Arborize

Release 4.0.0b0

Robin De Schepper

Nov 15, 2023

CONTENTS:

Arborize lets you describe your model definitions, import schematics from multiple sources and build equivalent multicompartmental models on the supported backends (Arbor and NEURON).

CHAPTER

ONE

GETTING STARTED

To build our first model, we start by creating a definition consisting of *soma* and *basal_dendrite*.

We'll begin by defining *hh* and *pas* in the soma, and *pas* in the dendrites:

```
from arborize import define_model

definition = define_model({
  "cable_types": {
    "soma": {
      "cable": {
        "Ra": 100,
        "cm": 1,
      },
      "mechanisms": {
        "hh": {
          "gnabar": 0.12,
          "gkbar": 0.036,
          "gl": 0.0003,
          "el": -54.3,
        },
      },
    },
    "basal_dendrite": {
      "cable": {
        "Ra": 100,
        "cm": 1,
      },
      "mechanisms": {
        "pas": {
          "g": 0.001,
          "e": -65,
        },
      },
    },
  },
})
```

Next up we need to get a schematic, download this morphology from NeuroMorpho as *morpho.swc*, then we can create a file schematic from it:

```
from arborize import file_schematic

schematic = file_schematic("morpho.swc", definition)
```

Hint: Arborize uses MorphIO to load schematics from file. The points are labelled with the `SectionType` enum.

We're ready to build a cell:

```
from arborize import neuron_build

cell = neuron_build(schematic)
```

Hint: Arborize's NEURON builder uses Patch to construct NEURON objects. It provides many convenience functions.

Let's record the soma and plot the results:

```
from patch import p
import plotly.express as px

r = cell.soma[0].record()
cell.basal_dendrite[0].iclamp(delay=5, duration=1, amplitude=0.1, x=1)
t = p.time
p.celsius = 6.3
p.run(100)
px.line(x=list(r), y=list(t)).show()
```

To create variants of your cell model, pass the template definition to `define_model`:

```
variant = define_model(definition, {
  "cable_types": {
    "soma": {
      "mechanisms": {
        "hh": {
          "gkbar": 0.172
        }
      }
    }
  }
})
```

We can now plot the difference between the 2 cell types:

```
from patch import p
import plotly.express as px

wildtype_schematic = file_schematic("morpho.swc", definition)
wildtype_cell = neuron_build(schematic)
variant_schematic = file_schematic("morpho.swc", variant)
variant_cell = neuron_build(variant_schematic)
```

(continues on next page)

(continued from previous page)

```
wildtype_cell.basal_dendrite[0].iclamp(delay=5, duration=1, amplitude=0.1, x=1)
variant_cell.basal_dendrite[0].iclamp(delay=5, duration=1, amplitude=0.1, x=1)
r_wt = wildtype_cell.soma[0].record()
r_var = variant_cell.soma[0].record()
t = p.time
p.celsius = 6.3
p.run(100)
px.line(x=list(r), y=list(t)).show()
```

CHAPTER

TWO

MODEL DEFINITIONS

CHAPTER

THREE

MODEL SCHEMATICS

CHAPTER

FOUR

MODEL BUILDERS

CHAPTER

FIVE

ARBORIZE

5.1 arborize package

5.1.1 Subpackages

arborize.builders package

Module contents

arborize.schematics package

Module contents

5.1.2 Submodules

5.1.3 arborize.definitions module

class arborize.definitions.**Assert**

Bases: `object`

assert_()

class arborize.definitions.**CableProperties**(*Ra: float = None*, *cm: float = None*)

Bases: *Copy*, *Merge*, *Assert*

Ra: **float = None**

cm: **float = None**

Axial resistivity in ohm/cm

copy()

class arborize.definitions.**CableType**

Bases: `object`

add_ion(*key: str*, *ion:* Ion)

add_mech(*mech_id: str | tuple[str] | tuple[str, str] | tuple[str, str, str]*, *mech:* Mechanism)

add_synapse(*label: str | tuple[str] | tuple[str, str] | tuple[str, str, str]*, *synapse:* Synapse)

static anchor(*defs: Iterable[*CableType*]*, *synapses: dict[str,* arborize.definitions.Synapse*]* | *None* = *None*, *use_defaults: bool* = *False*) → *CableType*

assert_()

cable: *CableProperties*

copy()

classmethod default()

ions: dict[str, *arborize.definitions.Ion*]

mechs: dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]], *arborize.definitions.Mechanism*]

merge(*def_right:* CableType)

set(*param:* Parameter)

synapses: dict[str, *arborize.definitions.Synapse*]

class arborize.definitions.**Copy**

Bases: object

copy()

class arborize.definitions.**Ion**(*rev_pot: float* = *None*, *int_con: float* = *None*, *ext_con: float* = *None*)

Bases: *Copy*, *Merge*, *Assert*

ext_con: float = None

int_con: float = None

rev_pot: float = None

class arborize.definitions.**Mechanism**(*parameters*)

Bases: object

copy()

merge(*other*)

parameters: dict[str, float]

class arborize.definitions.**Merge**

Bases: object

merge(*other*)

class arborize.definitions.**ModelDefinition**(*use_defaults=False*)

Bases: object

add_cable_type(*label: str*, *def_:* CableType)

add_synapse_type(*label: str* | *tuple[str]* | *tuple[str, str]* | *tuple[str, str, str]*, *synapse:* Synapse)

copy()

get_cable_types()

get_synapse_types()

class arborize.definitions.**Synapse**(*parameters*, *mech_id: str | tuple[str] | tuple[str, str] | tuple[str, str, str]*)

Bases: *Mechanism*

copy()

mech_id: tuple[str] | tuple[str, str] | tuple[str, str, str]

class arborize.definitions.**default_ions_dict**

Bases: dict

arborize.definitions.**define_model**(*templ_or_def*, *def_dict=None*, */*, *use_defaults=False*)

arborize.definitions.**is_mech_id**(*mech_id*)

class arborize.definitions.**mechdict**

Bases: dict

arborize.definitions.**to_mech_id**(*mech_id: str | tuple[str] | tuple[str, str] | tuple[str, str, str]*) → tuple[str] | tuple[str, str] | tuple[str, str, str]

5.1.4 arborize.exceptions module

exception arborize.exceptions.**ArborizeError**(**args*, ***kwargs*)

Bases: DetailedException

ArborizeError exception

exception arborize.exceptions.**ConstructionError**(**args*, ***kwargs*)

Bases: *SchematicError*

ConstructionError exception

exception arborize.exceptions.**FrozenError**(**args*, ***kwargs*)

Bases: *SchematicError*

FrozenError exception

exception arborize.exceptions.**ModelDefinitionError**(**args*, ***kwargs*)

Bases: *ArborizeError*

ModelDefinitionError exception

exception arborize.exceptions.**ModelError**(**args*, ***kwargs*)

Bases: *ArborizeError*

ModelError exception

exception arborize.exceptions.**SchematicError**(**args*, ***kwargs*)

Bases: *ArborizeError*

SchematicError exception

exception arborize.exceptions.**TransmitterError**(**args*, ***kwargs*)

Bases: *ModelError*

TransmitterError exception

exception arborize.exceptions.**UnknownLocationError**(**args*, ***kwargs*)

Bases: *ModelError*

UnknownLocationError exception

exception arborize.exceptions.**UnknownSynapseError**(**args*, ***kwargs*)

Bases: *ModelError*

UnknownSynapseError exception

5.1.5 arborize.parameter module

class arborize.parameter.**CableParameter**(*prop: str*, *value: float*)

Bases: *Parameter*

set_cable_params(*cable*)

class arborize.parameter.**IonParameter**(*ion: str*, *prop: str*, *value: float*)

Bases: *Parameter*

class arborize.parameter.**MechParameter**

Bases: *Parameter*

class arborize.parameter.**Parameter**

Bases: object

5.1.6 arborize.schematic module

class arborize.schematic.**Branch**

Bases: object

append(*point*)

points: list[*arborize.schematic.Point*]

class arborize.schematic.**CableBranch**

Bases: *Branch*

append(*loc*, *coords*, *radius*, *labels*)

points: list[*arborize.schematic.Point*]

class arborize.schematic.**Point**(*loc*, *branch*, *coords*, *radius*)

Bases: object

branch: *UnitBranch*

class arborize.schematic.**Schematic**(*name=None*)

Bases: object

create_empty()

create_location(*location*, *coords*, *radii*, *labels*, *endpoint=None*)

create_name()

property definition

freeze()

get_cable_types()

get_synapse_types()

property name

set_param(*location: tuple[int, int] | tuple[tuple[int, int], tuple[int, int]] | str*, *param:* Parameter)

class arborize.schematic.**UnitBranch**

Bases: *Branch*

children: list['UnitBranch']

definition: *CableType*

labels: list[str]

parent: *UnitBranch* | None

arborize.schematic.**throw_frozen**()

5.1.7 arborize.synapse module

class arborize.synapse.**Synapse**(*cell*, *section*, *point_process_name*, *attributes={}*, *variant=None*, *type=None*, *source=None*)

Bases: object

presynaptic(*section*, *x=0.5*, ***kwargs*)

record()

stimulate(**args*, ***kwargs*)

5.1.8 Module contents

CHAPTER

SIX

DEFINING A MODEL

Model definitions consist of *cable_types* and *synapse_types*. Cable types define the cable and ionic properties of the types of compartments in the model, such as the soma, axon and dendrites, and which biophysical mechanisms should be present. Synapse types define the properties of the synapses that can be inserted.

```
{
   "cable_types": {...},
   "synapse_types": {...}
}
```

You then pass this dictionary style definition to the *define_model()* function:

```
from arborize import define_model

definition = define_model({
   "cable_types": {...},
   "synapse_types": {...},
})
```

A *definition* can then be combined with a *schematic* by a *builder* to create instances of your model.

6.1 Cable types

Each cable type has a name. A cable type definition consists of *cable*, *ions*, *mechanisms*, and *synapses*. During the schematic consutrction multiple cable types can be applied to a single piece of cable in the model.

```
{
   "cable_types": {
      "soma": {
         "cable": {...},
         "ions": {...},
         "mechanisms": {...},
         "synapses": {...}
      }
   }
}
```

6.1.1 Cable properties

- `Ra`: Axial resistivity (ohm/cm)
- `cm`: Membrane capacitance (uF/cm^2)

```
{
   "cable": {
      "Ra": 0.34,
      "cm": 1,
   }
}
```

6.1.2 Ion properties

The *ions* block can contain any key, representing the ion name, conventionally lowercase, and each can set the following properties:

- `e`: Reversal potential

```
{
   "ions": {
      "h": {"rev_pot": 0},
      "ca": {"rev_pot": 30},
   }
}
```

6.1.3 Mechanisms

Mechanisms are defined by their mechanism ID, which is either a string name, or a tuple of up to 3 strings: `name`, `variant`, and `package`; and a set of parameters.

```
{
   "mechanisms": {
      "Kv1": {
         "gbar": 1.4
      },
      ("Kv1", "burst"): {
         "gbar": 1.4
      }
   }
}
```

Hint: Arborize uses Glia to manage the mechanisms for the NEURON and Arbor backends. In order to use mechanisms you need to add them to your Glia library.

6.1.4 Synapses

A synapse definition is defined by its name, mechanism ID and parameter set. If you do not specify a mechanism ID, the name is used instead.

```
{
   "synapses": {
      "AMPA": {
         "gmax": 3200
      },
      "depressing": {
         "mechanism": ("AMPA", "TM"),
         "parameters": {
            "gmax": 1300,
            "U": 0.60
         }
      },
      "facilitating": {
         "mechanism": ("AMPA", "TM"),
         "parameters": {
            "gmax": 10000,
            "U": 0.12,
            "tau_facil": 1.1
         }
      },
   }
}
```

6.2 Synapse types

The previous section described how to add synapses to a cable type, but you can also define a synapse type available on the entire model by adding them to the *synapse_types* field.

```
{
   "cable_types": {...},
   "synapse_types": {
      "AMPA": {
         "gmax": 3200
      }
   }
}
```

6.3 Drawing a schematic

Schematics can come from any source, and Arborize supports 2 sources out of the box:

- BSB schematics from BSB morphologies and parameters.
- File schematics load morphologies from file using MorphIO.

```
from arborize import file_schematic, define_model, neuron_build

definition = define_model({
   "cable_types": {...},
   "synapse_types": {...},
})
schematic = file_schematic("my_cell.swc", definition)
n_cells = 100
cells = [neuron_build(schematic) for i in range(n_cells)]
```

PYTHON MODULE INDEX

a

INDEX

I

L

M

N

P

R

S

T

U