
LFPykit

Release 0.5.1

Team LFPy

Apr 12, 2023

CONTENTS:

1	LFPykit	1
1.1	Build Status	1
1.2	Features	2
1.3	Usage	3
1.4	Physical units	5
1.5	Dimensionality	5
1.6	Documentation	5
1.7	Dependencies	5
1.8	Installation	6
1.8.1	From development sources (https://github.com/LFPy/LFPykit)	6
1.8.2	Installation of stable releases on PyPI.org (https://www.pypi.org)	6
1.8.3	Installation of stable releases on conda-forge (https://conda-forge.org)	6
2	Module <code>lfpypykit</code>	7
3	class <code>CellGeometry</code>	9
4	Module <code>lfpypykit.models</code>	11
4.1	class <code>LinearModel</code>	11
4.2	class <code>CurrentDipoleMoment</code>	12
4.3	class <code>PointSourcePotential</code>	13
4.4	class <code>LineSourcePotential</code>	15
4.5	class <code>RecExtElectrode</code>	17
4.6	class <code>RecMEAElectrode</code>	22
4.7	class <code>OneSphereVolumeConductor</code>	26
4.8	class <code>LaminarCurrentSourceDensity</code>	29
4.9	class <code>VolumetricCurrentSourceDensity</code>	31
5	Module <code>lfpypykit.eegmegcalc</code>	33
5.1	class <code>eegmegcalc.FourSphereVolumeConductor</code>	33
5.2	class <code>eegmegcalc.InfiniteVolumeConductor</code>	35
5.3	class <code>eegmegcalc.InfiniteHomogeneousVolCondMEG</code>	36
5.4	class <code>eegmegcalc.SphericallySymmetricVolCondMEG</code>	39
5.5	class <code>eegmegcalc.NYHeadModel</code>	41
6	Indices and tables	45
	Bibliography	47
	Python Module Index	49

LFPYKIT

This Python module contains freestanding implementations of electrostatic forward models incorporated in LFPy (<https://github.com/LFPy/LFPy>, <https://LFPy.readthedocs.io>).

The aim of the `LFPykit` module is to provide electrostatic models in a manner that facilitates forward-model predictions of extracellular potentials and related measures from multicompartment neuron models, but without explicit dependencies on neural simulation software such as NEURON (<https://neuron.yale.edu>, <https://github.com/neuronsimulator/nrn>), Arbor (<https://arbor.readthedocs.io>, <https://github.com/arbor-sim/arbor>), or even LFPy. The `LFPykit` module can then be more easily incorporated with these simulators, or in various projects that utilize them such as LFPy (<https://LFPy.rtd.io>, <https://github.com/LFPy/LFPy>). BMTK (<https://alleninstitute.github.io/bmtk/>, <https://github.com/AllenInstitute/bmtk>), etc.

Its main functionality is providing class methods that return two-dimensional linear transformation matrices $\mathbf{M}$ between transmembrane currents $\mathbf{I}$ of multicompartment neuron models and some measurement $\mathbf{Y}$ given by $\mathbf{Y}=\mathbf{MI}$.

The presently incorporated volume conductor models have been incorporated in LFPy (<https://LFPy.rtd.io>, <https://github.com/LFPy/LFPy>), as described in various papers and books:

1. Linden H, Hagen E, Leski S, Norheim ES, Pettersen KH, Einevoll GT (2014) LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons. *Front. Neuroinform.* 7:41. doi: 10.3389/fninf.2013.00041
2. Hagen E, Næss S, Ness TV and Einevoll GT (2018) Multimodal Modeling of Neural Network Activity: Computing LFP, ECoG, EEG, and MEG Signals With LFPy 2.0. *Front. Neuroinform.* 12:92. doi: 10.3389/fninf.2018.00092
3. Ness, T. V., Chintaluri, C., Potworowski, J., Leski, S., Glabska, H., Wójcik, D. K., et al. (2015). Modelling and analysis of electrical potentials recorded in microelectrode arrays (MEAs). *Neuroinformatics* 13:403–426. doi: 10.1007/s12021-015-9265-6
4. Nunez and Srinivasan, Oxford University Press, 2006
5. Næss S, Chintaluri C, Ness TV, Dale AM, Einevoll GT and Wójcik DK (2017). Corrected Four-sphere Head Model for EEG Signals. *Front. Hum. Neurosci.* 11:490. doi: 10.3389/fnhum.2017.00490

1.1 Build Status

1.2 Features

LFPykit presently incorporates different electrostatic forward models for extracellular potentials and magnetic signals that has been derived using volume conductor theory. In volume-conductor theory the extracellular potentials can be calculated from a distance-weighted sum of contributions from transmembrane currents of neurons. Given the same transmembrane currents, the contributions to the magnetic field recorded both inside and outside the brain can also be computed.

The module presently incorporates different classes. To represent the geometry of a multicompartment neuron model we have:

- **CellGeometry**: Base class representing a multicompartment neuron geometry in terms of segment x-, y-, z-coordinates and diameter.

Different classes built to map transmembrane currents of **CellGeometry** like instances to different measurement modalities:

- **LinearModel**: Base class representing a generic forward model for subclassing
- **CurrentDipoleMoment**: Class for predicting current dipole moments
- **PointSourcePotential**: Class for predicting extracellular potentials assuming point sources and point contacts
- **LineSourcePotential**: Class for predicting extracellular potentials assuming line sources and point contacts
- **RecExtElectrode**: Class for simulations of extracellular potentials
- **RecMEAElectrode**: Class for simulations of in vitro (slice) extracellular potentials
- **OneSphereVolumeConductor**: For computing extracellular potentials within and outside a homogeneous sphere
- **LaminarCurrentSourceDensity**: For computing the ‘ground truth’ current source density across cylindrical volumes aligned with the z-axis
- **VolumetricCurrentSourceDensity**: For computing the ‘ground truth’ current source density on regularly spaced 3D grid

Different classes built to map current dipole moments (i.e., computed using **CurrentDipoleMoment**) to extracellular measurements:

- **eegmegcalc.FourSphereVolumeConductor**: For computing extracellular potentials in 4-sphere head model (brain, CSF, skull, scalp) from current dipole moment
- **eegmegcalc.InfiniteVolumeConductor**: To compute extracellular potentials in infinite volume conductor from current dipole moment

- `eegmegcalc.InfiniteHomogeneousVolCondMEG`: Class for computing magnetic field from current dipole moments under the assumption of infinite homogeneous volume conductor model
- `eegmegcalc.SphericallySymmetricVolCondMEG`: Class for computing magnetic field from current dipole moments under the assumption of a spherically symmetric volume conductor model
- `eegmegcalc.NYHeadModel`: Class for computing extracellular potentials in detailed head volume conductor model (<https://www.parralab.org/nyhead>)

Each class (except `CellGeometry`) should have a public method `get_transformation_matrix()` that returns the linear map between the transmembrane currents or current dipole moment and corresponding measurements (see Usage below)

1.3 Usage

A basic usage example using a mock 3-segment stick-like neuron, treating each segment as a point source in a linear, isotropic and homogeneous volume conductor, computing the extracellular potential in ten different locations alongside the cell geometry:

```
>>> # imports
>>> import numpy as np
>>> from lfpypykit import CellGeometry, PointSourcePotential
>>> n_seg = 3
>>> # instantiate class `CellGeometry`:
>>> cell = CellGeometry(x=np.array([[0.] * 2] * n_seg), # ( $\mu\text{m}$ )
                        y=np.array([[0.] * 2] * n_seg), # ( $\mu\text{m}$ )
                        z=np.array([[10. * x, 10. * (x + 1)]
                                   for x in range(n_seg)]), # ( $\mu\text{m}$ )
                        d=np.array([1.] * n_seg)) # ( $\mu\text{m}$ )
>>> # instantiate class `PointSourcePotential`:
>>> psp = PointSourcePotential(cell,
                               x=np.ones(10) * 10,
                               y=np.zeros(10),
                               z=np.arange(10) * 10,
                               sigma=0.3)
>>> # get linear response matrix mapping currents to measurements
>>> M = psp.get_transformation_matrix()
>>> # transmembrane currents (nA):
>>> imem = np.array([[-1., 1.],
                     [0., 0.],
                     [1., -1.]])
>>> # compute extracellular potentials (mV)
>>> V_ex = M @ imem
>>> V_ex
array([[ -0.01387397,  0.01387397],
       [ -0.00901154,  0.00901154],
       [ 0.00901154, -0.00901154],
       [ 0.01387397, -0.01387397],
       [ 0.00742668, -0.00742668],
       [ 0.00409718, -0.00409718],
       [ 0.00254212, -0.00254212],
       [ 0.00172082, -0.00172082],
       [ 0.00123933, -0.00123933],
```

(continues on next page)

(continued from previous page)

```
[ 0.00093413, -0.00093413]])
```

A basic usage example using a mock 3-segment stick-like neuron, treating each segment as a point source, computing the current dipole moment and computing the potential in ten different remote locations away from the cell geometry:

```
>>> # imports
>>> import numpy as np
>>> from lfpypykit import CellGeometry, CurrentDipoleMoment, \
>>>     eegmegcalc
>>> n_seg = 3
>>> # instantiate class `CellGeometry`:
>>> cell = CellGeometry(x=np.array([[0.] * 2] * n_seg), # (μm)
>>>                     y=np.array([[0.] * 2] * n_seg), # (μm)
>>>                     z=np.array([[10. * x, 10. * (x + 1)]
>>>                               for x in range(n_seg)]), # (μm)
>>>                     d=np.array([1.] * n_seg)) # (μm)
>>> # instantiate class `CurrentDipoleMoment`:
>>> cdp = CurrentDipoleMoment(cell)
>>> M_I_to_P = cdp.get_transformation_matrix()
>>> # instantiate class `eegmegcalc.InfiniteVolumeConductor` and map dipole moment to
>>> # extracellular potential at measurement sites
>>> ivc = eegmegcalc.InfiniteVolumeConductor(sigma=0.3)
>>> # compute linear response matrix between dipole moment and
>>> # extracellular potential
>>> M_P_to_V = ivc.get_transformation_matrix(np.c_[np.ones(10) * 1000,
>>>                                                np.zeros(10),
>>>                                                np.arange(10) * 100])
>>> # transmembrane currents (nA):
>>> imem = np.array([[-1., 1.],
>>>                  [0., 0.],
>>>                  [1., -1.]])
>>> # compute extracellular potentials (mV)
>>> V_ex = M_P_to_V @ M_I_to_P @ imem
>>> V_ex
array([[ 0.00000000e+00,  0.00000000e+00],
       [ 5.22657054e-07, -5.22657054e-07],
       [ 1.00041193e-06, -1.00041193e-06],
       [ 1.39855769e-06, -1.39855769e-06],
       [ 1.69852477e-06, -1.69852477e-06],
       [ 1.89803345e-06, -1.89803345e-06],
       [ 2.00697409e-06, -2.00697409e-06],
       [ 2.04182029e-06, -2.04182029e-06],
       [ 2.02079888e-06, -2.02079888e-06],
       [ 1.96075587e-06, -1.96075587e-06]])
```

1.4 Physical units

Notes on physical units used in LFPykit:

- There are no explicit checks for physical units
- Transmembrane currents are assumed to be in units of (nA)
- Spatial information is assumed to be in units of (μm)
- Voltages are assumed to be in units of (mV)
- Extracellular conductivities are assumed to be in units of (S/m)
- current dipole moments are assumed to be in units of (nA μm)
- Magnetic fields are assumed to be in units of (nA/ μm)

1.5 Dimensionality

- Transmembrane currents are represented by arrays with shape (n_seg, n_timesteps) where n_seg is the number of segments of the neuron model.
- Current dipole moments are represented by arrays with shape (3, n_timesteps)
- Response matrices **M** have shape (n_points, input.shape[0]) where n_points is for instance the number of extracellular recording sites and input.shape[0] the first dimension of the input; that is, the number of segments in case of transmembrane currents or 3 in case of current dipole moments.
- predicted signals (except magnetic fields using eegmegcalc.InfiniteHomogeneousVolCondMEG or eegmegcalc.SphericallySymmetricVolCondMEG) have shape (n_points, n_timesteps)

1.6 Documentation

The online Documentation of LFPykit can be found here: <https://lfpkit.readthedocs.io/en/latest>

1.7 Dependencies

LFPykit is implemented in Python and is written (and continuously tested) for Python ≥ 3.7 . The main LFPykit module depends on numpy, scipy and MEAutility (<https://github.com/alejoe91/MEAutility>, <https://meautility.readthedocs.io/en/latest/>).

Running all unit tests and example files may in addition require py.test, matplotlib, neuron (<https://www.neuron.yale.edu>), (arbor coming) and LFPy (<https://github.com/LFPy/LFPy>, <https://LFPy.readthedocs.io>).

1.8 Installation

1.8.1 From development sources (<https://github.com/LFPy/LFPykit>)

Install the current development version on <https://GitHub.com> using `git` (<https://git-scm.com>):

```
$ git clone https://github.com/LFPy/LFPykit.git
$ cd LFPykit
$ python setup.py install # --user optional
```

or using `pip`:

```
$ pip install . # --user optional
```

For active development, link the repository location

```
$ python setup.py develop # --user optional
```

1.8.2 Installation of stable releases on PyPI.org (<https://www.pypi.org>)

Installing from the Python Package Index (<https://www.pypi.org/project/lfpkit>):

```
$ pip install lfpkit # --user optional
```

To upgrade the installation using `pip`:

```
$ pip install --upgrade --no-deps lfpkit
```

1.8.3 Installation of stable releases on conda-forge (<https://conda-forge.org>)

Installing `lfpkit` from the `conda-forge` channel can be achieved by adding `conda-forge` to your channels with:

```
$ conda config --add channels conda-forge
```

Once the `conda-forge` channel has been enabled, `lfpkit` can be installed with:

```
$ conda install lfpkit
```

It is possible to list all of the versions of `lfpkit` available on your platform with:

```
$ conda search lfpkit --channel conda-forge
```

MODULE LFPYKIT

Initialization of LFPykit

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Classes

- **CellGeometry:**
Base class representing a multicompartment neuron geometry for subclassing
- **LinearModel:**
Base class representing a generic forward model for subclassing
- **CurrentDipoleMoment:**
Class for predicting current dipole moments
- **PointSourcePotential:**
Class for predicting extracellular potentials assuming point sources and contacts
- **LineSourcePotential:**
Class for predicting extracellular potentials assuming line sources and point contacts
- **RecExtElectrode:**
Class for simulations of extracellular potentials
- **RecMEAElectrode:**
Class for simulations of in vitro (slice) extracellular potentials
- **OneSphereVolumeConductor:**
For computing extracellular potentials within and outside a homogeneous sphere
- **LaminarCurrentSourceDensity:**
For computing the ground truth current source density in cylindrical volumes aligned with the z-axis.
- **VolumetricCurrentSourceDensity:**
For computing the ground truth current source density in cubic volumes with bin edges defined by x, y, z
- **eegmegcalc.FourSphereVolumeConductor:**
For computing extracellular potentials in 4-sphere model (brain, CSF, skull, scalp) from current dipole moment

- **eegmegcalc.InfiniteVolumeConductor:**
To compute extracellular potentials with current dipole moments in infinite volume conductor
- **eegmegcalc.InfiniteHomogeneousVolCondMEG:**
Class for computing magnetic field from current dipole moments assuming an infinite homogeneous volume conductor
- **eegmegcalc.SphericallySymmetricVolCondMEG:**
Class for computing magnetic field from current dipole moments assuming a spherically symmetric volume conductor
- **eegmegcalc.NYHeadModel:**
Class for computing extracellular potentials in detailed head volume conductor model (<https://www.parralab.org/nyhead>)

Modules

- cellgeometry
- models
- eegmegcalc
- lfpcalc

CLASS CELLGEOMETRY

class `lfpykit.CellGeometry(x, y, z, d)`

Bases: `object`

Base class representing the geometry of multicompartment neuron models.

Assumptions

- each compartment is piecewise linear between their start and endpoints
- each compartment has a constant diameter
- the transmembrane current density is constant along the compartment axis

Parameters

x: ndarray of floats

shape (n_seg x 2) array of start- and end-point coordinates of each compartment along x-axis in units of (μm)

y: ndarray

shape (n_seg x 2) array of start- and end-point coordinates of each compartment along y-axis in units of (μm)

z: ndarray

shape (n_seg x 2) array of start- and end-point coordinates of each compartment along z-axis in units of (μm)

d: ndarray

shape (n_seg) or shape (n_seg x 2) array of compartment diameters in units of (μm). If the 2nd axis is equal to 2, conical frusta is assumed.

Attributes

totnsegs: int

number of compartments

x: ndarray of floats

shape (totnsegs x 2) array of start- and end-point coordinates of each compartment along x-axis in units of (μm)

y: ndarray

shape (totnsegs x 2) array of start- and end-point coordinates of each compartment along y-axis in units of (μm)

z: ndarray

shape (totnsegs x 2) array of start- and end-point coordinates of each compartment along z-axis in units of (μm)

d: ndarray

shape (totnsegs) array of compartment diameters in units of (μm)

length: ndarray

length of each compartment in units of μm

area: ndarray

array of compartment surface areas in units of μm^2

MODULE LFPYKIT.MODELS

Copyright (C) 2020 Computational Neuroscience Group, NMBU.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

4.1 class LinearModel

class lfpypkit.LinearModel(*cell*)

Bases: Model

Base class that defines a 2D linear response matrix **M** between transmembrane currents **I** (nA) of a multicompartment neuron model and some measurement **Y** as

$$\mathbf{Y} = \mathbf{M}\mathbf{I}$$

LinearModel only creates a mapping that returns the currents themselves. The class is suitable as a base class for other linear model implementations, see for example class CurrentDipoleMoment or PointSourcePotential

Parameters

cell: object

lfpypkit.CellGeometry instance or similar. Can also be set to None which allows setting the attribute cell after class instantiation.

get_transformation_matrix()

Get linear response matrix

Returns

response_matrix: ndarray

shape (n_seg, n_seg) ndarray

Raises

AttributeError

if cell is None

4.2 class CurrentDipoleMoment

class lfpypykit.CurrentDipoleMoment(*cell*)

Bases: [LinearModel](#)

[LinearModel](#) subclass that defines a 2D linear response matrix **M** between transmembrane current array **I** (nA) of a multicompartment neuron model and the corresponding current dipole moment **P** (nA μm) [1] as

$$\mathbf{P} = \mathbf{M}\mathbf{I}$$

The current **I** is an ndarray of shape (n_seg, n_tsteps) with unit (nA), and the rows of **P** represent the *x*-, *y*- and *z*-components of the current dipole moment for every time step.

The current dipole moment can be used to compute distal measures of neural activity such as the EEG and MEG using `lfpypykit.eegmegcalc.FourSphereVolumeConductor` or `lfpypykit.eegmegcalc.MEG`, respectively

Parameters

cell: object

CellGeometry instance or similar.

See also:

[LinearModel](#)

[eegmegcalc.FourSphereVolumeConductor](#)

[eegmegcalc.MEG](#)

[eegmegcalc.NYHeadModel](#)

References

[1]

Examples

Compute the current dipole moment of a 3-compartment neuron model:

```
>>> import numpy as np
>>> from lfpypykit import CellGeometry, CurrentDipoleMoment
>>> n_seg = 3
>>> cell = CellGeometry(x=np.array([[0.]*2]*n_seg),
                        y=np.array([[0.]*2]*n_seg),
                        z=np.array([[1.*x, 1.*(x+1)]
                                    for x in range(n_seg)]),
                        d=np.array([1.]*n_seg))
>>> cdm = CurrentDipoleMoment(cell)
>>> M = cdm.get_transformation_matrix()
>>> imem = np.array([[-1., 1.],
                    [0., 0.],
                    [1., -1.]])
>>> P = M@imem
>>> P
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 2., -2.]])
```

get_transformation_matrix()

Get linear response matrix

Returns**response_matrix: ndarray**
shape (3, n_seg) ndarray**Raises****AttributeError**
if cell is None

4.3 class PointSourcePotential

class lfpypykit.PointSourcePotential(*cell, x, y, z, sigma=0.3*)Bases: *LinearModel**LinearModel* subclass that defines a 2D linear response matrix **M** between transmembrane current array **I** (nA) of a multicompartment neuron model and the corresponding extracellular electric potential V_{ex} (mV) as

$$V_{ex} = \mathbf{M}\mathbf{I}$$

The current **I** is an ndarray of shape (n_seg, n_tsteps) with unit (nA), and each row indexed by *j* of V_{ex} represents the electric potential at each measurement site for every time step.The elements of **M** are computed as

$$M_{ji} = 1/(4\pi\sigma|\mathbf{r}_i - \mathbf{r}_j|)$$

where σ is the electric conductivity of the extracellular medium, $\mathbf{r}_i$ the midpoint coordinate of segment *i* and $\mathbf{r}_j$ the coordinate of measurement site *j* [1], [2].

Assumptions:

- the extracellular conductivity σ is infinite, homogeneous, frequency independent (linear) and isotropic.
- each segment is treated as a point source located at the midpoint between its start and end point coordinate.
- each measurement site $\mathbf{r}_j = (x_j, y_j, z_j)$ is treated as a point.
- $|\mathbf{r}_i - \mathbf{r}_j| \geq d_i/2$, where d_i is the segment diameter.

Parameters

cell: object
CellGeometry instance or similar.

x: ndarray of floats
x-position of measurement sites (μm)

y: ndarray of floats
y-position of measurement sites (μm)

z: ndarray of floats
z-position of measurement sites (μm)

sigma: float > 0
scalar extracellular conductivity (S/m)

See also:

[*LinearModel*](#)
[*LineSourcePotential*](#)
[*RecExtElectrode*](#)

References

[1], [2]

Examples

Compute the current dipole moment of a 3-compartment neuron model:

```
>>> import numpy as np
>>> from lfpypykit import CellGeometry, PointSourcePotential
>>> n_seg = 3
>>> cell = CellGeometry(x=np.array([[0.]*2]*n_seg),
                        y=np.array([[0.]*2]*n_seg),
                        z=np.array([[10.*x, 10.*(x+1)]
                                   for x in range(n_seg)]),
                        d=np.array([1.]*n_seg))
>>> psp = PointSourcePotential(cell,
                               x=np.ones(10)*10,
                               y=np.zeros(10),
                               z=np.arange(10)*10,
                               sigma=0.3)
>>> M = psp.get_transformation_matrix()
>>> imem = np.array([[-1., 1.],
                    [0., 0.],
                    [1., -1.]])
>>> V_ex = M @ imem
>>> V_ex
array([[ -0.01387397,  0.01387397],
       [ -0.00901154,  0.00901154],
       [ 0.00901154, -0.00901154],
       [ 0.01387397, -0.01387397],
       [ 0.00742668, -0.00742668],
       [ 0.00409718, -0.00409718],
       [ 0.00254212, -0.00254212],
       [ 0.00172082, -0.00172082],
       [ 0.00123933, -0.00123933],
       [ 0.00093413, -0.00093413]])
```

`get_transformation_matrix()`

Get linear response matrix

Returns

response_matrix: ndarray
shape (n_coords, n_seg) ndarray

Raises

AttributeError
if cell is None

4.4 class LineSourcePotential

class lfpkit.LineSourcePotential(*cell, x, y, z, sigma=0.3*)

Bases: [LinearModel](#)

LinearModel subclass that defines a 2D linear response matrix **M** between transmembrane current array **I** (nA) of a multicompartment neuron model and the corresponding extracellular electric potential V_{ex} (mV) as

$$V_{ex} = \mathbf{M}\mathbf{I}$$

The current **I** is an ndarray of shape (n_seg, n_tsteps) with unit (nA), and each row indexed by *j* of V_{ex} represents the electric potential at each measurement site for every time step.

The elements of **M** are computed as

$$M_{ji} = \frac{1}{4\pi\sigma L_i} \log \left| \frac{\sqrt{h_{ji}^2 + r_{ji}^2} - h_{ji}}{\sqrt{l_{ji}^2 + r_{ji}^2} - l_{ji}} \right|$$

Segment length is denoted L_i , perpendicular distance from the electrode point contact to the axis of the line segment is denoted r_{ji} , longitudinal distance measured from the start of the segment is denoted h_{ji} , and longitudinal distance from the other end of the segment is denoted $l_{ji} = L_i + h_{ji}$ [1], [2].

Assumptions:

- the extracellular conductivity σ is infinite, homogeneous, frequency independent (linear) and isotropic
- each segment is treated as a straight line source with homogeneous current density between its start and end point coordinate
- each measurement site $\mathbf{r}_j = (x_j, y_j, z_j)$ is treated as a point
- The minimum distance to a line source is set equal to segment radius.

Parameters

- cell:** object
CellGeometry instance or similar.
- x:** ndarray of floats
x-position of measurement sites (μm)
- y:** ndarray of floats
y-position of measurement sites (μm)
- z:** ndarray of floats
z-position of measurement sites (μm)
- sigma:** float > 0
scalar extracellular conductivity (S/m)

See also:

[LinearModel](#)
[PointSourcePotential](#)
[RecExtElectrode](#)

References

[1], [2]

Examples

Compute the current dipole moment of a 3-compartment neuron model:

```
>>> import numpy as np
>>> from lfpypykit import CellGeometry, LineSourcePotential
>>> n_seg = 3
>>> cell = CellGeometry(x=np.array([[0.]*2]*n_seg),
                        y=np.array([[0.]*2]*n_seg),
                        z=np.array([[10.*x, 10.*(x+1)]
                                   for x in range(n_seg)]),
                        d=np.array([1.]*n_seg))
>>> lsp = LineSourcePotential(cell,
                             x=np.ones(10)*10,
                             y=np.zeros(10),
                             z=np.arange(10)*10,
                             sigma=0.3)
>>> M = lsp.get_transformation_matrix()
>>> imem = np.array([[-1., 1.],
                    [0., 0.],
                    [1., -1.]])
>>> V_ex = M @ imem
>>> V_ex
array([[ -0.01343699,  0.01343699],
       [ -0.0084647 ,  0.0084647 ],
       [ 0.0084647 , -0.0084647 ],
       [ 0.01343699, -0.01343699],
       [ 0.00758627, -0.00758627],
       [ 0.00416681, -0.00416681],
       [ 0.002571 , -0.002571 ],
       [ 0.00173439, -0.00173439],
       [ 0.00124645, -0.00124645],
       [ 0.0009382 , -0.0009382 ]])
```

get_transformation_matrix()

Get linear response matrix

Returns

response_matrix: ndarray
shape (n_coords, n_seg) ndarray

Raises

AttributeError
if cell is None

4.5 class RecExtElectrode

```
class lfpypykit.RecExtElectrode(cell, sigma=0.3, probe=None, x=None, y=None, z=None, N=None, r=None, n=None, contact_shape='circle', method='linesource', verbose=False, seedvalue=None, **kwargs)
```

Bases: [LinearModel](#)

class RecExtElectrode

Main class that represents an extracellular electric recording devices such as a laminar probe.

This class is a [LinearModel](#) subclass that defines a 2D linear response matrix $\mathbf{M}$ between transmembrane current array $\mathbf{I}$ (nA) of a multicompartment neuron model and the corresponding extracellular electric potential $\mathbf{V}_{ex}$ (mV) as

$$\mathbf{V}_{ex} = \mathbf{M}\mathbf{I}$$

The current $\mathbf{I}$ is an ndarray of shape (n_seg, n_tsteps) with unit (nA), and each row indexed by j of $\mathbf{V}_{ex}$ represents the electric potential at each measurement site for every time step.

The class differ from [PointSourcePotential](#) and [LineSourcePotential](#) by:

- supporting anisotropic volume conductors [1]
- supporting probe geometry specifications using [MEAutility](#) (<https://meautility.readthedocs.io/en/latest/>, <https://github.com/alejoe91/MEAutility>).
- supporting electrode contact points with finite extents [2], [3]
- switching between point- and linesources, and a combined method that assumes that the root element at segment index 0 is spherical.

Parameters

cell: object

CellGeometry instance or similar.

sigma: float or list/ndarray of floats

extracellular conductivity in units of (S/m). A scalar value implies an isotropic extracellular conductivity. If a length 3 list or array of floats is provided, these values corresponds to an anisotropic conductor with conductivities $[\sigma_x, \sigma_y, \sigma_z]$.

probe: MEAutility MEA object or None

MEAutility probe object

x, y, z: ndarray

coordinates or same length arrays of coordinates in units of (μm).

N: None or list of lists

Normal vectors [x, y, z] of each circular electrode contact surface, default None

r: float

radius of each contact surface, default None (μm)

n: int

if N is not None and $r > 0$, the number of discrete points used to compute the n-point average potential on each circular contact point.

contact_shape: str

'circle'/'square' (default 'circle') defines the contact point shape. If 'circle' r is the radius, if 'square' r is the side length

method: str

switch between the assumption of 'linesource', 'pointsource', 'root_as_point' to represent each compartment when computing extracellular potentials

verbose: bool

Flag for verbose output, i.e., print more information

seedvalue: int

random seed when finding random position on contact with $r > 0$

****kwargs:**

Additional keyword arguments parsed to *RecExtElectrode.lfp_method()* which is determined by *method* parameter.

See also:

[*LinearModel*](#)

[*PointSourcePotential*](#)

[*LineSourcePotential*](#)

References

[1], [2], [3]

Examples

Mock cell geometry and transmembrane currents:

```
>>> import numpy as np
>>> from lfpypykit import CellGeometry, RecExtElectrode
>>> # cell geometry with three segments (μm)
>>> cell = CellGeometry(x=np.array([[0, 0], [0, 0], [0, 0]]),
>>>                      y=np.array([[0, 0], [0, 0], [0, 0]]),
>>>                      z=np.array([[0, 10], [10, 20], [20, 30]]),
>>>                      d=np.array([1, 1, 1]))
>>> # transmembrane currents, three time steps (nA)
>>> I_m = np.array([[0., -1., 1.], [-1., 1., 0.], [1., 0., -1.]])
>>> # electrode locations (μm)
>>> r = np.array([[28.24653166, 8.97563241, 18.9492774, 3.47296614,
>>>                  1.20517729, 9.59849603, 21.91956616, 29.84686727,
>>>                  4.41045505, 3.61146625],
>>>                [24.4954352, 24.04977922, 22.41262238, 10.09702942,
>>>                  3.28610789, 23.50277637, 8.14044367, 4.46909208,
>>>                  10.93270117, 24.94698813],
>>>                [19.16644585, 15.20196335, 18.08924828, 24.22864702,
>>>                  5.85216751, 14.8231048, 24.72666694, 17.77573431,
>>>                  29.34508292, 9.28381892]])
>>> # instantiate electrode, get linear response matrix
>>> el = RecExtElectrode(cell=cell, x=r[0, ], y=r[1, ], z=r[2, ],
>>>                      sigma=0.3,
>>>                      method='pointsource')
>>> M = el.get_transformation_matrix()
>>> # compute extracellular potential
```

(continues on next page)

(continued from previous page)

```
>>> M @ I_m
array([[ -4.11657148e-05,  4.16621950e-04, -3.75456235e-04],
       [ -6.79014892e-04,  7.30256301e-04, -5.12414088e-05],
       [ -1.90930536e-04,  7.34007655e-04, -5.43077119e-04],
       [  5.98270144e-03,  6.73490846e-03, -1.27176099e-02],
       [ -1.34547752e-02, -4.65520036e-02,  6.00067788e-02],
       [ -7.49957880e-04,  7.03763787e-04,  4.61940938e-05],
       [  8.69330232e-04,  1.80346156e-03, -2.67279180e-03],
       [ -2.04546513e-04,  6.58419628e-04, -4.53873115e-04],
       [  6.82640209e-03,  4.47953560e-03, -1.13059377e-02],
       [ -1.33289553e-03, -1.11818140e-04,  1.44471367e-03]])
```

Compute extracellular potentials after simulating and storage of transmembrane currents with the LFPy.Cell class:

```
>>> import os
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import LFPy
>>> from lfpykit import CellGeometry, RecExtElectrode
>>>
>>> cellParameters = {
>>>     'morphology': os.path.join(LFPy.__path__[0], 'test',
>>>                               'ball_and_sticks.hoc'),
>>>     'v_init': -65,                    # initial voltage
>>>     'cm': 1.0,                        # membrane capacitance
>>>     'Ra': 150,                        # axial resistivity
>>>     'passive': True,                  # insert passive channels
>>>     'passive_parameters': {"g_pas": 1./3E4,
>>>                             "e_pas": -65}, # passive params
>>>     'dt': 2**-4,                      # simulation time res
>>>     'tstart': 0.,                     # start t of simulation
>>>     'tstop': 50.,                     # end t of simulation
>>> }
>>> cell = LFPy.Cell(**cellParameters)
>>>
>>> synapseParameters = {
>>>     'idx': cell.get_closest_idx(x=0, y=0, z=800), # segment
>>>     'e': 0,                                     # reversal potential
>>>     'syntype': 'ExpSyn',                        # synapse type
>>>     'tau': 2,                                   # syn. time constant
>>>     'weight': 0.01,                             # syn. weight
>>>     'record_current': True                      # syn. current record
>>> }
>>> synapse = LFPy.Synapse(cell, **synapseParameters)
>>> synapse.set_spike_times(np.array([10., 15., 20., 25.]))
>>>
>>> cell.simulate(rec_imem=True)
>>>
>>> N = np.empty((16, 3))
>>> for i in range(N.shape[0]): N[i,] = [1, 0, 0] # normal vectors
>>> electrodeParameters = {                      # parameters for RecExtElectrode class
```

(continues on next page)

(continued from previous page)

```

>>> 'sigma': 0.3,                # Extracellular potential
>>> 'x': np.zeros(16)+25,        # Coordinates of electrode contacts
>>> 'y': np.zeros(16),
>>> 'z': np.linspace(-500,1000,16),
>>> 'n': 20,
>>> 'r': 10,
>>> 'N': N,
>>> }
>>> electrode = RecExtElectrode(cell, **electrodeParameters)
>>> M = electrode.get_transformation_matrix()
>>> V_ex = M @ cell.imem
>>> plt.matshow(V_ex)
>>> plt.colorbar()
>>> plt.axis('tight')
>>> plt.show()

```

Compute extracellular potentials during simulation:

```

>>> import os
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import LFPy
>>> from lfpykit import CellGeometry, RecExtElectrode
>>>
>>> cellParameters = {
>>>     'morphology': os.path.join(LFPy.__path__[0], 'test',
>>>                                 'ball_and_sticks.hoc'),
>>>     'v_init': -65,                # initial voltage
>>>     'cm': 1.0,                    # membrane capacitance
>>>     'Ra': 150,                    # axial resistivity
>>>     'passive': True,              # insert passive channels
>>>     'passive_parameters': {"g_pas": 1./3E4,
>>>                             "e_pas": -65}, # passive params
>>>     'dt': 2**-4,                  # simulation time res
>>>     'tstart': 0.,                 # start t of simulation
>>>     'tstop': 50.,                # end t of simulation
>>> }
>>> cell = LFPy.Cell(**cellParameters)
>>>
>>> synapseParameters = {
>>>     'idx': cell.get_closest_idx(x=0, y=0, z=800), # compartment
>>>     'e': 0,                                # reversal potential
>>>     'syntype': 'ExpSyn',                    # synapse type
>>>     'tau': 2,                                # syn. time constant
>>>     'weight': 0.01,                          # syn. weight
>>>     'record_current': True                    # syn. current record
>>> }
>>> synapse = LFPy.Synapse(cell, **synapseParameters)
>>> synapse.set_spike_times(np.array([10., 15., 20., 25.]))
>>>
>>> N = np.empty((16, 3))
>>> for i in range(N.shape[0]): N[i,] = [1, 0, 0] #normal vec. of contacts

```

(continues on next page)

(continued from previous page)

```

>>> electrodeParameters = {          # parameters for RecExtElectrode class
>>>     'sigma': 0.3,                # Extracellular potential
>>>     'x': np.zeros(16)+25,        # Coordinates of electrode contacts
>>>     'y': np.zeros(16),
>>>     'z': np.linspace(-500,1000,16),
>>>     'n': 20,
>>>     'r': 10,
>>>     'N': N,
>>> }
>>> electrode = RecExtElectrode(cell, **electrodeParameters)
>>> cell.simulate(probes=[electrode])
>>> plt.matshow(electrode.data)
>>> plt.colorbar()
>>> plt.axis('tight')
>>> plt.show()

```

Use MEAutility to to handle probes

```

>>> import os
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import MEAutility as mu
>>> import LFPy
>>> from lfpykit import CellGeometry, RecExtElectrode
>>>
>>> cellParameters = {
>>>     'morphology': os.path.join(LFPy.__path__[0], 'test',
>>>                                'ball_and_sticks.hoc'),
>>>     'v_init': -65,                # initial voltage
>>>     'cm': 1.0,                    # membrane capacitance
>>>     'Ra': 150,                    # axial resistivity
>>>     'passive': True,              # insert passive channels
>>>     'passive_parameters': {"g_pas":1./3E4,
>>>                             "e_pas":-65}, # passive params
>>>     'dt': 2**-4,                  # simulation time res
>>>     'tstart': 0.,                 # start t of simulation
>>>     'tstop': 50.,                 # end t of simulation
>>> }
>>> cell = LFPy.Cell(**cellParameters)
>>>
>>> synapseParameters = {
>>>     'idx': cell.get_closest_idx(x=0, y=0, z=800), # compartment
>>>     'e': 0,                          # reversal potential
>>>     'syntype': 'ExpSyn',              # synapse type
>>>     'tau': 2,                         # syn. time constant
>>>     'weight': 0.01,                  # syn. weight
>>>     'record_current': True           # syn. current record
>>> }
>>> synapse = LFPy.Synapse(cell, **synapseParameters)
>>> synapse.set_spike_times(np.array([10., 15., 20., 25.]))
>>>
>>> cell.simulate(rec_imem=True)

```

(continues on next page)

(continued from previous page)

```

>>>
>>> probe = mu.return_mea('Neuropixels-128')
>>> electrode = RecExtElectrode(cell, probe=probe)
>>> V_ex = electrode.get_transformation_matrix() @ cell.imem
>>> mu.plot_mea_recording(V_ex, probe)
>>> plt.axis('tight')
>>> plt.show()

```

get_transformation_matrix()

Get linear response matrix

Returns

response_matrix: ndarray
 shape (n_contacts, n_seg) ndarray

Raises

AttributeError
 if cell is None

4.6 class RecMEAElectrode

```

class lfpkit.RecMEAElectrode(cell, sigma_T=0.3, sigma_S=1.5, sigma_G=0.0, h=300.0, z_shift=0.0,
                             steps=20, probe=None, x=array([0]), y=array([0]), z=array([0]), N=None,
                             r=None, n=None, method='linesource', verbose=False, seedvalue=None,
                             squeeze_cell_factor=None, **kwargs)

```

Bases: [RecExtElectrode](#)

class RecMEAElectrode

Electrode class that represents an extracellular in vitro slice recording as a Microelectrode Array (MEA). Inherits RecExtElectrode class

Illustration:

```

      Above neural tissue (Saline) -> sigma_S
<-----*-----> z = z_shift + h

      Neural Tissue -> sigma_T
      o -> source_pos = [x',y',z']

<-----*-----> z = z_shift + 0
      \-> elec_pos = [x,y,z]

      Below neural tissue (MEA Glass plate) -> sigma_G

```

For further details, see reference [1].

Parameters

cell: object
 GeometryCell instance or similar.

sigma_T: float
extracellular conductivity of neural tissue in unit (S/m)

sigma_S: float
conductivity of saline bath that the neural slice is immersed in [1.5] (S/m)

sigma_G: float
conductivity of MEA glass electrode plate. Most commonly assumed non-conducting [0.0] (S/m)

h: float, int
Thickness in um of neural tissue layer containing current the current sources (i.e., in vitro slice or cortex)

z_shift: float, int
Height in um of neural tissue layer bottom. If e.g., top of neural tissue layer should be $z=0$, use $z_shift=-h$. Defaults to $z_shift = 0$, so that the neural tissue layer extends from $z=0$ to $z=h$.

squeeze_cell_factor: float or None
Factor to squeeze the cell in the z-direction. This is needed for large cells that are thicker than the slice, since no part of the cell is allowed to be outside the slice. The squeeze is done after the neural simulation, and therefore does not affect neuronal simulation, only calculation of extracellular potentials.

probe: MEAutility MEA object or None
MEAutility probe object

x, y, z: np.ndarray
coordinates or arrays of coordinates in units of (um). Must be same length

N: None or list of lists
Normal vectors [x, y, z] of each circular electrode contact surface, default None

r: float
radius of each contact surface, default None

n: int
if N is not None and $r > 0$, the number of discrete points used to compute the n-point average potential on each circular contact point.

contact_shape: str
'circle'/'square' (default 'circle') defines the contact point shape If 'circle' r is the radius, if 'square' r is the side length

method: str
switch between the assumption of 'linesource', 'pointsource', 'root_as_point' to represent each compartment when computing extracellular potentials

verbose: bool
Flag for verbose output, i.e., print more information

seedvalue: int
random seed when finding random position on contact with $r > 0$

See also:

[*LinearModel*](#)
[*PointSourcePotential*](#)
[*LineSourcePotential*](#)
[*RecExtElectrode*](#)

References

[1]

Examples

Mock cell geometry and transmembrane currents:

```
>>> import numpy as np
>>> from lfpypykit import CellGeometry, RecMEAElectrode
>>> # cell geometry with four segments (μm)
>>> cell = CellGeometry(
>>>     x=np.array([[0, 10], [10, 20], [20, 30], [30, 40]]),
>>>     y=np.array([[0, 0], [0, 0], [0, 0], [0, 0]]),
>>>     z=np.array([[0, 0], [0, 0], [0, 0], [0, 0]]) + 10,
>>>     d=np.array([1, 1, 1, 1]))
>>> # transmembrane currents, three time steps (nA)
>>> I_m = np.array([[0.25, -1., 1.],
>>>                 [-1., 1., -0.25],
>>>                 [1., -0.25, -1.],
>>>                 [-0.25, 0.25, 0.25]])
>>> # electrode locations (μm)
>>> r = np.stack([np.arange(10)*4 + 2, np.zeros(10), np.zeros(10)])
>>> # instantiate electrode, get linear response matrix
>>> el = RecMEAElectrode(cell=cell,
>>>                       sigma_T=0.3, sigma_S=1.5, sigma_G=0.0,
>>>                       x=r[0, :], y=r[1, :], z=r[2, :],
>>>                       method='pointsource')
>>> M = el.get_transformation_matrix()
>>> # compute extracellular potential
>>> M @ I_m
array([[ -0.00233572, -0.01990957,  0.02542055],
       [ -0.00585075, -0.01520865,  0.02254483],
       [ -0.01108601, -0.00243107,  0.01108601],
       [ -0.01294584,  0.01013595, -0.00374823],
       [ -0.00599067,  0.01432711, -0.01709416],
       [  0.00599067,  0.01194602, -0.0266944 ],
       [  0.01294584,  0.00953841, -0.02904238],
       [  0.01108601,  0.00972426, -0.02324134],
       [  0.00585075,  0.01075236, -0.01511768],
       [  0.00233572,  0.01038382, -0.00954429]])
```

See also <LFPy>/examples/example_MEA.py

```
>>> import os
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import LFPy
>>> from lfpypykit import CellGeometry, RecMEAElectrode
>>>
>>> cellParameters = {
>>>     'morphology': os.path.join(LFPy.__path__[0], 'test',
>>>                                'ball_and_sticks.hoc'),
```

(continues on next page)

(continued from previous page)

```

>>> 'v_init': -65,                # initial voltage
>>> 'cm': 1.0,                    # membrane capacitance
>>> 'Ra': 150,                    # axial resistivity
>>> 'passive': True,              # insert passive channels
>>> 'passive_parameters': {"g_pas": 1./3E4,
>>>                        "e_pas": -65}, # passive params
>>> 'dt': 2**-4,                  # simulation time res
>>> 'tstart': 0.,                 # start t of simulation
>>> 'tstop': 50.,                 # end t of simulation
>>> }
>>> cell = LFPy.Cell(**cellParameters)
>>> cell.set_rotation(x=np.pi/2, z=np.pi/2)
>>> cell.set_pos(z=100)
>>> synapseParameters = {
>>>     'idx': cell.get_closest_idx(x=800, y=0, z=100), # segment
>>>     'e': 0,                                         # reversal potential
>>>     'syntype': 'ExpSyn',                             # synapse type
>>>     'tau': 2,                                       # syn. time constant
>>>     'weight': 0.01,                                # syn. weight
>>>     'record_current': True                           # syn. current record
>>> }
>>> synapse = LFPy.Synapse(cell, **synapseParameters)
>>> synapse.set_spike_times(np.array([10., 15., 20., 25.]))
>>>
>>> MEA_electrode_parameters = {
>>>     'sigma_T': 0.3,      # extracellular conductivity
>>>     'sigma_G': 0.0,      # MEA glass electrode plate conductivity
>>>     'sigma_S': 1.5,      # Saline bath conductivity
>>>     'x': np.linspace(0, 1200, 16), # 1d vector of positions
>>>     'y': np.zeros(16),
>>>     'z': np.zeros(16),
>>>     "method": "pointsource",
>>>     "h": 300,
>>>     "squeeze_cell_factor": 0.5,
>>> }
>>> cell.simulate(rec_imem=True)
>>>
>>> MEA = RecMEAElectrode(cell, **MEA_electrode_parameters)
>>> V_ext = MEA.get_transformation_matrix() @ lfp_cell.imem
>>>
>>> plt.matshow(V_ext)
>>> plt.colorbar()
>>> plt.axis('tight')
>>> plt.show()

```

distort_cell_geometry(axis='z', nu=0.0)

Distorts cellular morphology with a relative squeeze_cell_factor along a chosen axis preserving Poisson's ratio. A ratio nu=0.5 assumes incompressible and isotropic media that embeds the cell. A ratio nu=0 will only affect geometry along the chosen axis. A ratio nu=-1 will isometrically scale the neuron geometry along each axis. This method does not affect the underlying cable properties of the cell, only predictions of extracellular measurements (by affecting the relative locations of sources representing the compartments).

Parameters

axis: str

which axis to apply compression/stretching. Default is “z”.

nu: float

Poisson’s ratio. Ratio between axial and transversal compression/stretching. Default is 0.

get_transformation_matrix()

Get linear response matrix

Returns

response_matrix: ndarray

shape (n_contacts, n_seg) ndarray

Raises

AttributeError

if cell is None

4.7 class OneSphereVolumeConductor

class lfpypykit.**OneSphereVolumeConductor**(cell, r, R=10000.0, sigma_i=0.3, sigma_o=0.03)

Bases: [LinearModel](#)

Computes extracellular potentials within and outside a spherical volume- conductor model that assumes homogeneous, isotropic, linear (frequency independent) conductivity in and outside the sphere with a radius R. The conductivity in and outside the sphere must be greater than 0, and the current source(s) must be located within the radius R.

The implementation is based on the description of electric potentials of point charge in an dielectric sphere embedded in dielectric media [1], which is mathematically equivalent to a current source in conductive media.

This class is a [LinearModel](#) subclass that defines a 2D linear response matrix **M** between transmembrane current array **I** (nA) of a multicompartment neuron model and the corresponding extracellular electric potential V_{ex} (mV) as

$$\mathbf{V}_{ex} = \mathbf{M}\mathbf{I}$$

The current **I** is an ndarray of shape (n_seg, n_tsteps) with unit (nA), and each row indexed by *j* of $\mathbf{V}_{ex}$ represents the electric potential at each measurement site for every time step.

Parameters

cell: object or None

CellGeometry instance or similar.

r: ndarray, dtype=float

shape(3, n_points) observation points in space in spherical coordinates (radius, theta, phi) relative to the center of the sphere.

R: float

sphere radius (μm)

sigma_i: float

electric conductivity for radius $r \leq R$ (S/m)

sigma_o: float

electric conductivity for radius $r > R$ (S/m)

References

[1]

Examples

Compute the potential for a single monopole along the x-axis:

```
>>> # import modules
>>> from lfpypykit import OneSphereVolumeConductor
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> # observation points in spherical coordinates (flattened)
>>> X, Y = np.mgrid[-15000:15100:1000., -15000:15100:1000.]
>>> r = np.array([np.sqrt(X**2 + Y**2).flatten(),
>>>               np.arctan2(Y, X).flatten(),
>>>               np.zeros(X.size)])
>>> # set up class object and compute electric potential in all locations
>>> sphere = OneSphereVolumeConductor(cell=None, r=r, R=10000.,
>>>                                   sigma_i=0.3, sigma_o=0.03)
>>> Phi = sphere.calc_potential(rs=8000, current=1.).reshape(X.shape)
>>> # plot
>>> fig, ax = plt.subplots(1,1)
>>> im=ax.contourf(X, Y, Phi,
>>>               levels=np.linspace(Phi.min(),
>>>                                   np.median(Phi[np.isfinite(Phi)]) * 4, 30))
>>> circle = plt.Circle(xy=(0,0), radius=sphere.R, fc='none', ec='k')
>>> ax.add_patch(circle)
>>> fig.colorbar(im, ax=ax)
>>> plt.show()
```

calc_potential(rs, current, min_distance=1.0, n_max=1000)

Return the electric potential at observation points for source current as function of time.

Parameters

rs: float

monopole source location along the horizontal x-axis (μm)

current: float or ndarray, dtype float

float or shape (n_steps,) array containing source current (nA)

min_distance: None or float

minimum distance between source location and observation point (μm) (in order to avoid singularities)

n_max: int

Number of elements in polynomial expansion to sum over (see [1]).

Returns

Phi: ndarray

shape (n-points,) ndarray of floats if I is float like. If I is an 1D ndarray, and shape (n-points, I.size) ndarray is returned. Unit (mV).

References

[1]

get_transformation_matrix(*n_max=1000*)

Compute linear mapping between transmembrane currents of CellGeometry like object instance and extracellular potential in and outside of sphere.

Parameters

n_max: int

Number of elements in polynomial expansion to sum over (see [1]).

Returns

ndarray

Shape (n_points, n_compartments) mapping between individual segments and extracellular potential in extracellular locations

Raises

AttributeError

if cell is None

Notes

Each segment is treated as a point source in space. The minimum source to measurement site distance will be set to the diameter of each segment

References

[1]

Examples

Compute extracellular potential in one-sphere volume conductor model from LFPy.Cell object:

```
>>> # import modules
>>> import LFPy
>>> from lfpypykit import CellGeometry,          >>> OneSphereVolumeConductor
>>> import os
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from matplotlib.collections import PolyCollection
>>> # create cell
>>> cell = LFPy.Cell(morphology=os.path.join(LFPy.__path__[0], 'test',
>>>                                           'ball_and_sticks.hoc'),
>>>                  tstop=10.)
>>> cell.set_pos(z=9800.)
>>> # stimulus
>>> syn = LFPy.Synapse(cell, idx=cell.totnsegs-1, syntype='Exp2Syn',
>>>                    weight=0.01)
>>> syn.set_spike_times(np.array([1.]))
>>> # simulate
```

(continues on next page)

(continued from previous page)

```

>>> cell.simulate(rec_imem=True)
>>> # observation points in spherical coordinates (flattened)
>>> X, Z = np.mgrid[-500:501:10., 9500:10501:10.]
>>> Y = np.zeros(X.shape)
>>> r = np.array([np.sqrt(X**2 + Z**2).flatten(),
>>>               np.arccos(Z / np.sqrt(X**2 + Z**2)).flatten(),
>>>               np.arctan2(Y, X).flatten()])
>>> # set up class object and compute mapping between segment currents
>>> # and electric potential in space
>>> sphere = OneSphereVolumeConductor(cell, r=r, R=10000.,
>>>                                   sigma_i=0.3, sigma_o=0.03)
>>> M = sphere.get_transformation_matrix(n_max=1000)
>>> # pick out some time index for the potential and compute potential
>>> ind = cell.tvec==2.
>>> V_ex = (M @ cell.imem)[: , ind].reshape(X.shape)
>>> # plot potential
>>> fig, ax = plt.subplots(1,1)
>>> zips = []
>>> for x, z in cell.get_idx_polygons(projection=('x', 'z')):
>>>     zips.append(list(zip(x, z)))
>>> polycol = PolyCollection(zips,
>>>                          edgecolors='none',
>>>                          facecolors='gray')
>>> vrange = 1E-3 # limits for color contour plot
>>> im=ax.contour(X, Z, V_ex,
>>>               levels=np.linspace(-vrange, vrange, 41))
>>> circle = plt.Circle(xy=(0,0), radius=sphere.R, fc='none', ec='k')
>>> ax.add_collection(polycol)
>>> ax.add_patch(circle)
>>> ax.axis(ax.axis('equal'))
>>> ax.set_xlim(X.min(), X.max())
>>> ax.set_ylim(Z.min(), Z.max())
>>> fig.colorbar(im, ax=ax)
>>> plt.show()

```

4.8 class LaminarCurrentSourceDensity

class lfpypykit.LaminarCurrentSourceDensity(*cell, z, r*)

Bases: [LinearModel](#)

Facilitates calculations of the ground truth Current Source Density (CSD) in cylindrical volumes aligned with the z-axis based on [1] and [2].

The implementation assumes piecewise linear current sources similar to [LineSourcePotential](#), and accounts for the fraction of each segment's length within each volume, see Eq. 11 in [2].

This class is a [LinearModel](#) subclass that defines a 2D linear response matrix **M** between transmembrane current array **I** (nA) of a multicompartiment neuron model and the corresponding CSD **C** (nA/μm³) as

$$\mathbf{C} = \mathbf{M}\mathbf{I}$$

The current **I** is an ndarray of shape (n_seg, n_steps) with unit (nA), and each row indexed by *j* of **C** represents the CSD in each volume for every time step as the sum of currents divided by the volume.

Parameters**cell:** object or None

CellGeometry instance or similar.

z: ndarray, dtype=floatshape (n_volumes, 2) array of lower and upper edges of each volume along the z-axis in units of (μm). The lower edge value must be below the upper edge value.**r:** ndarray, dtype=floatshape (n_volumes,) array with assumed radius of each cylindrical volume. Each radius must be greater than zero, and in units of (μm)**Raises****AttributeError**

inputs z and r must be ndarrays of correct shape etc.

See also:

[*LinearModel*](#)[*VolumetricCurrentSourceDensity*](#)**References**

[1], [2]

Examples

Mock cell geometry and transmembrane currents:

```

>>> import numpy as np
>>> from lfpypykit import CellGeometry, LaminarCurrentSourceDensity
>>> # cell geometry with three segments ( $\mu\text{m}$ )
>>> cell = CellGeometry(x=np.array([[0, 0], [0, 0], [0, 0]]),
>>>                    y=np.array([[0, 0], [0, 0], [0, 0]]),
>>>                    z=np.array([[0, 10], [10, 20], [20, 30]]),
>>>                    d=np.array([1, 1, 1]))
>>> # transmembrane currents, three time steps (nA)
>>> I_m = np.array([[0., -1., 1.], [-1., 1., 0.], [1., 0., -1.]])
>>> # define geometry (z - upper and lower boundary; r - radius)
>>> # of cylindrical volumes aligned with the z-axis ( $\mu\text{m}$ )
>>> z = np.array([[ -10., 0.], [0., 10.], [10., 20.],
>>>               [20., 30.], [30., 40.]])
>>> r = np.array([100., 100., 100., 100., 100.])
>>> # instantiate electrode, get linear response matrix
>>> csd = LaminarCurrentSourceDensity(cell=cell, z=z, r=r)
>>> M = csd.get_transformation_matrix()
>>> # compute current source density [nA/ $\mu\text{m}^3$ ]
>>> M @ I_m
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00, -3.18309886e-06,  3.18309886e-06],
       [-3.18309886e-06,  3.18309886e-06,  0.00000000e+00],
       [ 3.18309886e-06,  0.00000000e+00, -3.18309886e-06],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00]])

```

get_transformation_matrix()

Get linear response matrix

Returns

response_matrix: ndarray
 shape (n_volumes, n_seg) ndarray

Raises

AttributeError
 if cell is None

4.9 class VolumetricCurrentSourceDensity

class lfpypykit.VolumetricCurrentSourceDensity(*cell, x=None, y=None, z=None, dl=1.0*)

Bases: [LinearModel](#)

Facilitates calculations of the ground truth Current Source Density (CSD) across 3D volumetric grid with bin edges defined by parameters *x*, *y* and *z*.

The implementation assumes piecewise constant current sources similar to [LineSourcePotential](#), and accounts for the fraction of each segment's length within each volume by counting the number of points representing partial segments with max length *dl* divided by the number of partial segments.

This class is a [LinearModel](#) subclass that defines a 4D linear response matrix *M* of shape (*x.size-1*, *y.size-1*, *z.size-1*, *n_seg*) between transmembrane current array *I* (nA) of a multicompartment neuron model and the corresponding CSD *C* (nA/μm³) as

$$\mathbf{C} = \mathbf{M}\mathbf{I}$$

The current *I* is an ndarray of shape (*n_seg*, *n_steps*) with unit (nA), and each row indexed by *j* of *C* represents the CSD in each bin for every time step as the sum of currents divided by the volume.

Parameters

cell: object or None

CellGeometry instance or similar.

x, y, z: ndarray, dtype=float

shape (n,) array of bin edges of each volume along each axis in units of (μm). Must be monotonously increasing.

dl: float

discretization length of compartments before binning (μm). Default=1. Lower values will result in more accurate estimates as each line source gets split into more points.

Raises

See also:

[LinearModel](#)

[LaminarCurrentSourceDensity](#)

Notes

The resulting mapping `M` may be very sparse (i.e. mostly made up by zeros) and can be converted into a sparse array for more efficient multiplication for the same result:

```
>>> import scipy.sparse as ss
>>> M_csc = ss.csc_matrix(M.reshape((-1, M.shape[-1])))
>>> C = M_csc @ I_m
>>> np.all(C.reshape((M.shape[:-1] + (-1,))) == (M @ I_m))
True
```

Examples

Mock cell geometry and transmembrane currents:

```
>>> import numpy as np
>>> from lfpypykit import CellGeometry, VolumetricCurrentSourceDensity
>>> # cell geometry with three segments ( $\mu\text{m}$ )
>>> cell = CellGeometry(x=np.array([[0, 0], [0, 0], [0, 0]]),
>>>                      y=np.array([[0, 0], [0, 0], [0, 0]]),
>>>                      z=np.array([[0, 10], [10, 20], [20, 30]]),
>>>                      d=np.array([1, 1, 1]))
>>> # transmembrane currents, three time steps (nA)
>>> I_m = np.array([[0., -1., 1.], [-1., 1., 0.], [1., 0., -1.]])
>>> # instantiate probe, get linear response matrix
>>> csd = VolumetricCurrentSourceDensity(cell=cell,
>>>                                       x=np.linspace(-20, 20, 5),
>>>                                       y=np.linspace(-20, 20, 5),
>>>                                       z=np.linspace(-20, 20, 5), dl=1.)
>>> M = csd.get_transformation_matrix()
>>> # compute current source density [ $\text{nA}/\mu\text{m}^3$ ]
>>> M @ I_m
array([[[[ 0.,  0.,  0.],
          [ 0.,  0.,  0.],
          [ 0.,  0.,  0.],
          [ 0.,  0.,  0.]],
        ...
```

`get_transformation_matrix()`

Get linear response matrix

Returns

response_matrix: ndarray

shape (x.size-1, y.size-1, z.size-1, n_seg) ndarray

Raises

AttributeError

if cell is None

MODULE `LFYKIT.EEGMEGCALC`

Collection of classes defining forward models applicable with current dipole moment predictions.

Copyright (C) 2017 Computational Neuroscience Group, NMBU. This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

5.1 class `eegmegcalc.FourSphereVolumeConductor`

```
class lfykit.eegmegcalc.FourSphereVolumeConductor(r_electrodes, radii=[79000.0, 80000.0, 85000.0,  
90000.0], sigmas=[0.3, 1.5, 0.015, 0.3],  
          iter_factor=2.02020202020204e-08)
```

Bases: `Model`

Main class for computing extracellular potentials in a four-sphere volume conductor model that assumes homogeneous, isotropic, linear (frequency independent) conductivity within the inner sphere and outer shells. The conductance outside the outer shell is 0 (air).

This class implements the corrected 4-sphere model described in [1], [2]

Parameters

`r_electrodes`: ndarray, dtype=float

Shape (n_contacts, 3) array containing n_contacts electrode locations in cartesian coordinates in units of (μm). All `r_el` in `r_electrodes` must be less than or equal to scalp radius and larger than the distance between dipole and sphere center: $|r_z| < r_{el} \leq radii[3]$.

`radii`: list, dtype=float

Len 4 list with the outer radii in units of (μm) for the 4 concentric shells in the four-sphere model: brain, csf, skull and scalp, respectively.

`sigmas`: list, dtype=float

Len 4 list with the electrical conductivity in units of (S/m) of the four shells in the four-sphere model: brain, csf, skull and scalp, respectively.

`iter_factor`: float

iteration-stop factor

See also:

[*InfiniteVolumeConductor*](#)
MEG

References

[1], [2]

Examples

Compute extracellular potential from current dipole moment in four-sphere head model:

```
>>> from lfpypykit.eegmegcalc import FourSphereVolumeConductor
>>> import numpy as np
>>> radii = [79000., 80000., 85000., 90000.] # (μm)
>>> sigmas = [0.3, 1.5, 0.015, 0.3] # (S/m)
>>> r_electrodes = np.array([[0., 0., 90000.], [0., 85000., 0.]]) # (μm)
>>> sphere_model = FourSphereVolumeConductor(r_electrodes, radii,
>>>                                           sigmas)
>>> # current dipole moment
>>> p = np.array([[10.]*10, [10.]*10, [10.]*10]) # 10 timesteps (nA μm)
>>> dipole_location = np.array([0., 0., 78000.]) # (μm)
>>> # compute potential
>>> sphere_model.get_dipole_potential(p, dipole_location) # (mV)
array([[1.06247669e-08, 1.06247669e-08, 1.06247669e-08, 1.06247669e-08,
        1.06247669e-08, 1.06247669e-08, 1.06247669e-08, 1.06247669e-08,
        1.06247669e-08, 1.06247669e-08],
       [2.39290752e-10, 2.39290752e-10, 2.39290752e-10, 2.39290752e-10,
        2.39290752e-10, 2.39290752e-10, 2.39290752e-10, 2.39290752e-10,
        2.39290752e-10, 2.39290752e-10]])
```

get_dipole_potential(p, dipole_location)

Return electric potential from current dipole moment **p** in location **dipole_location** in locations **r_electrodes**

Parameters

p: ndarray, dtype=float

Shape (3, n_timesteps) array containing the x,y,z components of the current dipole moment in units of (nA*μm) for all timesteps.

dipole_location: ndarray, dtype=float

Shape (3,) array containing the position of the current dipole in cartesian coordinates. Units of (μm).

Returns

potential: ndarray, dtype=float

Shape (n_contacts, n_timesteps) array containing the electric potential at contact point(s) `FourSphereVolumeConductor.rxyz` in units of (mV) for all timesteps of current dipole moment **p**.

get_transformation_matrix(dipole_location)

Get linear response matrix mapping current dipole moment in (nA μm) located in location **rz** to extracellular potential in (mV) at recording sites `FourSphereVolumeConductor.rxyz` (μm)

Parameters

dipole_location: ndarray, dtype=float

Shape (3,) array containing the position of the current dipole in cartesian coordinates. Units of (μm).

Returns

response_matrix: ndarray
 shape (n_contacts, 3) ndarray

5.2 class eegmegcalc.InfiniteVolumeConductor

class lfpypykit.eegmegcalc.**InfiniteVolumeConductor**(sigma=0.3)

Bases: Model

Main class for computing extracellular potentials with current dipole moment **P** in an infinite 3D volume conductor model that assumes homogeneous, isotropic, linear (frequency independent) conductivity σ . The potential V is computed as [1]:

$$V = \frac{\mathbf{P} \cdot \mathbf{r}}{4\pi\sigma r^3}$$

Parameters

sigma: float
 Electrical conductivity in extracellular space in units of (S/cm)

See also:

[*FourSphereVolumeConductor*](#)
MEG

References

[1]

Examples

Computing the potential from dipole moment valid in the far field limit.

```
>>> from lfpypykit.eegmegcalc import InfiniteVolumeConductor
>>> import numpy as np
>>> inf_model = InfiniteVolumeConductor(sigma=0.3)
>>> p = np.array([[10.], [10.], [10.]]) # (nA * μm)
>>> r = np.array([[1000.], [0.], [5000.]]) # (μm)
>>> inf_model.get_dipole_potential(p, r) # (mV)
array([[1.20049432e-07]])
```

get_dipole_potential(p, r)

Return electric potential from current dipole moment **p** in locations **r** relative to dipole

Parameters

p: ndarray, dtype=float
 Shape (3, n_timesteps) array containing the x,y,z components of the current dipole moment in units of (nA*μm) for all timesteps

r: ndarray, dtype=float
 Shape (n_contacts, 3) array containing the displacement vectors from dipole location to measurement location

Returns**potential: ndarray, dtype=float**

Shape (n_contacts, n_timesteps) array containing the electric potential at contact point(s)
 r in units of (mV) for all timesteps of current dipole moment p

get_transformation_matrix(r)

Get linear response matrix mapping current dipole moment in (nA μm) to extracellular potential in (mV)
 at recording sites r (μm)

Parameters**r: ndarray, dtype=float**

Shape (n_contacts, 3) array containing the displacement vectors from dipole location to
 measurement location (μm)

Returns**response_matrix: ndarray**

shape (n_contacts, 3) ndarray

5.3 class eegmegcalc.InfiniteHomogeneousVolCondMEG

```
class lfpypykit.eegmegcalc.InfiniteHomogeneousVolCondMEG(sensor_locations,
                                                         mu=1.2566370614359173e-06)
```

Bases: Model

Basic class for computing magnetic field from current dipole moment in an infinite homogeneous volume conductor model. For this purpose we use the Biot-Savart law derived from Maxwell's equations under the assumption of negligible magnetic induction effects [1]:

$$\mathbf{H} = \frac{\mathbf{p} \times \mathbf{R}}{4\pi R^3}$$

where $\mathbf{p}$ is the current dipole moment, $\mathbf{R}$ the vector between dipole source location and measurement location, and $R = |\mathbf{R}|$

Note that the magnetic field $\mathbf{H}$ is related to the magnetic field $\mathbf{B}$ as

$$\mu_0 \mathbf{H} = \mathbf{B} - \mathbf{M}$$

where μ_0 is the permeability of free space (very close to permeability of biological tissues). $\mathbf{M}$ denotes material magnetization (also ignored)

Parameters**sensor_locations: ndarray, dtype=float**

shape (n_locations x 3) array with x,y,z-locations of measurement devices where magnetic
 field of current dipole moments is calculated. In unit of (μm)

mu: float

Permeability. Default is permeability of vacuum ($\mu_0 = 4 * \pi * 10^{-7} \text{ T*m/A}$)

Raises**AssertionError**

If dimensionality of sensor_locations is wrong

See also:

SphericallySymmetricVolCondMEG
FourSphereVolumeConductor
InfiniteVolumeConductor

References

[1]

Examples

Define cell object, create synapse, compute current dipole moment:

```
>>> import LFPy, os, numpy as np, matplotlib.pyplot as plt
>>> from lfpypykit import CurrentDipoleMoment
>>> from lfpypykit.eegmegcalc import InfiniteHomogeneousVolCondMEG as MEG
>>> # create LFPy.Cell object
>>> cell = LFPy.Cell(morphology=os.path.join(LFPy.__path__[0], 'test',
>>>                                         'ball_and_sticks.hoc'),
>>>
>>>                    passive=True)
>>> cell.set_pos(0., 0., 0.)
>>> # create single synaptic stimuli at soma (idx=0)
>>> syn = LFPy.Synapse(cell, idx=0, syntype='ExpSyn', weight=0.01, tau=5,
>>>                    record_current=True)
>>> syn.set_spike_times_w_netstim()
>>> # simulate, record current dipole moment
>>> cell.simulate(rec_imem=True)
>>> # Compute current dipole moment
>>> cdp = CurrentDipoleMoment(cell=cell)
>>> M_cdp = cdp.get_transformation_matrix()
>>> current_dipole_moment = M_cdp @ cell.imem
>>> # Compute the dipole location as an average of segment locations
>>> # weighted by membrane area:
>>> dipole_location = (cell.area * np.c_[cell.x.mean(axis=-1),
>>>                                     cell.y.mean(axis=-1),
>>>                                     cell.z.mean(axis=-1)].T
>>>
>>>                    / cell.area.sum()).sum(axis=1)
>>> # Define sensor site, instantiate MEG object, get transformation matrix
>>> sensor_locations = np.array([[1E4, 0, 0]])
>>> meg = MEG(sensor_locations)
>>> M = meg.get_transformation_matrix(dipole_location)
>>> # compute the magnetic signal in a single sensor location:
>>> H = M @ current_dipole_moment
>>> # plot output
>>> plt.figure(figsize=(12, 8), dpi=120)
>>> plt.subplot(311)
>>> plt.plot(cell.tvec, cell.somav)
>>> plt.ylabel(r'$V_{soma}$ (mV)')
>>> plt.subplot(312)
>>> plt.plot(cell.tvec, syn.i)
>>> plt.ylabel(r'$I_{syn}$ (nA)')
>>> plt.subplot(313)
>>> plt.plot(cell.tvec, H[0].T)
```

(continues on next page)

(continued from previous page)

```

>>> plt.ylabel(r'$H$ (nA/um)')
>>> plt.xlabel('$t$ (ms)')
>>> plt.legend(['$H_x$', '$H_y$', '$H_z$'])
>>> plt.show()

```

calculate_B(*p*, *r_p*)

Compute magnetic field **B** from single current dipole **p** localized somewhere in space at **r_p**.

This function returns the magnetic field **B** = **H**.

Parameters

p: ndarray, dtype=float

shape (3, n_timesteps) array with x,y,z-components of current- dipole moment time series data in units of (nA μm)

r_p: ndarray, dtype=float

shape (3,) array with x,y,z-location of dipole in units of (μm)

Returns

ndarray, dtype=float

shape (n_locations x 3 x n_timesteps) array with x,y,z-components of the magnetic field **B** in units of (nA/μm)

calculate_H(*current_dipole_moment*, *dipole_location*)

Compute magnetic field **H** from single current-dipole moment localized in an infinite homogeneous volume conductor.

Parameters

current_dipole_moment: ndarray, dtype=float

shape (3, n_timesteps) array with x,y,z-components of current- dipole moment time series data in units of (nA μm)

dipole_location: ndarray, dtype=float

shape (3,) array with x,y,z-location of dipole in units of (μm)

Returns

ndarray, dtype=float

shape (n_locations x 3 x n_timesteps) array with x,y,z-components of the magnetic field **H** in units of (nA/μm)

Raises

AssertionError

If dimensionality of current_dipole_moment and/or dipole_location is wrong

get_transformation_matrix(*dipole_location*)

Get linear response matrix mapping current dipole moment in (nA μm) located in location dipole_location to magnetic field **H** in units of (nA/μm) at sensor_locations

Parameters

dipole_location: ndarray, dtype=float

shape (3,) array with x,y,z-location of dipole in units of (μm)

Returns

response_matrix: ndarray

shape (n_contacts, 3, 3) ndarray

5.4 class eegmegcalc.SphericallySymmetricVolCondMEG

class lfpypykit.eegmegcalc.SphericallySymmetricVolCondMEG(*r*, *mu*=1.2566370614359173e-06)

Bases: Model

Computes magnetic fields from current dipole in spherically-symmetric volume conductor models.

This class facilitates calculations according to eq. (34) from [1] (see also [2]) defined as:

$$\mathbf{H} = \frac{1}{4\pi} \frac{F \mathbf{p} \times \mathbf{r}_p - (\mathbf{p} \times \mathbf{r}_p \cdot \mathbf{r}) \nabla F}{F^2}, \text{ where}$$

$$F = a(ra + r^2 - \mathbf{r}_p \cdot \mathbf{r}),$$

$$\nabla F = (r^{-1}a^2 + a^{-1}\mathbf{a} \cdot \mathbf{r} + 2a + 2r)\mathbf{r} - (a + 2r + a^{-1}\mathbf{a} \cdot \mathbf{r})\mathbf{r}_p,$$

$$\mathbf{a} = \mathbf{r} - \mathbf{r}_p,$$

$$a = |\mathbf{a}|,$$

$$r = |\mathbf{r}|.$$

Here, $\mathbf{p}$ is the current dipole moment, $\mathbf{r}$ the measurement location(s) and $\mathbf{r}_p$ the current dipole location.

Note that the magnetic field $\mathbf{H}$ is related to the magnetic field $\mathbf{B}$ as

$$\mu_0 \mathbf{H} = \mathbf{B} - \mathbf{M},$$

where μ_0 denotes the permeability of free space (very close to permeability of biological tissues). $\mathbf{M}$ denotes material magnetization (which is ignored).

Parameters

r: ndarray

sensor locations, shape (n, 3) where n denotes number of locations, unit [μm]

mu: float

Permeability. Default is permeability of vacuum ($\mu_0 = 4\pi 10^{-7} \text{ Tm/A}$)

Raises

AssertionError

If dimensionality of sensor locations $\mathbf{r}$ is wrong

See also:

[*InfiniteHomogeneousVolCondMEG*](#)

References

[1], [2]

Examples

Compute the magnetic field from a current dipole located

```
>>> import numpy as np
>>> from lfpypykit.eegmegcalc import SphericallySymmetricVolCondMEG
>>> p = np.array([[0, 1, 0]]).T # tangential current dipole (nAμm)
>>> r_p = np.array([0, 0, 900000]) # dipole location (μm)
```

(continues on next page)

(continued from previous page)

```

>>> r = np.array([[0, 0, 92000]]) # measurement location (μm)
>>> m = SphericallySymmetricVolCondMEG(r=r)
>>> M = m.get_transformation_matrix(r_p=r_p)
>>> H = M @ p
>>> H # magnetic field (nA/μm)
array([[9.73094081e-09],
       [0.00000000e+00],
       [0.00000000e+00]])

```

calculate_B(p, r_p)

Compute magnetic field **B** from single current dipole **p** localized somewhere in space at **r_p**.

This function returns the magnetic field **B** = **H**.

Parameters

p: ndarray, dtype=float

shape (3, n_timesteps) array with x,y,z-components of current- dipole moment time series data in units of (nA μm)

r_p: ndarray, dtype=float

shape (3,) array with x,y,z-location of dipole in units of (μm)

Returns

ndarray, dtype=float

shape (n_locations x 3 x n_timesteps) array with x,y,z-components of the magnetic field **B** in units of (nA/μm)

calculate_H(p, r_p)

Compute magnetic field **H** from single current dipole **p** localized somewhere in space at **r_p**

Parameters

p: ndarray, dtype=float

shape (3, n_timesteps) array with x,y,z-components of current- dipole moment time series data in units of (nA μm)

r_p: ndarray, dtype=float

shape (3,) array with x,y,z-location of dipole in units of (μm)

Returns

ndarray, dtype=float

shape (n_locations x 3 x n_timesteps) array with x,y,z-components of the magnetic field **H** in units of (nA/μm)

Raises

AssertionError

If dimensionality of current_dipole_moment **p** and/or dipole_location **r_p** is wrong

get_transformation_matrix(r_p)

Get linear response matrix mapping current dipole moment in (nA μm) located in location **r_p** to magnetic field **H** in units of (nA/μm) at sensor locations **r**

Parameters

r_p: ndarray, dtype=float

shape (3,) array with x,y,z-location of dipole in units of (μm)

Returns

response_matrix: ndarray
 shape (n_sensors, 3, 3) ndarray

Raises**AssertionError**

If dipole location `r_p` has the wrong shape or if its radius is greater than radius to any sensor location in `<object>.r`

5.5 class eegmegcalc.NYHeadModel

class lfpypykit.eegmegcalc.NYHeadModel(*nyhead_file=None*)

Bases: Model

Main class for computing EEG signals from current dipole moment **P** in New York Head Model [1], [2]

Assumes units of nA * um for current dipole moment, and mV for the EEG

Parameters

nyhead_file: str [optional]

Location of file containing New York Head Model. If empty (or None), it will be looked for in the present working directory. If not present the user is asked if it should be downloaded from https://www.parralab.org/nyhead/sa_nyhead.mat

See also:

FourSphereVolumeConductor

MEG

Notes

The original unit of the New York model current dipole moment is (probably?) mA*m, and the EEG output unit is V. LFPykit's current dipole moments have units nA*um, and EEGs from the NYhead model is here recomputed in units of mV.

References

[1], [2]

Examples

Computing EEG from dipole moment.

```
>>> from lfpypykit.eegmegcalc import NYHeadModel
```

```
>>> nyhead = NYHeadModel()
```

```
>>> nyhead.set_dipole_pos('parietal_lobe') # predefined example location
>>> M = nyhead.get_transformation_matrix()
```

```
>>> # Rotate to be along normal vector of cortex
>>> p = nyhead.rotate_dipole_to_surface_normal([[0.], [0.], [1.]])
>>> eeg = M @ p # (mV)
```

find_closest_electrode()

Returns minimal distance (mm) and closest electrode idx to dipole location specified in self.dipole_pos.

get_transformation_matrix()

Get linear response matrix mapping from current dipole moment (nA μ m) to EEG signal (mV) at EEG electrodes (n=231)

Returns

response_matrix: ndarray
shape (231, 3) ndarray

return_closest_idx(pos)

Returns the index of the closest vertex in the brain to a given position (in mm).

Parameters**pos**

[array of length (3)] [x, y, z] of a location in the brain, given in mm, and not in μ m which is the default position unit in LFPy

Returns**idx**

[int] Index of the vertex in the brain that is closest to the given location

rotate_dipole_to_surface_normal(p, orig_ax_vec=[0, 0, 1])

Returns rotated dipole moment, p_rot, oriented along the normal vector of the cortex at the dipole location

Parameters**p**

[np.ndarray of size (3, num_timesteps)] Current dipole moment from neural simulation, [p_x(t), p_y(t), p_z(t)]. If z-axis is the depth axis of cortex in the original neural simulation p_x(t) and p_y(t) will typically be small, and orig_ax_vec = [0, 0, 1].

orig_ax_vec

[np.ndarray or list of length (3)] Original surface vector of cortex in the neural simulation. If depth axis of cortex is the z-axis, orig_ax_vec = [0, 0, 1].

Returns**p_rot**

[np.ndarray of size (3, num_timesteps)] Rotated current dipole moment, oriented along cortex normal vector at the dipole location

References

See: https://en.wikipedia.org/wiki/Rotation_matrix under “Rotation matrix from axis and angle”

set_dipole_pos(*dipole_pos=None*)

Sets the dipole location in the brain

Parameters

dipole_pos: None, str or array of length (3) [x, y, z] (mm)

Location of the dipole. If no argument is given (or `dipole_pos=None`), a location, ‘`motorsensory_cortex`’, from `self.dipole_pos_dict` is used. If `dipole_pos` is an array of length 3, the closest vertex in the brain will be set as the dipole location.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [1] H. Linden, K. H. Pettersen, G. T. Einevoll (2010). Intrinsic dendritic filtering gives low-pass power spectra of local field potentials. *J Comput Neurosci*, 29:423–444. DOI: 10.1007/s10827-010-0245-4
- [1] Linden H, Hagen E, Leski S, Norheim ES, Pettersen KH, Einevoll GT (2014) LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons. *Front. Neuroinform.* 7:41. doi: 10.3389/fninf.2013.00041
- [2] Hagen E, Næss S, Ness TV and Einevoll GT (2018) Multimodal Modeling of Neural Network Activity: Computing LFP, ECoG, EEG, and MEG Signals With LFPy 2.0. *Front. Neuroinform.* 12:92. doi: 10.3389/fninf.2018.00092
- [1] Linden H, Hagen E, Leski S, Norheim ES, Pettersen KH, Einevoll GT (2014) LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons. *Front. Neuroinform.* 7:41. doi: 10.3389/fninf.2013.00041
- [2] Hagen E, Næss S, Ness TV and Einevoll GT (2018) Multimodal Modeling of Neural Network Activity: Computing LFP, ECoG, EEG, and MEG Signals With LFPy 2.0. *Front. Neuroinform.* 12:92. doi: 10.3389/fninf.2018.00092
- [1] Ness, T. V., Chintaluri, C., Potworowski, J., Leski, S., Glabska, H., Wojcik, D. K., et al. (2015). Modelling and analysis of electrical potentials recorded in microelectrode arrays (MEAs). *Neuroinformatics* 13:403–426. doi: 10.1007/s12021-015-9265-6
- [2] Linden H, Hagen E, Leski S, Norheim ES, Pettersen KH, Einevoll GT (2014) LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons. *Front. Neuroinform.* 7:41. doi: 10.3389/fninf.2013.00041
- [3] Hagen E, Næss S, Ness TV and Einevoll GT (2018) Multimodal Modeling of Neural Network Activity: Computing LFP, ECoG, EEG, and MEG Signals With LFPy 2.0. *Front. Neuroinform.* 12:92. doi: 10.3389/fninf.2018.00092
- [1] Ness, T. V., Chintaluri, C., Potworowski, J., Leski, S., Glabska, H., Wojcik, D. K., et al. (2015). Modelling and analysis of electrical potentials recorded in microelectrode arrays (MEAs). *Neuroinformatics* 13:403–426. doi: 10.1007/s12021-015-9265-6
- [1] Shaozhong Deng (2008), *Journal of Electrostatics* 66:549-560. DOI: 10.1016/j.elstat.2008.06.003
- [1] Shaozhong Deng (2008), *Journal of Electrostatics* 66:549-560. DOI: 10.1016/j.elstat.2008.06.003
- [1] Shaozhong Deng (2008), *Journal of Electrostatics* 66:549-560. DOI: 10.1016/j.elstat.2008.06.003
- [1] Pettersen KH, Hagen E, Einevoll GT (2008) Estimation of population firing rates and current source densities from laminar electrode recordings. *J Comput Neurosci* (2008) 24:291–313. DOI 10.1007/s10827-007-0056-4
- [2] Hagen E, Fossum JC, Pettersen KH, Alonso JM, Swadlow HA, Einevoll GT (2017) *Journal of Neuroscience*, 37(20):5123-5143. DOI: <https://doi.org/10.1523/JNEUROSCI.2715-16.2017>
- [1] Næss S, Chintaluri C, Ness TV, Dale AM, Einevoll GT and Wójcik DK (2017) Corrected Four-sphere Head Model for EEG Signals. *Front. Hum. Neurosci.* 11:490. doi: 10.3389/fnhum.2017.00490

- [2] Hagen E, Næss S, Ness TV and Einevoll GT (2018) Multimodal Modeling of Neural Network Activity: Computing LFP, ECoG, EEG, and MEG Signals With LFPy 2.0. *Front. Neuroinform.* 12:92. doi: 10.3389/fninf.2018.00092
- [1] Nunez and Srinivasan, Oxford University Press, 2006
- [1] Nunez and Srinivasan, Oxford University Press, 2006
- [1] Hämäläinen M., et al., *Reviews of Modern Physics*, Vol. 65, No. 2, April 1993.
- [2] Sarvas J., *Phys.Med. Biol.*, 1987, Vol. 32, No 1, 11-22.
- [1] Huang, Parra, Haufe (2016) The New York Head—A precise standardized volume conductor model for EEG source localization and tES targeting. *Neuroimage* 140:150–162. doi: 10.1016/j.neuroimage.2015.12.019
- [2] Naess et al. (2020) Biophysical modeling of the neural origin of EEG and MEG signals. *bioRxiv* 2020.07.01.181875. doi: 10.1101/2020.07.01.181875

PYTHON MODULE INDEX

|

lfpkit, [7](#)

lfpkit.eegmegcalc, [33](#)

lfpkit.models, [11](#)

C

`calc_potential()` (if-
pykit.OneSphereVolumeConductor method),
 27

`calculate_B()` (if-*pykit.eegmegcalc.InfiniteHomogeneousVolCondMEG*
 method), 38

`calculate_B()` (if-*pykit.eegmegcalc.SphericallySymmetricVolCondMEG*
 method), 40

`calculate_H()` (if-*pykit.eegmegcalc.InfiniteHomogeneousVolCondMEG*
 method), 38

`calculate_H()` (if-*pykit.eegmegcalc.SphericallySymmetricVolCondMEG*
 method), 40

`CellGeometry` (class in *lfpykit*), 9

`CurrentDipoleMoment` (class in *lfpykit*), 12

D

`distort_cell_geometry()` (if-*pykit.RecMEAElectrode*
 method), 25

F

`find_closest_electrode()` (if-
pykit.eegmegcalc.NYHeadModel method),
 42

`FourSphereVolumeConductor` (class in if-
pykit.eegmegcalc), 33

G

`get_dipole_potential()` (if-
pykit.eegmegcalc.FourSphereVolumeConductor
 method), 34

`get_dipole_potential()` (if-
pykit.eegmegcalc.InfiniteVolumeConductor
 method), 35

`get_transformation_matrix()` (if-
pykit.CurrentDipoleMoment method), 12

`get_transformation_matrix()` (if-
pykit.eegmegcalc.FourSphereVolumeConductor
 method), 34

`get_transformation_matrix()` (if-
pykit.eegmegcalc.InfiniteHomogeneousVolCondMEG
 method), 38

`get_transformation_matrix()` (if-
pykit.eegmegcalc.InfiniteVolumeConductor
 method), 36

`get_transformation_matrix()` (if-
pykit.eegmegcalc.NYHeadModel method),
 42

`get_transformation_matrix()` (if-
pykit.eegmegcalc.SphericallySymmetricVolCondMEG
 method), 40

`get_transformation_matrix()` (if-
pykit.LaminarCurrentSourceDensity method),
 30

`get_transformation_matrix()` (if-*pykit.LinearModel*
 method), 11

`get_transformation_matrix()` (if-
pykit.LineSourcePotential method), 16

`get_transformation_matrix()` (if-
pykit.OneSphereVolumeConductor method),
 28

`get_transformation_matrix()` (if-
pykit.PointSourcePotential method), 14

`get_transformation_matrix()` (if-
pykit.RecExtElectrode method), 22

`get_transformation_matrix()` (if-
pykit.RecMEAElectrode method), 26

`get_transformation_matrix()` (if-
pykit.VolumetricCurrentSourceDensity
 method), 32

I

`InfiniteHomogeneousVolCondMEG` (class in if-
pykit.eegmegcalc), 36

`InfiniteVolumeConductor` (class in if-
pykit.eegmegcalc), 35

L

`LaminarCurrentSourceDensity` (class in *lfpykit*), 29

`lfpykit`

module, 7

`lfpykit.eegmegcalc`

module, 33

`lfpykit.models`

module, 11

LinearModel (*class in lfpypykit*), 11

LineSourcePotential (*class in lfpypykit*), 15

M

module

lfpypykit, 7

lfpypykit.eegmegcalc, 33

lfpypykit.models, 11

N

NYHeadModel (*class in lfpypykit.eegmegcalc*), 41

O

OneSphereVolumeConductor (*class in lfpypykit*), 26

P

PointSourcePotential (*class in lfpypykit*), 13

R

RecExtElectrode (*class in lfpypykit*), 17

RecMEAElectrode (*class in lfpypykit*), 22

return_closest_idx() (*lfpypykit.eegmegcalc.NYHeadModel method*),
42

rotate_dipole_to_surface_normal() (*lfpypykit.eegmegcalc.NYHeadModel method*),
42

S

set_dipole_pos() (*lfpypykit.eegmegcalc.NYHeadModel method*), 43

SphericallySymmetricVolCondMEG (*class in lfpypykit.eegmegcalc*), 39

V

VolumetricCurrentSourceDensity (*class in lfpypykit*),
31