inknest

Release 1.0

Gianluca Becuzzi

Sep 01, 2021

Contents

1 Generated code description

Code description generated automatically from docstrings.

1.1 model.py

class model.**Model**(*log_prior*, *log_likelihood*, *space_bounds*)

Class to describe models

log_prior

the logarithm of the prior pdf

Type function

log_likelihood

the logarithm of the likelihood function

Type function

space_bounds

the coordinate of the two vertices of the hyperrectangle defining the bounds of the parameters

Type 2-tuple of `np.ndarray`

Note: The log_prior and logg_likelihood functions are user defined and must have **one argument only**.

They also must be capable of managing (*,*, .., space_dimension)-shaped arrays, so make sure every operation is done on the **-1 axis of input**.

If input is a single point of shape (space_dimension,) both the functions must return a float (not a (1,)-shaped array)

__init__(*log_prior*, *log_likelihood*, *space_bounds*)

Initialise the sampler.

By default the starting point of the markov chain are uniformly distributed over all space.

is_inside_bounds(*points*)

Checks if a point is inside the space bounds.

Parameters **points** (`np.ndarray`) – point to be checked. Must have shape (*,space_dim,).

Returns

True if all the coordinates lie between bounds

False if at least one is outside.

The returned array has shape (*,) = `utils.pointshape(point)`

Return type np.ndarray

`log_chi`(*points*)

Logarithm of the characteristic function of the domain. Is equivalent to

```
>>> np.log(model.is_inside_bounds(point).astype(float))
```

Parameters **`points`** (`np.ndarray`) – point to be checked. Must have shape (*,space_dim,).

Returns

0 if all the coordinates lie between bounds

`-np.inf` if at least one is outside

The returned array has shape (*,) = `utils.pointshape(point)`

Return type np.ndarray

`new_is_inside_bounds`(*points*)

Same as `is_inside_bounds`.

Shorter but slower (allegedly due to high processing time of numpy broadcasting).

`pointshape`(*x*)

`self` shorthand for `utils.pointshape(x, dim = self.space_dim)`

`model.`**`unpack_variables`**(*x*)

Helper function that performs values shapecasting.

Given a `np.ndarray` of shape `(n1,n2,--, space_dim)` returns an unpackable array of shape `(space_dim, n1,n2, --)`.

Note: if any of the n1, n2, – other dimension is equal to 1, it gets squeezed as it is an unnecessary nesting. Indeed

- `(x,y,space_dim)` -> `(space_dim,x,y)` is fine
- `(x,y,1)` ~ `(x,y)` -> `(x,y)` (1D case)

Parameters **`x`** (`np.ndarray`) – the array to be casted

Returns an unpackable array

Return type tuple

Example

It can be used to define models:

```
>>> def log_prior(x):
>>>     x1,x2,x3 = unpack_variables(x)
>>>     return x1/x2*x3
```

Warning: It may be computationally expensive. Check for improvements.

1.2 samplers.py

Module containing the samplers used in main calculations.

Since almost every sampler is defined by a markov chain, basic attributes are the model and the length of the chain.

Each sampler shoud be capable of tackling with discontinuous functions.

Since is intended to be used in nested sampling, each sampler should support likelihood constrained prior sampling (LCPS).

class samplers.AIESampler(*model*, *mcmc_length*, *nwalkers=10*, *space_scale=4*, *verbosity=0*)

The Affine-Invariant Ensemble sampler (Goodman, Weare, 2010).

After a uniform initialisation step, for each particle k selects a *pivot* particle an then proposes

$$j = k + random(0 \to n)$$
$$z \sim g(z)$$
$$y = x_j + z(x_k - x_j)$$

and then executes a MH-acceptance over y (more information at <https://msp.org/camcos/2010/5-1/camcos-v5-n1-p04-p.pdf>).

__init__(*model*, *mcmc_length*, *nwalkers=10*, *space_scale=4*, *verbosity=0*)

Initialise the chain uniformly over the space bounds.

get_stretch(*size=1*)

Generates the stretch values given the scale_parameter `a`.

Output is distibuted as $\frac{1}{\sqrt{z}}$ in $[1/a, a]$. Uses inverse transform sampling

join_chains(*burn_in=0.02*)

Joins the chains for the ensemble after removing `burn_in` % of each single_particle chain.

Parameters **burn_in** (`float, optional`) – the burn_in percentage.

Must be `burn_in` > 0 and `burn_in` < 1.

sample_prior()

Samples prior.

Returns the chain obtained

Return type np.ndarray

class samplers.**Sampler**(*model*, *mcmc_length*, *nwalkers*, *verbosity=0*)

Produces samples from model.

It is intended as a base class that has to be further defined. For generality the attribute *nwalkers* is present, but it can be one for not ensamble-based samplers.

model

Model defined as the set of (log_prior, log_likelihood , bounds)

Type *model.Model*

mcmc_lenght

the lenght of the single markov chain

Type int

nwalkers

the number of walkers the ensamble is made of

Type int

__init__(*model*, *mcmc_length*, *nwalkers*, *verbosity=0*)

Initialise the chain uniformly over the space bounds.

1.3 utils.py

utils.**logsubexp**(*x1*, *x2*)

Helper function to execute $\log\left(e^{x_1} - e^{x_2}\right)$

Parameters

- **x1** (`float`) –
- **x2** (`float`) –

utils.**logsumexp**(*arg*)

Utility to sum over log_values. Given a vector [a1,a2,a3, …] returns $\log\left(e^{a1} + e^{a2} + ...\right)$

Parameters **arg** (`np.ndarray`) – the array of values to be log-sum-exponentiated

Returns $\log\left(e^{a1} + e^{a2} + ...\right)$

Return type float

utils.**pointshape**(*x*, *dim=None*)

Gives the shape of an array of points of dimension `space_dim`. Basically pops the last item of `x.shape` and checks whether it's fine.

Parameters

- **x** (`np.ndarray`) –
- **dim** (`int`, optional) – the space dimension

Returns the shape of x considering last axis made of ()-shaped items.

Return type tuple

Python Module Index