be interpreted. However, this comes with a substantial computational penalty as numerical methods are required to address computational intractabilities. Furthermore, the state-of-the-art numerical methods are challenging to use without expert knowledge of both the problem at hand and the numerical algorithm. This leads to significant challenges for users of machine learning methods to work on interpretable Bayesian models.

The method we have proposed in this paper allows for robust and efficient Bayesian computations. Our method provides similar or better performance compared to competing well-tuned approaches across several tasks, importantly without the need to set a large number of problem-specific parameters. We believe that by providing a robust, easy to use, and computationally efficient method for Bayesian computation we can widen the acceptance of these machine learning methods to larger audiences.

5 Acknowledgements

We thank Virginia D'Emilio, Rhys Green and Vivien Raymond (Cardiff University) for useful discussions relating to gravitational-wave physics, Ivan Ustyuzhaninov (University of Tübingen) for useful input relating to the representer points, and Ieva Kazlauskaite, Vidhi Lalchand and Pierre Thodoroff (University of Cambridge) for comments that greatly improved the manuscript. This project was supported by the Engineering and Physical Sciences Research Council (EPSRC) Doctoral Training Partnership, the Hans Werthén Fund at The Royal Swedish Academy of Engineering Sciences, EPSRC CAMERA project (EP/M023281/1), and the Royal Society.

References

[1] L. S. Collaboration, V. Collaboration, *et al.*, "Gw190412: Observation of a binary-black-hole coalescence with asymmetric masses," *arXiv preprint arXiv:2004.08342*, 2020.

[2] B. Abbott, R. Abbott, T. Abbott, S. Abraham, F. Acernese, K. Ackley, C. Adams, R. Adhikari, V. Adya, C. Affeldt, *et al.*, "Gwtc-1: A gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs," *Physical Review X*, vol. 9, no. 3, p. 031040, 2019.

[3] I. Mandel and T. Fragos, "An alternative interpretation of gw190412 as a binary black hole merger with a rapidly spinning secondary," *The Astrophysical Journal Letters*, vol. 895, no. 2, p. L28, 2020.

[4] C. Kalaghatgi, M. Hannam, and V. Raymond, "Parameter estimation with a spinning multimode waveform model," *Phys. Rev. D*, vol. 101, p. 103004, May 2020.

[5] C. J. Geyer, "Practical markov chain monte carlo," *Statistical science*, pp. 473–483, 1992.

[6] R. M. Neal *et al.*, "Mcmc using hamiltonian dynamics," *Handbook of markov chain monte carlo*, vol. 2, no. 11, p. 2, 2011.

[7] E. Higson, W. Handley, M. Hobson, and A. Lasenby, "Dynamic nested sampling: an improved algorithm for parameter estimation and evidence calculation," *Statistics and Computing*, vol. 29, no. 5, pp. 891–913, 2019.

[8] F. H. Vivanco, R. Smith, E. Thrane, P. D. Lasky, C. Talbot, and V. Raymond, "Measuring the neutron star equation of state with gravitational waves: The first forty binary neutron star merger observations," *Physical Review D*, vol. 100, no. 10, p. 103009, 2019.

[9] R. A. Kronmal and A. V. Peterson Jr, "On the alias method for generating random variables from a discrete distribution," *The American Statistician*, vol. 33, no. 4, pp. 214–218, 1979.

[10] A. Wilson and R. Adams, "Gaussian process kernels for pattern discovery and extrapolation," in *International conference on machine learning*, pp. 1067–1075, 2013.

[11] R. M. Neal, "Slice sampling," *Annals of statistics*, pp. 705–741, 2003.

[12] J. Ellis and R. van Haasteren, "jellis18/ptmcmcsampler: Official release," Oct. 2017.

[13] J. S. Speagle, "dynesty: a dynamic nested sampling package for estimating bayesian posteriors and evidences," *Monthly Notices of the Royal Astronomical Society*, vol. 493, no. 3, pp. 3132–3158, 2020.

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[15] G. Ashton, M. Hübner, P. D. Lasky, C. Talbot, K. Ackley, S. Biscoveanu, Q. Chu, A. Divakarla, P. J. Easter, B. Goncharov, *et al.*, "Bilby: A user-friendly bayesian inference library for gravitational-wave astronomy," *The Astrophysical Journal Supplement Series*, vol. 241, no. 2, p. 27, 2019.

[16] S. Bochner, *Lectures on Fourier integrals*, vol. 42. Princeton University Press, 1959.

[17] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the lipschitz constant," *Journal of optimization Theory and Applications*, vol. 79, no. 1, pp. 157–181, 1993.

[18] K. Tanaka and A. A. Toda, "Discrete approximations of continuous distributions by maximum entropy," *Economics Letters*, vol. 118, no. 3, pp. 445–450, 2013.

[19] K. Tanaka and A. A. Toda, "Discretizing distributions with exact moments: Error estimate and convergence analysis," *SIAM Journal on Numerical Analysis*, vol. 53, no. 5, pp. 2158–2177, 2015.

[20] L. E. Farmer and A. A. Toda, "Discretizing nonlinear, non-gaussian markov processes with exact conditional moments," *Quantitative Economics*, vol. 8, no. 2, pp. 651–683, 2017.

[21] N. Nguyen-Hong, H. Nguyen-Duc, and Y. Nakanishi, "Optimal sizing of energy storage devices in isolated wind-diesel systems considering load growth uncertainty," *IEEE Transactions on Industry Applications*, vol. 54, no. 3, pp. 1983–1991, 2018.

[22] X. Ai, J. Wen, T. Wu, and W.-J. Lee, "A discrete point estimate method for probabilistic load flow based on the measured data of wind power," *IEEE Transactions on Industry Applications*, vol. 49, no. 5, pp. 2244–2252, 2013.

[23] A. C. Miller III and T. R. Rice, "Discrete approximations of probability distributions," *Management science*, vol. 29, no. 3, pp. 352–362, 1983.

[24] E. A. DeVuyst and P. V. Preckel, "Gaussian cubature: A practitioner's guide," *Mathematical and Computer Modelling*, vol. 45, no. 7-8, pp. 787–794, 2007.

[25] D. J. Earl and M. W. Deem, "Parallel tempering: Theory, applications, and new perspectives," *Physical Chemistry Chemical Physics*, vol. 7, no. 23, pp. 3910–3916, 2005.

[26] C. J. Geyer, "Markov chain monte carlo maximum likelihood," 1991.

[27] E. Marinari and G. Parisi, "Simulated tempering: a new monte carlo scheme," *EPL (Europhysics Letters)*, vol. 19, no. 6, p. 451, 1992.

[28] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: a review for statisticians," *CoRR*, 2016.

[29] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," vol. 14, no. 1, pp. 1303–1347, 2013.

[30] R. Ranganath, S. Gerrish, and D. Blei, "Black Box Variational Inference," in *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics* (S. Kaski and J. Corander, eds.), vol. 33 of *Proceedings of Machine Learning Research*, (Reykjavik, Iceland), pp. 814–822, PMLR, 22–25 Apr 2014.

[31] S. A. Smolyak, "Quadrature and interpolation formulas for tensor products of certain classes of functions," vol. 148, pp. 1042–1045, 1963.

[32] T. Gerstner and M. Griebel, "Numerical integration using sparse grids," *Numerical Algorithms*, vol. 18, pp. 209–232, 1998.

[33] J. Hewitt and J. A. Hoeting, "Approximate bayesian inference via sparse grid quadrature evaluation for hierarchical models," *CoRR*, 2019.

[34] M. Osborne, R. Garnett, Z. Ghahramani, D. K. Duvenaud, S. J. Roberts, and C. E. Rasmussen, "Active learning of model evidence using bayesian quadrature," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 46–54, Curran Associates, Inc., 2012.

[35] C. Rasmussen and Z. Ghahramani, "Bayesian monte carlo," pp. 489–496, 01 2002.

[36] P. Hennig, M. A. Osborne, and M. Girolami, "Probabilistic numerics and uncertainty in computations," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 471, no. 2179, 2015.

[37] L. Acerbi, "Variational bayesian monte carlo," in *Advances in Neural Information Processing Systems*, pp. 8213–8223, 2018.

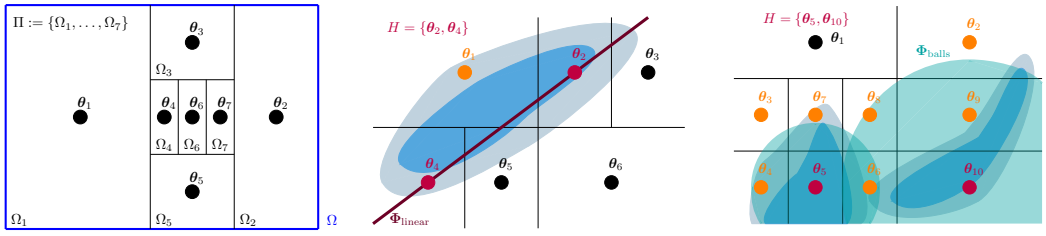
[38] J. W. J. Williams, "Algorithm 232: heapsort," *Commun. ACM*, vol. 7, pp. 347–348, 1964.

[39] R. Harman and V. Lacko, "On decompositional algorithms for uniform sampling from n-spheres and n-balls," *Journal of Multivariate Analysis*, vol. 101, no. 10, pp. 2297–2304, 2010.

[40] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Info. Pro. Lett.*, vol. 1, pp. 132–133, 1972.

[41] J. Veitch, V. Raymond, B. Farr, W. Farr, P. Graff, S. Vitale, B. Aylott, K. Blackburn, N. Christensen, M. Coughlin, *et al.*, "Parameter estimation for compact binaries with ground-based gravitational-wave observations using the lalinference software library," *Physical Review D*, vol. 91, no. 4, p. 042003, 2015.

A Algorithm



(a) *Partioning definitions* **(b)** *Locally linear structure* **(c)** *Local neighbourhoods*

Figure 7: *Illustration of partition division. (a) The domain* Ω *is divided according to a* partitioning Π *composed of partitions* $\{\Omega_i\}$ *with corresponding centroids* $\{\boldsymbol{\theta}_i\}$. *(b) Locally linear structure: High mass partitions* $H = \{\boldsymbol{\theta}_2, \boldsymbol{\theta}_4\}$ *define a linear subspace* $\boldsymbol{\Phi}_{\text{linear}}$ *used to find candidates for division (e.g.* $\boldsymbol{\theta}_1$*). (c) Local neighbourhoods: High mass partitions* $H = \{\boldsymbol{\theta}_5, \boldsymbol{\theta}_{10}\}$ *define neighbourhood D-balls* $\boldsymbol{\Phi}_{\text{balls}}$ *used to find candidates for division (e.g.* $\boldsymbol{\theta}_6, \boldsymbol{\theta}_9, \ldots$*).*

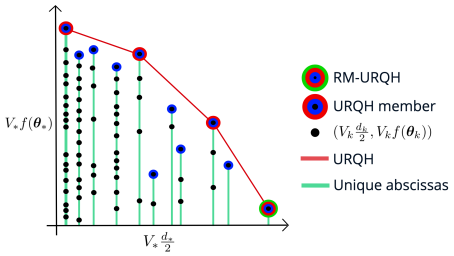
Given the representation and requirements, we construct the approximation through an iterative refinement procedure of a recursive tree data structure. The algorithm starts from the base case of the whole sample space being one partition, with its density evaluated at the centre. The outline of the algorithm is shown in Alg. 1. The approximation of f is parameterised using the data structures presented in Section 2. The set of current partitions at a given iteration t in the sequence of decisions is denoted by Π_t. We define $N_t := |\Pi_t|$ to be the number of partitions at iteration t. At each iteration, we divide partitions in Π_t that meet any of *three individually sufficient criteria* to produce an updated set of partitions Π_{t+1} (referred to as `to_divide` in Alg. 1). For details on the data structure updates and how a partition is divided, we refer to Section C.1.

We now detail the three individually sufficient criteria for division, that we denote CR1 to CR3. With the aim of robustness to multi-modality and degeneracies in f, to maintain an informed and efficient exploration, we will re-interpret [17] from the global optimization literature. To check our criterion for mass-based prioritization over the whole function, while avoiding explicit assumptions of the Lipschitz constant, we will map it into a format that allows similar sub-routines to be used for efficiently checking it. This first sufficient criterion (CR1) will also address the asymptotic guarantee by eventually being fulfilled for *all* partitions. Following it, we will address additional criterion that exploits structures of density functions, motivated for increased sample-efficiency in handling strong correlations.

CR1: Sufficient partition division criterion 1 A partition Ω_k will be divided if there exists *any* rate-of-change constant $\bar{K} > 0$ such that both

$$V_k \cdot \left(f(\boldsymbol{\theta}_k) + \bar{K}\frac{d_k}{2} \right) \geq V_i \cdot \left(f(\boldsymbol{\theta}_i) + \bar{K}\frac{d_i}{2} \right), \quad \forall\, i \in [1, N_t] \tag{2}$$

$$\text{and} \quad V_k \cdot \left(f(\boldsymbol{\theta}_k) + \bar{K}\frac{d_k}{2} \right) \geq \beta \frac{\hat{Z}}{N_t + 1}, \tag{3}$$



where $\hat{Z} = \sum_{k=1}^{N_t} \hat{Z}_k = \sum_{k=1}^{N_t} V_k \cdot f(\boldsymbol{\theta}_k)$ is the normalizing constant of the approximation, V_k is the volume of the partition Ω_k, $\boldsymbol{\theta}_k$ is the centroid of the partition, and d_k is the diameter of the partition $d_k = \sup_{\boldsymbol{\theta}_i, \boldsymbol{\theta}_j \in \Omega_k} ||\boldsymbol{\theta}_i - \boldsymbol{\theta}_j||$. Here, β is a positive parameter specifying what constitutes a non-trivial amount of *upper bound mass* of a partition in relation to the current estimate of average mass; we fix $\beta = 1$ for all the experiments in the paper. Note that β only has the role of controlling precision relative to the average partition mass contribution so far, and can in general be left at default.

The first statement (Eq. 2) is true if the partition of index k has an upper bound on its true mass greater or equal to the upper bound of all the partitions under *any* choices of $\bar{K} > 0$. The second statement (Eq. 3) is true if the partition, under that choice of $\bar{K}$, has an upper bound on the true mass that constitutes a non-trivial amount of mass.

We check the first statement using the following. After construction of a partition Ω_j, with parameters $(f(\boldsymbol{\theta}_j), \boldsymbol{\theta}_j \in \Omega_i)$, we map it to ordinate $V_j \cdot f(\boldsymbol{\theta}_j)$ and abscissa $V_j \cdot {}^{d_j}/_2$ in a 2D space, see illustration next to Eq. 2 and 3. This coordinate will be stored in a data structure for use at future iterations. The criterion will be fulfilled for a partition Ω_k *if and only if* its corresponding coordinate is a member of the upper-right quadrant of the convex hull (URQH) in this 2D space.

The second statement is checked by considering the *upper bound* of the $\bar{K}$ that a given partition Ω_k used to fulfil the first statement. This would be the $\bar{K}$ that, were it any larger, would result in the right hand neighbor on the URQH having a larger upper bound of the mass $V_{i+1} \cdot (f(\boldsymbol{\theta}_{i+1}) + \bar{K}\frac{d_{i+1}}{2}) > V_i \cdot (f(\boldsymbol{\theta}_{i+1}) + \bar{K}\frac{d_i}{2})$, which would violate the first criterion. In other words, we have $\bar{K}_i^{\text{upper}} := 2\frac{V_{i+1} f(\boldsymbol{\theta}_{i+1}) - V_i f(\boldsymbol{\theta}_i)}{V_i d_i - V_{i+1} d_{i+1}}$, except for the right-most member which will have positively infinite upper-bound on $\bar{K}$. As such, statement two is fulfilled if $V_i \cdot (f(\boldsymbol{\theta}_i) + \bar{K}_i^{\text{upper}}\frac{d_i}{2}) \geq \beta\frac{\hat{Z}}{N_t+1}$ is true.

Fulfilment of asymptotic guarantee As of Eq. 2 & 3, it is clear that the right-most member of URQH (RM-URQH) will always be divided as its upper bound is positively infinite. Furthermore, as the RM-URQH will always be among the partitions with the largest diameter, the largest diameter will asymptotically tend to zero. As a consequence, all partitions will eventually be divided, which in turn guarantees asymptotic convergence to f as of a partitioning of the whole domain constitutes a Riemann sum.

Checking these statements has an average time complexity of $\mathcal{O}(U \log U + \log N_t)$ where U is the number of *unique* abscissas. As will become clear, due to all the symmetries produced during the recursive partitioning of the space, the number of unique abscissas U will be very small in practice and virtually independent of N_t.

Exploiting characteristics of density functions We now add search behaviour to exploit desirable heuristics targeting both linear correlations and the clustering of mass (based on proximity in Ω); in contrast to just carrying out division when CR1 is true, these behaviours consider non axis-aligned directions.

Set of high mass partitions We define H as a set of high (relative) mass partitions at the current iteration. A partition Ω_j will be a member of this set when

$$\hat{Z}_j \geq \hat{Z}_M \quad \text{and} \quad \hat{Z}_j \geq \alpha\frac{\hat{Z}}{N_t + 1} \tag{4}$$

subject to the constraints that $1 < |H| \leq M$ and that their centroids are non-collinear. $\hat{Z}_M$ is defined to be the mass of the partition of the M^{th} highest mass, $\hat{Z}_* = V_* \cdot f(\boldsymbol{\theta}_*)$, and α is a positive parameter specifying what constitutes a *large* mass ratio relative to the average partition. In practice, we set $M = \min(5, D)$ and fix $\alpha = 20$ for all experiments in the paper.

Linear correlations Fig. 7(b) illustrates how we address linear correlations. Existing partitions with a high estimated mass, relative to other partitions, are grouped in the set H and denoted with red centroids. Were we to assume a linear correlation between the centroids in H, we may also assume that high mass is more likely to concentrate along an affine subspace defined by linear combinations of the centroids denoted Φ_{linear}, the maroon line in the figure. Partitions intersecting this hyper-plane are candidates for division.

CR2: Sufficient partition division criterion 2 Given the set H of high mass partitions, we construct a finite set $\mathbf{R}_{\text{linear}}$ of *representer points* based on the affine subspace constructed from linear combinations of the centroids of the partitions in H that we denote Φ_{linear}. Details on the representer points are provided in Section C.2 in the appendix. A partition Ω_k will be divided if any point $\mathbf{r} \in \mathbf{R}_{\text{linear}}$ falls within the partition. That is, the set $\{\mathbf{r} \mid \mathbf{r} \in \Omega_k, \mathbf{r} \in \mathbf{R}_{\text{linear}}\}$ is not empty.

Neighbourhood correlations We will now address the neighbourhoods of the H partitions, illustrated in Fig. 7(c). The figure shows that we consider spatial proximity to elements of H through sets of D-dimensional balls Φ_{balls} centred on elements of H. We define Φ_{balls} as the union of the interiors of D-dimensional balls centred at respective centroid of the partitions H. The diameter of a given D-ball corresponding to partition $\Omega_j \in H$ is $\phi\, d_h$, where ϕ is constrained to be $\phi > 1$ to guarantee that positions *outside* of Ω_j exists in all directions from $\boldsymbol{\theta}_j$. A given ϕ leads to a volume ratio ν of Ω_j relative to the ball as $\nu = {V_j \cdot \Gamma(D/2+1)}/{(\sqrt{\pi}\frac{\phi d_j}{2})^D}$. In practice we fix $\phi = 1.2$ in all experiments in the paper.

CR3: Sufficient partition division criterion 3 As for the linear case, we take the set H of high mass partitions and construct a discrete set $\mathbf{R}_{\text{balls}}$ of additional representer points based on spatial proximity to the centroids in H, denoted Φ_{balls}, and check the condition analogously to CR2.

The time complexity of our check of the CR2 criterion has time complexity $\mathcal{O}(MD^2 + \log N_t)$, and the CR3 criterion has time complexity $\mathcal{O}(MD + \log N_t)$. As D is constant with respect to N_t and M is upper bounded by D, we summarize this as time complexity of $\mathcal{O}(\log N_t)$.

Summary We have now addressed the criteria CR1 to CR3, describing the full algorithm Alg. 1. The combined time complexity of a step t is $\mathcal{O}(\log N_t + U \log U)$, which for an average number of steps proportional to N_T results in a total average time complexity of the algorithm as $\mathcal{O}(N_T(\log N_T + \tilde{U} \log \tilde{U}))$ where $\tilde{U}$ is the average number of unique abscissas. The total space complexity of the algorithm is linear, i.e. remains proportional to the number of observations. For the choice of representer points, complexity analysis derivations of all steps, and other algorithmic details, we refer to Section C. In Fig 8 we show that $\tilde{U}$ is sufficiently small, and close to constant with respect to N_T, resulting in a fast and scalable algorithm.

B Background

The need for and approach of substituting continuous distributions or processes with discrete approximations exists in various forms in literature, such as in optimisation problems in economics [18, 19, 20], or in engineering [21, 22]. These methods typically assume that the distribution is known, often along with requirements of its parametric form. Also, a good discrete set of sample points is assumed to be known, or that the problem is in one dimension [23], or that the discrete set is small [24]. This makes our method have more in common with popular families of Bayesian methods. However, differently from our method, these methods solve tasks in isolation, such as typically posterior sampling, but not evidence estimation, and vice versa.

The literature provides several families of methods for posterior sampling. On one end of the spectrum lie stochastic methods such as Markov Chain Monte Carlo (MCMC) that obtain samples via random walks. The benefit is that they provide asymptotic guarantees given that only a few weak criteria [5]

are met, and has shown to be effective in very high dimension where other strategies may struggle [6]. However, as by the definition of a Markov chain, MCMC is memory-less. The locally defined random walks are known to struggle in handling multi-modality efficiently, as of the low probability per step of leaving high-density regions. To address practical performance issues due to multi-modality, techniques such as parallel-tempering [25, 26, 27] using many random walks at different temperatures are often deployed. From a sample-efficiency stand-point, there are opportunities for improvement by instead making globally informed decisions about where to evaluate the density function. In addition, many of the used techniques have a large number of tuning-parameters that are hard to use in an efficient manner for practitioners without significant experience, or in automated settings.

On the other end of the spectrum are deterministic methods such as Variational Inference (VI) [28, 29] where an explicit approximation is formulated, leading to an optimisation problem. While variational methods can often be made efficient and require much less tuning compared to MCMC methods, they are, by construction, never exact and lack the guarantees of the stochastic approaches. In recent years several approaches have been proposed that relaxes the explicit form of the approximation to one implicitly represented using samples [30]. In either case, the optimisation problem of VI generally requires differentiability and known gradients, which is not met in our setting.

A classic approach to estimating an integral numerically is through quadrature. These methods partition the domain and convert the integral into a weighted summation of integrands at each partition. By creating a partitioning of the full domain, quadrature methods provide asymptotic guarantees by design. Furthermore, quadrature methods can be completely robust to the characteristics of the function surface, such as discontinuities and zero density regions, as it is the domain partitioning that drives the asymptotic behaviour. Importantly, this means that we can construct reliable algorithms that require little or no tuning. The downside of such an approach is practical efficiency, and as such naive quadrature rules are often used as an example to illustrate the curse of dimensionality.

A significant amount of work has been done to address the issues with the scalability of quadrature methods in higher dimensions. One approach is to use a sparse grid formulation [31, 32] that provides asymptotically the same coverage as dense grids. By introducing assumptions on the smoothness of the function, a significant reduction in the number of needed function evaluations can be achieved. In practice, these sparse grid formulations work well when the function is a low degree polynomial with low-order interaction terms [33]. Bayesian Quadrature (BQ) methods [34, 35] include a probabilistic model of the function. Making use of the function model, a sequential decision process balancing exploration and exploitation is used to evaluate the integrand where it is believed to be most informative of the unknown integral. With the "right" function model, these methods can lead to a significant reduction of the number of needed function evaluations.

Treatment of inference as a sequential decision-problem more generally is found in the literature of probabilistic numerics [36], where decisions are informed by observed quantities to estimate latent quantities, BQ being one example of a method in this realm. This beneficial strategy was used in a recent approach [37] where BQ and VI were combined to enable sample-efficient parameter estimation; the motivating application considered very expensive likelihoods where only a small number of density evaluations are available. The computational cost associated with updating the model means that these methods are mainly suitable for scenarios where the evaluation of the integrand is associated with a high cost. In this work, we will rely on a series of heuristics, which will approximate the expensive decision-oriented methods, with the goal of retaining sample-efficiency while maintaining a low cost of making decisions. We motivate this by the cost trade-off; at a relatively low cost of evaluating the function, spending some extra evaluations can be worthwhile if that can significantly reduce algorithmic cost. To put these costs in perspective, evaluating the density function in typical inference scenarios comes at a cost in the range of milliseconds. In practice, this means that we must aim for decision costs per evaluation that have a similar or lower cost. If this cost remains within that range even for millions of evaluations, then the algorithm will be applicable to many problems where sampling-based approaches like MCMC and nested sampling are popular.

C Algorithmic details

C.1 Data structures and associated updates

Search-tree To form the search-tree described in Section 3 of the paper, we store each partition Ω_i together with its associated density function observation $f(\boldsymbol{\theta}_i)$ in a node. When a partition is

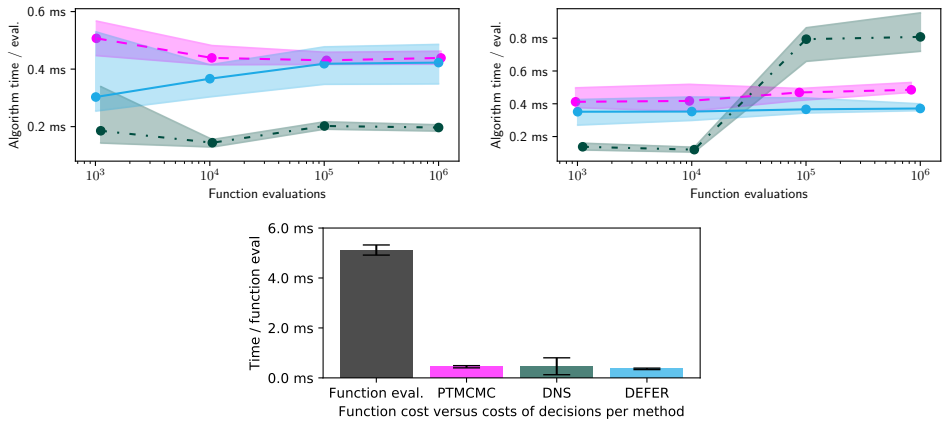


Figure 8: *Algorithmic time. Shown in the upper plots is the associated runtime cost of each algorithm to decide where to evaluate the density function, for the Student's t 10D density function and gravitational-wave example, respectively. Shown on in the lower plot is the time spent on evaluating the function versus the algorithm cost of respective method, with mean and standard deviation across 20 runs using the various settings of function evaluation budgets N_T.*

divided (see Algorithm 1), a set of child nodes are created and stored in an array within the parent node. Each partition (Ω_i) is represented using two arrays; with the lower and upper bounds for each dimension, respectively. From the limits, the associated centroid and volume of a partition can easily be calculated.

A query for the unique *leaf* node that is associated with a given $\boldsymbol{\theta}$ can be performed by traversing the tree from the root. If a called node (starting from the root) have children, only one of the children can (and must) be associated with $\boldsymbol{\theta}$, i.e. where $\boldsymbol{\theta} \in \Omega_j$. The correct child to call next is found using the limits of the partition corresponding to the respective child. The procedure is repeated recursively until the called node does not have any children; this means it is the leaf node that contains the queried $\boldsymbol{\theta}$. The leaf node is returned, holding the associated partition together with a density value.

Convex hull check associated with CR1 The ordinate and abscissa described under CR1 in Section 4 is stored in a hash map of heaps, i.e. where each entry of a hash table maps to a heap [38] as its associated value. This data structure map is used to efficiently provide access to the *maximum* ordinate partition per unique abscissa at every iteration t, which are kept sorted using the individual heaps. These partitions are the only ones that have corresponding coordinates that *may* be part of the URQH described, and the only ones that need to be a part of a convex hull calculation. The number of unique abscissas is very small in relation to the total number of partitions due to all the symmetries produced by the symmetric divide operation (see "Divide operation"); and remains near-constant for higher numbers of partitions as a result of the algorithm. In practice in experiments, we noted that the unique number of abscissa were rarely larger than 20 even after millions of partitions in any of the up to ten-dimensional problems tested. This near constant behaviour will be evidenced by the experiments in the "Runtime performance" section.

The hash function (forming the hash table key) is a floating-point hash on the abscissa. Each time a partition is added (see Algorithm 1. in the paper) the associated coordinate needs to be added to this data structure, and each time one is removed, it will also need to be removed from here. Note that this data structure only keeps the coordinates of *leaf* nodes, i.e. nodes corresponding to the current set of all partitions constituting the Riemann sum described in Section 3.

Current total mass and top M mass partitions At every step of the algorithm, we keep track of the total mass estimate and the top M mass partitions, as required by the algorithm.

We implement this by aggregators which are updated at each *include* and *exclude* of a partition in the *leaf set*. The leaf set is the partitions that cover the domain in a non-overlapping fashion at the currently finest resolution, constituting the current Riemann sum. "Include" refers to the creation of a new child node (and partition), and "exclude" to the removal of the previous (parent) node from this

set. To easily keep track of all the updates a partition division should lead to, we wrap the "divide function" in another function that after division (the forming of child partitions), makes all the related updates, including the exclusion of the divided parent node.

The aggregator for the total mass keeps track of the accumulate sum (which we represent using a 128-bit float) where inclusion is addition and exclusion is subtraction. The aggregator for the top M partitions keeps track of such a current set of size up to M, where set updates are handled within the inclusion and exclusion functions.

C.2 Representer points

In Section 4 associated with CR2 and CR3 there are representer points to be chosen. These are used in the algorithm to efficiently (and approximately) check for overlaps between respective partitions and the two spaces $\boldsymbol{\Phi}_{\text{linear}}$ and $\boldsymbol{\Phi}_{\text{balls}}$, associated with CR2 and CR3 respectively. For each representer point, a tree-search is performed (see Search-tree) to find the associated partition, which will be divided. If a partition fulfils multiple sufficient criteria (CR1 to CR3), or in this case is "hit" by multiple representer points, it will still only be divided once.

CR2 We begin with the determination of $\boldsymbol{\Phi}_{\text{linear}}$ followed by $\boldsymbol{R}_{\text{linear}}$. To form the linear subspace we carry out the following steps. Obtain the corresponding centroids Θ_H of the H partitions as stacked vectors. Θ_H will now be used to form a basis for the $(H-1)$-dimensional hyperplane being $\boldsymbol{\Phi}_{\text{linear}}$. Let $\boldsymbol{\theta}_j$ be the centroid (column vector) among them closest to the center of the unit cube, and $E = \Theta_{H,\neq j} - \boldsymbol{\theta}_j$ be the stacked basis vectors of the hyperplane of which we form an orthogonal basis A. The orthogonal matrix A is determined from E using QR factorization (in Python available through 'numpy.linalg.qr'). Let $\tilde{E}$ be the re-scaled version of E having unit-norm. Now we can map a vector u from a unit-cube onto the hyper-plane as $\boldsymbol{\theta}_j + uA\tilde{E}$.

From $\boldsymbol{\Phi}_{\text{linear}}$ a discrete set of points $\boldsymbol{R}_{\text{linear}}$ is now to be chosen. We will concentrate the points in particular on all lower-dimensional hyper-planes formed by all combinations of the H high mass partitions, which in line with the assumed heuristic that mass tends to concentrate on linear subspaces formed by these partitions. In practice this is implemented by ahead of time determining all $\sum_{s=1}^{S} \binom{M}{s+1}$ combinations of indices up to M where s is the dimensionality of a lower-dimensional hyperplane. These index combinations are then iterated through at runtime at step t, each one creating a subset H_* of high mass partitions forming a linear subspace; if a given set of centroids is colinear, it is skipped. For each H_* we add a representer point at their average location in Ω (the weighted centre of the simplex they form). In addition, to cheaply emulate 'exploration' of the linear subspaces (and the constituent lower-dimensional linear subspaces), we add l representer points per H_* by sampling uniformly in a unit cube and map the points onto the corresponding linear subspace using the procedure described above (we use $l = 1$ in all experiments).

CR3 We now address the choice of the discrete set of points $\boldsymbol{R}_{\text{balls}}$ from $\boldsymbol{\Phi}_{\text{balls}}$. For each partition in H, we sample b points uniformly within the corresponding D-ball using the Muller method [39]. In practice we set $b = D$ in all experiments.

C.3 Algorithmic complexity analysis

For simplicity of analysis, we will assume that the number of iterations of the algorithm (Algorithm 1) is proportional to N_T. It will always be true that the number of iterations *is less than* N_T, as multiple partitions will be divided at each iteration, and also each partition division will yield multiple function evaluations.

CR1 Updating the hash map of heaps for a new partition has time complexity $\mathcal{O}(\log \frac{N_t}{U})$, where U is the number of unique abscissas, as the heap (found in constant time using the hash map) provides logarithmic time updates and N_t/U is the average number of elements in the heap. As $N_t > U$ this simplifies as the worst-case $\mathcal{O}(\log \frac{N_t}{U}) = \mathcal{O}(\log N_t - \log U) = \mathcal{O}(\log N_t)$.

Retrieving all top ordinate partitions elements from this structure at a given iteration t has time complexity $\mathcal{O}(U)$ as the hash table with U entries is traversed linearly and the root of each heap is available in constant time. Using the U obtained coordinates the convex hull is computed in

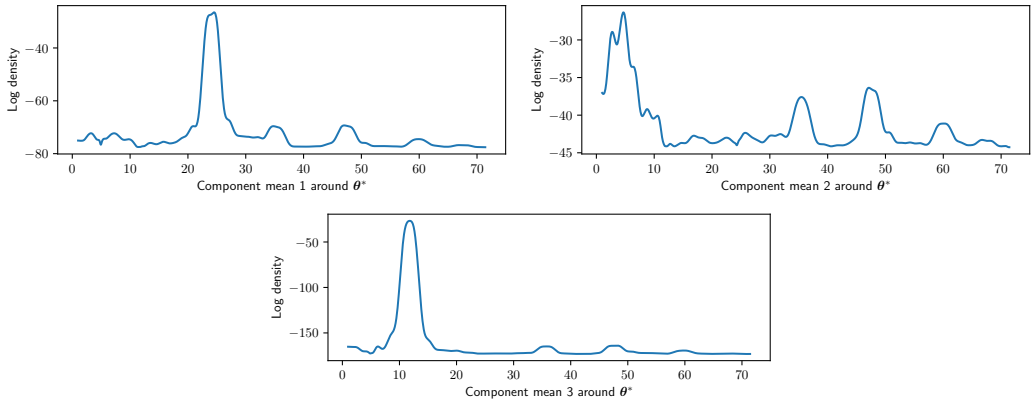


Figure 9: *Multi-modality in the log density surface over the parameters of the Spectral Mixture kernel. Shown is how the log density of the model changes with respect to each component mean within the Gaussian mixture representing the spectral density of the kernel. The slicing is centred around the (estimated) mode of the parameter posterior. The estimation of the posterior mode was done using DEFER.*

$\mathcal{O}(U \log U)$ using Graham's scan [40]. Following this, the upper-right quadrant of the hull is obtained in $\mathcal{O}(U)$.

In summary we obtain the worst-case time complexity $\mathcal{O}(\log N_t + U \log U)$ for checking this criteria for all partitions at step t. The space complexity of the hash map of heaps data structure is linear with respect to N_t.

CR2 The checking of this criterion entails computation associated with a potentially large constant (wrt. N_t) number $\sum_{s=1}^{S} \binom{M}{s+1}$ of lower-dimensional linear subspaces. The number of partitions that will be divided as a result grows proportionally to this constant, so we simplify the analysis to treating one linear subspace and an associated constant set of representer points. The two largest terms come from the QR factorization, with time complexity $\mathcal{O}(M^2 D)$, and the tree-search for the partition of a representer point with time complexity $\mathcal{O}(\log N_t)$. The space complexity is $\mathcal{O}(M)$ of keeping track of the high mass partitions. As D is constant with respect to N_t and M is upper bounded by D, we summarize this as time complexity of $\mathcal{O}(\log N_t)$ and the space complexity is constant.

CR3 For the check of this criterion all up to M high mass partitions are sampled in $\mathcal{O}(M D)$ time, and the tree-search for a partition is $\mathcal{O}(\log N_t)$. Analogous to CR2 we summarize this as a time complexity $\mathcal{O}(\log N_t)$ and the space complexity is constant.

Total In total we obtain the time complexity $\mathcal{O}(\log N_t + U \log U)$ of a step t and space complexity $\mathcal{O}(N_t)$. For an average number of steps proportional to N_T this results in a total average time complexity of the algorithm as $\mathcal{O}(N_T(\log N_T + \tilde{U} \log \tilde{U}))$ where $\tilde{U}$ is the average number of unique abscissas. The total space complexity of the algorithm is linear, i.e. remains proportional to the number of observations.

C.4 Divide operation

For simplicity and adequacy, we use the same partition division routine as in [17], which we refer to for further details. It entails a sequential dimension-wise division into three sub-partitions with equal side-length for each divided dimension. The set of dimensions to divide is the set of dimensions of maximum length for the partition. The divide order of the dimensions is determined by the direction of the highest observed function value.

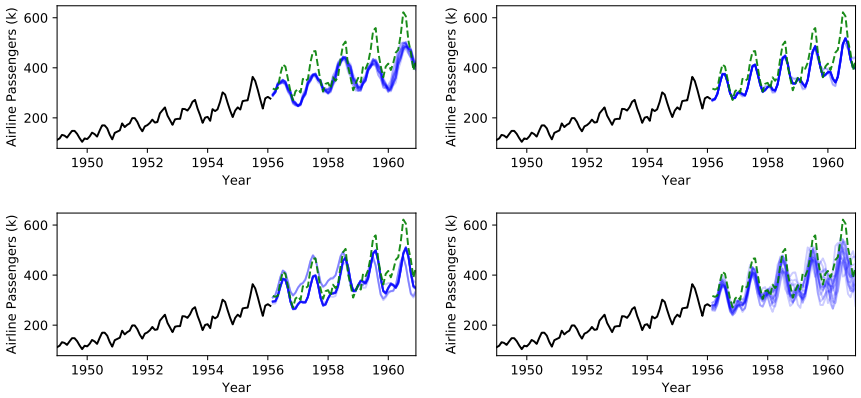


Figure 10: *Time-series forecasting using Gaussian Processes with the Spectral Mixture [10] kernel. The data is shown in black, ground truth in green and posterior GP samples in blue. Shown in the top is slice sampling, followed by PTMCMC, DNS and lastly DEFER in the bottom.*

D Experiments

D.1 Setup and baselines

We used the following implementations: DNS [13], slice sampling [14], and PTMCMC [12].

Slice sampling We use an initial step size corresponding to 0.05 of the domain sides, with max_doublings set to 5. The used burn-in ratio is 25%.

PTMCMC We follow the default settings and sample p_0 uniformly within the domain and set the initial covariance matrix to be diagonal with variance 0.01. We use covUpdate=500, 25% burn-in and all parameters set to the default.

DNS We use the default (as all other settings) of nlive_init=500, in practice nlive_init=min(500, 2 + int(N_T / 10)) as the max number of function evaluations N_T in a few experiments is small. For 'Posterior mode' pfrac = 1.0, and for 'Evidence mode' pfrac = 0.0.

D.2 Runtime performance

The DNS and DEFER were run (single-threaded) on an Intel Core i7, and PTMCMC used multiple cores due to its implementation. In Figure 8 we note both that DEFER has a similar algorithmic cost to the other methods and that the DEFER has a near constant cost per function evaluation with respect to N_T. We also illustrate the common situation where the function evaluation time dominates the decision time, making sample-efficiency the critical concern for total efficiency.

D.3 Time-series prediction example

As discussed in the experiments section of the paper, the Spectral Mixture kernel is known to be challenging to optimize. This we suggest has much to do with the density surface of the model induced by the parameters of the kernel. In Figure 9, we show the heavy multi-modality present in the landscape. Such structures may cause local optimizers and step-wise sampling techniques like MCMC to get stuck in poor solutions.

In Figure 10 the predictions for respective method are shown.

D.4 All quantitative comparison plots

In Figure 11 and Figure 12 some additional density function experiments are included. Uniform partitioning is included for reference as well.

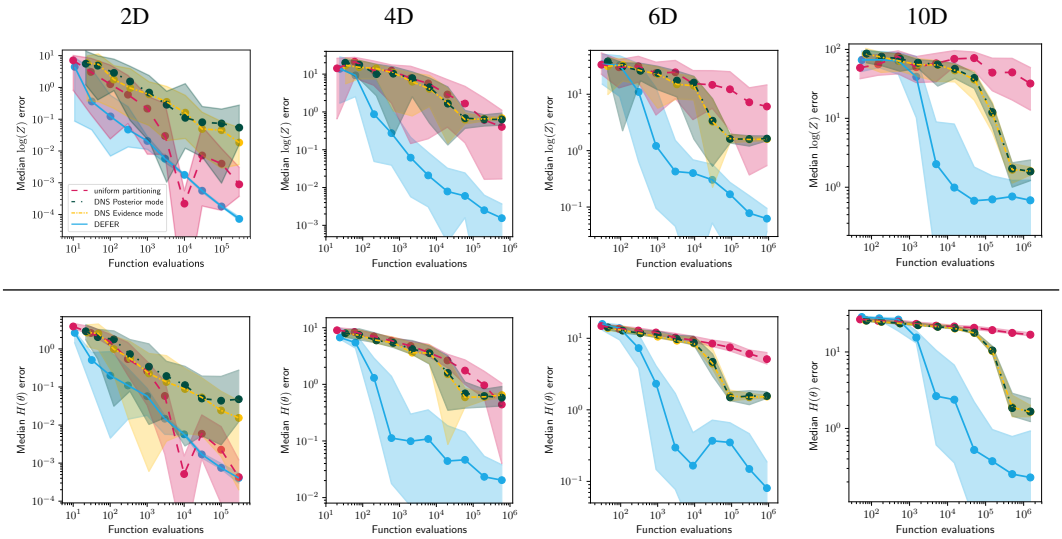


Figure 11: *Student's t-distribution in various dimensionalities. Upper row: Median absolute error of* $\log Z$*. Lower row: Median absolute error of the entropy* $H(\boldsymbol{\theta})$*. For both metrics, shaded areas are 95 % CI of the median.*

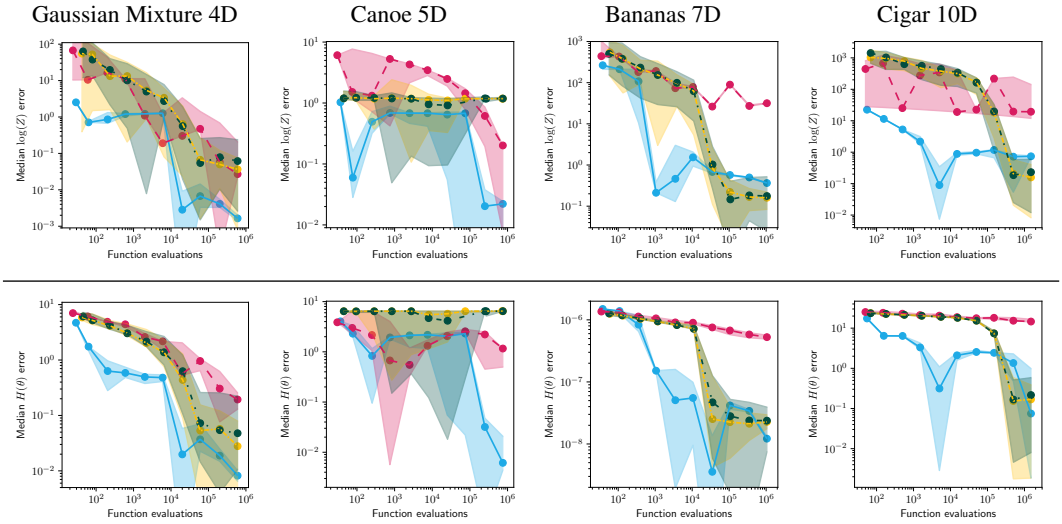


Figure 12: *Upper row: Median absolute error of* $\log Z$*. Lower row: Median absolute error of the entropy* $H(\boldsymbol{\theta})$*. For both metrics, shaded areas are 95 % CI of the median. For legend see Fig 11.*

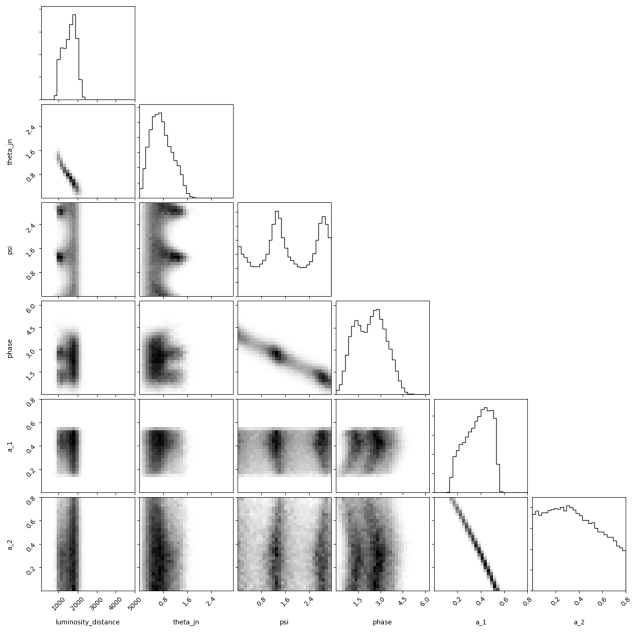


Figure 13: *PTMCMC after 2M density function evaluations.*

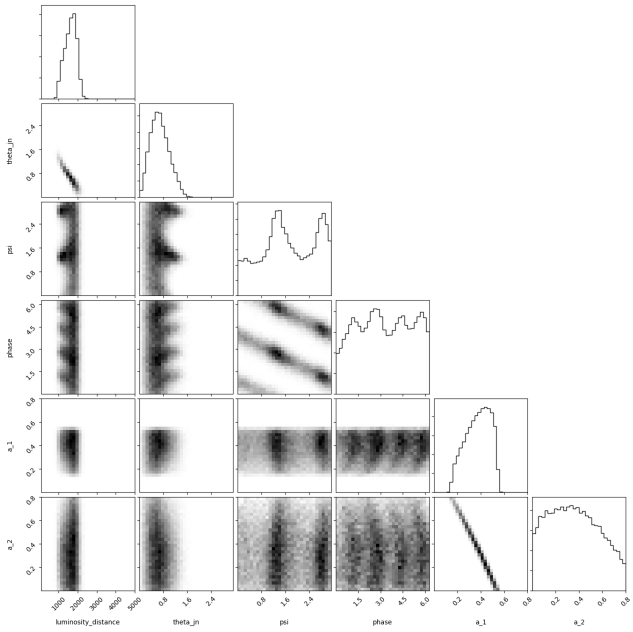


Figure 14: *DNS after 2M density function evaluations.*

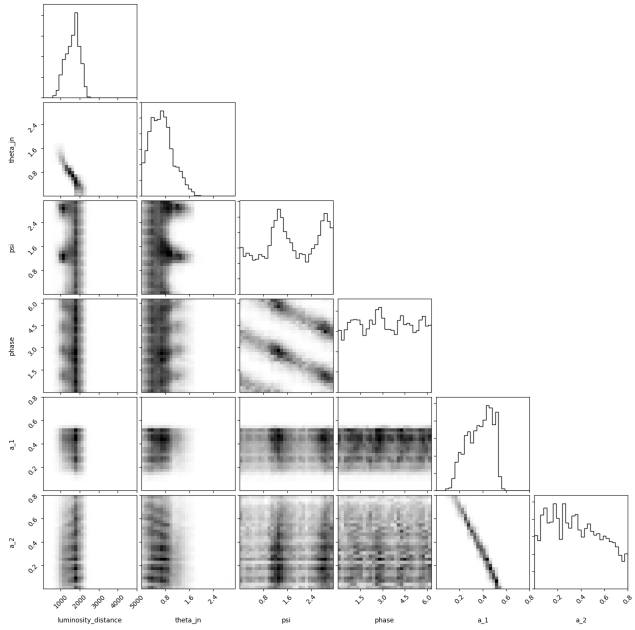


Figure 15: *DEFER after 2M density function evaluations.*

D.4.1 Gravitional-wave physics problem

Injected parameter 4D example. This 4D example was turned into a 6D example with the injection of all parameters except the following six parameters:

- The luminosity distance to the source dL.
- Phase: The orbital phase of the binary at the reference time.
- Theta JN: The inclination of the system's total angular momentum with respect to the line of sight.
- Psi: The polarisation angle describes the orientation of the projection of the binary's orbital momentum vector onto the plane on the sky.
- Dimensionless spin magnitude a_1.
- Dimensionless spin magnitude a_2.

For more details, see [41]. To produce a deterministic surface (by fixing the global seed used by the waveform approximant) while not fixing the seed for the third-party stochastic methods, we maintain the pseudo-random generator used within and outside the function, e.g. "inside_state = np.random.get_state(); np.random.set_state(outside_state)".

For corner plots, see Figure 13 to 15.

D.4.2 Spectral Mixture kernel problem code

PyTorch Spectral Mixture kernel GP example. This example was adapted for the Airline Passengers dataset used in [10].

D.5 Code

An Python implementation of the algorithm will be made available as part of an open-source library. The code for the density functions are found in Figure 16 to 20.

```
from scipy.stats import multivariate_normal
import tensorflow_probability as tfp
import tensorflow as tf
tfd = tfp.distributions

def load_students_t(ndims, loc):
    # loc is uniformly sampled each run within the unit space
    assert len(loc) == ndims
    D = ndims
    theta_space = unit_space(ndims=D, dtype=np.float64)
    assert is_point_within(space=theta_space, loc)

    df = 2.5 + (D / 2)
    scale = 0.01 * np.ones([D])

    dist = tfd.StudentT(
        df=df,
        loc=loc,
        scale=scale
    )

    @tf.function(
    input_signature=(tf.TensorSpec(shape=D, dtype=np.float64))
    )
    def tf_f(x):
        dist_inside = tfd.StudentT(
            df=df,
            loc=loc,
            scale=scale
        )
        return tf.reduce_sum(dist_inside.log_prob(x))

    def log_density_func(x):
        return tf_f(x).numpy()

    return log_density_func
```

Figure 16: *Student's t density function code*

```
def canoe(x, D):
    mean = 0.5 * np.ones([D])
    inner_c = .95
    inner_cov =
    inner_c * np.ones([D, D]) + \
            (1 - inner_c) * np.eye(D)
    inner_cov *= 0.01

    outer_c = .6
    outer_cov =
    outer_c * np.ones([D, D]) + \
            (1 - outer_c) * np.eye(D)
    outer_cov *= 0.02

    # Base
    base = 1

    # Bump
    log_outer = multivariate_normal.logpdf(
        x,
        mean=mean,
        cov=outer_cov
    )
    outer = np.exp(np.float128(log_outer))

    # Dip
    log_inner = multivariate_normal.logpdf(
        x,
        mean=mean,
        cov = inner_cov
    )
    inner = np.exp(np.float128(log_inner))

    a = 20
    b = 5
    v_unconstrained = \
            2 * base - a * outer + b * inner
    v = np.maximum(v_unconstrained, 0)
    return np.float64(np.log(v))
```

Figure 17: *Canoe density function code*

```
def mixture_of_gaussians(x):

    width_a = 0.010
    width_b = 0.015

    def base_covariance(width):
        covar = np.diag([(1.5 * width) ** 2] * D)
        covar[0, 1] = - (1 * width) ** 2
        covar[1, 0] = - (1 * width) ** 2
        return covar

    covar_a = base_covariance(width_a)
    covar_b = base_covariance(width_b)

    correlation_width = np.minimum(
        width_a, width_b
    )

    covar_b[0, 2] = + correlation_width ** 2
    covar_b[2, 0] = + correlation_width ** 2

    covar_b[0, 3] = - correlation_width ** 2
    covar_b[3, 0] = - correlation_width ** 2

    a_log = multivariate_normal.logpdf(
        x,
        mean=[ # Pre-sampled uniformly
            0.63263116, 0.74013062,
            0.72316873, 0.24708191
        ],
        cov=covar_a
    )
    b_log = multivariate_normal.logpdf(
        x,
        mean=[ # Pre-sampled uniformly
            0.51387785, 0.46667482,
            0.37765008, 0.79952093
        ],
        cov=covar_b
    )
    a = np.exp(np.float128(a_log))
    b = np.exp(np.float128(b_log))
    v = np.log(2.5 * a + b)
    return v.astype(np.float64)
```

Figure 18: *Mixture of Gaussians density function code*

```
def cigar(x, D):
    mean = 0.5 * np.ones([D])
    c = 0.99
    cov = c * np.ones([D, D]) + (1 - c) * np.eye(D)
    cov *= 0.01
    return multivariate_normal.logpdf(
        x,
        mean=mean,
        cov=cov
    )
```

Figure 19: *Cigar density function code*

```
def load_bananas(ndims):
    assert 2 <= ndims <= 30

    # Pre-sampled uniformly
    available_means = np.array(
    [ 1.82894125,  3.65177327, -1.43438539, -3.26737271,  0.11502106,
    1.11453084, -1.30059343,  3.13096048,  1.68656209,  0.51719625,
    -0.21618269,  0.73703613, -2.26608059,  0.59231536, -3.04444646,
    -1.29475342, -0.70038633, -0.20978783, -1.45007999,  2.99177333,
    -3.3860525 , -1.45444106,  1.50009961,  2.02561428, -3.7998913 ,
    -0.29527712, -2.36989081, -1.34796447,  1.69551565,  2.1853957 ])

    def _forward(x):
        y_0 = x[..., 0:1]
        y_1 = x[..., 1:2] - y_0 ** 2 - 1
        y_tail = x[..., 2:-1]
        return tf.concat([y_0, y_1, y_tail], axis=-1)

    def _inverse(y):
        x_0 = y[..., 0:1]
        x_1 = y[..., 1:2] + x_0 ** 2 + 1
        x_tail = y[..., 2:-1]
        return tf.concat([x_0, x_1, x_tail], axis=-1)

    def _inverse_log_det_jacobian(y):
        return tf.zeros(shape=())

    theta_space = space_from_sequence(
        [(-5, 5)] * ndims,
        dtype=np.float64
    )

    dtype = tf.float64

    @tf.function(
        input_signature=(
            tf.TensorSpec(shape=[2], dtype=dtype),
            tf.TensorSpec(shape=[2], dtype=dtype),
        )
    )
    def tf_f_split(x_a, x_b):
        Sigma_a = np.float32(np.eye(N=2)) + 0.3 * np.eye(N=2)[::-1]
        Sigma_b = np.float32(np.eye(N=2)) + 0.95 * np.eye(N=2)[::-1]
        p_x_a = tfp.distributions.MultivariateNormalTriL(
            scale_tril=tf.linalg.cholesky(Sigma_a)
        )
        p_x_b = tfp.distributions.MultivariateNormalTriL(
            scale_tril=tf.linalg.cholesky(Sigma_b)
        )
        banana = tfp.bijectors.Inline(
            forward_fn=_forward,
            inverse_fn=_inverse,
            inverse_log_det_jacobian_fn=
            _inverse_log_det_jacobian,
            inverse_min_event_ndims=1,
            is_constant_jacobian=True,
        )
        p_y_a = tfd.TransformedDistribution(
            distribution=p_x_a, bijector=banana)
        p_y_b = tfd.TransformedDistribution(
            distribution=p_x_b, bijector=banana)
        return p_y_a.prob(x_a) + p_y_b.prob(x_b)

    def tf_f(x):
        x_a = (x + tf.constant([-2,  1], dtype=dtype)) * \
              tf.constant([1, -1], dtype=dtype)
        x_b = (x + tf.constant([2, -2], dtype=dtype)) * \
              tf.constant([1, 1], dtype=dtype)
        return tf.math.log(tf_f_split(x_a, x_b))

    has_extra_dims = ndims > 2

    if has_extra_dims:
        extra_indices = np.arange(2, ndims)
        extra_means = available_means[:len(extra_indices)]

    def log_density_func(x):
        indices = [0, 1]
        bananas_log_f = tf_f(x[indices]).numpy()

        log_f = bananas_log_f
        if has_extra_dims:
            gaussian_log_f = multivariate_normal.logpdf(
            x[extra_indices],
            mean=extra_means,
            cov=0.01 * np.eye(len(extra_indices))
            )
            log_f += gaussian_log_f

        return log_f

    return log_density_func
```

Figure 20: *Bananas density function code*