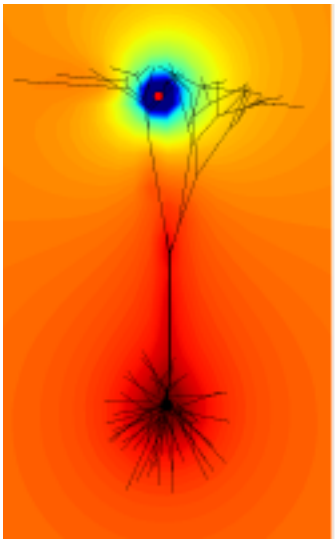
LFPy Homepage

Release 2.0.2

Espen Hagen

Mar 05, 2019

Contents



LFPy

Local Field Potentials in Python

(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.

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 previous versions (v1.x.x), LFPy was 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".

Since v2, LFPy also supports networks of multicompartment neurons, calculations of current-dipole moments, and predictions of both electric potentials and magnetic signals from a series of different volume-conductor models, as described in the bioRXiv preprint "Multimodal modeling of neural network activity: computing LFP, ECoG, EEG and MEG signals with LFPy2.0" by Espen Hagen, Solveig Næss, Torbjørn V Ness, Gaute T Einevoll, found at https://doi.org/10.1101/281717.

Citations:

- LFPy v2.x: Hagen E, Næss S, Ness TV and Einevoll GT (2018). Multimodal modeling of neural network activity: computing LFP, ECoG, EEG and MEG signals with LFPy2.0. bioRXiv. doi: 10.1101/281717

- LFPy v1.x: Linden H, Hagen E, Leski S, Norheim ES, Pettersen KH and Einevoll GT (2013). 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 Mathemathical Sciences and Technology, at the Norwegian University of Life Sciences , in collaboration with the Laboratory of Neuroinformatics, Nencki Institute of Experimental Biology, Warsaw, Poland, and Center for Integrative Neuroplasticity (CINPLA) at the University of Oslo, Norway. The effort was supported by International Neuroinformatics Coordinating Facility (INCF), The Research Council of Norway (eScience, NevroNor, COBRA) and EU-FP7 (BrainScaleS).

This scientific software is released under the GNU Public License GPLv3.

CHAPTER 1

Tutorial slides on LFPy

Slides for OCNS 2018 meeting tutorial T5: Modeling and analysis of extracellular potentials hosted in Seattle, USA on LFPy: CNS2018_LFPy_tutorial.pdf

Slides for OCNS 2017 meeting tutorial T4: Modeling and analysis of extracellular potentials hosted in Antwerp, Belgium on LFPy and hybridLFPy: CNS2017_LFPy_tutorial.pdf

Slides from OCNS 2015 meeting tutorial T2: Modeling and analysis of extracellular potentials hosted in Prague, Czech Republic on LFPy and hybridLFPy: CNS2015_LFPy_tutorial.pdf

Slides from OCNS 2014 meeting tutorial T4: Modeling and analysis of extracellular potentials hosted in Quebec City: hybridlfpy_tutorial_OCNS2014.pdf

As part of the OCNS 2013 meeting workshop Modeling and interpretation of extracellular potentials, there was also a talk on LFPy. The slides can be found here: lfpy-tutorial_OCNS2013.pdf.

CHAPTER 2

Related projects

LFPy has been used extensively in ongoing and published work, and may be a required dependency by the publicly available Python modules:

- ViSAPy - Virtual Spiking Activity in Python (https://github.com/espenhgn/ViSAPy, http://software.incf.org/software/visapy), as described in Hagen, E., et al. (2015), J Neurosci Meth, DOI:10.1016/j.jneumeth.2015.01.029
- ViMEAPy that can be used to incorporate heterogeneous conductivity in calculations of extracellular potentials with LFPy (https://bitbucket.org/torbness/vimeapy, http://software.incf.org/software/vimeapy). ViMEAPy and it's application is described in Ness, T. V., et al. (2015), Neuroinform, DOI:10.1007/s12021-015-9265-6.
- hybridLFPy - biophysics-based hybrid scheme for calculating the local field potential (LFP) of spiking activity in simplified point-neuron network models (https://github.com/INM-6/hybridLFPy), as described in Hagen, E. and Dahmen, D., et al. (2016), Cereb Cortex, DOI:10.1093/cercor/bhw237

CHAPTER 3

Contents

3.1 Download LFPy

Downlad the latest stable version of LFPy from the Python Package Index: http://pypi.python.org/pypi/LFPy

Or, download the development version of LFPy using git from GitHub.com into a local folder:

```
cd <where to put repositories>
git clone https://github.com/LFPy/LFPy.git
```

The LFPy source code and examples is then found under "LFPy".

The stable versions of LFPy can be accessed by listing and checking out tags, e.g.,

```
cd <path to LFPy>
git tag -l
git checkout <tag name>
```

To browse the documentation and source codes online, see http://lfpy.readthedocs.io/classes.html or https://github.com/LFPy/LFPy

3.2 Developing LFPy

As development of LFPy is now moved onto GitHub (https://github.com/LFPy), one can now fully benefit on working with forks of LFPy, implement new features, and share code improvements through pull requests. We hope that LFPy can be improved continously through such a collaborative effort.

A list of various LFPy issues, bugs, feature requests etc. is found here. If you want to contribute code fixes or new features in LFPy, send us a pull request.

3.3 Getting started

3.3.1 Dependencies

To install LFPy you will need the following:

1. Python. LFPy's unit-test suite is integrated with and continuously tested using Travis-CI. Tests are run using NEURON 7.4 and Python 2.7, 3.4, 3.5 and 3.6, as well as other Python dependencies listed next. The code build testing status, results of the last test, test coverage test using Coveralls and documentation status can be seen here:

2. Python modules setuptools, numpy, scipy, matplotlib, Cython, h5py, mpi4py, csa

   Depending on how LFPy is obtained, missing dependencies will be installed automatically (using `pip`). Manual installation of dependencies can be done using `pip` and the `requirements.txt` file which is supplied with the LFPy source codes (since v2.0). See next section for details.

3. Some example files may rely on additional Python dependencies. If some examples should fail to run due to some `ImportError`, identify the missing dependency name and run

   ```
   pip install <package>
   ```

   Alternatively, use the system default package manager, for example

   ```
   sudo apt install python-<package> # or
   conda install package
   ```

4. NEURON compiled as a Python module, so the following should execute without error in Python console:

   ```
   import neuron
   neuron.test()
   ```

   If this step fails, see the next section.

   LFPy now requires NEURON 7.4 or newer, and should be compiled with MPI in order to support new parallel functionality in LFPy, i.e., execution of networks in parallel.

5. Cython (C-extensions for python) to speed up simulations of extracellular fields. Tested with version > 0.14, and known to fail with version 0.11. LFPy works without Cython, but simulations may run slower and is therefore not recommended.

3.3.2 Installing LFPy

There are few options to install LFPy:

1. From the Python Package Index with only local access using pip

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

   as sudoer (in general not recommended as system Python files may be overwritten):

   ```
   sudo pip install LFPy
   ```

   Upgrading LFPy from the Python package index (without attempts at upgrading dependencies):

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

   LFPy release candidates can be installed as

```
pip install --pre --index-url https://test.pypi.org/simple/ --extra-index-url␣
↪https://pypi.org/simple LFPy --user
```

2. From source:

```
tar -xzf LFPy-x.x.tar.gz
cd LFPy-x.x
(sudo) pip install -r requirements.txt (--user) # install dependencies
(sudo) python setup.py install (--user)
```

3. From development version in our git (https://git-scm.com) repository:

```
git clone https://github.com/LFPy/LFPy.git
cd LFPy
# git tag -l # list versions
# git checkout <tag name> # choose particular version
(sudo) pip install -r requirements.txt (--user) # install dependencies
(sudo) python setup.py install (--user)
```

4. If you only want to have LFPy in one place, you can also build the LFPy Cython and NEURON NMODL extensions in the source directory. That can be quite useful for active LFPy development.

```
python setup.py develop --user
```

With any changes in LFPy `*.pyx` source files, rebuild LFPy.

In case of problems, it may be necessary to remove temporary and compiled files from the git repository before new attempts at building LFPy can be made:

```
git clean -n # list files that will be removed
git clean -fd # remove files
```

In a fresh terminal and python-session you should now be able to issue:

```
import LFPy
```

3.3.3 Uninstalling LFPy

Some times it may be necessary to remove installed versions of LFPy. Depending on how LFPy was installed in the first place, it should under most circumstances suffice to execute

```
(sudo) pip uninstall LFPy
```

If several versions was installed in the past, repeat until no more LFPy files are found.

3.4 Documentation

To generate the html documentation using Sphinx, issue from the LFPy source code directory:

```
sphinx-build -b html <path to LFPy>/doc <path to output>
```

The main html file is in `<path to output>/index.html`. Numpydoc and the ReadTheDocs theme may be needed:

```
pip install numpydoc --user
pip install sphinx-rtd-theme --user
```

3.4.1 Installing NEURON with Python

For most users, and even though NEURON (http://neuron.yale.edu) provides a working Python interpreter, making NEURON work as a Python module may be quite straightforward using pre-built Python distributions such as the Anaconda Scientific Python distribution (http://continuum.io) or Enthought Canopy (https://www.enthought.com/products/canopy/). We here provide some short step-by-step recipes on how to set up a working Python environment using Anaconda with the standard pre-built NEURON binaries on Linux, OSX and Windows.

Ubuntu 18.04.1 LTS 64-bit with Anaconda Scientific Python distribution

Probably the simplest solution relying on no source code compilation. This recipe is tested on a clean installation of Ubuntu 18.04.1 running in VirtualBox (https://www.virtualbox.org). The VirtualBox guest additions were installed.

1. Download and install Anaconda using the 64-bit Linux installer script from https://www.anaconda.com/download (https://repo.anaconda.com/archive/Anaconda3-5.3.1-Linux-x86_64.sh)

   Open Terminal application, then issue:

   ```
   cd $HOME/Downloads
   bash Anaconda3-5.3.1-Linux-x86_64.sh
   ```

   Accept the license and default options. Allow the installer to modify your `.bashrc` file. Installing MS Visual Studio Code (VSCode) is optional.

2. Download and install the 64-bit Debian/Ubuntu .deb file with NEURON from https://www.neuron.yale.edu/neuron/download (https://neuron.yale.edu/ftp/neuron/versions/v7.6/7.6.2/nrn-7.6.2.x86_64-linux-py-36-35-27.deb)

3. The `readline`, `ncurses`, `libtool` library files as well as `make` may be needed by NEURON. Install them from the terminal calling

   ```
   sudo apt install libreadline-dev libncurses-dev libtool make
   ```

4. Edit your `.bashrc` or similar file located in the `$HOME` folder, e.g., by calling in the terminal `gedit $HOME/.bashrc`, to include the lines:

   ```
   # make NEURON python module available to Anaconda python
   export PYTHONPATH="/usr/local/nrn/lib/python/:$PYTHONPATH"
   ```

5. Open a fresh terminal window (or type `source ~/.bashrc` in the terminal)

6. Activate the `base` conda environment:

   ```
   conda activate
   ```

7. Install LFPy dependencies (not installed by default) using conda

   ```
   conda install mpi4py # numpy matplotlib scipy h5py
   ```

8. Clone into LFPy using Git (https://git-scm.com), installable by calling `sudo apt install git` in the terminal:

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

9. Build LFPy from source inplace (without moving files)

```
python setup.py develop --user
```

or perform a local installation of LFPy:

```
python setup.py install --user
```

9. Test the installation from the terminal

```
nosetests
```

which will run through the LFPy test suite. Hopefully without errors.

Python 2.7

1. Follow the above steps up until point 6. Then in the terminal create a new environment based on Python 2.7 (assuming that the Python 3 version of Anaconda was installed):

```
conda create -n py27 python=2.7 anaconda
conda activate py27
```

2. Continue with step 7 above.

Ubuntu 18.04.1 LTS w. system Python 3.6

Another option is to rely on the system Python installation (Python 3.6.7). Make sure that nothing in `$PATH` points to an Anaconda-installed version of Python (may break e.g., `pip3`)

1. Install dependencies through the terminal:

```
sudo apt install libreadline-dev libncurses-dev libtool make
sudo apt install ipython3 cython3 jupyter-notebook python3-nose \
  python3-numpy python3-scipy python3-matplotlib python3-mpi4py python3-h5py
```

2. Download and install the 64-bit Debian/Ubuntu .deb file with NEURON from https://www.neuron.yale.edu/neuron/download (https://neuron.yale.edu/ftp/neuron/versions/v7.6/nrn-7.6.x86_64-linux.deb)

3. Edit your `.bashrc` or similar file located in the `$HOME` folder, e.g., by calling in the terminal `gedit $HOME/.bashrc`, to include the lines:

```
# make NEURON python module available to Anaconda python
export PYTHONPATH="/usr/local/nrn/lib/python/:$PYTHONPATH"
```

4. Open a fresh terminal window (or type `source ~/.bashrc` in the terminal)

5. Clone into LFPy using Git (https://git-scm.com), installable by calling `sudo apt install git` in the terminal:

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

6. Build LFPy from source inplace (without moving files)

```
python3 setup.py develop --user
```

or perform a local installation of LFPy:

```
python3 setup.py install --user
```

7. Test the installation from the terminal

```
nosetests3
```

which will run through the LFPy test suite. Hopefully without errors.

OSX 10.12.x with Anaconda Scientific Python distribution

By far the simplest solution relying on no source code compilation.

1. Download and install Anaconda using the 64-bit graphical installer from http://continuum.io/downloads
2. Download and install the 64-bit Mac `.pkg` file with NEURON from http://www.neuron.yale.edu/neuron/download. Do not choose to let the NEURON installer edit the `~/.bash_profile` file. The default file to edit is `~/.profile` (see below).
3. Edit your .profile or similar file located in the $HOME folder, e.g., by calling in the Terminal.app `open -t $HOME/.profile`, to include the lines:

```
# make nrniv, mknrnivmodl, etc. available from the command line
export PATH=/Applications/NEURON-7.5/nrn/x86_64/bin/:$PATH

# Append the path to the NEURON python extension module to PYTHONPATH
export PYTHONPATH=/Applications/NEURON-7.5/nrn/lib/python:$PYTHONPATH
```

4. Open a fresh terminal window
5. Install LFPy dependencies (not installed by default) using conda

```
conda install mpi4py
```

6. Clone into LFPy using Git:

```
git clone https://github.com/LFPy/LFPy.git
```

7. Build LFPy from source (without moving files)

```
python setup.py develop
```

8. Test the installation from the terminal

```
python -c "import LFPy"
NEURON -- VERSION 7.5 master (6b4c19f) 2017-09-25
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2016
See http://neuron.yale.edu/neuron/credits
```

If everything worked, you now have a working Python/NEURON/LFPy environment.

Windows with Anaconda Scientific Python distribution

Windows 10 Pro/Education (64-bit) install instructions:

1. Download and install Anaconda Python from https://www.anaconda.com/download.
2. Download and install NEURON from https://www.neuron.yale.edu/neuron/download. Tick the box to "Set DOS environment" (Otherwise Anaconda Python will not find the NEURON python module)
3. Download and install the Visual Studio C++ Build Tools 2015 from: http://landinghub.visualstudio.com/visual-cpp-build-tools. Choose the Visual C++ 2015 Build Tools option.
4. Download and install Git from https://git-scm.com/downloads
5. Download and install Microsoft MPI from the Official Microsoft Download Center: https://www.microsoft.com/en-us/download/details.aspx?id=55494. Choose the file "MSMpiSetup.exe".
6. Open the Anaconda Prompt under the Anaconda* folder in the start menu
7. Install additional LFPy dependencies using `conda` (to avoid package clashes with i.e., `pip install <package_name>`)

```
conda install mpi4py
```

8. Clone into LFPy using Git:

```
git clone https://github.com/LFPy/LFPy.git
```

9. Build LFPy from source (without moving files)

```
python setup.py develop
```

10. NEURON NMODL (.mod) files will not be autocompiled when building LFPy as on MacOS/Linux, as the mknrndll script cannot be run directly in the Anaconda Prompt. To fix this, run the `bash` file in the NEURON program group, change directory within "bash" to the `<LFPy>/LFPy/test` folder, then run `mknrndll`

3.4.2 Installing NEURON with Python from source

Some users have difficulties installing NEURON as a Python module, depending on their platform. We will provide some explanations here, and otherwise direct to the NEURON download pages; https://www.neuron.yale.edu/neuron/download and https://www.neuron.yale.edu/neuron/download/getstd. The NEURON forum (https://www.neuron.yale.edu/phpBB/) is also a useful resource for installation problems.

Dependencies: Ubuntu 18.04 LTS and other Debian-based Linux versions

The instructions below show how to meet all the requirements starting from a clean Ubuntu 18.4 for the installation of NEURON from the development branch.

Start by installing required packages. We aim to link with the system Python installation, not Anaconda Python. For Anaconda installations, make sure that the correct Python installations is found during NEURON's configure step below.

```
sudo apt install git build-essential autoconf libtool
sudo apt install libxext-dev libncurses-dev zlib1g-dev
sudo apt install bison flex libx11-dev
sudo apt install openmpi-bin libopenmpi-dev
sudo apt install python3-dev python3-numpy python3-scipy python3-matplotlib
sudo apt install ipython3 cython3
```

Linux/Unix installation of NEURON from source

Fetch the source code of NEURON using git

```
cd $HOME
mkdir neuron
cd neuron

git clone https://github.com/neuronsimulator/iv.git
git clone https://github.com/neuronsimulator/nrn.git
```

Set compilers, here using the GNU Compiler Collection (GCC). A compute cluster may have the Intel Compiler binaries (`icc/icpc/mpiicc/mpiicpc`)

```
export CC=gcc
export CXX=g++
export MPICC=mpicc
export MPICXX=mpicxx
```

Optional, compile and install InterViews binaries to the folder `$HOME/.local` folder (which should be in the default `$PATH` on most systems)

```
cd iv
sh build.sh
./configure --prefix=~/.local
make -j4
make install
```

Compile and install NEURON with InterViews and MPI. To disable InterViews, use `--without-iv` during the configuration step.

```
cd ../nrn
sh build.sh
./configure --prefix=~/.local --with-iv=~/.local --with-nrnpython=/usr/bin/python3.6 -
↪-with-mpi=/usr/bin/mpirun --with-paranrn
make -j4
make install
```

You might want to add the folder with NEURON binaries to your `$PATH`, so that you can easily compile NEURON mechanisms using `nrnivmodl` from the terminal. Add the following line to your `$HOME/.bashrc` (or equivalent) file:

```
export PATH=$HOME/.local/x86_64/bin:$PATH
```

Start a new terminal tab or type `source $HOME/.bashrc` to activate.

Install NEURON as a Python module

```
cd src/nrnpython/
python3 setup.py install --user
```

Now you should be able to `import neuron` from Python console and run a small test with success;

```
cd $HOME
ipython3
>>> import neuron
>>> neuron.test()
```

NEURON dependencies and installation on Mac OSX from source

Most of the development work and testing of LFPy has been done on MacOS (10.6-). Our preferred way of building Python has been through MacPorts; http://www.macports.org. Here is an step-by-step explanation on how to compile NEURON agains that installation of Python. Simpler solutions are stipulated above.

To start using MacPorts, follow the instructions on http://www.macports.org/install.php.

Building a python 2.7 environment using MacPorts issue in Terminal:

```
sudo port install python27 py27-ipython py27-numpy py27-matplotlib py27-scipy py27-
↪cython py27-mpi4py py27-h5py
```

Make the installed Python and IPython default:

```
sudo port select --set python python27
sudo port select --set ipython ipython27
```

Install the necessary packages for cloning into repository and compiling NEURON:

```
sudo port install automake autoconf libtool xorg-libXext ncurses mercurial bison flex
```

Install NEURON from the bleeding edge source code. The following recipe assumes a 64 bit build of NEURON and Python on MacOS 10.12, so change "x86_64-apple-darwin16.7.0" throughout to facilitate your system accordingly, as found by running `./config.guess` in the root of the NEURON source code folder;

```
#create a directory in home directory
cd $HOME
mkdir nrn64
cd nrn64

#creating directories
sudo mkdir /Applications/NEURON-7.5
sudo mkdir /Applications/NEURON-7.5/iv
sudo mkdir /Applications/NEURON-7.5/nrn

#Downloading bleeding edge source code
hg clone http://www.neuron.yale.edu/hg/neuron/iv
hg clone http://www.neuron.yale.edu/hg/neuron/nrn
cd iv

#compiling and installing IV under folder /Applications/nrn7.5
sh build.sh
./configure --prefix=/Applications/NEURON-7.5/iv \
        --build=x86_64-apple-darwin16.7.0 --host=x86_64-apple-darwin16.7.0 \
        --x-includes=/usr/X11/include --x-libraries=/usr/X11/lib
make
sudo make install

#Building NEURON with InterViews, you may have to alter the path --with-nrnpython=/
↪python-path
cd $HOME/nrn64/nrn
sh build.sh
./configure --prefix=/Applications/NEURON-7.5/nrn \
        --with-nrnpython=/opt/local/Library/Frameworks/Python.framework/Versions/2.7/
↪Resources/Python.app/Contents/MacOS/Python \
        --host=x86_64-apple-darwin16.7.0 --build=x86_64-apple-darwin16.7.0 \
        --with-paranrn \
```

(continues on next page)

(continued from previous page)

```
        --with-mpi \
        --with-iv=/Applications/NEURON-7.5/iv \
        CFLAGS='-O3 -Wno-return-type -Wno-implicit-function-declaration -Wno-implicit-
↪int -fPIC' \
        CXXFLAGS='-O3 -Wno-return-type -fPIC'
make
sudo make install
sudo make install after_install

#You should now have a working NEURON application under Applications. Small test;
#sudo /Applications/NEURON-7.5/nrn/x86_64/bin/neurondemo

#Final step is to install neuron as a python module
cd src/nrnpython
sudo python setup.py install
```

3.5 LFPy on the Neuroscience Gateway Portal

LFPy is installed on the Neuroscience Gateway Portal (NSG, see http://www.nsgportal.org), and can be used to execute simulations with LFPy both serially and in parallel applications on high-performance computing facilities. The access to the NSG is entirely free, and access to other neuronal simulation software (NEST, NEURON, etc.) is also provided. The procedure for getting started with LFPy on the NSG is quite straightforward through their web-based interface:

1. First, apply for a NSG user account by filling out their application form and sending it by email (follow instructions on http://www.nsgportal.org/portal2)
2. After approval, log in using your credentials, change password if necessary
3. As a first step after log in, create a new folder, e.g., named "LFPyTest" and with some description. This will be the home for your input files and output files, and should contain empty Data and Tasks folders
4. Press the "Data (0)" folder in the left margin. Press the "Upload/Enter Data" button, showing the Upload File interface. Add a label, e.g., "LFPyTest".
5. Next, LFPy simulation files have to be uploaded. As an example, download the example LFPy files https://github.com/espenhgn/LFPy/blob/master/examples/nsg_example/L5_Mainen96_wAxon_LFPy.hoc and https://github.com/espenhgn/LFPy/blob/master/examples/nsg_example/nsg_example.py into a new local folder "nsg_example". Modify as needed.
6. Zip the "nsg_example" folder, upload it to the NSG (cf. step 4) and press "Save"
7. Press "Tasks (0)" in the left margin and "Create New Task"
8. Enter some Description, e.g., "LFPyTest", and "Select Input Data". Hook off "LFPyTest" and press "Select Data"
9. Next, press "Select Tool", and then "Python (2.7.x)"
10. Then, go to the "Set Parameters" tab. This allows for specifying simulation time, main simulation script, and number of parallel threads. Set "Maximum Hours" to 0.1, and "Main Input Python Filename" to "nsg_example.py". Node number and number of cores per node should both be 1. Press "Save Parameters"
11. Everything that is needed has been set up, thus "Save and Run Task" in the Task Summary tab is all that is needed to start the job, but expect some delay for it to start.
12. Once the job is finished, you will be notified by email, or keep refreshing the Task window. The simulation output can be accessed through "View Output". Download the "output.tar.gz" file and unzip it. Among the output files, including stdout.txt and stderr.txt text files and jobscript details, the included folder "nsg_example"

will contain the input files and any output files. For this particular example, only a pdf image file is generated, "nsg_example.pdf"

3.6 LFPy Tutorial

This tutorial will guide you through a few first steps with LFPy. It is based on `example1.py` in the `LFPy/examples` folder. In order to obtain all necessary files, please obtain the LFPy source files as described on the Download page

Change directory to the LFPy examples folder, and start the interactive IPython interpreter

```
$ cd <path to LFPy>/examples
$ ipython
```

Let us start by importing `LFPy`, as well as the `numpy` and `os` modules

```
>>> import os
>>> import LFPy
>>> import numpy as np
```

Then we define a dictionary which describes the properties of the cell we want to simulate

```
>>> cell_parameters = {
>>>     'morphology' : os.path.join('morphologies', 'L5_Mainen96_LFPy.hoc'),     #␣
↪Mainen&Sejnowski, Nature, 1996
>>>     'tstart' : 0.,           # start time of simulation, recorders start at t=0
>>>     'tstop' : 20.,           # stop simulation at 20 ms.
>>>     'v_init' : -70.0,        # initial voltage at tstart
>>>     'Ra' : 35.4,             # axial resistivity
>>>     'cm' : 1.0,              # membrane capacitance
>>>     'passive' : True,        # switch on passive leak resistivity
>>>     'passive_parameters' : {'e_pas': -70.0, # passive leak reversal potential
>>>                             'g_pas': 0.001} # passive leak conductivity
>>> }
```

The only mandatory entry is `morphology`, which should point to a `hoc` file specifying the neuron's morphology. Here we also set the start and end times (in milliseconds). Many more options are available (such as adding custom NEURON mechanisms), but for now we leave other parameters at their default values.

Then, the Cell object is created issuing

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

Let us now add a synapse to the cell. Again, we define the synapse parameters first

```
>>> synapse_parameters = {
>>>     'idx' : cell.get_closest_idx(x=0., y=0., z=800.), # segment index for synapse
>>>     'e' : 0.,                   # reversal potential
>>>     'syntype' : 'ExpSyn',       # synapse type
>>>     'tau' : 2.,                 # syn. time constant
>>>     'weight' : .1,              # syn. weight
>>>     'record_current' : True           # record syn. current
>>> }
```

and create a Synapse object connected to our cell

```
>>> synapse = LFPy.Synapse(cell, **synapse_parameters)
```

Let us assume we want the synapse to be activated by a single spike at t = 5 ms. We have to pass the spike time to the synapse using

```
>>> synapse.set_spike_times(np.array([5.]))
```

We now have a cell with a synapse, and we can run the simulation

```
>>> cell.simulate(rec_imem=True)
```

Note the `rec_imem=True` argument, this means that the transmembrane currents will be saved - we need the currents to calculate the extracellular potential. An array with all transmembrane currents for the total number of segments `cell.totnsegs` is now accessible as `cell.imem`, the somatic voltage as `cell.tvec`, all with time stamps `cell.tvec`.

```
>>> print(cell.tvec.shape, cell.somav.shape, cell.totnsegs, cell.imem.shape)
```

The final element is the extracellular recording electrode. Again, we start by defining the parameters

```
>>> electrode_parameters = {
>>>     'sigma' : 0.3,   # extracellular conductivity
>>>     'x' : np.array([0]),
>>>     'y' : np.array([0]),
>>>     'z' : np.array([50])
>>> }
```

Here we define a single electrode contact at x = 0, y = 0, z = 50 µm, but a whole array of electrodes can be specified by passing array arguments.

The electrode (recording from `cell`) is created using

```
>>> electrode = LFPy.RecExtElectrode(cell, **electrode_parameters)
```

Finally, we calculate the extracellular potential at the specified electrode location

```
>>> electrode.calc_lfp()
```

The resulting LFP is stored in `electrode.LFP`.

```
>>> print(electrode.LFP.shape)
```

Finally, the cell morphology and synapse location can be plotted

```
>>> from matplotlib.collections import PolyCollection
>>> import matplotlib.pyplot as plt
>>> zips = []
>>> for x, z in cell.get_idx_polygons(projection=('x', 'z')):
>>>     zips.append(zip(x, z))
>>> polycol = PolyCollection(zips,
>>>                          edgecolors='none',
>>>                          facecolors='gray')
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.add_collection(polycol)
>>> ax.plot(cell.xmid[synapse.idx], cell.zmid[synapse.idx], 'ro')
>>> ax.axis(ax.axis('equal'))
>>> plt.show()
```

As well as the simulated output

```
>>> fig = plt.figure()
>>> plt.subplot(311)
>>> plt.plot(cell.tvec, synapse.i)
>>> plt.subplot(312)
>>> plt.plot(cell.tvec, cell.somav)
>>> plt.subplot(313)
>>> plt.plot(cell.tvec, electrode.LFP.T)
>>> plt.show()
```

3.6.1 More examples

More examples of LFPy usage are provided in the `trunk/examples` folder in the source code release, displaying different usages of LFPy.

The examples rely on files present inside the examples folder, such as morphology files (.hoc) and NEURON NMODL (.mod) files.

The easiest way of accessing all of these files is cloning the examples directory using git (https://git-scm.com):

```
$ git clone https://github.com/LFPy/LFPy.git
$ cd LFPy/examples
```

The files provided are

- example1.py
- example2.py
- example3.py
- example4.py
- example5.py
- example6.py
- example7.py
- example8.py
- example_mpi.py
- example_EPFL_neurons.py
- example_LFPyCellTemplate.py
- example_MEA.py
- example_anisotropy.py
- example_loadL5bPCmodelsEH.py
- example_network/example_Network.py
- example_EEG.py

3.7 Notes on LFPy

3.7.1 Morphology files

Cell morphologies can be specified manually in a `.hoc` file. For a simple example, see `examples/morphologies/example_morphology.hoc`. Note the following conventions:

- Sections should be named according to the following scheme:
  - `soma*[]` for somatic sections, `*` is optional
  - `dend*[]` for dendritic sections
  - `apic*[]` for apical dendrite sections
  - `axon*[]` for axonal sections
- Sections must be defined as types Section or SectionList (as `soma[1]` or `soma`)

Also the morphologies exported from the NEURON simulator (for example using Cell Builder -> Export) should work with LFPy, but some times `create soma` may have to be corrected to `create soma[1]` directly in the files. Multi-sectioned somas may occurr e.g., due to faulty conversion from NeuroLucida or SWC format, however, we recommend that these files are corrected. It may not affect the predictions of intracellular voltages, but have implications for predictions of extracellular potentials. We usually assume that the soma is a single compartment and that it is the root section for all other sections.

NEURON convention for creating morphology files in `hoc`:

```
/* -----------------------------------------------------
example_morphology.hoc

This hoc file creates a neuron of the following shape:

            \
             \
              \    /
               \  /
                V
                |
                |
                |
                O

Note the conventions:
 - use soma for the soma compartment,
 - use a name starting with dend or apic for the dendrites.
-----------------------------------------------------*/


create soma[1]
create dend[3]

soma[0] {
    pt3dadd(0, 0, 0, 25)
    pt3dadd(0, 0, 35, 25)
}

dend[0] {
    pt3dadd(0, 0, 35, 5)
```

(continues on next page)

(continued from previous page)

```
    pt3dadd(0, 0, 150, 5)
}

dend[1] {
    pt3dadd(0, 0, 150, 2)
    pt3dadd(-50, 20, 200, 1)
}

dend[2] {
    pt3dadd(0, 0, 150, 2)
    pt3dadd(30, 0, 160, 2)
}

connect dend[0](0), soma[0](1)
connect dend[1](0), dend[0](1)
connect dend[2](0), dend[0](1)
```

Other file formats

Support for SWC, NeuroLucida3 and NeuroML morphology file formats is added in LFPy, but consider this an experimental feature as the functionality is not properly tested. If there is something wrong with the files, strange behaviour may occur or LFPy may even fail to load the desired morphology.

3.7.2 Using NEURON NMODL mechanisms

Custom NEURON mechanisms can be loaded from external `.hoc`- or `.py`-files - see `example2.py` and `example3.py`. Python function definitions with arguments can also be given as input to the `Cell`-class, specifying model specific conductances etc. Remeber to compile any `mod` files (if needed) using `nrnivmodl` (or `mknrdll` on Windows).

These model specific declarations is typically run after the `Cell`-class try to read the morphology file, and before running the `_set_nsegs()` and `_collect_geometry()` procedures. Hence, code that modifies the segmentation of the morphology will affect the properties of the instantiated `LFPy.Cell` object.

3.7.3 Units

Units follow the NEURON conventions found here. The units in LFPy for given quantities are:

Quantity	Unit
Potential	[mV]
Current	[nA]
Conductance	[S/cm2]
Extracellular conductivity	[S/m]
Capacitance	[μF/cm2]
Dimension	[μm]
Synaptic weight	[µS]
Current dipole moment	[nA µm]
Magnetic field (H)	[nA/µm]
Permeability (µ)	[T m/A]

Note: resistance, conductance and capacitance are usually specific values, i.e per membrane area (lowercase r_m, g, c_m) Depending on the mechanism files, some may use different units altogether, but this should be taken care of internally by NEURON.

3.7.4 Contributors

LFPy was developed by (as per commit):

- Henrik Lindén https://lindenh.wordpress.com
- Espen Hagen http://www.mn.uio.no/fysikk/english/?vrtx=person-view&uid=espehage
- Szymon Łęski
- Torbjørn V. Ness
- Solveig Næss
- Alessio Buccino
- Eivind Norheim
- Klas H. Pettersen http://www.med.uio.no/imb/english/?vrtx=person-view&uid=klashp
- Gaute T. Einevoll https://www.nmbu.no/ans/gaute.einevoll

3.7.5 Contact

If you want to contact us with questions, bugs and comments, please create an issue on GitHub.com/LFPy/LFPy/issues. We are of course happy to receive feedback of any kind.

3.8 Module `LFPy`

CHAPTER 4

Indices and tables

- genindex
- modindex
- search