# arianna

*Release 0.0.1*

**Minas Karamanis**

**Apr 26, 2021**

# CONTENTS:

**arianna**[1] is a Python implementation of the *Metropolis–Coupled Slice Sampling* method that generates posterior samples from high-dimensional and strongly multimodal distributions. Apart from *Bayesian parameter inference*, **arianna** also provides unbiased and low-variance estimates of the *model evidence (aka marginal likelihood)* at no additional cost. The sampler is modular and does not require any hand-tuning from the user. Its parallel and black-box nature renders it ideal for computationally expensive models with high number of parameters often met in the physical sciences.

[1] Named after Dr. Arianna W. Rosenbluth, one of the main developers of the Metropolis algorithm and the first person in history to ever code a Markov Chain Monte Carlo algorithm.

CHAPTER

ONE

# DOCUMENTATION

## 1.1 Installation

### 1.1.1 Dependencies

**arianna** depends on `matplotlib`, `numpy`, and optionally `scipy`.

### 1.1.2 Using pip

The easiest way to install the most recent stable version of `arianna` is with pip:

```
pip install arianna
```

### 1.1.3 From source

Alternatively, you can get the source by cloning the git repository:

```
git clone https://github.com/minaskar/arianna.git
```

Once you've downloaded the source, you can navigate into the root source directory and run:

```
pip install .
```

## 1.2 Quickstart guide

## 1.3 API Reference

**arianna** consists mainly of five parts:

### 1.3.1 The Ensemble Sampler

**class** arianna.**ReplicaExchangeSampler**(*ntemps*, *nwalkers*, *ndim*, *loglike_fn*, *logprior_fn*, *loglike_args=None*, *loglike_kwargs=None*, *logprior_args=None*, *logprior_kwargs=None*, *prior_transform=None*, *betas=None*, *swap=1.0*, *tune=False*, *tolerance=0.05*, *patience=5*, *maxsteps=10000*, *mu=1.4*, *maxiter=10000*, *pool=None*, *vectorize=False*, *verbose=True*, *check_walkers=True*, *shuffle_ensemble=True*, *light_mode=False*, *adapt=False*, *t0=1000.0*, *gamma=1.0*, *hist=0.0*)

A Replica Exchange Slice Sampler.

**Parameters**

- **nwalkers** (*int*) – The number of walkers in the ensemble.
- **ndim** (*int*) – The number of dimensions/parameters.
- **loglike_fn** (*callable*) – A python function that takes a vector in the parameter space as input and returns the natural logarithm of the likelihood function at that position.
- **logprior_fn** (*callable*) – A python function that takes a vector in the parameter space as input and returns the natural logarithm of the prior pdf at that position.
- **args** (*list*) – Extra arguments to be passed into the logp.
- **kwargs** (*list*) – Extra arguments to be passed into the logp.
- **tune** (*bool*) – Tune the scale factor to optimize performance (Default is True.)
- **tolerance** (*float*) – Tuning optimization tolerance (Default is 0.05).
- **patience** (*int*) – Number of tuning steps to wait to make sure that tuning is done (Default is 5).
- **maxsteps** (*int*) – Number of maximum stepping-out steps (Default is 10^4).
- **mu** (*float*) – Scale factor (Default value is 1.0), this will be tuned if tune=True.
- **maxiter** (*int*) – Number of maximum Expansions/Contractions (Default is 10^4).
- **pool** (*bool*) – External pool of workers to distribute workload to multiple CPUs (default is None).
- **vectorize** (*bool*) – If true (default is False), logprob_fn receives not just one point but an array of points, and returns an array of likelihoods.
- **verbose** (*bool*) – If True (default) print log statements.
- **check_walkers** (*bool*) – If True (default) then check that `nwalkers >= 2*ndim` and even.
- **shuffle_ensemble** (*bool*) – If True (default) then shuffle the ensemble of walkers in every iteration before splitting it.
- **light_mode** (*bool*) – If True (default is False) then no expansions are performed after the tuning phase. This can significantly reduce the number of log likelihood evaluations but works best in target distributions that are apprroximately Gaussian.

**property act**

Integrated Autocorrelation Time (IAT) of the `beta = 1` Markov Chain.

**Returns** Array with the IAT of each parameter.

**property chain**
Returns the chains.

**Returns** Returns the chains of shape (nsteps, nwalkers, ndim).

**compute_log_prob**(*coords*, *betas*)
Calculate the vector of log-probability for the walkers

**Parameters**

- **coords** – (ndarray[..., ndim]) The position vector in parameter space where the probability should be calculated.
- **betas** – (ndarray[...]) The beta values for each walker in `coords`.

**Returns** A vector of log-probabilities with one entry for each walker in this sub-ensemble. log_prior: A vector of log-prior values with one entry for each walker in this sub-ensemble. log_like: A vector of log-likelihood values with one entry for each walker in this sub-ensemble.

**Return type** log_prob

**property efficiency**
Effective Samples per Log Probability Evaluation.

**Returns** efficiency

**property ess**
Effective Sampling Size (ESS) of the `beta = 1` Markov chain walkers.

**Returns** ESS

**get_chain**(*flat=False*, *thin=1*, *discard=0*, *warm=False*)
Get the Markov chain containing the samples.

**Parameters**

- **flat** (*bool*) – If True then flatten the chain into a 2D array by combining all walkers (default is False).
- **thin** (*int*) – Thinning parameter (the default value is 1).
- **discard** (*int*) – Number of burn-in steps to be removed from each walker (default is 0). A float number between 0.0 and 1.0 can be used to indicate what percentage of the chain to be discarded as burnin.
- **hot** (*bool*) – If True then return warm chains too, else return only the cold (beta=1) chain (default is False).

**Returns** Array object containg the Markov chain samples (2D if flat=True, 3D if flat=False).

**get_last_log_prob**()
Return the log probability values for the last position of the walkers.

**get_last_sample**()
Return the last position of the walkers.

**get_log_prob**(*flat=False*, *thin=1*, *discard=0*)
Get the value of the log probability function evalutated at the samples of the Markov chain.

**Parameters**

- **flat** (*bool*) – If True then flatten the chain into a 1D array by combining all walkers (default is False).
- **thin** (*int*) – Thinning parameter (the default value is 1).

- **discard** (*int*) – Number of burn-in steps to be removed from each walker (default is 0). A float number between 0.0 and 1.0 can be used to indicate what percentage of the chain to be discarded as burnin.

**Returns** Array containing the value of the log probability at the samples of the Markov chain (1D if flat=True, 2D otherwise).

**get_logz**(*discard=0.5*, *correction=False*)

Calculate the natural logarithm of model evidence (aka marginal likelihood) logZ using the trapezoidal rule.

**Parameters**

- **discard** – (float or int) Number of burn-in steps to be removed from each walker (default is 0). A float number between 0.0 and 1.0 can be used to indicate what percentage of the chain to be discarded as burnin.
- **correction** – (bool) If True (default) then the result is calculated using the *Frier et al. (2014)* corrected trapezoidal rule that takes into account the variance of the log-likelihood.

**Returns** (float) log model evidence

**reset**()

Reset the state of the sampler. Delete any samples stored in memory.

**run_mcmc**(*start*, *nsteps=1000*, *thin=1*, *progress=True*, *log_like0=None*, *log_prior0=None*, *thin_by=1*)

Run MCMC.

**Parameters**

- **start** (*float*) – Starting point for the walkers. If `None` then the sampler proceeds from the last known position of the walkers.
- **nsteps** (*int*) – Number of steps/generations (default is 1000).
- **thin** (*float*) – Thin the chain by this number (default is 1, no thinning).
- **progress** (*bool*) – If True (default), show progress bar.
- **log_prob0** (*float*) – Log probability values of the walkers. Default is `None`.
- **thin_by** (*float*) – If you only want to store and yield every `thin_by` samples in the chain, set `thin_by` to an integer greater than 1. When this is set, `iterations * thin_by` proposals will be made.

**sample**(*start*, *log_like0=None*, *log_prior0=None*, *iterations=1*, *thin=1*, *thin_by=1*, *progress=True*)

Advance the chain as a generator. The current iteration index of the generator is given by the `sampler.iteration` property.

**Parameters**

- **start** (*float*) – Starting point for the walkers.
- **log_prob0** (*float*) – Log probability values of the walkers. Default is `None`.
- **iterations** (*int*) – Number of steps to generate (default is 1).
- **thin** (*float*) – Thin the chain by this number (default is 1, no thinning).
- **thin_by** (*float*) – If you only want to store and yield every `thin_by` samples in the chain, set `thin_by` to an integer greater than 1. When this is set, `iterations * thin_by` proposals will be made.
- **progress** (*bool*) – If True (default), show progress bar.

**property scale_factor**
Scale factor values during tuning.

**Returns** scale factor mu

### 1.3.2 The Chain Manager

**class** arianna.**ChainManager**(*nchains=1*, *comm=None*)
Class to serve as context manager to handle to MPI-related issues, specifically, the managing of `MPIPool` and splitting of communicators. This class can be used to run `nchains` in parallel with each chain having its own `MPIPool` of parallel walkers.

**Parameters**

- **nchains** (*int*) – the number of independent chains to run concurrently
- **comm** (*MPI.Communicator*) – the global communicator to split

**allgather**(*x*)
Allgather method to gather `x` in all chains. This is equivalent to first `scatter` and then `bcast`.

**Parameters** **x** (*Python object*) – The python object to be gathered.

**Returns** **x** – The python object, gathered in all ranks.

**Return type** Python object

**bcast**(*x*, *root*)
Broadcast method to send `x` from `rank = root` to all chains.

**Parameters**

- **x** (*Python object*) – The python object to be send.
- **root** (*int*) – The rank of the origin chain from which the object x is sent.

**Returns** **x** – The input object x in all ranks.

**Return type** Python object

**gather**(*x*, *root*)
Gather method to gather `x` in `rank = root` chain.

**Parameters**

- **x** (*Python object*) – The python object to be gathered.
- **root** (*int*) – The rank of the chain that x is gathered.

**Returns** **x** – The input object x gathered in `rank = root`.

**Return type** Python object

**property get_pool**
Get parallel `pool` of workers that correspond to a specific chain. This should be used to parallelize the walkers of each `chain` (not the chains themselves). This includes the `map` method that `zeus` requires.

**property get_rank**
Get `rank` of current `chain`. The minimum `rank` is `0` and the maximum is `nchains-1`.

**scatter**(*x*, *root*)
Scatter method to scatter `x` from `rank = root` chain to the rest.

**Parameters**

- **x** (*Python object*) – The python object to be scattered.

- **root** (*int*) – The rank of the origin chain from which the x is scattered.

**Returns** **x** – Part of the input object x that was scattered along the ranks.

**Return type** Pythonn object

CHAPTER

TWO

# ATTRIBUTION

Please cite the following if you find this code useful in your research. The BibTeX entry for the paper is:

```
@article{arianna,
    title={arianna: A Metropolis--Coupled Slice Sampler for Bayesian Inference and
↪Model Selection},
    author={Minas Karamanis and Florian Beutler},
    year={2021},
    note={in prep}
}
```

CHAPTER

THREE

# AUTHORS & LICENSE

CHAPTER

FOUR

# CHANGELOG

### 0.0.1 (14/03/21)

- First version

# INDEX