Primer on `LFPy2.0`

CNS 2018 Tutorial T4 part II

Espen Hagen – espen.hagen@fys.uio.no

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

Seattle, July 13, 2018

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?

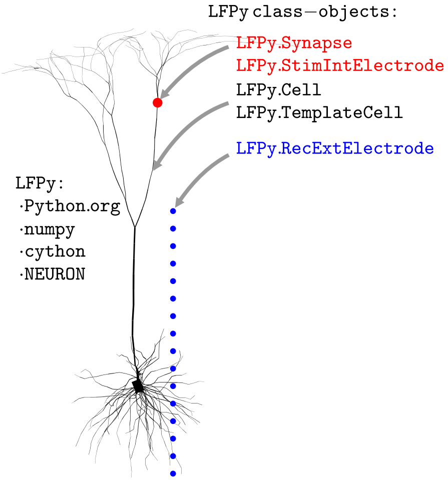
- **Improve understanding of experimental measurements:**
  - Action potential waveforms:
    - Gold et al. *J Neurophysiol* (2006)
    - Pettersen & Einevoll. *Biophys J* (2008)
    - Hagen et al. *J Neurosci Methods* (2015)
    - Ness et al. *Neuroinform* (2015)
  - Active ion channel component of LFP:
    - Schomburg et al. *J. Neurosci* (2012)
    - Reimann et al. *Neuron* (2013)
    - Ness et al. *J Physiol* (2016); *J Neurosci* (2018)
  - 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)
    - Hagen et al. *J Neurosci* (2017) (monosynaptic)

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, networks
  - extracellular potentials
  - ‘distal’ EEG and MEG signals
- Implemented in `Python`
- Uses `NEURON` under the hood
- Class objects represent:
  - cells, populations, networks
  - synapses
  - intracellular electrodes
  - extracellular electrodes
- Homepages w. documentation:
  http://LFPy.rtfd.io
  https://github.com/LFPy/LFPy



LFPy

Developers:

- Henrik Lindén, Espen Hagen, Szymon Łęski, Eivind S. Norheim, Klas H. Pettersen, Torbjørn V. Ness, Solveig Næss, Alessio Buccino, Svenn-Arne Dragly, Alex Stasik, Gaute T. Einevoll
- It's open source - anyone can contribute!

Homepages:

- https://LFPy.rtfd.io (documentation)
- https://github.com/LFPy/LFPy
  (sources, revision and issue tracking, pull requests)

LFPy
Local Field Potentials in Python

LFPy Homepage

(Looking for the old LFPy documentation? Follow link)

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

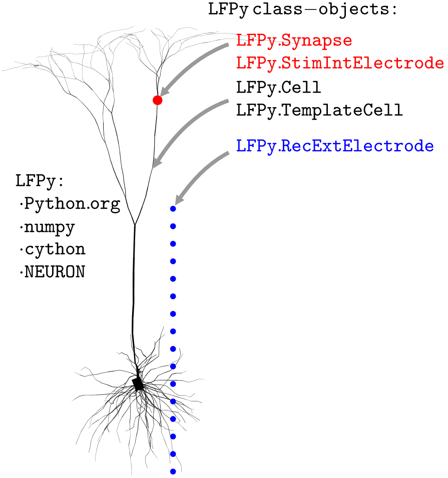
Active development of LFPy, as well as issue tracking and revision tracking, relies on GitHub.com and git (git-scm.com). Clone LFPy on GitHub.com: `git clone https://github.com/LFPy/LFPy.git`

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 or NeuroML2 already, it is likely that it can be adapted to work with LFPy.

LFPy - Introduction

Why Python?

- Open source
- Easy, flexible coding
- Plethora of available packages for visualizations and analysis
- http://pypi.python.org: $\mathcal{O}(140000)$ projects
- Interfacing other programming languages and software
  - C, C++, Fortran, ...
  - NEURON, NEST, Brian, PyNN, Nengo, ...



LFPy - Introduction

Python class objects

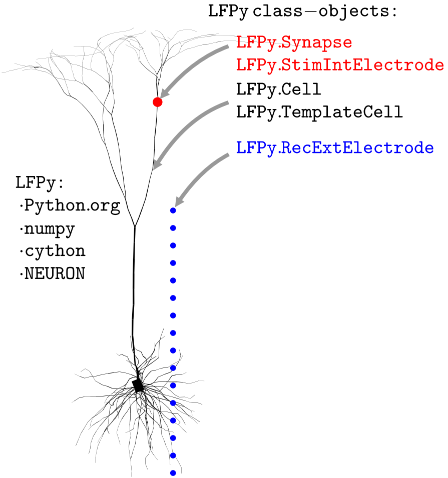
- Object Oriented Programming (OOP)
- arbitrary amounts and kinds of data
- contains methods and attributes
- created runtime, modifiable

```
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:

- Essential:
  - `neuron`
  - setuptools>=23.1.0
  - numpy>=1.8
  - scipy>=0.14
  - Cython>=0.20
  - h5py>=2.5
  - mpi4py>=1.2
  - matplotlib>=2.0
  - csa >= 0.1.8
- Soft:
  - `ipython`
  - `jupyter notebook`
  - `nose`



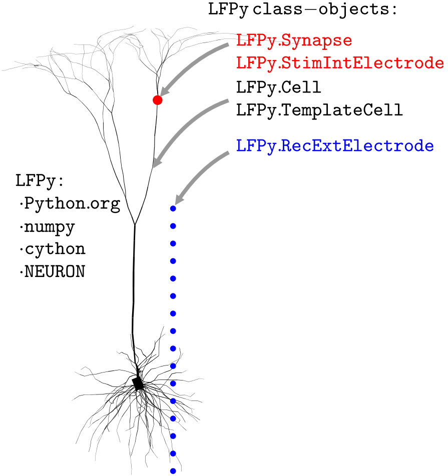
LFPy - Requirements

Python distributions:

- Python.org
- Anaconda/Miniconda
- Enthought Canopy
- Python(x,y)
- ...

LFPy platforms:

- *nix (Linux, Unix)
- macOS
- Windows



LFPy - Installation

Installation:

Easy (recommended) method:

- `pip install LFPy --user`

As super user:

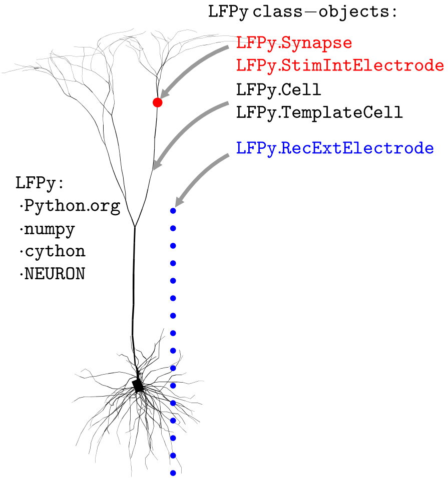
- `sudo pip install LFPy`

Upgrading previous install:

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

Getting rid of it (but why?):

- `pip uninstall LFPy`



LFPy - Installation

Install using `LFPy` source files:

- Tar.gz-archive:

  ```
  cd $HOME/Sources
  wget https://github.com/LFPy/LFPy/archive/v2.0.0tar.gz
  tar -xzvf v2.0.0tar.gz
  ```

  (unzipped in folder `LFPy-2.0.0`)

- Development version:

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

- Tagged versions:

  ```
  git tag -l # list tags
  git checkout v2.0.0
  ```

- `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
pip install -r requirements.txt --user
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 develop --user
```

LFPy - Installation

Test installation:

With Python:

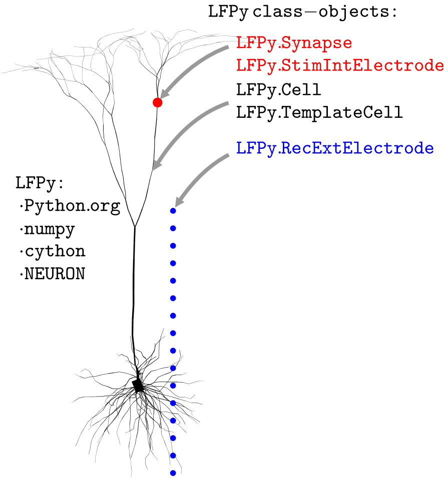
- `python -c "import LFPy"`

  `NEURON -- VERSION 7.5`
  `master (6b4c19f)...`

With NEURON:

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

  `NEURON -- VERSION 7.5`
  `master (6b4c19f)...`



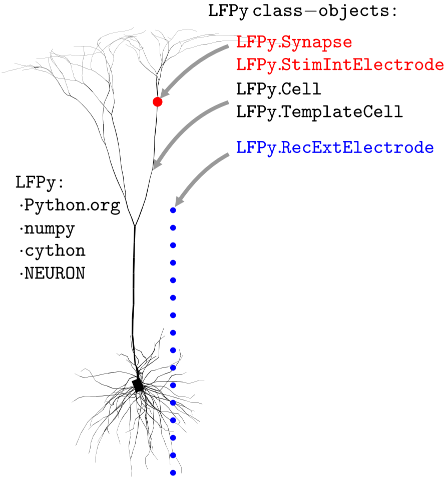
LFPy - Class overview

Main `LFPy` classes:

- `Cell, TemplateCell, NetworkCell`
- `Network, NetworkPopulation`
- `Synapse, StimIntElectrode`
- `RecExtElectrode, RecMEAElectrode`
- `OneSphereVolumeConductor, FourSphereVolumeConductor, InfiniteVolumeConductor`

Auxilliary classes and functions:

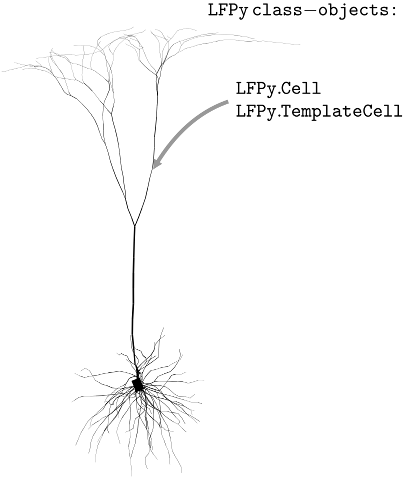
- `lfpcalc.calc_lfp.*`
- `inputgenerators.*`
- `tools.*`
- and more...



LFPy - Class overview

LFPy.Cell:

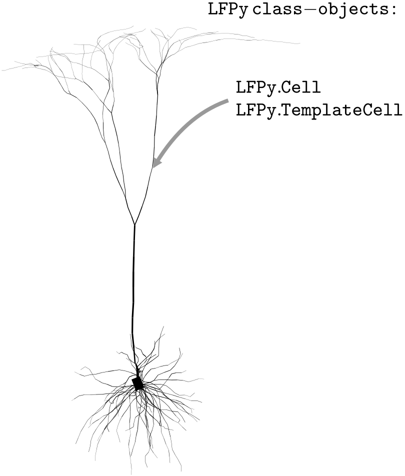
- Uses NEURON under the hood
- Sets neuron properties:
  - neuron geometry
  - membrane mechanisms ('pas', 'hh', ...)
  - number of compartments ('d_lambda' rule; Hines&Carnevale. *Neuroscientist* (2001))
  - Sets cell location and rotation
- Simulation control
  - duration
  - record variables



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`)

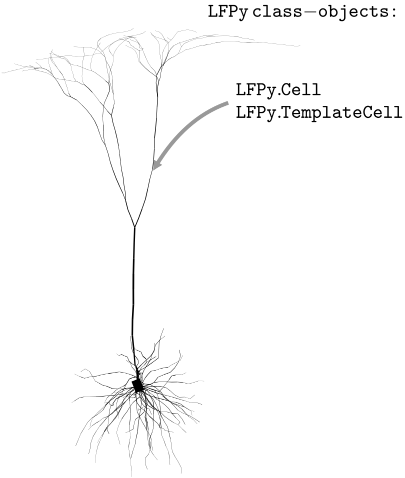


LFPy - Class overview

LFPy.Cell:

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

```
# Define cell parameters
cell_parameters = dict(
    morphology='j4a.hoc',
    cm = 1.,        # uF cm-2
    Ra = 150.,      # ohm cm
    v_init = -65., # mV
    passive = True,
    passive_parameters = dict(
      g_pas = 1./3E4, # S cm-2
      e_pas = -65.),  # mV
    tstop = 100.,   # ms
    )
```



LFPy - Class overview

LFPy.Cell:

- Create cell object:

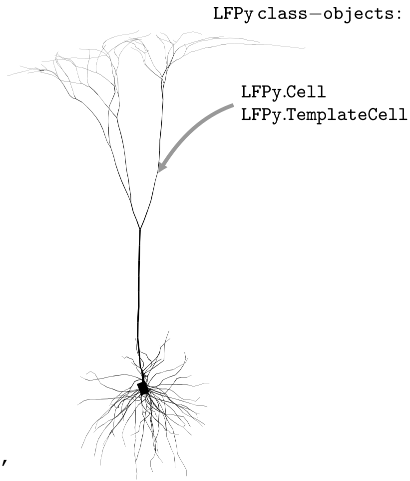
```
cell = LFPy.Cell(
        **cell_parameters)
```

- 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,
              rec_imem=True/False,
              rec_vmem=True/False)
plt.plot(cell.tvec, cell.somav)
```



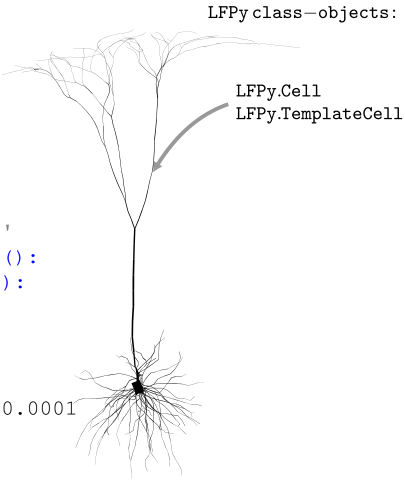
LFPy - Class overview

LFPy.Cell:

- Customizing the model:
  - passive mechanism disabled by default
  - custom_fun argument

```
def my_biophys():
    '''set custom parameters'''
    for sec in neuron.h.allsec():
        if "soma" in sec.name():
            sec.insert("hh")
        else:
            sec.insert("pas")
            for seg in sec:
                seg.pas.g_pas=0.0001
cell = LFPy.Cell(morphology,
    custom_fun=[my_biophys],
    **cell_parameters)
```

  - custom_code point to code files



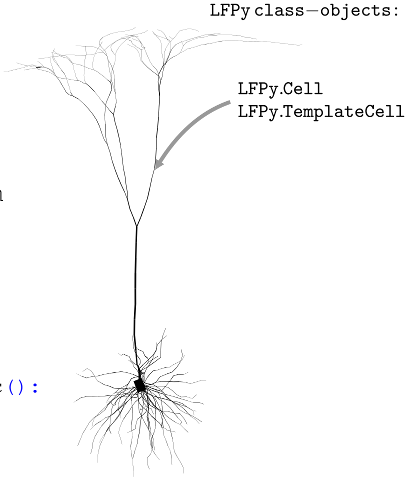
LFPy - Class overview

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

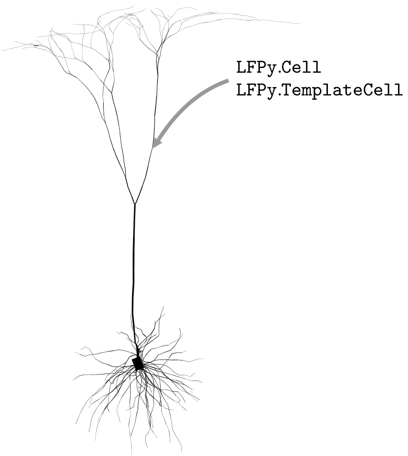
LFPy.Cell:

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

```
import LFPy
import neuron.h as nrn

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

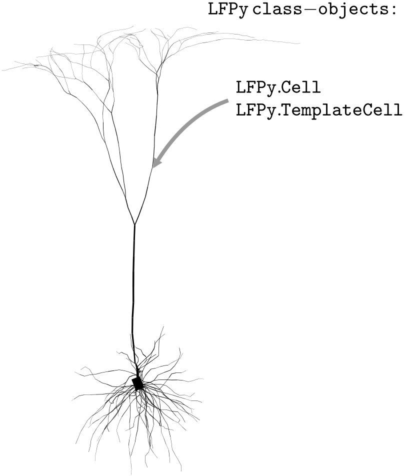
(only ‘HH’ conductances in soma)



LFPy - Class overview

LFPy.Cell: Sections created in Python:

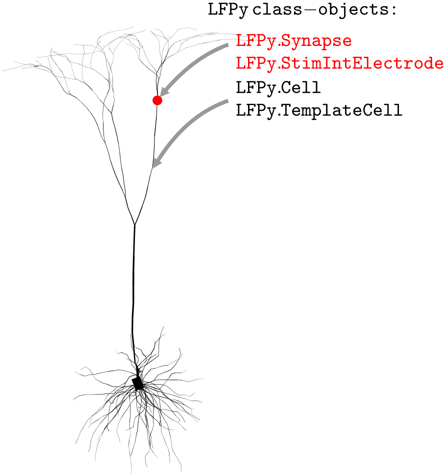
```
import neuron, LFPy
soma = neuron.h.Section(name='soma')
dend = neuron.h.Section(name='dend')
soma.L = 30
soma.diam = 30
dend.L = 300
dend.diam = 2
dend.connect(soma, 1, 0)
cell = LFPy.Cell(morphology=None,
                 delete_sections=False,
                 rm = 30000,
                 cm = 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

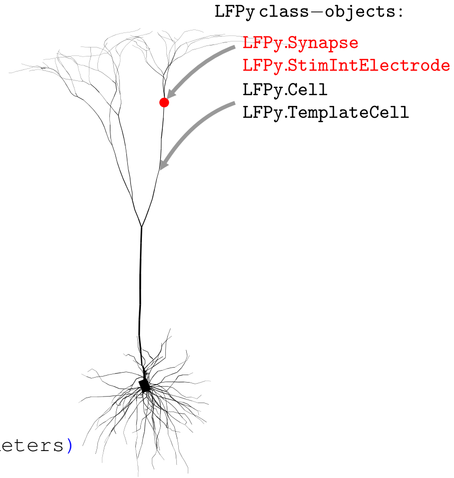
LFPy.Synapse:

- 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

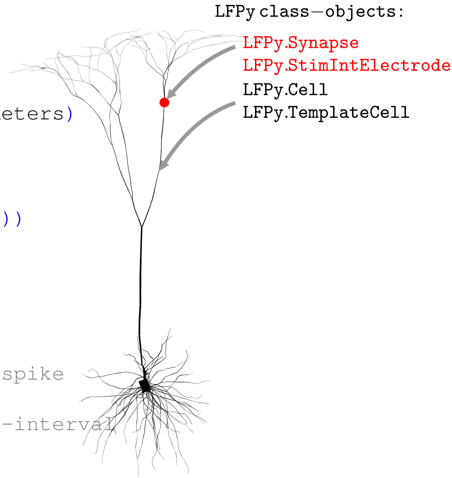
```
syn = LFPy.Synapse(cell,
                   **synapse_parameters)
```

- set offline activation time(s)

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

- generate activation time(s) on the fly

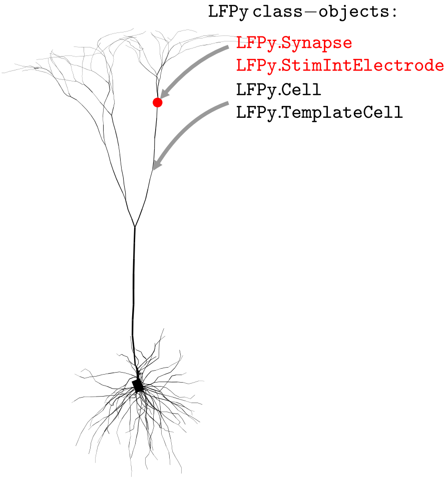
```
syn.set_spike_times_w_netstim(
    noise=1, # Poisson statistics
    start=0, # likely time of 1st spike
    number=1E3, # number of spikes
    interval=20, # mean interspike-interval
    )
```



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

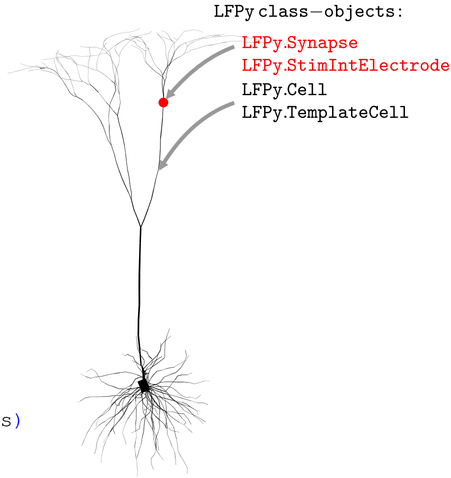
LFPy.StimIntElectrode:

- 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

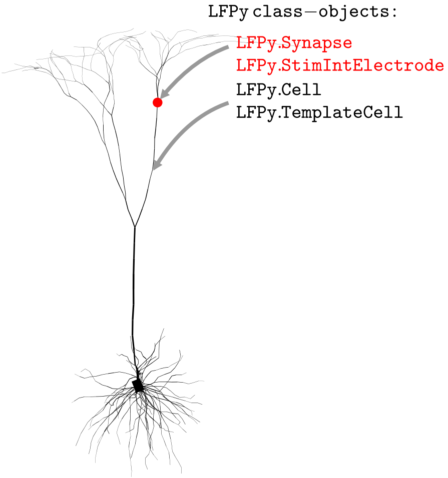
```
cell.simulate(rec_isyn=True,
              rec_istim=True)
```

- 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)
```

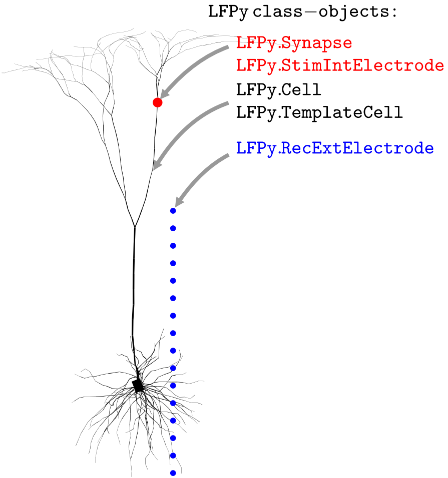


Questions?

LFPy - Class overview

LFPy.RecExtElectrode:

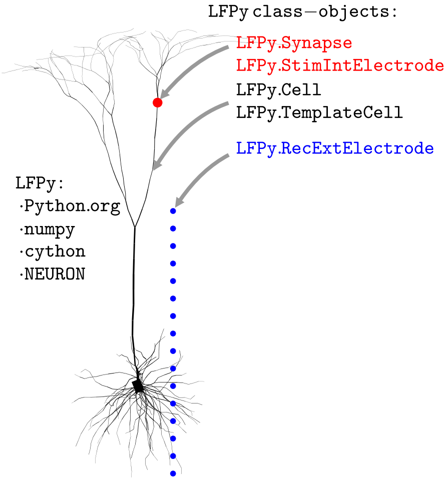
- Extracellular recording devices
- Main arguments:
  - `cell` objects (geometry, currents)
  - contact point coordinates `x,y,z`
  - extracellular conductivity `sigma`
  - `method` (pointsource/linesource)
- Optional:
  - radius and surface normal vectors of contacts
  - *n*-point surface area averaged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



LFPy - Class overview

LFPy.RecExtElectrode:

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



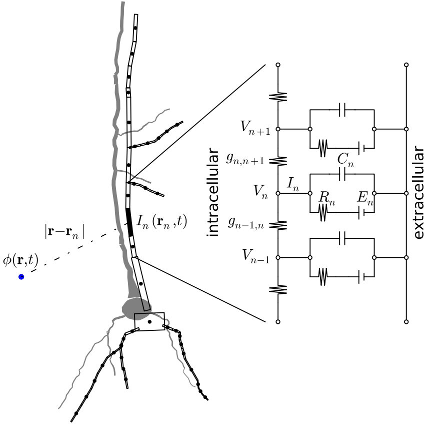
Forward modeling of extracellular potentials

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)

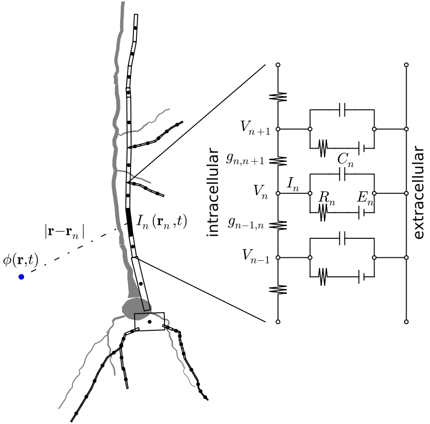
Forward modeling of extracellular potentials

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



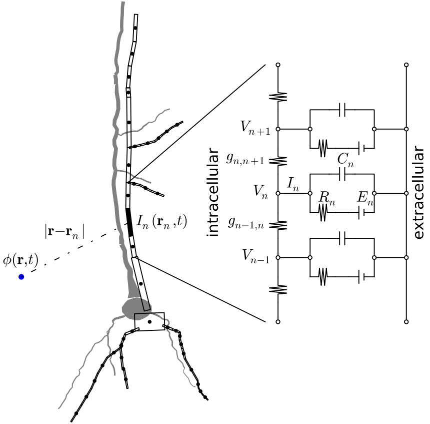
Lindén et al. (2014)

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:

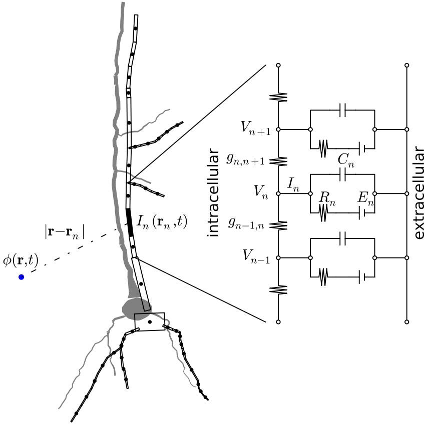
$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}$$

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

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

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

- $\mathbf{E}$ - electric field; ρ - charge density; ϵ_0 - free space permittivity; $\mathbf{B}$ - magnetic field; μ - permeability; $\mathbf{J}$ - 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

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

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

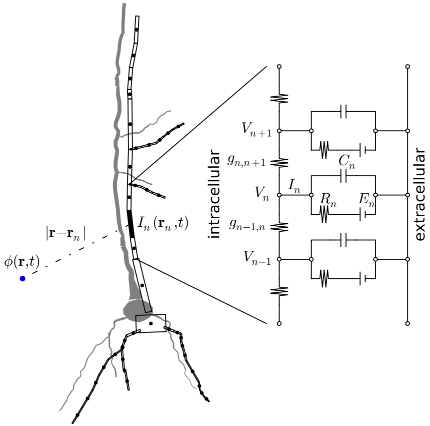
$$\nabla \times \mathbf{E} = 0$$

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

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

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

- σ - assumed scalar, real



Lindén et al. (2014)

Forward modeling of extracellular potentials

Biophysical background:

- Assuming a point current source

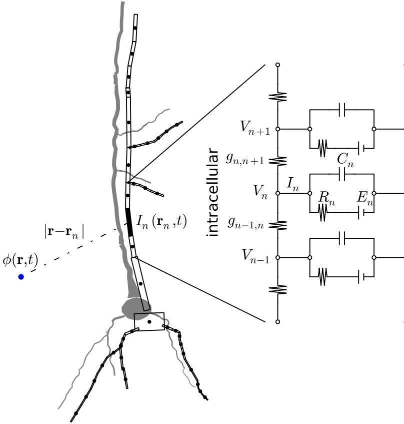
$$\mathbf{J} = \frac{I_0}{4\pi r^2}\hat{\mathbf{r}} \ , \ \hat{\mathbf{r}} : \text{ radial unit vector}$$

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

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

- 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:

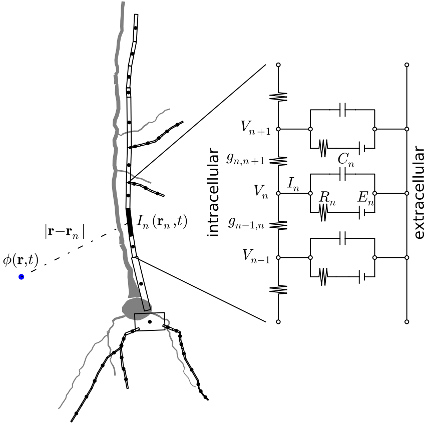
- Point current source

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

  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

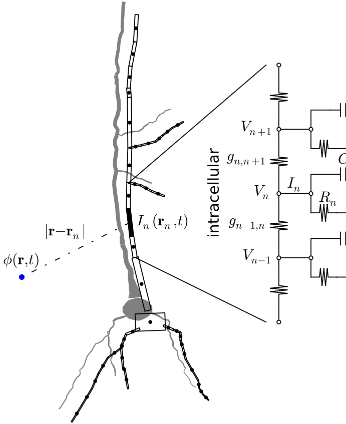
Biophysical background:

- 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

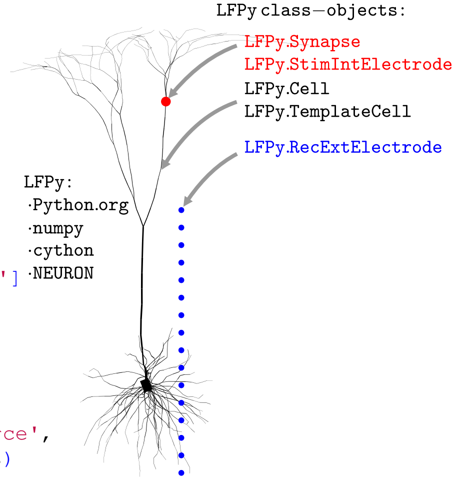
LFPy.RecExtElectrode:

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

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

- Usage

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



LFPy - Class overview

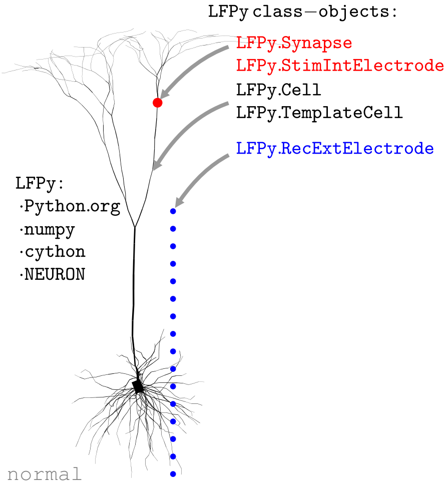
LFPy.RecExtElectrode:

- 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
```

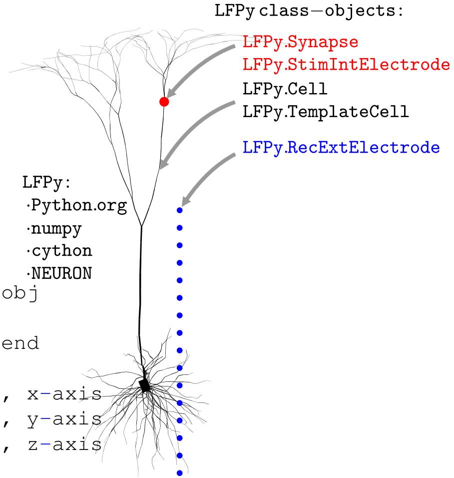


LFPy - Class overview

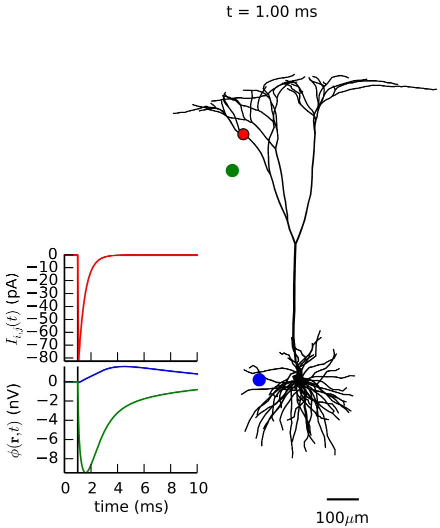
LFPy.lfpcalc.calc_lfp_*():

- 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
```

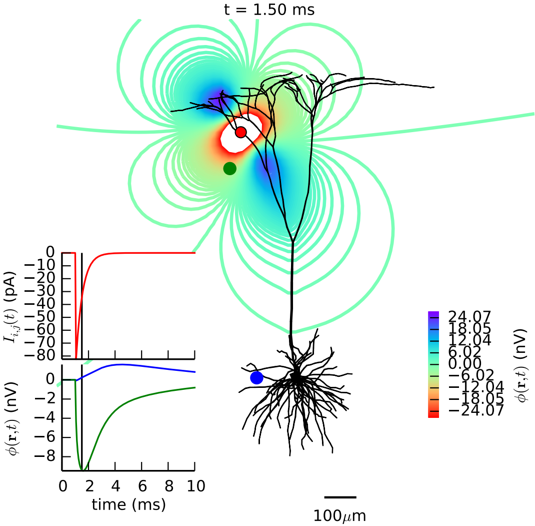


Forward modeling of extracellular potentials



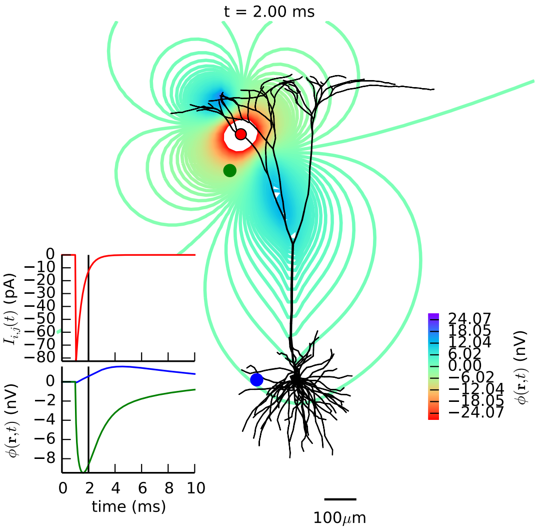
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



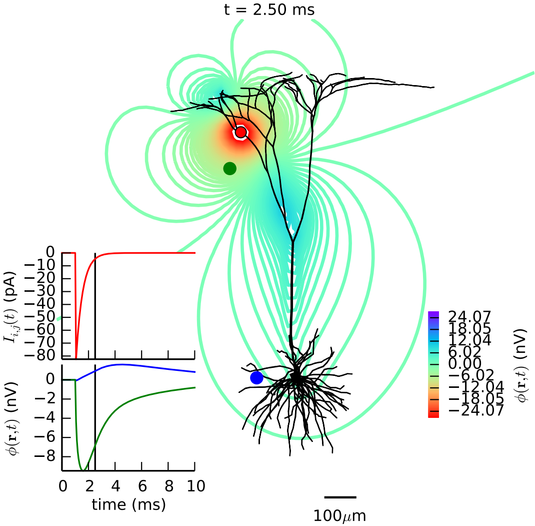
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



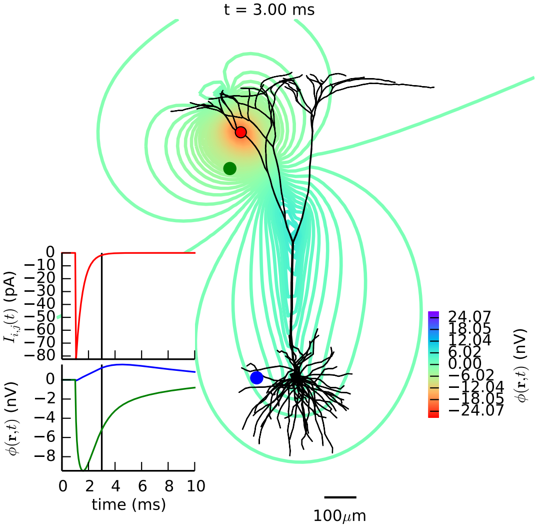
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



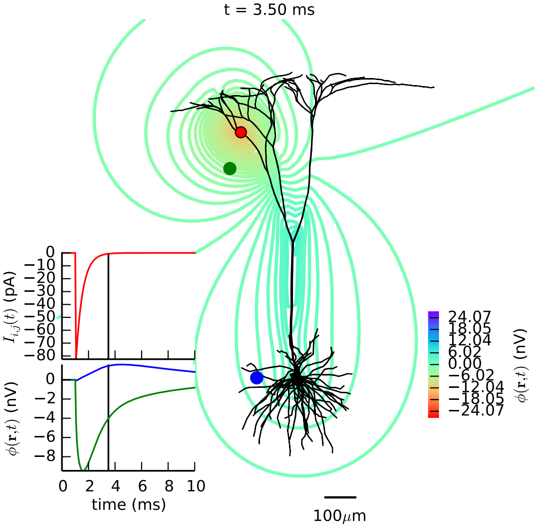
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



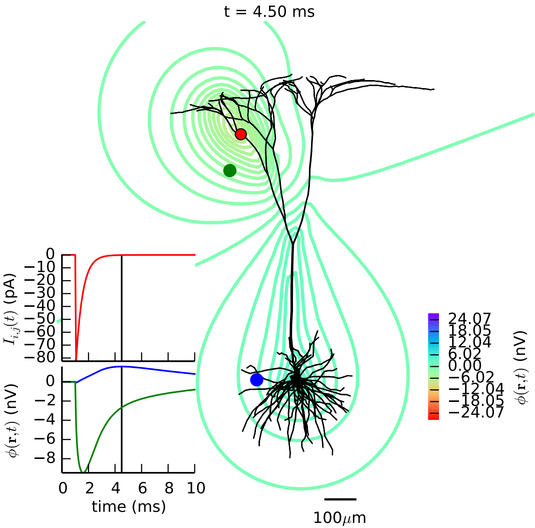
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



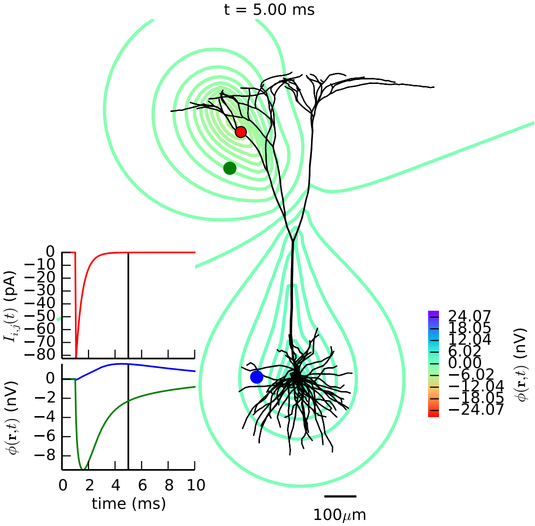
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



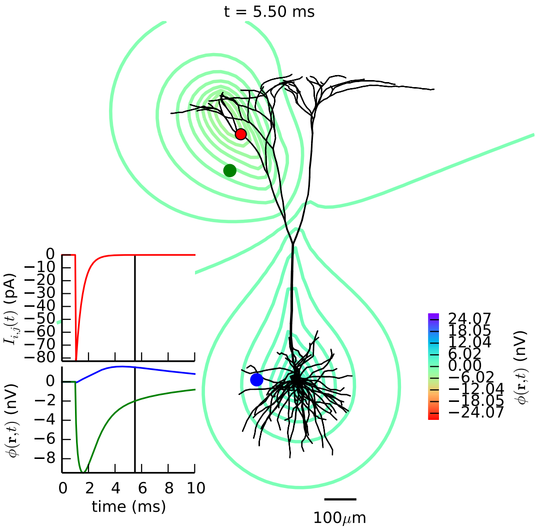
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



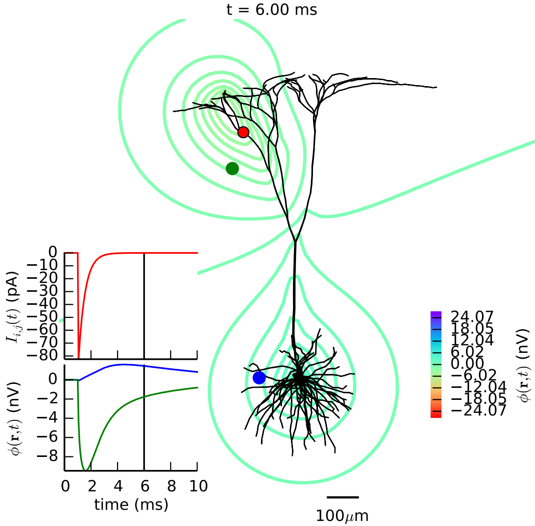
Passive propagation of synapse current input in passive cable model

Forward modeling of extracellular potentials



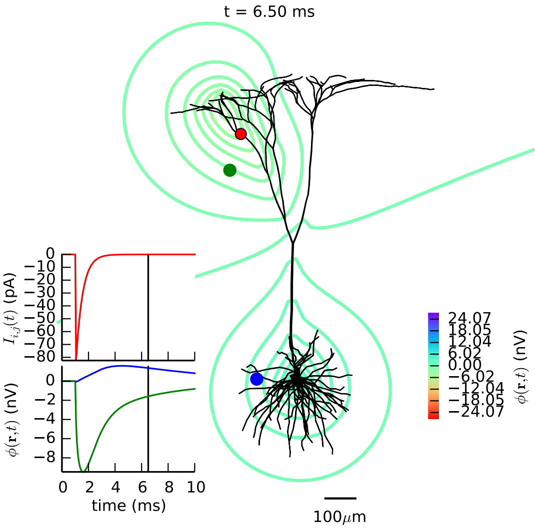
Passive propagation of synapse current input in passive cable model

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

EEG and MEG signal predictions

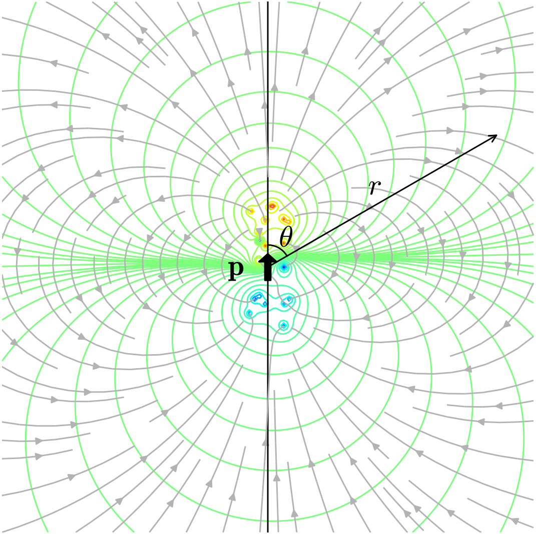
- Multipole expansion of potential of arbitrary distribution:

$$\begin{aligned}
\phi(\mathbf{r}) &= \frac{1}{4\pi\sigma}\sum_{n=1}^{N}\frac{I_n}{|\mathbf{r}-\mathbf{r}_n|}d\mathbf{r} \\
&\equiv \sum_{m=1}^{\infty}\phi_m
\end{aligned}$$

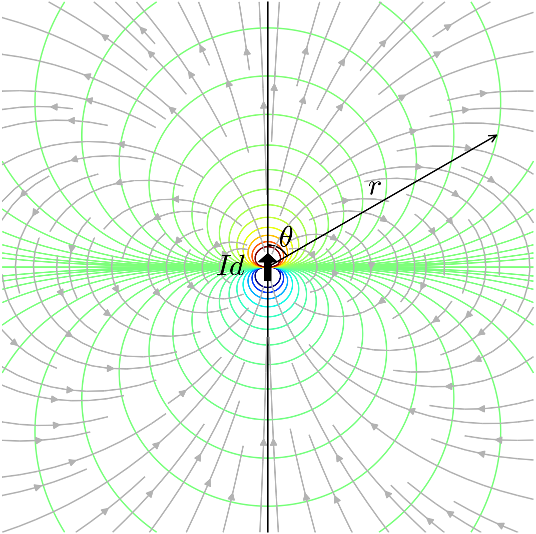
$$\begin{aligned}
\phi_1 &= \frac{\sum_{n=1}^{N} I_n}{4\pi\sigma r}, \phi_2 = \frac{\mathbf{p}\cdot\mathbf{r}}{4\pi\sigma r^3} \\
\phi_3 &\propto \frac{1}{r^3},\ \phi_4 \propto \frac{1}{r^4}, \ldots
\end{aligned}$$

- current dipole: $\mathbf{p} = \sum_{n=1}^{N} I_n \mathbf{r}_n$
- neurons closed electric circuits: no monopoles!
- quadropoles and higher terms can be ignored

EEG and MEG signal predictions



EEG and MEG signal predictions



EEG and MEG signal predictions

- Electric potential in infinite homogeneous medium

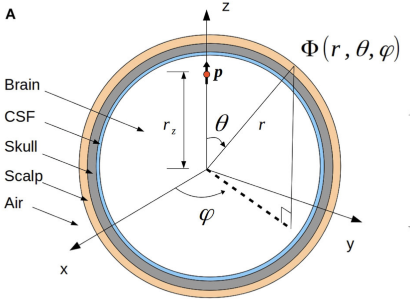
$$V = \frac{\mathbf{p} \cdot \mathbf{r}}{4\pi\sigma r^3}$$

EEG and MEG signal predictions

- Electric potential in infinite homogeneous medium

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

- The head is not a homogeneous volume conductor
- Analytical four-sphere volume conductor model (Næss et al., Front Human Neurosci, 2017)



EEG and MEG signal predictions

class `LFPy.FourSphereVolumeConductor`:

```
from LFPy import Cell, FourSphereVolumeConductor
import numpy as np
# ....
cell.simulate(rec_current_dipole_moment=True)
# geometry
radii = [79000., 80000., 85000., 90000.] # um
sigmas = [0.3, 1.5, 0.015, 0.3] # S/m
sensor_locations = np.array([[0., 0., 90000.]]) # um
dipole_location = np.array([0., 0., 78000.]) # um
# instantiate 4-sphere model class, compute potential
sphere = LFPy.FourSphereVolumeConductor(radii, sigmas,
                                        sensor_locations)
phi = sphere_model.calc_potential(cell.current_dipole_moment,
                                  dipole_location)
```

EEG and MEG signal predictions

- Magnetic field of dipole (magnetostatic Biot-Savart law)

$$\mathbf{B} = \frac{\mu}{4\pi}\frac{\mathbf{p}\times\mathbf{r}}{r^3}\,.$$

$$\mathbf{B} = \mu\mathbf{H} + M$$

  permeability $\mu \approx \mu_0 \equiv 4\pi \cdot 10^{-7}$ Tm/A,
  magnetization $M \approx 0$

- class `LFPy.MEG`:

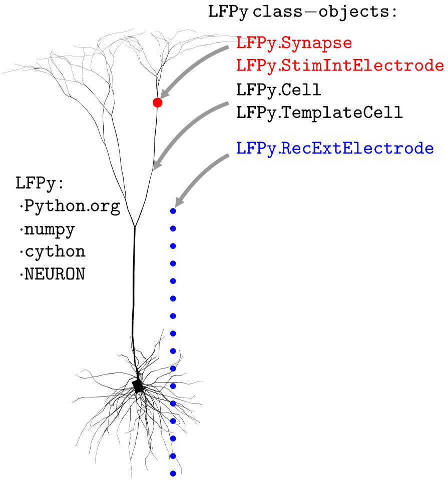
```
cell.simulate(rec_current_dipole_moment=True)
dipole_location = np.array([0, 0, 0]) # um
sensor_locations = np.array([[1E4, 0, 0]]) # um
meg = LFPy.MEG(sensor_locations, mu=4*np.pi*1E-7)
H = meg.calculate_H(cell.current_dipole_moment, dipole_location)
B = H*meg.mu
```

- Negligible contribution of volume currents in spherically symmetric conductors (Hämäläinen et al., Rev Modern Physics 1993)

LFPy - Class overview

Documentation and resources:

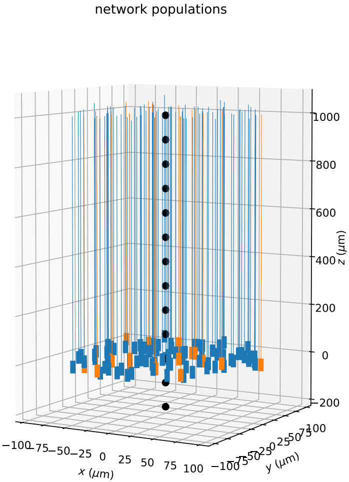
- LFPy homepage (http://LFPy.rtfd.io)
- autodoc w. `sphinx`:
  `cd /path/to/LFPy`
  `sphinx-build -b html documentation docs`
  see `docs/index.html`
- IPython magic (`numpy.sin?`, `LFPy.Synapse??`)
- NEURON homepage (http://www.neuron.yale.edu/)



LFPy - Class overview

LFPy.Network:

- Facilitates creation of networks:
  - populations
  - neurons
  - connections
  - parallel management
  - forward-model predictions (through `RexExtElectrode`)



- Main arguments and setup:

```
from LFPy import Network
networkParams = dict(
    dt = 2**-4,
    tstop = 1200.,
    v_init = -65.,
    celsius = 6.5,
    OUTPUTPATH = OUTPUTPATH)
network = Network(**networkParams)
```

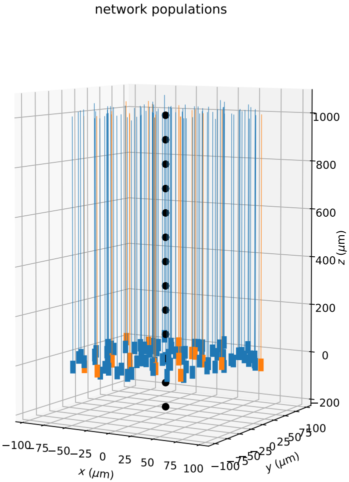
LFPy - Class overview

LFPy.NetworkCell:

- Inherited from `TemplateCell`
- Retains all `Cell/TemplateCell` functionality
- Adds methods for spike detection/connections
- Parameterization:

```
cellParams = dict(
    morphology='BallAndStick.hoc',
    templatefile='BallAndStickTemplate.ho
    templatename='BallAndStickTemplate',
    templateargs=None,
    delete_sections=False,)
cell = LFPy.NetworkCell(**cellParams)
```

- Cells set up as part of a `NetworkPopulation`



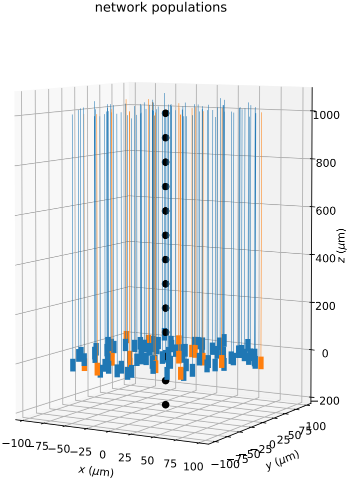
LFPy - Class overview

LFPy.NetworkPopulation:

- Represents populations of N NetworkCell instances
- Cells distributed across MPI ranks

```
popParams = dict(
    Cell=NetworkCell,
    cell_args = cellParams,
    pop_args = dict(
        radius=100.,
        loc=0.,
        scale=20.),
    rotation_args = dict(x=0., y=0.),
)
network.create_population(name=name,
                          POP_SIZE=size,
                          **popParams)
```

- Cells set up as part of a population



`LFPy` - Class overview

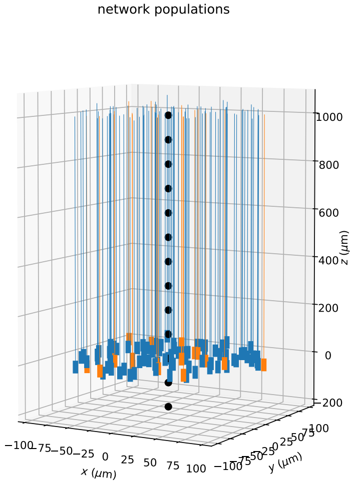
LFPy.Network:

- `Network` objects are transparent

  `network.populations[name].cells[cellID]`

- Create connections:

```
# connection matrix (boolean)
connectivity =
network.get_connectivity_rand(
     pre=name,
     post=name, connprob=connprob)
# connect populations
conncount, syncount = network.connect(
     pre=name, post=name,
     connectivity=connectivity,
     syntype=synapseModel,
     synparams=synapseParameters,
     **kwargs)
```



LFPy - Class overview

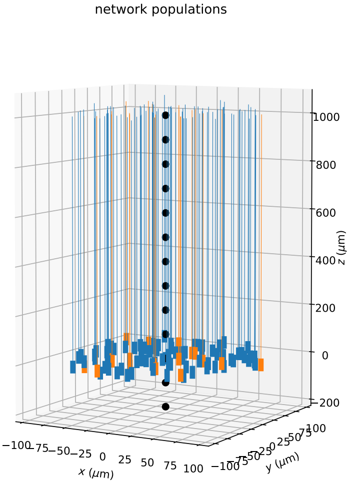
LFPy.Network:

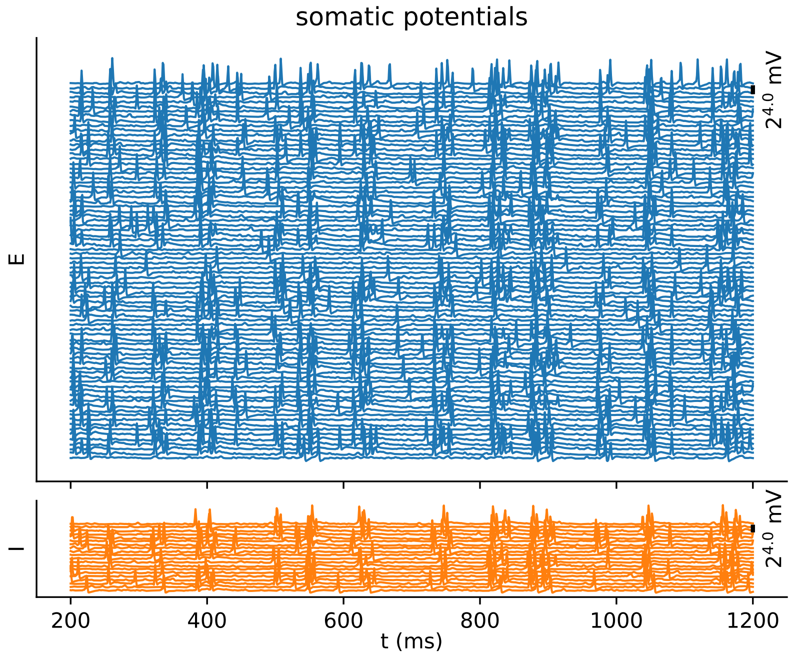
- Extracellular recording device:

```
electrodeParams = dict(**kwargs)
electrode =
RecExtElectrode(**electrodeParams)
```

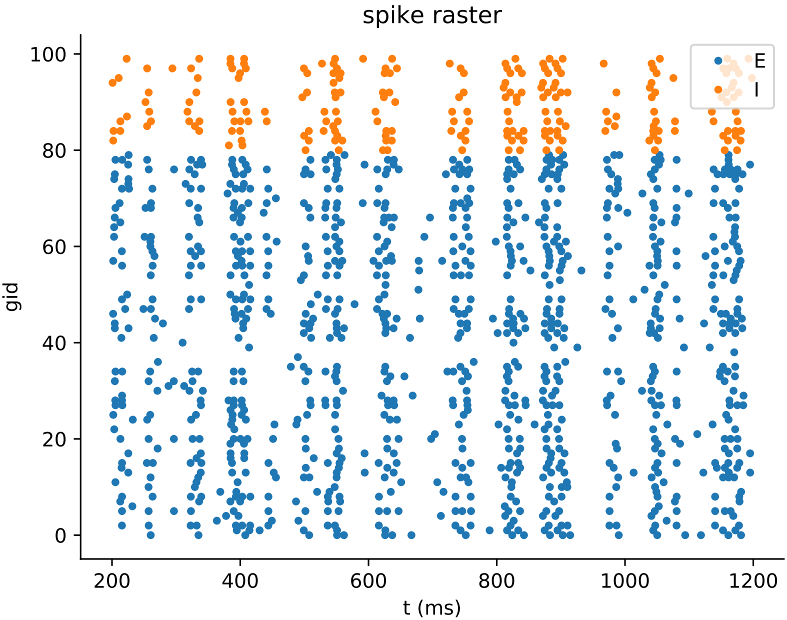
- Running simulation with measurements:

```
# method Network.simulate() parameters:
networkSimulationArguments = dict(
    rec_current_dipole_moment = True,
    rec_pop_contributions = True,
    to_memory = True,
    to_file = False
)
# run simulation:
SPIKES, OUTPUT, DIPOLEMOMENT =
network.simulate(
     electrode=electrode,
     **networkSimulationArguments
```

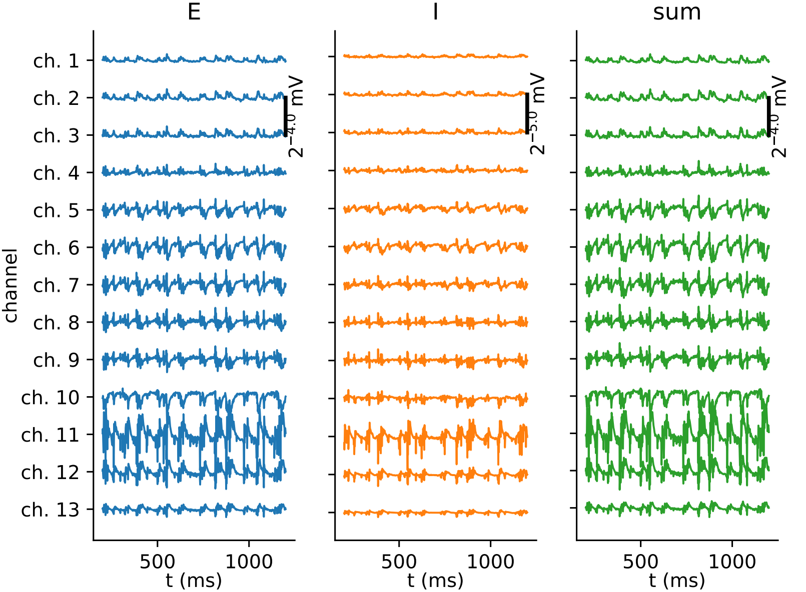


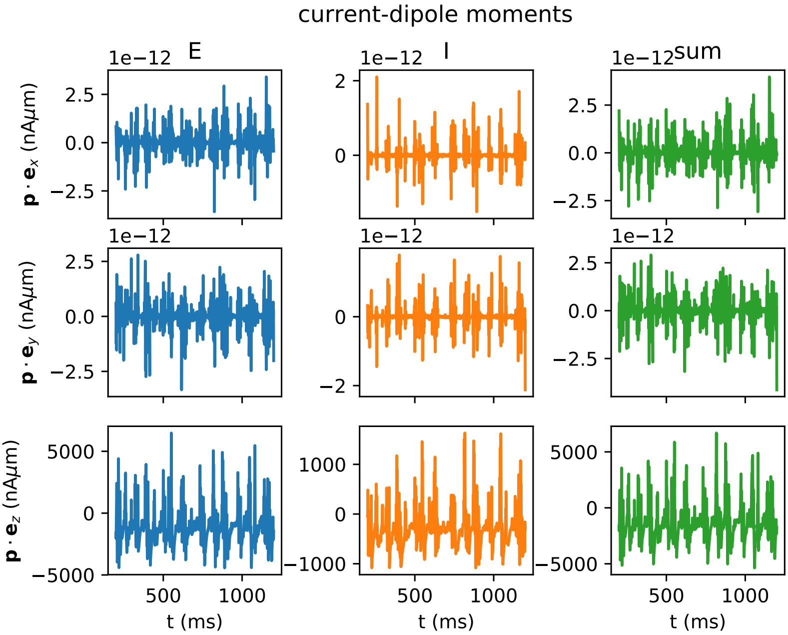


somatic potentials
E
I
24.0 mV
24.0 mV
200
400
600
800
1000
1200
t (ms)


spike raster
E
I
gid
t (ms)
0
20
40
60
80
100
200
400
600
800
1000
1200

extracellular potentials




current-dipole moments
E
I
sum
1e−12
$\mathbf{p} \cdot \mathbf{e}_x$ (nAμm)
$\mathbf{p} \cdot \mathbf{e}_y$ (nAμm)
$\mathbf{p} \cdot \mathbf{e}_z$ (nAμm)
t (ms)

Questions?

Hear more about LFPy2.0 @ P232!
Related posters:

- P79: Berthet et al.: Selectivity and sensitivity of cortical neurons to electric stimulation using ECoG electrode arrays
- 199: Tran et al.: Simulating extracellular signatures of action potentials using single compartment neurons and geometrical filtering
- P257: Skaar et al.: Estimation of model parameters from LFPs of spiking neuron networks using deep learning
- P233: Von Papen et al.: Quantitative comparison of a mesocircuit model with motor cortical resting state activity in the macaque monkey

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

New Results

Multimodal modeling of neural network activity: computing LFP, ECoG, EEG and MEG signals with LFPy2.0

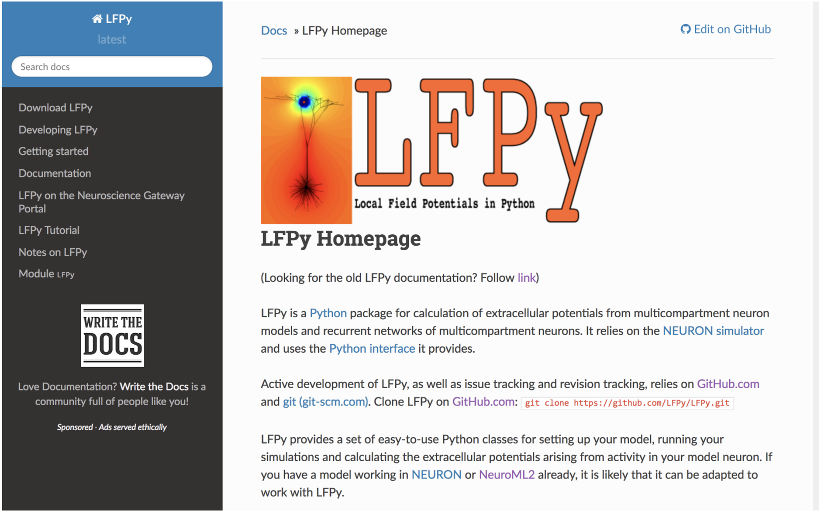
Espen Hagen, Solveig Næss, Torbjørn V Ness, Gaute T Einevoll

doi: https://doi.org/10.1101/281717

Abstract

Recordings of extracellular electrical, and later also magnetic, brain signals have been the dominant technique for measuring brain activity for decades. The interpretation of such signals is however nontrivial, as the measured signals result from both local and distant neuronal activity. In volume-conductor theory the extracellular potentials can be calculated from a distance-weighted sum of contributions from transmembrane

`LFPy` - Further reading and material



LFPy - Further reading and material

LFPy
latest

Search docs

- Download LFPy
- Developing LFPy
- Getting started
- Documentation
- LFPy on the Neuroscience Gateway Portal
- LFPy Tutorial
- Notes on LFPy

⊟ Module `LFPy`

- class `Cell`
- class `TemplateCell`
- class `NetworkCell`
- class `PointProcess`
- class `Synapse`
- class `StimIntElectrode`
- class `RecExtElectrode`
- class `Network`
- class `NetworkPopulation`

Docs » Module `LFPy` Edit on GitHub

Module `LFPy`

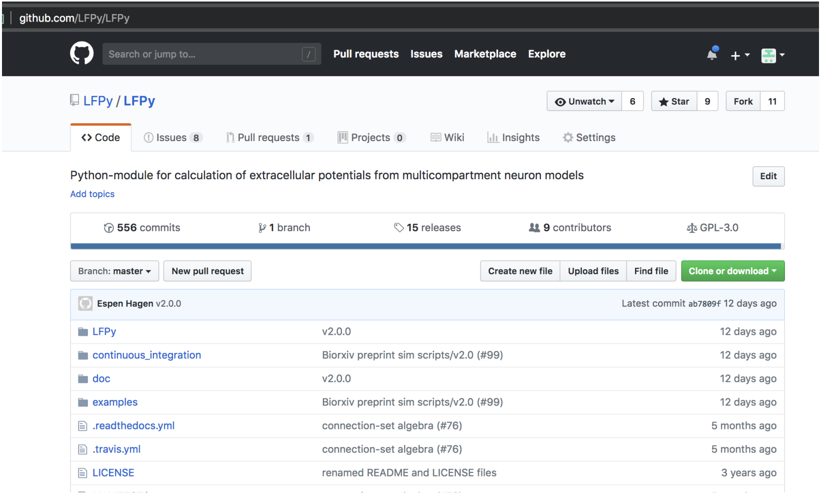
Initialization of LFPy, a Python module for simulating extracellular potentials.

Group of Computational Neuroscience, Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences.

Classes:

- Cell - The pythonic neuron object itself laying on top of NEURON representing cells
- TemplateCell - Similar to Cell, but for models using cell templates
- Synapse - Convenience class for inserting synapses onto Cell objects
- StimIntElectrode - Convenience class for inserting electrodes onto Cell objects
- PointProcess - Parent class of Synapse and StimIntElectrode
- RecExtElectrode - Class for performing simulations of extracellular potentials

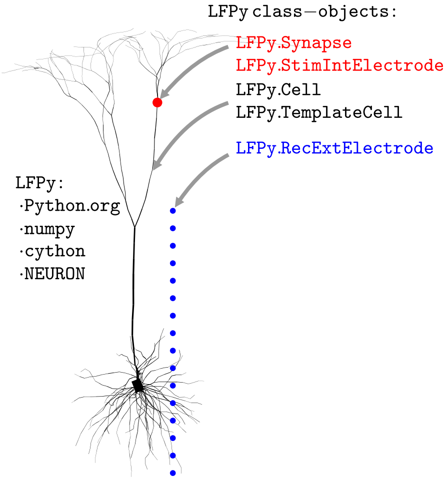
LFPy - Further reading and material



LFPy - Examples

Example **Python** `files`

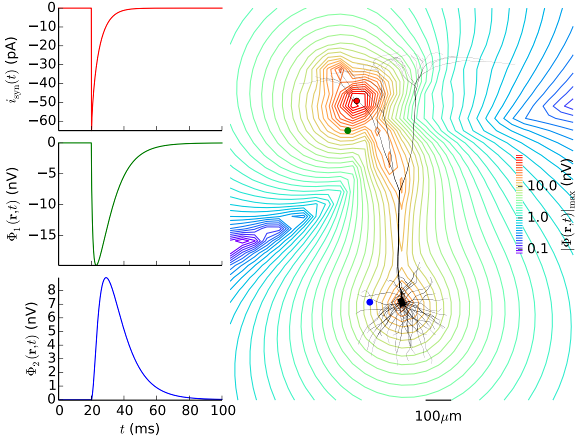
- `/path/to/LFPy/examples`
- Compute extracellular potentials
  - passive vs. active models
  - single-synapse vs. multi-synapse responses
  - extracellular action potential waveforms
  - population signal
- All use **`LFPy.Cell`**, **`LFPy.Synapse`**, **`LFPy.RecExtElectrode`**, ...



LFPy - Examples

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

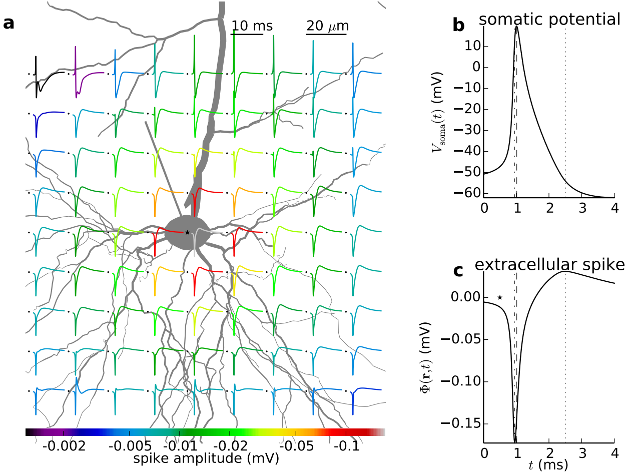
Apical synapse response, passive cable model



LFPy - Examples

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

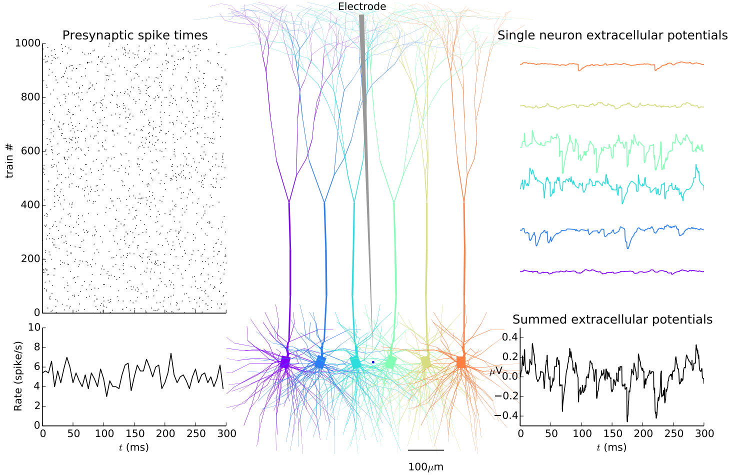
Layer 5b action potential (Hay et al. 2011), `LFPy.TemplateCell`



LFPy - Examples

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

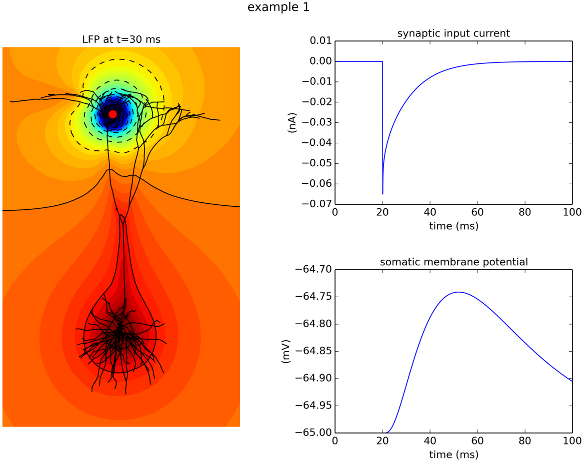
Extracellular potentials of small model population, shared input



LFPy - Examples

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

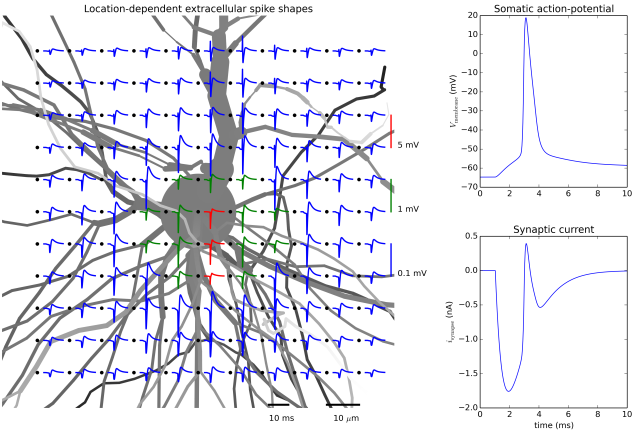
Extracellular potentials, single-synapse input current



LFPy - Examples

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

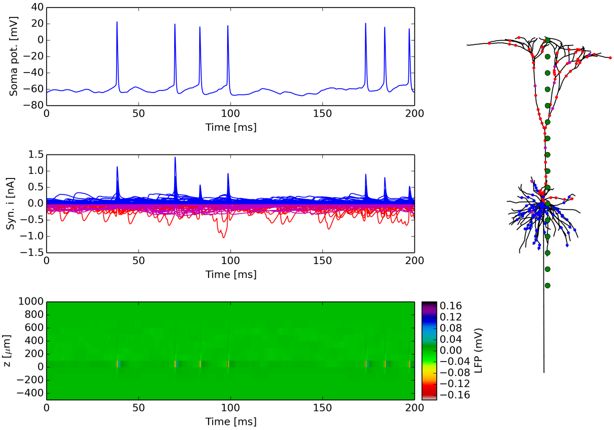
Extracellular potentials for action-potential of L5 pyramidal cell



LFPy - Examples

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

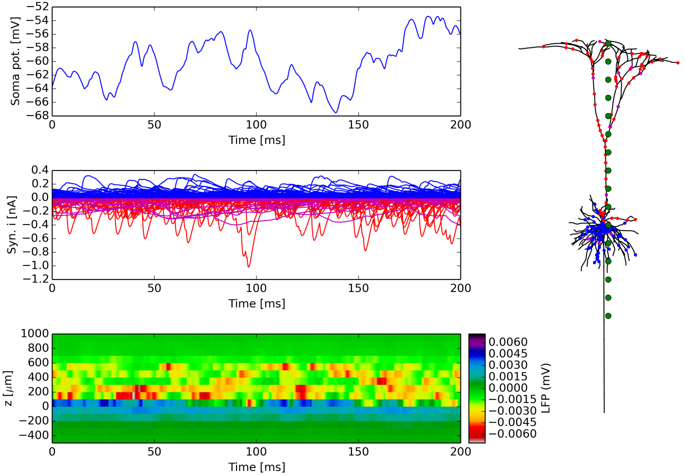
Extracellular potentials, synapse currents, somatic voltage, distributed synapses, active model



LFPy - Examples

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

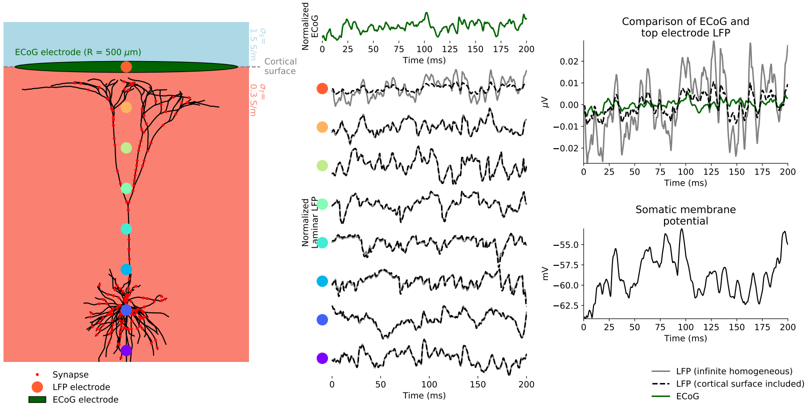
Extracellular potentials, synapse currents, somatic voltage, distributed synapses, passive model



LFPy - Examples

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

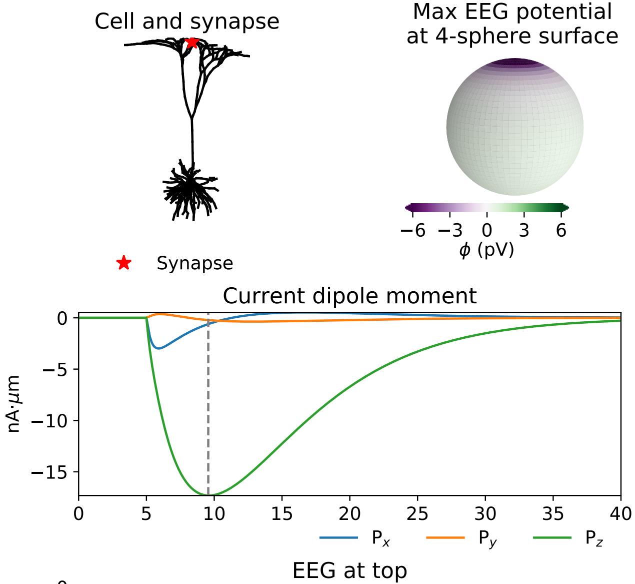
Extracellular potentials as recorded by ECoG contact



LFPy - Examples

/path/to/LFPy/examples/example_EEG.py

Single-synapse contribution to surface EEG



OXFORD

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

Questions?

Acknowledgements

- The European Union Horizon 2020 Research and Innovation Programme under Grant Agreement No. 720270/785907 [Human Brain Project (HBP) SGA1/SGA2]
- The Norwegian Ministry of Education and Research (SUURPh Programme)
- The Norwegian Research Council (NFR) through COBRA, NOTUR - NN4661K
- Organization for Computational Neurosciences

LFPy - Class overview

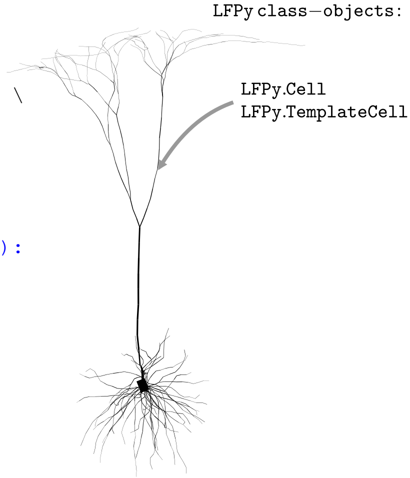
LFPy.Cell:

- 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 - Unit tests

unittest module:

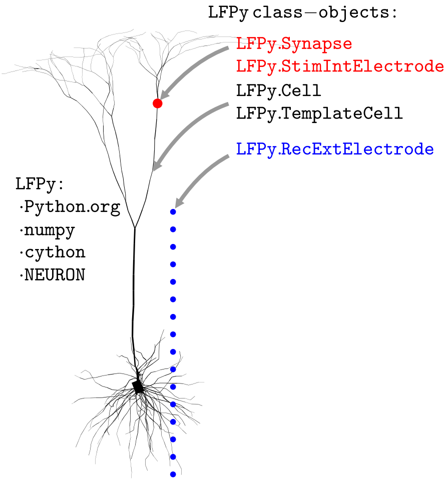
- runs code
- check if output is correct (validate LFPy output against analytical expressions for equivalent ball&stick models)
- run tests using nose module:

```
cd /path/to/LFPy
nosetests
```

- output:

```
...........
..................
------------------------------
Ran 279 tests in 79.512s

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()
```