neuronmi

Feb 13, 2020

Contents:

Neuronmi is a Python package to simulate neuronal activity with finite element methods with a high-level and accessible API.

With `neuronmi` one can:

- generate 3D meshes with neurons and extracellular probes
- run simulations with different kind of solvers

CHAPTER 1

Installation

1.1 Using docker (recommended)

We recommend using our docker container which has all the dependencies pre-installed.

You can run the docker image with:

```
docker run mirok/neuronmi
```

1.2 Manual installation

The following are dependencies of *neuronmi* and how they can be obtained.

IMPORTANT: the current version runs on Python 2.7. We are currently working on a Python 3 update.

1. Generating meshes for neuron simulations with EMI models We rely on Gmsh for both mesh generation and geometry definition. All is done via the python API of Gmsh. The gmsh module has to be on python path. For the current shell session this can be accomplished by running

```
export PYTHONPATH=`pwd`:"$PYTHONPATH"
```

in the directory where *gmsh.py* resides (e.g. /usr/local/lib/).

2. Partial differential equation part of EMI The solver requires FEniCS version 2017.2.0. In our experience the simplest way of installation that also plays along nicely with Gmsh is by using the dedicated Ubuntu package.

3. Ordinary differential equation part of EMI Membrane physics is solved for using cbc.beat (which depends on dolfin-adjoint).

1.3 Testing

Run from current directory

```
python -m unittest discover ./test/mesh;
python -m unittest discover ./test/simulators/solver;
```

CHAPTER 2

Overview

`neuronmi` is a Python package with an high-level API to simulate neurons with finite element methods.

It allows users to build meshes with neurons and recording devices, and to simulate the neuronal activity with different models:

- 3D-3D EMI formulation
- 3D-1D EMI formulation
- *hybrid* solution (in progress)

The EMI model (Extracellular-Membrane-Intracellular) is the most advanced of these models, as it explicitly represents the intracellular and extracellular spaces, and the neuronal membrane. This formulation enables users to simulate and study complex phenomena, including ephaptic effects between neurons and the effect of neural devices on the recorded signals.

CHAPTER 3

Generating a mesh

The `mesh` module provides functions and utilities to ease the creation of meshes with neurons and neural devices. The mesh is generated using Gmsh as backend.

The user can create a mesh with the `generate_mesh()` function.

```
mesh_folder = neuronmi.generate_mesh(neurons='bas', probe='microwire', mesh_
↪resolution=3,
                                     box_size=3)
```

This snippet of code will generate a mesh with a ball-and-stick neuron (bas) and a microwire in the extracellular space. The `mesh_resolution` controls the resolution of the mesh (0 - fine resolution, 5 - coarse recolution). The `box_size` controls the size of the bounding box.

There are two kinds of neurons and three kinds of probes built-in.

Neurons:

- `'bas'` : ball-and-stick neuron
- `'tapered'` : similar to a ball-and-stick, but the connection between the soma and the dendrite/axon is tapered

Probes:

- `'microwire'` : cylindrical probe sampling at its tip
- `'neuronexus'` : Multi-Electrode Array from Neuronexus Technologies (A1x32-Poly3-5mm-25s-177-CM32)
- `'neuropixels'` : Multi-electrode Array of Neuropixels technology

In order to retrieve the default parameters of a neuron or a probe, one can run:

```
neuron_params = neuronmi.get_neuron_params('bas')
probe_params = neuronmi.get_probe_params('neuropixels')
```

Once the parameters are retrieved, they can be modified and used in the generate mesh function. In this example, the position of the probe is modified.

```
probe_params = neuronmi.get_probe_params('microwire')
probe_params['tip_x] = 30

mesh_folder = neuronmi.generate_mesh(neurons='bas', probe='microwire', mesh_
↪resolution=3,
                                     box_size=3, probe_params=probe_params)
```

While there can be at most one probe in the mesh, there can be multiple neurons. In order to simulate more than one neuron, the user can use a list a the `neurons` parameter. In this case, the `neuron_params` must also be a list:

```
neuron_params_1 = neuronmi.get_neuron_params('bas')
neuron_params_2 = neuronmi.get_neuron_params('bas')

# Displace two neurons
neuron_params_1['soma_y'] = -20
neuron_params_2['soma_y'] = 20

mesh_folder = neuronmi.generate_mesh(neurons=['bas', 'bas'], probe='microwire', mesh_
↪resolution=3,
                                     box_size=3, neuron_params=[neuron_params_1,␣
↪neuron_params_2])
```

Finally, one can also instantiate neurons and probes outside and pass them to the `generate_mesh()` function:

```
neuron = neuronmi.mesh.shapes.TaperedNeuron({'dend_len': 400, 'axon_len': 200})
microwire_probe = neuronmi.mesh.shapes.MicrowireProbe({'tip_x': 30})
mesh_folder = neuronmi.generate_mesh(neurons=neuron, probe=microwire_probe,
                                     mesh_resolution=3, box_size=3)
```

CHAPTER 4

Running simulations

The `simulators` module provides solvers for the neuronal activity.

Given a previously simulated mesh in `mesh_folder`, the user can set some parameters of the simulator and run the simulation in a few lines.

To run a simulation with default parameters, one can simply run:

```
_ = neuronmi.simulate_emi(mesh_folder)
```

4.1 Changing simulation parameters

TODO

4.2 Other solvers

TODO

CHAPTER 5

Indices and tables

- genindex
- modindex
- search