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Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

DRAFT

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 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, drawing some additional functionality from scipy. Under the most basic usage of lintsampler, the user provides a PDF function 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

Below is a (non-exhaustive) list of 'use cases', i.e., situations where a user might find lintsampler (and/or the the linear interpolant sampling algorithm underpinning it) to be preferable over other random sampling techniques.

In all of these use cases, it is assumed that the dimension of the problem is not too high. lintsampler works by evaluating a given PDF on the nodes of a grid (or grid-like structure, such as a tree), so the number of evaluations (and memory occupancy) grows exponentially with the number of dimensions. As a consequence, many of the efficiency arguments given for lintsampler below don't apply to higher dimensional problems. We probably wouldn't use lintsampler in more than 6 dimensions, but there is no hard limit here: the question of how many dimensions is too many will depend on the problem at hand.

A second assumption made below is that the target PDF the user wishes to sample from does not have its own exact sampling algorithm (such as the Box-Muller transform for a Gaussian PDF). The power of lintsampler lies in its applicability to arbitrary PDFs for which tailor-made sampling algorithms are not available.

Use Cases

1. Expensive PDF

If the PDF being sampled from is expensive to evaluate and a large number of samples is desired, then lintsampler might be the most cost-effective option. This is because lintsampler does not evaluate the PDF for each sample (as would be the case for importance sampling, rejection sampling or MCMC), but on the nodes of the user-chosen grid. Particularly in a low-dimensional setting where the grid does not have too many nodes, this can mean far fewer PDF evaluations. This point is demonstrated in the first example notebook in the `lintsampler` docs.

Similarly, there might be situations where the user is not so concerned about strict statistical representativeness but wants to generate a huge number of samples from a target PDF with the least possible computational cost (such as e.g., generating realistic point cloud distributions in video game graphics). They can use `lintsampler` with a coarse grid (so minimal PDF evaluations), then `sample()` to their heart's content.

2. Multimodal PDF

If the target PDF has a highly complex structure with multiple, well-separated modes, then `lintsampler` might be the *easiest* option (in terms of user configuration). In such scenarios, MCMC might struggle unless the walkers are carefully preconfigured to start near the modes. Similarly, rejection sampling or importance sampling would be highly suboptimal unless the proposal distribution is carefully chosen to match the structure of the target PDF. With `lintsampler`, the user need only ensure that the resolution of their chosen grid is sufficient to resolve the PDF structure, and so the setup remains straightforward. This is demonstrated in the second example notebook in the `lintsampler` docs.

It is worth noting that in these kinds of complex, multimodal problems, a single fixed grid might not be the most cost-effective sampling domain. For this reason, `lintsampler` also provides simple functionality for sampling over multiple disconnected grids.

3. PDF with large dynamic range

If the target PDF has a very large dynamic range, then the `DensityTree` object provided by `lintsampler` might be an effective solution. Here, the PDF is evaluated not on a fixed grid, but on the leaves of a tree which is refined such that regions of concentrated probability are more finely resolved. Such an example is shown in the third example notebook in the `lintsampler` docs.

4. Noise needs to be minimised

In Quasi-Monte Carlo (QMC) sampling, one purposefully generates more 'balanced' (and thus less random) draws from a target PDF, so that sampling noise decreases faster than $N^{-1/2}$. `lintsampler` allows easy quasi-Monte Carlo sampling with arbitrary PDFs. We are not aware of such capabilities with any other package. We give an example of using `lintsampler` for QMC in the fourth example notebook in the `lintsampler` docs.

'Real World' Example

Any one of the four use cases above would serve by itself as a sufficient case for choosing `lintsampler`, but here we give an example of a real-world scenario that combines all of the use cases. It is drawn from our own primary research interests in computational astrophysics.

Much of computational astrophysics consists of large-scale high performance computational simulations of gravitating systems. For example, simulations of planets evolving and interacting within a solar system, simulations of stars interacting within a galaxy, or vast cosmological simulations in which a whole universe is grown **in silico**.

There exists a myriad of codes used to run these simulations, each using different algorithms to solve the governing equations. One class of simulation code that has gained much attention in recent years is the class of code employing basis function expansions. In these codes, the matter density at any point in space is represented by a sum over basis functions (not unlike a Fourier series), with the coefficients in the sum changing over space and time. In these simulations, the matter comprising the system is represented everywhere as a smooth, continous fluid, but for many applications and/or downstream analyses of the simulated system, one needs to

DRAFT

instead represent the system as a set of discrete particles. These particles can be obtained by drawing samples from the continuous density field.

This is a scenario that satisfies all four of the use cases list above. To explain further: - The PDF we are sampling from (i.e., the basis expansion representation of the matter density field) can be expensive to evaluate if a large number of terms is included in the sum. - The PDF can be highly multimodal when the system we are simulating comprises many distinct gravitating substructures, such as stellar clusters. - The PDF can have a large dynamic range. Astrophysical structures such as galaxies and dark matter 'haloes' often have power-law density profiles, such as the Navarro-Frenk-White profile. Further complicating the issue is that a typical dark matter halo will host several 'subhaloes', which in turn might host 'subsubhaloes', and so on. In short, a range of spatial scales needs to be resolved. - If the particle set being sampled is to be used for further simulation, it can be helpful to draw as 'noiseless' a sample as possible for reasons of numerical stability.

For these reasons, this kind of astrophysical simulation code provides an excellent example of a 'real world' application for `lintsampler`. Here, one would cover the simulation domain with a `DensityTree` instance (or several instances, one for each primary structure), call the `refine` method to better resolve the high-density regions, then feed the tree to a `LintSampler` instance and call `sample` to generate particles. The `qmc` flag can be passed to the sampler in order to employ Quasi-Monte Carlo sampling.

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

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

DRAFT