Py-Bussilab Documentation

Contents

Module `bussilab`

A package collecting a heterogeneous set of tools.

This package collects a number of tools that are useful enough to be distributed but too small to deserve being published as separate packages. Source code is on GitHub[1]. Submodules are listed at the end of the current page. Command-line tools are described at this page[2]. A test pdf manual is available here[3]. This is the documentation for version 0.0.48.

Install

This package is only compatible with Python >=3.6 (no compatibility with Python 2!). The recommended way to install this package depends on how you prefer to manage your python dependencies.

pip. If you manage your dependencies with pip and install packages in your home, use:

```
pip install --user bussilab
# make sure the user installed packages can be imported, or add this to your python path
export PYTHONPATH="$(python -c 'import site; print(site.USER_SITE)'):$PYTHONPATH"
# make sure the bussilab script is your execution the path, or add this to your shell path
export PATH="$(python -c 'import site; print(site.USER_BASE + "/bin")'):$PATH"
```

Required packages will be downloaded and installed automatically in your home.

pip + venv. If you manage your dependencies with pip and work in a virtual environment[4], use:

```
pip install bussilab
```

Required packages will be downloaded and installed automatically in the virtual environment.

conda. If you manage your dependencies with conda, use:

```
conda install -c conda-forge -c bussilab py-bussilab
```

[1] https://github.com/bussilab/py-bussilab
[2] cli_documentation.html
[3] ../bussilab.pdf
[4] https://docs.python.org/3/library/venv.html

Required packages will be downloaded and installed automatically in the active conda environment.

macports. If you manage your dependencies with macports you might prefer to install required packages first. Since the list of requirements might change, it is recommended to use the bussilab package itself to obtain the list of requirements. You can do it as follows: [5]

```
# install pip and setuptools first
sudo port install py39-pip py39-setuptools
# install a bare version of the package, without dependencies
pip-3.9 install --user --no-deps bussilab
# make sure the user installed packages can be imported, or add this to your python path
export PYTHONPATH="$(python3.9 -c 'import site; print(site.USER_SITE)'):$PYTHONPATH"
# make sure the bussilab script is your execution the path, or add this to your shell path
export PATH="$(python3.9 -c 'import site; print(site.USER_BASE + "/bin")'):$PATH"
# install the dependencies
sudo port install $(bussilab required --macports --pyver 39)
```

Notice that the list of required packages might change. It is thus recommended to run the commands above every time you update the bussilab package.

Checking the installation

Once the package is installed you should be able to import the module from the python interpreter:[6]

```
import bussilab
# this command can be used to check if all dependencies are in place:
bussilab.import_submodules()
```

You should also have access to an executable script that can be used from the command line:[7]

```
bussilab -h
# this command can be used to check if all dependencies are in place:
bussilab check --import
```

Setting autocompletion

In order to benefit from autocompletion for the executable script you should install the argcomplete[8] package (with pip, conda, or macports) and add the following command to your .bashrc file:[9]

```
eval "$(register-python-argcomplete bussilab)"
```

Getting started

Python

The bussilab module itself only contains some basic infrastructure. Most of the features are implemented in the submodules listed at the end of this page. Submodules should be explicitly imported using, e.g.:

```
from bussilab import wham
```

Check their documentation to see how to use them.

If you are using Python >=3.7, you can directly use the submodules without importing them explicitly, e.g.:

[5] Notice that on macports the name of the python executable is python3.9 and the name of the pip installer is `pip-3.9`. A different python version should work as well.

[6] If you installed the package using the `--user` option, in order to be able to import the package you will have to make sure that the directory returned by the command `python -c 'import site; print(site.USER_SITE)'` is included in your python search path. If not, you can add it to the environment variable PYTHONPATH.

[7] If you installed the package using the `--user` option, in order to be able to execute the script you will have to make sure that the directory returned by the command `python -c 'import site; print(site.USER_BASE + "/bin")'` is included in your PATH environment variable. Alternatively, if the bussilab script is not in your PATH you can run it as `python -m bussilab -h`.

[8] https://github.com/kislyuk/argcomplete

[9] If you are using macports, the command would be `eval "$(register-python-argcomplete-3.9 bussilab)"`.

```
import bussilab as bl
bl.wham.wham()
```

Examples

In the examples[10] directory you can find a number of notebooks that can be used as a source of inspiration.

Command line

In addition, the bussilab script allows to access some functionality directly from the command line, without entering python. For instance, you can execute the wham subcommand typing

```
bussilab wham
```

Check their documentation in the cli_documentation[11] submodule.

Advanced stuff

Notice that instructions below assume you are using pip. If you use conda or macports you might have to adjust the commands.

Install with no dependencies

You might want to ignore dependencies completely:

```
pip install --no-deps bussilab
```

If you proceed this way, you will still be able to import bussilab module and to execute the bussilab script, but some of the submodules might not be importable. The following command will report which submodules can be used then:

```
bussilab check --import
```

The result will depend on which of the required packages are already installed on your system.

Building documentation

You can build the documentation using pdoc3:

```
pip install pdoc3
pdoc3 -f --html -o doc/ bussilab
```

Documentation will be visible at `doc/bussilab/index.html`. This is normally not necessary, since the pre-built documentation of the latest version can be found at this link[12], but can be useful to test changes to the documentation before pushing them to GitHub.

Implementing your modifications

If you want to modify the Python source code you should download it from GitHub:

```
git clone https://github.com/bussilab/py-bussilab.git
cd py-bussilab
```

Since the module is written in pure python,[13] it can be used by just adding its path to PYTHONPATH and the bin directory to PATH:

```
export PATH="/path/to/py-bussilab/bin:$PATH"
export PYTHONPATH="/path/to/py-bussilab:$PYTHONPATH"
```

[10] ../examples
[11] cli_documentation.html
[12] https://bussilab.github.io/doc-py-bussilab
[13] This might change in the future.

Testing the code

If you modified the code, it is recommended to test that your changes did not break existing features.

You can run the tests using pytest:

```
pip install pytest
pytest
```

All tests should succeed. Notice that the tests will import the bussilab module. Thus, if you want the version modified by you to be tested (and not another version that you might have installed with pip already) you should properly set the PYTHONPATH variable as explained above.

Static types can be checked using mypy:

```
pip install mypy
mypy bussilab
```

This check should succeed.

Other static checks can be done with pyflakes:

```
pip install pyflakes
pyflakes bussilab
```

This check should succeed.

Correct code formatting can be checked using pylint:

```
pip install pylint
pylint -E bussilab
```

This check should succeed.

Running and rendering jupyter examples from the command line

The GitHub repository also contains jupyter examples. You can rerun all the jupyter examples from the command line:

```
cd examples
pip install jupyter jupyter_contrib_nbextensions matplotlib
./rerun.sh
```

You can render all the jupyter examples as html from the command line:

```
cd examples
pip install jupyter nbconvert
./render.sh
```

Sub-modules

- bussilab.ann
- bussilab.cli
- bussilab.cli_documentation
- bussilab.clustering
- bussilab.coretools
- bussilab.cron
- bussilab.jremote
- bussilab.lohman
- bussilab.maxent
- bussilab.notify
- bussilab.pip
- bussilab.potts
- bussilab.reports
- bussilab.wham

Functions

Function `describe_submodule`

```
def describe_submodule(
    module: str
) -> str
```

Return a short description of a submodule without importing it.

Parameters

`module : str` Name of the module.

Returns

str The docstring of the module. If the docstring is not present, returns an empty string. If an empy string is passed as module, the docstring of the main package is returned.

Raises

ModuleNotFoundError If the module does not exist.

Examples

```
from bussilab import describe_submodule
print(describe_submodule("lohman"))
```

Function `import_submodules`

```
def import_submodules() -> None
```

Import all the available submodules.

The main bussilab module does not explicitly import the available submodules so as not to slow down the behavior of the command line interface and to allow importing individual modules even if not all the dependencies of the other modules are installed. Use this function to import all the submodules (might take a few seconds).

Mostly for testing that the packages required by the available submodules are installed, but it can also be used to preload all the submodules (and required packages) making them load faster later.

Raises

If one or more submodules cannot be imported it will raise an Exception.

Function `list_submodules`

```
def list_submodules() -> List[str]
```

Return a list of all the available submodules.

It can be used to quickly show which submodules are available for importing.

Returns

list A list of names of available submodules.

Examples

Print the available submodules and a short description for each of them.

```
from bussilab import list_submodules, describe_submodule
for m in list_submodules():
    print(m, describe_submodule(m))
```

Function `required_conda`

```
def required_conda() -> str
```

Function `required_macports`

```
def required_macports(
    pyver=''
) -> str
```

Function `required_pip`

```
def required_pip() -> str
```

Module `bussilab.ann`

Module with artificial neural networks.

ANN can be constructed with `cuda=True`, in which case it will use cudamat.

Classes

Class `ANN`

```
class ANN(
    layers,
    random_weights=False,
    init_b=0.0,
    activation='softplus',
    cuda=False
)
```

Methods

Method `apply`

```
def apply(
    self,
    x
)
```

Method `applyVec`

```
def applyVec(
    self,
    x
)
```

Method `backward`

```
def backward(
    self,
    deriv,
    hidden
)
```

Method `backward_par`

```
def backward_par(
    self,
    deriv,
    hidden
)
```

Method `cuda_setup`

```
def cuda_setup(
    self
)
```

Method `deriv`

```
def deriv(
    self,
    x
)
```

Method `derivVec`

```
def derivVec(
    self,
    x
)
```

Method `derpar`

```
def derpar(
    self,
    x
)
```

Method `derparVec`

```
def derparVec(
    self,
    x
)
```

Method `dumpPlumed`

```
def dumpPlumed(
    self,
    path,
    style='ann',
    prefix=None,
    arguments=None
)
```

Method `forward`

```
def forward(
    self,
    x
)
```

Method `getpar`

```
def getpar(
    self
)
```

Method `setpar`

```
def setpar(
    self,
    par
)
```

Module `bussilab.cli`

Tools to implement the command line interface

This module is used internally to implement the command line interface. In addition, it can be used to run the commands that are available in the command line interface without the need to leave python:

```
from bussilab.cli import cli
cli("-h")
cli("wham -b bias") # provide the command line as a string
cli(["wham", "-b", "bias"]) # alternatively use a list
```

The documentation of all the commands can be found in the cli_documentation[14] submodule.

Notice however that these commands typically have alternative python implementations that allow you to work directly on data structures and are thus more flexible. For instance, wham()(bias), where bias is a numpy array, is often more convenient than `bussilab.cli.cli("wham -b bias")`, where bias is a file.

Functions

Function `arg`

```
def arg(
    *name,
    **kwargs
)
```

Decorator that adds an argument to a command line tool.

Parameters are passed to the parser.add_argument() function. It should be written **after** the command() decorator.

Function `cli`

```
def cli(
    arguments: str | List[str] = '',
    *,
    prog: str | None = '',
    use_argcomplete: bool = False,
    throws_on_parser_errors: bool = True
) -> int | None
```

Executes a command line tool from python.

This is the main function of this module and allows to launch all the subcommands available in the command line interface directly from python.

Parameters

`arguments` : str or list Command line arguments. If a string is passed, it is first split using shlex.split()

`prog` : str Name of the calling program. It is used to build help texts. **Mostly for internal use**.

`use_argcomplete` : bool If True, the autocomplete function of argcomplete module is called on the parser, so as to allow autocompletion in the command line tool. If argcomplete module is not installed, nothing is done and no failure is reported. **Mostly for internal use**.

[14] cli_documentation.html

throws_on_parser_errors : bool If True, in case of command line error it throws a TypeError exception. **Mostly for internal use**.

Returns

None or int If an error happens while parsing, it throws a TypeError exception, unless throws_on_parser_errors is set to false, in which ase it returns the corresponding error code. If an error happens while executing the requested command, an exception is thrown. If everything goes well, it returns None.

Function command

```
def command(
    name: str,
    help: str | None = None,
    description: str | None = None,
    **kwargs
)
```

Decorator that registers a function as a subcommand.

This decorator should be written **before** the other decorators arg(), group(), and endgroup().

Parameters

name : str Name of the subcommand (will be used on the command line)
help : str Short help message for the subcommand (one line).
description : str, optional Longer description. If not provided, it is set to a copy of help.
kwargs Other parameters are passed as is to the add_parser function of argparse.

Examples

Simple command line tool that accepts a single `--out` argument followed by a string and call the function do_something with that string as an argument.

```
from bussilab.cli import command, arg

@command("subcommand")
@arg("--out")
def myfunc(out):
    do_something(out)
```

Function endgroup

```
def endgroup(
    f: Callable | None = None
)
```

Decorator that ends a group of arguments for a command line tool.

See group().

Function group

```
def group(
    title: str | Callable | None = None,
    description: str | None = None,
    exclusive: bool | None = None,
    required: bool | None = None
)
```

Decorator that adds a group of arguments for a command line tool.

It should be written **after** the command() decorator. It should be followed by a number of arg() decorators and by a closing endgroup() decorator.

Parameters

title : str The name of the group. Can only be used for non exclusive groups.
description : str A description of the group. Can only be used for non exclusive groups.
exclusive : bool If True, the arguments belonging to this group are mutually exclusive
required : bool If True, one of the arguments at least should be passed. Can only be used for exclusive groups.

Examples

This is a simple command line tool that accepts three arguments (`-a`, `-b`, or `-c`), mutually exclusive. When ran, it will just print booleans showing if these arguments were passed.

```
from bussilab.cli import command, group, arg, endgroup

@command("doit")
@group(exclusive=True)
@arg("-a", action='store_true')
@arg("-b", action='store_true')
@arg("-c", action='store_true')
@endgroup
def check(a, b, c):
   print(a, b, c)
```

Module `bussilab.cli_documentation`

Documentation for command line tools

This module only contains the documentation of the subcommands used in the command line interface. For all the subcommands, a short help identical to that generated by the `-h` option is shown. In addition, if present, the docstring of the function is also shown here.

list

```
usage: bussilab list [-h]

List available python modules.

options:
  -h, --help  show this help message and exit
```

check

```
usage: bussilab check [-h] [--import]

Check installed features

options:
  -h, --help  show this help message and exit
  --import    check if all the submodules can be imported
```

wham

```
usage: bussilab wham [-h] -b BIAS [-o OUT] [--use-frame-weight]
                     [--traj-weight [TRAJ_WEIGHT ...]] [-T T] [-m MAXITER]
                     [-t THRESHOLD] [-v]

Perform binless WHAM

options:
```

```
  -h, --help            show this help message and exit
  -b BIAS, --bias BIAS  File containing bias potential (default: None)
  -o OUT, --out OUT     Output file with weights (default: None)
  --use-frame-weight
  --traj-weight [TRAJ_WEIGHT ...]
  -T T, --temperature T
                        system temperature in energy units (default: 1.0)
  -m MAXITER, --maxiter MAXITER
                        maximum number of iterations (default: 1000)
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for convergence (default: 1e-40)
  -v, --verbose
```

jrun

```
usage: bussilab jrun [-h] [-d] [--lab] [--port PORT] [--screen-cmd SCREEN_CMD]
                     [--screen-log SCREEN_LOG] [--python-exec PYTHON_EXEC]
                     [-S SOCKNAME] [--no-screen] [--keep-ld-library-path]
                     [--detach]

Run jupyter server

options:
  -h, --help            show this help message and exit
  -d, --dry-run         show command instead of executing it (default: False)
  --lab                 use jupyterlab (default: False)
  --port PORT           set port (default: 0)
  --screen-cmd SCREEN_CMD
                        screen command (default: screen)
  --screen-log SCREEN_LOG
                        screen logfile (no logfile by default) (default: )
  --python-exec PYTHON_EXEC
                        python executable (default: )
  -S SOCKNAME, --sockname SOCKNAME
                        screen sockname (default: (path):(port):jupyter)
  --no-screen           do not run screen (default: False)
  --keep-ld-library-path
                        (ignored, this is the default now) (default: False)
  --detach              detach screen (default: False)
```

This is a tool to run a jupyter server within a screen command. The typical usage would be

```
cd /path/to/your/notebook/dir
bussilab jrun
```

A free port is identified first (can be overridden with the `--port` option) and a jupyter server is then run inside a screen instance. You will thus have to type `CTRL+aCTRL+d` in order to detach the screen letting it run in the background.

Alternatively, you can immediately detach the screen with

```
cd /path/to/your/notebook/dir
bussilab jrun --detach
```

Notice that, since the server is run inside a screen instance, in order to visualize python outputs that has been sent directly to the terminal you should connect to the screen instance later. By default, a socket name containing the path where the server is running is used, with `/` replaced by `:`. It should thus be easy to use `screen -ls` to find the proper screen instance. ## jremote

```
usage: bussilab jremote [-h] [-d] [-l] [--port PORT] [-i INDEX]
                        [--python-exec PYTHON_EXEC] [--server-url SERVER_URL]
                        [--open-cmd OPEN_CMD]
```

```
                        server

Run jupyter client

positional arguments:
  server                server URL (e.g. giorgione.phys.sissa.it)

options:
  -h, --help            show this help message and exit
  -d, --dry-run         show command instead of executing it (default: False)
  -l, --list-only       only report a list or servers (default: False)
  --port PORT           set port (default: 0)
  -i INDEX, --index INDEX
                        choose server, by default interactive choice (default:
                        0)
  --python-exec PYTHON_EXEC
                        remote python executable (e.g. module load python3
                        python-home; python) (default: python)
  --server-url SERVER_URL
                        URL on server (default: choose interactively)
                        (default: )
  --open-cmd OPEN_CMD   open command (detected automatically by default)
                        (default: )
```

This is a tool to connect to a remote running jupyter server. The typical usage would be

```
bussilab jremote server.url
```

A list of jupyter servers running on the selected machines will be shown, and one of them can be picked typing its progressive number. In case there is a single server running, it will be opened by default.

Notice that if the name of the python executable on the server is different from plain python you can override it with `--python-exec`. You can also run other scripts before, for instance loading relevant modules:

```
bussilab jremote giorgione.phys.sissa.it --python-exec            --python-exec "module load python3
```

If you recurrently connect to the same workstation, it is convenient to write a small script like this one, call it jremote and put it in your path:

```
export PYTHONPATH=/path/to/bussilab/source
python -m bussilab jremote giorgione.phys.sissa.it --python-exec "module load python3 python-home ; 
```

Here replace python with the name of your python interpreter (might be python3.7). ## pip_upgrade_all

```
usage: bussilab pip_upgrade_all [-h] [--user]

Upgrade all packages with pip

options:
  -h, --help  show this help message and exit
  --user      install/upgrade in user location (default: False)
```

This is a tool to upgrade all your packages with pip. It is a convenient way to upgrade all the packages without the need to list them explicitly.

Warning: this uses pip, so it might not work as expected if you are working in conda.

The typical usage would be

```
bussilab pip_upgrade_all
```

If you installed packages in your home you should use

```
bussilab pip_upgrade_all --user
```

notify

```
usage: bussilab notify [-h] [-m MESSAGE]
                       [-c CHANNEL | -u UPDATE | -d DELETE | -r REPLY | -R REPLY_BROADCAST | -X REAC
                       [-f FILE] [-t TITLE] [--no-footer]
                       [--screenlog SCREENLOG]
                       [--screenlog-maxlines SCREENLOG_MAXLINES] [--type TYPE]
                       [--token TOKEN] [-q]

Send a notification to Slack

options:
  -h, --help            show this help message and exit
  -m MESSAGE, --message MESSAGE
                        message (default: None)
  -c CHANNEL, --channel CHANNEL
                        channel (check ~/.bussilabrc by default) (default:
                        None)
  -u UPDATE, --update UPDATE
                        url of the message to be updated (default: None)
  -d DELETE, --delete DELETE
                        url of the message to be deleted (default: None)
  -r REPLY, --reply REPLY
                        url of the message to be replied (default: None)
  -R REPLY_BROADCAST, --reply-broadcast REPLY_BROADCAST
                        url of the message to be broadcast-replied (default:
                        None)
  -X REACT, --react REACT
                        react to a message (default: None)
  -f FILE, --file FILE  path to a file to be uploaded (incompatible with -u
                        and -d) (default: None)
  -t TITLE, --title TITLE
                        title of the message (default: None)
  --no-footer           ignore footer (default: False)
  --screenlog SCREENLOG
                        screenlog file (default: None)
  --screenlog-maxlines SCREENLOG_MAXLINES
                        maximum number of lines in screenlog (0 means all)
                        (default: 0)
  --type TYPE           'plain_text' or 'mrkdwn' (default: mrkdwn)
  --token TOKEN         token (check ~/.bussilabrc by default (default: None)
  -q, --quiet           quiet (do not write output) (default: False)
```

This is a tool to send a notification to Slack. See the documentation of bussilab.notify. ## cron

```
usage: bussilab cron [-h] [--quick-start]
                     [--quick-start-skip-steps QUICK_START_SKIP_STEPS]
                     [--quick-start-event QUICK_START_EVENT]
                     [--cron-file CRON_FILE] [--screen-cmd SCREEN_CMD]
                     [--screen-log SCREEN_LOG] [--no-screen]
                     [--keep-ld-library-path] [-S SOCKNAME]
                     [--python-exec PYTHON_EXEC] [--detach] [--unique]
                     [--window] [--period PERIOD] [--max-times MAX_TIMES]

Run cron

options:
  -h, --help            show this help message and exit
  --quick-start         run immediately (default: False)
  --quick-start-skip-steps QUICK_START_SKIP_STEPS
```

```
                        skip steps on quick start (default: None)
  --quick-start-event QUICK_START_EVENT
                        event number for quick start (default: None)
  --cron-file CRON_FILE
                        path to cron file (default: None)
  --screen-cmd SCREEN_CMD
                        screen command (default: screen)
  --screen-log SCREEN_LOG
                        screen logfile (no logfile by default) (default: )
  --no-screen           do not run screen (default: False)
  --keep-ld-library-path
                        (ignored, this is the default now) (default: False)
  -S SOCKNAME, --sockname SOCKNAME
                        screen sockname (default: (path):cron)
  --python-exec PYTHON_EXEC
                        python executable (default: )
  --detach              detach screen (default: False)
  --unique              allow only one screen with this socket (default:
                        False)
  --window              run a new window within the same screen (default:
                        False)
  --period PERIOD       period (seconds) default read from cron file or set to
                        3600 (default: None)
  --max-times MAX_TIMES
                        maximum number of calls (default: None)
```

required

```
usage: bussilab required [-h] [--macports | --conda] [--pyver PYVER]

print requirements

options:
  -h, --help     show this help message and exit
  --macports     conda syntax (default: False)
  --conda        macports syntax (default: False)
  --pyver PYVER  pyversion (e.g. 38), for macports only (default: )
```

Module `bussilab.clustering`

Module with some clustering tools

Functions

Function `daura`

```
    def daura(
        adj,
        weights=None,
        *,
        min_size=0,
        max_clusters=None
    )
```

Clustering algorithm introduced in Daura et al, Angew. Chemie (1999).

WARNING: important fix in v0.0.39 for version with weights

Parameters

adj : array_like, square matrix adj[i,j] contains 1 (or True) if frames i and j are adjacent, 0 (or False) otherwise.
weights : array_like, optional weights[i] contains the weight of the i-th frame.
min_size : number Minimum cluster size. Clusters smaller than this size are not reported. When using weights, the cluster size is defined as the sum of the weights of the members of the cluster.
max_clusters : int Maximum number of clusters.

Example

```
import scipy.spatial.distance as distance
dist=distance.squareform(distance.pdist(trajectory))
clustering.daura(dist<0.7)
```

Function `max_clique`

```
def max_clique(
    adj,
    weights=None,
    *,
    min_size=0,
    max_clusters=None,
    use_networkit=False
)
```

Clustering algorithm used in Reisser et al, NAR (2020)[15].

Parameters

adj : array_like, square matrix adj[i,j] contains 1 (or True) if frames i and j are adjacent, 0 (or False) otherwise.
weights : array_like, optional weights[i] contains the weight of the i-th frame.
min_size : number Minimum cluster size. Clusters smaller than this size are not reported. When using weights, the cluster size is defined as the sum of the weights of the members of the cluster.
max_clusters : int Maximum number of clusters.
use_networkit : bool, optional if True, use a networkit implementation that seems to be faster. It requires python package networkit to be installed in advance!

Example

```
import scipy.spatial.distance as distance
dist=distance.squareform(distance.pdist(trajectory))
clustering.max_clique(dist<0.7)
```

Function `qt`

```
def qt(
    distances,
    cutoff,
    weights=None,
    *,
    min_size=0,
    max_clusters=None
)
```

Quality threshold clustering.

The method is explained in the original paper[16]. The implementation has been adapted from this one[17], which is also released under a GPL licence. Thus, if you use this algorithm please cite this article[18], which also discusses the important differences between this algorithm and the Daura et al algorithm in

[15] https://doi.org/10.1093/nar/gkz1184
[16] https://doi.org/10.1101/gr.9.11.1106
[17] https://github.com/rglez/QT
[18] https://doi.org/10.1021/acs.jcim.9b00558

the context of analysing molecular dynamics simulations. Additionally, mention which exact version of the bussilab package you used.

The implementation included here, at variance with the original one, allows passing weights and can be used with arbitrary metrics. In addition, it also reports clusters of size 1 (unless one passes max_clusters>1). The code is optimized when compared with the original one, and speed can be further increased by passing `np.array(distances,dtype='float32')` to distances.

WARNING: important fix in v0.0.39 for version with weights

As of version v0.0.40, clusters with the same number of members are prioritized based on their diameter (smaller diameter gets the priority). This is expected to make non-weighted calculations more reproducible, but might change some results when compared with previous versions. In addition, when growing a single candidate cluster, it two points are at the same distance from the growing cluster the one with higher weight is chosen. With these priorities, any choice where either (a) weights are float or (b) distances are float should lead to deterministing clustering irrespective of roundoff errors, assuming floats are never identical.

Parameters

`distances` : array_like, square matrix distances[i,j] contains the distance between i and j frame.
`cutoff` : number maximum distance for two frames to be included in the same cluster
`weights` : array_like, optional weights[i] contains the weight of the i-th frame.
`min_size` : number Minimum cluster size. Clusters smaller than this size are not reported. When using weights, the cluster size is defined as the sum of the weights of the members of the cluster.
`max_clusters` : int Maximum number of clusters.

Example

```
import scipy.spatial.distance as distance
dist=distance.squareform(distance.pdist(trajectory))
clustering.qt(dist,0.7)

clustering.qt(np.array(dist,dtype='float32')) # should be slightly faster
```

Classes

Class `ClusteringResult`

```
class ClusteringResult(
    *,
    method: str,
    clusters: list,
    weights: list | None
)
```

Result of a bussilab.clustering calculation.

Ancestors (in MRO)

- bussilab.coretools.Result
- builtins.dict

Instance variables

Variable `clusters` list of lists containing the members of each cluster.

Variable `method` str containing the name of the method used.

Variable `weights` list containing the weights of the clusters.

Module `bussilab.coretools`

General purpose tools.

Functions

Function `cd`

```
def cd(
    newdir: os.PathLike,
    *,
    create: bool = False
)
```

Context manager to temporarily change working directory.

Can be used to change working directory temporarily making sure that at the end of the context the working directory is restored. Notably, it also works if an exception is raised within the context.

Parameters

`newdir` : path Path to the desired directory.
`create` : bool (default False) Create directory first. If the directory exists already, no error is reported

Examples

```
from bussilab.coretools import cd
with cd("/path/to/dir"):
    do_something() # this is executed in the /path/to/dir directory
do_something_else() # this is executed in the original directory
```

Function `config`

```
def config(
    path: os.PathLike | None = None
)
```

Function `config_path`

```
def config_path(
    path: os.PathLike | None = None
)
```

Function `ensure_np_array`

```
def ensure_np_array(
    arg
) -> numpy.ndarray | None
```

Convert arg to np.array if necessary.

Function `file_or_path`

```
def file_or_path(
    arg,
    mode: str
)
```

Convert a path to an open file object if necessary.

Function `import_numba_jit`

```
def import_numba_jit()
```

Return a numba.njit object. If import fails, return a fake jit object and emits a warning.

Currently, the returned object can only be used as @njit (no option). It might be extended to allow more jit options.

Classes

Class `Result`

```
class Result(
    *args,
    **kwargs
)
```

Base class for objects returning results.

It allows one to create a return type that is similar to those created by scipy.optimize.minimize. The string representation of such an object contains a list of attributes and values and is easy to visualize on notebooks.

Examples The simplest usage is this one:

```
from bussilab import coretools

class MytoolResult(coretools.Result):
    """Result of a mytool calculation."""
    pass

def mytool():
    a = 3
    b = "ciao"
    return MytoolResult(a=a, b=b)

m=mytool()
print(m)
```

Notice that the class variables are dynamic: any keyword argument provided in the class constructor will be processed. If you want to enforce the class attributes you should add an explicit constructor. This will also allow you to add pdoc docstrings. The recommended usage is thus:

```
from bussilab import coretools

class MytoolResult(coretools.Result):
    """Result of a mytool calculation."""
    def __init__(a, b):
        super().__init__()
        self.a = a
        """Documentation for attribute a."""
        self.b = b
        """Documentation for attribute b."""

def mytool():
    a = 3
    b = "ciao"
    return MytoolResult(a=a, b=b)

m = mytool()
print(m)
```

Ancestors (in MRO)

- builtins.dict

Descendants

- bussilab.clustering.ClusteringResult
- bussilab.maxent.MaxentResult
- bussilab.potts.InferResult
- bussilab.wham.WhamResult

Class `TestCase`

```
class TestCase(
    methodName='runTest'
)
```

Improved base class for test cases.

Extends the unittest.TestCase class with some additional assertion.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

Ancestors (in MRO)

- unittest.case.TestCase

Methods

Method `assertEqualFile`

```
def assertEqualFile(
    self,
    file1: os.PathLike,
    file2: os.PathLike | None = None
)
```

Check if two files are equal.

Parameters

`file1` : path Path to the first file
`file2` : path, optional Path to the second file. If not provided, defaults to `file1+".ref"`.

Module `bussilab.cron`

Functions

Function `cron`

```
def cron(
    *,
    quick_start: bool = False,
    quick_start_skip_steps: int = 0,
    quick_start_event: int = 0,
    cron_file: str = '',
    screen_cmd: str = 'screen',
    screen_log: str = '',
    no_screen: bool = True,
    keep_ld_library_path: bool = True,
    sockname: str = '(path):cron',
    python_exec: str = '',
```

```
    detach: bool = False,
    period: int | None = None,
    max_times: int | None = None,
    unique: bool = False,
    window: bool = False
)
```

Module `bussilab.jremote`

Module implementing tools for remote jupyter connections.

This module contains some utilities that are mostly designed to be used as command line tools. The interface of the functions defined in this module is subject to changes. One should instead use the subcommands jrun and jremote in the cli_documentation[19] submodule.

Functions

Function `find_free_port`

```
def find_free_port()
```

Returns the number of a free port.

Function `remote`

```
def remote(
    server: str,
    python_exec: str = 'python',
    dry_run: bool = False,
    list_only: bool = False,
    server_url: str = '',
    port: int = 0,
    index: int = 0,
    open_cmd: str = ''
)
```

Function `run_server`

```
def run_server(
    dry_run: bool = False,
    port: int = 0,
    screen_cmd: str = 'screen',
    screen_log: str = '',
    no_screen: bool = False,
    keep_ld_library_path: bool = True,
    python_exec: str = '',
    sockname: str = '(path):(port):jupyter',
    lab: bool = False,
    detach: bool = False
)
```

Runs a jupyter server inside a screen command.

This function is only designed to be used as a command line tool.

Module `bussilab.lohman`

Module implementing Lohman model for helicases.

[19] cli_documentation.html

Functions

Function `lohman`

```
def lohman(
    t,
    ku: float,
    kd: float,
    n: int = 1,
    boundaries: Tuple[float, float] = (0.0, 1.0)
) -> float | numpy.ndarray
```

Lohman model for helicases.

Compute the fraction of unwound helices as a function of time. See Lucius et al, Biophys J 2003[20].

Parameters

`t` : float or sequence or np.ndarray Time. If a sequence or np.ndarray is provided, the function is computed for all values and an array is returned.
`ku` : float Unwinding rate.
`kd` : float Dissociation rate.
`n` : int, optional Step size
`boundaries` : tuple with 2 elements Result is mapped to this range.

Returns

float or np.ndarray The fraction of unfolded helices at a given time t. If an array is provided for t, an array is returned containing the fractions at all the times. If boundaries is provided, the fraction is linearly mapped into the boundaries[0], boundaries[1] range.

Module `bussilab.maxent`

Tools to perform reweighting using MaxEnt.

Functions

Function `maxent`

```
def maxent(
    traj,
    reference,
    *,
    logW=None,
    maxiter: int = 1000,
    verbose: bool = False,
    lambdas=None,
    l2=None,
    l1=None,
    method: str = 'L-BFGS-B',
    regularization: Callable | None = None,
    tol: float | None = None,
    options=None,
    cuda=False
)
```

Tool that computes new weights to enforce reference values.

This tools process a an array containing the observables computed along a trajectory and returns new weights that satisfy the maximum entropy principle and so that weighted averages agree with reference values.

[20] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1303449/

Parameters

traj : array_like A 2D array (lists or tuples are internally converted to numpy arrays). traj[i,j] is j-th observable computed in the i-th frame. If traj is a CUDAMatrix object, then cudamat is used irrespectively of the bool parameter cuda.

reference : array_like A 1D array (lists or tuples are internally converted to numpy arrays) containing the reference values to be enforced. If the i-th element is a tuple or an array with 2 elements, they are interpreted as boundaries. For instance, `reference=[1.0,(2.0,3.0)]` will make sure the first observable has value 1 and the second observable is within the range (2,3). Boundaries equal to `+np.inf` or `-np.inf` can be used to imply no boundary. Notice that boundaries in the form (A,B) where both A and B are finite are implemented by adding fictitious variables in a way that is transparent to the user. Boundaries in the form (A,B) where one of A or B is finite and the other is infinite are implemented as boundaries on lambdas. Boundaries in the form (A,A) are interpreted as constraints.

logW : array_like A 1D array (lists or tuples are internally converted to numpy arrays) containing the logarithm of the a priori weight of the provided frames.

lamdbas : array_like A 1D array with initial values of lambda. A good guess will minimize faster. A typical case would be recycling the lambdas obtained with slighlty different regularization parameters.

l2 : None, float, or array_like Prefactor for L2 regularization. If None, no regularization is applied. If float, the same factor is used on all the lambdas. If it is an array, it should have length equal to len(reference).

l1 : None, float, or array_like Prefactor for L1 regularization. If None, no regularization is applied. If float, the same factor is used on all the lambdas. If it is an array, it should have length equal to len(reference).

regularization : callable or None A function that takes as argument the current lambdas and return an tuple containing the regularization function and its derivatives. For instance, passing a function defined as `def reg(x): return (0.0001*0.5*np.sum(x**2),0.0001*x)` is equivalent to passing `l2=0.0001`.

verbose : bool If True, progress informations are written on stdout.

method : str Minimization method. See documentation of scipy.optimize.minimize.

maxiter : int Maximum number of iterations

tol : float or None Tolerance for minimization. See documentation of scipy.optimize.minimize.

options : dict Arbitrary options passed to scipy.optimize.minimize.

cuda : bool or None (default False) Use cuda. If None, chosen based on the availability of the cudamat library.

Notes On Using Cuda

Note that for normal datasets the cost of transfering the traj object to the GPU dominates. It it however possible to transfer the traj object first to the GPU with `cu_traj=cm.CUDAMatrix(traj)` and then reuse it for multiple calls (e.g. for a hyper parameter scan).

Classes

Class `MaxentResult`

```
class MaxentResult(
    *,
    logW_ME: numpy.ndarray,
    lambdas: numpy.ndarray,
    averages: numpy.ndarray,
    gamma: float,
    success: bool,
    message: str,
    nfev: int,
    nit: int
)
```

Result of a maxent() calculation.

Ancestors (in MRO)

- bussilab.coretools.Result
- builtins.dict

Instance variables

Variable `averages` np.ndarray with len(reference) elements, resulting averages.

Variable `gamma` float containing the resulting likelihood Gamma.

Variable `lambdas` np.ndarray with len(reference) elements, optimized Lagrangian multipliers.

Variable `logW_ME` np.ndarray with traj.shape[0] elements, logarithms of the optimized weights.

Variable `message` str reporting the possible reason of failuer of the minimizer.

Variable `nfev` int reporting the number of function evaluations.

Variable `nit` int reporting the number of iterations in the minimization procedure.

Variable `success` bool reporting the success of the minimizer.

Module `bussilab.notify`

Module implementing Slack notifications.

This module sends notification through an App installed in the Slack workspace. Some settings are needed first for authentication. It is recommended to add a file named .bussilabrc to your home directory with the following content:

```
notify:
  token: xoxb-00000
  channel: U00000
```

The token here should be provided by the administrator of your workspace. The channel should be the Slack ID associated to your user. It can be found looking in your Slack profile. With these settings, the tool will send notifications to you by default.

Notifications can then be sent using either the command line:

```
bussilab notify --message "text here"
```

or from python:

```
from bussilab.notify import notify
notify("text here")
```

Notice that the message is optional. Even with an empty message, the footer will allow you to reconstruct from which machine and directory the message was sent from. This might be sufficient for your goal.

You can also indicate a specific channel for the notification using the channel option:

```
bussilab notify --message "text here" --channel "project-myproject"
```

or from python:

```
from bussilab.notify import notify
notify("text here", channel="project-myproject")
```

This will only work if the App has been added to the specified channel.

The following syntax can be used to upload a file:

```
bussilab notify --message "text here" --file /path/to/file
```

or from python:

```
from bussilab.notify import notify
notify("text here",file="/path/to/file")
```

The commands above will return the URL of the message. This URL can be used later to update or delete them or to post reactions:

```
url=$(bussilab notify --message "text here")
bussilab notify --update $url --message "revised message"
bussilab notify --react $url:heart

# this will remove only the reaction:
bussilab notify --delete $url:heart

# this will remove the entire message:
bussilab notify --delete $url

url=$(bussilab notify --message "text here")
```

or from python:

```
from bussilab.notify import notify
url=notify("text here")
notify("revised message", update=url)
notify(react=url+":heart")
notify(delete=url+":heart")
notify(delete=url)
```

In these cases, the channel is not needed and should not be provided. Notice that you will only be able to update or delete messages sent through the App.

Functions

Function `notify`

```
def notify(
    message: str = '',
    channel: str = None,
    *,
    react: str = None,
    update: str = None,
    delete: str = None,
    reply: str = None,
    reply_broadcast: str = None,
    title: str = '',
    screenlog: str = '',
    screenlog_maxlines: int = 0,
    footer: bool = True,
    type: str = 'mrkdwn',
    file: str = '',
    token: str = None
)
```

Tool to send notifications to Slack.

Parameters

`message` : str A string that will form the body of the message.
`channel` : None or str The channel. By default, taken from your `~/.bussilabrc` configuration file.
`update` : None or str The URL of a message to be updated.

delete : None or str The URL of a message to be deleted. By passing a URL concatenated with the string `":name_of_reaction"` you can delete a reaction. Buy passing two comma-separated URLs you can delete both a file and the message with which it was shared.
reply : None or str The URL of a message to be replied
reply_broadcast : None or str The URL of a message to be broadcast-replied
react : None or str The URL of a message to which you want to add a reaction, followed by the string `:name_of_the_reaction`
file : None or str The path of a file to be uploaded
title : str The title of the notification.
footer : bool If True, a footer is added with current user, machine, and directory.
type : str The type of message. Can be "mrkdwn" or "plain_text".
token : None or str The token. By default, taken from your `~/.bussilabrc` configuration file.

Returns

```
str
    A string with the URL of the sent message.
    In case the <code>delete</code> keyword is used, it returns an empty
    string.
    In case a file is uploaded, it returns two comma-separated
    URLs corresponding to the message and to the file.
```

Example

```
from bussilab.notify import notify
notify("send this message")
```

See bussilab.notify for more examples.

Module `bussilab.pip`

Module implementing a small tool for installing and updating packages with pip.

Functions

Function `install`

```
def install(
    packages: str | List[str],
    *,
    upgrade: bool = False,
    user: bool = False,
    timeout: int | None = None
)
```

Install one or more packages with pip.

Install packages making sure they get installed with the currently used python interpreter.

Parameters

packages : str or list Package to be installed/upgraded. If a list is passed, multiple packages are installed/upgraded.
upgrade : bool, optional if True, run pip with `--upgrade`.
user : bool, optional if True, run pip with `--user`.

Function `upgrade_all`

```
def upgrade_all(
    user: bool = False,
    *,
    timeout: int | None = None
)
```

Upgrade all installed packages using pip.

Warning: it assumes all available packages are installed with pip.

Parameters

`user : bool, optional` if True, install/upgrade packages in with `--user` option.

Function `upgrade_self`

```
def upgrade_self(
    *,
    user: bool = False,
    timeout: int | None = None
)
```

Module `bussilab.potts`

Module containing a tool to solve Potts models by enumeration

See Model.

Classes

Class `InferResult`

```
class InferResult(
    *,
    h: numpy.ndarray,
    J: numpy.ndarray,
    averages: numpy.ndarray,
    loglike: float,
    success: bool,
    message: str,
    nfev: int,
    nit: int
)
```

Result of a Model.infer() calculation.

Ancestors (in MRO)

- bussilab.coretools.Result
- builtins.dict

Instance variables

Variable `J` np.ndarray, optimized h.

Variable `averages` np.ndarray, resulting averages.

Variable `h` np.ndarray, optimized h.

Variable `loglike` float containing the resulting likelihood Gamma.

Variable `message` str reporting the possible reason of failuer of the minimizer.

Variable `nfev` int reporting the number of function evaluations.

Variable `nit` int reporting the number of iterations in the minimization procedure.

Variable `success` bool reporting the success of the minimizer.

Class `Model`

```
class Model(
    size: int,
    colors: int = 1,
    shifted: bool = False,
    fullmatrix: bool = True
)
```

Init model. size: number of spins colors: number of colors fullmatrix: set to False to use less memory (slower)

Methods

Method `compute`

```
def compute(
    self,
    h: numpy.ndarray,
    J: numpy.ndarray
)
```

Compute averages <sigma_i,sigma_j> for a coupling matrix J. Returns (a,b) with a=free energy and b=averages

Method `draw`

```
def draw(
    self,
    h: numpy.ndarray,
    J: numpy.ndarray,
    n: int
)
```

Compute averages for a coupling matrix J sampling n states.

Method `fixJ`

```
def fixJ(
    self,
    J: numpy.ndarray
)
```

Method `infer`

```
def infer(
    self,
    averages: numpy.ndarray,
    nseq: int = 1,
    reg: Callable | None = None
)
```

Method `loglike`

```
def loglike(
    self,
    h: numpy.ndarray,
    J: numpy.ndarray,
    ave: numpy.ndarray
)
```

Compute -log likelihood for a coupling matrix J with averages ave. Returns (a,b) with a=-log likelihood and b=derivatives

Method `random_couplings`

```
def random_couplings(
    self,
    seed: int | None = None
)
```

Module `bussilab.reports`

Functions

Function `workstations`

```
def workstations(
    wks: List | None = None,
    short: bool = True
)
```

Module `bussilab.wham`

Module containing a WHAM implementation.

See wham().

Functions

Function `wham`

```
def wham(
    bias,
    *,
    frame_weight=None,
    traj_weight=None,
    T: float = 1.0,
    maxiter: int = 1000,
    threshold: float = 1e-20,
    verbose: bool = False,
    logZ: numpy.ndarray | None = None,
    logW: numpy.ndarray | None = None,
    normalize: bool | str = 'log',
    method: str = 'minimize',
    minimize_opt: dict | None = None
)
```

Compute weights according to binless WHAM.

The main input for this calculation is in the 2D array bias. Element bias[i, j] should contain the energy of the i-th frame computed according to the j-th Hamiltonian. Trajectories should be concatenated first, so that the total number of frames should be equal to the number of frames of each trajectory multiplied by the number of trajectories. However, it is also possible to contatenate simulations of different lengths. It is crucial however to compute the potential according to each of the employed Hamiltonian on **all the frames**, not only on those simulated using that Hamiltonian.

Notice that by default weights are normalized. It is possible to override this behavior with normalize=False. However, starting with v0.0.40, this should not be necessary. The new implementation of normalization should be numerically stable in all cases.

Bugs

Up to version 0.042, method="minimize" does not work correctly when setting traj_weights. As a consequence, results produced with v0.0.41, where this method is the default, might be incorrect. In v0.0.42 this is temporarily fixed by reverting to method="substitute" when using traj_weights. In v0.0.43 this should be finally fixed: both methods should equally work in all cases, and the default "minimize" method should require less iterations.

Combining Trajectories Of Different Length

Let's imagine three frames obtained from three Hamiltonians. Let's assume that the energy of frame i in Hamiltonian j is given by bias[i, j] defined as

```
import numpy as np
bias = np.array([[1, 10, 7],
                 [2, 9, 6],
                 [3, 8, 5]])
```

We can compute the weights with the following command:

```
np.exp(wham.wham(bias).logW)
```

```
array([0.41728571, 0.39684866, 0.18586563])
```

We now notice that the second and third columns of this matrix are equal except for a rigid shift. They thus correspond to Hamiltonians that are equivalent. We should have been able to obtain the same result saying that these frames were coming from two simulations. If we only pass the first two columns however we obtain different weights

```
np.exp(wham.wham(bias[:, 0:2]).logW)
```

```
array([0.28224026, 0.43551948, 0.28224026])
```

In order to correcly analyze these frames we should pass the information that the second Hamiltonian was used for twice the time:

```
np.exp(wham.wham(bias[:, 0:2], traj_weight=(1, 2)).logW)
```

```
array([0.41728571, 0.39684866, 0.18586563])
```

Notice that now the weights are identical to those computed in the first example.

Trusting More A Trajectory Than Another

When you concatenate trajectories, you might explicitly want to trust more a trajectory than another. For instance, two trajectories might have been accumulated with a different stride, and the reliability of each frame of the one with smaller stride should be lower. We thus would like to assign a weight to each frame that accounts for its reliability.

Consider the following command:

```
import numpy as np
np.exp(wham.wham([[3, 5],
                  [4, 4],
                  [4, 4],
                  [4, 4],
                  [4, 4],
                  [4, 4],
                  [4, 4],
                  [4, 4],
                  [4, 4],
                  [4, 4],
                  [4, 4]]).logW)
```

that results in the following weights

```
array([0.06421006, 0.09357899, 0.09357899, 0.09357899, 0.09357899,
       0.09357899, 0.09357899, 0.09357899, 0.09357899, 0.09357899,
       0.09357899])
```

Notice that all the frames except for the first one are identical. An equivalent result would have been obtained using

```
import numpy as np
np.exp(wham.wham([[3, 5],
                  [4, 4]], frame_weight=(1, 10)).logW)
```

```
array([0.06421006, 0.93578994])
```

Clearly, the weight of the second frame in this example is equal to ten times the weights of the ten corresponding frames in the previous example.

Parameters

bias : np.ndarray An array with shape (nframes, ntraj) containing the bias potential applied to each frame according to each of the Hamiltonians.

frame_weight : np.ndarray, optional An array with nframes elements. These elements should contain the reliability weight of the frames. By default, these weights are set to one.

traj_weight : np.ndarray, optional An array with ntraj elements. These elements should contain the total weight of each of the Hamiltonians. Should be used when combining trajectories of different lengths.

T : float, optional The temperature of the system. This number is just used to divide the bias array in order to make it adimensional. In case replicas are simulated at different temperatures, it is possible to pass an array here, with ntraj elements.

maxiter : int, optional Maximum number of iterations in the minimization procedure.

threshold : float, optional Threshold for the minimization procedure.

verbose : bool, optional If True, print information as the minimization proceeds.

logZ : np.ndarray, optional Array with ntraj elements. Initial value for the logarithm of the partition functions. If not provided, it is computed from the bias. Providing an initial guess that is close to the converged value (e.g. as obtained from a calculation with a limited number of frames) can speed up significantly the convergence.

logW : np.ndarray, optional Array with nframes elements. Initial value for the logarithm of the weights. If not provided, they are computed from the bias. Providing an initial guess that is close to the converged value can speed up significantly the convergence. *If logW is provided, logZ is ignored.*

normalize : bool or str, optional By default, "log", which properly normalizes weights in all cases. normalize=True or False is enabled for backward compatibility.

method : str, optional If "substitute", solve self-consistent equations by substitution. If "minimize", use a minimization as in J Chem Phys 136, 144102 (2012). Prior to version 0.0.40, the default was "substitute". Starting with version 0.0.41, the default is "minimize".

minimize_opt : dict, optional If method=="minimize", this dict can be used to pass options to scipy.minimize. Notice that by default the minimization is performed using 'L-BFGS-B'.

Classes

Class `WhamResult`

```
class WhamResult(
    *,
    logW: numpy.ndarray,
    logZ: numpy.ndarray,
    nit: int,
    nfev: int,
    eps: float
)
```

Result of a wham() calculation.

Ancestors (in MRO)

- bussilab.coretools.Result
- builtins.dict

Instance variables

Variable `eps` The final error in the iterative solution.

Variable `logW` numpy.ndarray containing the logarithm of the weight of the frames.

Variable `logZ` numpy.ndarray containing the logarithm of the partition function of each state.

Variable `nfev` The number of function evalutations (might differ from nit when using method='minimize').

Variable `nit` The number of performed iterations.
