Dendrify

Release 1.0.9

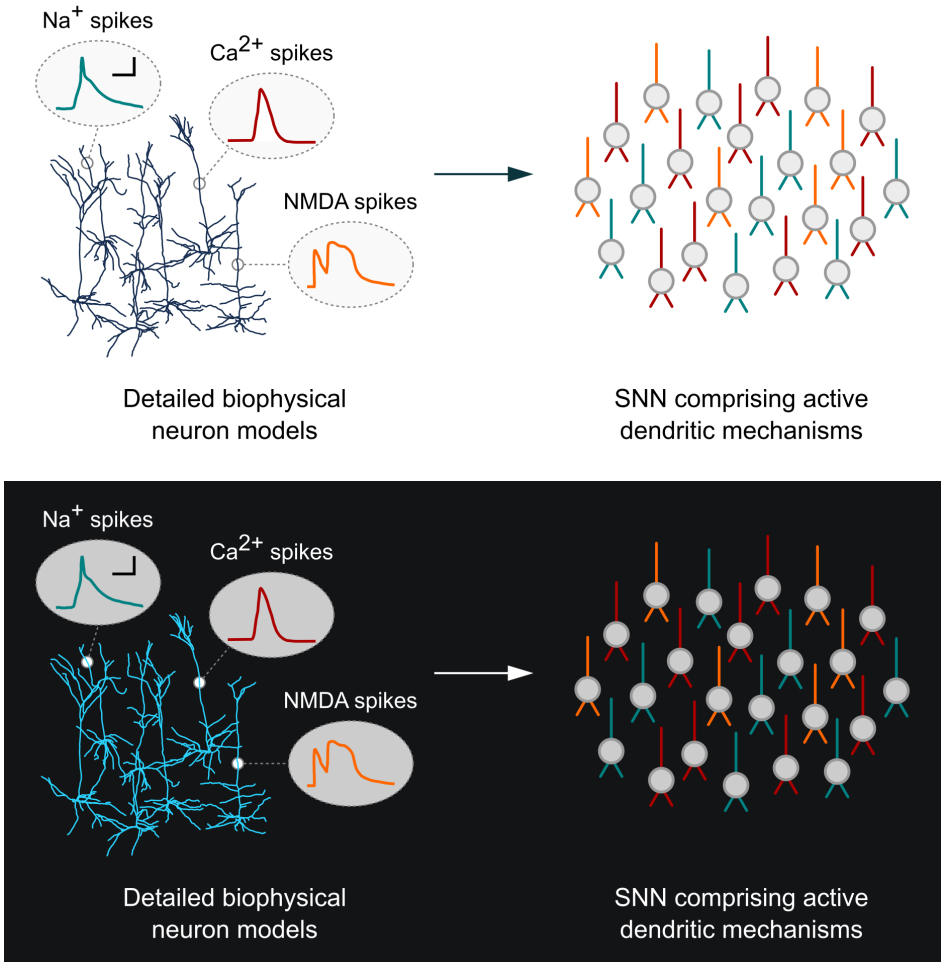
Michalis Pagkalos

Jul 07, 2023

GETTING STARTED

Although neuronal dendrites greatly influence how single neurons process incoming information, their role in network-level functions remain largely unexplored. Current SNNs are usually quite simplistic, overlooking essential dendritic properties. Conversely, circuit models with morphologically detailed neuron models are computationally costly, thus impractical for large-network simulations.

To bridge the gap between these two, we introduce Dendrify, a free, open-source Python package compatible with the Brian 2 simulator. Dendrify, through simple commands, automatically generates reduced compartmental neuron models with simplified yet biologically relevant dendritic and synaptic integrative properties. Such models strike a good balance between flexibility, performance, and biological accuracy, allowing us to explore dendritic contributions to network-level functions.



Tip: If you use Dendrify for your published research, we kindly ask you to cite our article: **Introducing the Dendrify framework for incorporating dendrites to spiking neural networks** M Pagkalos, S Chavlis, P Poirazi DOI: https://doi.org/10.1038/s41467-022-35747-8

CONTENTS:

CHAPTER

ONE

INSTALLATION

Dendrify is included in the Python package index: https://pypi.org/project/dendrify. The easiest way to install it is through `pip`, using the command:

```
pip install dendrify
```

1.1 Dependencies

- Brian 2 (required) is a simulator for spiking neural networks. It is written in Python and is available on almost all platforms. Brian is designed to be easy to learn and use, highly flexible and easily extensible.
  - How to install Brian 2
- Netwokx (optional) is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. If you wish Dendrify to have access to certain experimental model visualization features, you can install it using the command:

  ```
  pip install networkx
  ```

1.2 GPU support

Dendrify is compatible with Brian2CUDA, a Python package for simulating spiking neural networks on graphics processing units (GPUs). Brian2CUDA is an extension of Brian2 that uses the code generation system of the latter to generate simulation code in C++/CUDA, which is then executed on NVIDIA GPUs.

- How to install Brian2CUDA

CHAPTER

TWO

TUTORIAL

Coming soon

CHAPTER

THREE

EXAMPLES

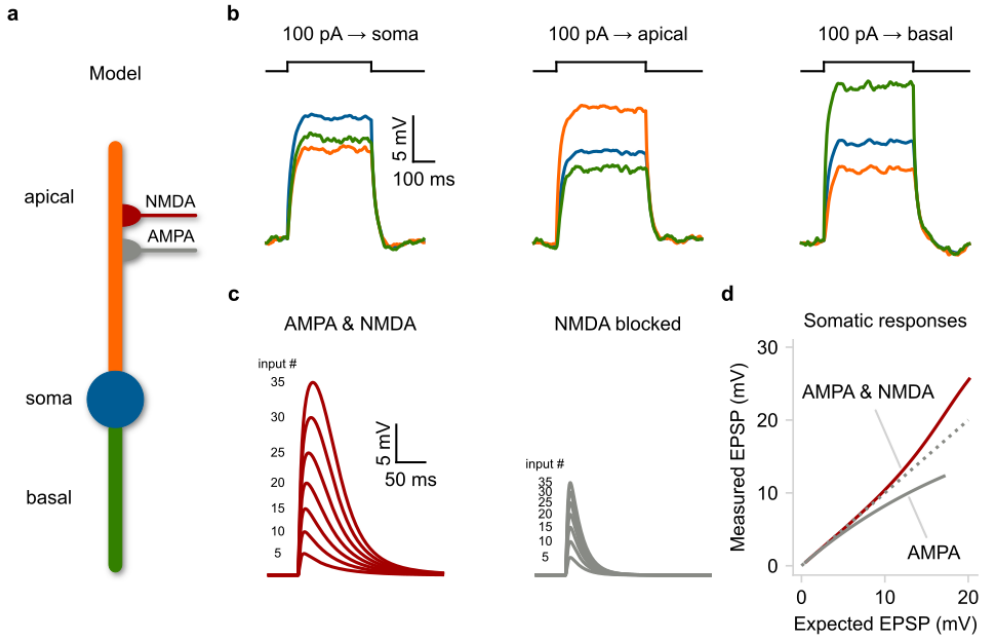
Bellow you will find two model examples adopted from the Dendrify paper.

- *Example 1 | A basic compartmental model with passive dendrites*
- *Example 2 | A reduced compartmental model capturing active dendritic properties*

Tip: By clicking the "**Open in Colab**" button located under each example, you can run in your browser (without locally installing Dendrify or Brian) an interactive Jupyter notebook that reproduces the respective neuron models and simulation results.

Example 1 | A basic compartmental model with passive dendrites.

In this example we show that even rudimentary models can reproduce essentia neuronal properties such as the electrical segmentation caused by dendrites This allows multiple integration sites to coexist within a neuron and dendrite to operate semi-autonomously from the soma, while greatly affecting neuronal output.

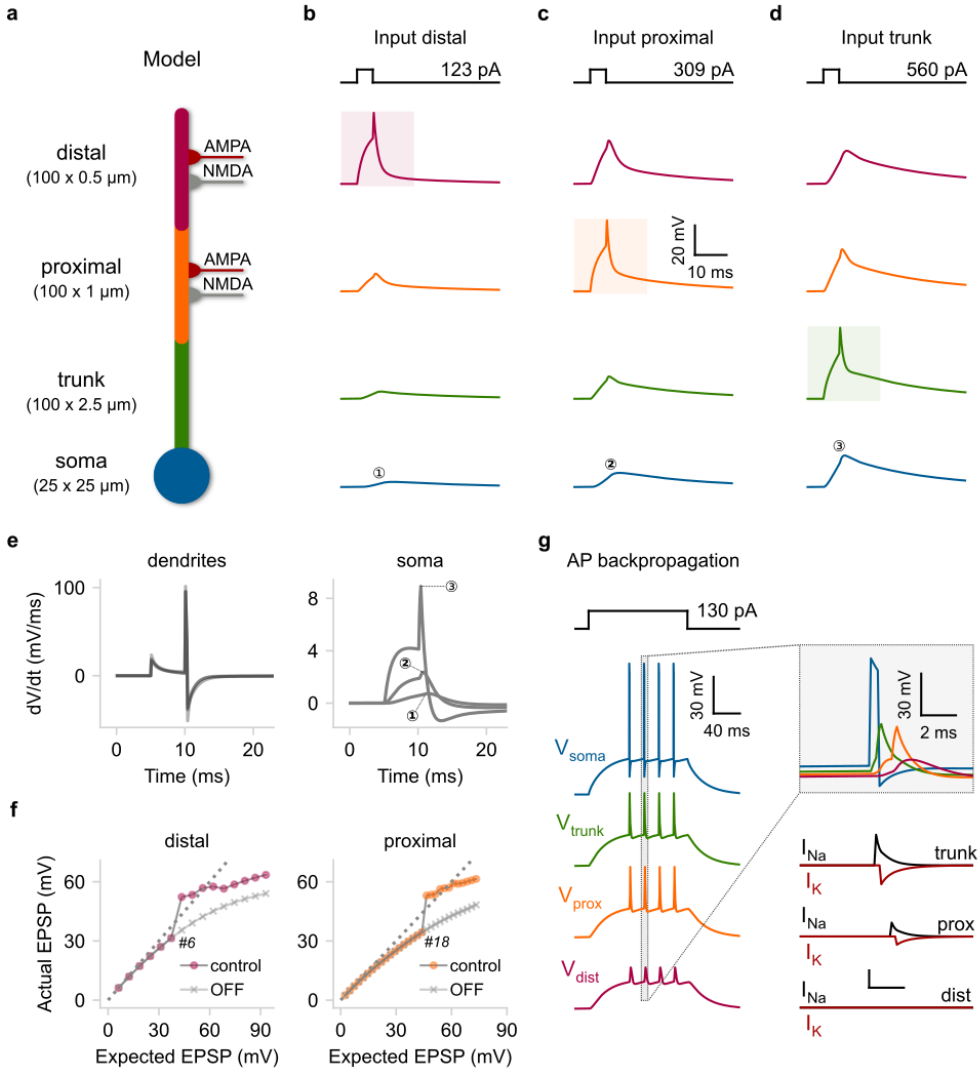


a) Schematic illustration of a compartmental model consisting of a soma (spiking unit) and two dendrites (passive integrators). The apical dendrite can integrate excitatory synapses comprising AMPA and NMDA currents. **b)** Membrane voltage responses to current injections of the same amplitude are applied individually to each compartment. Notice the electrical segregation caused by the resistance between the three neuronal compartments. **c** Somatic responses to a

varying number of simultaneous synaptic inputs (5–35 synapses). *Left*: control EPSPs, *Right*: EPSPs in the presence of NMDA blockers. **d)** Input-output function of the apical dendrite as recorded at the soma. The dotted line represents a linear function. Notice the shift from supralinear to the sublinear mode when NMDARs are blocked.

Example 2: A reduced compartmental model capturing active dendritic properties.

In this example we show that reduced compartmental I&F models, equipped with event-driven dendritic spiking mechanisms can faithfully reproduce a broad range of dendritic properties such as: i) Supralinear input integration, ii) dendrite-specific spiking threshold, iii) distance-dependent filtering, iv) backpropagation of somatic spikes.



a) Schematic illustration of a compartmental model consisting of a soma (leaky I&F) and three dendritic segments (trunk, proximal, distal) equipped with Na+ VGICs. The distal and proximal segments can also receive AMPA and NMDA synapses. **b–d)** Rheobase current injections (5ms square pulses) for dSpike generation were applied individually to each dendritic segment. *Shaded areas*: location of current injection and dSpike initiation. *Top*: stimulation protocol showing the current threshold for a single dSpike (rheobase current). **e)** First temporal derivative of dendritic (left) and somatic (right) voltage traces from panels (**b–d**). **f)** Input–output function of the distal (left) and proximal (right) segment as recorded from the corresponding dendritic locations. We also indicate the number of quasi-simultaneously activated synapses (ISI=0.1ms) needed to elicit a single dSpike in each case. *OFF*: deactivation

of Na+ dSpikes. *Dashed lines*: linear input–output relationship. **g)** *Left*: Backpropagating dSpikes are generated in response to somatic current injections. The short-amplitude spikelets detected in the distal branch are subthreshold voltage responses for dSpike initiation. *Right*: Magnified and superimposed voltage traces (top) from the dashed box (left). *Bottom*: dendritic voltage-activated currents responsible for dSpikes generation in each dendritic segment.

CHAPTER

FOUR

CLASSES

4.1 Compartment

class dendrify.compartment.**Compartment**(*name*, *model='passive'*, ***kwargs*)

Bases: `object`

A class that automatically generates and handles all differential equations and parameters needed to describe a single compartment and any currents (synaptic, dendritic, noise) passing through it.

Parameters

- **name** (*str*) – A unique name used to tag compartment-specific equations and parameters. It is also used to distinguish the various compartments belonging to the same *NeuronModel*.
- **model** (*str, optional*) – A keyword for accessing Dendrify's library models. Custom models can also be provided but they should be in the same formattable structure as the library models. Available options: `'passive'` (default), `'adaptiveIF'`, `'leakyIF'`, `'adex'`.
- **kwargs** (Quantity, optional) – Kwargs are used to specify important electrophysiological properties, such as the specific capacitance or resistance. For more information see: *EphysProperties*.

Examples

```
>>> # specifying equations only:
>>> compX = Compartment('nameX', 'leakyIF')
>>> # specifying equations and ephys properties:
>>> compY = Compartment('nameY', 'adaptiveIF', length=100*um, diameter=1*um,
>>>                     cm=1*uF/(cm**2), gl=50*uS/(cm**2))
```

Attributes:

area	A compartment's surface area (open cylinder) based on its length and diameter.
capacitance	A compartment's absolute capacitance based on its specific capacitance (cm) and surface area.
equations	All differential equations that have been generated for a single compartment.
g_leakage	A compartment's absolute leakage conductance based on its specific leakage conductance (gl) and surface area.
parameters	All parameters that have been generated for a single compartment.

Methods:

connect	Allows the connection (electrical coupling) of two compartments.
noise	Adds a stochastic noise current.
synapse	Adds synaptic currents equations and parameters.

property area

A compartment's surface area (open cylinder) based on its length and diameter.

Return type
Quantity

property capacitance

A compartment's absolute capacitance based on its specific capacitance (cm) and surface area.

Return type
Quantity

connect(*other*, *g='half_cylinders'*)

Allows the connection (electrical coupling) of two compartments.

Parameters

- **other** (Compartment) – Another compartment.
- **g** (str or Quantity, optional) – The coupling conductance. It can be set explicitly or calculated automatically (provided all necessary parameters exist). Available options: `'half_cylinders'` (default), `'cylinder_<compartment name>'`.

> **Warning:** The automatic approaches require that both compartments to be connected have specified **length**, **diameter** and **axial resistance**.

Examples

```
>>> compX, compY = Compartment('x', **kwargs), Compartment('y', **kwargs)
>>> # explicit approach:
>>> compX.connect(compY, g=10*nS)
>>> # half cylinders (default):
>>> compX.connect(compY)
>>> # cylinder of one compartment:
>>> compX.connect(compY, g='cylinder_x')
```

property equations

All differential equations that have been generated for a single compartment.

Return type
str

property g_leakage

A compartment's absolute leakage conductance based on its specific leakage conductance (gl) and surface area.

Return type
Quantity

noise(*tau=20. * msecond*, *sigma=3. * pamp*, *mean=0. * amp*)

Adds a stochastic noise current. For more information see the Noise section: of Models and neuron groups

Parameters

- **tau** (Quantity, optional) – Time constant of the Gaussian noise, by default `20*ms`
- **sigma** (Quantity, optional) – Standard deviation of the Gaussian noise, by default `3*pA`
- **mean** (Quantity, optional) – Mean of the Gaussian noise, by default `0*pA`

property parameters

All parameters that have been generated for a single compartment.

Return type
dict

synapse(*channel=None*, *pre=None*, *g=None*, *t_rise=None*, *t_decay=None*, *scale_g=False*)

Adds synaptic currents equations and parameters. When only the decay time constant `t_decay` is provided, the synaptic model assumes an instantaneous rise of the synaptic conductance followed by an exponential decay. When both the rise `t_rise` and decay `t_decay` constants are provided, synapses are modelled as a sum of two exponentials. For more information see: Modeling Synapses by Arnd Roth & Mark C. W. van Rossum

Parameters

- **channel** (*str*) – Synaptic channel type. Available options: `'AMPA'`, `'NMDA'`, `'GABA'`, by default `None`
- **pre** (*str*) – A unique name to distinguish synapses of the same type coming from different input sources, by default `None`
- **g** (Quantity) – Maximum synaptic conductance, by default `None`
- **t_rise** (Quantity) – Rise time constant, by default `None`
- **t_decay** (Quantity) – Decay time constant, by default `None`

- **scale_g** (*bool, optional*) – Option to add a normalization factor to scale the maximum conductance at 1 when synapses are modelled as a difference of exponentials (have both rise and decay kinetics), by default `False`.

Examples

```
>>> comp = Compartment('comp')
>>> # adding an AMPA synapse with instant rise & exponential decay:
>>> comp.synapse('AMPA', g=1*nS, t_decay=5*ms, pre='X')
>>> # same channel, different conductance & source:
>>> comp.synapse('AMPA', g=2*nS, t_decay=5*ms, pre='Y')
>>> # different channel with both rise & decay kinetics:
>>> comp.synapse('NMDA', g=1*nS, t_rise=5*ms, t_decay=50*ms, pre='X')
```

4.2 Soma

class dendrify.compartment.**Soma**(*name*, *model='leakyIF'*, ***kwargs*)

Bases: *Compartment*

A class that automatically generates and handles all differential equations and parameters needed to describe a somatic compartment and any currents (synaptic, dendritic, noise) passing through it.

See also:

Soma acts as a wrapper for Compartment with slight changes to account for certain somatic properties. For a full list of its methods and attributes, please see: *Compartment*.

Parameters

- **name** (*str*) – A unique name used to tag compartment-specific equations and parameters. It is also used to distinguish the various compartments belonging to the same *NeuronModel*.
- **model** (*str, optional*) – A keyword for accessing Dendrify's library models. Custom models can also be provided but they should be in the same formattable structure as the library models. Available options: `'leakyIF'` (default), `'adaptiveIF'`, `'adex'`.
- **kwargs** (Quantity, optional) – Kwargs are used to specify important electrophysiological properties, such as the specific capacitance or resistance. For more information see: *EphysProperties*.

Examples

```
>>> # specifying equations only:
>>> somaX = Soma('nameX', 'leakyIF')
>>> # specifying equations and ephys properties:
>>> somaY = Soma('nameY', 'adaptiveIF', length=100*um, diameter=1*um,
>>>              cm=1*uF/(cm**2), gl=50*uS/(cm**2))
```

4.3 Dendrite

class dendrify.compartment.**Dendrite**(*name*, *model='passive'*, ***kwargs*)

Bases: *Compartment*

A class that automatically generates and handles all differential equations and parameters needed to describe a dendritic compartment, its active mechanisms, and any currents (synaptic, dendritic, ionic, noise) passing through it.

See also:

Dendrite inherits all the methods and attributes of its parent class *Compartment*. For a complete list, please refer to the documentation of the latter.

Parameters

- **name** (*str*) – A unique name used to tag compartment-specific equations and parameters. It is also used to distinguish the various compartments belonging to the same *NeuronModel*.
- **model** (*str, optional*) – A keyword for accessing Dendrify's library models. Dendritic compartments are by default set to `'passive'`.

Methods:

dspikes	Adds the mechanisms and parameters needed for dendritic spiking.

Attributes:

event_actions	A string that is used to tell Brian how to handle the dSpike events.
events	A dictionary of all dSpike events created for a single dendrite.

dspikes(*channel*, *threshold=None*, *g_rise=None*, *g_fall=None*)

Adds the mechanisms and parameters needed for dendritic spiking. Under the hood, this method creates all equations, conditions and actions to utilize Brian's custom events functionality. Spikes are generated through the sequential activation of a positive (sodium or calcium-like) and a negative current (potassium-like current) when a specified dSpike threshold is crossed.

Hint: The dendritic spiking mechanism as implemented here has three distinct phases.

INACTIVE PHASE:

When the dendritic voltage is subthreshold OR the simulation step is within the refractory period. dSpikes cannot be generated during this phase.

DEPOLARIZATION PHASE:

When the dendritic voltage crosses the dSpike threshold AND the refractory period has elapsed. This triggers the instant activation of a positive current that enters the dendrite and then decays exponentially.

REPOLARIZATION PHASE:

This phase starts automatically after a specified delay from the initiation of the dSpike. A negative current is activated instantly and then decays exponentially. Also a new refractory period begins.

Parameters

- **channel** (*str*) – Ion channel type. Available options: `'Na'`, `'Ca'` (coming soon).
- **threshold** (`Quantity`) – The membrane voltage threshold for dendritic spiking, by default `None`.
- **g_rise** (`Quantity`) – The conductance of the current that is activated during the depolarization phase, by default `None`.
- **g_fall** (`Quantity`) – The conductance of the current that is activated during the repolarization phase, by default `None`.

property event_actions

A string that is used to tell Brian how to handle the dSpike events.

Returns
Executable code that runs automatically in the background.

Return type
str

property events

A dictionary of all dSpike events created for a single dendrite.

Returns
Keys: event names, values: events conditions.

Return type
dict

4.4 NeuronModel

class dendrify.neuronmodel.**NeuronModel**(*connections*, ***kwargs*)

Bases: `object`

Creates a multicompartmental neuron model by connecting individual compartments and merging their equations, parameters and custom events.This model can then be used for creating a population of neurons through Brian's NeuronGroup. This class also contains useful methods for managing model properties and for automating the initialization of custom events and simulation parameters.

Tip: Dendrify aims to facilitate the development of reduced, **few-compartmental** I&F models that help us study how key dendritic properties may affect network-level functions. It is not designed to substitute morphologically and biophysically detailed neuron models, commonly used for highly-accurate, single-cell simulations. If you are interested in the latter category of models, please see Brian's SpatialNeuron.

Parameters

- **connections** (*list[tuple[Compartment, Compartment, str | Quantity]]*) – A description of how the various compartments belonging to the same neuron model should be connected.

- **kwargs** (Quantity, optional) – Kwargs are used to specify important electrophysiological properties, such as the specific capacitance or resistance. For all available options see: *EphysProperties*.

Warning: Parameters set here affect all model compartments and can override any compartment-specific parameters.

Example

```
>>> # Valid format: [*(x, y, z)], where
>>> # x -> Soma or Dendrite object
>>> # y -> Soma or Dendrite object other than x
>>> # z -> 'half_cylinders' or 'cylinder_ + name' or brian2.nS unit
>>> #      (by default 'half_cylinders')
>>> soma = Soma('s', ...)
>>> prox = Dendrite('p', ...)
>>> dist = Dendrite('d', ...)
>>> connections = [(soma, prox, 15*nS), (prox, dist, 10*nS)]
>>> model = NeuronModel(connections)
```

Methods:

add_equations	Allows adding custom equations.
add_params	Allows specifying extra/custom parameters.
as_graph	Plots a graph-like representation of a NeuronModel using the Graph class and the Fruchterman-Reingold force-directed algorithm from Networkx.
dspike_properties	Allows specifying essential dSpike properties affecting all compartments.
link	Links a NeuronModel to a NeuronGroup.

Attributes:

equations	Merges all compartments' equations into a single string.
event_actions	Creates a list of all event actions for dendritic spiking.
events	Organizes all custom events for dendritic spiking into a dictionary.
parameters	Merges all compartments' parameters into a dictionary.

add_equations(*eqs*)

Allows adding custom equations.

Parameters

eqs (*str*) – A string of Brian-compatible equations.

add_params(*params_dict*)

Allows specifying extra/custom parameters.

Parameters
params_dict (*dict*) – A dictionary of parameters.

as_graph(*fontsize=10*, *fontcolor='white'*, *scale_nodes=1*, *color_soma='#4C6C92'*, *color_dendrites='#A7361C'*, *alpha=1*, *scale_edges=1*, *seed=None*)

Plots a graph-like representation of a NeuronModel using the Graph class and the Fruchterman-Reingold force-directed algorithm from Networkx.

Parameters

- **fontsize** (*int, optional*) – The size in pt of each node's name, by default 10.
- **fontcolor** (*str, optional*) – The color of each node's name, by default 'white'.
- **scale_nodes** (*float, optional*) – Percentage change in node size, by default 1.
- **color_soma** (*str, optional*) – Somatic node color, by default '#4C6C92'.
- **color_dendrites** (*str, optional*) – Dendritic nodes color, by default '#A7361C'.
- **alpha** (*float, optional*) – Nodes color opacity, by default 1.
- **scale_edges** (*float, optional*) – The percentage change in edges length, by default 1.
- **seed** (*int, optional*) – Set the random state for deterministic node layouts, by default None. .

dspike_properties(*channel=None*, *tau_rise=None*, *tau_fall=None*, *offset_fall=None*, *refractory=None*)

Allows specifying essential dSpike properties affecting all compartments.

Parameters

- **channel** (*str*) – Ion channel type. Available options: 'Na', 'Ca' (coming soon).
- **tau_rise** (Quantity) – The decay time constant of the current causing the dSpike's **depolarization** phase, by default None.
- **tau_fall** (Quantity) – The decay time constant of the current causing the dSpike's **repolarization** phase, by default None.
- **offset_fall** (Quantity) – The delay for starting the dSpike repolarization phase, by default None.
- **refractory** (Quantity) – The duration of the dSpike inactive period, by default None.

property equations

Merges all compartments' equations into a single string.

Returns
All model equations.

Return type
str

property event_actions

Creates a list of all event actions for dendritic spiking.

Returns
All event actions for dendritic spiking

Return type
list

property events

Organizes all custom events for dendritic spiking into a dictionary.

Returns
All model custom events for dendritic spiking.

Return type
dict

link(*ng*, *automate='all'*, *verbose=False*)

Links a NeuronModel to a NeuronGroup. This allows dendrify to automatically handle the initialization of important simulation parameters.

Parameters

- **ng** (*brian2.NeuronGroup*) – A NeuronGroup that was created using a NeuronModel.
- **automate** (*str, optional*) – What to automate. Available options: `'all'` (default), `'v_rest'`, `'events'`.
- **verbose** (*bool, optional*) – If `True` it prints all the code that was created and run in the background by dendrify, by default `False`

property parameters

Merges all compartments' parameters into a dictionary.

Returns
All model parameters.

Return type
dict

4.5 EphysProperties

class dendrify.ephysproperties.**EphysProperties**(*name=None*, *length=None*, *diameter=None*, *cm=None*, *gl=None*, *r_axial=None*, *v_rest=None*, *scale_factor=1.0*, *spine_factor=1.0*)

Bases: `object`

A class for calculating various important electrophysiological properties for a single compartment.

Note: An EphysProperties object is automatically created and linked to a *Compartment*, *Soma*, or *Dendrite* object during the instantiation of the latter.

Parameters

- **name** (*str, optional*) – A compartment's name, by default `None`
- **length** (*Quantity, optional*) – A compartment's length, by default `None`
- **diameter** (*Quantity, optional*) – A compartment's diameter, by default `None`
- **cm** (*Quantity, optional*) – Specific capacitance (usually F / cm^2), by default `None`
- **gl** (*Quantity, optional*) – Specific leakage conductance (usually S / cm^2), by default `None`
- **r_axial** (*Quantity, optional*) – Axial resistance (usually Ohm * cm), by default `None`

- **v_rest** (*Quantity, optional*) – Resting membrane voltage, by default None
- **scale_factor** (*float, optional*) – A global area scale factor, by default 1.0
- **spine_factor** (*float, optional*) – A dendritic area scale factor to account for spines, by default 1.0

Attributes:

area	A compartment's surface area (open cylinder) based on its length and diameter.
capacitance	A compartment's absolute capacitance based on its specific capacitance (cm) and surface area.
g_cylinder	The conductance (of coupling currents) passing through a cylindrical compartment based on its dimensions and its axial resistance.
g_leakage	A compartment's absolute leakage conductance based on its specific leakage conductance (gl) and surface area.
parameters	Returns a dictionary of all electrophysiological parameters.
total_area_factor	The total surface are factor.

Methods:

g_couple	The conductance (of coupling currents) between the centers of two adjacent cylindrical compartments, based on their dimensions and the axial resistance.

property area

A compartment's surface area (open cylinder) based on its length and diameter.

Returns
A compartment's surface area

Return type
Quantity

property capacitance

A compartment's absolute capacitance based on its specific capacitance (cm) and surface area.

Return type
Quantity

static g_couple(*comp1*, *comp2*)

The conductance (of coupling currents) between the centers of two adjacent cylindrical compartments, based on their dimensions and the axial resistance.

Parameters

- **comp1** (EphysProperties) – An EphysProperties object
- **comp2** (EphysProperties) – An EphysProperties object

Return type
Quantity

property g_cylinder

The conductance (of coupling currents) passing through a cylindrical compartment based on its dimensions and its axial resistance. To be used when then the total number of compartments is low and the adjacent-to-soma compartments are highly coupled with the soma.

Return type
Quantity

property g_leakage

A compartment's absolute leakage conductance based on its specific leakage conductance (gl) and surface area.

Return type
Quantity

property parameters

Returns a dictionary of all electrophysiological parameters.

Return type
dict

property total_area_factor

The total surface are factor.

Return type
float

CHAPTER

FIVE

MODEL LIBRARY

Note: Dendrify relies on Brian's Equations-based approach to define models as systems of first order ordinary differential equations. For convenience, Dendrify includes a library of default models (see below) however users can also provide custom model equations.

5.1 Somatic models[1][2]

5.1.1 Leaky Integrate-and-Fire

$$C\frac{dV}{dt} = -g_L(V - E_L) + I$$

where C is the membrane capacitance, V the membrane potential, g_L the leak conductance, E_L the leak reversal potential and I is the input current. When the firing threshold V_θ is crossed, V resets to a fixed value V_r.

5.1.2 Adaptive Integrate-and-Fire

$$C\frac{dV}{dt} = -g_L(V - E_L) - w + I$$

$$\tau_w\frac{dw}{dt} = a(V - E_L) - w$$

where w is the adaptation variable, a the adaptation coupling parameter and $_w$ is the adaptation time constant. When the firing threshold V_θ is crossed, V resets to a fixed value V_r and $w \rightarrow w + b$, where b is the spike-triggered adaptation current.

[1] https://neuronaldynamics.epfl.ch/online/Ch1.S3.html
[2] https://neuronaldynamics.epfl.ch/online/Ch6.S1.html

5.1.3 Adaptive Exponential Integrate-and-Fire

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right) - w + I$$

$$\tau_w \frac{dw}{dt} = a(V - E_L) - w$$

where Δ_T is the slope factor and V_T the voltage threshold. When the firing threshold V_θ is crossed, V resets to a fixed value V_r and $w \rightarrow w + b$, where b is the spike-triggered adaptation current.

5.2 Dendritic models

5.3 Synaptic models[Page 24, 34]

5.3.1 AMPA

$$I_{\text{AMPA}} = \bar{g}_{\text{AMPA}}(E_{\text{AMPA}} - V)s(t)$$

$$\frac{ds}{dt} = \frac{-s}{\tau^{\text{decay}}_{\text{AMPA}}}$$

where $\bar{g}_{\text{AMPA}}$ is the AMPA synaptic conductance, s the channel state variable, E_{AMPA} the AMPA reversal potential, V the membrane potential and $\tau^{\text{decay}}_{\text{AMPA}}$ the AMPA decay time constant. When a pre-synaptic spike arrives $s \rightarrow s + 1$.

5.3.2 AMPA (rise & decay)

$$I_{\text{AMPA}} = \bar{g}_{\text{AMPA}}(E_{\text{AMPA}} - V)x(t)$$

$$\frac{dx}{dt} = \frac{-x}{\tau^{\text{decay}}_{\text{AMPA}}} + s(t)$$

$$\frac{ds}{dt} = \frac{-s}{\tau^{\text{rise}}_{\text{AMPA}}}$$

where s and x describe the rise and decay kinetics of the channel respectively, $\tau^{\text{rise}}_{\text{AMPA}}$ is the AMPA rise time constant and $\tau^{\text{decay}}_{\text{AMPA}}$ is the AMPA decay time constant. When a pre-synaptic spike arrives $s \rightarrow s + 1$.

5.3.3 NMDA

$$I_{\text{NMDA}} = \bar{g}_{\text{NMDA}}(E_{\text{NMDA}} - V)s(t)\sigma(V)$$

$$\frac{ds}{dt} = \frac{-s}{\tau^{\text{decay}}_{\text{NMDA}}}$$

$$\sigma(V) = \frac{1}{1 + \frac{\left[\text{Mg}^{2+}\right]_o}{\beta} \cdot \exp\left(-\alpha\left(V - \gamma\right)\right)}$$

where $\bar{g}_{\text{NMDA}}$ is the NMDA synaptic conductance, s the channel state variable, E_{NMDA} the NMDA reversal potential, $\tau^{\text{decay}}_{\text{NMDA}}$ the NMDA decay time constant, β (mM), α (mV^{-1}) and γ (mV) control the magnesium and voltage dependencies and $[\text{Mg}^{2+}]_{\text{o}}$ denotes the external magnesium concentration (mM). When a pre-synaptic spike arrives $s \rightarrow s + 1$.

[3] https://neuronaldynamics.epfl.ch/online/Ch3.S1.html
[4] https://link.springer.com/chapter/10.1007/978-0-387-87708-2_7#Sec1

5.3.4 References

CHAPTER

SIX

INDEX

CHAPTER

SEVEN

IMPORTANT LITERATURE

Introducing the Dendrify framework for incorporating dendrites to spiking neural networks M Pagkalos, S Chavlis, P Poirazi DOI: https://doi.org/10.1038/s41467-022-35747-8

Brian 2, an intuitive and efficient neural simulator M Stimberg, R Brette, D FM Goodman DOI: https://doi.org/10.7554/eLife.47314

Contribution of sublinear and supralinear dendritic integration to neuronal computations A Tran-Van-Minh, R D Cazé, T Abrahamsson, L Cathala, B Gutkin, D A DiGregorio DOI: https://doi.org/10.3389/fncel.2015.00067

Pyramidal neurons: dendritic structure and synaptic integration N Spruston DOI: https://doi.org/10.1038/nrn2286

Reduced compartmental models of neocortical pyramidal cells P C Bush, T J Sejnowski DOI : https://doi.org/10.1016/0165-0270(93)90151-g

Book | Mathematical Foundations of Neuroscience (chapters 1, 2 & 7) G B Ermentrout, D H Terman Publisher's website: https://link.springer.com/book/10.1007/978-0-387-87708-2

Brian2CUDA: flexible and efficient simulation of spiking neural network models on GPUs D Alevi, M Stimberg, H Sprekeler, K Obermayer, M Augustin DOI: https://doi.org/10.3389/fninf.2022.883700

CHAPTER

EIGHT

RELEASE NOTES

8.1 Version 1.0.8

- Improved documentation.
- Minor improvements.

8.2 Version 1.0.5

- Improved documentation.
- Minor bug fixes.

8.3 Version 1.0.4

- Redesigned documentation page.
- Added more type hints.
- Improved compatibility with older Python versions.
- Minor bug fixes.

CHAPTER

NINE

CODE OF CONDUCT

9.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

9.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

9.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

9.4 Scope

This Code of Conduct applies within all project spaces, and it also applies when an individual is representing the project or its community in public spaces. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

9.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at mpagkalos93@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

9.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

INDEX