parasweep Documentation

Release 2021.01

Eviatar Bach

Jan 06, 2021

Contents:

CHAPTER 1

parasweep

parasweep is a free and open-source Python utility for facilitating parallel parameter sweeps with computational models. Instead of requiring parameters to be passed by command-line, which can be error-prone and time-consuming, parasweep leverages the model's existing configuration files using a template system, requiring minimal code changes. After the sweep values are specified, a parallel job is dispatched for each parameter set, with support for common high-performance computing job schedulers. Post-processing is facilitated by providing a mapping between the parameter sets and the simulations.

The following paper gives a description as well as a simple example to get started: https://arxiv.org/pdf/1905.03448.pdf. Please cite it if you find parasweep useful for your project!

- Free software: MIT license
- Documentation: http://www.parasweep.io
- Code: https://github.com/eviatarbach/parasweep

1.1 Dependencies

- Python 3.6+
- xarray
- numpy
- scipy
- Mako (optional)
- drmaa-python (optional)

1.2 Credits

Developed by Eviatar Bach <eviatarbach@protonmail.com>. Special thanks to Daniel Philipps (danielphili) for a bug fix and feature suggestion.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install parasweep, run this command in your terminal:

```
$ pip install parasweep
```

This is the preferred method to install parasweep, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for parasweep can be downloaded from the GitHub repo.

You can either clone the public repository:

```
$ git clone git://github.com/eviatarbach/parasweep
```

Or download the tarball:

```
$ curl -OL https://github.com/eviatarbach/parasweep/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Examples

As a toy "simulation", in these examples we make use of the standard Unix command `cat`, which concatenates the files given to it as arguments and prints the results.

3.1 Sweeps

3.1.1 CartesianSweep

In a Cartesian sweep, all the combinations of all the parameters are used (every member in the Cartesian product of the sets of parameter values).

template.txt:

```
Hello {x}, {y}, {z}
```

Command:

```
>>> from parasweep import run_sweep, CartesianSweep
>>> mapping = run_sweep(command='cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
...                     sweep=CartesianSweep({'x': [1, 2],
...                                           'y': [3, 4, 5],
...                                           'z': [6, 7]}),
...                     verbose=False, sweep_id='test')
Hello 1, 3, 6
Hello 1, 3, 7
Hello 1, 4, 6
Hello 1, 4, 7
Hello 1, 5, 6
Hello 1, 5, 7
Hello 2, 3, 6
Hello 2, 3, 7
```

(continues on next page)

(continued from previous page)

```
Hello 2, 4, 6
Hello 2, 4, 7
Hello 2, 5, 6
Hello 2, 5, 7
```

Note: Although the commands may execute in the order shown above, this is not guaranteed since the simulations are dispatched in parallel. To run serially, the option `serial=True` can be set.

An xarray `DataArray` will be returned between the parameters and the simulation IDs, which facilitates postprocessing:

```
>>> mapping
<xarray.DataArray 'sim_id' (x: 2, y: 3, z: 2)>
array([[['test_00', 'test_01'],
        ['test_02', 'test_03'],
        ['test_04', 'test_05']],

       [['test_06', 'test_07'],
        ['test_08', 'test_09'],
        ['test_10', 'test_11']]], dtype='<U7')
Coordinates:
  * x        (x) int64 1 2
  * y        (y) int64 3 4 5
  * z        (z) int64 6 7
```

In this case, it is three-dimensional, since three parameters are being swept over.

3.1.2 FilteredCartesianSweep

There is also a filtered Cartesian sweep, allowing for a Cartesian sweep where only elements meeting satisfying some condition on the parameters are used. For example, suppose we want to sweep over `x`, `y`, and `z` but only include those parameter sets where `x > y`:

template.txt:

```
Hello {x}, {y}, {z}
```

Command:

```
>>> from parasweep import run_sweep, FilteredCartesianSweep
>>> sweep = FilteredCartesianSweep({'x': [1, 2, 3], 'y': [1, 2, 3],
...                                 'z': [4, 5]},
...                                filter_func=lambda x, y, **kwargs: x > y)
>>> mapping = run_sweep(command='cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'], sweep=sweep,
...                     verbose=False, sweep_id='test')
Hello 2, 1, 4
Hello 2, 1, 5
Hello 3, 1, 4
Hello 3, 1, 5
Hello 3, 2, 4
Hello 3, 2, 5
```

Note: Parameters that are not used in the filtering function, `z` in this case, can be ignored in `filter_func` by using the `**kwargs` argument.

In this case, the parameter mapping is a dictionary like the following:

```
>>> mapping
{'test_0': {'x': 2, 'y': 1, 'z': 4},
 'test_1': {'x': 2, 'y': 1, 'z': 5},
 'test_2': {'x': 3, 'y': 1, 'z': 4},
 'test_3': {'x': 3, 'y': 1, 'z': 5},
 'test_4': {'x': 3, 'y': 2, 'z': 4},
 'test_5': {'x': 3, 'y': 2, 'z': 5}}
```

3.1.3 SetSweep

Instead of a Cartesian sweep, specific parameter sets can be used with `SetSweep`:

template.txt:

```
Hello {x}, {y}, {z}
```

Command:

```
>>> from parasweep import run_sweep, SetSweep
>>> mapping = run_sweep(command='cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
...                     sweep=SetSweep([{'x': 2, 'y': 8, 'z': 5},
...                                     {'x': 1, 'y': -4, 'z': 9}]),
...                     verbose=False, sweep_id='test')
Hello 2, 8, 5
Hello 1, -4, 9
```

Here, as with `FilteredCartesianSweep`, the parameter mapping is a dictionary:

```
>>> mapping
{'test_0': {'x': 2, 'y': 8, 'z': 5}, 'test_1': {'x': 1, 'y': -4, 'z': 9}}
```

3.1.4 RandomSweep

There is also a random sweep, where parameters are drawn from independent random distributions.

template.txt:

```
Hello {x}, {y}
```

Command:

```
>>> import scipy.stats
>>> from parasweep import run_sweep, RandomSweep
>>> mapping = run_sweep(command='cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
```

(continues on next page)

(continued from previous page)

```
...                     sweep=RandomSweep({'x': scipy.stats.norm(),
...                                        'y': scipy.stats.uniform()},
...                                       length=3),
...                     verbose=False, sweep_id='test')
Hello 0.9533238364874957, 0.8197338171659898
Hello -1.966220661588362, 0.3213785864763252
Hello -0.057572896338656816, 0.17615488655036005
```

Here, `x` is drawn from a standard normal distribution and `y` is uniform between 0 and 1.

The parameter mapping is again a dictionary:

```
>>> mapping
{'test_0': {'x': 0.9533238364874957, 'y': 0.8197338171659898},
 'test_1': {'x': -1.966220661588362, 'y': 0.3213785864763252},
 'test_2': {'x': -0.057572896338656816, 'y': 0.17615488655036005}}
```

3.1.5 Multiple configuration files

Multiple configuration files and their corresponding templates can be used:

template1.txt:

```
Hello {x},
```

template2.txt:

```
hello again {y}
```

Command:

```
>>> from parasweep import run_sweep, CartesianSweep
>>> mapping = run_sweep(command='cat {sim_id}_1.txt {sim_id}_2.txt',
...                     configs=['{sim_id}_1.txt', '{sim_id}_2.txt'],
...                     templates=['template1.txt', 'template2.txt'],
...                     sweep=CartesianSweep({'x': [1, 2, 3],
...                                           'y': [4]}),
...                     verbose=False)
Hello 1,
hello again 4
Hello 2,
hello again 4
Hello 3,
hello again 4
```

3.1.6 Sweep IDs

Sweep IDs are used to name the mapping structure if it is saved to disk, and also in assigning simulation IDs in some cases. If it is not provided explicitly it is generated based on the current time.

template.txt:

```
Hello {x},
```

Command:

```
>>> import os
>>> from parasweep import run_sweep, CartesianSweep
>>> mapping = run_sweep(command='cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
...                     sweep=CartesianSweep({'x': [1, 2, 3]}),
...                     verbose=False, sweep_id='test_sweep')
Hello 1
Hello 2
Hello 3
>>> os.path.exists('sim_ids_test_sweep.nc')
True
```

3.2 Dispatchers

3.2.1 Number of processes

By default, the maximum number of processes run simultaneously with `SubprocessDispatcher` is equal to the number of processors on the machine. We can choose a custom number, however.

Command:

```
>>> from parasweep import run_sweep, CartesianSweep
>>> from parasweep.dispatchers import SubprocessDispatcher
>>> dispatcher = SubprocessDispatcher(max_procs=2)
```

This dispatcher should then be passed to `run_sweep` as the `dispatcher` argument.

3.2.2 DRMAA

Instead of dispatching simulations with Python's `subprocess` module, we can use the Distributed Resource Management Application API (DRMAA) to interface with a number of high-performance computing systems. The following example assumes that DRMAA and an interface to a job scheduler are installed.

template.txt:

```
Hello {x}
```

Command:

```
>>> from parasweep import run_sweep, CartesianSweep
>>> from parasweep.dispatchers import DRMAADispatcher
```

We can specify a `JobTemplate` which specifies job options for DRMAA. Here, we set errors to output to `err_test.txt`.

```
>>> from drmaa import JobTemplate
>>> jt = JobTemplate(errorPath=':err_test.txt')
```

Note: Some options specific to each job scheduler, called the native specification, may have to be set using the `job_template.nativeSpecification` attribute, the options for which can be found in the job scheduler's DRMAA interface (e.g., slurm-drmaa for Slurm and pbs-drmaa for PBS).

We set the command to print the contents of the configuration file to `stderr` (this syntax may only work on bash):

```
>>> mapping = run_sweep(command='>&2 cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
...                     sweep=CartesianSweep({'x': [1]}),
...                     verbose=False, dispatcher=DRMAADispatcher(jt))
>>> with open('err_test.txt', 'r') as err_file:
...     print(err_file.read())
Hello 1
```

3.3 Template engines

3.3.1 Formatting

The following is an example of the basic formatting that can be done with the Python formatting templates:

template.txt:

```
Hello {x:.2f}
```

Command:

```
>>> from parasweep import run_sweep, CartesianSweep
>>> mapping = run_sweep(command='cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
...                     sweep=CartesianSweep({'x': [1/3, 2/3, 3/3]}),
...                     verbose=False)
Hello 0.33
Hello 0.67
Hello 1.00
```

3.3.2 Mako templates

Mako templates provide functionality that is not available with Python formatting templates, being able to insert code within the template:

template.txt:

```
Hello ${x*10}
```

Command:

```
>>> from parasweep import run_sweep, CartesianSweep
>>> from parasweep.templates import MakoTemplate
>>> mapping = run_sweep(command='cat {sim_id}.txt',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
...                     sweep=CartesianSweep({'x': [1, 2, 3]}),
...                     verbose=False, template_engine=MakoTemplate())
Hello 10
Hello 20
Hello 30
```

3.4 Naming

3.4.1 HashNamer

In the case where many parameter sweeps are run on the same model, it may be helpful to use `HashNamer` to avoid collision of the output files.

template.txt:

```
Hello {x}
```

Command:

```
>>> from parasweep import run_sweep, CartesianSweep
>>> from parasweep.namers import HashNamer
>>> mapping = run_sweep(command='echo {sim_id}',
...                     configs=['{sim_id}.txt'],
...                     templates=['template.txt'],
...                     sweep=CartesianSweep({'x': [1, 2, 3]}),
...                     namer=HashNamer(), verbose=False)
16bcb7a1
7e3245fa
1780e76b
```

Note: The hash for each parameter set is generated based on the parameter set itself as well as the sweep ID. Thus if the sweep IDs are different, hashes will vary between sweeps even if the parameters sets are identical. If `sweep_id` is not provided as an argument to `run_sweep` it will be generated based on the current time.

CHAPTER 4

parasweep.sweep module

Main sweep functionality.

parasweep.sweep.**run_sweep**(*command*, *configs*, *templates*, *sweep*, *namer=<parasweep.namers.SequentialNamer object>*, *dispatcher=<parasweep.dispatchers.SubprocessDispatcher object>*, *template_engine=<parasweep.templates.PythonFormatTemplate object>*, *sweep_id=None*, *excluded_sim_ids=[]*, *serial=False*, *wait=True*, *cleanup=False*, *verbose=True*, *overwrite=True*, *save_mapping=True*)

Run parameter sweeps.

Parameters

- **command** (*str*) – The command to run. Must include `{sim_id}` indicating where the simulation ID is to be inserted.
- **configs** (*list*) – List of paths indicating where the configuration files should be saved after substitution of the parameters into the templates. Each must include `{sim_id}` indicating where the simulation ID is to be inserted. Must be in the same order as `templates`.
- **templates** (*list*) – List of paths of templates to substitute parameters into. Must be in the same order as `configs`.
- **sweep** (*sweepers.Sweep instance*) – A *parasweep.sweepers.Sweep* object that specifies the type of sweep. By default, it is a Cartesian sweep.
- **namer** (*namers.Namer instance, optional*) – A *parasweep.namers.Namer* object that specifies how to assign simulation IDs. By default, assigns simulation IDs sequentially.
- **dispatcher** (*dispatchers.Dispatcher instance, optional*) – A *parasweep.dispatchers.Dispatcher* object that specifies how to run the jobs. By default, uses Python's `subprocess` module.
- **template_engine** (*templates.TemplateEngine instance, optional*) – A *parasweep.templates.TemplateEngine* object that specifies the template engine to use. By default, uses Python format strings.

- **sweep_id** (*str, optional*) – A name for the sweep. By default, the name is generated automatically from the date and time.
- **excluded_sim_ids** (*list, optional*) – A list of simulation IDs to exclude from the sweep.
- **serial** (*bool, optional*) – Whether to run simulations serially, i.e., to wait for each simulation to complete before executing the next one. Enabling this turns off parallelism. False by default.
- **wait** (*bool, optional*) – Whether to wait for all simulations to complete before returning. True by default.
- **cleanup** (*bool, optional*) – Whether to delete configuration files after all the simulations are done. This will cause the command to wait on all processes before returning (as with the `wait` argument). False by default.
- **verbose** (*bool, optional*) – Whether to print some information about each simulation as it is launched. True by default.
- **overwrite** (*bool, optional*) – Whether to overwrite existing files when creating configuration files. If False, a `FileExistsError` will be raised when a configuration filename coincides with an existing one in the same directory. True by default.
- **save_mapping** (*bool, optional*) – Whether to save a mapping between the parameters to the simulation IDs. The type of mapping will depend on the type of sweep (set with the `sweep` argument). The filename will be of the form `sim_ids_{sweep_id}`, with an extension depending on the mapping type. True by default.

Examples

Many examples are provided in *Examples*.

CHAPTER 5

parasweep package

5.1 Submodules

5.1.1 parasweep.dispatchers module

Dispatchers for running parallel jobs.

class parasweep.dispatchers.**Dispatcher**
Bases: abc.ABC

Abstract base class for dispatchers.

Subclasses should implement the `initialize_session`, `dispatch`, and `wait_all` methods.

initialize_session()
Must be called before dispatching jobs.

dispatch(*command*, *wait*)
Dispatch a command using the dispatcher.

Parameters

- **command** (*str*) – The command to dispatch.
- **wait** (*bool*) – Whether to wait for the process to finish before returning.

wait_all()
Wait for the running/scheduled processes to finish, then return.

class parasweep.dispatchers.**SubprocessDispatcher**(*max_procs=None*)
Bases: *parasweep.dispatchers.Dispatcher*

Dispatcher using subprocesses.

Parameters **max_procs** (*int, optional*) – The maximum number of processes to run simultaneously. By default, uses the number of processors on the machine (this is a good choice for CPU-bound work).

initialize_session()
Must be called before dispatching jobs.

dispatch(*command*, *wait*)
Dispatch a command using the dispatcher.

Parameters

- **command** (*str*) – The command to dispatch.
- **wait** (*bool*) – Whether to wait for the process to finish before returning.

wait_all()
Wait for the running/scheduled processes to finish, then return.

class parasweep.dispatchers.**DRMAADispatcher**(*job_template=None*)
Bases: *parasweep.dispatchers.Dispatcher*

Dispatcher for DRMAA.

Parameters **job_template** (*drmaa.JobTemplate instance, optional*) – A job template containing settings for running the jobs with the job scheduler. Documentation for the different options is available in the Python drmaa package. Some options specific to each job scheduler, called the native specification, may have to be set using the `job_template.nativeSpecification` attribute, the options for which can be found in the job scheduler's DRMAA interface (e.g., slurm-drmaa for Slurm and pbs-drmaa for PBS).

Examples

```
>>> import drmaa
>>> jt = drmaa.JobTemplate(hardWallclockTimeLimit=60)
>>> dispatcher = DRMAADispatcher(jt)
```

session = None

initialize_session()
Must be called before dispatching jobs.

dispatch(*command*, *wait*)
Dispatch a command using the dispatcher.

Parameters

- **command** (*str*) – The command to dispatch.
- **wait** (*bool*) – Whether to wait for the process to finish before returning.

wait_all()
Wait for the running/scheduled processes to finish, then return.

5.1.2 parasweep.namers module

Namers for generating simulation IDs.

class parasweep.namers.**Namer**
Bases: abc.ABC

Abstract class for assigning simulation IDs to simulation.

Only the `next` method has to be implemented.

start (*length*)
Initialize naming.

Parameters **length** (*int*) – Indicates how many total simulations are in the sweep.

generate_id (*param_set*, *sweep_id*)
Generate simulation ID for a given parameter set.

Parameters

- **param_set** (*dict*) – The parameter set
- **sweep_id** (*str*) – The sweep ID

class parasweep.namers.**SequentialNamer** (*zfill=None*, *start_at=0*)
Bases: *parasweep.namers.Namer*

Name simulations with consecutive numbers and leading zeros.

Parameters

- **zfill** (*None or int, optional*) – If provided, sets the width to which the name string is to be padded with zeros.
- **start_at** (*int, optional*) – Sets the integer to start at in the sequential naming.

Examples

```
>>> counter = SequentialNamer()
>>> counter.start(length=11)
>>> counter.generate_id({'key1': 'value1'}, '')
'00'
>>> counter.generate_id({'key2': 'value2'}, '')
'01'
>>> counter.start(length=2)
>>> counter.generate_id({'key1': 'value1'}, 'sweep_id')
'sweep_id_0'
```

start (*length*)
Initialize naming.

Parameters **length** (*int*) – Indicates how many total simulations are in the sweep.

generate_id (*param_set*, *sweep_id*)
Generate simulation ID for a given parameter set.

Parameters

- **param_set** (*dict*) – The parameter set
- **sweep_id** (*str*) – The sweep ID

class parasweep.namers.**HashNamer** (*hash_length=8*)
Bases: *parasweep.namers.Namer*

Name simulations using hashing.

Parameters **hash_length** (*int, optional*) – How many hexadecimal numbers to truncate the hash to. 6 by default.

Examples

```
>>> namer = HashNamer()
>>> namer.generate_id({'key1': 'value1'}, '')
'31fc462e'
>>> namer.generate_id({'key2': 'value2'}, '')
'9970c8f5'
```

generate_id(*param_set*, *sweep_id*)
Generate simulation ID for a given parameter set.

Parameters

- **param_set** (*dict*) – The parameter set
- **sweep_id** (*str*) – The sweep ID

class parasweep.namers.**SetNamer**(*names*)
Bases: *parasweep.namers.Namer*

Name simulations consecutively with a provided iterable.

Parameters **names** (*Iterable[str]*) – The sequence of names to assign to consecutive simulations.

Examples

```
>>> namer = SetNamer(['name1', 'name2'])
>>> namer.generate_id({'key1': 'value1'}, '')
'name1'
>>> namer.generate_id({'key2': 'value2'}, '')
'name2'
```

generate_id(*param_set*, *sweep_id*)
Generate simulation ID for a given parameter set.

Parameters

- **param_set** (*dict*) – The parameter set
- **sweep_id** (*str*) – The sweep ID

5.1.3 parasweep.sweepers module

class parasweep.sweepers.**Sweep**(**args*, ***kwargs*)
Bases: abc.ABC

Abstract class for parameter sweep types.

Sweeps must define an initialization using `__init__`, the sweep length using `__len__`, an iteration using `elements`, and a type of mapping using `mapping`.

elements()
Return the elements of the sweep.

May be lazily evaluated, depending on the type of sweep.

mapping(*sim_ids*, *sweep_id*, *save=True*)
Return a mapping between the simulation IDs and the parameter sets.

The mapping may be from simulation IDs to parameter sets or vice-versa, depending on what is convenient for the type of sweep.

Parameters

- **sim_ids** (*list*) – The simulation IDs that were assigned to each parameter set, in the same order as returned by `elements`.
- **sweep_id** (*str*) – The sweep ID
- **save** (*bool*) – Whether to save the mapping to disk. The filename should be of the form `sim_ids_{sweep_id}`, with an extension depending on the mapping type. True by default.

class parasweep.sweepers.**CartesianSweep**(*sweep_params*)
Bases: *parasweep.sweepers.Sweep*

A Cartesian product parameter sweep.

Parameters **sweep_params** (*dict*) – A dictionary containing the parameter names as keys and lists of values to sweep over as values.

elements()
Return the elements of the sweep.

May be lazily evaluated, depending on the type of sweep.

mapping(*sim_ids*, *sweep_id*, *save=True*)
Return a labelled array which maps parameters to simulation IDs.

See *parasweep.sweepers.Sweep.mapping()* for argument information. Returns a multidimensional labelled array (using xarray) which maps the parameters to the simulation IDs. The array coordinates correspond to each sweep parameter, while the values contain the simulation IDs. If `save=True`, this array will be saved as a netCDF file with the name `sim_ids_{sweep_id}.nc`.

class parasweep.sweepers.**FilteredCartesianSweep**(*sweep_params*, *filter_func*)
Bases: *parasweep.sweepers.Sweep*

A parameter sweep which uses specified parameter sets.

Parameters

- **sweep_params** (*dict*) – A dictionary containing the parameter names as keys and lists of values to sweep over as values.
- **filter_func** (*function*) – A boolean function of parameter values; only parameter sets that return true will be included in the sweep. The arguments of the function should be named after the corresponding parameters. If not all the parameters in the sweep are used in the filtering, the `**kwargs` argument should be defined, or else an error will be raised.

elements()
Return the elements of the sweep.

May be lazily evaluated, depending on the type of sweep.

mapping(*sim_ids*, *sweep_id*, *save=True*)
Return a dictionary which maps simulation IDs to parameter sets.

See *parasweep.sweepers.Sweep.mapping()* for argument information. If `save=True`, this dictionary will be saved as a JSON file with the name `sim_ids_{sweep_id}.json`.

class parasweep.sweepers.**SetSweep**(*param_sets*)
Bases: *parasweep.sweepers.Sweep*

A Cartesian product parameter sweep with filtering.

Parameters **param_sets** (*list*) – A list containing the parameter sets to use in the sweep as dictionaries.

elements()
Return the elements of the sweep.

May be lazily evaluated, depending on the type of sweep.

mapping(*sim_ids*, *sweep_id*, *save=True*)
Return a dictionary which maps simulation IDs to parameter sets.

See *parasweep.sweepers.Sweep.mapping()* for argument information. If `save=True`, this dictionary will be saved as a JSON file with the name `sim_ids_{sweep_id}.json`.

class parasweep.sweepers.**RandomSweep**(*sweep_params*, *length*, *random_state=None*)
Bases: *parasweep.sweepers.Sweep*

A random parameter sweep.

Each parameter is treated as an independent random variable with a given distribution.

Parameters

- **sweep_params** (*dict*) – A dictionary containing the parameter names as keys and SciPy random variables (i.e., instances of subclasses of `_RandomVariable`, or of `scipy.stats._distn_infrastructure.rv_generic`) as values.
- **length** (*int*) – The number of sets of random parameters to draw
- **random_state** (*numpy.random.RandomState instance, optional*) – If provided, will use the given random state in generating random numbers. By default, it uses the global random state.

elements()
Return the elements of the sweep.

May be lazily evaluated, depending on the type of sweep.

mapping(*sim_ids*, *sweep_id*, *save=True*)
Return a dictionary which maps simulation IDs to parameter sets.

See *parasweep.sweepers.Sweep.mapping()* for argument information. If `save=True`, this dictionary will be saved as a JSON file with the name `sim_ids_{sweep_id}.json`.

5.1.4 parasweep.templates module

Template engines for generating configuration files.

class parasweep.templates.**TemplateEngine**
Bases: `abc.ABC`

Abstract base class for template engines.

Subclasses should implement the `load` and `render` methods. Make sure to implement errors for providing parameters not present in the template, and for using parameters in the template that are not provided.

load(*paths*)
Load configuration templates.

Parameters **paths** (*list*) – List of paths of configuration templates to load.

render(*param_set*)
Render a configuration file with the template engine.

Parameters **param_set** (*dict*) – Dictionary with parameters and their values.

class parasweep.templates.**PythonFormatTemplate**
Bases: *parasweep.templates.TemplateEngine*

Template engine using Python's string formatting.

load(*paths*)
Load configuration templates.

Parameters **paths** (*list*) – List of paths of configuration templates to load.

render(*param_set*)
Render a configuration file with the template engine.

Parameters **param_set** (*dict*) – Dictionary with parameters and their values.

class parasweep.templates.**MakoTemplate**
Bases: *parasweep.templates.TemplateEngine*

Template engine using Mako.

load(*paths*)
Load configuration templates.

Parameters **paths** (*list*) – List of paths of configuration templates to load.

render(*param_set*)
Render a configuration file with the template engine.

Parameters **param_set** (*dict*) – Dictionary with parameters and their values.

5.2 Module contents

Top-level package for parasweep.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/eviatarbach/parasweep/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

parasweep could always use more documentation, whether as part of the official parasweep docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/eviatarbach/parasweep/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *parasweep* for local development.

1. Fork the *parasweep* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/parasweep.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv parasweep
$ cd parasweep/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests:

```
$ flake8 parasweep tests
$ python setup.py test or py.test
```

   To get flake8, just pip install it into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.

6.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_parasweep
```

CHAPTER 7

History

7.1 2021.01 (2021-01-06)

- Changing wait option to be True by default
- Minor fixes to Lorenz example

7.2 2020.10 (2020-10-27)

- Adding option to exclude simulation IDs
- Allowing empty sweep ID

7.3 2020.09 (2020-09-02)

- Incorporating sweep_id into SequentialNamer (thanks to danielphili)

7.4 2020.02 (2020-02-17)

- Fixing bug with default sweep_id on Windows (thanks to danielphili)
- Unicode support for PythonFormatTemplate

7.5 2019.02.3 (2019-02-20)

- Adding SetNamer naming

7.6 2019.02.2 (2019-02-18)

- Adding process limit for subprocess dispatching
- Adding RandomSweep sweep type
- Adding HashNamer naming
- Clarifying version dependencies
- More examples

7.7 2019.02 (2019-02-07)

- Separating sweep logic into a separate module
- Adding FilteredCartesianSweep sweep type
- Numerous documentation changes, including many more examples

7.8 2019.01 (2019-01-21)

- First release on PyPI

CHAPTER 8

Indices and tables

- genindex
- search

Python Module Index

p

Index

R

S

T

W