UNIVERSITY OF CENTRAL FLORIDA

LISA Users Manual

LARGE-SELECTION INTERFACE FOR SAMPLING ALGORITHMS

Authors:
Michael D. HIMES

Supervisor:
Dr. Joseph HARRINGTON

April 23, 2025

Contents

1 Team Members

- Michael Himes[1], University of Central Florida (mhimes@knights.ucf.edu)
- Joseph Harrington, University of Central Florida

2 Introduction

This document describes LISA, the Large-selection Interface for Sampling Algorithms. LISA provides a unified interface for different Bayesian samplers, both Markov chain Monte Carlo (MCMC) and nested sampling (NS).

Presently, LISA supports the DEMC algorithm of ter Braak (2006), DEMCzs of ter Braak & Vrugt (2008, 'snooker'), MultiNest of Feroz et al. (2008) via PyMultiNest (Buchner et al. 2014), UltraNest of Buchner (2016), dynesty of Speagle (2019), DNest4 of Brewer & Foreman-Mackey (2018), polychord of Handley et al. (2015a, 2015b), and MT-DREAMzs of Laloy and Vrugt (2012) via PyDREAM (https://github.com/LoLab-VU/PyDREAM). We welcome users to contribute wrappers for additional algorithms via a pull request on Github.

The detailed LISA code documentation and User Manual[2] are provided with the package to assist users. For additional support, contact the lead author (see Section 1).

LISA is released under the Reproducible Research Software License. For details, see https://planets.ucf.edu/resources/reproducible-research/software-license/.

The LISA package is organized as follows:

```
LISA
 ├─ doc
 ├─ example
 ├─ lib
 └─ modules
     └─ MCcubed
```

3 Installation

3.1 System Requirements

LISA was developed on a Linux machine using the following versions of packages:

- Python 3.7.2
- Numpy 1.16.2
- Matplotlib 3.0.2

[1]https://github.com/mdhimes/

[2]Most recent version of the manual available at https://exosports.github.io/LISA/doc/LISA_User_Manual.html

- mpi4py 3.0.3
- Scipy 1.5.3
- h5py 2.9.0
- MULTINEST 3.10
- pymultinest 2.10
- ultranest 2.2.2
- dynesty 1.0.1
- DNest4 0.2.4
- PyDREAM 2.0.0

3.2 Install and Compile

To begin, obtain the latest stable version of LISA.

First, decide on a local directory to hold LISA. Let the path to this directory be 'LISA'. Now, recursively clone the repository:

```
git clone --recursive https://github.com/exosports/LISA LISA/
cd LISA/
```

LISA contains a file to easily build a conda environment capable of executing the software. Create the environment and activate it via

```
conda env create -f environment.yml
conda activate lisa
```

On some systems, this approach may fail. For a more robust approach that requires additional steps, enter

```
conda create -n lisa python=3.7.2
conda activate lisa
conda env update --file environment.yml
```

This will build a base Python 3.7.2 environment, activate it, and then update it with the packages necessary to run LISA.

Mac users may need to install additional tools:

```
xcode-select --install
```

This installs make, gcc, git, and other utilities needed.

Now, install LISA:

```
    python setup.py install
```

This will compile required submodules and build an importable package.

To check that everything installed properly, start up a Python session and try

```
    import lisa
```

If it succeeds without warnings about missing MultiNest files, you are now ready to use LISA!

4 Example

The following script will walk a user through using LISA for a basic quadratic fit.
To begin, copy the requisite files to a directory parallel to LISA. Beginning from LISA/,

```
mkdir ../run
cp -a ./example/* ../run/.
cd ../run
```

Make your test data. Execute the provided script with desired parameters. For example, data following $3x^2 - x + 2$ would be created via

```
./make_data.py 3 -1 2
```

This will produce true.npy, data.npy, uncert.npy, and params.npy, which contain the true data, noisy data, uncertainty on each data point, and the true parameters, respectively. The x-axis is assumed to be the integers from -5 to 5, inclusive, and the uncertainty is assumed to be the square root of the true data.

Now, take a look at and execute LISA for each of the samplers:

```
./demc_example.py
./dnest4_example.py
./dynesty_example.py
./snooker_example.py
./multinest_example.py
./polychord_example.py
./ultranest_example.py
```

Each will produce some output files (logs, plots, etc.) in appropriately-named subdirectories.

5 Program Inputs

LISA's design as an importable package allows for convenient usage in existing codes. Users interact with LISA through its `setup or run functions. Each has 1 position argument (the desired algorithm; options are listed below), and they take keyword arguments for the different parameters of the sampling algorithm.`

5.1 Sampling Algorithm

LISA currently supports 8 Bayesian sampling algorithms:

- demc : DEMC MCMC algorithm of ter Braak (2006)
- dnest4 : Diffusive NS implementation of Brewer & Foreman-Mackey (2018)
- dream : MT-DREAMzs algorithm of Laloy and Vrugt (2012)
- dynesty: Dynamic NS of Speagle (2019)
- polychord: PolyChord NS algorithm of Handley et al. (2015a, 2015b)
- snooker : DEMCzs MCMC algorithm of ter Braak & Vrugt (2008)
- multinest: MultiNest NS algorithm of Feroz et al. (2008)
- ultranest: UltraNest NS algorithm of Buchner (2014, 2016, 2019).

The choice of sampling algorithm also decides the allowed inputs. Each sampler's inputs are listed in the following subsections. Bold indicates a parameter that must be specified (typically, they have no default); optional parameters are given reasonable defaults. Section 5.2 describes each parameter. Note that some 'optional' parameters can strongly influence the results (e.g., dlogz for nested samplers), so users are encouraged to experiment with them.

5.1.1 demc & snooker

- **burnin**
- **data**
- fbestp
- fext
- flog
- fsavefile
- fsavemodel
- hsize (only snooker)
- indparams
- kll
- **model**
- modelper

- **nchains**
- **niter**
- **outputdir**
- **pinit**
- **pmax**
- **pmin**
- pnames
- **pstep**
- resume
- thinning
- truepars
- **uncert**
- verb

5.1.2 dream

- burnin
- fbestp
- fext
- fprefix
- fsavefile
- **loglike**
- multitry
- **nchains**
- **niter**
- **outputdir**
- **pmax**
- **pmin**
- pnames
- pstep

- resume
- thinning
- truepars
- verb

5.1.3 dnest4

- beta
- fbestp
- fext
- fsavefile
- kll
- lam
- **loglike**
- **model**
- **niter**
- **nlevel**
- **nlevelint**
- **nperstep**
- **outputdir**
- **perturb**
- pnames
- **prior**
- **pstep**
- resample
- truepars
- verb

5.1.4 dynesty, multinest, polychord, & ultranest

- `bound (only dynesty)`
- `dlogz`
- `dumper`
- `fbestp`
- `fcheckpoint (only dynesty)`
- `fext`
- `frac_remain (only ultranest)`
- `fprefix (only multinest)`
- `fsavefile`
- `kll`
- `Lepsilon (only ultranest)`
- **`loglike`**
- `min_ess (only dynesty & ultranest)`
- **`model`**
- `niter`
- **`nlive`**
- **`nlive_batch`** `(only dynesty)`
- `nrepeat (only polychord)`
- **`outputdir`**
- `pnames`
- **`prior`**
- **`pstep`**
- `resume (all except dynesty)`
- `sample (only dynesty)`
- `truepars`
- `verb`

5.2 Parameter Dictionary

Users are only required to specify the relevant parameters listed in the previous section. Each parameter is described below, alphabetically. To utilize default values, do not include it in **kwargs.

- beta : float. DNest4 only. From their docs: strength of effect to force histogram to equal push. Default: 100.0
- bound : str. Dynesty only. Option to bound the target distribution. Choices: none (sample from unit cube), single (one ellipsoid), multi (multiple possibly overlapping ellipsoids), balls (overlapping balls centered on each live point), cubes (overlapping cubes centered on each live point). Default: multi
- burnin : int. Number of initial iterations to be discarded.
- data : array, Numpy binary. Measured data for inference. Must be Numpy array, list, or a path to a NPY file.
- dlogz : float. Target evidence uncertainty (stops when below this value). Lower = more accurate, but also requires longer runtime. Default: 0.1
- dumper : object. Polychord only. Function to output info during the inference. Default: None
- fbestp : str. Filename for array of best-fit parameters. Must be NPY file.
- fcheckpoint: str. Dynesty only. Path to save checkpoint file to allow for resuming the run. If a relative path, assumes it is relative to ‘outputdir‘. Default: dynesty.save, located in ‘outputdir‘
- fext : str. File extension for saved plots. Options: .png, .pdf Default: .png
- flog : str. MCMCs only. /path/to/log file to save out. Default: MCMC.log, located in ‘outputdir‘
- fprefix : str. Prefix for output filenames. Recommended to be a directory (possibly with a prefix for all produced files), as this will create a subdirectory within ‘outputdir‘. Default: pmn/ for multinest, run1 for polychord.
- fsavefile : str. Filename to store parameters explored. If relative path, it is considered with respect to ‘outputdir‘. Default: ‘outputd

- fsavemodel : str. MCMCs only (currently). Filename to store models, corresponding to the parameters. If relative path, it is considered with respect to ‘outputdir‘. If None, file is not saved. Beware: the file may be extremely large if the model output has high dimensionality. See ‘modelper’ to have it automatically split into subfiles. Default: None

- frac_remain : float. UltraNest only. Sets the fraction remainder when integrating the posterior.

- hsize : int. Snooker only. Number of samples per chain to seed the phase space. Default: 10

- indparams : list. MCMCs only. Additional parameters needed by ‘model‘.

- kll : object. Datasketches KLL object, for model quantiles. Use None if not desired or if Datasketches is not installed. Default: None

- lam : float. DNest 4 only. From their docs: backtracking scale length. Default: 5.0

- Lepsilon : float. UltraNest only. From their docs: "Terminate when live point likelihoods are all the same, within Lepsilon tolerance

- loglike : object. Nested samplers only. Function defining the log likelihood.

- min_ess : int. Minimum effective sample size (ESS). Default: 500

- model : object. Function defining the forward model.

- modelper : int. MCMCs only. Sets how to split ‘fsavemodel‘ into subfiles. If 0, does not split. If >0, saves out every ‘modelper‘ iterations. E.g., if nchains=10 and modelper=5, splits every 50 model evaluations. Default: 0

- multitry : int. DREAM only. Determines whether to use multi-try sampling. Default: 5

- nchains : int. Number of parallel samplers. Default: 1

- niter : int. Maximum number of iterations. Nested samplers default to no limit.

- nlive : int. (Minimum) number of live points to use. Default: 500

- nlive_batch: int. Dynesty only. From their docs: "The number of live points used when adding additional samples from a nested sampling run within each batch." Default: 500.

- nlevel : int. DNest4 only. From their docs: Maximum number of levels to create. Default: 30
- nlevelint : int. DNest4 only. Number of moves before creating new level. Default: 10000
- nperstep : int. DNest4 only. Number of moves per MCMC iteration. Default: 10000
- nrepeat : int. Polychord only. From their docs: The number of slice slice-sampling steps to generate a new point. Increasing [nrepeat] increases the reliability of the algorithm. Default: None (uses polychord's default of 5*ndims)
- outputdir : str. path/to/directory where output will be saved.
- perturb : object. DNest4 only. Function that proposes changes to parameter values.
- pinit : array, Numpy binary. For MCMCs, initial parameters for samplers. For nested sampling algorithms, values are used for parameters that are held constant, if any. Must be Numpy array, list, or a path to a NPY file.
- pmax : array, Numpy binary. Maximum value for each parameter. Must be Numpy array, list, or a path to a NPY file.
- pmin : array, Numpy binary. Minimum value for each parameter. Must be Numpy array, list, or a path to a NPY file.
- pnames : array. Name of each parameter (can use some LaTeX if desired). Must be Numpy array or list.
- prior : object. Function defining the prior.
- pstep : array. Step size for each parameter. For MCMCs, only matters for the initial samples and determining constant parameters, as step size is automatically adjusted. For nested sampling algorithms only used to determine constant parameters.
- resample : float. DNest4 only. Must be non-negative. If >0, corresponds to a factor affecting the number of draws from the posterior. If 0, no resampling is performed. Default: 100
- resume: bool. Determines whether to resume a previous run, if possible. Default: False
- sample : str. Dynesty only. Sampling method. Choices (descriptions from their docs): unif (uniform sampling), rwalk (random walks from current live point), rstagger (random "staggering" away from current live point), slice (multivariate slice sampling), rslice (random slice sampling), hslice ("Hamiltonian" slice sampling),

auto (automatically selected based on problem dimensionality). Default: auto

- thinning : int. Thinning factor for the posterior (keep every N iterations). Example: a thinning factor of 3 will keep every third iteration. Only recommended when the computed posterior is extremely large. Default: 1
- truepars : array, Numpy binary. Known true values for the model parameters. If unknown, use None. Default: None
- uncert : array, Numpy binary. Data uncertainties. Must be Numpy array, list, or a path to a NPY file.
- verb : int. Verbosity level. If 0, only essential messages are printed by LISA. If 1, prints additional messages (e.g., loading data files). In the future, additional levels may be added. Default: 0

6 Program Outputs

LISA has two main functions: setup and run. setup returns an initialized Sampler object. run returns the Sampler object (with additional attributes outp for the posterior and bestp for the best parameters) and produces at least 3 plots.

6.1 Returns

- samp: Sampler object. Contains the attributes listed Sections 5.1.1 -- 5.1.4.

6.2 Output Files

- pairwise: corner plot of histograms of the 2D marginalized posteriors.
- posterior: histogram plots of the 1D marginalized posteriors.
- trace: parameter history plots.

If fsavefile and fbestp are not None, there are two NPY files produced.

- bestp.npy: Array of best parameters.
- outp.npy: Array of the approximation to the posterior.

Except for dynesty, each sampler also has additional output files, briefly discussed below.

6.3 demc & snooker

- MCMC.log: text file with statistics about the MCMC run.

6.4 dnest4

DNest4 produces some files about the sampler's history. All are saved into 'outputdir' EXCEPT for sampler_state.txt, which is saved into the directory where LISA was executed from. For more details, see the DNest4 code/docs.

6.5 multinest

- marginals_multinest.png: plot of the 1D marginalized posteriors produced by multinest.

Additionally, a subdirectory containing the files necessary to resume the run as well as summary plots and files is created (default is pmn/). For more details, see the MultiNest/PyMultiNest docs.

6.6 polychord

- MCMC.log: text file with statistics about the MCMC run.

6.7 ultranest

UltraNest's output gets saved into a subdirectory named 'run' followed by a number. It contains a log file, chain histories, summary files, and plots. For more details, see UltraNest's docs.

7 FAQ

This section will cover some frequently asked questions about using LISA. Have a question that isn't answered below? Please send it to the corresponding author so we can add it to this section!

Q: What is effective sample size (ESS) and why should I care? A: As Bayesian samplers explore a phase space, the subsequent sets of parameters are based in part on earlier sets of parameters. Thus, iterations are correlated. After some number of samples, new iterations will ``forget'' some of the earlier samples. Yet, we are really interested in the number of independent samples, as that will inform how thoroughly the phase space has been explored (taking a bunch of correlated samples doesn't give us much new information!). The number of iterations necessary before a new sample is independent of a given earlier sample (steps per effective independent sample, SPEIS) informs the ESS. For example, if we have 20,000 samples, but a given sample is correlated with the previous 100 samples, then the ESS is just 20000 / 100, or 200. The

ESS value, in turn, informs us of the accuracy of a credible region determined from the posterior distribution. Without providing a detailed proof (interested readers can find it in Harrington et al. 2021, Appendix A), the relationship between the ESS, a credible region $\hat{C}$, and the uncertainty on the credible region $s_{\hat{C}}$ is

$$\mathrm{ESS} \approx \frac{\hat{C}(1-\hat{C})}{s_{\hat{C}}^2}. \tag{1}$$

Thus, for a 0.5% uncertainty on the 95.45% region (2σ for a Gaussian), an ESS $\approx$ 1700 is required. A SPEIS of 1000 would thus require 1.7 million iterations!

8 Be Kind

Please note that this software has not been officially released yet; see the license for some restrictions on its usage prior to release. Upon release, we will add the relevant citation here, with a Bibtex entry.
Thanks!