UltraNest - a robust, general purpose Bayesian inference engine

Johannes Buchner

1 Max Planck Institute for Extraterrestrial Physics, Giessenbachstrasse, 85741 Garching, Germany. 2 Millenium Institute of Astrophysics, Vicuña MacKenna 4860, 7820436 Macul, Santiago, Chile. 3 Pontificia Universidad Católica de Chile, Instituto de Astrofísica, Casilla 306, Santiago 22, Chile. 4 Excellence Cluster Universe, Boltzmannstr. 2, D-85748, Garching, Germany

Summary

UltraNest is a general-purpose Bayesian inference package for parameter estimation and model comparison. It allows fitting arbitrary models specified as likelihood functions written in Python, C, C++, Fortran, Julia or R. With a focus on correctness and speed (in that order), UltraNest is especially useful for multi-modal or non-Gaussian parameter spaces, computational expensive models, in robust pipelines. Parallelisation to computing clusters and resuming incomplete runs is available.

Outline

Statement of need

When scientific models are compared to data, two tasks are important: 1) constraining the model parameters and 2) comparing the model to other models. While several open source, Bayesian model fitting packages are available that can be easily tied to existing models, they are difficult to run such that the result is reliable and user interaction is minimized. A chicken-and-egg problem is that one does not know a priori the posterior distribution of a given likelihood, prior and data set, and cannot chose a sampler that performs well. For example, Markov Chain Monte Carlo convergence checks may suggest good results, while in fact another distant but important posterior peak has remained unseen. Current and upcoming large astronomical surveys require characterising a large number of highly diverse objects, which requires reliable analysis pipelines. This is what UltraNest was developed for.

Overview

Nested sampling (Skilling 2004) allows Bayesian inference on arbitrary user-defined likelihoods. Additional to computing parameter posterior samples, it also estimates the marginal likelihood ("evidence", \(Z\)). Bayes factors between two competing models \(B = \frac{Z_1}{Z_2}\) are a measure of the relative prediction parsimony of the models, and form the basis of Bayesian model comparison. By performing a global scan of the parameter space from the worst to best fits to the data, nested sampling performs well also in multi-modal settings.

In the last decade, several variants of nested sampling have been developed. These include (1) how better and better fits are found while respecting the priors, (2) whether it is allowed to go back to worse fits and explore the parameter space more, and (3) diagnostics through tests and visualisations. UltraNest develops novel, state-of-the-art techniques for
all of the above. They are especially remarkable for being free of tuning parameters and theoretically justified.

Currently available efficient nested sampling implementations such as MultiNest (Feroz, Hobson, and Bridges 2009) and its open-source implementations (e.g., nestle, dynasty) rely on the heuristic multi-ellipsoidal rejection algorithm which has shown biases when the likelihood contours are not ellipsoidal (Buchner 2014; Nelson et al. 2020). UltraNest instead implements better motivated self-diagnosing algorithms, and improved, conservative uncertainty propagation. In other words, UltraNest prioritizes robustness and correctness, and maximizes speed second. For potentially complex posteriors where the user is willing to invest computation for obtaining a gold-standard exploration of the entire posterior distribution in one run, UltraNest was developed.

This package provides feature parity with other packages (such as MultiNest), e.g., circular parameters, resuming incomplete runs, efficient multi-core computation on clusters, and provides additional convenience features for visualisation and diagnostics.

Figure 1: The logo of UltraNest is a hedgehog cautiously walking up a likelihood function; made by https://www.flaticon.com/authors/freepik

Method

Nested sampling methods are systematically reviewed in Buchner et al., submitted, highlighting also the approaches used in UltraNest.

The basic outline of vanilla nested sampling (see Skilling 2004 for details) is as follows:

A set of \( N \) live points is drawn from the prior. The unit hypercube is used as a natural space (\( u \)-space) and inverse cumulative prior transforms convert the point coordinates to physical parameter units (\( v \)-space). The likelihood \( L \) is then evaluated. Nested sampling then repeatedly identifies the current worst fit and replaces it with a better fit, through a likelihood-constrained prior sampling (LRPS) procedure. At each iteration (represented by the removed, dead point), the prior space investigated shrinks by approximately \( V_{i+1}/V_i = (N-1)/N \), starting from the entire prior volume \( V_1 = 1 \). Through a Lebesgue integration, the dead point becomes a posterior sample with weight \( w_i = L_i \times V_i \), and we can estimate \( Z_i = \sum_{j=1}^{i} w_i \), yielding the posterior distribution and evidence. The iteration procedure can terminate when the weight of the live points, e.g., estimated as \( w_{\text{live}} = V_{i+1} \max_{i=1}^{N} L_{\text{live},i} \), is small (\( w_{\text{live}} \ll Z_i \)).
Reactive Nested Sampling

Instead of iterating with a fixed array of live points, UltraNest uses a tree. The root of the tree represents the entire prior volume, and its child nodes are samples from the entire prior. A breadth-first search is run, which keeps a stack of the opened nodes sorted by likelihood. The volume shrinkage is the size of the stack after removing the currently visited node to the size of the stack with the children of the currently visited node added.

When encountering a node, attaching a child to it is decided by several criteria. These include the termination criterion, minimum number of live points, whether the maximum number of model evaluations has already been reached, whether more points are desired because of the number of identified cluster or because more posterior resolution is wanted in a particular likelihood interval.

The tree formulation easily allows resuming from a run later. This includes attaching more live points where desired, e.g., to improve the number of effective samples or the $Z$ uncertainty. For this, the uncertainty contribution is measured for each dead point. The dead points are then randomly sampled proportional to their contribution. Then, a child is added to the parents of the selected nodes. This decreases the weight of the selected node, as its volume is reduced.

Reactive Nested Sampling is a flexible generalisation of the Dynamic Nested Sampling (Higson et al. 2017), which used a simple heuristic for identifying where to add more live points. The tree formulation of Reactive Nested Sampling makes implementing error propagation and variable number of live points straight-forward.

Integration procedure

UltraNest tries hard to be conservative and provides uncertainties on $Z$ and the posterior weights. Instead of a single Reactive Nested Sampling explorer, it employs several, which are randomly blinded to some parts of the tree. In particular, they see a bootstrapped subsample of the root edges (called roots in the code).

Additionally, instead of static volume shrinkage, UltraNest incorporates the scatter in volume shrinkages by drawing samples from a Beta(1, $N$) distribution (Skilling 2004).

Each explorer provides for each sample a weight estimate (0 if it is blind to it), which provide uncertainty on the posterior distribution. Each explorer provides a $Z$ estimate; the dispersion quantifies the $Z$ uncertainty.

The bootstrapped integrators is an evolution over single-bulk evidence uncertainty measures and includes the scatter in volume estimates (by beta sampling) and and likelihood values (by bootstrapping).

LRPS procedures in UltraNest

The live points all fulfill the current likelihood threshold, therefore they can be used to trace out the neighbourhood where a new, independent prior sample can be generated that also fulfills the threshold. Region-based sampling uses rejection sampling using constructed geometries.

Region construction

Mukherjee, Parkinson, and Liddle (2006) proposed heuristically fitting an ellipsoid to the live points to achieve this, which is expanded by a empirically determined factor. Shaw, Bridges, and Hobson (2007) expanded this with multiple ellipsoids constructed via a recursive splitting heuristic, which are then again expanded by a factor. This works well for many problems. However, these heuristics are not well justified, and do not detect when
they work poorly. Buchner (2014) explored hyper-rectangle posteriors and a heavy-tailed distribution which show-case that the ellipsoids cut off too much of the prior space.

Buchner (2014) proposed a more robust algorithm, RadFriends. RadFriends creates an hyper-sphere around each live point. The radius of the spheres is determined by cross-validation: Some live points are randomly left out following a bootstrapping procedure, and based on sphere around the remainder the radius is chosen so that the former are recovered. This is repeated over many rounds, and the largest radius kept. In this fashion, some guarantees for the robustness of the region are available. An animation of this procedure is available. As a side-effect, RadFriends automatically provides parameter-free unsupervised clustering, which relies only on the uniformity assumption of the u-space.

MLFriends (Buchner 2019) makes several efficiency improvements to RadFriends: Firstly, when a secondary likelihood peak is dying out, RadFriends tends to drastically increase its radius to be able to recapture it from the other peak(s). This is avoided by not allowing the radius to increase. Secondly, instead of spheres, MLFriends learns a metric (hence the name), initially by taking a sample variance of the live points. In later iterations, the identified clusters are overlaid by subtracting their mean, and the sample variance of the co-centered points taken as the metric. This helps identify the shape within clusters and discards more space between the clusters.

The bootstrapping approach can also be applied the single ellipsoid method: The sample covariance of the selected live points identifies the shape of the ellipse, while the left-out live points identify the scale. A single ellipsoid performs best in ellipsoidal posteriors (such as gaussians). MLFriends performs best in complex posterior shapes with low dimension.

UltraNest therefore combines multiple region constructions, and uses their intersection: (1) MLFriends, (2) a bootstrapped single ellipsoid in u-space (3) a bootstrapped single ellipsoid in v-space. The last one drastically helps when one parameter constraint scales with another, (e.g., funnel shapes). By altering how the parameter is applied in the likelihood function and how its prior is transformed, the user can – without altering the posterior – help produce ellipsoidal shapes. The better parameterization narrows the sampling region, and leads to efficiency gains.

Region sampling

Samples are then drawn either from the entire prior, the single u-space ellipsoid or MLFriends ellipsoids (accounting for overlaps), and filtered by the other constraints (including the transformed v-space ellipsoid). UltraNest dynamically switches away from slow methods (e.g., sampling from the unit hypercube) when that is inefficient.

Once a region proposal is chosen, the likelihood function is evaluated, and proposals below the current threshold are rejected. The proposal process is repeated until success.

Instead of handling one proposal point at a time, many points are proposed and filtered at once (vectorization). Because Python function calls can be costly, this speeds up UltraNest, e.g., in comparison to dynesty.

Step sampling

Besides rejection sampling, UltraNest supports several types of Monte Carlo random walks. These are more efficient in some problems, in particular in high dimensions ($d > 20$). The step samplers include:

- Slice sampling (as in Polychord, Handley, Hobson, and Lasenby 2015)
- Hit-and-run sampling
- Constrained Hamiltonian Monte Carlo with No-U turn sampling (similar to NoGUTS, Griffiths and Wales 2019)
The user can choose whether to filter proposals with the constructed regions. This has some function call overhead but can reduce likelihood evaluations, especially important for slow models.

Variations that alter some parameters more often than others (fast-slow) are also implemented.

Diagnostics & Visualisations

Run-time visualisation

During the run, UltraNest by default shows a visualisation in the standard output where the current live points are distributed. Thus already during a run, the user can identify where the fit is spending its time and choose to abort the fit and alter the model. In Jupyter notebooks, a visualisation widget is shown, when run in the terminal, it is shown in the standard output. Additionally, log output gives information on the progress. Both can be turned off.

Posterior visualisations

Publication-read corner/pairs plots and trace plots (Higson et al. 2019) are created, based on code from corner (Foreman-Mackey 2016) and dynasty (Speagle 2020).

The weighted posterior samples (or a resampled set with uniform weighting) can however also be plotted with any other tool.

Diagnostic tests

Fowlie, Handley, and Su (2020) proposed a test for identifying LRPS biases during a run, by checking whether new children are inserted in the sorted stack at uniformly distributed positions (insertion order or rank). Buchner et al., in prep. presents a similar test based on rank statistics, which is slightly more sensitive and easier to implement. UltraNest prints the p-value of this test during the run. More experience is needed to identify reasonably rolling windows and thresholds for applying this test.

Features

Parallelisation

UltraNest allows parallelisation from laptops to computing clusters with the Message Passing Interface (MPI). Programs using UltraNest can be run with MPI without modification, if the mpi4py python library is installed. The region construction, step sampling, and multi-explorer approaches are all parallelised across cores and are kept on the same cores for efficiency.

Resuming

UltraNest can optionally write to a folder, where it will store summary statistics of the posterior and evidence, posterior chains, already sampled points, diagnostic log files, and visualisations. Resuming a run, also with different algorithm parameters is supported. Previously sampled points are stored in compressed HDF5, which saves disk space and avoids file corruption. It also avoids two processes trying to operate on the same file at once.
Languages

In the Github repository, wrappers are provided for models (prior transforms and likelihood functions) written in:

- Python
- C
- C++
- Fortran
- Julia
- R
- Javascript

For Julia, the dedicated wrapper package UltraNest.jl is available (Schulz and Buchner 2020).

For Javascript, a simpler implementation of MLFriends is available (Buchner 2018), which allows building interactive Bayesian inference websites.

Documentation

Extensive documentation is available. This includes several practical tutorials highlighting good practices of data analysis in astro-statistics, including:

- Fitting linear or powerlaw correlations
- Inferring parameter distributions from many uncertain parameter measurements (hierarchical Bayesian model)
- Different types of model comparison
- Prior and posterior predictive checks
- Uncertainty quantification

Basic Example

Finally, a simple example of running UltraNest is given in the following Python snippet:

```python
import ultranest

sampler = ultranest.ReactiveNestedSampler(param_names, my_likelihood, my_prior_transform)
result = sampler.run()

print('evidence estimate:', results['logz'])
print('first parameter posterior mean:', results['samples'][0].mean())
```

Here `param_names` is a list of strings, `my_prior_transform` is a function transforming from a sample from the unit cube to physical parameter, and `my_likelihood` is the likelihood function receiving such parameters and returning their likelihood value.

In case a output folder is used and resumed from:

```python
import ultranest

sampler = ultranest.ReactiveNestedSampler(param_names, my_likelihood, my_prior_transform,
                                          log_dir="myanalysis", resume=True)
result = sampler.run()
sampler.print_results()
sampler.plot()
```

More advanced uses are show-cased in the documentation, including speed-ups, parallelisation and step samplers.
Acknowledgements

I am very thankful to Fred Beaujean, Josh Speagle and J. Michael Burgess for insightful conversations.

References

Buchner, Johannes. 2014. “A statistical test for Nested Sampling algorithms.” Statistics and Computing, July, 1–10. https://doi.org/10.1007/s11222-014-9512-y.

———. 2018. “Ultranest-Js.” GitHub Repository. GitHub. https://github.com/JohannesBuchner/ultranest-js.

Buchner, Johannes. 2019. “Collaborative Nested Sampling: Big Data versus Complex Physical Models” 131 (1004): 108005. https://doi.org/10.1088/1538-3873/aac7fc.

Feroz, F., M. P. Hobson, and M. Bridges. 2009. “MULTINEST: an efficient and robust Bayesian inference tool for cosmology and particle physics” 398 (October): 1601–14. https://doi.org/10.1088/1/1365-2966.2009.14548.x.

Foreman-Mackey, Daniel. 2016. “Corner.py: Scatterplot Matrices in Python.” The Journal of Open Source Software 1 (2): 24. https://doi.org/10.21105/joss.00024.

Fowlie, Andrew, Will Handley, and Liangliang Su. 2020. “Nested sampling cross-checks using order statistics.” arXiv E-Prints, June, arXiv:2006.03371. http://arxiv.org/abs/2006.03371.

Griffiths, Matthew, and David J Wales. 2019. “Nested Basin-Sampling.” Journal of Chemical Theory and Computation 15 (12): 6865–81. https://www.researchgate.net/profile/Matthew_Griffiths12/publication/336097602_Nested_Basin-Sampling/links/5dd52651a6fddc37897ad881/Nested-Basin-Sampling.pdf.

Handley, W. J., M. P. Hobson, and A. N. Lasenby. 2015. “POLYCHORD: next-generation nested sampling” 453 (November): 4384–98. https://doi.org/10.1093/mnras/stv1911.

Higson, Edward, Will Handley, Michael Hobson, and Anthony Lasenby. 2019. “NESTCHECK: diagnostic tests for nested sampling calculations” 483 (2): 2044–56. https://doi.org/10.1093/mnras/sty3090.

Higson, E., W. Handley, M. Hobson, and A. Lasenby. 2017. “Dynamic nested sampling: an improved algorithm for parameter estimation and evidence calculation.” ArXiv E-Prints, April. http://arxiv.org/abs/1704.03459.

Mukherjee, P., D. Parkinson, and A. R. Liddle. 2006. “A Nested Sampling Algorithm for Cosmological Model Selection” 638 (February): L51–L54. https://doi.org/10.1086/501068.

Nelson, Benjamin E., Eric B. Ford, Johannes Buchner, Ryan Cloutier, Rodrigo F. Díaz, João P. Faria, Nathan C. Harra, Vinesh M. Rajpaul, and Surangkhan Rukdee. 2020. “Quantifying the Bayesian Evidence for a Planet in Radial Velocity Data” 159 (2): 73. https://doi.org/10.3847/1538-3881/ab5190.

Schulz, Oliver, and Johannes Buchner. 2020. “UltraNest.jl.” GitHub Repository. GitHub. https://github.com/bat/UltraNest.jl.

Shaw, J. R., M. Bridges, and M. P. Hobson. 2007. “Efficient Bayesian inference for multimodal problems in cosmology” 378 (July): 1365–70. https://doi.org/10.1086/1/1365-2966.2007.11871.x.

Skilling, J. 2004. “Nested Sampling.” Edited by Roland Preuss Rainer Fischer and Udo von Toussaint. AIP Conference Proceedings 735 (1): 395. https://doi.org/10.1063/1.1835238.
Speagle, Joshua S. 2020. “DYNESTY: a dynamic nested sampling package for estimating Bayesian posteriors and evidences” 493 (3): 3132–58. https://doi.org/10.1093/mnras/staa278.

Johannes Buchner. (). UltraNest - a robust, general purpose Bayesian inference engine. . () . https://doi.org/