Primer on LFPy

CNS 2017 Tutorial T4 part II

Espen Hagen, espen.hagen@fys.uio.no

Department of Physics
Centre for Integrative Neuroplasticity (CINPLA.org)
University of Oslo, Norway

Antwerp, July 15, 2017

Outline

Topics

- Why model extracellular potentials?
- What is **LFPy**?
- Why `Python`
  - `class` introduction
- **LFPy**:
  - Introduction
  - Requirements
  - Installation
  - Class overview
  - Examples
  - Further reading
- Extracellular potentials

Why model extracellular potentials?

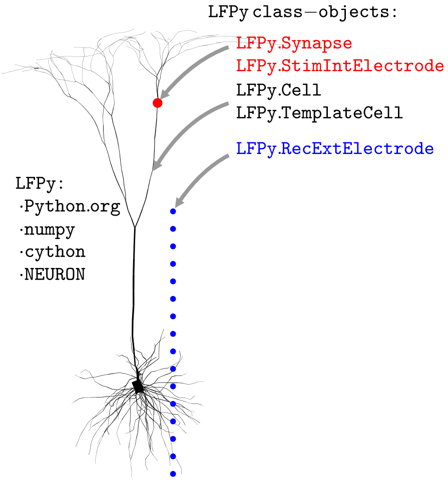
- **Improve understanding of experimental measurements:**
  - Action potential waveform:
    - Gold et al. *J Neurophysiol* (2006)
    - Pettersen & Einevoll. *Biophys J* (2008)
    - Hagen et al. *J Neurosci Methods* (2015)
    - Ness et al. *Neuroinform* (2015)
  - Active ion channel component of LFP:
    - Schomburg et al. *J. Neurosci* (2012)
    - Reimann et al. *Neuron* (2013)
    - Ness et al. *J Physiol* (2016)
  - Spectral content of LFP:
    - Lindén et al. *J Comput Neurosci* (2010)
    - Tomsett et al. *Brain Struct Funct* (2014)
  - Reach of LFP:
    - Lindén et al. *Neuron* (2011)
    - Łęski et al. *PLOS Comput Biol* (2013)

Why model extracellular potentials?

- **Improve understanding of experimental measurements:**
  - Effect of network correlations:
    - Hagen et al. *Cereb Cortex* (2016)
  - Single-axon pre- and post-synaptic LFPs
    - McColgan et al. *BioR$_x$iv* (2017)
    - Hagen et al. *J Neurosci* (2017)
- **Methods validation (with known ground truth):**
  - Spike sorting:
    - Franke et al. *Proc IEEE Eng Med Biol Soc* (2010)
    - Einevoll et al. *Curr Op Neurobiol* (2012),
    - Thorbergsson et al. *J Neurosci Methods* (2013)
    - Hagen et al. *J Neurosci Methods* (2015)
  - Current-source density (CSD) reconstruction:
    - Pettersen et al. *J Comput Neurosci* (2008)
    - Łęski et al. *Neuroinform* (2011)
    - Głąbska et al. *PLOS One* (2014)
    - Ness et al. *Neuroinform* (2015)

LFPy - Introduction

- Methods implementation
  - multicompartment neurons
  - extracellular potentials
  - (networks)
- Implemented in `Python`
- Uses `NEURON` under the hood
- Class objects represent:
  - cells
  - synapses
  - intracellular electrodes
  - extracellular electrodes
- Homepages w. documentation:
  http://LFPy.github.io
  https://github.com/LFPy/LFPy



LFPy

Developers:

- Henrik Lindén
- Espen Hagen
- Szymon Łęski
- Eivind S. Norheim
- Klas H. Pettersen
- Gaute T. Einevoll
- It's open source - anyone can contribute!

Homepage:

- https://LFPy.github.io

compneuro.umb.no/LFPy/

LFPy 1.0.0 documentation » next | modules | index

Table Of Contents

Next topic

Download LFPy

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

LFPy

Local Field Potentials in Python

LFPy Homepage

LFPy is a Python package for calculation of extracellular potentials from multicompartment neuron models. It relies on the NEURON simulator and uses the Python interface it provides.

LFPy provides a set of easy-to-use Python classes for setting up your model, running your simulations and calculating the extracellular potentials arising from activity in your model neuron. If you have a model working in NEURON already, it is likely that it can be adapted to work with LFPy.

The extracellular potentials are calculated from transmembrane currents in multi-compartment neuron models using the line-source method (Holt & Koch, J Comp Neurosci 1999), but a simpler point-source method is also available. The calculations assume that the neuron are surrounded by an infinite extracellular medium with homogeneous and frequency independent conductivity, and compartments are assumed to be at least at a minimal distance from the electrode (which can be specified by the user). For more information on the biophysics underlying the numerical framework used see this coming book chapter:

- K.H. Pettersen, H. Linden, A.M. Dale and G.T. Einevoll: Extracellular spikes and current-source density, in *Handbook of Neural Activity Measurement*, edited by R. Brette and A. Destexhe, Cambridge, to appear [preprint PDF, 5.7MB]

In the present version, LFPy is mainly designed for simulation of single neurons, but the forward modeling scheme is in general applicable to neuronal populations. These aspects, and the biophysical assumptions behind LFPy is described in our paper on the package appearing in Frontiers in Neuroinformatics, entitled "LFPy: A tool for biophysical simulation of extracellular potentials generated by detailed model neurons", appearing as part of the research topic "Python in Neuroscience II".

Citation:

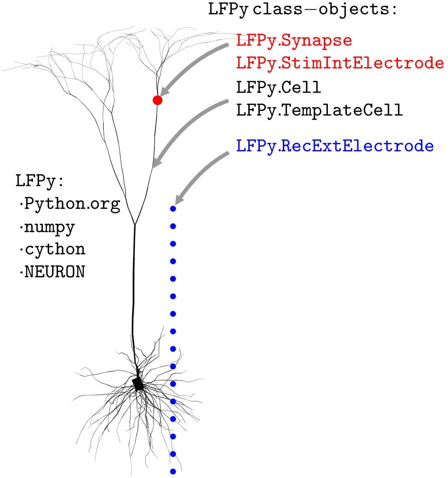
- Linden H, Hagen E, Leski S, Norheim ES, Pettersen KH and Einevoll GT (2014). LFPy: A tool for biophysical simulation of extracellular potentials generated by detailed model neurons. Front. Neuroinform. 7:41. doi: 10.3389/fninf.2013.00041

LFPy was developed in the Computational Neuroscience Group, Department of Mathematical Sciences and Technology, at the Norwegian University of Life Sciences , in collaboration with the Laboratory of Neuroinformatics, Nencki Institute of Experimental Biology, Warsaw, Poland. The effort was supported by International Neuroinformatics Coordinating Facility (INCF), The Research Council of Norway (eScience, NevroNor) and EU-FP7 (BrainScaleS).

LFPy - Introduction

Why Python?

- Open source
- Easy, flexible coding
- Plethora of available packages for visualizations and analysis
- http://pypi.python.org: $>$ 120000 packages
- Interfacing other programming languages and software
  - C, C++, Fortran, ...
  - NEURON, NEST, Brian, Genesis, PyNN, ...



LFPy - Introduction

Python class objects

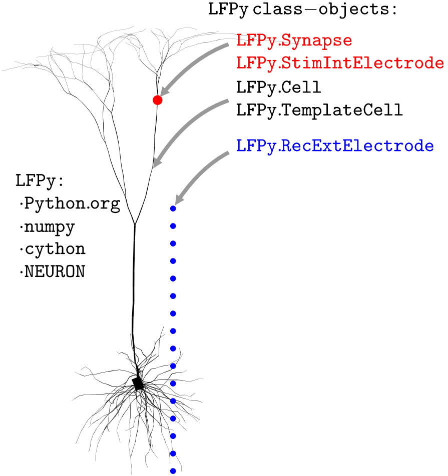
- Object Oriented Programming (OOP)
- arbitrary amounts and kinds of data
- contains methods and attributes
- created runtime, modifiable

```
class MyClass(object):
    def __init__(self, arg0=1, arg1='hi!'):
        '''init class MyClass'''
        self.arg0 = arg0
        self.arg1 = arg1
    def myClassMethod(self, arg=2):
        '''do some operation'''
        return self.arg0 + arg
if __name__ == "__main__":
    c = MyClass(arg0=3,
                arg1='hello')
    print c.myClassMethod(arg2=3)
    print c.arg1
```

LFPy - Requirements

Python dependencies:

- Hard:
  - neuron
  - Cython
  - numpy
  - scipy
  - matplotlib
  - unittest
- Soft:
  - ipython
  - jupyter notebook
  - h5py
  - mpi4py



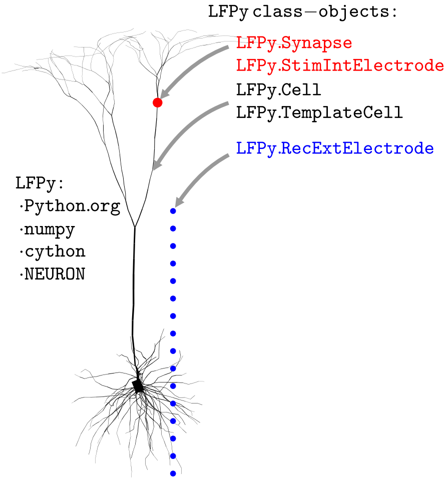
LFPy - Requirements

Python distributions:

- Anaconda
- Enthought Canopy
- Python(x,y)
- ...

LFPy platforms:

- *nix (Linux, Unix)
- OS X/macOS
- Windows



LFPy - Installation

Installation:

Easy method:

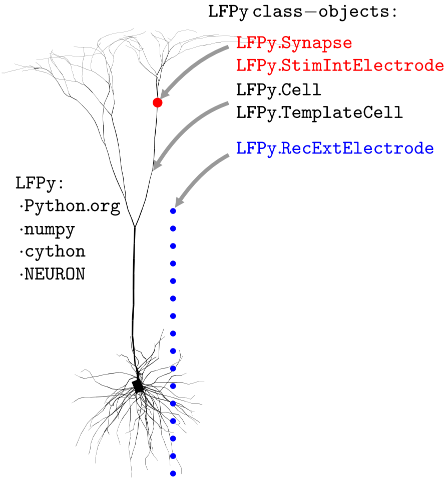
- `easy_install pip --user`
- `pip install LFPy --user`

or as super user:

- `sudo easy_install pip`
- `sudo pip install LFPy`

Upgrading previous install:

- `pip install --upgrade LFPy --no-deps`



LFPy - Installation

Install using `LFPy` source files:

- Tar.gz-archive:

```
cd $HOME/Sources
wget https://github.com/LFPy/LFPy/archive/v1.1.3.tar.gz
tar -xzvf v1.1.3.tar.gz
```

  (unzipped in folder `LFPy-1.1.3`)

- Development version:

```
cd $HOME/Sources
git clone https://github.com/LFPy/LFPy.git LFPy
git checkout v1.1.3
```

- `git`: Much-used distributed source code management system. See https://git-scm.com

LFPy - Installation

Install using `LFPy` source files:

- Perform a local installation:

```
cd $HOME/Sources/LFPy
python setup.py install --user
```

- Global installation (super user):

```
sudo python setup.py install
```

- Use LFPy from source folder (for active development):

```
python setup.py build_ext -i
export PYTHONPATH=$HOME/Repos/LFPy/:$PYTHONPATH
```

LFPy - Installation

Test installation:

With Python:

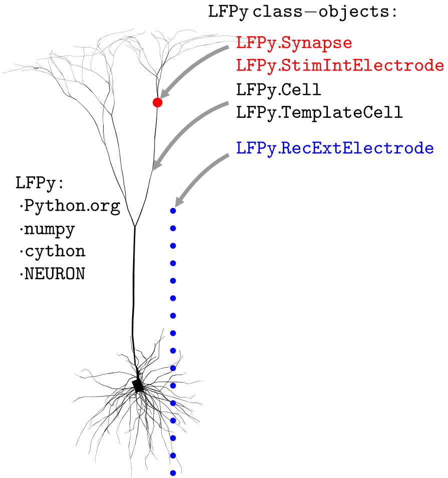
- `python -c "import LFPy"`

  `NEURON -- Release 7.3 (1089:ecf32eddfbc7)...`

With NEURON:

- `nrngui -python -c "import LPFy"`

  `NEURON -- Release 7.3 (1089:ecf32eddfbc7)...`



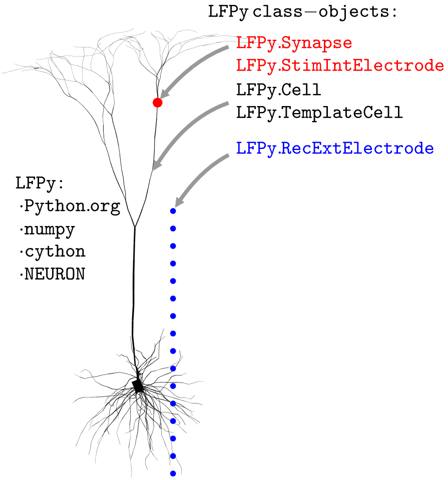
LFPy - Class overview

Main LFPy classes:

- Cell
- Synapse
- StimIntElectrode
- RecExtElectrode

Auxilliary classes and functions:

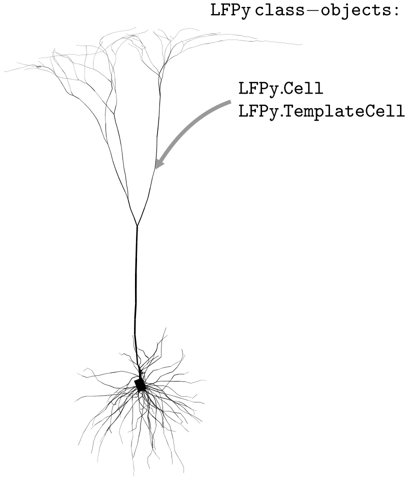
- TemplateCell
- lfpcalc.calc_lfp.*
- inputgenerators.*
- tools.*



LFPy - Class overview

LFPy.Cell:

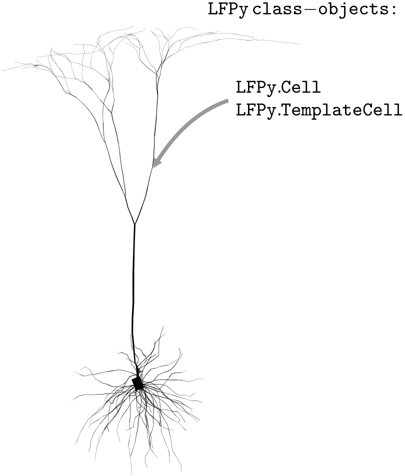
- Uses NEURON under the hood
- Sets neuron properties:
  - neuron geometry
  - membrane mechanisms ('pas', 'hh', ...)
  - number of compartments ('d_lambda' rule; Hines&Carnevale. *Neuroscientist* (2001))
  - Sets cell location and rotation
- Simulation control
  - duration
  - record variables



LFPy - Class overview

LFPy.Cell:

- Keyword arguments:
  - morphology file (morphology)
  - passive parameters (rm, cm, Ra, V_init, e_pas)
  - time and space discretization (nsegs_methods)
  - simulation duration (tstopms)
  - custom codes (custom_code)

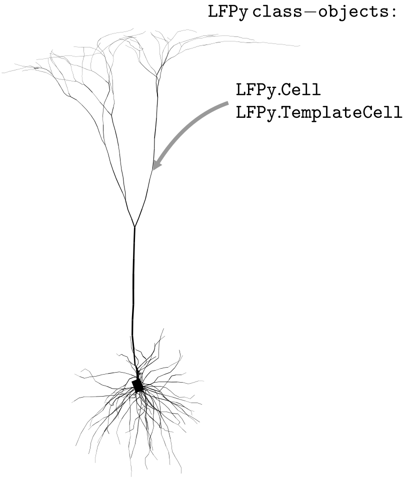


LFPy - Class overview

LFPy.Cell:

- Download morphology (`j4a.hoc` file) https://goo.gl/twpdrX
- Create parameter dictionary

```
# Define cell parameters
cell_parameters = dict(
    morphology='j4a.hoc',
    rm = 30000.,    #ohm cm2
    cm = 1.,        #uF cm-2
    Ra = 150.,      #ohm cm
    v_init = -65., #mV
    e_pas = -65.,  #mV
    tstopms = 100.,#ms
    custom_code = None,
    )
```



LFPy - Class overview

LFPy.Cell:

- Create `cell` object:

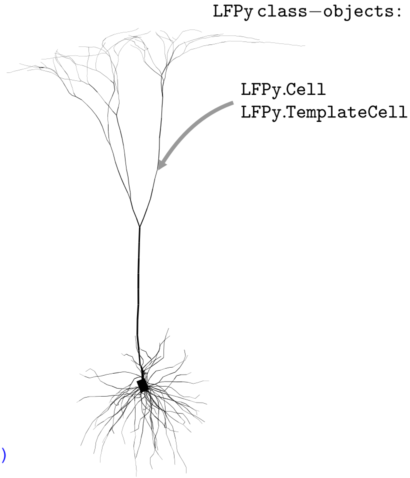
```
cell = LFPy.Cell(
         **cell_parameters)
```

- Position and align cell:

```
cell.set_pos(0., 0. ,0.)
cell.set_rotation(x=4.99,
         y=-4.33, z=3.14)
```

- (`cell` stimulation)
- simulate & plot `cell` response

```
cell.simulate(rec_isyn=True/False,
              rec_istim=True/False)
plt.plot(cell.tvec, cell.somav)
```



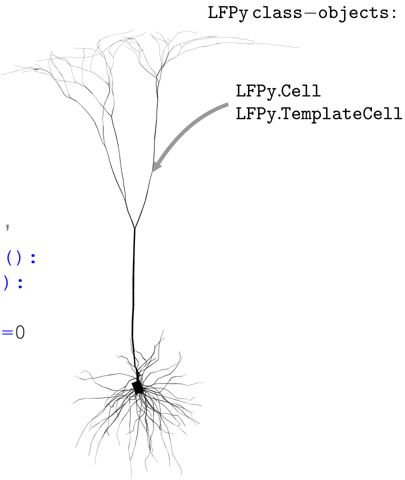
LFPy - Class overview

LFPy.Cell:

- Customizing the model:
  - **LFPy.Cell** defaults to passive membrane mechanism
  - custom_fun argument

```
def my_biophys():
    '''set custom parameters'''
    for sec in neuron.h.allsec():
        if "soma" in sec.name():
            sec.insert("hh")
            sec(0.5).pas.g_pas=0
        else:
            pass
cell = LFPy.Cell(morphology,
    custom_fun =
[my_biophys],
    **cell_parameters)
```

  - custom_code point to code files



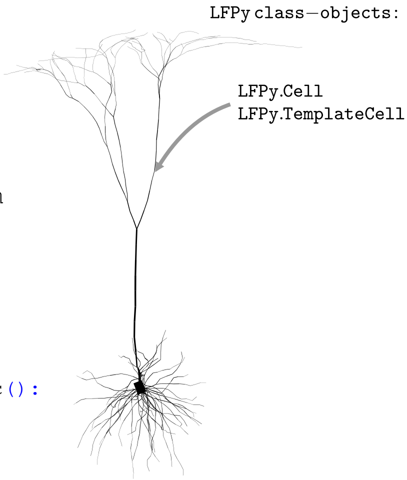
LFPy - Class overview

LFPy.Cell:

- Important `Cell`-class methods:
  - `c.get_idx`
  - `c.get_closest_idx`
  - `c.get_rand_idx_area_norm`
  - `c.get_idx_name`
- Important class attributes:
  - `c.totnsegs`

```
i = 0:
for sec in neuron.h.allsec():
    for seg in sec:
        i += 1
```

  - `c.*start, c.*mid, c.*end`
    $\star \in$ [x, y, z]



LFPy - Class overview

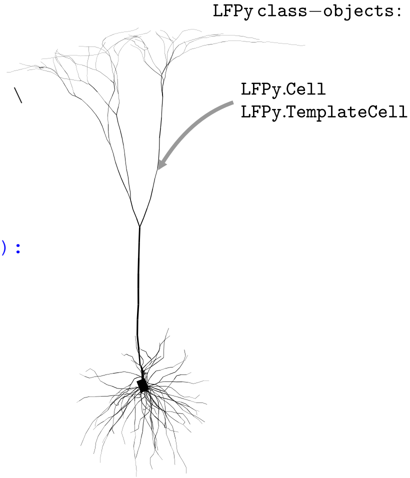
LFPy.Cell:

- Tip on drawing `cell`:

```
from matplotlib.collections import \
    PolyCollection
import matplotlib.pyplot as plt

cell = LFPy.Cell('j4a.hoc')
zips = []
for x, z in cell.get_idx_polygons():
    zips.append(zip(x, z))
polycol = PolyCollection(zips,
           edgecolors='none',
          facecolors='gray')

fig, ax = plt.subplots(1)
ax.add_collection(polycol)
ax.axis(ax.axis('equal'))
plt.show()
```



LFPy - Class overview

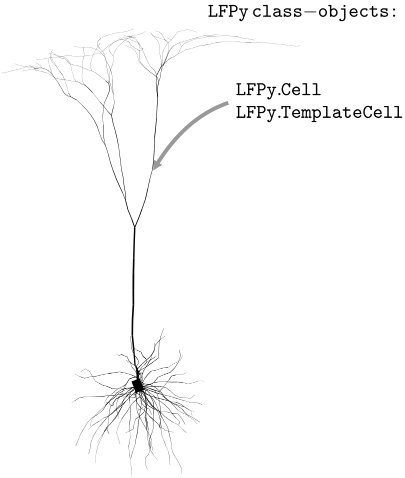
LFPy.Cell:

- **LFPy.Cell** objects are transparent to NEURON:

```
import LFPy
import neuron.h as nrn

cell = LFPy.Cell('j4a.hoc')
for sec in nrn.soma:
    sec.insert("hh")
    for seg in sec:
        seg.pas.g_pas = 0.
```

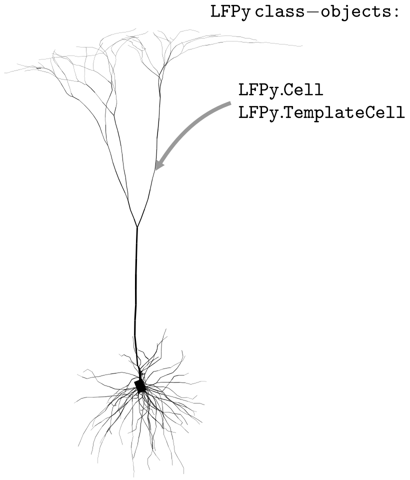
(only 'HH' conductances in soma)



LFPy - Class overview

LFPy.Cell: Sections created in Python:

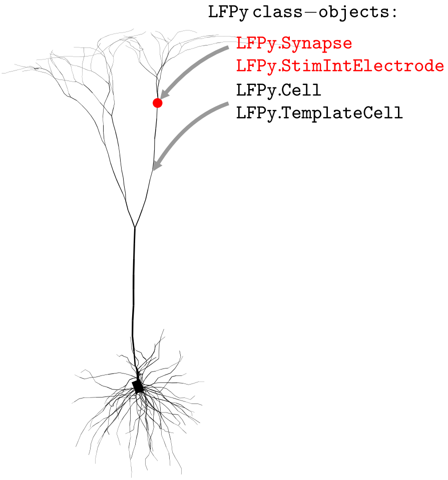
```
import neuron, LFPy
soma = neuron.h.Section(name='soma')
dend = neuron.h.Section(name='dend')
soma.L = 30
soma.diam = 30
dend.L = 300
dend.diam = 2
dend.connect(soma, 1, 0)
cell = LFPy.Cell(morphology=None,
                 delete_sections=False,
                 rm = 30000,
                 cm = 1.0,
                 Ra = 150,
                 tstopms = 100)
...
cell.simulate()
plt.plot(cell.tvec, cell.somav)
```



LFPy - Class overview

LFPy.Synapse:

- Attach synapse-objects onto cell
- event-activated point currents
- Keyword arguments:
  - cell-object
  - compartment index (idx)
  - synapse type (ExpSyn, Exp2syn, AlphaSynapse)
  - mechanism arguments (e, tau, weight,...)
  - record synapse current (record_current)
- Feed in activation times: drawn offline or on the fly



LFPy - Class overview

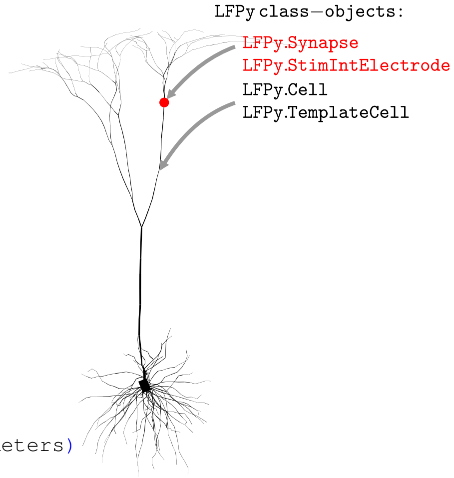
LFPy.Synapse:

- Define synapse parameters

```
synapse_parameters = dict(
    idx = cell.get_closest_idx(
                x=-200.,
                y=0.,
                z=800.),
    syntype = 'ExpSyn',
    e = 0.,
    tau = 5.,
    weight = .001,
    record_current = True,)
```

- Create synapse, set activation time

```
syn = LFPy.Synapse(cell,
                   **synapse_parameters)
```



LFPy - Class overview

LFPy.Synapse:

- Create synapse, set activation time

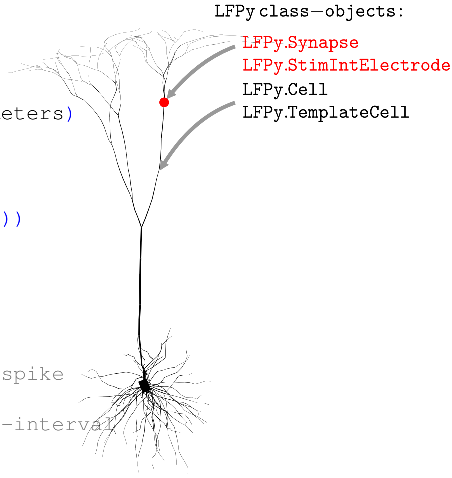
```
syn = LFPy.Synapse(cell,
                   **synapse_parameters)
```

- set offline activation time(s)

```
syn.set_spike_times(np.array([20.]))
```

- generate activation time(s) on the fly

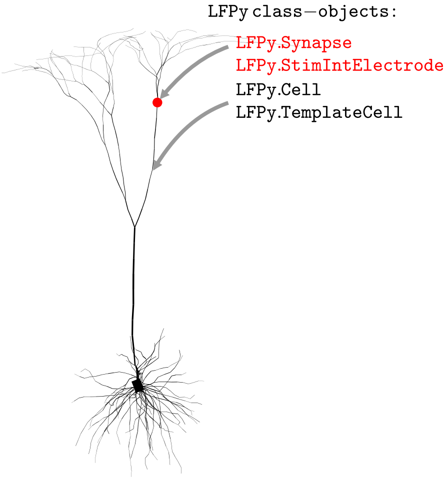
```
syn.set_spike_times_w_netstim(
    noise=1, # Poisson statistics
    start=0, # likely time of 1st spike
    number=1E3, # number of spikes
    interval=20, # mean interspike-interval
    )
```



LFPy - Class overview

LFPy.StimIntElectrode:

- Represents intracellular point electrodes
  - voltage clamp (`VClamp`)
  - current clamp (`IClamp`)
  - single-electrode V clamp (`SEClamp`)
- Not modeled as transmembrane currents
- currents into intracellular medium
- Mimics experimental setups



LFPy - Class overview

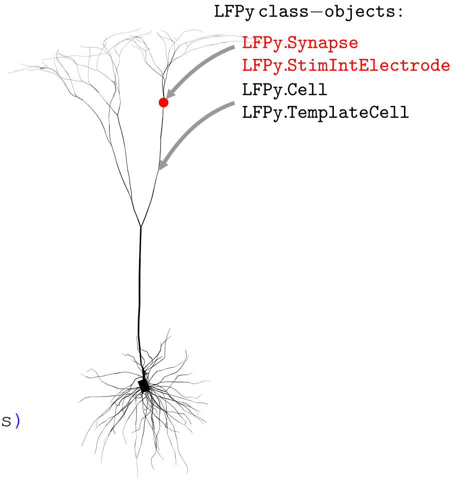
LFPy.StimIntElectrode:

- Define point process parameters

```
# Define synapse parameters
pointproc_parameters = dict(
    idx = 0,
    record_current = True,
    pptype = 'IClamp',
    amp = 1,
    dur = 20,
    delay = 10)
```

- Create point process:

```
stim =
LFPy.StimIntElectrode(cell,
            **pointproc_parameters)
```



LFPy - Class overview

Plotting stimulus currents

- run

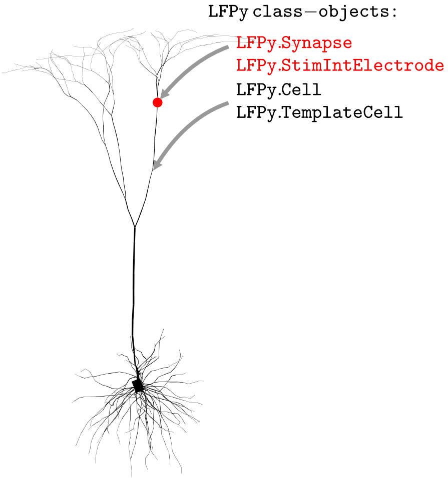
```
cell.simulate(rec_isyn=True,
              rec_istim=True)
```

- draw `LFPy.Synapse` current

```
plt.subplot(211)
plt.plot(cell.tvec, syn.i)
```

- draw `LFPy.StimIntElectrode` current

```
plt.subplot(212)
plt.plot(cell.tvec, stim.i)
```

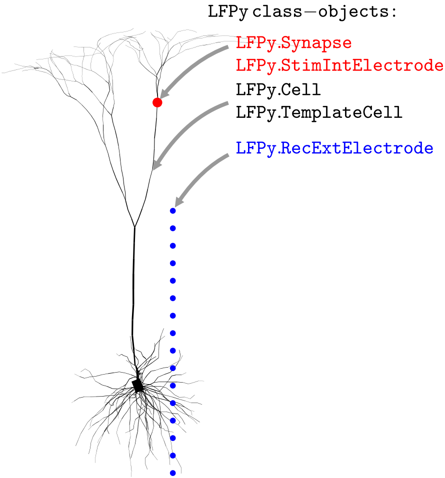


Questions?

LFPy - Class overview

LFPy.RecExtElectrode:

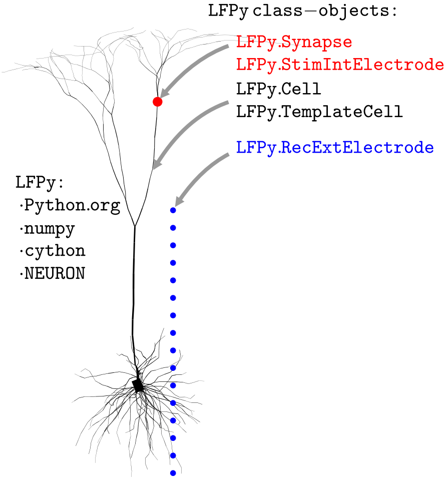
- Extracellular recording devices
- Main arguments:
  - `cell` objects (geometry, currents)
  - contact point coordinates `x,y,z`
  - extracellular conductivity `sigma`
  - `method` (pointsource/linesource)
- Optional:
  - radius and surface normal vectors of contacts
  - *n*-point surface area averaged potential



LFPy - Class overview

LFPy.RecExtElectrode:

```
# Run simulation, record currents
cell.simulate(rec_imem=True,
              rec_isyn=True)
# Define electrode parameters
electrode_parameters = {
    'sigma' : 0.3,
    'x' : [-130., -220.],
    'y' : [   0.,    0.],
    'z' : [   0.,  700.],
}
# Create electrode object
electrode = LFPy.RecExtElectrode(
            cell,
            **electrode_parameters)
# Calculate LFPs
electrode.calc_lfp()
plt.plot(cell.tvec, electrode.LFP.T)
plt.show()
```



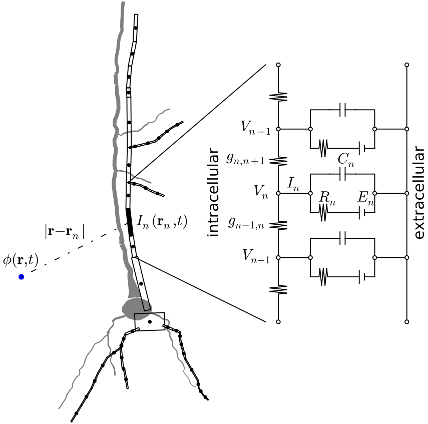
Forward modeling of extracellular potentials

Biophysical background:

- Current balance intracellular node point, compartment n:

$$I_n = C_n \frac{dV_n}{dt} - \frac{V_n - E_n}{R_n} =$$
$$g_{n,n+1}(V_{n+1} - V_n)$$
$$-g_{n-1,n}(V_n - V_{n-1})$$

- Simulated using **NEURON** (neuron.yale.edu) Hines et al. (2009))
- Extracellular potentials are computed from I_n



Lindén et al. (2014)

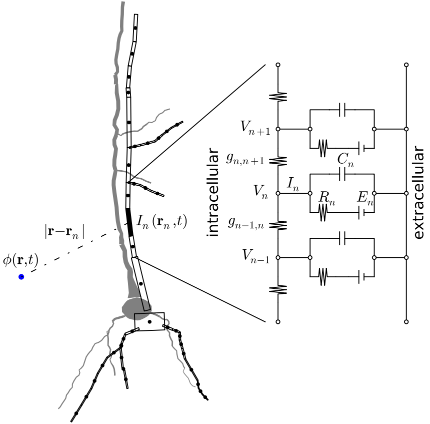
Forward modeling of extracellular potentials

Biophysical background:

- Poisson's equation in electrostatics

$$\nabla \cdot (\sigma \nabla \phi) = -C$$

$\phi(\mathbf{r}, t)$ - electric potential
$C(\mathbf{r}, t)$ - current source density
$\sigma(\mathbf{r})$ - conductivity



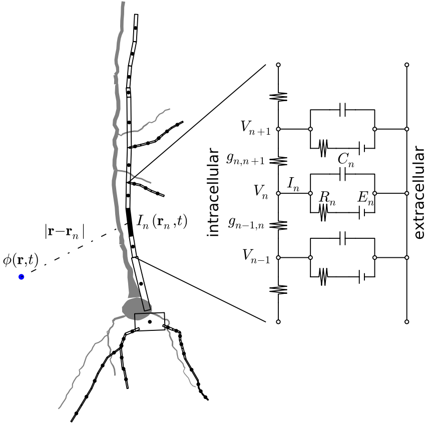
Lindén et al. (2014)

Forward modeling of extracellular potentials

Biophysical background:

- Assumptions:
    - Quasi-static approximation of Maxwell's equations
    - Extracellular medium:
        - linear
        - isotropic
        - homogeneous
        - ohmic

      (scalar, real σ)
- $\phi(r \to \infty) = 0$



Lindén et al. (2014)

Forward modeling of extracellular potentials

Biophysical background:

- Quasi-static approximation of Maxwell's equations:

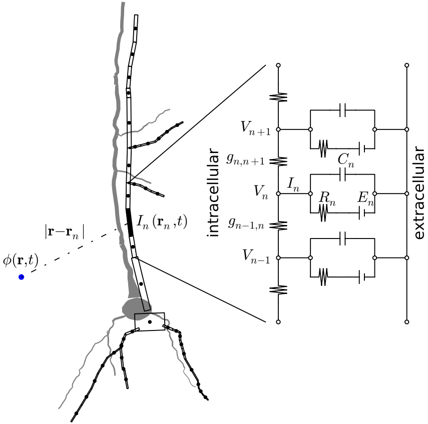
$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \approx 0$$

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \times \mathbf{B} = \mu(\mathbf{J} + \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}) \approx \mu \mathbf{J}$$

- **E** - electric field; ρ - charge density; ϵ_0 - free space permittivity; **B** - magnetic field; μ - permeability; **J** - sum of ohmic and polarization currents



Lindén et al. (2014)

Forward modeling of extracellular potentials

Biophysical background:

- Source current in conductive media
- Ohm's law in passive nonmagnetic media

$$\mathbf{J} = \sigma \mathbf{E} + \frac{\partial \mathbf{P}}{\partial t} \approx \sigma \mathbf{E}$$

$$(\mathbf{P} = (\epsilon - \epsilon_0)\mathbf{E} : \text{ polarization})$$

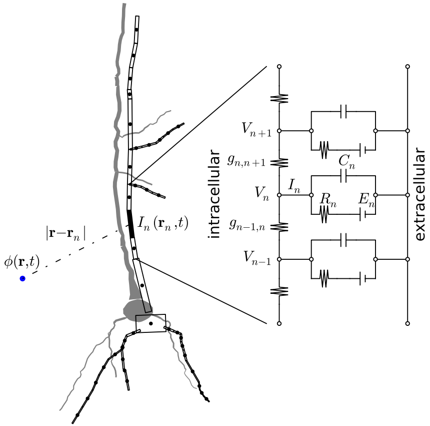
$$\nabla \times \mathbf{E} = 0$$

$$\rightarrow \mathbf{E} = -\nabla \phi \text{ , thus}$$

$$\mathbf{J} = -\sigma \nabla \phi$$

$$(C \equiv \nabla \cdot \mathbf{J})$$

- σ - assumed scalar, real



Lindén et al. (2014)

Forward modeling of extracellular potentials

Biophysical background:

- Assuming a point current source

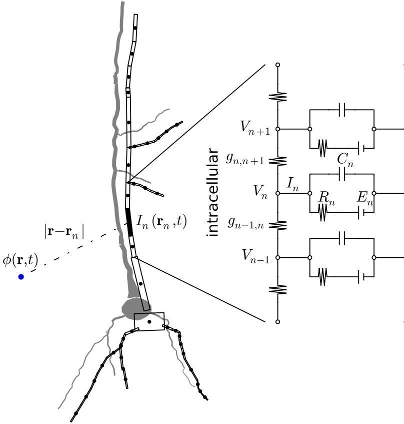
$$\mathbf{J} = \frac{I_0}{4\pi r^2}\hat{\mathbf{r}} \ , \ \hat{\mathbf{r}} : \text{ radial unit vector}$$

$$-\sigma\nabla\phi = \frac{I_0}{4\pi r^2}\hat{\mathbf{r}} \ , \ \nabla\phi = \frac{\partial\phi}{\partial r}$$

$$\frac{\partial\phi}{\partial r} = -\frac{I_0}{4\pi\sigma r^2}$$

- integration w. respect to r yields

$$\phi(r) = \frac{I_0}{4\pi\sigma r}$$



Lindén et al. (2014)

Forward modeling of extracellular potentials

Biophysical background:

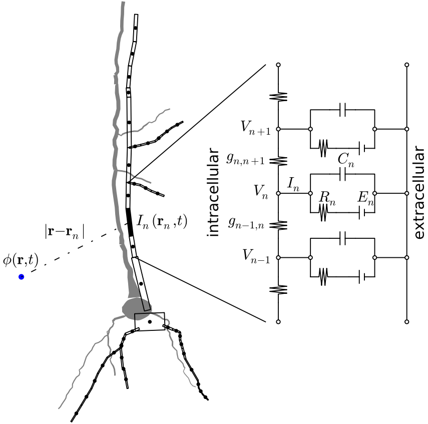
- Point current source

$$\phi(\mathbf{r}, t) = \frac{1}{4\pi\sigma} \frac{I_0(t)}{|\mathbf{r} - \mathbf{r}_0|} \ ,$$

where $\mathbf{r}$ is measurement location,
$\mathbf{r}_0$ source location

- Linear summation N point sources

$$\phi(\mathbf{r}, t) = \frac{1}{4\pi\sigma} \sum_{n=1}^{N} \frac{I_n(t)}{|\mathbf{r} - \mathbf{r}_n|}$$



Lindén et al. (2014)

Forward modeling of extracellular potentials

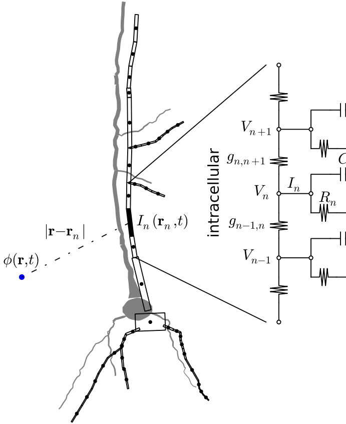
Biophysical background:

- Line sources (homog. current density)

$$\phi(\mathbf{r}, t) = \frac{1}{4\pi\sigma} \sum_{n=1}^{N} I_n(t) \int \frac{d\mathbf{r}_n}{|\mathbf{r} - \mathbf{r}_n|}$$

$$= \frac{1}{4\pi\sigma} \sum_{n=1}^{N} \frac{I_n(t)}{\Delta s_n} \ln \left| \frac{\sqrt{h_n^2 + r_{\perp n}^2} - h_n}{\sqrt{l_n^2 + r_{\perp j}^2} - l_n} \right|$$

- Δs_n - segment length; h_n - longitudal distance to one end of segment; $r_{\perp n}$ - perpendicular distance to segment axis; $l_n = \delta s_n + h_n$.
- see Holt & Koch. (1999), *J Comput Neurosci* 6:169-184



Lindén et al. (2014)

LFPy - Class overview

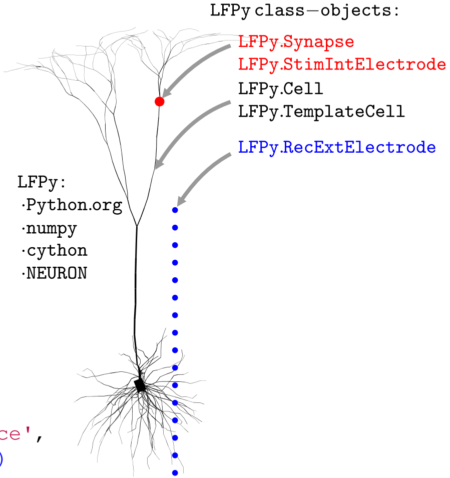
LFPy.RecExtElectrode:

- class supports
  - point sources
  - line sources
  - point soma - line dendrites
  - keyword argument

```
method in ['pointsource',
           'linesource',
           'som_as_point']
```

- Usage

```
# Create electrode object
electrode = LFPy.RecExtElectrode(
            cell, method='linesource',
            **electrode_parameters)
```



LFPy - Class overview

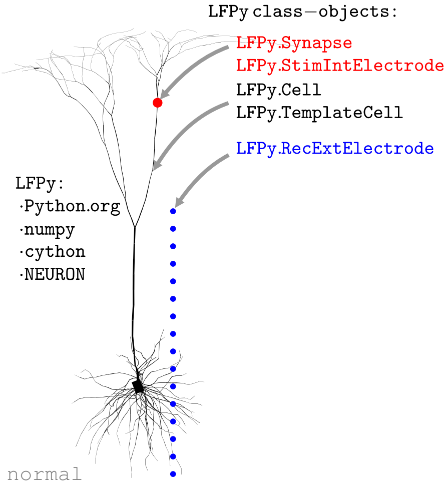
LFPy.RecExtElectrode:

- So far - point electrodes
- Real electrodes have finite extent
- "disk" electrode approximation

$$\phi_{\text{disc}}(\mathbf{u}, t) = \frac{1}{A_S} \iint_S \phi(\mathbf{u}, t)\, d^2 r$$
$$\approx \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{u}_i, t)$$

- keyword arguments:

```
r = 10. #contact radius
n = 50  #n-point average
N = np.array([[0, 1, 0]]) #surface normal
```

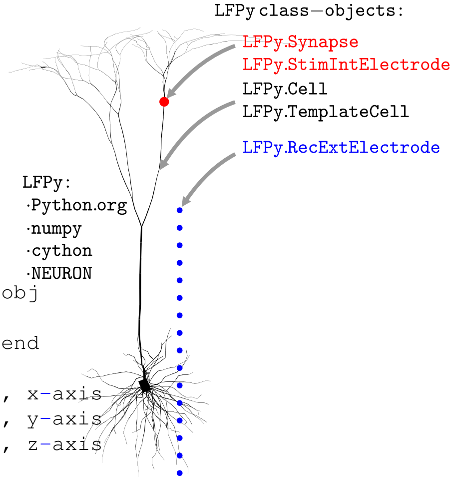


LFPy - Class overview

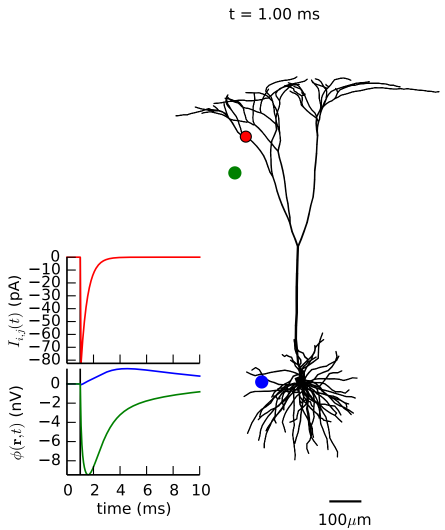
LFPy.lfpcalc.calc_lfp_*():

- Public methods
- used by **LFPy.RecExtElectrode**
  - calc_lfp_pointsource()
  - calc_lfp_linesource()
  - calc_lfp_som_as_point()
- keyword arguments:

```
cell: LFPy.Cell/LFPy.TemplateCell obj
    cell.imem
    cell.*start, cell.*mid, cell.*end
    cell.diam
x : double, extracellular position, x-axis
y : double, extracellular position, y-axis
z : double, extracellular position, z-axis
sigma : double, conductivity
```

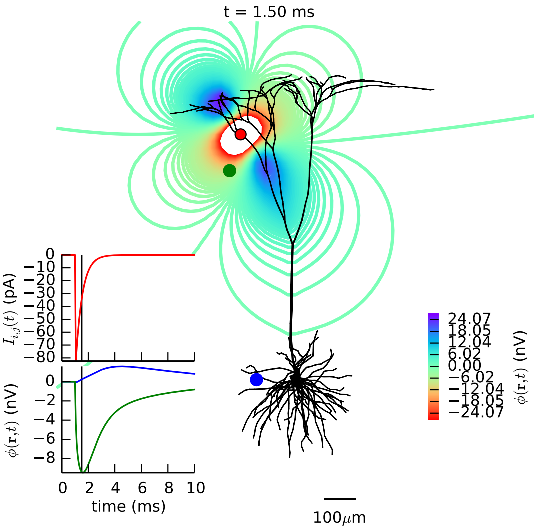


Forward modeling of extracellular potentials



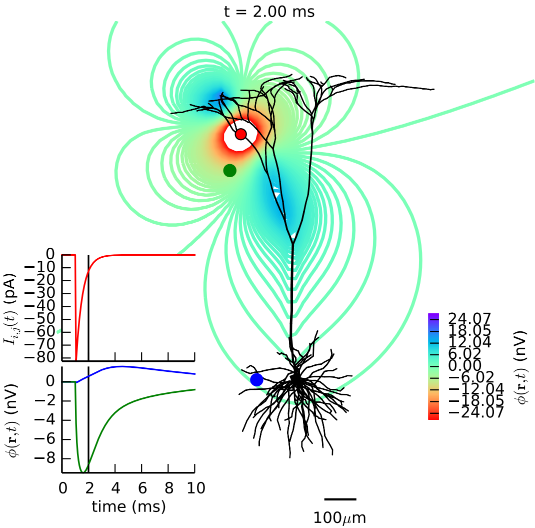
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



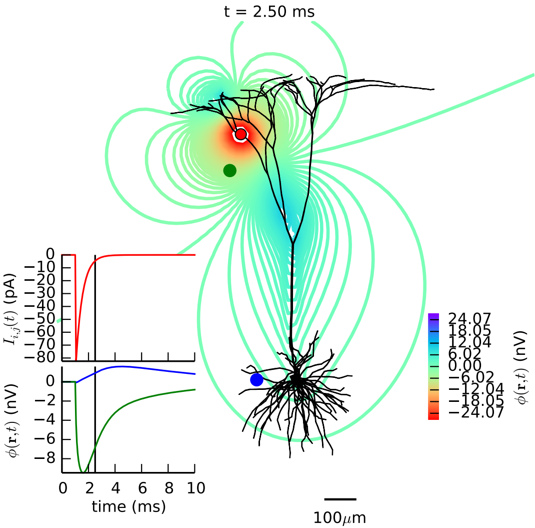
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



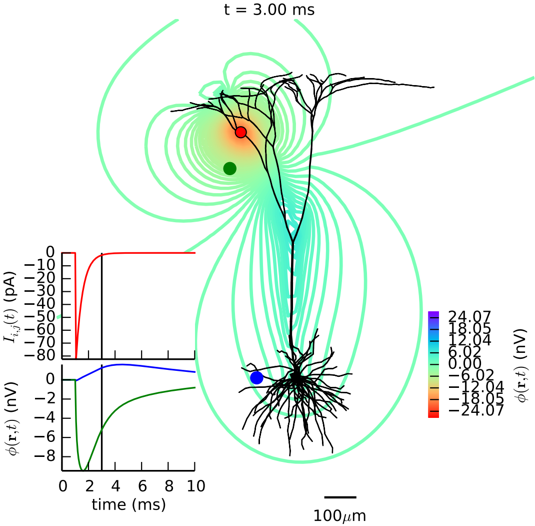
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



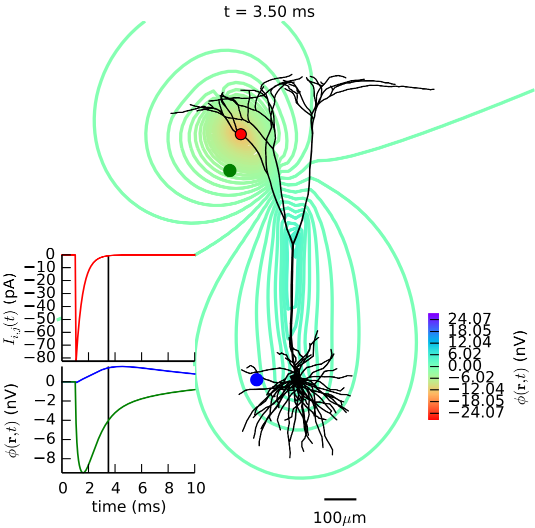
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



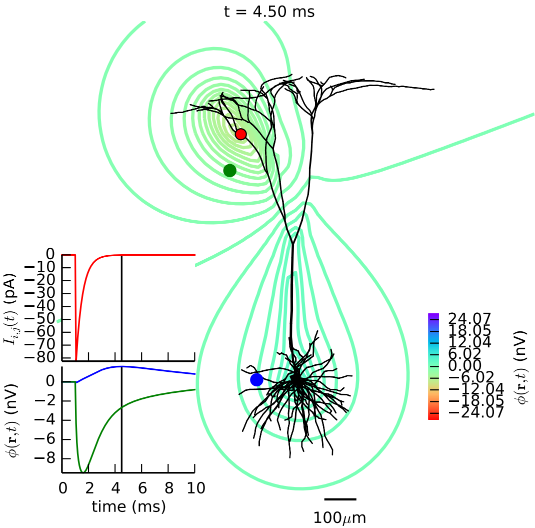
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



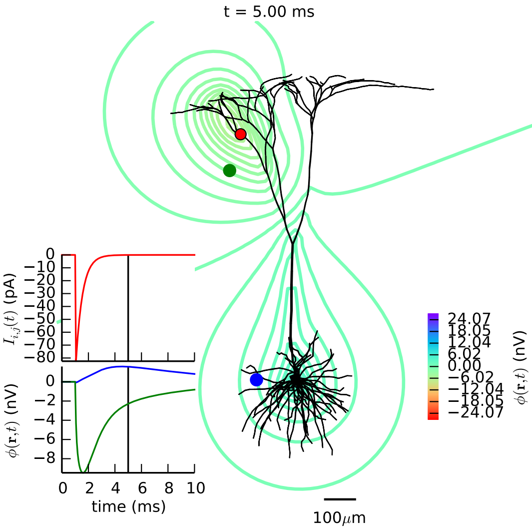
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



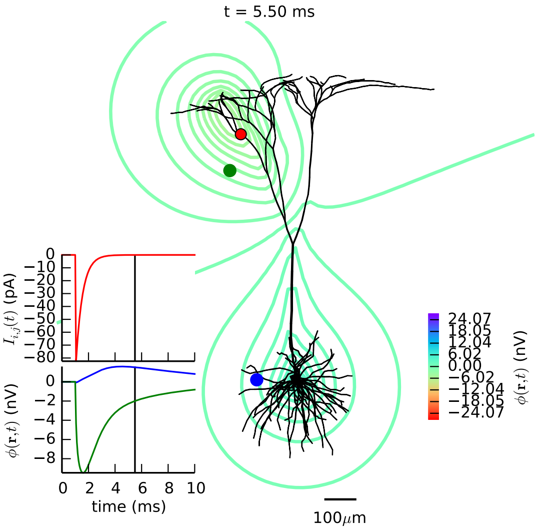
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



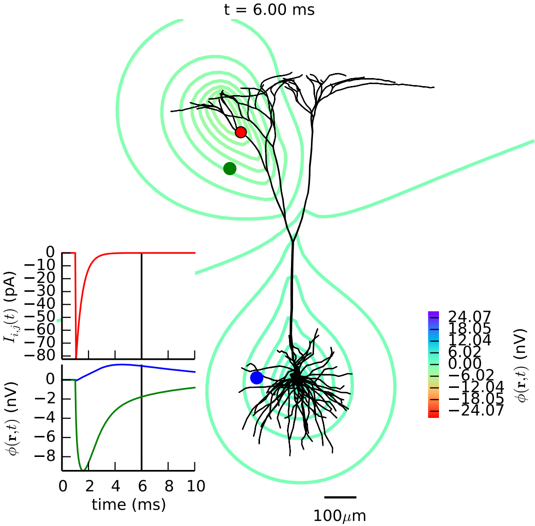
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



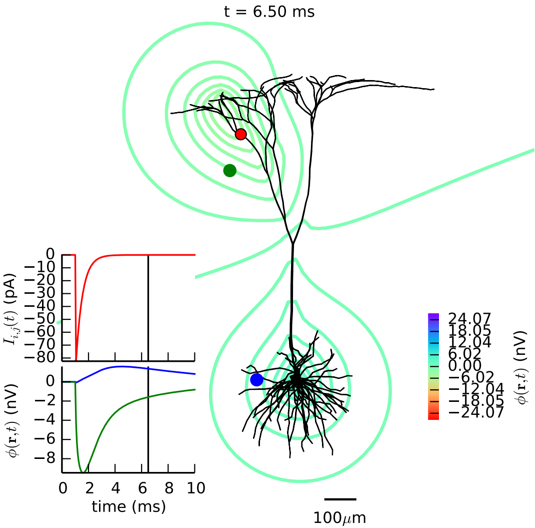
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials

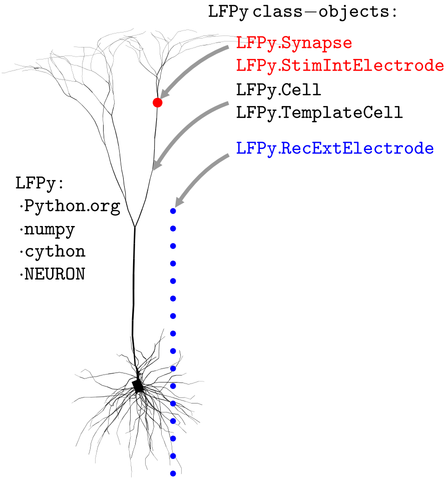


Passive propagation of synapse current input in passive cable model

LFPy - Class overview

Documentation and resources:

- LFPy homepage (http://LFPy.github.io)
- autodoc w. `sphinx`:
  `cd /path/to/LFPy`
  `sphinx-build -b html documentation docs`
  see `docs/index.html`
- IPython magic (`numpy.sin?`, `LFPy.Synapse??`)
- NEURON homepage (http://www.neuron.yale.edu/)



LFPy - Unit tests

unittest module:

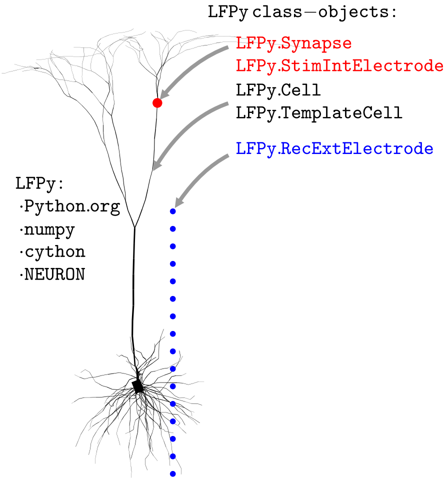
- runs code
- check if output is correct
- `LFPy` tests validate model output against analytical expressions for equivalent ball&stick models
- run tests:

```
cd /path/to/LFPy/unittests
python test_LFPy.py
```

- output:

```
----------------------------
Ran 25 tests in 9.008s

OK
```



LFPy - Ephaptic interactions

- Neuron dynamics independent of extracellular predictions!
- `LFPy.Cell.insert_v_ext(v_ext, t_ext):`

```
import LFPy, matplotlib.pyplot as plt, numpy as np
# create cell
cell = LFPy.Cell('morphologies/example_morphology.hoc')
# time vector and extracellular potential for each segment:
dt = cell.timeres_python
t_ext = np.arange(100 / dt + 1) * dt
v_ext = np.random.rand(cell.totnsegs, t_ext.size)-0.5
# insert potentials and record response:
cell.insert_v_ext(v_ext, t_ext)
cell.simulate(rec_imem=True, rec_vmem=True)
# plot
plt.matshow(v_ext); plt.axis('tight'); plt.colorbar()
plt.matshow(cell.imem); plt.axis('tight'); plt.colorbar()
plt.matshow(cell.vmem); plt.axis('tight'); plt.colorbar()
plt.show()
```

LFPy - Further reading and material

frontiers in
NEUROINFORMATICS

ORIGINAL RESEARCH ARTICLE
published: 16 January 2014
doi: 10.3389/fninf.2013.00041

LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons

Henrik Lindén[1,2†], Espen Hagen[1†], Szymon Łęski[1,3], Eivind S. Norheim[1], Klas H. Pettersen[1,4] and Gaute T. Einevoll[1*]

[1] *Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, Ås, Norway*
[2] *Department of Computational Biology, School of Computer Science and Communication, Royal Institute of Technology (KTH), Stockholm, Sweden*
[3] *Department of Neurophysiology, Nencki Institute of Experimental Biology, Warsaw, Poland*
[4] *CIGENE, Norwegian University of Life Sciences, Ås, Norway*

Edited by:
Andrew P. Davison, Centre National de la Recherche Scientifique, France

Reviewed by:
Nicholas T. Carnevale, Yale University School of Medicine, USA
Shyam Diwakar, Amrita University, India

Electrical extracellular recordings, i.e., recordings of the electrical potentials in the extracellular medium between cells, have been a main work-horse in electrophysiology for almost a century. The high-frequency part of the signal ($\gtrsim$500 Hz), i.e., the *multi-unit activity (MUA)*, contains information about the firing of action potentials in surrounding neurons, while the low-frequency part, the *local field potential (LFP)*, contains information about how these neurons integrate synaptic inputs. As the recorded extracellular signals arise from multiple neural processes, their interpretation is typically ambiguous and

http://dx.doi.org/10.3389/fninf.2013.00041

LFPy - Further reading and material

lfpy.github.io

LFPy 1.1.0 documentation » next | modules | index

Table Of Contents

Next topic

Download LFPy

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

LFPy Homepage

LFPy is a Python package for calculation of extracellular potentials from multicompartment neuron models. It relies on the NEURON simulator and uses the Python interface it provides.

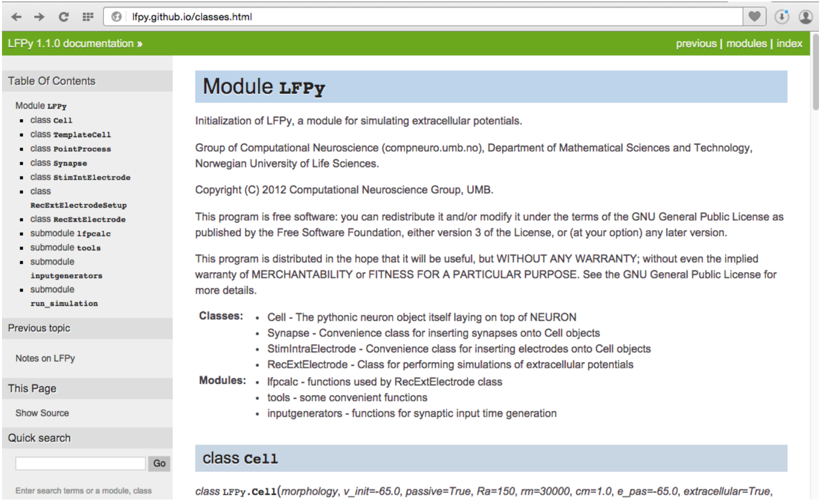
Clone LFPy on GitHub.com: `git clone https://github.com/LFPy/LFPy.git`

LFPy provides a set of easy-to-use Python classes for setting up your model, running your simulations and calculating the extracellular potentials arising from activity in your model neuron. If you have a model working in NEURON already, it is likely that it can be adapted to work with LFPy.

The extracellular potentials are calculated from transmembrane currents in multi-compartment neuron models using the line-source method (Holt & Koch, J Comp Neurosci 1999), but a simpler point-source method is also available. The calculations assume that the neuron are surrounded by an infinite extracellular medium with homogeneous and frequency independent conductivity, and compartments are assumed to be at least at a minimal distance from the electrode (which can be specified by the user). For more information on the biophysics underlying the numerical framework used see this coming book chapter:

- K.H. Pettersen, H. Linden, A.M. Dale and G.T. Einevoll: Extracellular spikes and current-source density, in *Handbook of Neural Activity Measurement*, edited by R. Brette and A. Destexhe, Cambridge, to appear [preprint PDF, 5.7MB]

LFPy - Further reading and material



lfpy.github.io/classes.html

LFPy 1.1.0 documentation »

previous | modules | index

Table Of Contents

Previous topic

Notes on LFPy

This Page

Show Source

Quick search

Go

Enter search terms or a module, class

Module LFPy

Initialization of LFPy, a module for simulating extracellular potentials.

Group of Computational Neuroscience (compneuro.umb.no), Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences.

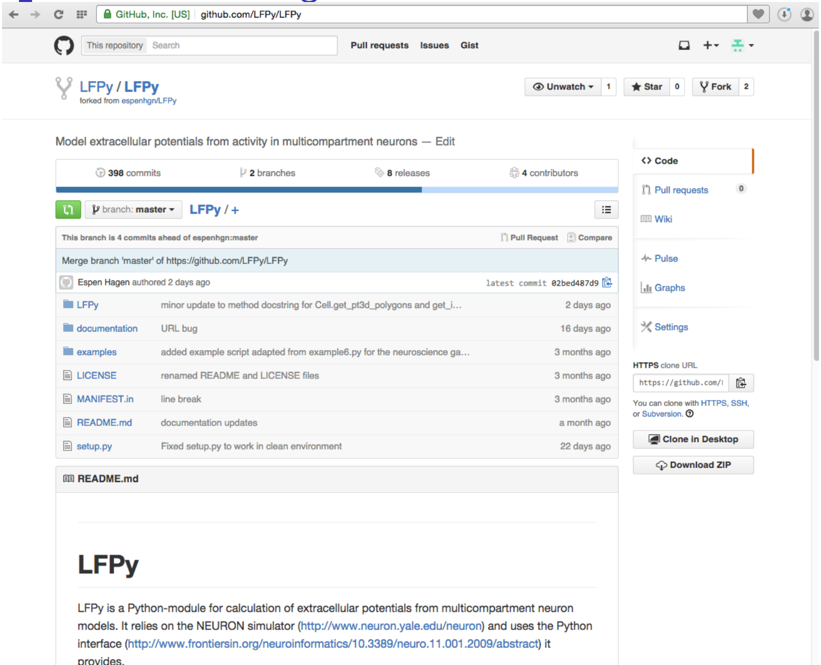
Classes:
- Cell - The pythonic neuron object itself laying on top of NEURON
- Synapse - Convenience class for inserting synapses onto Cell objects
- StimIntraElectrode - Convenience class for inserting electrodes onto Cell objects
- RecExtElectrode - Class for performing simulations of extracellular potentials

Modules:
- lfpcalc - functions used by RecExtElectrode class
- tools - some convenient functions
- inputgenerators - functions for synaptic input time generation

class Cell

class LFPy.**Cell**(*morphology, v_init=-65.0, passive=True, Ra=150, rm=30000, cm=1.0, e_pas=-65.0, extracellular=True,*

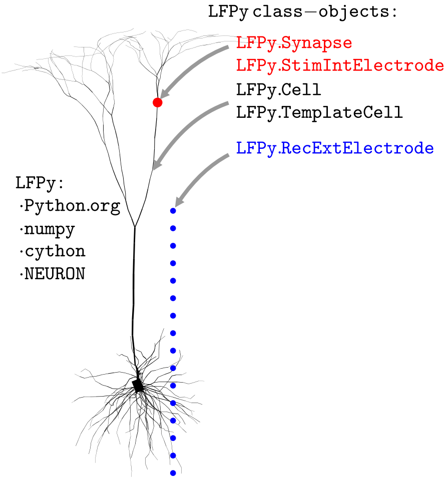
LFPy - Further reading and material



LFPy - Examples

Example **Python** `files`

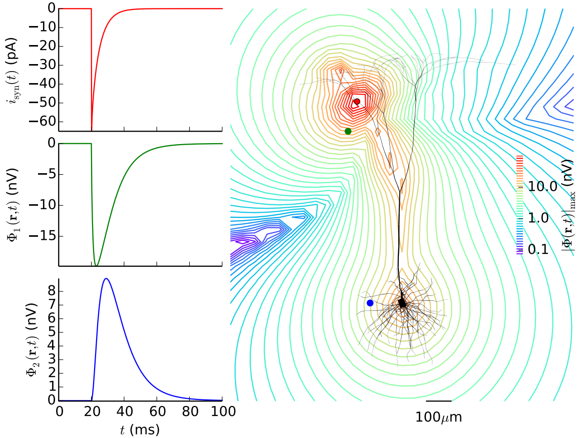
- `/path/to/LFPy/examples`
- Compute extracellular potentials
  - passive vs. active models
  - single-synapse vs. multi-synapse responses
  - extracellular action potential waveforms
  - population signal
- All use **`LFPy.Cell`**, **`LFPy.Synapse`**, **`LFPy.RecExtElectrode`**, **`...`**



LFPy - Examples

/path/to/LFPy/examples/example1.py

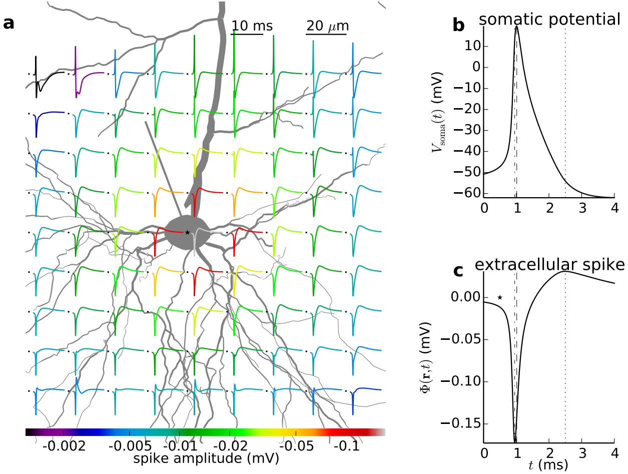
Apical synapse response, passive cable model



LFPy - Examples

`/path/to/LFPy/examples/example2.py`

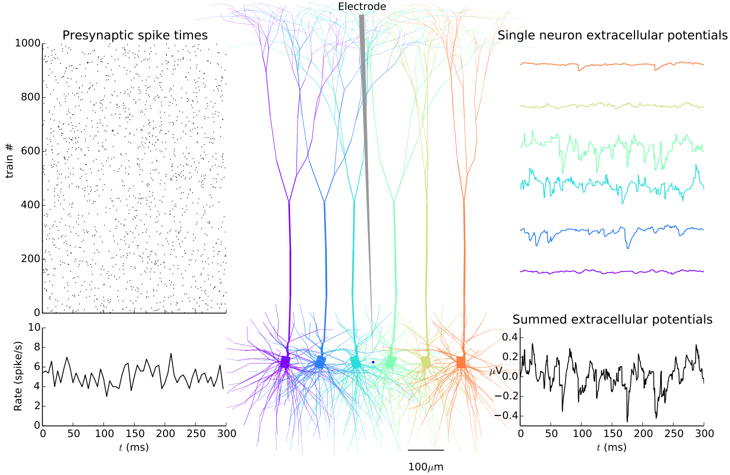
Layer 5b action potential (Hay et al. 2011), `LFPy.TemplateCell`



LFPy - Examples

`/path/to/LFPy/examples/example3.py`

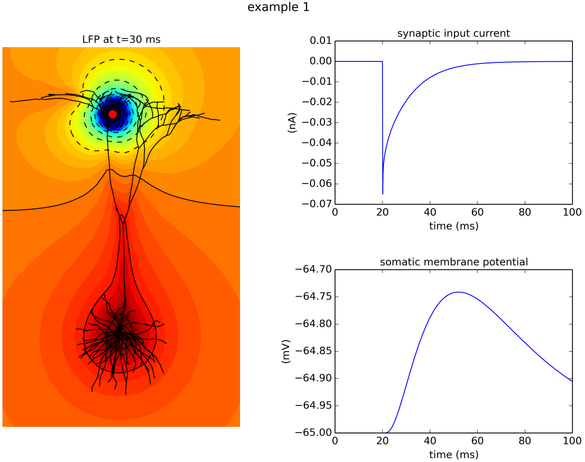
Extracellular potentials of small model population, shared input



LFPy - Examples

/path/to/LFPy/examples/example4.py

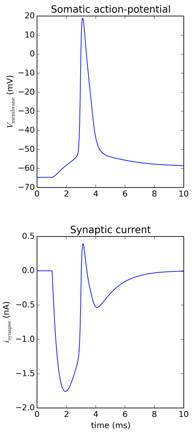
Extracellular potentials, single-synapse input current



LFPy - Examples

`/path/to/LFPy/examples/example5.py`

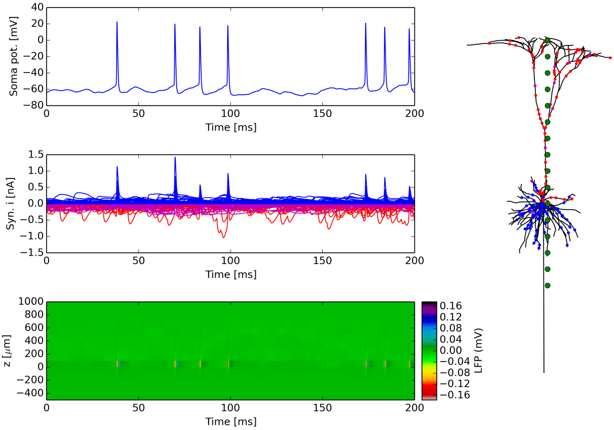
Extracellular potentials for action-potential of L5 pyramidal cell



LFPy - Examples

`/path/to/LFPy/examples/example6.py`

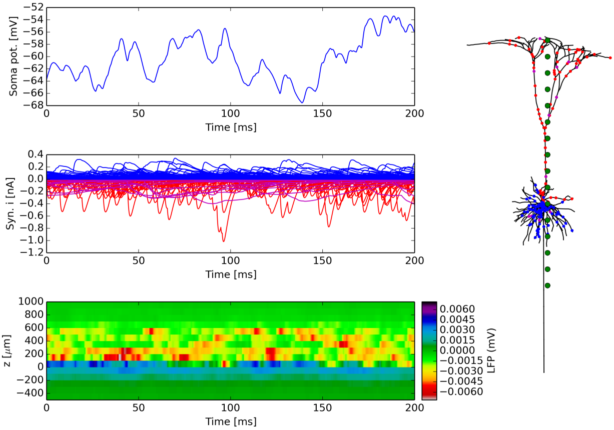
Extracellular potentials, synapse currents, somatic voltage, distributed synapses, active model



LFPy - Examples

`/path/to/LFPy/examples/example7.py`

Extracellular potentials, synapse currents, somatic voltage, distributed synapses, passive model



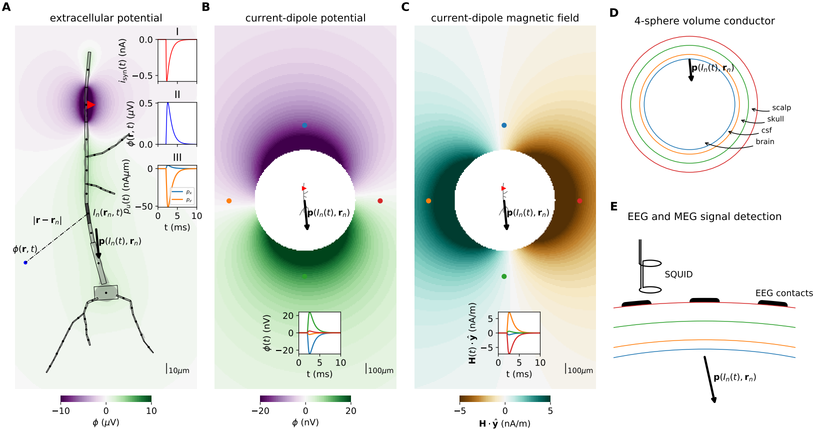
Questions?

LFPy version 2

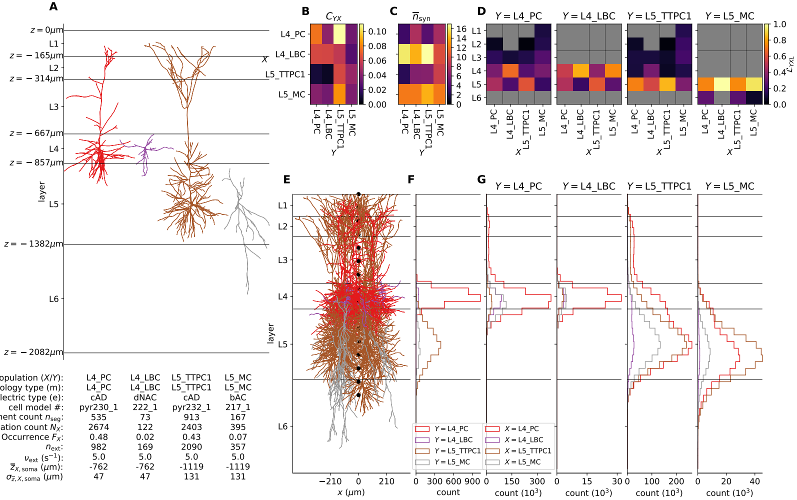
- Under active development
- Features:
  - simulation of network activity
  - concurrent EP predictions (no $I_m(t)$ storage)
  - connection-set algebra (CSA, M Djurfeldt 2012)
  - calculations of current-dipole moments
  - support for anisotropic and inhomogeneous media
  - 4-sphere volume-conductor model
  - scalp EEG and MEG signal predictions
  - MPI parallelism for HPC facilities
  - Python 2.7, 3.4-3.6 support
- Check out poster #127!
- Development codes on GitHub:

```
git clone https://github.com/LFPy/LFPy.git
cd LFPy
git checkout master
```

LFPy v.2

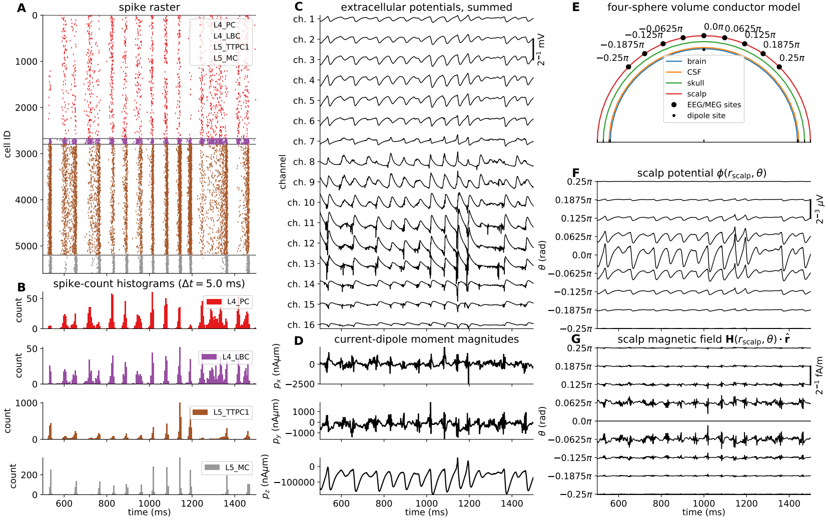

A
extracellular potential
B
current-dipole potential
C
current-dipole magnetic field
D
4-sphere volume conductor
scalp
skull
csf
brain
E
EEG and MEG signal detection
SQUID
EEG contacts
t (ms)
10μm
100μm
ϕ (μV)
ϕ (nV)
H · ŷ (nA/m)

LFPy v.2



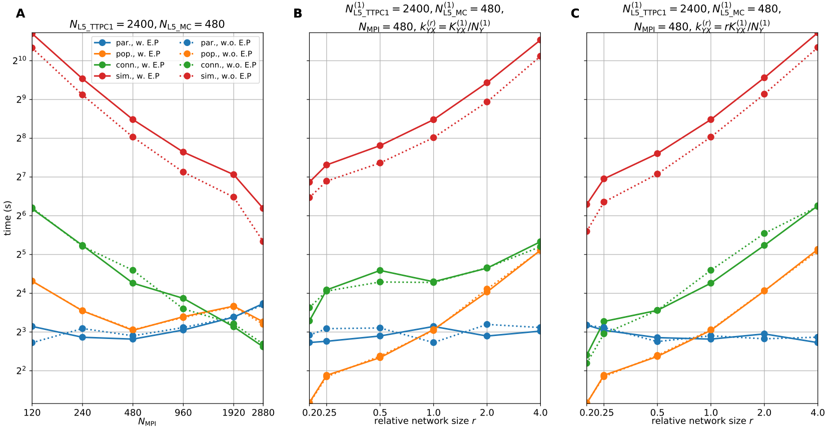
population (X/Y):	L4_PC	L4_LBC	L5_TTPC1	L5_MC
morphology type (m):	L4_PC	L4_LBC	L5_TTPC1	L5_MC
electric type (e):	cAD	dNAC	cAD	bAC
cell model #:	pyr230_1	222_1	pyr232_1	217_1
segment count n_{seg}:	535	73	913	167
population count N_X:	2674	122	2403	395
Occurrence F_X:	0.48	0.02	0.43	0.07
n_{ext}:	982	169	2090	357
ν_{ext} (s^{-1}):	5.0	5.0	5.0	5.0
$\bar{z}_{X,soma}$ (μm):	-762	-762	-1119	-1119
$\sigma_{\bar{z},X,soma}$ (μm):	47	47	131	131

LFPy v.2



A
spike raster
L4_PC
L4_LBC
L5_TTPC1
L5_MC
cell ID
time (ms)
B
spike-count histograms (Δt = 5.0 ms)
count
C
extracellular potentials, summed
channel
2^-1 mV
D
current-dipole moment magnitudes
p_x (nAμm)
p_y (nAμm)
p_z (nAμm)
E
four-sphere volume conductor model
brain
CSF
skull
scalp
EEG/MEG sites
dipole site
F
scalp potential φ(r_scalp, θ)
θ (rad)
2^-3 μV
G
scalp magnetic field H(r_scalp, θ) · r̂
2^-1 fA/m

LFPy v.2



A
$N_{L5\_TTPC1} = 2400, N_{L5\_MC} = 480$
par., w. E.P
pop., w. E.P
conn., w. E.P
sim., w. E.P
par., w.o. E.P
pop., w.o. E.P
conn., w.o. E.P
sim., w.o. E.P
time (s)
2^2 2^3 2^4 2^5 2^6 2^7 2^8 2^9 2^{10}
120 240 480 960 1920 2880
N_{MPI}
B
$N^{(1)}_{L5\_TTPC1} = 2400, N^{(1)}_{L5\_MC} = 480,$
$N_{MPI} = 480, k^{(r)}_{YX} = K^{(1)}_{YX}/N^{(1)}_{Y}$
0.20.25 0.5 1.0 2.0 4.0
relative network size r
C
$N^{(1)}_{L5\_TTPC1} = 2400, N^{(1)}_{L5\_MC} = 480,$
$N_{MPI} = 480, k^{(r)}_{YX} = rK^{(1)}_{YX}/N^{(1)}_{Y}$
0.20.25 0.5 1.0 2.0 4.0
relative network size r

Questions?

Cerebral Cortex, December 2016;26: 4461–4496

doi: 10.1093/cercor/bhw237
Advance Access Publication Date: 20 October 2016
Original Article

ORIGINAL ARTICLE

Hybrid Scheme for Modeling Local Field Potentials from Point-Neuron Networks

Espen Hagen[1,2,†], David Dahmen[1,†], Maria L. Stavrinou[2,3], Henrik Lindén[4,5], Tom Tetzlaff[1], Sacha J. van Albada[1], Sonja Grün[1,6], Markus Diesmann[1,7,8], and Gaute T. Einevoll[2,9]

[1]Institute of Neuroscience and Medicine (INM-6) and Institute for Advanced Simulation (IAS-6) and JARA BRAIN Institute I, Jülich Research Centre, 52425 Jülich, Germany, [2]Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, 1430 Ås, Norway, [3]Department of Psychology, University of Oslo, 0373 Oslo, Norway, [4]Department of Neuroscience and Pharmacology, University of Copenhagen, 2200 Copenhagen, Denmark, [5]Department of Computational Biology, School of Computer Science and Communication, Royal Institute of Technology, 100 44 Stockholm, Sweden, [6]Theoretical Systems Neurobiology, RWTH Aachen University, 52056 Aachen, Germany, [7]Department of Psychiatry, Psychotherapy and Psychosomatics, Medical Faculty, RWTH Aachen University, 52074 Aachen, Germany, [8]Department of Physics, Faculty 1, RWTH Aachen University, 52062 Aachen, Germany, and [9]Department of Physics, University of Oslo, 0316 Oslo, Norway

Why hybrid model scheme? - `hybridLFPy`

Extracellular potentials in neural tissue

- Low-frequency part; the local field potential (LFP, $f \lesssim 100$ Hz)
  - Highly ambiguous, difficult to analyze
  - Large number of contributing sources
  - Reflect integration of synaptic inputs, synchrony, ...
  - Local and non-local network interactions
- High-frequency part ($f \gtrsim 500$ Hz)
  - Contain information of spiking activity
  - Single-unit activity (spikes)
  - Multi-unit activity (MUA)
  - Fewer contributing sources
  - Easier to interpret
  - (channel noise ++)

Why hybrid model scheme? - `hybridLFPy`

Extracellular potentials in neural tissue

- Point-neuron network models:
  - Accurate predictions of population spiking activity
  - Efficient, easy to constrain models
  - Poor predictors of extracellular signals (e.g., `rate != LFP`)
- Biophysically detailed network models
  - Demanding to implement
  - Difficult to constrain
  - Computationally expensive
  - Extracellular signal predictions rare

Why hybrid model scheme? - `hybridLFPy`

Extracellular potentials in neural tissue

- Large-scale models necessary:
  - LFP reflects synaptic input also generated by remote populations (cortical & subcortical areas)
  - Theoretical description of the LFP needs to account for:
    - Anatomical and electrophysiological features of proximal neurons
    - Activity in the local microcircuitry
    - Large-scale (O(brain)) neuronal circuitry generating synaptic input
  - Reducibility of asynchronous networks is fundamentally limited (**O1**: Albada et al. (2015), http://arxiv.org/abs/1411.4770v3):
    - Methods to conserve 1st order statistics of network dynamics exist
    - Limitations arise if also 2nd-order statistics are to be maintained
    - Preserving correlations require preserving effective connectivity
    - Adjust synapse strength $j \propto 1/K$ and background input mean & variance

Why hybrid model scheme? - `hybridLFPy`

Extracellular potentials in neural tissue

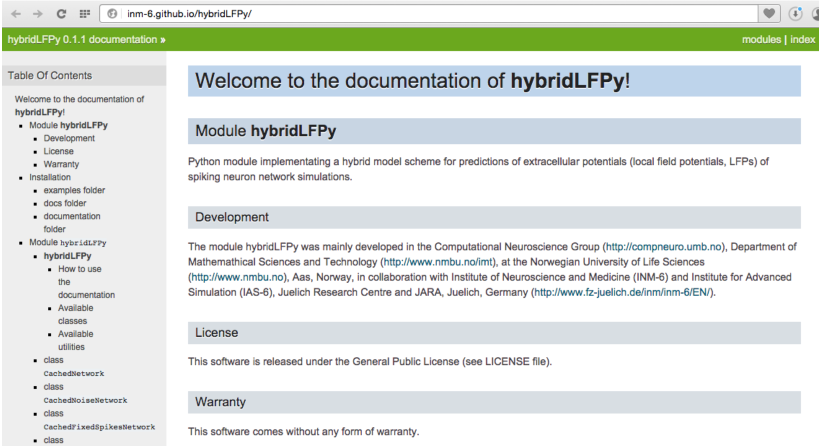
- Here:
  Hybrid scheme interfacing point-neuron network models with biophysically justified forward modelling scheme for extracellular potentials
  - LFP, CSD
  - (EEG, MEG, VSDi, ...)
- Benefits of hybrid scheme:
  - Relate network spiking activity to LFPs
  - Introduce spatial features (morphology, connectivity)
  - Simplified, passive membrane model
  - Preserve network features (cell count, synapse model ...)
  - Massive parallelism not necessary

hybridLFPy - Python package overview

Our hybrid scheme for LFP predictions is public:

- Documentation: http://inm-6.github.io/hybridLFPy
- Sources: https://github.com/INM-6/hybridLFPy
- Preprint: http://arxiv.org/abs/1511.01681
- Main classes and functions:
  - `hybridLFPy.CachedNetwork`
  - `hybridLFPy.Population`
  - `hybridLFPy.Postprocess`
  - `hybridLFPy.setup_file_dest`
- Example files - `/path/to/hybridLFPy/examples`
  - Network model: `/brunel_alpha_nest.py`
  - Hybrid model application: `/example_brunel.py`
- Python dependencies: `LFPy, nest, mpi4py, h5py, sqlite3, NeuroTools`

hybridLFPy - Python package overview



inm-6.github.io/hybridLFPy/

hybridLFPy 0.1.1 documentation » modules | index

Table Of Contents

Welcome to the documentation of **hybridLFPy**!

Module **hybridLFPy**

Python module implementating a hybrid model scheme for predictions of extracellular potentials (local field potentials, LFPs) of spiking neuron network simulations.

Development

The module hybridLFPy was mainly developed in the Computational Neuroscience Group (http://compneuro.umb.no), Department of Mathemathical Sciences and Technology (http://www.nmbu.no/imt), at the Norwegian University of Life Sciences (http://www.nmbu.no), Aas, Norway, in collaboration with Institute of Neuroscience and Medicine (INM-6) and Institute for Advanced Simulation (IAS-6), Juelich Research Centre and JARA, Juelich, Germany (http://www.fz-juelich.de/inm/inm-6/EN/).

License

This software is released under the General Public License (see LICENSE file).

Warranty

This software comes without any form of warranty.

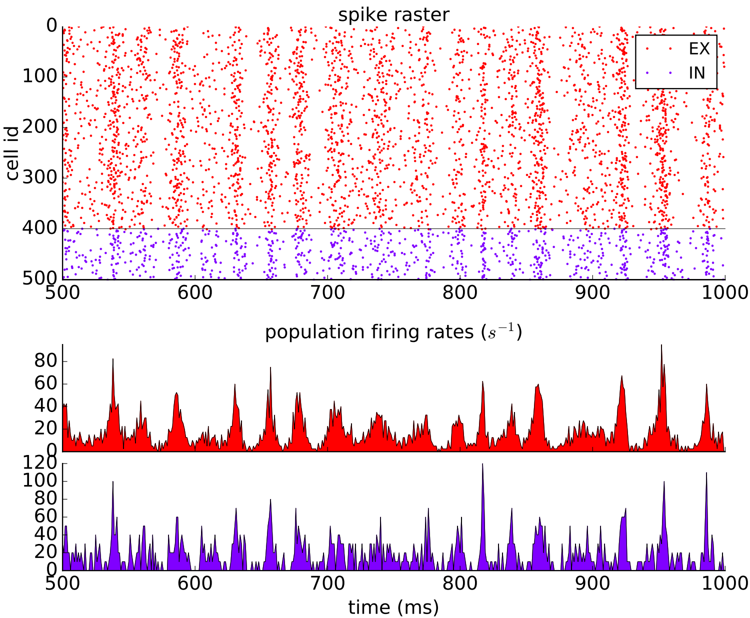
http://inm-6.github.io/hybridLFPy

`hybridLFPy` - Application with E-I network

Network model

- Two-population E-I network (Brunel, *J Comput Neurosci* (2000))
  - leaky integrate-and-fire (LIF) neurons
  - current based synapses
  - alpha-shaped PSCs
  - adapted from NEST (github.com/nest/nest-simulator) example: `/pynest/examples/brunel_alpha_nest.py`
- Modifications:
  - $N_{\mathrm{E}} + N_{\mathrm{I}} = 500$ neurons
  - $J = 1.$ mV
  - $g = -6.$
  - External Poisson spike generators removed
  - DC current input ($I_{\mathrm{DC}} \approx 300$ nA)
  - All spike events dumped to disk

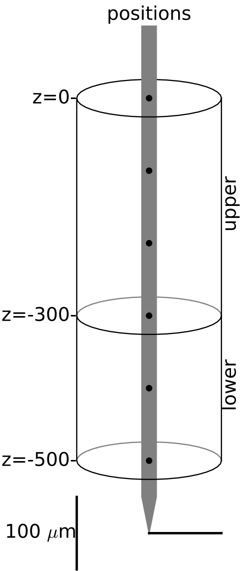
hybridLFPy - Application with E-I network



hybridLFPy - Application with E-I network

Model configuration

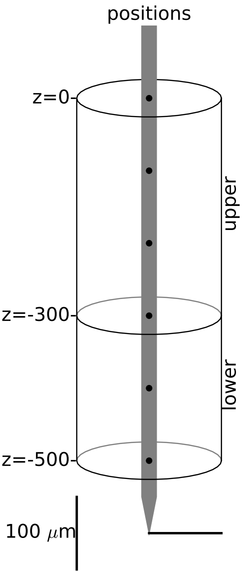
- "Layers":
  - upper $z \in [-300, 0]$ μm
  - lower $z \in [-500, -300]$ μm



hybridLFPy - Application with E-I network

Model configuration

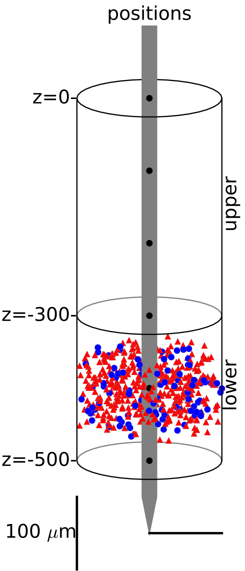
- "Layers":
  - upper $z \in [-300, 0]$ μm
  - lower $z \in [-500, -300]$ μm
- Measurements:
  - 6-channel laminar "electrode"
  - 100 μm between contacts
  - (laminar) current-source density (CSD)
  - local field potentials (LFP)



hybridLFPy - Application with E-I network

Model configuration

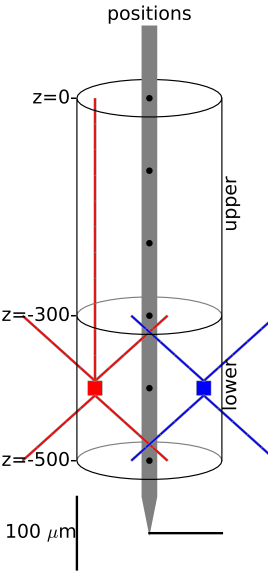
- "Layers":
  - upper $z \in [-300, 0]$ μm
  - lower $z \in [-500, -300]$ μm
- Measurements:
  - 6-channel laminar "electrode"
  - 100 μm between contacts
  - (laminar) current-source density (CSD)
  - local field potentials (LFP)
- Random cell positions
  - $z \in [-450, -350]$ μm
  - $R < 100$ μm



hybridLFPy - Application with E-I network

Model configuration

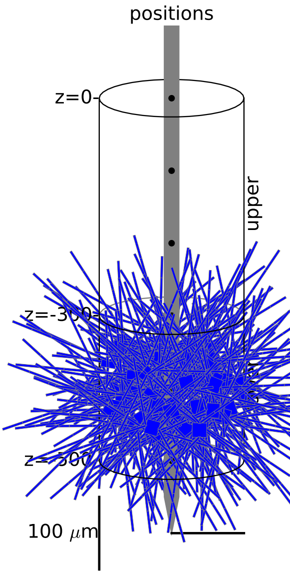
- Simplified morphologies:
  - "EX" - "pyramidal neuron"
  - "IN" - "interneuron"
  - passive cable models
  - membrane time constant of LIF neurons
  - spatially discretized into compartments



hybridLFPy - Application with E-I network

Model configuration

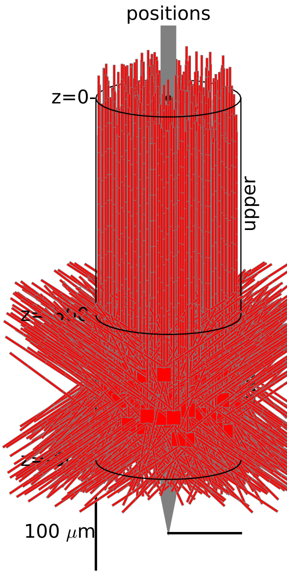
- Simplified morphologies:
  - “EX” - “pyramidal neuron”
  - “IN” - “interneuron”
  - passive cable models
  - membrane time constant of LIF neurons
  - spatially discretized into compartments
- IN - Random rotation around x, y, z-axis



hybridLFPy - Application with E-I network

Model configuration

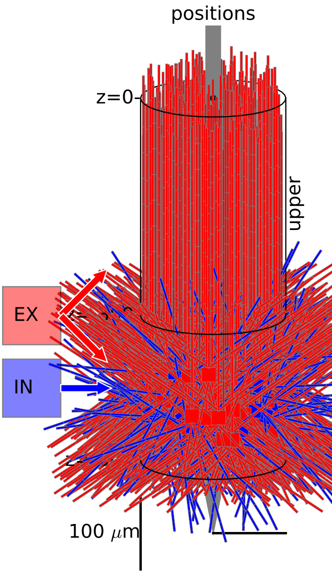
- Simplified morphologies:
  - “EX” - “pyramidal neuron”
  - “IN” - “interneuron”
  - passive cable models
  - membrane time constant of LIF neurons
  - spatially discretized into compartments
- IN - Random rotation around x, y, z-axis
- EX - Vertically aligned apical stick random rotation around z-axis



hybridLFPy - Application with E-I network

Model configuration

- Connectivity:
  - Mean in-degree from the network
  - EX→EX: 50/50% in upper/lower layer
  - EX→IN: 100% in lower layer
  - IN→EX: 100% in lower layer
  - IN→IN: 100% in lower layer
  - Only IN inputs on soma
  - Within layers - conn.-prob. normalized by surface area
- Synapse model
  - inherited from network
  - `NEURON` `NMODL` language `examples/alphaisyn.mod`



hybridLFPy - Application with E-I network

example_brunel.py initialization:

```
#import necessary classes and functions
...
from hybridLFPy import PostProcess, Population, CachedNetwork
from hybridLFPy import setup_file_dest
from NeuroTools.parameters import ParameterSet
from mpi4py import MPI

#MPI Initialization
COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()

#Parameters defined in pynest example script,
#brunel_alpha_nest.py, adapted from NEST v2.4.1 release:
import brunel_alpha_nest as BN
#note: will not execute model
```

hybridLFPy - Application with E-I network

File hierarchy:

- simulation_output_example_brunel/
  - simscripts/
  - cells/
  - populations/
  - spiking_output_path/
  - figures/

hybridLFPy - Application with E-I network

example_brunel.py file hierarchy:

```
PS = ParameterSet(dict()) #initialize
savefolder = 'simulation_output_example_brunel'
PS.update(
    #Main destination destination
    savefolder = savefolder,
    #copy of simulation files
    sim_scripts_path = os.path.join(savefolder, 'sim_scripts'),
    #single-cell output
    cells_path = os.path.join(savefolder, 'cells'),
    #destination compound signals
    populations_path = os.path.join(savefolder, 'populations'),
    #spike output from the network model
    spike_output_path = BN.spike_output_path,
    #figure destination
    figures_path = os.path.join(savefolder, 'figures')
))
#set up file destination, clear old results
setup_file_dest(PS, clearDestination=True)
```

hybridLFPy - Application with E-I network

example_brunel.py parameter setup:

```
#population (and cell type) specific parameters
PS.update(dict(
    #population names
    X = ["EX", "IN"],
    #population-specific LFPy.Cell parameters
    cellParams = dict(
        #excitory cells
        EX = dict(
            morphology = 'morphologies/ex.hoc',
            v_init = BN.neuron_params['E_L'],
            rm = BN.neuron_params['tau_m']*1E3/1.,
            cm = 1.0, Ra = 150,
            e_pas = BN.neuron_params['E_L'],
            timeres_NEURON = BN.dt,
            timeres_python = BN.dt,
            tstopms = BN.simtime,),
        #inhibitory cells
        IN = dict(morphology =
'morphologies/in.hoc', ...))
```

hybridLFPy - Application with E-I network

example_brunel.py parameter setup:

```
#population (and cell type) specific parameters
PS.update(dict(
    #cylindrical model populations
    populationParams = dict(
        EX = dict(
            number = BN.NE,
            radius = 100,
            z_min = -450,
            z_max = -350,
            min_cell_interdist = 1.,),
        IN = dict(number = BN.NI, ...),
    ),
    #set the boundaries between the
    #"upper" and "lower" layer:
    layerBoundaries = [[0.,   -300],
                       [-300, -500]]),
```

hybridLFPy - Application with E-I network

example_brunel.py parameter setup:

```
#set the geometry of the virtual recording device
PS.update(dict(
    electrodeParams = dict(
            #contact locations:
            x = [0]*6,
            y = [0]*6,
            z = [x*-100. for x in range(6)],
            #extracellular conductivity:
            sigma = 0.3,
            #contact surface normals, radius, n-point averaging
            N = [[1, 0, 0]]*6,
            r = 5,
            n = 20,
            seedvalue = None,
            #dendrite line sources, soma as sphere source (Linden2014)
            method = 'som_as_point',
            #no somas within the constraints of the "electrode shank":
            r_z = [[-1E199, -600, -550, 1E99],[0, 0, 10, 10]],)))
```

hybridLFPy - Application with E-I network

example_brunel.py parameter setup:

```
#layer- and population-specific
#connection parameters
PS.update(dict(
    #number of connections per layer per cell
    #from each presynaptic population
    k_yXL = dict(
        EX = [[int(0.5*BN.CE), 0],
              [int(0.5*BN.CE), BN.CI]],
        IN = [[0,      0],
              [BN.CE, BN.CI]],),

    #Connection weights (current amplitudes)
    J_yX = dict(
        EX = [BN.J_ex*1E-3, BN.J_in*1E-3],
        IN = [BN.J_ex*1E-3, BN.J_in*1E-3],),
```

hybridLFPy - Application with E-I network

example_brunel.py parameter setup:

```
#set up synapse parameters as derived from the network
PS.update(dict(
    synParams = dict(
        EX = dict(
            section = ['apic', 'dend'],
            syntype = 'AlphaISyn'),
        IN = dict(section = ['dend', 'soma'],  ...)),
    #set up table of synapse time constants
    tau_yX = dict(
        EX = [BN.tauSyn, BN.tauSyn],
        IN = [BN.tauSyn, BN.tauSyn]),
    #fixed delays of network
    synDelayLoc = dict(
        EX = [BN.delay, BN.delay],
        IN = [BN.delay, BN.delay]),
    #no distribution of delays
    synDelayScale = dict(
        EX = [None, None],
        IN = [None, None]),
```

hybridLFPy - Application with E-I network

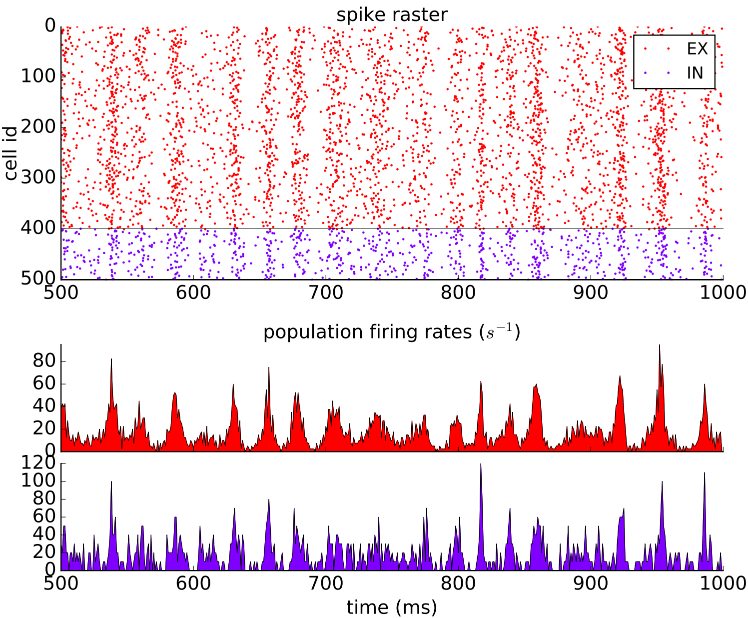
example_brunel.py network spikes:

```
#execute network simulation
BN.simulate()

#wait for the network simulation to finish, resync MPI threads
COMM.Barrier()

#Create an object representation containing the spiking activity of
#the network simulation output that uses sqlite3. Again, kwargs are
#derived from the brunel network instance.
networkSim = CachedNetwork(
    simtime = BN.simtime,
    dt = BN.dt,
    spike_output_path = BN.spike_output_path,
    label = BN.label,
    ext = 'gdf',
    GIDs = {'EX' : [1, BN.NE],
            'IN' : [BN.NE+1, BN.NI]},
)
```

hybridLFPy - Application with E-I network

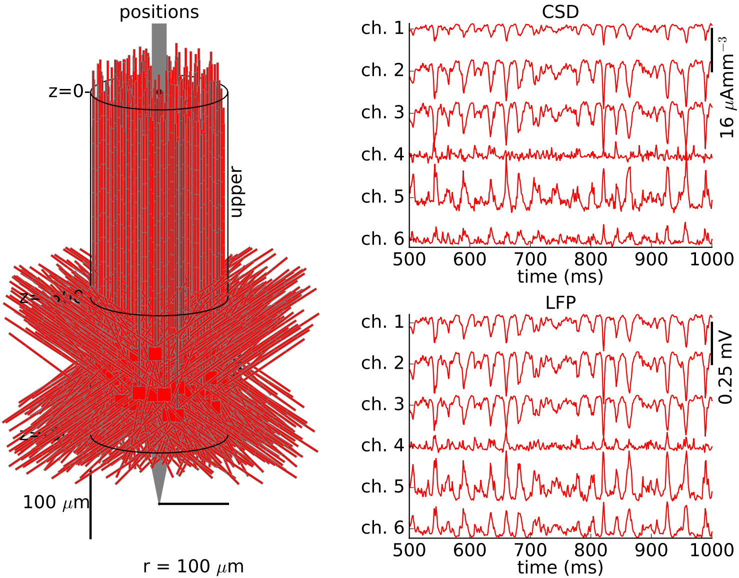


hybridLFPy - Application with E-I network

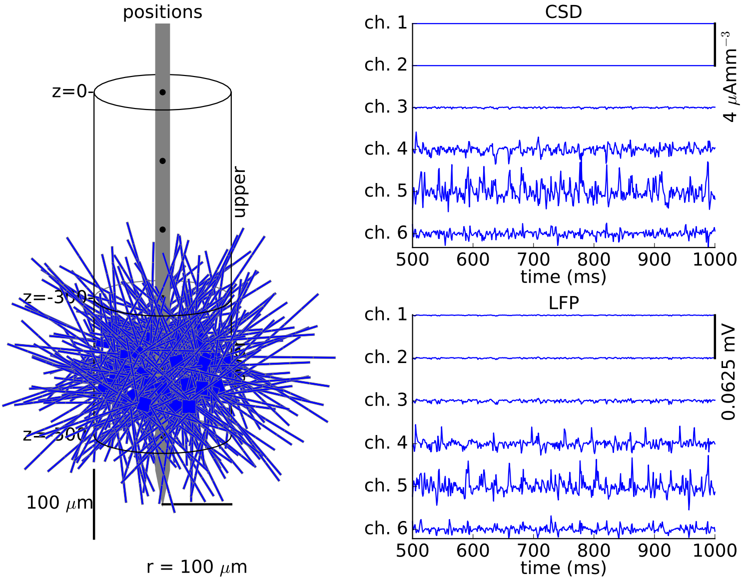
example_brunel.py running single-cell simulations:

```
for i, Y in enumerate(PS.X):
    #create population:
    pop = Population(
            cellParams = PS.cellParams[Y],
            rand_rot_axis = PS.rand_rot_axis[Y],
            simulationParams = PS.simulationParams,
            populationParams = PS.populationParams[Y],
            layerBoundaries = PS.layerBoundaries,
            electrodeParams = PS.electrodeParams,
            ...
            networkSim = networkSim,
            k_yXL = PS.k_yXL[Y],
            synParams = PS.synParams[Y],
            synDelayLoc = PS.synDelayLoc[Y],
            synDelayScale = PS.synDelayScale[Y],
            J_yX = PS.J_yX[Y], tau_yX = PS.tau_yX[Y])
    # run simulation, process single-cell data
    pop.run()
    pop.collect_data()
```

hybridLFPy - Application with E-I network



hybridLFPy - Application with E-I network

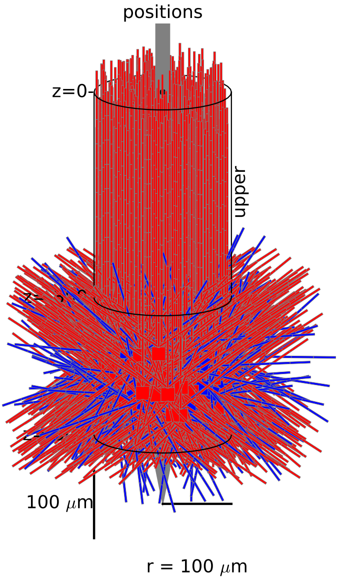


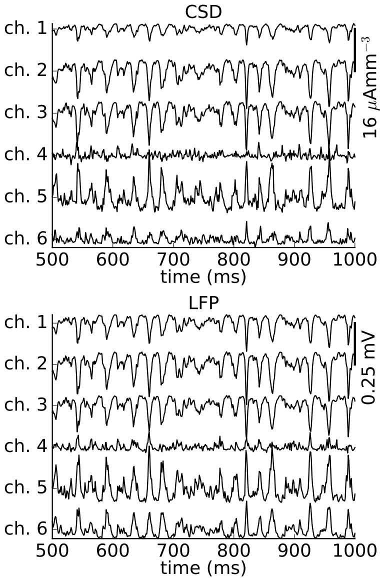
hybridLFPy - Application with E-I network

example_brunel.py creating compound signals:

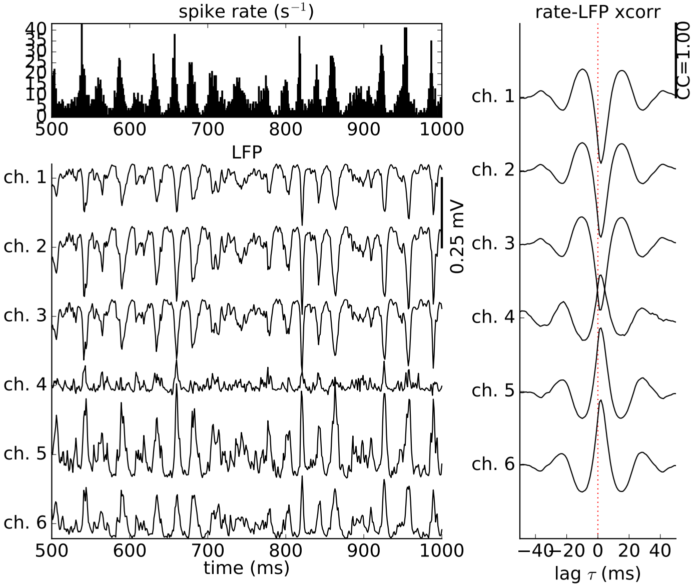
```
#Postprocessing of population output,
#(superposition of population LFPs, CSDs)
postproc = PostProcess(y = PS.X,
                       dt_output = PS.dt_output,
                       savefolder = PS.savefolder,
                       mapping_Yy = PS.mapping_Yy,
                       )
#run procedure
postproc.run()
```

hybridLFPy - Application with E-I network



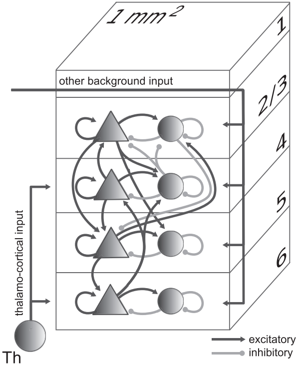


hybridLFPy - Application with E-I network



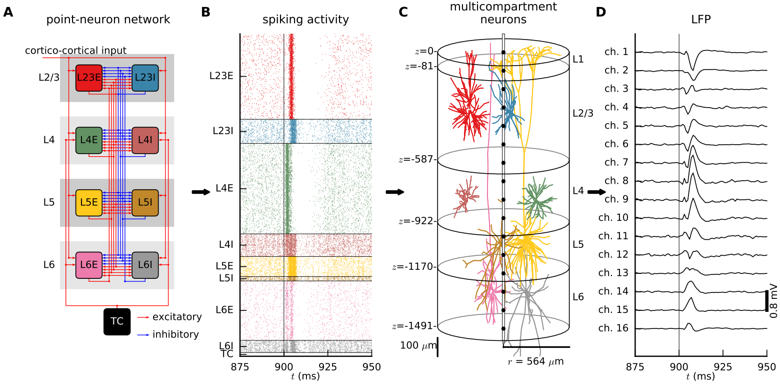
hybridLFPy - Application with microcircuit model

- microcircuit model:
  - local circuitry under 1mm^2 (cat VC)
  - 80k LIF point neurons
  - 300M current-based synapses
  - 4 layers
  - 2 populations (E,I) per layer
  - equal dynamics of E-I neurons
  - layer- and type-specific random connectivity



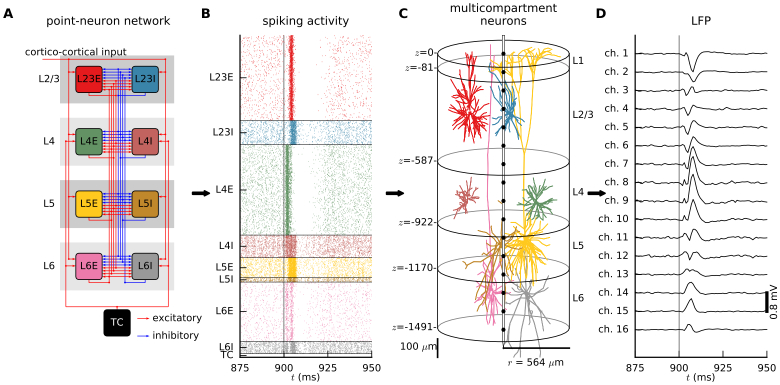
Potjans&Diesmann, *Cereb Cortex* (2014)

hybridLFPy - Application with microcircuit model



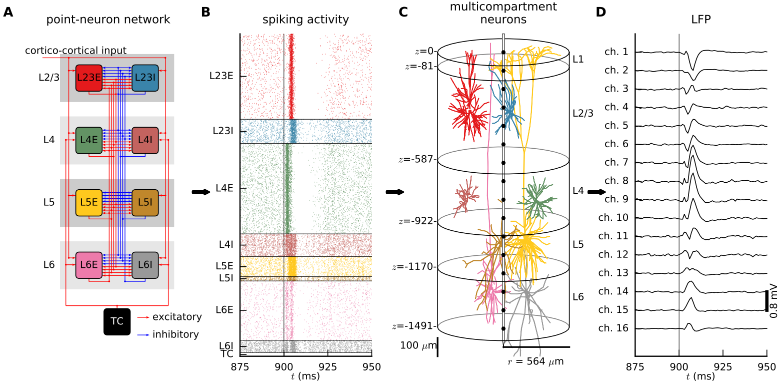
- point-neuron network, Potjans&Diesmann, *Cereb Cortex* (2014) (http://www.opensourcebrain.org)

hybridLFPy - Application with microcircuit model



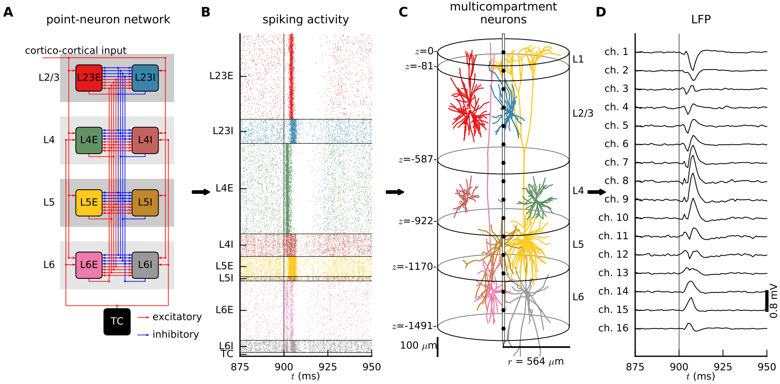
- point-neuron network, Potjans&Diesmann, *Cereb Cortex* (2014) (http://www.opensourcebrain.org)
  - network spikes → synaptic activation times

hybridLFPy - Application with microcircuit model



- point-neuron network, Potjans&Diesmann, *Cereb Cortex* (2014) (http://www.opensourcebrain.org)
  - network spikes → synaptic activation times
  - cell-type and layer specific connectivity

hybridLFPy - Application with microcircuit model



- point-neuron network, Potjans&Diesmann, *Cereb Cortex* (2014) (http://www.opensourcebrain.org)
  - network spikes → synaptic activation times
  - cell-type and layer specific connectivity
  - multi-compartment neurons: “antennas” for LFP generation

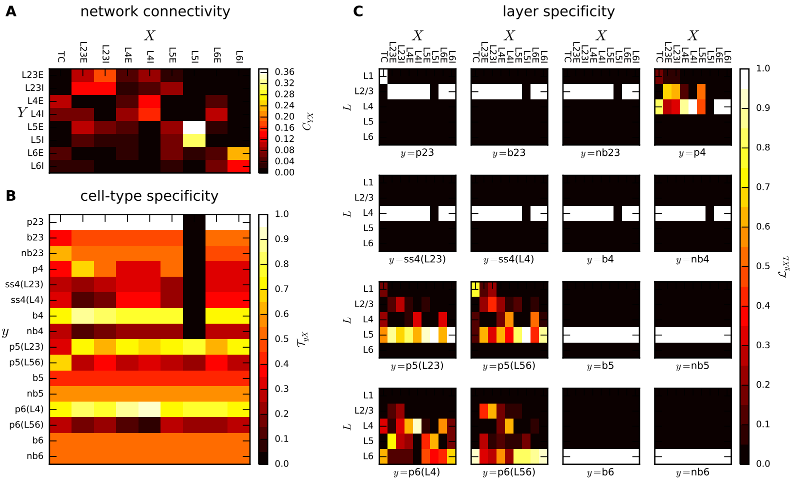
hybridLFPy - Application with microcircuit model

presynaptic neurons

postsynaptic neurons		percent of cells	number of synapses	nb1	p2/3	b2/3	nb2/3	ss4(L4)	ss4(L2/3)	p4	b4	nb4	p5(L2/3)	p5(L5/6)	b5	nb5	p6(L4)	p6(L5/6)	b6	nb6	corticocortical	TCs	TCn	TIs	TIn	TRN
nb1		1.5	8890	10.1	6.3	0.6	1.1	.	.	0.1	.	.	0.1	.	.	.	.	.	.	.	77.6	.	4.1	.	.	.
p2/3	L2/3	26	5800	.	59.9	9.1	4.4	0.6	6.9	7.7	.	0.8	7.4	.	.	.	2.3	.	.	0.8	.	.	.	.	.	.
	L1		1306	10.2	6.3	0.1	1.1	.	.	0.1	.	.	0.1	.	.	.	.	.	.	.	78	.	4.1	.	.	.
b2/3		3.1	3854	1.3	51.6	10.6	3.4	0.5	5.8	6.6	.	0.8	6.3	.	.	.	2.1	.	.	0.7	9.8	.	0.5	.	.	.
nb2/3		4.2	3307	1.7	48.6	11.4	3.3	0.5	5.5	6.2	.	0.8	5.9	.	.	.	1.8	.	.	0.6	13	.	0.7	.	.	.
ss4(L4)		9.2	5792	.	2.7	0.2	0.6	11.9	3.7	4.1	7.1	2	0.8	0.1	.	.	32.7	.	.	5.8	25.3	1.7	1.3	.	.	.
ss4(L2/3)		9.2	4989	.	5.6	0.4	0.8	11.3	3.8	4.3	7.2	2.1	1.1	0.1	.	.	31.1	.	.	5.5	23.9	1.7	1.3	.	.	.
p4	L4	9.2	5031	.	4.3	0.2	0.6	11.5	3.6	4.2	7.2	2.1	1.2	0.1	.	.	31.4	0.1	.	5.9	24.5	1.7	1.3	.	.	.
	L2/3		866	.	63.1	5.1	4.1	0.6	7.2	8.1	.	0.6	7.8	.	.	.	2.5	.	.	0.8	.	.	.	.	.	.
	L1		806	10.2	6.3	0.1	1.1	.	.	0.1	.	.	0.1	.	.	.	.	.	.	.	78	.	4.1	.	.	.
b4		5.4	3230	.	5.8	0.5	0.8	11	3.8	4.2	8.4	2.4	1.1	.	.	.	30.3	.	.	5.4	23.3	1.6	1.2	.	.	.
nb4		1.5	3688	.	2.7	0.2	0.6	11.7	3.6	4	8.2	2.3	0.8	0.1	.	.	32.2	.	.	5.7	24.9	1.7	1.3	.	.	.
p5(L2/3)	L5	4.8	4316	.	45.9	1.8	0.3	3.3	2	7.5	.	0.9	11.7	1	0.8	1.1	2.3	2.1	.	11.5	7.2	0.1	0.4	.	.	.
	L4		283	.	2.8	0.1	0.7	12.2	3.8	4.2	5.2	1.5	0.8	0.1	.	.	33.7	.	.	5.9	26	1.8	1.4	.	.	.
	L2/3		412	.	63.1	5.1	4.1	0.6	7.2	8.1	.	0.6	7.8	.	.	.	2.5	.	.	0.8	.	.	.	.	.	.
	L1		185	10.2	6.3	0.1	1.1	.	.	0.1	.	.	0.1	.	.	.	.	.	.	.	78	.	4.1	.	.	.
p5(L5/6)	L5	1.3	5101	.	44.3	1.7	0.2	3.2	2	7.3	.	0.8	11.3	1.2	0.8	1.1	2.3	2.5	0.3	11.3	9.2	0.2	0.5	.	.	.
	L4		949	.	2.8	0.1	0.7	12.2	3.8	4.2	5.2	1.5	0.8	0.1	.	.	33.7	.	.	5.9	26	1.8	1.4	.	.	.
	L2/3		1367	.	63.1	5.1	4.1	0.6	7.2	8.1	.	0.6	7.8	.	.	.	2.5	.	.	0.8	.	.	.	.	.	.
	L1		5658	10.2	6.3	0.1	1.1	.	.	0.1	.	.	0.1	.	.	.	.	.	.	.	78	.	4.1	.	.	.
b5		0.6	2981	.	45.5	2.3	0.2	3.3	2	7.5	.	1.1	11.6	1	0.9	1.3	2.3	2	.	11.4	7.2	0.1	0.4	.	.	.
nb5		0.8	2981	.	45.5	2.3	0.2	3.3	2	7.5	.	1.1	11.6	1	0.9	1.3	2.3	2	.	11.4	7.2	0.1	0.4	.	.	.
p6(L4)	L6	13.6	3261	.	2.5	0.1	0.1	0.7	0.9	1.3	.	0.1	0.1	4.9	.	0.3	1.2	13.2	7.7	7.7	55.7	0.6	2.9	.	.	.
	L5		1066	.	46.8	0.8	0.3	3.4	2.1	7.7	.	0.6	11.9	1	0.6	0.8	2.3	2.1	.	11.7	7.4	0.1	0.4	.	.	.
	L4		1915	.	2.8	0.1	0.7	12.2	3.8	4.2	5.2	1.5	0.8	0.1	.	.	33.7	.	.	5.9	26	1.8	1.4	.	.	.
	L2/3		121	.	63.1	5.1	4.1	0.6	7.2	8.1	.	0.6	7.8	.	.	.	2.5	.	.	0.8	.	.	.	.	.	.
p6(L5/6)	L6	4.5	5573	.	2.5	0.1	0.1	0.7	0.9	1.3	.	0.1	0.1	4.9	.	0.3	1.2	13.2	7.8	7.8	55.7	0.6	2.9	.	.	.
	L5		257	.	46.8	0.8	0.3	3.4	2.1	7.7	.	0.6	11.9	1	0.6	0.8	2.3	2.1	.	11.7	7.4	0.1	0.4	.	.	.
	L4		243	.	2.8	0.1	0.7	12.2	3.8	4.2	5.2	1.5	0.8	0.1	.	.	33.7	.	.	5.9	26	1.8	1.4	.	.	.
	L2/3		286	.	63.1	5.1	4.1	0.6	7.2	8.1	.	0.6	7.8	.	.	.	2.5	.	.	0.8	.	.	.	.	.	.
	L1		62	10.2	6.3	0.1	1.1	.	.	0.1	.	.	0.1	.	.	.	.	.	.	.	78	.	4.1	.	.	.
b6		2	3220	.	2.5	0.1	0.1	0.7	0.9	1.3	.	0.1	0.1	4.9	.	0.4	1.2	13.2	7.7	7.7	55.7	0.6	2.9	.	.	.
nb6		2	3220	.	2.5	0.1	0.1	0.7	0.9	1.3	.	0.1	0.1	4.9	.	0.4	1.2	13.2	7.7	7.7	55.7	0.6	2.9	.	.	.
				brainstem		sensory																				
TCs		0.5	4000	31	.	7.1	.	.	.	.	.	.	.	.	.	.	23	8	.	.	.	.	.	5	.	25.9
TCn		0.5	4000	31	.	7.1	.	.	.	.	.	.	14	3.8	.	.	.	13.2	.	.	.	.	.	.	5	25.9
TIs		0.1	3000	13.5	.	48.7	.	.	.	.	.	.	.	.	.	.	9.8	3.3	.	.	.	0.4	.	24.4	.	.
TIn		0.1	3000	13.4	.	48.7	.	.	.	.	.	.	5.8	1.6	.	.	.	5.4	.	.	.	.	0.6	.	24.4	.
TRN		0.5	4000	40	.	.	.	.	.	.	.	.	.	.	.	.	30	.	.	.	.	10	10	.	.	10

Cat VC connectivity of Binzegger et al. *J Neurosci* (2004)

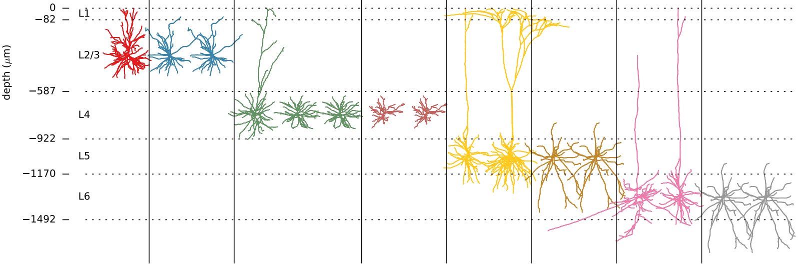
hybridLFPy - Application with microcircuit model



Network connectivity and layer specificity of connections

hybridLFPy - Application with microcircuit model

Population Y:	L23E	L23I		L4E			L4I		L5E		L5I		L6E		L6I	
Pop. size N_Y:	20683	5834		21915			5479		4850		1065		14395		2948	
Cell type y:	p23	b23	nb23	p4	ss4(L23)	ss4(L4)	b4	nb4	p5(L23)	p5(L56)	b5	nb5	p6(L4)	p6(L56)	b6	nb6
Morphology M_y:	p23	i23	i23	p4	ss4	ss4	i4	i4	p5v1	p5v2	i5	i5	p6	p5v1	i5	i5
Segments n_{comp}:	689	324	324	603	339	339	230	230	423	1078	283	283	456	447	283	283
Occurrence F_y:	26.8%	3.2%	4.3%	9.5%	9.5%	9.5%	5.6%	1.5%	4.9%	1.3%	0.6%	0.8%	14.0%	4.6%	1.9%	1.9%
Rel. Occurr. F_{yY}:	100.0%	42.5%	57.5%	33.3%	33.3%	33.3%	78.2%	21.7%	78.7%	21.3%	42.8%	57.1%	75.1%	24.9%	50.0%	50.0%
Cell count N_y:	20683	2477	3356	7305	7305	7305	4287	1191	3816	1033	456	608	10816	3578	1474	1474

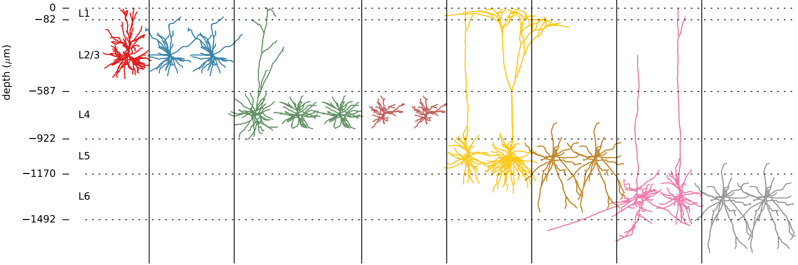


Morphologies and layer boundaries: [Stepanyants et al. *Cereb Cortex* (2008)]

- reconstructions from cat (visual/somatosensory cortices)
- limited data availability: reuse files across cell types

hybridLFPy - Application with microcircuit model

Population Y:	L23E	L23I		L4E			L4I		L5E		L5I		L6E		L6I	
Pop. size N_Y:	20683	5834		21915			5479		4850		1065		14395		2948	
Cell type y:	p23	b23	nb23	p4	ss4(L23)	ss4(L4)	b4	nb4	p5(L23)	p5(L56)	b5	nb5	p6(L4)	p6(L56)	b6	nb6
Morphology M_y:	p23	i23	i23	p4	ss4	ss4	i4	i4	p5v1	p5v2	i5	i5	p6	p5v1	i5	i5
Segments n_{comp}:	689	324	324	603	339	339	230	230	423	1078	283	283	456	447	283	283
Occurrence F_y:	26.8%	3.2%	4.3%	9.5%	9.5%	9.5%	5.6%	1.5%	4.9%	1.3%	0.6%	0.8%	14.0%	4.6%	1.9%	1.9%
Rel. Occurr. F_{yY}:	100.0%	42.5%	57.5%	33.3%	33.3%	33.3%	78.2%	21.7%	78.7%	21.3%	42.8%	57.1%	75.1%	24.9%	50.0%	50.0%
Cell count N_y:	20683	2477	3356	7305	7305	7305	4287	1191	3816	1033	456	608	10816	3578	1474	1474



Extrapolation from anatomical connectivity data:

- cell-type specific connectivity: 16 cell types
- layer specificity of connections

[Binzegger et al. (2004)]

hybridLFPy - Application with microcircuit model

Microcircuit model example

- Main example scripts:

  ```
  example_microcircuit.py
  example_microcircuit_params.py
  binzegger_connectivity_table.json
  morphologies/ballnsticks/*.hoc
  expsyni.mod
  ```

hybridLFPy - Application with microcircuit model

Microcircuit model example

- Main example scripts:

```
example_microcircuit.py
example_microcircuit_params.py
binzegger_connectivity_table.json
morphologies/ballnsticks/*.hoc
expsyni.mod
```

- Execute model:

```
cd /PATH/TO/hybridLFPy/examples/
nrnivmodl
mpirun -np 128 python example_microcircuit.py
```

hybridLFPy - Application with microcircuit model

Microcircuit model example - example_microcircuit_params.py

- Defines and derives parameter values
- Process anatomical connectivity
- Map network connectivity onto LFP model
- Defined through parameter class objects

```
class general_params(object):
    '''class defining general model parameters'''

class point_neuron_network_params(general_params):
    '''class defining point-neuron network parameters'''

class multicompartment_params(point_neuron_network_params):
    '''class defining additional attributes needed by
    hybridLFPy.Population and hybridLFPy.DummyNetwork'''
```

hybridLFPy - Application with microcircuit model

Microcircuit model example - example_microcircuit.py

- Load parameter objects

```
networkParams = point_neuron_network_params()
params = multicompartment_params() # all parameters
```

- Executes network simulation

```
sli_run(parameters=networkParams, fname='microcircuit.sli')
merge_gdf(networkParams, ...)
networkSim = CachedNetwork(**params.networkSimParams)
```

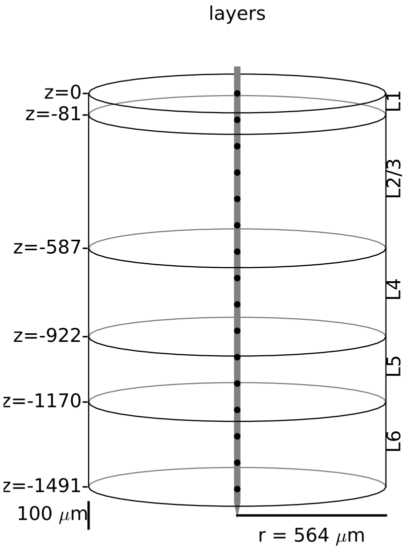
- Calculate extracellular potentials

```
for i, y in enumerate(params.y):
    pop = Population(cellParams=params.yCellParams[y], **args)
    pop.run()
    pop.collect_data()
```

hybridLFPy - Application with microcircuit model

Microcircuit model example

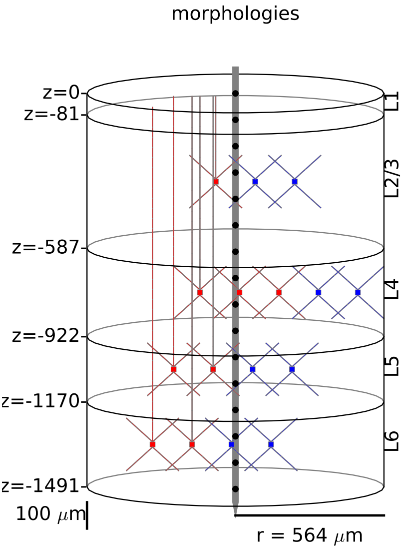
- $1mm^2$ cortical surface area
- Layer boundaries from Stepanyants et al. (2008)



hybridLFPy - Application with microcircuit model

Microcircuit model example

- $1mm^2$ cortical surface area
- Layer boundaries from Stepanyants et al. (2008)
- Simplified morphologies for each cell type

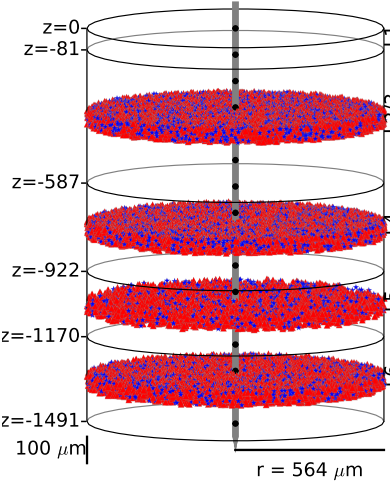


hybridLFPy - Application with microcircuit model

Microcircuit model example

- $1mm^2$ cortical surface area
- Layer boundaries from Stepanyants et al. (2008)
- Simplified morphologies for each cell type
- Random soma positions within layers

soma positions

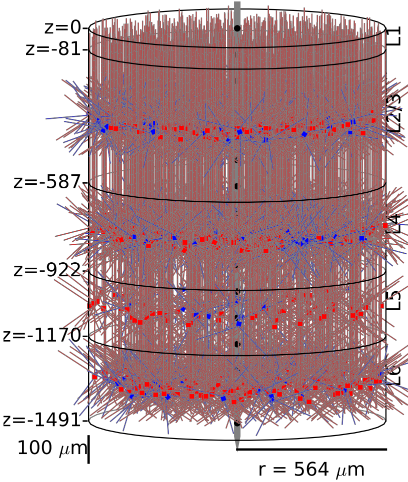


hybridLFPy - Application with microcircuit model

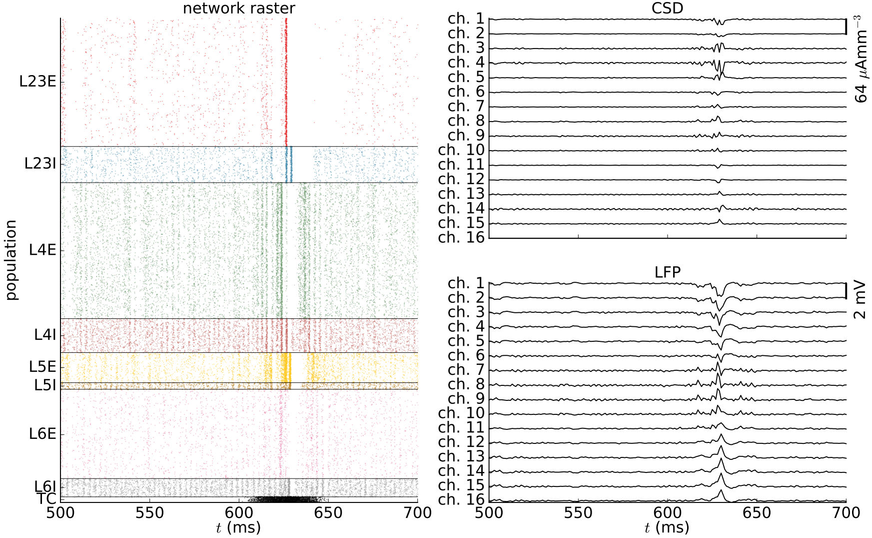
Microcircuit model example

- 1mm^2 cortical surface area
- Layer boundaries from Stepanyants et al. (2008)
- Simplified morphologies for each cell type
- Random soma positions within layers
- 78000 multicompartment neurons generate LFP

LFP generators



hybridLFPy - Application with microcircuit model



Summary & Outlook

- Forward modeling scheme:
  - Derived using standard electrostatic theory
  - Multicompartment neuron models
- **`LFPy`**; http://LFPy.github.io:
  - Extracellular potentials of single-neuron models
  - Anisotripic extracellular medium
  - Method of Images (MoI) for inhomogeneous media
  - Support parallel network implementations
- **`hybridLFPy`**; http://inm-6.github.com/hybridLFPy:
  - LFP predictions of point neuron network models
  - Main application: Potjans&Diesmann, *Cereb Cortex* (2014)
  - Multi-area model predictions
  - Network models with distance dependent connections
  - Verify simplified LFP prediction schemes

Acknowledgements

- Helmholtz Association: HASB and portfolio theme SMHB
- Jülich Aachen Research Alliance (JARA)
- EU grant 269921 (BrainScaleS)
- EU Grant 604102 (Human Brain Project, HBP)
- Research Council of Norway (NFR) through NevroNor, eNEURO, Notur, NN4661K.

Questions?

Questions?

If not - feel free to try out

- **iCSD** and **kCSD** tools
  - https://github.com/espenhgn/iCSD
  - https://github.com/INCF/pykCSD
  - (ElePhAnT ElectroPhysiology Analysis Toolkit https://github.com/NeuralEnsemble/elephant, https://github.com/ccluri/elephant)
- **LFPy**
  - https://github.com/LFPy/LFPy
  - http://LFPy.github.io
- **hybridLFPy**
  - https://github.com/INM-6/hybridLFPy
  - http://inm-6.github.io/hybridLFPy