MEArec Documentation

Alessio Buccino

Jan 17, 2022

Contents

Python package for fast and customuzable biophysical simulation of extracellular recordings. MEArec also implements a command-line interface. From terminal just run `mearec` to show the list of available commands.

CHAPTER 1

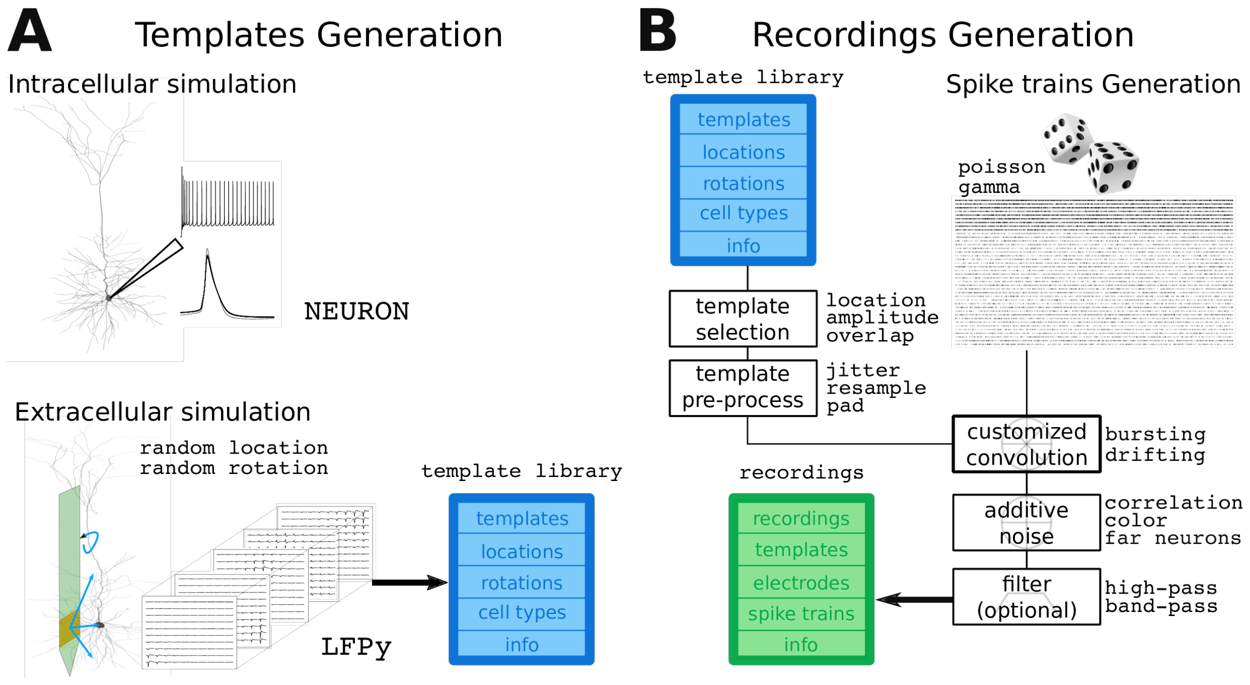
Contents

The following sections will guide you through definitions and handling of MEA models, as well as electrical stimulation and plotting functions.

1.1 Overview

MEArec is a Python package designed to make simulation of extracellular recordings easy, fast, and highly controllable.

The core idea of MEArec is explained in this figure.



The biophysical simulations of extracellular action potentials (EAP or templates) allows to generate a large number of templates from different cell models placed at random locations and with varying rotations around the probe. Further details and parameters of the template generation are explained *Generating Templates*.

When templates are simulated, extracellular recordings can be built. First, a set of spike trains is generated following user defined setting. Second, templates are chosen among the available ones and they are convoluted with the spike trains with an amplitude modulation, to ensure that the generated spike instances have physiological variability. Finally, additive noise is added to the recordings, which can also be arbitrarily filtered. Further details and parameters of the recordings generation are explained *Generating Recordings*.

The template generation is the most computationally expensive process. Recording generation is extremely fast and all aspects of the recordings can be finely controlled.

MEArec comes with a command-line-interface (CLI), which makes possible to use it outside of the Python environment and it allows for scripting for generating large number of recordings.

1.2 Installation

MEArec is a Python package and it can be easily installed using pip:

```
pip install MEArec
```

If you want to install from sources and be updated with the latest development you can install with:

```
git clone https://github.com/alejoe91/MEArec
cd MEArec
python setup.py install (or develop)
```

1.2.1 Requirements

The following are the Python requirements, which are installed when running the pip installer.

- numpy
- click
- pyyaml
- matplotlib
- neo
- elephant
- h5py
- MEAutility

1.2.2 Additional requirements for template generatiom

Installing NEURON

The template generation requires the NEURON simulator to be installed. From MEArec version 1.7.0, NEURON version 7.8 is supported. On UNIX systems NEURON can be installed with:

```
pip install neuron
```

On Windows machines, NEURON can be downloaded and installed from this link.

Installing LFPy

LFPy is used to generate extracellular templates. It is not installed by default, but it can be easily installed with:

```
pip install LFPy>=2.2
```

1.2.3 Test the installation

You can test that MEArec is correctly imported in python:

```
import MEArec as mr
```

And that the CLI is working. Open a terminal and run:

```
mearec
```

You should get the list of available commands:

```
Usage: mearec [OPTIONS] COMMAND [ARGS]...

  MEArec: Fast and customizable simulation of extracellular recordings on
  Multi-Electrode-Arrays

Options:
  --help  Show this message and exit.

Commands:
  available-probes        Print available probes.
  default-config          Print default configurations.
  gen-recordings          Generates RECORDINGS from TEMPLATES.
  gen-templates           Generates TEMPLATES with biophysical simulation.
  set-cell-models-folder  Set default cell_models folder.
  set-recordings-folder   Set default recordings output folder.
  set-recordings-params   Set default templates output folder.
  set-templates-folder    Set default templates output folder.
  set-templates-params    Set default templates output folder.
```

1.3 Command Line Interface (CLI)

MEArec implements a command line interface (CLI) to make templates and recordings generation easy to use and to allow for scripting. In order to discover the available commands, the user can use the `--help` option:

```
>> mearec --help
```

which outputs:

```
Usage: mearec [OPTIONS] COMMAND [ARGS]...

  MEArec: Fast and customizable simulation of extracellular recordings on
  Multi-Electrode-Arrays

Options:
  --help  Show this message and exit.

Commands:
  available-probes        Print available probes.
  default-config          Print default configurations.
  gen-recordings          Generates RECORDINGS from TEMPLATES.
  gen-templates           Generates TEMPLATES with biophysical simulation.
  set-cell-models-folder  Set default cell_models folder.
  set-recordings-folder   Set default recordings output folder.
  set-recordings-params   Set default recordings parameter file.
  set-templates-folder    Set default templates output folder.
  set-templates-params    Set default templates parameter file.
```

Each available command can be inspected using the texttt{-{}-help} option:

```
>> mearec "command" --help
```

A list of available probes can be found by running the `mearec available-probes` command.

1.3.1 Setting global configurations

At installation, MEArec creates a configuration folder (`.config/mearec/`) in which global settings are stored. The default paths to cell models folder, templates and recordings output folders and parameters can be set using the `set-` commands. By default, these files and folders are located in the configuration folder.

```
>> mearec default-config
```

which outputs:

```
{'cell_models_folder': path-to-cell_models,
 'recordings_folder': path-to-recordings-folder,
 'recordings_params': path-to-recordings-params.yaml,
 'templates_folder': path-to-templates-folder,
 'templates_params': path-to-templates-params.yaml}
```

At time of installation, some default files are copied in the `.config/mearec` folder including:

- a small set of 13 cell models (`.config/mearec/cell_models/bbp/`)
- default parameters for templates generation (`.config/mearec/default_params/templates_params.yaml`)
- default parameters for recordings generation (`.config/mearec/default_params/recordings_params.yaml`)

These provide the default folders to look for cell models and parameters. From the command line and Python interface both the cell models folder and all parameters related to template and recording generation can be overridden.

The `.config/mearec/recordings` and `.config/mearec/templates` are the default output folders where the templates and recordings will be saved, respectively.

The default settings can be changed with the following commands:

```
>> mearec set-cell-models-folder
>> mearec set-recordings-folder
>> mearec set-recordings-params
>> mearec set-templates-folder
>> mearec set-templates-params
```

1.4 Generating Templates

Templates refer to the extracellular action potentials (EAPs) generated by a neuron on the recording sites. Templates are generated using NEURON and LFPy packages.

The current version (1.0.4) only supports biophysical multi-compartment models from the Neocortical Microcircuit Collaboration Portal (NMC). A set of 13 cell models from layer 5 is included in the basic installation and copied in `.config/mearec/cell_models/bbp`. In order to add more cell models, you can simply download the zip files from the download page, move them to the cell model folder (which can be retrieved with the `mearec default-config` command or with the Python code: `mr.get_default_cell_models_folder()`), and unzip them. Note also custom models cane be used. In this notebook we show how to use models from the Allen database to build templates (and recordings).

Templates are generated in two steps:

1. The multi-compartment model is run with (Intracellular simulations – only needs to run once)
2. The transmembrane currents are used to generate EAPs (Extracellular simulations)

1.4.1 Intracellular simulations

For the intracellular simulation a NEURON simulation of `sim_time` (default 1 s) is performed. A constant current is applied to the soma after a `delay` (default 10 ms) so that the number of elicited spiked is in the `target_spikes` range (default [3, 50]). If the number of spikes is less or more than the target spikes boundaries, the stimulation weight is increased/decreased using the `weights` parameter (default [0.25, 0.75]). The user can also set the time period `dt` (defualt 0.03125 ms, corresponding to 32 kHz) and the `cut_out` times before and after the spike peak (default 2 ms before and 5 ms after).

The intracellular simulation will save the transmembrane currents for each cell model in the `intracellular` folder of the default output templates folder, so that currents do not need to be recomputed all the time.

Intracellular parameters summary

```
# intracellular simulation settings
sim_time: 1 # intracellular simulation time in s
target_spikes: [3, 50]  # min-max number of spikes in sim_time
cut_out: [2, 5]  # pre-post peak cut_out in ms
dt: 0.03125  # time step (2**-5) in ms
delay: 10  # stimulation delay in ms
weights: [0.25, 1.75]  # weights to multiply stimulus amplitude if number of spikes␣
↪is above (0.25) or above (1.25) target spikes
```

1.4.2 Extracellular simulations and parameters

Once the transmembrane currents are computed, they can be used to simulate extracellular action potentials using LFPy. In brief, currents are loaded to the cell model, which is randomly placed (and optionally rotated) around the selected probe before the extracellular potentials are computed. This process is performed `n` times (defualt 50). Rotations are optional and can be chosen with the `rot` parameter among `norot, physrot, 3drot`.

The `physrot` rotation is default and it is designed for L5 models of the NMC portal. This kind of rotation applies a physiological rotation to the models: for example, pyramidal cells are rotated along the z-axis (depth) and a small rendom tilt is applied (15 degrees). Some interneurons, that do not show a preferred orientation, are rotated randomly in 3d.

The `probe` parameter allows the user to choose which neural probe has to be used. Probes are handled with the MEAutility package (automatically instlled), and can be (custom designed). There is a number of pre-installed probes, such as Neuronexus-32, Neuropixels, tetrodes, and various square MEAs with varying pitches and sizes. The default probe is the Neuronexus-32 (A1x32-Poly3). The MEA probes are located on the yz plane, with an adjustable x-offset (`offset`) set to 0 μm by default.

The limits for the locations of cells can be set using the `xlim`, `ylim`, and `zlim`. If set to `null` (default for `ylim` and `zlim`), the boundary is set by the maximum and minimum electrode position in the respective axis plus the `overhang` parameter, which is 30 μm by default.

The `ncontacts` parameter can be used to simulate the spatial extent of the electrodes. For example, if set to 10, 10 points will be randomly drawn from the area of each electrode and the resulting potential is computed as the average of the 10 electric potentials of those points.

Only templates larger than the `det_thresh` parameter (30 μV by default) will be saved.

For reproducibility, the `seed` can be manually set by the user (if `null` a random seed is used).

Extracellular parameters summary

```
# extracellular simulation settings
rot: physrot # random rotation to apply to cell models (norot, physrot, 3drot)
probe: Neuronexus-32 # extracellular probe (if None probes are listed)
ncontacts: 1 # number of contacts per recording site
overhang: 30 # extension in un beyond MEA boundaries for neuron locations (if lim is␣
↪null)
offset: 0 # plane offset (um) for MEA
xlim: [10,80] # limits ( low high ) for neuron locations in the x-axis (depth)
ylim: null # limits ( low high ) for neuron locations in the y-axis
zlim: null # limits ( low high ) for neuron locations in the z-axis
det_thresh: 30 # detection threshold for EAPs
n: 50 # number of EAPs per cell model
seed: null # random seed for positions and rotations
```

1.4.3 Drifting templates

MEArec allows to generate recordings with drifting units over time. In order to do so, drifting templates have to be generated. Note that drifting recordings can be simulated ONLY from drifting templates.

To generate drifting, set the `drifting` parameter to `True`. Drifting is simulated as follows: first, an initial position is chosen so that the resulting EAP is above the detection threshold. Second, a final position is chosen so that i) the EAP is above threshold and ii) the drifting distance is between `min_drift` (defualt 20 μm) and `max_drift` defualt 100 μm. Third, the neuron is moved along the straight line connecting the initial and final position for `drift_steps` points (default 50). The `drift_x_lim`, `drift_y_lim`, and `drift_z_lim` can be used to decide the drift directions. For example, in the default case `drift_x_lim` is [-10, 10], `drift_y_lim` is [-10, 10], and `drift_z_lim` is [20, 80] and the final position will be pointing upwards in the z-direction, with some small shifts in the x- abd y-axes.

Drifting parameters summary

```
drifting: False # if True, drifting templates are simulated
max_drift: 100  # max distance from the initial and final cell position
min_drift: 30   # min distance from the initial and final cell position
drift_steps: 50 # number of drift steps
drift_x_lim: [-10, 10] # drift limits in the x-direction
drift_y_lim: [-10, 10] # drift limits in the y-direction
drift_z_lim: [20, 80]  # drift limits in the z-direction
```

1.4.4 Running template generation using CLI

Templates can be generated using the CLI with the command: `mearec gen-templates`. Run `mearec gen-templates --help` to display the list of available arguments, that can be used to overwrite the default parameters or to point to another parameter .yaml file.

The output templates are saved in .h5 format to the default templates output folder.

1.4.5 Running template generation using Python

Templates can also be generated using a Python script, or a jupyter notebook.

```
import MEArec as mr
tempgen = mr.gen_templates(cell_models_folder, params=None, templates_tmp_folder=None,
↪ intraonly=False, parallel=True,
                           recompile=False, n_jobs=None, delete_tmp=True,␣
↪verbose=False)
```

The `cell_models_folder` has to be passed as an argument. The `params` argument can be the path to a .yaml file or a dictionary containing the parameters (if None default parameters are used). The `templates_tmp_folder` points to the output temporary folder used to save generated templates. If not specified it will use the current directory. If `intraonly` is True, only the intracellular simulation is run. Simulations are run in parallel if `parallel` is True and the temporary processing folder is deleted if `delete_tmp` is True. If `n_jobs` is None, the function will use as many jobs as available cell models (if run in parallel). Finally, the `recompile` argument forces a recompilation of the models (use this if you added new cell models in the `cell_models_folder`). If `verbose` is True, the output shows the progress of the template simulation.

The `gen_templates()` function returns a gen_templates `TemplateGenerator` object (`tempgen`).

The TemplateGenerator object

The `TemplateGenerator` class contains several fields:

- templates: numpy array with (n_templates, n_electrodes, n_points) - not drifting - or (n_templates, n_drift_steps, n_electrodes, n_points) for drifting ones
- locations: (n_templates) 3D locations for the templates (for not drifting) or (n_templates, n_drift_steps) 3D locations for drifting templates.
- rotations: (n_templates) 3D rotations applied to the cell model before computing the template (for drifting templates rotation is fixed)
- celltypes: (n_templates) cell types of the generated templates
- info: dictionary with parameters used

TemplateGenerator can be saved to .h5 files as follows:

```
import MEArec as mr
mr.save_template_generator(tempgen, filename=None)
```

where tempgen is a TemplateGenerator object and filename is the output file name.

1.5 Generating Recordings

Recordings are generated combining templates and spike trains. The recordings parameters are divided in different sections:

- spiketrains
- templates
- cell-types
- recordings
- seeds

The spiketrains part deals with the generation of spike trains, while the templates, cell-types, and recordings sections specify parameters to assemble spike trains and templates and build the extracellular recordings. The seeds contains all the random seeds involved in the simulations, to ensure reproducibility.

1.5.1 Spike trains generation

The first step is the spike train generation. The user can specify the number and type of cells in 2 ways:

1. providing a list of rates and corresponding types: e.g. rates = [3, 3, 5], types = ['E', 'E', 'E'] will generate 3 spike trains with average firing rates 3, 3, and 5 Hz and respectively excitatory, excitatory , and inhibitory type. 2. providing n_exc, n_inh, f_exc, f_inh, st_exc, st_min: in this case there will be generated n_exc excitatory spike trains with average firing rate of f_exc and firing rate standard deviation of st_exc (same for inhibitory spike trains)

The firinga rates generated with the second option have a minimum firing rate of min_rate (default 0.5 Hz).

Spike trains are simulated as Poisson or Gamma processes (chosen with the parameter process) and in the latter case the gamma parameter controls the curve shape.

Spikes violating the refreactory period ref_per (default is 2 ms) are removed.

t_start (0 s by default) is the start timestamp of the recordings in second and duration will correspond to the duration of the recordings.

Spike trains parameters section summary

```
spiketrains:
  # Default parameters for spike train generation (spiketrain_gen.py)

  # spike train generation parameters

  # rates: [3,3,5] # individual spike trains rates
  # types: ['E', 'E', 'I'] # individual spike trains class (exc-inh)
  # alternative to rates - excitatory and inhibitory settings
```

(continues on next page)

(continued from previous page)

```
  n_exc: 2 # number of excitatory cells
  n_inh: 1 # number of inhibitory cells
  f_exc: 5 # average firing rate of excitatory cells in Hz
  f_inh: 15 # average firing rate of inhibitory cells in Hz
  st_exc: 1 # firing rate standard deviation of excitatory cells in Hz
  st_inh: 3 # firing rate standard deviation of inhibitory cells in Hz
  min_rate: 0.5 # minimum firing rate in Hz
  ref_per: 2 # refractory period in ms
  process: poisson # process for spike train simulation (poisson-gamma)
  gamma_shape: 2 # gamma shape (for gamma process)
  t_start: 0 # start time in s
  duration: 10 # duration in s
```

1.5.2 Recordings Generation

Specifying excitatory and inhibitory cell-types

In order to select the proper cell type (excitatory - inhibitory) the `cell-types` section of the parameters allows the user to specify which strings to look for in the cell model name (from the NMC database) to assign it to the excitatory or inhibitory set. In this example from L5 cells, all cells contining LBC (Large Basket Cells) will be marked as inhibitory, and so on. If you use custom cell models, you should overwrite this section as shown in this notebook using cell models from Allen database.

Cell-types parameters section summary

```
cell_types:
  # excitatory and inhibitory cell names
  excitatory: ['STPC', 'TTPC1', 'TTPC2', 'UTPC']
  inhibitory: ['BP', 'BTC', 'ChC', 'DBC', 'LBC', 'MC', 'NBC', 'NGC', 'SBC']
```

Template selection and parameters

Templates are selected so that they match the excitatory-inhibitory spike trains (if the `cell-types` section is provided) and they follow the following rules:

- neuron locations cannot be closer than the `min_dist` parameter (default 25 μm)
- templates must have an amplitude of at least `min_amp` (default 50 μV) and at most `max_amp`

(default 500 μV) * if specified, neuron locations are selected within the `xlim`, `ylim`, and `zlim` limits

Once the templates are selected and matched to the corresponding spike train, temporal jitter is added to them to simulate the uncertainty of the spike event within the sampling period. `n_jitters` (default is 10) templates are created by upsampling the original templates by `upsample` times (default is 8) and shifting them within a sampling period. During convolution, randomly a jittered version of the spike is selected. Finally, the templates are linearly padded on both sides (`pad_len` by default pads 3 ms before and 3 after the duration of the template) to ensure a smooth convolution.

The `overlap_threshold` allows to define spatially overlapping templates. For example, if set to 0.9 (by default) template A and template B are marked as overlapping if on the electrode with the largest peak for template A, template B's amplitude is greater or equal than the 90% of its peak amplitude.

Templates parameters section summary

```
templates:
  # recording generation parameters
  min_dist: 25 # minimum distance between neurons
  min_amp: 50 # minimum spike amplitude in uV
  max_amp: 500 # minimum spike amplitude in uV
  xlim: null # limits for neuron depths (x-coord) in um [min, max]
  ylim: null # limits for neuron depths (y-coord) in um [min, max]
  zlim: null # limits for neuron depths (z-coord) in um [min, max]
  # (e.g 0.8 -> 80% of template B on largest electrode of template A)
  n_jitters: 10 # number of temporal jittered copies for each eap
  upsample: 8 # upsampling factor to extract jittered copies
  pad_len: [3, 3] # padding of templates in ms
  overlap_threshold: 0.8 # threshold to consider two templates spatially overlapping
  seed: null # random seed to draw eap templates
```

Other recordings settings

After the templates are selected, jittered, and padded, clean recordings are generated by convolving each template with its corresponding spike train. The `fs` parameters permits to resample the recordings and if it is not provided recordings are created with the same sampling frequency as the templates. Recordings can be split in times chunks using the `chunk_duration` (20 s by default) parameter. Chunks can be processed in parallel.

If `sync_rate` is greater than 0 (and <= 1, default is 0), synchrony is added to spatially overlapping templates. For example, if `sync_rate` is 0.2, 1 out of 5 spikes on spike trains with overlapping templates will be temporally coincident. `sync_jitt` (default 1 ms) controls the jittering in time for added spikes.

The `modulation` parameter is extremely important, as it controls the variablility of the amplitude modulation: * if `modulation` id `none`, spikes are not modulated and each instance will have the same aplitude * if `modulation` id `template`, each spike event is modulated with the same amplitude for all electrodes * if `modulation` id `electrode`, each spike event is modulated with different amplitude for each electrode

For the `template` and `electrode` modulations, the amplitude is modulated as a Normal distribution with amplitude 1 and standard deviation of `sdrand` (default is 0.05).

Bursting behavior can be selected by setting `bursting` to True. The number of bursting units can be selected using the `n_bursting` parameter. By default, if bursting is used, all units are bursty. When bursting is selected, on top of the gaussian modulation the amplitude is modulated by the previous inter-spike-intervals, to simulate the amplitude decay due to bursting. In this case, the `max_burst_duration` and `n_burst_spikes` parameters control the maximum length and maximum number of spikes of a bursting event. During a bursting event, the amplitude modulation, previous to the gaussian one, is computed as:

$$mod = (\frac{avg_{ISI}/n_{consecutive}}{mem_{ISI}})^{exp}$$

where mod is the resulting amplitude modulation, avg_{ISI} is the average ISI so far during the bursting event, $n_{consecutive}$ is the number of spikes occurred in the bursting period (maximum is `n_burst_spikes`) and `exp` is the exponent of the decay (0.1 by default).

In addition to amplitude modulation, bursting can also modulate the spike shape. In order to model this, if `shape_mod` is True, then the templates are *stretched* depending on the mod value. The stretching is obtained by projecting the template on a sigmoid-transformed scale, which effectively stretches the waveform. The `shape_stretch` parameter controls the amount of stretching (default 30). Larger `shape_stretch` will result in more shape modulation, lower values in less shape modulation. The templates are stretched with the same value on all electrodes, and then, in case of an `electrode`-type modulation, the eap on each electrode to match the specific mod for the electrode. Also for an `template`-type modulation, the eap is rescaled at the template level.

Next, noise is added to to the clean recordings. Three different noise modes can be used (using the noise_mode parameter):

1. uncorrelated: additive gaussian noise (default) with a standard deviation of noise_level (10 μV by default)

2. distance-correlated: noise is generated as a multivariate normal with covariance matrix decaying with distance between electrodes. The noise_half_distance parameter is the distance for which correlation is 0.5.

3. far-neurons: noise is generated by the activity of far_neurons_n far neurons (default 300). In order to use this mode, it is recommended to generate templates with a small or null maximum amplitude. In fact, far neurons if their maximum amplitude is below far_neurons_max_amp (default 10 μV) and with an excitatory/inhibitory ratio of far_neurons_exc_inh_ratio (default 0.8). Finally, a random gaussian noise floor is added, with a standard deviation far_neurons_noise_floor times the one from the far neurons' activity, and the noise level is adjusted to match noise_level.

When selecting uncorrelated or distance-correlated, one can use the noise_color option (default is False), so that the noise spectrum is similar to biological noise. If noise_color is True, the gaussian noise is filtered with an IIR resonant filter with a peak at color_peak (default 500) and quality factor color_q (default 1). Moreover, a gaussian noise floor is added to the noise. The amplitude of the gaussian added noise is controlled by random_noise_floor (default 1), which is the percent of gaussian noise over the colored noise (when random_noise_floor=1 50% of the noise is additive gaussian. The final noise level is adjusted so that the overall standard deviation is equal to noise_level.

Finally, and optionally, the recordings can be filtered (if filter is True) with a high-pass or band-pass filter with filter_cutoff frequency(ies) ([300, 6000] by default). If filter_cutoff is a scalar, the signal is high-pass filtered. The order of the Butterworth filter can be adjusted with the filter_order frequency(ies) param.

For further analysis, spike events can be annotated as "TO" (temporal overlapping) or "SO" (spatio-temporal overlapping) when overlap is set to True. The waveforms can also be extracted and loaded to the Neo.Spiketrain object if the extract_waveforms is True. Note that this might take some time for long recordings.

Recordings parameters section summary

```
recordings:
  fs: null # sampling frequency in kHz (corresponds to dt=0.03125 ms)

  sync_rate: 0 # added synchrony rate for spatilly overlapping templates
  sync_jitt: 1 # jitter in ms for added spikes

  modulation: electrode # type of spike modulation [none (no modulation) |
    # template (each spike instance is modulated with the same value on each␣
↪electrode) |
    # electrode (each electrode is modulated separately)]
  sdrand:  0.05 # standard deviation of gaussian modulation
  bursting: True # if True, spikes are modulated in amplitude depending on the isi␣
↪and in shape (if shape_mod is True)
  exp_decay: 0.1 # with bursting modulation experimental decay in aplitude between␣
↪consecutive spikes
  n_burst_spikes: 10 # max number of 'bursting' consecutive spikes
  max_burst_duration: 100 # duration in ms of maximum burst modulation
  shape_mod: True # if True waveforms are modulated in shape with a low pass filter␣
↪depending on the isi
  shape_stretch: 30.  # min and max frequencies to be mapped to modulation value
  n_bursting: 3  # number of bursting units
  chunk_duration: 20 # chunk duration for convolution (if running into MemoryError)
```

(continues on next page)

(continued from previous page)

```
  noise_level: 0 # noise standard deviation in uV
  noise_mode: uncorrelated # [uncorrelated | distance-correlated | far-neurons]
  noise_color: False # if True noise is colored resembling experimental noise
  noise_half_distance: 30 # (distance-correlated noise) distance between electrodes␣
↪in um for which correlation is 0.5
  far_neurons_n: 300 # number of far noisy neurons to be simulated
  far_neurons_max_amp: 10 # maximum amplitude of far neurons
  far_neurons_noise_floor: 0.5 # percent of random noise
  far_neurons_exc_inh_ratio: 0.8 # excitatory / inhibitory noisy neurons ratio
  color_peak: 500 # (color) peak / curoff frequency of resonating filter
  color_q: 1 # (color) quality factor of resonating filter
  random_noise_floor: 1 # (color) additional noise floor

  filter: True # if True it filters the recordings
  filter_cutoff: [300, 6000] # filter cutoff frequencies in Hz
  filter_order: 3 # filter order

  overlap: False # if True, temporal and spatial overlap are computed for each spike␣
↪(it may be time consuming)
  extract_waveforms: False # if True, waveforms are extracted from recordings
```

Drifting recordings

When drifting templates are generated (*Drifting templates*), drifting recordings can be simulated when `drifting` is set to `True`. The `preferred_dir` parameter indicates the 3D vector with the preferred direction of drift ([0,0,1], default, is upwards in the z-direction) and the `angle_tol` (default is 15 degrees) corresponds to the tolerance in this direction. There are three types of `drift_mode`: slow, fast, and slow+fast. The different modalities vary in terms of how the drifting template is selected for each spike during the modulated convolution.

For slow drifts, a new position is calculated moving from the initial position along the drifting direction with a velocity of `slow_drift_velocity` (default 5 μm/min). If a boundary position is reached (initial or final positions), the drift direction is reversed.

For fast drifts, the user can set the frequency at which fast drift events occur (every `fast_drift_period` s, default 20 s). When a fast drift event happens, a new template position is selected randomly among the drifting templates for each drifting neuron, so that the amplitude of the new template on the channel in which the old template has the largest peak is within `fast_drift_min_jump` and `fast_drift_min_jump` (defaults 5-20). This is to ensure that fast drifts are not too abrupt.

Finally, when the slow+fast mode is selected, the two previously described modes are combined.

```
drifting: False # if True templates are drifted
drift_mode: 'slow' # drifting mode can be ['slow', 'fast', 'slow+fast']
n_drifting: null # number of drifting units
preferred_dir: [0, 0, 1]  # preferred drifting direction ([0,0,1] is positive z,␣
↪direction)
angle_tol: 15  # tolerance for direction in degrees
slow_drift_velocity: 5  # drift velocity in um/min.
fast_drift_period: 10  # period between fast drift events
fast_drift_max_jump: 20 # maximum 'jump' in um for fast drifts
fast_drift_min_jump: 5 # minimum 'jump' in um for fast drifts
t_start_drift: 0  # time in s from which drifting starts
```

Random seeds

The `seeds` section of the recording parameters contains all the random seeds for: spike train generation (`spiketrains`), template selection (`templates`), convolution operations (`convolution` - including modulation, jittering, and drifting), and noise generation (`noise`). If seeds are not set, a random seed will be generated and saved, to ensure full reproducibility of the simulations.

```
seeds:
  spiketrains: null # random seed for spiketrain generation
  templates: null # random seed for template selection
  convolution: null # random seed for jitter selection in convolution
  noise: null # random seed for noise
```

1.5.3 Running recording generation using CLI

Recordings can be generated using the CLI with the command: `mearec gen-recordings`. Run `mearec gen-recordings --help` to display the list of available arguments, that can be used to overwrite the default parameters or to point to another parameter .yaml file. In order to run a recording simulation, the `--templates` or `-t` must be given to point to the templates to be used.

The output recordings are saved in .h5 format to the default recordings output folder.

1.5.4 Running recording generation using Python

Recordings can also be generated using a Python script, or a jupyter notebook.

```
import MEArec as mr
recgen = mr.gen_recordings(params=None, templates=None, tempgen=None, n_jobs=None,␣
↪verbose=False)
```

The `params` argument can be the path to a .yaml file or a dictionary containing the parameters (if None default parameters are used). On of the `templates` or `tempgen` parameters must be indicated, the former pointing to a generated templates file, the latter instead is a `TemplateGenerator` object. The `n_jobs` argument indicates how many jobs will be used in parallel (for parallel processing, more than 1 chunks are required). If `verbose` is True, the output shows the progress of the template simulation. :code:‘verbose‘=True corresponds to :code:‘verbose‘=1. For a higher level of verbosity also :code:‘verbose‘=2 can be used.

The `gen_recordings()` function returns a gen_templates `RecordingGenerator` object (`recgen`).

The RecordingGenerator object

The `RecordingGenerator` class contains several fields:

- recordings: (n_electrodes, n_samples) recordings
- spiketrains: list of (n_spiketrains) `neo.Spiketrain` objects
- templates: (n_spiketrains, n_jitters, n_electrodes, n_templates samples) templates –

(n_spiketrains, n_drifting_steps, n_jitters, n_electrodes, n_templates samples) for drifting recordings * templates_celltypes: (n_spiketrains) templates cell type * templates_locations: (n_spiketrains, 3) templates soma locations * templates_rotations: (n_spiketrains, 3) 3d model rotations * channel_positions: (n_electrodes, 3) electrodes 3D positions * timestamps: (n_samples) timestamps in seconds (quantities) * voltage_peaks: (n_spiketrains, n_electrodes) average voltage peaks on the electrodes * spike_traces: (n_spiketrains, n_samples) clean spike trace for each spike train * info: dictionary with parameters used

`RecordingGenerator` can be saved to .h5 files as follows:

```python
import MEArec as mr
mr.save_recording_generator(recgen, filename=None)
```

where `recgen` is a `RecordingGenerator` object and `filename` is the output file name.

1.6 Using the generated data

The generated templates and recordings can be easily loaded using the MEArec API.

```python
import MEArec as mr

# load recordings
tempgen = mr.load_templates('path-to-templates.h5')

# load recordings
recgen = mr.load_recordings('path-to-recording.h5')
```

The `tempgen` is a `TemplateGenerator` objects and contains the following fields:

- templates: numpy array with (n_templates, n_electrodes, n_points) - not drifting - or (n_templates, n_drift_steps, n_electrodes, n_points) for drifting ones
- locations: (n_templates) 3D locations for the templates (for not drifting) or (n_templates, n_drift_steps) 3D locations for drifting templates.
- rotations: (n_templates) 3D rotations applied to the cell model before computing the template (for drifting templates rotation is fixed)
- celltypes: (n_templates) cell types of the generated templates
- info: dictionary with parameters used

The `recgen` is a `RecordingGenerator` objects and contains the following fields:

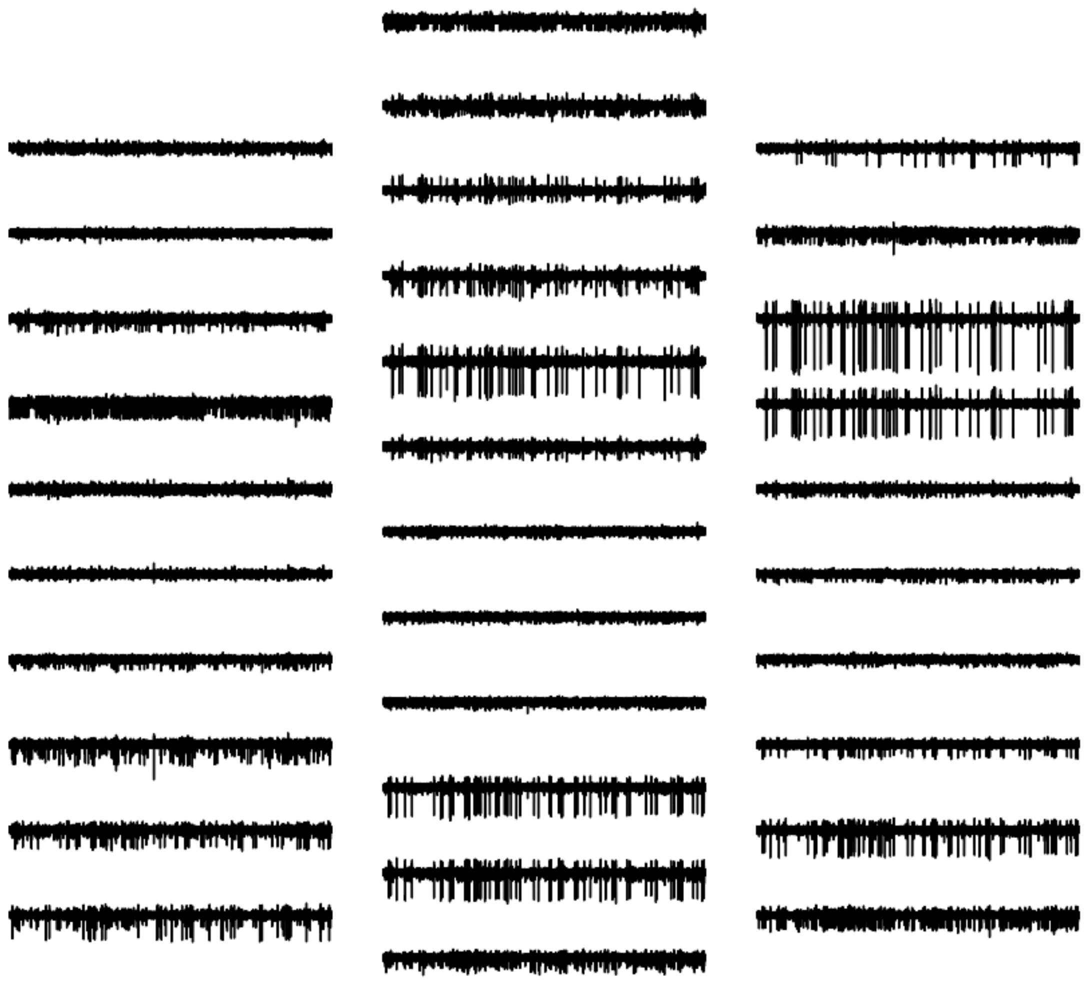
- recordings: (n_electrodes, n_samples) recordings
- spiketrains: list of (n_spiketrains) `neo.Spiketrain` objects
- templates: (n_spiketrains, n_electrodes, n_templates samples) templates
- templates_celltypes: (n_spiketrains) templates cell type
- templates_locations: (n_spiketrains, 3) templates soma locations
- templates_rotations: (n_spiketrains, 3) 3d model rotations
- channel_positions: (n_electrodes, 3) electrodes 3D positions
- timestamps: (n_samples) timestamps in seconds (quantities)
- voltage_peaks: (n_spiketrains, n_electrodes) average voltage peaks on the electrodes
- spike_traces: (n_spiketrains, n_samples) clean spike trace for each spike train
- info: dictionary with parameters used

There are several notebooks available here that show MEArec applications.

1.6.1 Plotting

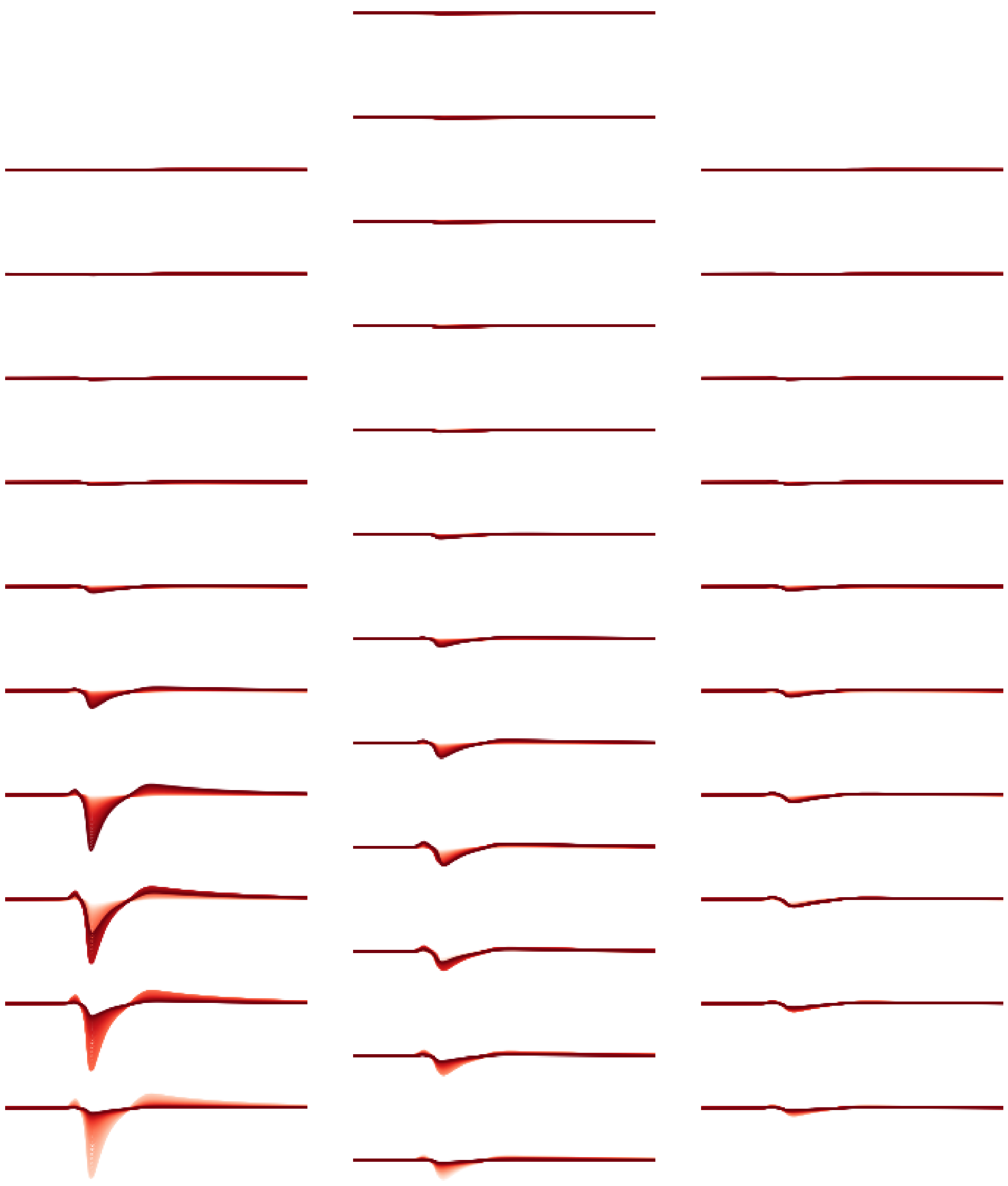
MEArec contains several plotting routines to facilitate data inspection. For example, a `RecordingGenerator` object can be plotted as follows:

```
mr.plot_recordings(recgen)
```



Similarly, templates can be plotted with the `plot_templates` function. This function also allows to plot drifting templates:

```
# load drifting templates
tempgen = mr.load_recordings('path-to-drifting-template.h5')
# drifting templates can be plotted one at a time
mr.plot_templates(tempgen, template_ids=100, drifting=True, cmap='Reds)
```



1.6.2 Integration with SpikeInterface

MEArec is designed to help validating spike sorting algorithms. Hence, its integration with the SpikeInterface project, a Python framework for spike sorting analysis, comparison, and validation, is extremely straightforward.

After installing the spikeinterface package, one can easily load a MEArec generated recording, run several spike sorting algorithms, and compare/validate their output:

```
import spikeinterface.extractors as se
import spikeinterface.sorters as ss
import spikeinterface.comparison as sc

# load recordings and spiketrains with MEArecExtractors
recording = se.MEArecRecordingExtractor('path-to-recording.h5')
sorting_GT = se.MEArecSortingExtractor('path-to-recording.h5')

# run spike sorting
sorting_MS4 = ss.mountainsort4(recording)
sorting_KS = ss.kilosort(recording)

# compare results to ground-truth
comp_MS = sc.compare_sorter_to_ground_truth(sorting_GT, sorting_MS4)
comp_KS = sc.compare_sorter_to_ground_truth(sorting_GT, sorting_MS4)

# get performance
comp_MS.get_performance()
comp_KS.get_performance()
```

More information about the SpikeInterface framework in the docs and the manuscript.

1.7 API reference

1.7.1 Module `MEArec.generators`

class `TemplateGenerator`

class `generators.`**`TemplateGenerator`**(*cell_models_folder=None, templates_folder=None, temp_dict=None, info=None, tempgen=None, params=None, intraonly=False, parallel=True, recompile=False, n_jobs=None, joblib_backend='loky', delete_tmp=True, verbose=False*)

Bases: `object`

Class for generation of templates called by the gen_templates function. The list of parameters is in default_params/templates_params.yaml.

Parameters

- **`cell_models_folder`** (*str*) – Path to folder containing Blue Brain Project cell models
- **`templates_folder`** (*str*) – Path to output template folder (if not in params)
- **`temp_dict`** (*dict*) –

  Dictionary to instantiate TemplateGenerator with existing data. It contains the following fields:

  – templates : float (n_templates, n_electrodes, n_timepoints)

  – locations : float (n_templates, 3)

  – rotations : float (n_templates, 3)

  – celltypes : str (n_templates)

- **info** (*dict*) –

  Info dictionary to instantiate TemplateGenerator with existing data. It contains the following fields:

  – params : dict with template generation parameters

  – electrodes : dict with probe info (from MEAutility.return_mea_info('probe-name'))

- **tempgen** (*TemplateGenerator*) – If a TemplateGenerator is passed, the cell types, locations, and rotations of the templates will be set using the provided templates
- **params** (*dict*) – Dictionary with parameters to simulate templates. Default values can be retrieved with mr.get_default_template_params()
- **intraonly** (*bool*) – If True, only intracellular simulations are performed
- **parallel** (*bool*) – If True, cell models are run in parallel
- **recompile** (*bool*) – If True, cell models are recompiled (suggested if new models are added)
- **n_jobs** (*int*) – If None, all cpus are used
- **delete_tmp** (*bool*) – If True, temporary files are removed
- **verbose** (*bool*) – If True, output is verbose

generate_templates()
Generate templates.

class SpikeTrainGenerator

class generators.**SpikeTrainGenerator**(*params=None*, *spiketrains=None*, *seed=None*, *verbose=False*)

Bases: object

Class for generation of spike trains called by the gen_recordings function. The list of parameters is in default_params/recordings_params.yaml (spiketrains field).

Parameters

- **params** (*dict*) – Dictionary with parameters to simulate spiketrains. Default values can be retrieved with mr.get_default_recordings_params()['spiketrains']
- **spiketrains** (*list of neo.SpikeTrain*) – List of neo.SpikeTrain objects to instantiate a SpikeTrainGenerator with existing data
- **verbose** (*bool*) – If True, output is verbose

add_synchrony(*idxs*, *rate=0.05*, *time_jitt=array(1.) * ms*, *verbose=False*)
Adds synchronous spikes between pairs of spike trains at a certain rate.

Parameters

- **idxs** (*list or array*) – Spike train indexes to add synchrony to
- **rate** (*float*) – Rate of added synchrony spike to spike train idxs[1] for each spike of idxs[0]
- **time_jitt** (*quantity*) – Maximum time jittering between added spikes
- **verbose** (*bool*) – If True output is verbose

Returns

- **sync_rate** (*float*) – New synchrony rate
- **fr1** (*quantity*) – Firing rate spike train 1
- **fr2** (*quantity*) – Firing rate spike train 2

generate_spikes()

Generate spike trains based on default_params of the SpikeTrainGenerator class. self.spiketrains contains the newly generated spike trains

set_spiketrain(*idx*, *spiketrain*)

Sets spike train idx to new spiketrain.

Parameters

- **idx** (*int*) – Index of spike train to set
- **spiketrain** (*neo.SpikeTrain*) – New spike train

class RecordingGenerator

class generators.**RecordingGenerator**(*spgen=None*, *tempgen=None*, *params=None*, *rec_dict=None*, *info=None*)

Bases: object

Class for generation of recordings called by the gen_recordings function. The list of parameters is in default_params/recordings_params.yaml.

Parameters

- **spgen** (SpikeTrainGenerator) – SpikeTrainGenerator object containing spike trains
- **tempgen** (TemplateGenerator) – TemplateGenerator object containing templates
- **params** (*dict*) – Dictionary with parameters to simulate recordings. Default values can be retrieved with mr.get_default_recording_params()
- **rec_dict** (*dict*) –

  Dictionary to instantiate RecordingGenerator with existing data. It contains the following fields:

  – recordings : float (n_electrodes, n_samples)

  – spiketrains : list of neo.SpikeTrains (n_spiketrains)

  – templates : float (n_spiketrains, 3)

  – template_locations : float (n_spiketrains, 3)

  – template_rotations : float (n_spiketrains, 3)

  – template_celltypes : str (n_spiketrains)

  – channel_positions : float (n_electrodes, 3)

  – timestamps : float (n_samples)

  – voltage_peaks : float (n_spiketrains, n_electrodes)

  – spike_traces : float (n_spiketrains, n_samples)
- **info** (*dict*) – Info dictionary to instantiate RecordingGenerator with existing data. Same fields as 'params'

annotate_overlapping_spikes(*parallel=True*)

Annnotate spike trains with overlapping information.

parallel [bool] If True, spike trains are annotated in parallel

extract_templates(*cut_out=[0.5, 2], recompute=False*)

Extract templates from spike trains.

Parameters

- **cut_out** (*float or list*) – Ms before and after peak to cut out. If float the cut is symmetric.
- **recompute** (*bool*) – If True, templates are recomputed from extracted waveforms

extract_waveforms(*cut_out=[0.5, 2]*)

Extract waveforms from spike trains and recordings.

Parameters **cut_out** (*float or list*) – Ms before and after peak to cut out. If float the cut is symmetric.

generate_recordings(*tmp_mode=None, tmp_folder=None, verbose=None, template_ids=None, n_jobs=0*)

Generates recordings

Parameters

- **tmp_mode** (*None, 'memmap'*) – Use temporary file h5 memmap or None None is no temporary file and then use memory.
- **tmp_folder** (*str or Path*) – In case of tmp files, you can specify the folder. If None, then it is automatic using tempfile.mkdtemp()
- **n_jobs** (*int*) – if >1 then use joblib to execute chunk in parallel else in loop
- **template_ids** (*list or None*) – If None, templates are selected randomly based on selection rules. If a list of indices is provided, the indices are used to select templates (template selection is bypassed)
- **verbose** (*bool or int*) – Determines the level of verbose. If 1 or True, low-level, if 2 high level, if False, not verbose

1.7.2 Module `MEArec.generation_tools`

function `gen_templates`

generation_tools.**gen_templates**(*cell_models_folder, params=None, templates_tmp_folder=None, tempgen=None, intraonly=False, parallel=True, n_jobs=None, joblib_backend='loky', recompile=False, delete_tmp=True, verbose=True*)

Parameters

- **cell_models_folder** (*str*) – path to folder containing cell models
- **params** (*str or dict*) – Path to parameters yaml file or parameters dictionary
- **templates_tmp_folder** (*str*) – Path to temporary folder where templates are temporarily saved
- **tempgen** (*TemplateGenerator*) – If a TemplateGenerator is passed, the cell types, locations, and rotations of the templates will be set using the provided templates

- **intraonly** (*bool*) – if True, only intracellular simulation is run
- **parallel** (*bool*) – if True, multi-threading is used
- **n_jobs** (*int*) – Number of jobs to run in parallel (If None all cpus are used)
- **joblib_backend** (*str*) – The joblib backend to use when n_jobs > 1 (default 'loky')
- **recompile** (*bool*) – If True, cell models are recompiled
- **delete_tmp** – if True, the temporary files are deleted
- **verbose** (*bool*) – If True, the output is verbose

Returns Generated template generator object

Return type *TemplateGenerator*

function gen_spiketrains

generation_tools.**gen_spiketrains**(*params=None*, *spiketrains=None*, *seed=None*, *verbose=False*)

Generates spike trains.

Parameters

- **params** (*str or dict*) – Path to parameters yaml file or parameters dictionary
- **spiketrains** (*list*) – List of neo.SpikeTrains (alternative to params definition)
- **verbose** (*bool*) – If True, the output is verbose

Returns Generated spike train generator object

Return type *SpikeTrainGenerator*

function gen_recordings

generation_tools.**gen_recordings**(*params=None*, *templates=None*, *tempgen=None*, *spgen=None*, *verbose=True*, *tmp_mode='memmap'*, *template_ids=None*, *tmp_folder=None*, *n_jobs=0*)

Generates recordings.

Parameters

- **templates** (*str*) – Path to generated templates
- **params** (*dict or str*) – Dictionary containing recording parameters OR path to yaml file containing parameters
- **tempgen** (TemplateGenerator) – Template generator object
- **spgen** (SpikeTrainGenerator) – Spike train generator object. If None spike trains are created from params['spiketrains']
- **verbose** (*bool or int*) – Determines the level of verbose. If 1 or True, low-level, if 2 high level, if False, not verbose
- **tmp_mode** (*None, 'h5' 'memmap'*) – Use temporary file h5 memmap or None None is no temporary file
- **template_ids** (*list or None*) – If None, templates are selected randomly based on selection rules. If a list of indices is provided, the indices are used to select templates (template selection is bypassed)

- **tmp_folder** (*str or Path*) – In case of tmp files, you can specify the folder. If None, then it is automatic using tempfile.mkdtemp()

Returns Generated recording generator object

Return type *RecordingGenerator*

1.7.3 Module `MEArec.tools`

For further details, please refer to the MEArec preprint

CHAPTER 2

Contact

If you have questions or comments, contact Alessio Buccino: alessiop.buccino@gmail.com

Python Module Index

Index