The Journal of Open Source Software

lintsampler: Easy random sampling via linear interpolation

Aneesh P. Naik [1¶] **and Michael S. Petersen** [1]

1 Institute for Astronomy, University of Edinburgh, UK ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

`lintsampler` provides a Python implementation of a technique we term 'linear interpolant sampling': an algorithm to efficiently draw pseudo-random samples from an arbitrary probability density function (PDF). First, the PDF is evaluated on a grid-like structure. Then, it is assumed that the PDF can be approximated between grid vertices by the (multidimensional) linear interpolant. With this assumption, random samples can be efficiently drawn via inverse transform sampling (Devroye, 1986).

`lintsampler` is primarily written with `numpy` (Harris et al., 2020), drawing some additional functionality from `scipy` (Virtanen et al., 2020). Under the most basic usage of `lintsampler`, the user provides a Python function defining the target PDF and some parameters describing a grid-like structure to the `LintSampler` class, and is then able to draw samples via the `sample` method. Additionally, there is functionality for the user to set the random seed, employ quasi-Monte Carlo sampling, or sample within a premade grid (`DensityGrid`) or tree (`DensityTree`) structure.

Statement of need

For a small number of well-studied PDFs, optimised algorithms exist to draw samples cheaply. However, one often wishes to draw samples from an arbitrary PDF for which no such algorithm is available. In such situations, the method of choice is typically some flavour of Markov chain Monte Carlo (MCMC), a powerful class of methods with many excellent Python implementations (Coullon & Nemeth, 2022; Fonnesbeck et al., 2015; Foreman-Mackey et al., 2019; Marignier, 2023). One drawback of MCMC techniques is that they typically require a degree of tuning during the setup (e.g. choice of proposal distribution, initial walker positions, etc.), and a degree of inspection afterward to check for convergence. This additional work is a price worth paying for many use cases, but can feel excessive in scenarios where the user is less concerned with strict sampling accuracy or minimising PDF evaluations, and would prefer a simpler means to generate an approximate sample.

`lintsampler` was designed with such situations in mind. In the simplest use case, the user need only provide a Python function defining the target PDF and some one-dimensional arrays representing a grid, and a set of samples will be generated. Compared with MCMC, there is rather less work involved on the part of the user, but there compensating disadvantages. First, some care needs to be taken to ensure the grid has sufficient resolution for the use case. Second, in high dimensional scenarios with finely resolved grids, the PDF might well be evaluated many more times than with MCMC.

We anticipate `lintsampler` finding use in many applications in scientific research and other areas underpinned by statistics. In such fields, pseudo-random sampling fulfils a myriad of purposes, such as Monte Carlo integration, Bayesian inference, or the generation of initial

conditions for numerical simulations. The linear interpolant sampling algorithm underpinning `lintsampler` is a simple and effective alternative to existing techniques, and has no publicly available implementation at present.

Usage

`lintsampler` is designed with a interface that makes sampling from an input PDF straightforward. For example, if you have PDF with multiple separated peaks:

```
import numpy as np
from scipy.stats import norm

def gmm_pdf(x):
    mu = np.array([-3.0, 0.5, 2.5])
    sig = np.array([1.0, 0.25, 0.75])
    w = np.array([0.4, 0.25, 0.35])
    return np.sum([w[i] * norm.pdf(x, mu[i], sig[i]) for i in range(3)], axis=0)
```

`lintsampler` can efficiently draw samples from it on some defined interval:

```
from lintsampler import LintSampler

grid = np.linspace(-7,7,100)
samples = LintSampler(grid,pdf=gmm_pdf).sample(N=10000)
```

`samples` is then an array of 10000 samples drawn from the PDF. Apart from defining the PDF, `lintsampler` enables creating discrete samples from a continuous PDF in a small handful of lines.

Features

Although `lintsampler` is written in pure Python, making the code highly readable, the methods make extensive use of `numpy` functionality to provide rapid sampling. After the structure spanning the domain has been constructed, sampling proceeds with computational effort scaling linearly with number of sample points.

We provide two methods to define the domain, both optimised with `numpy` functionality for efficient construction. The `DensityGrid` class takes highly flexible inputs for defining a grid. In particular, the grid need not be evenly spaced (or even continuous) in any dimension; the user can preferentially place grid elements near high-density regions. The `DensityTree` class takes error tolerance parameters and constructs an adaptive structure to achieve the specified tolerance. We also provide a base class (`DensityStructure`) such that the user could extend the methods for spanning the domain.

Documentation for `lintsampler`, including example notebooks demonstrating a range of problems, is available via a readthedocs page. The documentation also has an extensive explanation of the interfaces, including optimisation parameters for increasing the efficiency in sampling.

Acknowledgements

We would like to thank Sergey Koposov for useful discussions. APN acknowledges funding support from an Early Career Fellowship from the Leverhulme Trust. MSP acknowledges funding support from a UKRI Stephen Hawking Fellowship.

References

Coullon, J., & Nemeth, C. (2022). SGMCMCJax: A lightweight JAX library for stochastic gradient Markov chain Monte Carlo algorithms. *Journal of Open Source Software*, *7*(72), 4113. https://doi.org/10.21105/joss.04113

Devroye, L. (1986). *Non-uniform random variate generation*. Springer-Verlag.

Fonnesbeck, C., Patil, A., Huard, D., & Salvatier, J. (2015). *PyMC: Bayesian Stochastic Modelling in Python*. Astrophysics Source Code Library, record ascl:1506.005.

Foreman-Mackey, D., Farr, W., Sinha, M., Archibald, A., Hogg, D., Sanders, J., Zuntz, J., Williams, P., Nelson, A., de Val-Borro, M., Erhardt, T., Pashchenko, I., & Pla, O. (2019). emcee v3: A Python ensemble sampling toolkit for affine-invariant MCMC. *The Journal of Open Source Software*, *4*(43), 1864. https://doi.org/10.21105/joss.01864

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Marignier, A. (2023). PxMCMC: A Python package for proximal Markov chain Monte Carlo. *The Journal of Open Source Software*, *8*(87), 5582. https://doi.org/10.21105/joss.05582

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2