

---

# neuron morphology

*Release 1.1.7*

Apr 12, 2023



---

## Contents

---

<b>1</b>	<b>Feature Extraction</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Running Feature Extraction from the Command Line . . . . .	1
1.3	Running in Python/Notebooks . . . . .	1
<b>2</b>	<b>IVSCC Spatial Transformation</b>	<b>3</b>
2.1	Components . . . . .	3
2.2	Command-line invocation . . . . .	4
2.3	Putting it all together . . . . .	4
<b>3</b>	<b>Install</b>	<b>5</b>
3.1	requirements . . . . .	5
3.2	managing your Python environment . . . . .	5
3.3	install non pip requirements . . . . .	5
3.4	installing package . . . . .	6
3.5	installing for development . . . . .	6
<b>4</b>	<b>API Reference</b>	<b>7</b>
4.1	neuron_morphology . . . . .	7
	<b>Python Module Index</b>	<b>115</b>
	<b>Index</b>	<b>117</b>



### 1.1 Introduction

Morphological features are useful for investigating and clustering neuron morphologies. The Feature Extractor package is designed to allow flexible morphological feature extraction from swc neuron reconstruction files and supplementary data. The default\_feature set is a combination of [L-measure](#) and other features used by the Allen Institute.

### 1.2 Running Feature Extraction from the Command Line

The feature extractor module is an [argschema module](#), which can be run from the command line:

```
feature_extractor --input_json path_to_inputs.json --output_json write_outputs_here.  
↪ json
```

Please see the [schema file](#) for usage details and options.

### 1.3 Running in Python/Notebooks

You can take advantage of all of the capabilities of Feature Extractor by running it in python and jupyter notebooks. By running in python and notebooks, you can easily add your own features, create different feature sets, and customize your feature extractor to meet your needs.

Here are two basic examples for running IVSCC and fMOST data:

- [IVSCC example notebook](#)
- [fMOST example notebook](#)

For a more detailed look at the feature extractor capabilities, checkout [feature\\_extractor\\_example](#)



---

### IVSCC Spatial Transformation

---

For [feature extraction](#) or visualization, we often need to apply a transformation to the space in which our reconstruction dwells. Some examples:

- **unshrink** : If a neuron is reconstructed from slice, the depth dimension may not scaled equivalently to the width and height dimensions, due to tissue shrinkage. In this case, the neuron must be rescaled along the depth dimension in order for features like compartment volume to be meaningful.
- **upright** : Images of single cortical neurons reconstructed in slice may be rotated arbitrarily. In order to visualize or calculate the symmetry of a neuron’s apical dendrites, we must rotate the neuron so that the piaward direction is “up”.

The `neuron_morphology` repository contains a set of utilities for calculating and applying such transformations. These utilities are ones that we, the Allen Institute, use for processing our in-vitro single cell characterization data (IVSCC, [whitepaper here](#)), but you may also find them handy if your data are similar.

## 2.1 Components

Here are the spatial transform components that we use for our IVSCC data. For each one, we’ve also included a link to the detailed input and output specification for that executable.

- `pia_wm_streamlines` ([schema](#)) : Given 2D linestrings describing the pia and white matter surfaces local to a neuron, calculate a cortical depth field, whose values are the depth between pia and white matter.
- `upright_angle` ([schema](#)) : Given an swc-formatted reconstruction and the outputs of `pia_wm_streamlines`, find the angle of rotation about the soma which will align the “y” axis of the reconstruction with the piaward direction.
- `apply_affine_transform` ([schema](#)) : Given a 3D affine transform and an swc-formatted reconstruction, produce a transformed reconstruction also in swc format.

## 2.2 Command-line invocation

Once you have installed `neuron_morphology`, you can run these utilities from the command line as you would any `argschema` module. Here is an example:

```
pia_wm_streamlines --input_json path_to_inputs.json --output_json write_outputs_here.  
↪ json
```

In this case, the contents of `path_to_inputs.json` might look like:

```
{  
  "pia_path_str": "10.0,1.0,10.0,3.0,9.0,5.0",  
  "wm_path_str": ".0,1.0,1.0,3.0,1.0,4.0"  
}
```

Please see the `schema` file for more details and options.

## 2.3 Putting it all together

Most likely, you would like to run several of these components in sequence. Here is a `jupyter notebook` which demonstrates in depth how to go from a “raw” morphology and cortical boundaries to an upright-transformed morphology.



### 3.1 requirements

We support Python 3.8-3.9 on Linux, OSX, and Windows. Similar Python versions (e.g. 3.7-3.10) will probably work, but we don't regularly test using those versions. Also, the [streamlines] feature set, which is required for doing morphology uprighting requires the *fenics-dolfinx* package which is not easily available for Windows and so we don't recommend using Windows if you want those features of this package.

### 3.2 managing your Python environment

We recommend installing *neuron\_morphology* into a managed Python environment. Having multiple isolated environments lets you install incompatible packages (or different versions of the same package!) simultaneously and prevents unexpected behavior by utilities that rely on the system Python installation.

Two popular tools for managing Python environments are [anaconda](#) and [venv](#). The rest of this document assumes that you have created and activated an environment using one of these tools. Using *anaconda*, this looks like:

```
conda create -y --name environment-name python=3.9
conda activate environment-name
```

and using *venv*:

```
python -m venv path/to/environment
source path/to/environment/bin/activate
```

### 3.3 install non pip requirements

If you want to utilize the streamlines feature set of this package, which is required to do uprighting and layer aligning morphologies with respect to layers in cortex, for example, then we recommend installing some requirements via *conda*.

```
conda install fenics-dolfinx python-gmsh
```

NOTE: these packages are not available on conda for Windows, or some older versions of python.

The rest of the requirements should install fine via pip, or if you don't need these features, you can skip this step.

## 3.4 installing package

We recommend installing *neuron\_morphology* via pip:

```
pip install neuron_morphology
```

## 3.5 installing for development

If you want to work on *neuron\_morphology*, you should first clone the repository, then install it in editable mode so that you can easily test your changes:

First do the conda environment and requirements installation described above.

```
git clone https://github.com/alleninstitute/neuron_morphology
cd neuron_morphology
pip install -e .
```

This page contains auto-generated API reference documentation<sup>1</sup>.

### 4.1 neuron\_morphology

Top-level package for neuron\_morphology.

#### 4.1.1 Subpackages

`neuron_morphology.feature_extractor`

##### Submodules

`neuron_morphology.feature_extractor.__main__`

##### Module Contents

##### Functions

---

<sup>1</sup> Created with sphinx-autoapi

---

<code>extract_multiple</code> (reconstructions: List[Dict[str, Any]], feature_set: str, heavy_output_path: str, required_marks: Optional[List[str]] = None, only_marks: Optional[List[str]] = None, num_processes: Optional[int] = None, global_parameters: Optional[Dict[str, Any]] = None, output_table_path: Optional[str] = None)	For each path in swc_paths, load the file into a morphology and (attempt
---	--

---

```
main()
```

---

```
neuron_morphology.feature_extractor.__main__.extract_multiple(reconstructions:
    List[Dict[str,
    Any]],          fea-
    ture_set:      str,
    heavy_output_path:
    str,          re-
    quired_marks:
    Op-
    tional[List[str]]
    =            None,
    only_marks: Op-
    tional[List[str]]
    =            None,
    num_processes:
    Optional[int]
    =            None,
    global_parameters:
    Optional[Dict[str,
    Any]]          =
    None,          out-
    put_table_path:
    Optional[str]  =
    None)
```

For each path in swc\_paths, load the file into a morphology and (attempt to) extract each feature in the set specified by feature\_set.

Because of how Windows handles multiprocessing, run\_feature\_extraction must be in another py file.

### Parameters

**reconstructions** [specify the reconstructions on which to compute features]  
**feature\_set** [names the set of features for which calculation will be] attempted  
**heavy\_output\_path** [write “heavy” outputs, such as arrays, to this h5 file]  
**only\_marks** [names marks to which calculation will be restricted]  
**required\_marks** [raise an exception if these named marks fail validation]  
**num\_processes** [use this many cores in the multiprocessing pool.]  
**global\_parameters** [a dictionary specifying cross-reconstruction] parameters  
**output\_table\_path** [if not none, write a flattened table of features here]

### Returns

a dictionary whose keys are reconstruction identifiers and whose values are the outputs of run\_feature\_extraction for those reconstructions.

```
neuron_morphology.feature_extractor.__main__.main()
```

```
neuron_morphology.feature_extractor._schemas
```

## Module Contents

### Classes

<i>ReferenceLayerDepths</i>	mm.Schema class with support for making fields default to
<i>Reconstruction</i>	mm.Schema class with support for making fields default to
<i>GlobalParameters</i>	mm.Schema class with support for making fields default to
<i>InputParameters</i>	The base marshmallow schema used by ArgSchema-Parser to identify
<i>OutputParameters</i>	mm.Schema class with support for making fields default to

### Functions

<i>validate_point_depths_path</i> (path: str)	Check whether a layered point depths path is usable.
---	--

```
neuron_morphology.feature_extractor._schemas.validate_point_depths_path(path:
                                                                    str)
```

Check whether a layered point depths path is usable.

```
class neuron_morphology.feature_extractor._schemas.ReferenceLayerDepths (only=None,
                                                                    ex-
                                                                    clude=(),
                                                                    many=False,
                                                                    con-
                                                                    text=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False,
                                                                    un-
                                                                    known=None)
```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

**key**

**names**

**boundaries**

**classmethod** `is_valid(cls, value)`

```
class neuron_morphology.feature_extractor._schemas.Reconstruction (only=None,  
                                                                exclude=(),  
                                                                many=False,  
                                                                con-  
                                                                text=None,  
                                                                load_only=(),  
                                                                dump_only=(),  
                                                                par-  
                                                                tial=False,  
                                                                un-  
                                                                known=None)
```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

**swc\_path**

**identifier**

**layered\_point\_depths\_path**

```
class neuron_morphology.feature_extractor._schemas.GlobalParameters (only=None,  
                                                                ex-  
                                                                clude=(),  
                                                                many=False,  
                                                                con-  
                                                                text=None,  
                                                                load_only=(),  
                                                                dump_only=(),  
                                                                par-  
                                                                tial=False,  
                                                                un-  
                                                                known=None)
```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

**reference\_layer\_depths**

```
class neuron_morphology.feature_extractor._schemas.InputParameters (only=None,  
                                                                ex-  
                                                                clude=(),  
                                                                many=False,  
                                                                con-  
                                                                text=None,  
                                                                load_only=(),  
                                                                dump_only=(),  
                                                                par-  
                                                                tial=False,  
                                                                un-  
                                                                known=None)
```

Bases: `argschema.schemas.ArgSchema`

The base marshmallow schema used by `ArgSchemaParser` to identify `input_json` and `output_json` files and the `log_level`

**reconstructions**

**heavy\_output\_path**

**feature\_set**

```

only_marks
required_marks
output_table_path
num_processes
global_parameters

```

```

class neuron_morphology.feature_extractor._schemas.OutputParameters (only=None,
                                                                    ex-
                                                                    clude=(),
                                                                    many=False,
                                                                    con-
                                                                    text=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False,
                                                                    un-
                                                                    known=None)

```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

```

inputs
results

```

`neuron_morphology.feature_extractor.data`

## Module Contents

### Classes

---

*Data*

---

### Functions

---

<code>get_morphology(data: MorphologyLike)</code>	Decay a Data to a Morphology, leaving Morphologies untouched
---	--

---

```

class neuron_morphology.feature_extractor.data.Data (morphology:      Morphology,
                                                       **other_things)

```

```

    __hash__ (self)
        Return hash(self).

```

`neuron_morphology.feature_extractor.data.MorphologyLike`

```

neuron_morphology.feature_extractor.data.get_morphology (data: MorphologyLike)
    Decay a Data to a Morphology, leaving Morphologies untouched

```

`neuron_morphology.feature_extractor.feature_extraction_run`

## Module Contents

### Classes

---

*FeatureExtractionRun*

---

**class** `neuron_morphology.feature_extractor.feature_extraction_run.FeatureExtractionRun` (*data*)

**select\_marks** (*self*, *marks*: *Collection*[*Type*[*Mark*]], *required\_marks*: *AbstractSet*[*Type*[*Mark*]] = *frozenset*())

Choose marks for this run by validating a set of candidates against the data.

#### Parameters

**marks** [candidate marks to be validated]

**required\_marks** [if provided, raise an exception if any of these marks] do not validate successfully

#### Returns

**self** [This *FeatureExtractionRun*, with *selected\_marks* updated]

**select\_features** (*self*, *features*: *Collection*[*MarkedFeature*], *only\_marks*: *Optional*[*AbstractSet*[*Type*[*Mark*]]] = *None*)

Choose features to be calculated for this run on the basis of selected marks.

#### Parameters

**features** [Candidates features for selection]

**only\_marks** [if provided, reject features not marked with marks in] this set

#### Returns

**self** [This *FeatureExtractionRun*, with *selected\_features* updated]

**extract** (*self*)

For each selected feature, carry out calculation on this run's dataset.

#### Returns

**self** [This *FeatureExtractionRun*, with results updated]

**serialize** (*self*)

Return a dictionary describing this run

`neuron_morphology.feature_extractor.feature_extractor`

## Module Contents

### Classes



---

*FeatureExtractor*

---

neuron\_morphology.feature\_extractor.feature\_extractor.**RegistrableFeature**

**class** neuron\_morphology.feature\_extractor.feature\_extractor.**FeatureExtractor** (*features: Sequence[Feature]*  
=  
*tuple()*)

**register\_features** (*self, features: Sequence[RegistrableFeature]*)

Add a new feature to the list of options

**Parameters**

**features** [the features to be registered. If it is not already marked,] it will be registered with no marks

**extract** (*self, data: Data, only\_marks: Optional[AbstractSet[Type[Mark]]] = None, required\_marks: AbstractSet[Type[Mark]] = frozenset()*)

Run the feature extractor for a single dataset

**Parameters**

**data** [the dataset from which features will be calculated]

**only\_marks** [if provided, reject marks not in this set]

**required\_marks** [if provided, raise an exception if any of these marks] do not validate successfully

**Returns**

The calculated features, along with a record of the marks and features selected.

neuron\_morphology.feature\_extractor.feature\_specialization

## Module Contents

### Classes

---

*FeatureSpecialization*

---

*AxonSpec*

---

*ApicalDendriteSpec*

---

*BasalDendriteSpec*

---

*DendriteSpec*

---

*AllNeuriteSpec*

---

*AxonCompareSpec*

---

*ApicalDendriteCompareSpec*

---

*BasalDendriteCompareSpec*

---

*DendriteCompareSpec*

---

*AllNeuriteCompareSpec*

---

neuron\_morphology.feature\_extractor.feature\_specialization.**Fs**

```
class neuron_morphology.feature_extractor.feature_specialization.FeatureSpecialization
```

```
    name :str
```

```
    marks :Set[Type[Mark]]
```

```
    kwargs :Dict[str, Any]
```

```
    classmethod factory (cls: Type[Fs], name: str, marks: Set[Type[Mark]], kwargs: Dict[str, Any],  
                          display_name: Optional[str] = None)
```

A utility for quickly generating feature specializations

#### Parameters

**name** [The name of the generated class. If display\_name is not] provided, this will also be used as the name attribute of the generated class

**marks** [the marks which this specialization implies.]

**kwargs** [the keyword argument values defining this specialization]

**display\_name** [if provided, the name attribute of the generated] specialization.

#### Returns

A generated FeatureSpecialization subclass

```
neuron_morphology.feature_extractor.feature_specialization.SpecializationOption
```

```
neuron_morphology.feature_extractor.feature_specialization.SpecializationSet
```

```
neuron_morphology.feature_extractor.feature_specialization.SpecializationSets
```

```
class neuron_morphology.feature_extractor.feature_specialization.AxonSpec
```

```
    Bases: neuron_morphology.feature_extractor.feature_specialization.  
           FeatureSpecialization
```

```
    name = axon
```

```
    marks
```

```
    kwargs
```

```
class neuron_morphology.feature_extractor.feature_specialization.ApicalDendriteSpec
```

```
    Bases: neuron_morphology.feature_extractor.feature_specialization.  
           FeatureSpecialization
```

```
    name = apical_dendrite
```

```
    marks
```

```
    kwargs
```

```
class neuron_morphology.feature_extractor.feature_specialization.BasalDendriteSpec
```

```
    Bases: neuron_morphology.feature_extractor.feature_specialization.  
           FeatureSpecialization
```

```
    name = basal_dendrite
```

```
    marks
```

```
    kwargs
```

```
class neuron_morphology.feature_extractor.feature_specialization.DendriteSpec
```

```
    Bases: neuron_morphology.feature_extractor.feature_specialization.  
           FeatureSpecialization
```

```

    name = dendrite
    marks
    kwargs

class neuron_morphology.feature_extractor.feature_specialization.AllNeuriteSpec
    Bases:      neuron_morphology.feature_extractor.feature_specialization.
                FeatureSpecialization
    name = all_neurites
    marks
    kwargs

neuron_morphology.feature_extractor.feature_specialization.NEURITE_SPECIALIZATIONS

class neuron_morphology.feature_extractor.feature_specialization.AxonCompareSpec
    Bases:      neuron_morphology.feature_extractor.feature_specialization.
                FeatureSpecialization
    name = axon
    marks
    kwargs

class neuron_morphology.feature_extractor.feature_specialization.ApicalDendriteCompareSpec
    Bases:      neuron_morphology.feature_extractor.feature_specialization.
                FeatureSpecialization
    name = apical_dendrite
    marks
    kwargs

class neuron_morphology.feature_extractor.feature_specialization.BasalDendriteCompareSpec
    Bases:      neuron_morphology.feature_extractor.feature_specialization.
                FeatureSpecialization
    name = basal_dendrite
    marks
    kwargs

class neuron_morphology.feature_extractor.feature_specialization.DendriteCompareSpec
    Bases:      neuron_morphology.feature_extractor.feature_specialization.
                FeatureSpecialization
    name = dendrite
    marks
    kwargs

class neuron_morphology.feature_extractor.feature_specialization.AllNeuriteCompareSpec
    Bases:      neuron_morphology.feature_extractor.feature_specialization.
                FeatureSpecialization
    name = all_neurites
    marks
    kwargs

```

neuron\_morphology.feature\_extractor.feature\_specialization.NEURITE\_COMPARISON\_SPECIALIZATION

neuron\_morphology.feature\_extractor.feature\_writer

Utilities (mainly the FeatureWriter class) used by the feature extractor executable to format and write outputs.

Module Contents

Classes

<i>FeatureWriter</i>	
<i>FeatureFormatter</i>	Format feature results for output

Functions

<i>has_subkey</i> (subkey: str, key: str)	Check whether a string occurs as one of the “.”-separated members of
<i>add_layer_histogram</i> (writer: FeatureWriter, owner: str, key: str, histogram: LayerHistogram) → str	Add a layer histogram to this writer’s heavy data
<i>process_earth_movers_distance</i> (_writer: FeatureWriter, _owner: str, _key: str, value: EarthMoversDistanceResult) → Dict[str, Union[str, float]]	Convert an EarthMoversDistanceResult to a form suitable for json

```
class neuron_morphology.feature_extractor.feature_writer.FeatureWriter (heavy_path:
    str,
    table_path:
    Optional[str]
    =
    None,
    formatters:
    Optional[Iterable['FeatureForm
    =
    None,
    filemode:
    Optional[str]
    =
    'w')
```

```
    add_run (self, identifier: str, run: Dict[str, Any])
        Add the results of a feature extraction run to this writer

    Parameters
```

**identifier** [the unique identifier for this run]

**run** [will be added]

**process\_feature** (*self*, *owner*: str, *key*: str, *value*: Any)

**Processes a feature for writing. This may involve:**

1. changing its type to something json serializable
2. adding its value to this writer's heavy output

**Parameters**

**owner** [identifies the reconstruction that owns this feature]

**key** [the name of the feature]

**value** [the feature's raw value]

**Returns**

**Transformed feature value**

**write** (*self*)

Write this writer's output to disk

**Returns**

**This writer's outputs as a dictionary**

**validate\_table\_extension** (*self*)

If an output table was requested, check that the path has a supported extension.

**build\_output\_table** (*self*)

Convert this writer's output to a reconstruction X feature table

**Returns**

**the generated table**

**write\_table** (*self*)

Construct and write a reconstructions X features table from this writer.

**register\_formatters** (*self*, *formatters*: Iterable['FeatureFormatter'])

Add formatters to this writer. The order matters! If multiple formatters match a feature, only the first will be applied.

**Parameters**

**foratters** [an ordered collection of formatter to register]

neuron\_morphology.feature\_extractor.feature\_writer.**FeatureOutputHandler**

neuron\_morphology.feature\_extractor.feature\_writer.**FeatureOutputCheck**

**class** neuron\_morphology.feature\_extractor.feature\_writer.**FeatureFormatter**

Bases: typing.NamedTuple

Format feature results for output

**name** :str

**check** :FeatureOutputCheck

**handler** :FeatureOutputHandler

`neuron_morphology.feature_extractor.feature_writer.has_subkey(subkey: str, key: str)`

Check whether a string occurs as one of the “.”-separated members of another.

`neuron_morphology.feature_extractor.feature_writer.add_layer_histogram(writer: FeatureWriter, owner: str, key: str, histogram: LayerHistogram)`  
→  
str

Add a layer histogram to this writer’s heavy data

#### Parameters

**owner** [identifies the reconstruction that owns this feature]

**key** [the name of the histogram]

**histogram** [the histogram’s data]

#### Returns

the path at which this writer’s heavy data will be stored

`neuron_morphology.feature_extractor.feature_writer.process_earth_movers_distance(_writer: FeatureWriter, _owner: str, _key: str, value: EarthMoversDistanceResult)`  
→  
Dict[str, Union[str, float]]

Convert an EarthMoversDistanceResult to a form suitable for json serialization

#### Parameters

**\_\*** [these are unused]

**value** [The result to be transformed]

#### Returns

a dictionary like:

```
{ "result": <float result>, "interpretation" : <name of interpretation enum value>
}
```

`neuron_morphology.feature_extractor.feature_writer.numpy_array_formatter`

`neuron_morphology.feature_extractor.feature_writer.normalized_depth_histogram_formatter`

`neuron_morphology.feature_extractor.feature_writer.earth_movers_distance_formatter`

`neuron_morphology.feature_extractor.feature_writer.DEFAULT_FEATURE_FORMATTERS`

`neuron_morphology.feature_extractor.mark`

## Module Contents

### Classes

<i>Mark</i>	A tag, intended for use in feature selection.
<i>RequiresLayerAnnotations</i>	A tag, intended for use in feature selection.
<i>Intrinsic</i>	Indicates intrinsic features that don't rely on a ccf or scale.
<i>Geometric</i>	Indicates features that change depending on coordinate frame.
<i>AllNeuriteTypes</i>	Indicates features that are calculated for all neurite types.
<i>RequiresDendrite</i>	This feature can only be calculated for neurons with at least one
<i>RequiresRelativeSomaDepth</i>	This feature can only be calculated for relative soma depth
<i>RequiresSoma</i>	Indicates that these features require a soma.
<i>RequiresApical</i>	Indicates that these features require an apical dendrite.
<i>RequiresBasal</i>	Indicates that these features require a basal dendrite.
<i>RequiresAxon</i>	Indicates that these features require an axon.
<i>RequiresRoot</i>	Indicates that this features require a root. Warns if the root
<i>BifurcationFeatures</i>	Indicates a feature calculated on bifurcations.
<i>CompartmentFeatures</i>	Indicates a feature calculated on compartments.
<i>TipFeatures</i>	Indicates a feature calculated on tips (leaf nodes).
<i>NeuriteTypeComparison</i>	Indicates a feature that is a comparison between neurite types.
<i>RequiresRadii</i>	This feature can only be calculated if the radii of nodes are annotated.
<i>RequiresReferenceLayerDepths</i>	This feature can only be calculated if a referenceset of average depths
<i>RequiresLayeredPointDepths</i>	This feature can only be calculated if (cortical) points are annotated
<i>RequiresRegularPointSpacing</i>	This features can only be (meaningfully) calculated if the points (e.g.

## Functions

---

<code>check_nodes_have_key</code> (data: Data, key: str) → bool	Checks whether each node in a morphology is annotated with some key.
---	--

---

`neuron_morphology.feature_extractor.mark.Mr`

**class** `neuron_morphology.feature_extractor.mark.Mark`

A tag, intended for use in feature selection.

**classmethod** `validate` (cls, data: Data)

Determine if this feature is calculable from the provided data.

**Parameters**

**data** [Data from a single morphological reconstruction]

**Returns**

**whether marked features can be calculated from these data**

**classmethod** `factory` (cls: Type[Mr], name: str)

**class** `neuron_morphology.feature_extractor.mark.RequiresLayerAnnotations`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

A tag, intended for use in feature selection.

**classmethod** `validate` (cls, data: Data)

Checks whether each node in the data's morphology is annotated with a cortical layer. Returns False if any are missing.

**class** `neuron_morphology.feature_extractor.mark.Intrinsic`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates intrinsic features that don't rely on a ccf or scale.

**class** `neuron_morphology.feature_extractor.mark.Geometric`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates features that change depending on coordinate frame.

**class** `neuron_morphology.feature_extractor.mark.AllNeuriteTypes`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates features that are calculated for all neurite types.

**class** `neuron_morphology.feature_extractor.mark.RequiresDendrite`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

This feature can only be calculated for neurons with at least one dendrite node

**classmethod** `validate` (cls, data: Data)

Determine if this feature is calculable from the provided data.

**Parameters**

**data** [Data from a single morphological reconstruction]

**Returns**

**whether marked features can be calculated from these data**



**class** `neuron_morphology.feature_extractor.mark.RequiresRelativeSomaDepth`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

This feature can only be calculated for relative soma depth

**classmethod** `validate(cls, data: Data)`

Determine if this feature is calculable from the provided data.

#### Parameters

**data** [Data from a single morphological reconstruction]

#### Returns

**whether marked features can be calculated from these data**

**class** `neuron_morphology.feature_extractor.mark.RequiresSoma`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates that these features require a soma.

**classmethod** `validate(cls, data: Data)`

Determine if this feature is calculable from the provided data.

#### Parameters

**data** [Data from a single morphological reconstruction]

#### Returns

**whether marked features can be calculated from these data**

**class** `neuron_morphology.feature_extractor.mark.RequiresApical`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates that these features require an apical dendrite.

**classmethod** `validate(cls, data: Data)`

Determine if this feature is calculable from the provided data.

#### Parameters

**data** [Data from a single morphological reconstruction]

#### Returns

**whether marked features can be calculated from these data**

**class** `neuron_morphology.feature_extractor.mark.RequiresBasal`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates that these features require a basal dendrite.

**classmethod** `validate(cls, data: Data)`

Determine if this feature is calculable from the provided data.

#### Parameters

**data** [Data from a single morphological reconstruction]

#### Returns

**whether marked features can be calculated from these data**

**class** `neuron_morphology.feature_extractor.mark.RequiresAxon`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates that these features require an axon.

**classmethod validate** (*cls*, *data*: *Data*)

Determine if this feature is calculable from the provided data.

**Parameters**

**data** [Data from a single morphological reconstruction]

**Returns**

**whether marked features can be calculated from these data**

**class** `neuron_morphology.feature_extractor.mark.RequiresRoot`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates that this features require a root. Warns if the root is not unique

**classmethod validate** (*cls*, *data*: *Data*)

Determine if this feature is calculable from the provided data.

**Parameters**

**data** [Data from a single morphological reconstruction]

**Returns**

**whether marked features can be calculated from these data**

**class** `neuron_morphology.feature_extractor.mark.BifurcationFeatures`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates a feature calculated on bifurcations.

**class** `neuron_morphology.feature_extractor.mark.CompartmentFeatures`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates a feature calculated on compartments.

**class** `neuron_morphology.feature_extractor.mark.TipFeatures`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates a feature calculated on tips (leaf nodes).

**class** `neuron_morphology.feature_extractor.mark.NeuriteTypeComparison`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

Indicates a feature that is a comparison between neurite types.

Function should be decorated with the appropriate RequiresType marks

**class** `neuron_morphology.feature_extractor.mark.RequiresRadii`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

This feature can only be calculated if the radii of nodes are annotated.

**classmethod validate** (*cls*, *data*: *Data*)

Determine if this feature is calculable from the provided data.

**Parameters**

**data** [Data from a single morphological reconstruction]

**Returns**

**whether marked features can be calculated from these data**

**class** `neuron_morphology.feature_extractor.mark.RequiresReferenceLayerDepths`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

This feature can only be calculated if a referenceset of average depths for cortical layers is provided. See `features.layer.reference_layer_depths` for more information.

**classmethod `validate`** (*cls*, *data*: *Data*)

Determine if this feature is calculable from the provided data.

#### Parameters

**data** [Data from a single morphological reconstruction]

#### Returns

whether marked features can be calculated from these data

**class** `neuron_morphology.feature_extractor.mark.RequiresLayeredPointDepths`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

This feature can only be calculated if (cortical) points are annotated with a collection of within-layer depths. See `features.layer.layered_point_depths` for more information.

**classmethod `validate`** (*cls*, *data*: *Data*)

Determine if this feature is calculable from the provided data.

#### Parameters

**data** [Data from a single morphological reconstruction]

#### Returns

whether marked features can be calculated from these data

**class** `neuron_morphology.feature_extractor.mark.RequiresRegularPointSpacing`

Bases: `neuron_morphology.feature_extractor.mark.Mark`

This features can only be (meaningfully) calculated if the points (e.g. node positions) on which it is based are resampled to have regular spacing.

`neuron_morphology.feature_extractor.mark.check_nodes_have_key` (*data*: *Data*, *key*: *str*) → bool

Checks whether each node in a morphology is annotated with some key.

`neuron_morphology.feature_extractor.marked_feature`

## Module Contents

### Classes

---

`MarkedFeature`

---

### Functions

<code>specialize</code> (feature: Feature, specialization_set: SpecializationSet) → Dict[str, MarkedFeature]	Bind some of a feature's keyword arguments, using provided
<code>nested_specialize</code> (feature: Feature, specialization_sets: SpecializationSets) → Dict[str, MarkedFeature]	Apply specializations hierarchically to a base feature. Generating a
<code>marked</code> (mark: Type[Mark])	Decorator for adding a mark to a function.

neuron\_morphology.feature\_extractor.marked\_feature.**FeatureFn**

neuron\_morphology.feature\_extractor.marked\_feature.**M**

```
class neuron_morphology.feature_extractor.marked_feature.MarkedFeature (marks:
                                                                    Set[Type[Mark]],
                                                                    fea-
                                                                    ture:
                                                                    Fea-
                                                                    ture,
                                                                    name:
                                                                    Op-
                                                                    tional[str]
                                                                    =
                                                                    None,
                                                                    pre-
                                                                    serve_marks:
                                                                    bool
                                                                    =
                                                                    True)
```

**\_\_slots\_\_** = ['marks', 'feature', 'name']

**\_\_name\_\_**

**\_\_repr\_\_** (self)  
Return repr(self).

**\_\_hash\_\_** (self)  
Return hash(self).

**add\_mark** (self, mark: Type[Mark])  
Assign an additional mark to this feature

**\_\_call\_\_** (self, \*args, \*\*kwargs)  
Execute the underlying feature, passing along all arguments

**deepcopy** (self, \*\*kwargs)  
Make a deep copy of this marked feature

**partial** (self, \*args, \*\*kwargs)  
Fix one or more parameters on this feature's callable

**specialize** (self, option: SpecializationOption)  
Apply a specialization option to this feature. This binds parameters on the feature's **\_\_call\_\_** method, sets 0 or more additional marks, and namespaces the feature's name.

#### Parameters

**option** [The specialization option with which to specialize this] feature.

#### Returns

a deep copy of this feature with updated callable, marks and name

**classmethod ensure** (cls: Type[M], feature: Feature)  
If a function is not a MarkedFeature, convert it.

#### Parameters

**feature** [the feature to be converted]

#### Returns

Either a marked feature generated from the input, or the input marked feature.

`neuron_morphology.feature_extractor.marked_feature.Feature`

`neuron_morphology.feature_extractor.marked_feature.specialize` (*feature: Feature, specialization\_set: SpecializationSet*) → Dict[str, MarkedFeature]

Bind some of a feature's keyword arguments, using provided specialization options.

#### Parameters

**feature** [will be used as a basis for specialization]

**specialization\_set** [each element defines a particular specialization (i.e) a set of keyword argument values and marks) to be applied to the feature

#### Returns

A dictionary mapping (namespaced) feature names to specialized features.

Note that names are formatted as “specialization\_name.base\_feature\_name”

`neuron_morphology.feature_extractor.marked_feature.nested_specialize` (*feature: Feature, specialization\_sets: SpecializationSets*) → Dict[str, MarkedFeature]

Apply specializations hierarchically to a base feature. Generating a new collection of specialized features.

#### Parameters

**feature** [will be used as a basis for specialization]

**specialization\_sets** [each element describes a set of specialization] options. The output will have one specialization for each element of the cartesian product of these sets.

#### Returns

A dictionary mapping namespaced feature names to specialized features.

### Notes

Specializations are applied from the start of the specialization\_sets to the end. This means that the generated names are structures like:

“last\_spec.middle\_spec.first\_spec.base\_feature\_name”

`neuron_morphology.feature_extractor.marked_feature.marked` (*mark: Type[Mark]*)  
Decorator for adding a mark to a function.

**Parameters**

**mark** [the mark to be applied]

**Examples**

```
@marked(RequiresA) @marked(RequiresB) def some_feature_requiring_a_and_b(...):  
    ...
```

`neuron_morphology.feature_extractor.run_feature_extraction`

**Module Contents****Functions**

<code>resolve_reference_layer_depths</code> (key=None, names=None, boundaries=None)	Given either the name of a well known depths set or a set of names and
<code>hydrate_parameters</code> (parameters: Dict[str, Any]) → Dict[str, Any]	Resolve argued feature parameters to a format comprehensible by
<code>setup_data</code> (reconstruction: Dict[str, Any], global_parameters: Dict[str, Any]) → Tuple[str, Data]	Construct a Data for extracting features from a single reconstruction.
<code>run_feature_extraction</code> (reconstruction_spec: Dict[str, Any], feature_set: str, only_marks: List[str], required_marks: List[str], global_parameter_spec: Dict[str, Any]) → Tuple[str, Dict]	Run feature extraction for a single reconstruction.

`neuron_morphology.feature_extractor.run_feature_extraction.well_known_marks :Dict[str, Type`

`neuron_morphology.feature_extractor.run_feature_extraction.item`

`neuron_morphology.feature_extractor.run_feature_extraction.known_feature_sets`

`neuron_morphology.feature_extractor.run_feature_extraction.resolve_reference_layer_depths (A`

Given either the name of a well known depths set or a set of names and corresponding boundaries, produce a ReferenceLayerDepths

**Parameters**

**key** [of a well known reference layer]

**names** [the names of each layer in a custom sequence]

**boundaries** [the upper and lower depths of each layer in a custom sequence]

**Returns**

**the requested reference layer depths**

`neuron_morphology.feature_extractor.run_feature_extraction.hydrate_parameters` (*parameters:* *Dict[str, Any]*)  
 → *Dict[str, Any]*

Resolve argued feature parameters to a format comprehensible by the features. e.g. loading data from a path.

#### Parameters

**parameters** [to be hydrated]

#### Returns

**The hydrated parameters**

`neuron_morphology.feature_extractor.run_feature_extraction.setup_data` (*reconstruction:* *Dict[str, Any]*, *global\_parameters:* *Dict[str, Any]*)  
 → *Tuple[str, Data]*

Construct a Data for extracting features from a single reconstruction.

#### Parameters

**reconstruction** [The reconstruction to be setup. Must specify an `swc_path`]

**global\_parameters** [any cross-reconstruction feature parameters]

#### Returns

**identifier** [a label for this reconstruction]

**data suitable for feature extraction**

`neuron_morphology.feature_extractor.run_feature_extraction.run_feature_extraction` (*reconstruction:* *Dict[str, Any]*, *feature\_set:* *str*, *only\_marks:* *List[str]*, *required\_marks:* *List[str]*, *global\_parameters:* *Dict[str, Any]*)  
 → *Tuple[str, Dict]*

Run feature extraction for a single reconstruction.

#### Parameters

**reconstruction\_spec** [a dictionary specifying a reconstruction. Must] have an `swc_path`.

**feature\_set** [names the set of features for which calculation will be] attempted

**only\_marks** [names marks to which calculation will be restricted]

**required\_marks** [raise an exception if these named marks fail validation]

**global\_parameter\_spec** [a dictionary specifying cross-reconstruction] parameters

#### Returns

**identifier** [a label for this reconstruction]

**A dict with keys:** `results` - a dict, mapping features to calculated values `selected_marks` - the set of marks that passed validation `selected_features` - the set of features for which calculation was

`attempted`

### `neuron_morphology.feature_extractor.utilities`

A collection of miscellaneous tools used by the feature extractor

## Module Contents

### Functions

---

<code>unnest</code> (inputs: Dict[str, Any], _prefix="") → Dict[str, Any]	Convert nested dictionaries (with string keys) to a dot-notation flat
---	---

---

`neuron_morphology.feature_extractor.utilities.unnest` (inputs: Dict[str, Any], \_prefix="") → Dict[str, Any]

Convert nested dictionaries (with string keys) to a dot-notation flat dictionary.

inputs: The dictionary to unnest. Must have all string keys `_prefix` : Used during recursion to build up a dot-notation prefix. Don't

argue this yourself!

#### Returns

**a flattened dictionary**

### `neuron_morphology.features`

#### Subpackages

`neuron_morphology.features.branching`

#### Submodules

`neuron_morphology.features.branching.bifurcations`



## Module Contents

### Functions

<code>calculate_outer_bifs</code> (morphology: Morphology, soma: Dict, node_types: Optional[List[int]]) → int	Counts the number of bifurcation points beyond the a sphere
<code>num_outer_bifurcations</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → int	Feature Extractor interface to calculate_outer_bifurcations. Returns
<code>mean_bifurcation_angle_local</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Compute the average angle between child segments at
<code>mean_bifurcation_angle_remote</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Compute the average angle between the next branch point or terminal

`neuron_morphology.features.branching.bifurcations.calculate_outer_bifs` (morphology: Morphology, soma: Dict, node\_types: Optional[List[int]]) → int

Counts the number of bifurcation points beyond the a sphere with 1/2 the radius from the soma to the most distant point in the morphology, with that sphere centered at the soma.

#### Parameters

**morphology:** Describes the structure of a neuron

**soma:** Must have keys “x”, “y”, and “z”, describing the position of this morphology’s soma in

**node\_types:** Restrict included nodes to these types. See `neuron_morphology.constants` for available node types.

#### Returns

the number of bifurcations

`neuron_morphology.features.branching.bifurcations.num_outer_bifurcations` (data: MorphologyLike, node\_types: Optional[List[int]] = None) → int

Feature Extractor interface to calculate\_outer\_bifurcations. Returns the number of bifurcations (branch points), excluding those too close to the root (threshold is 1/2 the max distance from the root to any node).

**Parameters**

**data** [Holds a morphology object. No additional data is required]

**node\_types** [Restrict included nodes to these types. See] neuron\_morphology.constants for available node types.

neuron\_morphology.features.branching.bifurcations.mean\_bifurcation\_angle\_local (data: MorphologyLike, node\_types: Optional[List[int]] = None) → float

Compute the average angle between child segments at bifurcations throughout the morphology. Trifurcations are ignored. Note: this introduces possible segmentation artifacts if trifurcations are due to large segment sizes.

**Parameters**

**data:** The reconstruction whose max euclidean distance will be calculated

**node\_types:** restrict consideration to these types

**Returns**

Scalar value

neuron\_morphology.features.branching.bifurcations.mean\_bifurcation\_angle\_remote (data: MorphologyLike, node\_types: Optional[List[int]] = None) → float

Compute the average angle between the next branch point or terminal tip of child segments at each bifurcation. Trifurcations are ignored. Note: this introduces possible segmentation artifacts if trifurcations are due to large segment sizes.

**Parameters**

**data:** The reconstruction whose max euclidean distance will be calculated

**node\_types:** restrict consideration to these types

**Returns**

Scalar value, nan if no nodes

`neuron_morphology.features.layer`

## Submodules

`neuron_morphology.features.layer.layer_histogram`

## Module Contents

### Classes

<code>LayerHistogram</code>	The results of calculating a within-layer depth histogram of points
<code>EarthMoversDistanceInterpretation</code>	Describes how to understand an earth mover's distance result. This is
<code>EarthMoversDistanceResult</code>	The result of comparing two histograms using earth mover's distance

### Functions

<code>ensure_tuple</code> (inputs: Any, item_type: Type, if_none: Union[str, Tuple] = 'raise') → Tuple	Try to smartly coerce inputs to a tuple.
<code>ensure_node_types</code> (node_types)	Make sure the argued node types are a tuple
<code>ensure_layers</code> (layers)	Make sure the argued layer array is a tuple
<code>earth_movers_distance</code> (data: Data, node_types: Sequence[int], node_types_to_compare: Sequence[int], bin_size: float = 5) → Dict[str, EarthMoversDistanceResult]	Calculate the earth mover's distance between normalized histograms of
<code>histogram_earth_movers_distance</code> (from_hist: np.ndarray, to_hist: np.ndarray) → EarthMoversDistanceResult	Calculate the earth mover's distance between to histograms, normalizing
<code>normalized_depth_histogram</code> (data: Data, node_types: Optional[Sequence[int]] = None, bin_size=5.0) → Dict[str, LayerHistogram]	Calculates for each cortical layer a histogram of node depths within
<code>normalized_depth_histograms_across_layers</code> (data: Data, point_types: Optional[Tuple[int]] = None, only_layers: Optional[Tuple[str]] = None, bin_size=5.0) → Dict[str, LayerHistogram]	A helper function for running cortical depth histograms across multiple
<code>normalized_depth_histogram_within_layer</code> (local_layer_pia_side_depths: np.ndarray, local_layer_wm_side_depths: np.ndarray, reference_layer_depths: ReferenceLayerDepths, bin_size: float) → np.ndarray	Calculate a histogram of node depths within a single (cortical) layer.

`neuron_morphology.features.layer.layer_histogram.ensure_tuple` (inputs: Any, item\_type: Type, if\_none: Union[str, Tuple] = 'raise') → Tuple

Try to smartly coerce inputs to a tuple.

**Parameters**

**inputs** [the data to be coerced]

**item\_type** [which type do/should the elements of the tuple have?]

**if\_none** [if the inputs are none, return this value. If the value is] “raise”, instead raise an exception

**Returns**

**the coerced inputs**

`neuron_morphology.features.layer.layer_histogram.ensure_node_types(node_types)`  
Make sure the argued node types are a tuple

`neuron_morphology.features.layer.layer_histogram.ensure_layers(layers)`  
Make sure the argued layer array is a tuple

**class** `neuron_morphology.features.layer.layer_histogram.LayerHistogram`  
Bases: `typing.NamedTuple`

The results of calculating a within-layer depth histogram of points within some cortical layer.

**counts** :`np.ndarray`

**bin\_edges** :`np.ndarray`

**class** `neuron_morphology.features.layer.layer_histogram.EarthMoversDistanceInterpretation`  
Bases: `enum.Enum`

Describes how to understand an earth mover’s distance result. This is useful in the case that one or both histograms are all 0.

**BothPresent** = 0

**OneEmpty** = 1

**BothEmpty** = 2

**class** `neuron_morphology.features.layer.layer_histogram.EarthMoversDistanceResult`  
Bases: `typing.NamedTuple`

The result of comparing two histograms using earth mover’s distance

**result** :`float`

**interpretation** :`EarthMoversDistanceInterpretation`

**to\_dict\_human\_readable** (*self*)

```
neuron_morphology.features.layer.layer_histogram.earth_movers_distance(data:
                                                                    Data,
                                                                    node_types:
                                                                    Se-
                                                                    quence[int],
                                                                    node_types_to_compare:
                                                                    Se-
                                                                    quence[int],
                                                                    bin_size:
                                                                    float
                                                                    =
                                                                    5)
                                                                    →
                                                                    Dict[str,
                                                                    Earth-
                                                                    Movers-
                                                                    Dis-
                                                                    tanceRe-
                                                                    sult]
```

Calculate the earth mover's distance between normalized histograms of node depths within cortical layers. Calculates one distance for each layer.

#### Parameters

**data** [Must be endowed with `layered_point_depths` and `reference_layer_depths`.] The morphology is not actually used directly.

**node\_types** [Defines one set of points whose histograms to compare.]

**node\_types\_to\_compare** [Defines the other set of points]

**bin\_size** [the size of each depth bin. Default is appropriate if the units] are microns.

#### Returns

A mapping from layers to distances between histograms within those layers.

```
neuron_morphology.features.layer.layer_histogram.histogram_earth_movers_distance(from_hist:
                                                                    np.ndarray,
                                                                    to_hist:
                                                                    np.ndarray)
                                                                    →
                                                                    Earth-
                                                                    Movers-
                                                                    Dis-
                                                                    tanceRe-
                                                                    sult
```

Calculate the earth mover's distance between two histograms, normalizing each. If one histogram is empty, return the sum of the other and a flag. If both are empty, return 0 and a flag.

#### Parameters

**from\_hist** [distance is calculated between (the normalized form of) this] histogram and `to_hist`. The result is symmetric.

**to\_hist** [distance is calculated between (the normalized form of) this] histogram and `from_hist`

#### Returns

The distance between input histograms, along with an enum indicating

whether one or both of the histograms was all 0.

```
neuron_morphology.features.layer.layer_histogram.normalized_depth_histogram(data:  
                                                                           Data,  
                                                                           node_types:  
                                                                           Op-  
                                                                           tional[Sequence[int]]  
                                                                           =  
                                                                           None,  
                                                                           bin_size=5.0)  
                                                                           →  
                                                                           Dict[str,  
                                                                           Layer-  
                                                                           er-  
                                                                           His-  
                                                                           togram]
```

Calculates for each cortical layer a histogram of node depths within that layer.

#### Parameters

**data** [Must have the following attributes:]

**reference\_layer\_depths** [A dictionary mapping layer names (str) to] ReferenceLayerDepths objects describing the average pia and white- matter side depths of this each layer.

**layered\_point\_depths** [A LayeredPointDepths defining for each point a] depth from pia. See LayeredPointDepths for more information.

**node\_types** [for which to calculate the histograms]

**bin\_size** [the size of each depth bin. Default is appropriate if the units] are microns.

```
neuron_morphology.features.layer.layer_histogram.normalized_depth_histograms_across_layers
```

A helper function for running cortical depth histograms across multiple layers.

#### Parameters

**data** [must have reference\_layer\_depths and layered\_point\_depths]

**point\_types** [calculate histograms for points labeled with these types]

**only\_layers** [exclude other layers from this calculation]

**bin\_size** [the size of each depth bin. Default is appropriate if the units] are microns.

`neuron_morphology.features.layer.layer_histogram.normalized_depth_histogram_within_layer` (pa

Calculates a histogram of node depths within a single (cortical) layer. Uses reference information about layer boundaries to normalize these depths for cross-reconstruction comparison.

#### Parameters

**depths** [Each item corresponds to a point of interest (such as a node) in a morphological reconstruction). Values are the depths of these points of interest from the pia surface.

**local\_layer\_pia\_side\_depths** [Each item corresponds to a point of interest.] Values are the depth of the intersection point between a path of steepest descent from the pia surface to the point of interest and the upper surface of the layer.

**local\_layer\_wm\_side\_depths** [Each item corresponds to a point of interest.] Values are the depth of the intersection point between the layer's lower boundary and the path described above.

**reference\_layer\_depths** [Used to provide normalized depths suitable] for comparison across reconstructions. Should provide a generic equivalent of local layer depths for a population or reference space.

**bin\_size** [The width of each bin, in terms of depths from pia in the] reference space. Provide only one of `bin_edges` or `bin_size`.

#### Returns

A numpy array listing for each depth bin the number of nodes falling within that bin.

#### Notes

This function relies on the notion of a steepest descent path through cortex, but is agnostic to the method used to obtain such a path and to features of the path (e.g. whether it is allowed to curve). Rather the caller must ensure that all depths have been calculated according to a consistent scheme.

neuron\_morphology.features.layer.layered\_point\_depths

Module Contents

Classes

---

<i>LayeredPointDepths</i>
---------------------------

---

```
class neuron_morphology.features.layer.layered_point_depths.LayeredPointDepths (ids:
    Sequence,
    layer_name:
    Sequence[str],
    depth:
    Sequence,
    lo-
    cal_layer_pia_si
    Sequence,
    lo-
    cal_layer_wm_si
    Sequence,
    point_type:
    Sequence)

    DF_COLS
    to_csv (self, path: str)
    classmethod from_dataframe (cls, df: pd.DataFrame)
    classmethod from_csv (cls, path: str)
    classmethod from_hdf5 (cls, path: str)
    classmethod read (cls, path: str)
```

neuron\_morphology.features.layer.reference\_layer\_depths

Module Contents

Classes

---

<i>ReferenceLayerDepths</i>	Reference (e.g. average across specimens and regions) depths of
-----------------------------	---

---

```
class neuron_morphology.features.layer.reference_layer_depths.ReferenceLayerDepths
    Bases: typing.NamedTuple
```



Reference (e.g. average across specimens and regions) depths of cortical layer boundaries. Depths are given from pia. Units are not specified, but the user should ensure they are consistent with other positional and size units (e.g. node positions and radii, point depths). Several features in this package specify defaults in microns; if you provide reference layer depths in other units, you should review features which use these depths and ensure that any default values agree with your units.

#### Attributes

**pia\_side** [the (average) depth of the upper surface of the layer]

**wm\_side** [the (average) depth of the lower (closer to white matter) surface] of the layer

**scale** [if True, these depths are taken as describing the upper and lower] surfaces of a real feature of the data. If False, one or both of them is taken to represent a user-selected boundary. In the latter case, features such as the layer histograms will not attempt to rescale point depths based on observed local layer thicknesses.

**pia\_side** :float

**wm\_side** :float

**scale** :bool = True

**thickness**

**classmethod sequential** (*cls*, *names*: Sequence[str], *boundaries*: Sequence[float],  
last\_is\_scale=False)

A utility for constructing multiple ordered reference layer depths without intervening space.

#### Parameters

**names** [The name of each layer]

**boundaries** [The pia and wm side depth of each layer. Should be a flat] sequence that has 1 more element than names.

**last\_is\_scale** [If True, the last boundary will be interpreted as a] true anatomical boundary. If false, as an arbitrary cutoff.

`neuron_morphology.features.layer.reference_layer_depths.DEFAULT_HUMAN_ME_MET_REFERENCE_LAYER_I`

`neuron_morphology.features.layer.reference_layer_depths.DEFAULT_HUMAN_MTG_REFERENCE_LAYER_I`

`neuron_morphology.features.layer.reference_layer_depths.DEFAULT_MOUSE_ME_MET_REFERENCE_LAYER_I`

`neuron_morphology.features.layer.reference_layer_depths.DEFAULT_MOUSE_REFERENCE_LAYER_DEPTH`

`neuron_morphology.features.layer.reference_layer_depths.WELL_KNOWN_REFERENCE_LAYER_DEPTHS`

`neuron_morphology.features.statistics`

#### Submodules

`neuron_morphology.features.statistics.coordinates`

#### Module Contents

#### Classes

<i>COORD_TYPE</i>	Generic enumeration.
<i>NodeSpec</i>	
<i>BifurcationSpec</i>	
<i>CompartmentSpec</i>	
<i>TipSpec</i>	

## Functions

<i>get_compartment_coordinates</i> (morphology, node_types: Optional[List[int]] = None)	Return the coordinates of the midpoint of each compartment
<i>get_bifurcation_coordinates</i> (morphology, node_types: Optional[List[int]] = None)	Return the coordinates of each bifurcation in the morphology
<i>get_tip_coordinates</i> (morphology, node_types: Optional[List[int]] = None)	Return the coordinates of each tip in the morphology
<i>get_node_coordinates</i> (morphology, node_types: Optional[List[int]] = None)	Return the coordinates of each node in the morphology
<i>get_coordinates</i> (morphology: Morphology, coordinate_type: COORD_TYPE = COORD_TYPE.NODE, node_types: Optional[List[int]] = None)	

**class** neuron\_morphology.features.statistics.coordinates.COORD\_TYPE

Bases: enum.Enum

Generic enumeration.

Derive from this class to define new enumerations.

**NODE** = 0

**COMPARTMENT** = 1

**BIFURCATION** = 2

**TIP** = 3

**get\_coordinates** (self, morphology, node\_types: Optional[List[int]] = None)

**class** neuron\_morphology.features.statistics.coordinates.NodeSpec

Bases: neuron\_morphology.feature\_extractor.feature\_specialization.FeatureSpecialization

**name** = node

**marks**

**kwargs**

**class** neuron\_morphology.features.statistics.coordinates.BifurcationSpec

Bases: neuron\_morphology.feature\_extractor.feature\_specialization.FeatureSpecialization

**name** = bifurcation

**marks**

**kwargs**

```
class neuron_morphology.features.statistics.coordinates.CompartmentSpec
    Bases: neuron_morphology.feature_extractor.feature_specialization.
            FeatureSpecialization
```

```
    name = compartment
```

```
    marks
```

```
    kwargs
```

```
class neuron_morphology.features.statistics.coordinates.TipSpec
    Bases: neuron_morphology.feature_extractor.feature_specialization.
            FeatureSpecialization
```

```
    name = tip
```

```
    marks
```

```
    kwargs
```

```
neuron_morphology.features.statistics.coordinates.COORD_TYPE_SPECIALIZATIONS
```

```
neuron_morphology.features.statistics.coordinates.get_compartment_coordinates (morphology,
                                         node_types:
                                         Optional[List[int]]
                                         =
                                         None)
```

Return the coordinates of the midpoint of each compartment in the morphology

#### Parameters

**morphology:** Morphology object

**node\_types:** list (AXON, BASAL\_DENDRITE, APICAL\_DENDRITE)

#### Returns

**list:** list of coordinates [x, y, z]

```
neuron_morphology.features.statistics.coordinates.get_bifurcation_coordinates (morphology,
                                         node_types:
                                         Optional[List[int]]
                                         =
                                         None)
```

Return the coordinates of each bifurcation in the morphology

#### Parameters

**morphology:** Morphology object

**node\_types:** list (AXON, BASAL\_DENDRITE, APICAL\_DENDRITE)

#### Returns

**list:** list of coordinates [x, y, z]

```
neuron_morphology.features.statistics.coordinates.get_tip_coordinates (morphology,
                                         node_types:
                                         Optional[List[int]]
                                         =
                                         None)
```

Return the coordinates of each tip in the morphology

**Parameters****morphology:** Morphology object**node\_types:** list (AXON, BASAL\_DENDRITE, APICAL\_DENDRITE)**Returns****list:** list of coordinates [x, y, z]

```
neuron_morphology.features.statistics.coordinates.get_node_coordinates (morphology,  
                                                                    node_types:  
                                                                    Optional[List[int]]  
                                                                    =  
                                                                    None)
```

Return the coordinates of each node in the morphology

**Parameters****morphology:** Morphology object**node\_types:** list (AXON, BASAL\_DENDRITE, APICAL\_DENDRITE)**Returns****list:** list of coordinates [x, y, z]

```
neuron_morphology.features.statistics.coordinates.get_coordinates (morphology:  
                                                                    Morphol-  
                                                                    ogy, coordi-  
                                                                    nate_type:  
                                                                    CO-  
                                                                    ORD_TYPE  
                                                                    = CO-  
                                                                    ORD_TYPE.NODE,  
                                                                    node_types:  
                                                                    Op-  
                                                                    tional[List[int]]  
                                                                    = None)
```

`neuron_morphology.features.statistics.moments`**Module Contents****Functions**

---

<code>moments</code>	<code>(data: Data, node_types: Optional[List] = None, coord_type: COORD_TYPE = COORD_TYPE.NODE)</code>	Calculate the moments of specific coordinate type and node type
----------------------	--	---

---

```
neuron_morphology.features.statistics.moments.moments (data: Data, node_types:  
                                                         Optional[List] = None, coord_type: COORD_TYPE =  
                                                         COORD_TYPE.NODE)
```

Calculate the moments of specific coordinate type and node type

**Parameters**

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see neuron\_morphology constants)

**coord\_type:** Restrict analysis to specific coordinate type (see neuron\_morphology.features.statistics.coordinates for options) neu-

`neuron_morphology.features.statistics.moments_along_max_distance_projection`

## Module Contents

### Functions

---

`moments_along_max_distance_projection(data)` Calculate the distance projections of a specific compartment and coordinate type along the line segment connecting soma to the most distant (from soma) node of that compartment.  
 Data, node\_types: Optional[List] = None, coord\_type: COORD\_TYPE = COORD\_TYPE.BIFURCATION)

---

`neuron_morphology.features.statistics.moments_along_max_distance_projection.moments_along_r`

Calculate the distance projections of a specific compartment and coordinate type along the line segment connecting soma to the most distant (from soma) node of that compartment.

#### Parameters

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see neuron\_morphology constants)

**coord\_type:** Restrict which coordinate types are measured (i.e. projected along line segment) (see neuron\_morphology.features.statistics.coordinates for options)

#### Returns

**summary\_dict:** summary stats of distances projected along specified line segment

`neuron_morphology.features.statistics.overlap`

## Module Contents

### Functions

<code>calculate_coordinate_overlap_from_min_max</code>	Return the % of coordinates that are above the max, between, or below the min
<code>calculate_coordinate_overlap</code>	Return the % of coordinates_a that are above, overlapping, and below coordinates_b, and the same for b over a
<code>overlap</code>	Compares the locations of node_types to node_types_to_compare

neuron\_morphology.features.statistics.overlap.**calculate\_coordinate\_overlap\_from\_min\_max**(coordinates: np.ndarray, minv: float, maxv: float, dimension: int = 1)

Return the % of coordinates that are above the max, between, or below the min

#### Parameters

- coordinates:** np.ndarray with x, y, z columns
- minv:** min to check against
- maxv:** max to check against
- dimension:** dimension to compare (0, 1, 2 for x, y, z), default 1 (y)

neuron\_morphology.features.statistics.overlap.**calculate\_coordinate\_overlap**(coordinates\_a, coordinates\_b, dimension: int = 1)

Return the % of coordinates\_a that are above, overlapping, and below coordinates\_b, and the same for b over a

#### Parameters

- coordinates\_a:** 2d array-like with x, y, z cols
- coordinates\_b:** 2d array-like with x, y, z cols
- dimension:** dimension to compare (0, 1, 2 for x, y, z), default 1 (y)

#### Returns

- dict:** a\_above\_b, a\_overlap\_b, a\_below\_b, or -1's if coordinates\_b is empty

```
neuron_morphology.features.statistics.overlap.overlap(data: Data, node_types:
Optional[List[int]] = None,
node_types_to_compare:
Optional[List[int]] = None,
coord_type: COORD_TYPE
= COORD_TYPE.NODE,
dimension: int = 1)
```

Compares the locations of node\_types to node\_types\_to\_compare Calculate % of coordinates of node\_types that are above, overlapping, and below the coordinates of node\_types\_to\_compare

**Example: calculate\_overlap**(

```
morphology, node_types=[AXON], node_types_to_compare=[APICAL_DENDRITE,
BASAL_DENDRITE])
```

will return the percentage of AXON nodes that are above, overlapping, and below DENDRITE nodes

#### Parameters

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see neuron\_morphology constants)

**node\_types\_to\_compare:** a list of node types (see neuron\_morphology constants)

**coord\_type:** Restrict analysis to specific coordinate type (see neuron\_morphology.features.statistics.coordinates for options)

**dimension:** dimension to compare (0, 1, 2 for x, y, z), default 1 (y)

## Submodules

`neuron_morphology.features.default_features`

## Module Contents

`neuron_morphology.features.default_features.default_features`

`neuron_morphology.features.dimension`

## Module Contents

## Functions

---

```
dimension(data: Data, node_types: Optional[List] = None, coord_type: COORD_TYPE = COORD_TYPE.NODE, signed_bias=(False, True, False))
```

---

```
neuron_morphology.features.dimension.dimension(data: Data, node_types: Optional[List] = None, coord_type: COORD_TYPE = COORD_TYPE.NODE, signed_bias=(False, True, False))
```

Get the height, width, depth, minimum, and maximum values of specific coordinate type and node type centered

about the root

#### Parameters

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see `neuron_morphology` constants)

**coord\_type:** Restrict analysis to specific coordinate type (see `neuron_morphology.features.statistics.coordinates` for options)

**signed\_bias:** boolean tuple for whether the bias measure should be signed for (x, y, z)

`neuron_morphology.features.intrinsic`

## Module Contents

### Functions

<code>num_tips(data: Data, node_types: Optional[List] = None)</code>	Calculate number of tips
<code>num_nodes(data: Data, node_types: Optional[List] = None)</code>	Calculate number of nodes of a given type
<code>child_ids_by_type(node_id, morphology, node_types=None)</code>	Helper function for the traversal functions
<code>calculate_branches_from_root(morphology, root, node_types=None)</code>	Calculate the number of branches of a specific neuron type
<code>num_branches(data: Data, node_types: Optional[List] = None)</code>	Calculate number of branches
<code>calculate_mean_fragmentation_from_root(morphology, root, node_types=None)</code>	Calculate the mean fragmentation from a root
<code>mean_fragmentation(data: Data, node_types: Optional[List] = None)</code>	Calculate the mean number of compartments per branch
<code>calculate_max_branch_order_from_root(morphology, root, node_types=None)</code>	Calculate the greatest number of branches encountered among all
<code>max_branch_order(data: Data, node_types: Optional[List] = None)</code>	Calculate mean fragmentation

`neuron_morphology.features.intrinsic.num_tips` (*data: Data, node\_types: Optional[List] = None*)

Calculate number of tips

#### Parameters

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see `neuron_morphology` constants)

`neuron_morphology.features.intrinsic.num_nodes` (*data: Data, node\_types: Optional[List] = None*)

Calculate number of nodes of a given type

#### Parameters

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see `neuron_morphology` constants)



```
neuron_morphology.features.intrinsic.child_ids_by_type (node_id, morphology,
                                                         node_types=None)
```

Helper function for the traversal functions

```
neuron_morphology.features.intrinsic.calculate_branches_from_root (morphology,
                                                                    root,
                                                                    node_types=None)
```

Calculate the number of branches of a specific neuron type in a morphology. A branch is defined as being between two bifurcations or between a bifurcation and a tip if a node has three or more children, it is treated as successive bifurcations, e.g a trifurcation: `_/_/_` creates 4 branches since the branch between the two bifurcations counts

#### Parameters

**morphology:** a morphology object

**root:** the root node to traverse from

**node\_types:** a list of node types (see neuron\_morphology constants)

```
neuron_morphology.features.intrinsic.num_branches (data: Data, node_types: Optional[List] = None)
```

Calculate number of branches

#### Parameters

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see neuron\_morphology constants)

```
neuron_morphology.features.intrinsic.calculate_mean_fragmentation_from_root (morphology,
                                                                                root,
                                                                                node_types=None)
```

Calculate the mean fragmentation from a root in a morphology. Mean fragmentation is the number of compartments over the number of branches. A branch is defined as being between two bifurcations or between a bifurcation and a tip if a node has three or more children, it is treated as successive bifurcations, e.g a trifurcation: `_/_/_` creates 4 branches since the branch between the two bifurcations counts

#### Parameters

**morphology:** a morphology object

**root:** the root node to traverse from

**node\_types:** a list of node types (see neuron\_morphology constants)

```
neuron_morphology.features.intrinsic.mean_fragmentation (data: Data, node_types: Optional[List] = None)
```

Calculate the mean number of compartments per branch

#### Parameters

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see neuron\_morphology constants)

```
neuron_morphology.features.intrinsic.calculate_max_branch_order_from_root (morphology,
                                                                                root,
                                                                                node_types=None)
```

Calculate the greatest number of branches encountered among all directed paths from the morphology's root to its leaves. A branch is defined as a root->leaf ordered path for which:

1. the first node on the path is either
  - a. a bifurcation (has > 1 children)

- b. the root node
- 2. the last node on the path is either
  - a. a bifurcation
  - b. a leaf node (has 0 children)

**Parameters**

**morphology:** the reconstruction whose max branch order will be calculated

**root:** treat this node as root

**node\_types:** If not None, consider only root->leaf paths whose leaf nodes are among these types (see neuron\_morphology constants)

**Returns**

The greatest branch count encountered among all considered root->leaf paths

neuron\_morphology.features.intrinsic.max\_branch\_order(*data: Data, node\_types: Optional[List] = None*)

Calculate mean fragmentation

**Parameters**

**data:** Data Object containing a morphology

**node\_types:** a list of node types (see neuron\_morphology constants)

neuron\_morphology.features.path

**Module Contents****Functions**

---

<code>_calculate_max_path_distance(morphology, root, node_types)</code>	
<code>calculate_max_path_distance(morphology, root, node_types=None)</code>	Helper for max_path_distance. See below for more information.
<code>max_path_distance(data: MorphologyLike, node_types: Optional[List[int]] = None) → float</code>	Calculate the distance, following the path of adjacent neurites, from
<code>early_branch_path(data: MorphologyLike, node_types: Optional[List[int]] = None, soma: Optional[Dict] = None) → float</code>	Returns the ratio of the longest ‘short’ branch from a bifurcation to
<code>_calculate_mean_contraction(morphology, reference, root, node_types)</code>	Calculate the average contraction of all sections. In other words,
<code>calculate_mean_contraction(morphology, root=None, node_types=None)</code>	See mean_contraction
<code>mean_contraction(data: MorphologyLike, node_types: Optional[List[int]] = None) → float</code>	Calculate the average contraction of all sections. In other words,

---

neuron\_morphology.features.path.\_calculate\_max\_path\_distance(*morphology, root, node\_types*)

`neuron_morphology.features.path.calculate_max_path_distance` (*morphology*, *root*,  
*node\_types=None*)

Helper for `max_path_distance`. See below for more information.

`neuron_morphology.features.path.max_path_distance` (*data: MorphologyLike*, *node\_types: Optional[List[int]] = None*) → float

Calculate the distance, following the path of adjacent neurites, from the soma to the furthest compartment. This is equivalent to the distance to the furthest SWC node.

#### Parameters

**data** [the input reconstruction]

**node\_types** [if provided, restrict the calculation to nodes of these] types

#### Returns

**The along-path distance from the soma to the farthest (in the along-path sense) node.**

`neuron_morphology.features.path.early_branch_path` (*data: MorphologyLike*, *node\_types: Optional[List[int]] = None*, *soma: Optional[Dict] = None*) → float

Returns the ratio of the longest 'short' branch from a bifurcation to the maximum path length of the tree. In other words, for each bifurcation, the maximum path length below that branch is calculated, and the shorter of these values is used. The maximum of these short values is divided by the maximum path length.

#### Parameters

**data** [the input reconstruction]

**node\_types** [if provided, restrict the calculation to nodes of these] types

**soma** [if provided, use this node as the root, otherwise infer the root] from the argued morphology

#### Returns

**ratio of max short branch to max path length**

`neuron_morphology.features.path._calculate_mean_contraction` (*morphology*, *reference*, *root*,  
*node\_types*)

Calculate the average contraction of all sections. In other words, calculate the average ratio of euclidean distance to path distance between all bifurcations in the morphology. Trifurcations are treated as bifurcations.

#### Parameters

**morphology:** Morphology object

**reference:** dict

**This is the node of the previous bifurcation**

**root:** dict

**This is the node from which to measure branch contraction under**

**node\_types:** list (AXON, BASAL\_DENDRITE, APICAL\_DENDRITE)

**Type to restrict search to**

#### Returns

**Two scalars: euclidean distance, path distance**

These are the total bif-bif and bif-tip distances under this root

```
neuron_morphology.features.path.calculate_mean_contraction(morphology,  
                                                            root=None,  
                                                            node_types=None)
```

See `mean_contraction`

```
neuron_morphology.features.path.mean_contraction(data: MorphologyLike, node_types:  
                                                  Optional[List[int]] = None) → float
```

Calculate the average contraction of all sections. In other words, calculate the average ratio of euclidean distance to path distance between all bifurcations in the morphology. Trifurcations are treated as bifurcations.

#### Parameters

**data** [the input reconstruction]

**node\_types** [if provided, restrict the calculation to nodes of these] types

#### Returns

The average contraction across all sections in this reconstruction

`neuron_morphology.features.size`

## Module Contents

### Functions

<code>total_length</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Calculate the total length across all compartments in a reconstruction
<code>total_surface_area</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Calculates the sum of lateral surface areas across all compartments
<code>total_volume</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Calculates the sum of volumes across all compartments (linked pairs of
<code>mean_diameter</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Calculates the mean diameter of all nodes
<code>parent_daughter_ratio_visitor</code> (node: Dict[str, Any], morphology: Morphology, counters: Dict[str, Union[int, float]], node_types: Optional[List[int]] = None)	Calculates for a single node the ratio of the node's parent's radius to
<code>mean_parent_daughter_ratio</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Calculate the average ratio of parent radii to child radii across a
<code>max_euclidean_distance</code> (data: MorphologyLike, node_types: Optional[List[int]] = None) → float	Calculate the furthest distance, in 3-space, of a compartment's end from

```
neuron_morphology.features.size.total_length(data: MorphologyLike, node_types: Op-  
                                              tional[List[int]] = None) → float
```

Calculate the total length across all compartments in a reconstruction

#### Parameters

**data** [the input reconstruction]

**node\_types** [if provided, restrict the calculation to compartments] involving these types

#### Returns

**The sum of segment lengths across all segments in the reconstruction****Notes****Excludes compartments where the parent is:**

1. the soma
2. a root of the reconstruction

The logic here is that the soma root is likely to substantially overlap any of its compartments, while non-root soma nodes will be closer to the soma surface.

`neuron_morphology.features.size.total_surface_area` (*data: MorphologyLike, node\_types: Optional[List[int]] = None*) → float

Calculates the sum of lateral surface areas across all compartments (linked pairs of nodes) in a reconstruction. This approximates the total surface area of the reconstruction. See `Morphology.get_compartment_surface_area` for details.

**Parameters**

**data** [The reconstruction whose surface area will be computed]

**node\_types** [restrict the calculation to compartments involving these node] types

**Returns****The sum of compartment lateral surface areas across this reconstruction**

`neuron_morphology.features.size.total_volume` (*data: MorphologyLike, node\_types: Optional[List[int]] = None*) → float

Calculates the sum of volumes across all compartments (linked pairs of nodes) in a reconstruction. This approximates the total volume of the reconstruction. See `Morphology.get_compartment_volume` for details.

**Parameters**

**data** [The reconstruction whose volume will be computed]

**node\_types** [restrict the calculation to compartments involving these node] types

**Returns****The sum of compartment volumes across this reconstruction**

`neuron_morphology.features.size.mean_diameter` (*data: MorphologyLike, node\_types: Optional[List[int]] = None*) → float

Calculates the mean diameter of all nodes

**Parameters**

**morphology** [The reconstruction whose mean diameter]

**node\_types** [restrict the calculation to compartments involving these node] types

**Returns****The average diameter across selected nodes**

```
neuron_morphology.features.size.parent_daughter_ratio_visitor (node: Dict[str, Any], morphology: Morphology, counters: Dict[str, Union[int, float]], node_types: Optional[List[int]] = None)
```

Calculates for a single node the ratio of the node's parent's radius to the node's radius. Stores these values in a provided dictionary.

#### Parameters

**node** [The node under consideration]  
**morphology** [The reconstruction to which this node belongs]  
**counters** [a dictionary used for storing running ratio totals and counts.]  
**node\_types** [skip nodes not of one of these types]

#### Notes

see `mean_parent_daughter_ratio` for usage

```
neuron_morphology.features.size.mean_parent_daughter_ratio (data: Morphology-Like, node_types: Optional[List[int]] = None) → float
```

Calculate the average ratio of parent radii to child radii across a reconstruction.

#### Parameters

**data** [The reconstruction whose mean parent daughter ratio will be computed]  
**node\_types** [restrict the calculation to compartments involving these node] types

#### Notes

Note that this function differs from the L-measure parent daughter ratio, which calculates the ratio of the child node size to the parent. Note also that both the parent and child must be in `node_types` in order for a compartment to be included in the calculation

```
neuron_morphology.features.size.max_euclidean_distance (data: Morphology-Like, node_types: Optional[List[int]] = None) → float
```

Calculate the furthest distance, in 3-space, of a compartment's end from the soma. This is equivalent to the distance to the furthest SWC node.

#### Parameters

**data:** The reconstruction whose max euclidean distance will be calculated  
**node\_types:** restrict consideration to these types

#### Returns

The distance between the soma and the farthest-from-soma node in this

**morphology.**

**neuron\_morphology.features.soma**

## Module Contents

### Functions

<code>calculate_soma_surface(data: Data) → float</code>	Approximates the surface area of the soma. Morphologies with only
<code>calculate_relative_soma_depth(data: Data) → float</code>	Calculate the soma depth relative to pia/wm
<code>calculate_soma_features(data: Data)</code>	Calculate the soma features
<code>calculate_stem_exit_and_distance(data: Data, node_types: Optional[List[int]], z_scale=3.0)</code>	Returns the relative radial position (stem_exit) on the soma where the

`neuron_morphology.features.soma.calculate_soma_surface(data: Data) → float`  
Approximates the surface area of the soma. Morphologies with only a single soma node are supported.

#### Parameters

**data:** Data Object containing a morphology

#### Returns

Scalar value

`neuron_morphology.features.soma.calculate_relative_soma_depth(data: Data) → float`  
Calculate the soma depth relative to pia/wm

#### Parameters

**data:** Data Object containing a morphology

#### Returns

Scalar value

`neuron_morphology.features.soma.calculate_soma_features(data: Data)`  
Calculate the soma features

#### Parameters

**data:** Data Object containing a morphology

#### Returns

soma\_features

`neuron_morphology.features.soma.calculate_stem_exit_and_distance(data: Data, node_types: Optional[List[int]], z_scale=3.0)`  
Returns the relative radial position (stem\_exit) on the soma where the tree holding the tree connects to the soma. 0 is on the bottom, 1 on the top, and 0.5 out a side. Also returns the distance (stem\_distance) between the tree root and the soma surface.

#### Parameters

**data:** Data Object containing a morphology

**soma:** dict

**soma node**

**node\_types:** list (AXON, BASAL\_DENDRITE, APICAL\_DENDRITE)

Type to restrict search to

#### Returns

(float, float):

First value is relative position (height, on [0,1]) of a

tree on soma. Second value is distance of the root from soma

`neuron_morphology.layered_point_depths`

#### Submodules

`neuron_morphology.layered_point_depths.__main__`

#### Module Contents

#### Functions

<code>translate_field</code> (field: xr.DataArray, by_x: float, by_y: float, inplace: bool = False)	Translate a spatial xarray dataset
<code>setup_interpolator</code> (field: xr.DataArray, dim: Optional[str], **kwargs) → RegularGridInterpolator	Build a regular grid interpolator from a dataarray
<code>containing_layer</code> (pos: Tuple[float, float], layers: List[Dict]) → Optional[str]	Find the layer in which a point is contained
<code>tplize</code> (arr: np.array) → Tuple	Convert an array to a tuple
<code>step_from_node</code> (pos: Tuple[float, float], depth_interp: RegularGridInterpolator, dx_interp: RegularGridInterpolator, dy_interp: RegularGridInterpolator, surface: LineString, step_size: float, max_iter: int, adaptive_scale: int = 32) → Optional[float]	Walk through a gradient field, until a defined surface is passed.
<code>get_node_intersections</code> (node: Dict, depth_interp: RegularGridInterpolator, dx_interp: RegularGridInterpolator, dy_interp: RegularGridInterpolator, layers: List[Dict], step_size: float, max_iter: int) → Dict	Given a node, find its layer and intersection depths. Then return a row
<code>setup_layers</code> (layers: List[Dict])	Convert layer bounds, pia, and white matter surfaces to shapely objects
<code>run_layered_point_depths</code> (swc_path: str, depth: Dict, layers: List[Dict], step_size: float, max_iter: int, output_path: str)	
<code>main</code> ()	



```
neuron_morphology.layered_point_depths.__main__.translate_field(field:
                                                                xr.DataArray,
                                                                by_x:    float,
                                                                by_y:    float,
                                                                inplace: bool
                                                                = False)
```

Translate a spatial xarray dataset

#### Parameters

**field** [to be translated]  
**by\_x** [=the translation along x]  
**by\_y** [the translation along y]  
**inplace** [If True, modify this dataarray, otherwise modify a copy]

#### Returns

**translated dataarray, potentiall the same as the input**

```
neuron_morphology.layered_point_depths.__main__.setup_interpolator(field:
                                                                    xr.DataArray,
                                                                    dim:    Op-
                                                                    tional[str],
                                                                    **kwargs)
                                                                    → Regu-
                                                                    larGridIn-
                                                                    terpolator
```

Build a regular grid interpolator from a dataarray

#### Parameters

**field** [Must have dimensions “x” and “y”. May have dimension “dim”]  
**dim** [base the interpolator on values from this dim slice. If None, ignore] *dim*  
**\*\*kwargs** [passed to interpolator constructor]

#### Returns

**a callable interpolator**

```
neuron_morphology.layered_point_depths.__main__.containing_layer(pos:    Tu-
                                                                    ple[float,
                                                                    float], layers:
                                                                    List[Dict])
                                                                    → Op-
                                                                    tional[str]
```

Find the layer in which a point is contained

#### Parameters

**pos** [the coordinate of the point]  
**layers** [Each has “name” - a string and “bounds” - a Polygon]

#### Returns

**The name of the containing layer or None if no containing layer was found**

```
neuron_morphology.layered_point_depths.__main__.tupleize(arr: np.array) → Tuple
```

Convert an array to a tuple

```
neuron_morphology.layered_point_depths.__main__.step_from_node (pos:      Tuple[float, float],  
                                                                depth_interp: RegularGrid-  
                                                                Interpolator,  
                                                                dx_interp: RegularGrid-  
                                                                Interpolator,  
                                                                dy_interp: RegularGridInter-  
                                                                polator, surface: LineString,  
                                                                step_size: float,  
                                                                max_iter: int,  
                                                                adaptive_scale: int = 32) →  
                                                                Optional[float]
```

Walk through a gradient field, until a defined surface is passed.

#### Parameters

**pos** [the start position]

**depth\_interp** [callable mapping positions to scalar depth values]

**dx\_interp** [callable mapping positions to the x component of the gradient]

**dy\_interp** [callable mapping positions to the y component of the gradient]

**surface** [Check for the intersection of the path with this surface]

**step\_size** [Each step proceeds in the direction of the local gradient,] scaled to this step size

**max\_iter** [give up (return None) if the surface is not intersected in this] many steps

#### Returns

**The depth of the intersection between the path walked and the given surface**

```

neuron_morphology.layered_point_depths.__main__.get_node_intersections (node:
                                                                    Dict,
                                                                    depth_interp:
                                                                    Reg-
                                                                    u-
                                                                    larGrid-
                                                                    In-
                                                                    ter-
                                                                    po-
                                                                    la-
                                                                    tor,
                                                                    dx_interp:
                                                                    Reg-
                                                                    u-
                                                                    larGrid-
                                                                    In-
                                                                    ter-
                                                                    po-
                                                                    la-
                                                                    tor,
                                                                    dy_interp:
                                                                    Reg-
                                                                    u-
                                                                    larGrid-
                                                                    In-
                                                                    ter-
                                                                    po-
                                                                    la-
                                                                    tor,
                                                                    layers:
                                                                    List[Dict],
                                                                    step_size:
                                                                    float,
                                                                    max_iter:
                                                                    int)
                                                                    →
                                                                    Dict

```

Given a node, find its layer and intersection depths. Then return a row of LayeredPointDepths for this node.

#### Parameters

**node** [Of a Morphology. must have:] “id” - unique identifier “type” - which kind of node is this? “x”, “y” - positions in x and y of this node

**depth\_interp** [callable mapping positions to scalar depth values]

**dx\_interp** [callable mapping positions to the x component of the gradient]

**dy\_interp** [callable mapping positions to the y component of the gradient]

**layers** [Each has] “name” - an identifier “bounds” - a Polygon describing the entire boundary  
 “pia\_surface” - a LineString describing the piaward surface of this  
 layer

“wm\_surface” - a LineString describing the white matter-wise surface of this layer

**step\_size** [Each step proceeds in the direction of the local gradient,] scaled to this step size

**max\_iter** [give up (return None) if the surface is not intersected in this] many steps

### Returns

A dictionary representing a single row of **LayeredPointDepths**. Has Keys: “ids” - the identifier of this node “layer\_name” - the layer containing this node “depth” - the depth of this node “local\_layer\_pia\_side\_depth” - the depth of the intersection between

this node’s steepest ascent path and the piaward surface of its containing layer

“local\_layer\_wm\_side\_depth” - the depth of the intersection between this node’s steepest ascent path and the white matterward surface of its containing layer

“point\_type”: The type of this node

`neuron_morphology.layered_point_depths.__main__.setup_layers (layers: List[Dict])`  
Convert layer bounds, pia, and white matter surfaces to shapely objects

### Parameters

**layers** [Mutated inplace. Has keys:] “bounds” - a Polygon describing the entire boundary “pia\_surface” - a LineString describing the piaward surface of this

layer

“wm\_surface” - a LineString describing the white matter-wise surface of this layer

`neuron_morphology.layered_point_depths.__main__.run_layered_point_depths (swc_path: str, depth: Dict, layers: List[Dict], step_size: float, max_iter: int, output_path: str)`

`neuron_morphology.layered_point_depths.__main__.main()`

`neuron_morphology.layered_point_depths._schemas`

## Module Contents

### Classes

<i>DepthField</i>	mm.Schema class with support for making fields default to
-------------------	---

Continued on next page

Table 33 – continued from previous page

<i>Layer</i>	mm.Schema class with support for making fields default to
<i>InputParameters</i>	The base marshmallow schema used by ArgSchema-Parser to identify
<i>OutputParameters</i>	mm.Schema class with support for making fields default to

```
class neuron_morphology.layered_point_depths._schemas.DepthField (only=None,
                                                                    exclude=(),
                                                                    many=False,
                                                                    con-
                                                                    text=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False,
                                                                    un-
                                                                    known=None)
```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

**gradient\_field\_path**

**depth\_field\_path**

**soma\_origin**

**pia\_sign**

```
class neuron_morphology.layered_point_depths._schemas.Layer (only=None,
                                                                    exclude=(),
                                                                    many=False,
                                                                    context=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    partial=False,
                                                                    unknown=None)
```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

**name**

**bounds**

**pia\_surface**

**wm\_surface**

```
class neuron_morphology.layered_point_depths._schemas.InputParameters (only=None,
                                                                    ex-
                                                                    clude=(),
                                                                    many=False,
                                                                    con-
                                                                    text=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False,
                                                                    un-
                                                                    known=None)
```

Bases: `argschema.schemas.ArgSchema`

The base marshmallow schema used by ArgSchemaParser to identify input\_json and output\_json files and the log\_level

**swc\_path**

**depth**

**layers**

**step\_size**

**output\_path**

**max\_iter**

```
class neuron_morphology.layered_point_depths._schemas.OutputParameters (only=None,
                                                                    ex-
                                                                    clude=(),
                                                                    many=False,
                                                                    con-
                                                                    text=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False,
                                                                    un-
                                                                    known=None)
```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

**inputs**

**output\_path**

`neuron_morphology.pipeline`

**Submodules**

`neuron_morphology.pipeline._schemas`

**Module Contents**

## Classes

<i>PathResolution</i>	mm.Schema class with support for making fields default to
<i>PrimaryBoundaries</i>	mm.Schema class with support for making fields default to
<i>S3LandingBucket</i>	mm.Schema class with support for making fields default to
<i>InputParameters</i>	The base marshmallow schema used by ArgSchema-Parser to identify

```
class neuron_morphology.pipeline._schemas.PathResolution (only=None, exclude=(),
                                                         many=False, context=None, load_only=(),
                                                         dump_only=(), partial=False, unknown=None)
```

Bases: argschema.schemas.DefaultSchema

mm.Schema class with support for making fields default to values defined by that field's arguments.

**path**

**resolution**

```
class neuron_morphology.pipeline._schemas.PrimaryBoundaries (only=None,
                                                             exclude=(),
                                                             many=False,
                                                             context=None,
                                                             load_only=(),
                                                             dump_only=(),
                                                             partial=False,
                                                             unknown=None)
```

Bases: argschema.schemas.DefaultSchema

mm.Schema class with support for making fields default to values defined by that field's arguments.

**Soma**

**White\_Matter**

**Pia**

```
class neuron_morphology.pipeline._schemas.S3LandingBucket (only=None, exclude=(), many=False,
                                                             context=None,
                                                             load_only=(),
                                                             dump_only=(),
                                                             partial=False, unknown=None)
```

Bases: argschema.schemas.DefaultSchema

mm.Schema class with support for making fields default to values defined by that field's arguments.

**name**

**region**

**credentials\_file**

```
class neuron_morphology.pipeline._schemas.InputParameters (only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None)
```

Bases: `argschema.schemas.ArgSchema`

The base marshmallow schema used by ArgSchemaParser to identify input\_json and output\_json files and the log\_level

**destination\_bucket**

**neuron\_reconstruction\_id**

**specimen\_id**

**primary\_boundaries**

**swc\_file**

**cell\_depth**

**cut\_thickness**

**marker\_file**

**ccf\_soma\_xyz**

**slice\_transform**

**slice\_image\_flip**

`neuron_morphology.pipeline.post_data_to_s3`

## Module Contents

### Functions

<code>get_credentials(credentials_file)</code>	get credentisals from credentials_file
<code>zip_files(file_dict)</code>	zip files into an archive in memory
<code>post_object_to_s3(data, name, bucket, region, access_key_id=None, secret_access_key=None)</code>	post object (bytes) in memory to S3 bucket
<code>main()</code>	Usage:

`neuron_morphology.pipeline.post_data_to_s3.get_credentials(credentials_file)`  
get credentisals from credentials\_file

#### Parameters

**credentials\_file:** file path to credentials\_file

`neuron_morphology.pipeline.post_data_to_s3.zip_files(file_dict)`  
zip files into an archive in memory

#### Parameters

**file\_dict:** file name: file paths or file in bytes to be archived



```
neuron_morphology.pipeline.post_data_to_s3.post_object_to_s3(data, name,
                                                             bucket, region, ac-
                                                             cess_key_id=None,
                                                             se-
                                                             cret_access_key=None)
```

post object (bytes) in memory to S3 bucket

#### Parameters

**data:** the object data

**name:** the object data's name in s3

**region:** where the s3 bucket located

**bucket:** s3 bucket name or arn

**access\_key\_id, secret\_access\_key:** aws user's credentials implicitly the credentials file located in ~/.aws/credentials or set AWS\_SHARED\_CREDENTIALS\_FILE to the credentials file in your environment

```
neuron_morphology.pipeline.post_data_to_s3.main()
Usage: python post_data_to_s3.py -input_json INPUT_JSON
```

## neuron\_morphology.snap\_polygons

### Submodules

#### neuron\_morphology.snap\_polygons.\_\_main\_\_

An executable for finding close-fit boundaries between cortical layer polygons.

### Module Contents

#### Functions

<code>run_snap_polygons(layer_polygons, pia_surface, wm_surface, layer_order, working_scale: float, surface_distance_threshold: float, multipolygon_error_threshold: float, images=None)</code>	Finds and returns close fit boundaries. May write diagnostic images as
<code>main()</code>	CLI endpoint for snapping polygons

```
neuron_morphology.snap_polygons.__main__.run_snap_polygons(layer_polygons,
                                                             pia_surface,
                                                             wm_surface,
                                                             layer_order, work-
                                                             ing_scale: float, sur-
                                                             face_distance_threshold:
                                                             float, multipoly-
                                                             gon_error_threshold:
                                                             float, images=None)
```

Finds and returns close fit boundaries. May write diagnostic images as a side effect.

```
neuron_morphology.snap_polygons.__main__.main()
```

CLI entrypoint for snapping polygons

## neuron\_morphology.snap\_polygons.\_from\_lims

This module contains utilities for running snap\_polygons directly from the Allen Institute’s internal Laboratory Information Management System.

### Example Usage

```
python -m neuron_morphology.snap_polygons --host <lims host> --port <lims port> --user <username> --password
<password> --database <lims db> --focal_plane_image_series_id 522408212 # for instance --image_output_root
/some_directory
```

### Module Contents

#### Classes

<i>PostgresInputConfigSchema</i>	The parameters required to query a postgres database.
<i>FromLimsSchema</i>	The parameters required to query LIMS for a set of cortical layer
<i>FromLimsSource</i>	An alternate argschema source which gets its inputs from lims directly

#### Functions

<i>query_for_layer_polygons</i> (query_engine: QueryEngineType, focal_plane_image_series_id: int, validate_polys: bool = True, treatment: str = ‘Biocytin’) → List[Dict[str, Union[NicePathType, str]]]	Get all layer polygons for this image series
<i>query_for_cortical_surfaces</i> (query_engine: QueryEngineType, focal_plane_image_series_id: int) → Tuple[Dict[str, Union[NicePathType, str]], Dict[str, Union[NicePathType, str]]]	Return the pia and white matter surface drawings for this image series
<i>query_for_images</i> (query_engine: QueryEngineType, focal_plane_image_series_id: int, output_dir: str) → List[Dict[str, str]]	Return Biocytin and DAPI images associated with a focal plane image
<i>query_for_image_dims</i> (query_engine: QueryEngineType, focal_plane_image_series_id: int) → Tuple[float, float]	Find the dimensions of the Biocytin image associated with a focal plane
<i>get_inputs_from_lims</i> (host: str, port: int, database: str, user: str, password: str, imser_id: int, image_output_root: Optional[str])	Utility for building module inputs from a direct LIMS query

neuron\_morphology.snap\_polygons.\_from\_lims.**QueryEngineType**

```
neuron_morphology.snap_polygons._from_lims.query_for_layer_polygons(query_engine:
                                                                    QueryEngine-
                                                                    Type, fo-
                                                                    cal_plane_image_series_id:
                                                                    int, vali-
                                                                    date_polys:
                                                                    bool =
                                                                    True,
                                                                    treat-
                                                                    ment: str
                                                                    = 'Bio-
                                                                    cytin')
                                                                    →
                                                                    List[Dict[str,
                                                                    Union[NicePathType,
                                                                    str]]]
```

Get all layer polygons for this image series

#### Parameters

**query\_engine** [executes a query, passed in as a string. Must not require] any additional database information.

**focal\_plane\_image\_series\_id** [used to determine which polygons to fetch]

**validate\_polys** [if True, fail when]

- a label is associated with multiple distinct valid geometries
- a label is associated with one or more geometries, but none are valid

**treatment: The layer polygons are associated with Biocytin and DAPI** treatments. We only need one.

#### Returns

**A collection of labelled polygons.**

```
neuron_morphology.snap_polygons._from_lims.query_for_cortical_surfaces(query_engine:
                                                                    QueryEngine-
                                                                    Type, fo-
                                                                    cal_plane_image_series_id:
                                                                    int)
                                                                    →
                                                                    Tuple[Dict[str,
                                                                    Union[NicePathType,
                                                                    str]],
                                                                    Dict[str,
                                                                    Union[NicePathType,
                                                                    str]]]
```

Return the pia and white matter surface drawings for this image series

```
neuron_morphology.snap_polygons._from_lims.query_for_images(query_engine:
                                                                    QueryEngine-
                                                                    Type, fo-
                                                                    cal_plane_image_series_id:
                                                                    int, output_dir: str)
                                                                    → List[Dict[str, str]]
```

Return Biocytin and DAPI images associated with a focal plane image series

```
neuron_morphology.snap_polygons._from_lims.query_for_image_dims(query_engine:
                                                                    QueryEngine-
                                                                    Type, focal_
                                                                    cal_plane_image_series_id:
                                                                    int) → Tu-
                                                                    ple[float,
                                                                    float]
```

Find the dimensions of the Biocytin image associated with a focal plane image series

```
neuron_morphology.snap_polygons._from_lims.get_inputs_from_lims(host: str, port:
                                                                    int, database:
                                                                    str, user: str,
                                                                    password:
                                                                    str, imser_id:
                                                                    int, im-
                                                                    age_output_root:
                                                                    Optional[str])
```

Utility for building module inputs from a direct LIMS query

```
class neuron_morphology.snap_polygons._from_lims.PostgresInputConfigSchema(only=None,
                                                                              ex-
                                                                              clude=(),
                                                                              many=False,
                                                                              con-
                                                                              text=None,
                                                                              load_only=(),
                                                                              dump_only=(),
                                                                              par-
                                                                              tial=False,
                                                                              un-
                                                                              known=None)
```

Bases: `marshmallow.Schema`

The parameters required to query a postgres database.

**host**  
**database**  
**user**  
**password**  
**port**

```
class neuron_morphology.snap_polygons._from_lims.FromLimsSchema(only=None,
                                                                    exclude=(),
                                                                    many=False,
                                                                    context=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False, un-
                                                                    known=None)
```

Bases: `neuron_morphology.snap_polygons._from_lims.PostgresInputConfigSchema`

The parameters required to query LIMS for a set of cortical layer polygons and cortical surface boundaries.

**focal\_plane\_image\_series\_id**

`image_output_root`

**class** `neuron_morphology.snap_polygons._from_lims.FromLimsSource`

Bases: `argschema.sources.ArgSource`

An alternate argschema source which gets its inputs from lims directly

**ConfigSchema**

**get\_dict** (*self*)

`neuron_morphology.snap_polygons._schemas`

CLI schemas for the inputs to and outputs from `snap_polygons`.

## Module Contents

### Classes

<i>SimpleGeometry</i>	A named planar geometry
<i>Image</i>	A specification for a diagnostic overlay image
<i>InputParameters</i>	Top-level schema for inputs to <code>snap_polygons</code>
<i>OutputImage</i>	Metadata describing an output diagnostic overlay image.
<i>OutputParameters</i>	Top-level schema for <code>snap_polygons</code> outputs.

**class** `neuron_morphology.snap_polygons._schemas.SimpleGeometry` (*only=None*,  
*exclude=()*,  
*many=False*,  
*context=None*,  
*load\_only=()*,  
*dump\_only=()*,  
*partial=False*,  
*unknown=None*)

Bases: `argschema.schemas.DefaultSchema`

A named planar geometry

**name**

**path**

**class** `neuron_morphology.snap_polygons._schemas.Image` (*only=None*,  
*exclude=()*,  
*many=False*,  
*context=None*,  
*load\_only=()*,  
*dump\_only=()*,  
*partial=False*,  
*unknown=None*)

Bases: `argschema.schemas.DefaultSchema`

A specification for a diagnostic overlay image

**input\_path**

**output\_path**

**downsample**

**overlay\_types**

```
class neuron_morphology.snap_polygons._schemas.InputParameters (only=None,  
exclude=(),  
many=False,  
context=None,  
load_only=(),  
dump_only=(),  
partial=False, unknown=None)
```

Bases: `argschema.schemas.ArgSchema`

Top-level schema for inputs to snap\_polygons

**layer\_polygons**

**pia\_surface**

**wm\_surface**

**working\_scale**

**images**

**layer\_order**

**surface\_distance\_threshold**

**multipolygon\_error\_threshold**

```
class neuron_morphology.snap_polygons._schemas.OutputImage (only=None,  
exclude=(),  
many=False,  
context=None,  
load_only=(),  
dump_only=(),  
partial=False, unknown=None)
```

Bases: `argschema.schemas.DefaultSchema`

Metadata describing an output diagnostic overlay image.

**input\_path**

**output\_path**

**downsample**

**overlay\_type**

```
class neuron_morphology.snap_polygons._schemas.OutputParameters (only=None,  
exclude=(),  
many=False,  
context=None,  
load_only=(),  
dump_only=(),  
partial=False, unknown=None)
```

Bases: `argschema.schemas.DefaultSchema`

Top-level schema for snap\_polygons outputs.

**inputs**  
**polygons**  
**surfaces**  
**images**

`neuron_morphology.snap_polygons.bounding_box`

Contains a simple utility for representing a “growable” 2D rectangle

## Module Contents

### Classes

<i>BoundingBox</i>	Represents the bounds of a set of 2D objects
--------------------	--

```
class neuron_morphology.snap_polygons.bounding_box.BoundingBox (horigin: float,  

vorigin: float,  

hextent: float,  

vextent: float)
```

Represents the bounds of a set of 2D objects

#### Parameters

**vorigin, horigin** [the near corner of the box]

**vextent, hextent** [the far corner of the box]

```
__slots__ = ['vorigin', 'horigin', 'vextent', 'hextent']
```

**width :float**

Horizontal side length of the box.

**height :float**

Vertical side length of the box.

**aspect\_ratio :float**

Width / height ratio of the box.

**origin**

Coordinates of the box’s origin.

**extent**

Coordinates of the box’s extent (opposite corner from the origin).

**coordinates**

Origin and extent coordinates.

```
__repr__ (self)
```

Return repr(self).

```
update (self, horigin: float, vorigin: float, hextent: float, vextent: float)
```

Potentially enlarges this box.

#### Parameters

As to the constructor of **BoundingBox**. The new shape of this box is the smallest box enclosing both this and the inputs.

**transform** (*self*, *transform*: *TransformType*, *inplace*: *bool* = *False*)

Apply a transform to this box

#### Parameters

**transform** [A callable which maps (vertical, horizontal) coordinates to] new (vertical, horizontal) coordinates.

**inplace** [if True, apply the transform to this object]

#### Returns

The transformed box (potentially self)

**copy** (*self*)

Duplicates this bounding box

#### Returns

A copy of this object.

**round** (*self*, *inplace*: *bool* = *False*, *origin\_via*: *Callable*[[*float*], *float*] = *np.around*, *extent\_via*: *Callable*[[*float*], *float*] = *np.around*)

Round the coordinates of this box

#### Parameters

**inplace** [If True, round the coordinates of this object]

**origin\_via** [method to use when rounding the origin]

**extent\_via** [method to use when rounding the extent]

#### Returns

The rounded box (potentially self).

## neuron\_morphology.snap\_polygons.cortex\_surfaces

This module contains utilities for processing cortical surface drawings. In general we take these as given (they even take precedence of e.g. the upper and lower surfaces of layers 1 and 6b for instance), but some drawings pose resolvable problems.

The main such problem occurs when cortical layer drawings extend far from the layer drawings. Extrapolating layer drawings into this space is dangerous and not very useful (only the drawings near the cell are useful downstream). The solution implemented here is to cut out a segment of each surface whose endpoints are sufficiently close to the layer drawings and discard the rest.

## Module Contents

### Functions

---

<code>trim_to_close</code> ( <i>geometry</i> : <i>BaseGeometry</i> , <i>threshold</i> : <i>float</i> , <i>linestring</i> : <i>LineType</i> , <i>iterations</i> : <i>int</i> = 10) → <i>LineString</i>	Find the longest segment of a linestring whose endpoints are within a
---	---

---

Continued on next page



Table 41 – continued from previous page

<code>find_transition</code> ( <code>unmet</code> : Point, <code>met</code> : Point, <code>condition</code> : ConditionFn, <code>iterations</code> : int) → Point	Given two points in space, one of which meets a condition, locate the
<code>first_met</code> ( <code>coords</code> : Sequence[Union[Point, Tuple]], <code>condition</code> : ConditionFn, <code>iterations</code> : int) → Tuple[int, Point]	Locate the first point along a coordinate sequence at which a condition
<code>remove_duplicates</code> ( <code>coords</code> : Sequence[Point]) → Sequence[Point]	Remove duplicate points from a coordinate sequence.
<code>trim_coords</code> ( <code>coords</code> : Sequence[Union[Point, Tuple]], <code>condition</code> : ConditionFn, <code>iterations</code> : int)	Find the longest subinterval of a coordinate sequence whose endpoints

neuron\_morphology.snap\_polygons.cortex\_surfaces.**ConditionFn**

neuron\_morphology.snap\_polygons.cortex\_surfaces.**trim\_to\_close** (*geometry*:  
BaseGeometry, *threshold*:  
float, *linestring*:  
LineType, *iterations*: int = 10)  
→ LineString

Find the longest segment of a linestring whose endpoints are within a specified distance of a geometry.

#### Parameters

- geometry** [Acceptable distances are defined as extending from this object.]
- threshold** [Acceptable distances are less than or equal to this value]
- linestring** [to be trimmed (not in place)]
- iterations** [Use this many iterations to refine the endpoints of the] linestring

#### Returns

a trimmed copy of the input linestring

neuron\_morphology.snap\_polygons.cortex\_surfaces.**find\_transition** (*unmet*: Point,  
*met*: Point,  
*condition*:  
ConditionFn,  
*iterations*:  
int) → Point

Given two points in space, one of which meets a condition, locate the position along a line segment between these points where the condition becomes true.

#### Parameters

- unmet** [a point at which the condition is not met]
- met** [a point at which the condition is met]
- condition** [used to evaluate intermediate points]
- iterations** [refine this many times]

#### Returns

A point along the input segment at which the condition is met.

## Notes

No such transition point is required to exist. In that case, this function will find an arbitrary condition-meeting point along the segment. For our use case, this misbehavior is tolerable because an exact transition point is not required.

```
neuron_morphology.snap_polygons.cortex_surfaces.first_met (coords:      Se-  
                                                             quence[Union[Point,  
                                                             Tuple]],      condition:  
                                                             ConditionFn,  itera-  
                                                             tions: int) → Tuple[int,  
                                                             Point]
```

Locate the first point along a coordinate sequence at which a condition is met.

### Parameters

**coords** [sequence to evaluate]

**condition** [used to evaluate points]

**iterations** [how many times to refine the transition point.]

### Returns

**The index and value of the transition point.**

```
neuron_morphology.snap_polygons.cortex_surfaces.remove_duplicates (coords: Se-  
                                                                    quence[Point])  
                                                                    →      Se-  
                                                                    quence[Point]
```

Remove duplicate points from a coordinate sequence.

### Parameters

**coords** [sequence with potential duplicates]

### Returns

**list of coordinates with duplicates removed**

```
neuron_morphology.snap_polygons.cortex_surfaces.trim_coords (coords:      Se-  
                                                                quence[Union[Point,  
                                                                Tuple]],      condi-  
                                                                tion:  ConditionFn,  
                                                                iterations: int)
```

Find the longest subinterval of a coordinate sequence whose endpoints meet some condition.

### Parameters

**coords** [sequence to trim]

**condition** [used to evaluate points]

**iterations** [how many times to refine the endpoints.]

### Returns

**Trimmed sequence**

**neuron\_morphology.snap\_polygons.geometries**

A collection of utilities used by snap polygons to manipulate shapely objects.

## Module Contents

### Classes

<i>Geometries</i>	A collection of polygons and lines
-------------------	------------------------------------

### Functions

<i>select_largest_subpolygon</i> (polygons: Union[Polygon, Iterable[Polygon]], error_threshold: float) → Polygon	Given a collection of polygons, find the largest by area.
<i>safe_linemerge</i> (linestrings: Union[LineString, Sequence[LineString]]) → LineString	Wrapper around shapely.ops.linemerge that no-ops in case a single
<i>rasterize</i> (geometry: shapely.geometry.base.BaseGeometry, box: BoundingBox) → np.array	Rasterize a shapely object to a grid defined by a provided bounding box.
<i>make_translation</i> (horizontal: float, vertical: float) → TransformType	Utility for building a 2D translation transform
<i>make_scale</i> (scale: float = 1.0) → TransformType	A utility for making a 2D scale transform, suitable for transforming
<i>clear_overlaps</i> (stack: Dict[str, np.ndarray])	Given a stack of masks, remove all inter-mask overlaps inplace
<i>closest_from_stack</i> (stack: Dict[str, np.ndarray])	Given a stack of images describing distance from several objects, find
<i>get_snapped_polys</i> (closest: np.ndarray, name_lut: Dict[int, str], multipolygon_resolver: MultiPolygonResolverType) → Dict[str, Polygon]	Obtains named shapes from a label image.
<i>find_vertical_surfaces</i> (polygons: Dict[str, Polygon], order: Sequence[str], pia: Optional[LineString] = None, white_matter: Optional[LineString] = None)	Given a set of polygons describing cortical layer boundaries, find the
<i>shared_faces</i> (poly: Polygon, others: Iterable[Polygon], snap_tolerance=10) → LineString	Given a polygon and a set of other polygons that could be adjacent on

neuron\_morphology.snap\_polygons.geometries.**select\_largest\_subpolygon** (*polygons:*  
*Union[Polygon,*  
*Iterable[Polygon]],*  
*error\_threshold:*  
*float*)  
 →  
 Polygon

Given a collection of polygons, find the largest by area.

#### Parameters

**polygons** [To be filtered]

**error\_threshold** [If the ratio of the largest polygon to the second] largest does not meet or exceed this value, reject the largest polygon.

**Returns****the largest polygon**

`neuron_morphology.snap_polygons.geometries.safe_linemerge` (*linestrings:*  
*Union[LineString, Sequence[LineString]]*  
*→ LineString*)

Wrapper around `shapely.ops.linemerge` that no-ops in case a single `LineString` or length-1 collection is argued.

**class** `neuron_morphology.snap_polygons.geometries.Geometries`

A collection of polygons and lines

**default\_multipolygon\_resolver**

By default, multiple polygons resulting from operations on these geometries are resolved by discarding all but the largest

**default\_multisurface\_resolver**

By default, multiple surfaces arising from operations on these geometries are merged back together (failing if this is not possible).

**close\_bounds**

The smallest bounding box enclosing these geometries.

**register\_polygon** (*self, name: str, path: PolyType*)

Adds a named polygon path to this object. Updates the close bounding box.

**Parameters**

**name** [identifier for this polygon]

**path** [defines the exterior of this (simple) polygon]

**\_register\_many** (*self, objects: Union[Dict[str, Union[LineType, PolyType]], Sequence[Dict[str, Union[LineType, PolyType]]], method: Callable[[str, Union[LineType, PolyType]], None]*)

Utility for registering many polygons or surfaces. See `register_polygons` and `register_surfaces` for use.

**register\_polygons** (*self, polygons: Union[Dict[str, PolyType], Sequence[Dict[str, PolyType]]]*)

utility for registering multiple polygons. See `register_polygon`

**register\_surface** (*self, name: str, path: LineType*)

Adds a line (e.g. the pia/wm surfaces) to this object. Updates the bounding box.

**Parameters**

**name** [identifier for this surface]

**path** [defines the surface]

**register\_surfaces** (*self, surfaces: Dict[str, LineType]*)

utility for registering multiple surfaces. See `register_surface`

**rasterize** (*self, box: Optional[BoundingBox] = None, polygons: Union[Sequence[str], bool] = True, surfaces: Union[Sequence[str], bool] = False*)

Rasterize one or more owned geometries. Produce a mapping from object names to masks.

**Parameters**

**shape** [if provided, the output image shape. Otherwise, use the] rounded close bounding box shape

**polygons** [a list of names. Alternatively all (True) or none (False)]

**lines** [a list of names. Alternatively all (True) or none (False)]

## Notes

uses rasterio.features.rasterize

**transform** (*self*, *transform*: *TransformType*)

Apply a transform to each owned geometry. Return a new collection.

### Parameters

**transform** [A callable which maps (vertical, horizontal) coordinates to] new (vertical, horizontal) coordinates.

**fill\_gaps** (*self*, *working\_scale*: *float* = *1.0*, *multipolygon\_resolver*: *Optional[MultiPolygonResolverType]* = *None*)

Expand this geometries' polygons to fill its bounding box, using distance to assign empty space.

### Parameters

**working\_scale** [The filling is carried out in a raster space, with 1] pixel corresponding to 1 unit in the coordinate system of your polygons. You can optionally rescale the polygons before rasterizing.

**multipolygon\_resolver** [This method might obtain multiple output] polygons for a given input polygon. This callable collapses them into a single geometry. The default selects the largest.

### Returns

**A copy of this geometries object with the entire bounding box having been filled.**

**cut** (*self*, *template*: *shapely.geometry.Polygon*, *multipolygon\_resolver*: *Optional[MultiPolygonResolverType]* = *None*, *multisurface\_resolver*: *Optional[MultiSurfaceResolvertype]* = *None*)

Crop this Geometries' polygons and surfaces onto a provided template.

### Parameters

**template** [portions of surfaces and polygons outside this shape will be] removed

**multipolygon\_resolver** [This callable is applied to the outputs of] the intersection operation in order to resolve cases where a polygon has been cut into multiple components. The default method selects the largest by area.

**multisurface\_resolver** [As multipolygon resolver, for surfaces. The] default method attempts to merge the surfaces.

### Returns

**A copy of this Geometries object, with polygons and surfaces cropped**

**convex\_hull** (*self*, *surfaces*: *bool* = *True*, *polygons*: *bool* = *True*)

Find the convex hull of these geometries.

### Parameters

**surfaces** [if True, include surfaces in the hull]

**polygons** [if True, include polygons in the hull]

### Returns

**The convex hull of the included geometries**

**to\_json** (*self*)

Write contained polygons to a json-serializable format

`neuron_morphology.snap_polygons.geometries.rasterize` (*geometry*:  
*shapely.geometry.base.BaseGeometry*,  
*box*: *BoundingBox*) →  
*np.array*

Rasterize a shapely object to a grid defined by a provided bounding box.

#### Parameters

**geometry** [to be rasterized]

**box** [defines the window (in the same coordinate space as the geometry)] into which the geometry will be rasterized

#### Returns

**A mask, where 1 indicates presence and 0 absence**

`neuron_morphology.snap_polygons.geometries.make_translation` (*horizontal*: *float*,  
*vertical*: *float*) →  
*TransformType*

Utility for building a 2D translation transform

#### Parameters

**horizontal** [translate by this much along the first axis]

**vertical** [translate by this much along the second axis]

#### Returns

**Function which applies the argued translation**

`neuron_morphology.snap_polygons.geometries.make_scale` (*scale*: *float* = 1.0) → *TransformType*

A utility for making a 2D scale transform, suitable for transforming bounding boxes and Geometries

#### Parameters

**scale** [isometric scale factor]

#### Returns

**A transform function**

`neuron_morphology.snap_polygons.geometries.clear_overlaps` (*stack*: *Dict[str, np.ndarray]*)

Given a stack of masks, remove all inter-mask overlaps inplace

#### Parameters

**stack** [Keys are names, values are masks (of the same shape). 0 indicates] absence

`neuron_morphology.snap_polygons.geometries.closest_from_stack` (*stack*: *Dict[str, np.ndarray]*)

Given a stack of images describing distance from several objects, find the closest object to each pixel.

#### Parameters

**stack** [Keys are names, values are ndarrays (of the same shape). Each pixel] in the values describes the distance from that pixel to the named object

#### Returns

**closest** [An integer array whose values are the closest object to each] pixel

**names** [A mapping from the integer codes in the “closest” array to names]

```
neuron_morphology.snap_polygons.geometries.get_snapped_polys (closest:
                                                                np.ndarray,
                                                                name_lut: Dict[int,
                                                                str],      multipoly-
                                                                gon_resolver:
                                                                MultiPolygonRe-
                                                                solverType)      →
                                                                Dict[str, Polygon]
```

Obtains named shapes from a label image.

#### Parameters

**closest** [label integer with integer codes]

**name\_lut** [look up table from integer codes to string names]

#### Returns

**mapping from names to polygons describing each labelled region**

```
neuron_morphology.snap_polygons.geometries.find_vertical_surfaces (polygons:
                                                                    Dict[str,
                                                                    Polygon],
                                                                    order: Se-
                                                                    quence[str],
                                                                    pia: Op-
                                                                    tional[LineString]
                                                                    = None,
                                                                    white_matter:
                                                                    Op-
                                                                    tional[LineString]
                                                                    = None)
```

Given a set of polygons describing cortical layer boundaries, find the boundaries between each layer.

#### Parameters

**polygons** [named layer polygons]

**order** [A sequence of names defining the order of the layer polygons from] pia to white matter

**pia** [The upper (from the perspective of cortex) pia surface.]

**white\_matter** [The lower (from the perspective of cortex) white matter] surface.

#### Returns

**dictionary whose keys are as “{name}\_{side}” and whose values are** linestrings describing these boundaries.

```
neuron_morphology.snap_polygons.geometries.shared_faces (poly: Polygon, oth-
                                                            ers: Iterable[Polygon],
                                                            snap_tolerance=10)      →
                                                            LineString
```

Given a polygon and a set of other polygons that could be adjacent on the same side, find and connect that shared face.

#### Parameters

**poly** [Polygon] Polygon whose boundary with others we want to identify

**others** [list] List of other Polygons

**Returns**

LineString representing the shared face

`neuron_morphology.snap_polygons.image_outputter`

Utilites for writing diagnostic overlay images

**Module Contents**

**Classes**

<i>ImageOutputter</i>	Overlays polygons and surfaces on provided images. Writes the
-----------------------	---

**Functions**

<i>write_figure</i> (fig: plt.Figure, *args, **kwargs)	Write a matplotlib figure without respect to the current figure.
<i>read_image</i> (path: str, decimate: int = 1)	Read an image. Dispatch to an appropriate library based on that
<i>read_with_ndimage</i> (path: str, decimate: int)	Read (and symmetrically decimate) an image file into a numpy array
<i>read_jp2</i> (path: str, decimate: int)	Read (and symmetrically decimate) a jp2 file into a numpy array
<i>fname_suffix</i> (path: str, suffix: str)	Utility for adding a suffix to a path string. The suffix will be
<i>make_pathpatch</i> (vertices: Sequence[Tuple[float, float]], **patch_kwargs) → mpl.patches.PathPatch	Utility for building a matplotlib pathpatch from an array of vertices



```

class neuron_morphology.snap_polygons.image_outputter.ImageOutputter (native_geo:
    Ge-
    ome-
    tries,
    re-
    sult_geo:
    Ge-
    ome-
    tries,
    im-
    age_specs:
    Op-
    tional[Sequence[Dict]],
    alpha:
    float
    = 0.4,
    color_cycle:
    Op-
    tional[Sequence]
    =
    None,
    save-
    fig_kwargs:
    Op-
    tional[Dict]
    =
    None)

```

Overlays polygons and surfaces on provided images. Writes the results to files.

#### Parameters

**native\_geo** [Layer geometries before gaps are filled]

**result\_geo** [Layer geometries after gaps are filled]

**image\_specs** [Each is a dictionary defining a single image. Must]

**provide string keys:**

- **input\_path** : read from here
- **output\_path** : write to (siblings of) this path
- **downsample** [the image will be scaled by this factor in each] dimension
- **overlay\_types** : produce these kinds of overlay for this image

**alpha** [of the transparent overlays]

**color\_cycle** [as polygon fills are drawn, cycle through these colors]

**savefig\_kwargs** [Passed directly to pyplot's savefig, use to specify] e.g dpi.

**DEFAULT\_COLOR\_CYCLE** = ['c', 'm', 'y', 'k', 'r', 'g', 'b']

#### OVERLAY\_TYPES

**\_draw\_geometries** (*self, geometries: Geometries, image: np.ndarray*)

Utility for overlaying polygons and surfaces on an image. See `draw_before` and `draw_after` for more details.

**draw\_before** (*self*, *image*: *np.ndarray*, *scale*: *float = 1.0*)

Display the pre-fill polygons and surfaces overlaid on an image.

**Parameters**

**image** [onto which objects will be drawn]

**scale** [required to transform from object space to image space]

**Returns**

A matplotlib figure containing the overlay

**draw\_after** (*self*, *image*: *np.ndarray*, *scale*: *float = 1.0*)

Display the post-fill polygons and surfaces overlaid on an image.

**Parameters**

**image** [onto which objects will be drawn]

**scale** [required to transform from object space to image space]

**Returns**

A matplotlib figure containing the overlay

**write\_images** (*self*)

For each image specified in this outputter and each overlay type requested for that image, produce and save an overlay.

`neuron_morphology.snap_polygons.image_outputter.write_figure` (*fig*: *plt.Figure*,  
\**args*, \*\**kwargs*)

Write a matplotlib figure without respect to the current figure.

**Parameters**

**fig** [the figure to be written]

\**args*, \*\**kwargs* [passed to plt.savefig]

`neuron_morphology.snap_polygons.image_outputter.read_image` (*path*: *str*, *decimate*:  
*int = 1*)

Read an image. Dispatch to an appropriate library based on that image's extension.

**Parameters**

**path** [to the image]

**decimate** [apply a decimation of this factor along each axis of the image]

`neuron_morphology.snap_polygons.image_outputter.read_with_ndimage` (*path*: *str*,  
*decimate*:  
*int*)

Read (and symmetrically decimate) an image file into a numpy array

`neuron_morphology.snap_polygons.image_outputter.read_jp2` (*path*: *str*, *decimate*: *int*)

Read (and symmetrically decimate) a jp2 file into a numpy array

`neuron_morphology.snap_polygons.image_outputter.fname_suffix` (*path*: *str*, *suffix*:  
*str*)

Utility for adding a suffix to a path string. The suffix will be inserted before the extension.

```
neuron_morphology.snap_polygons.image_outputter.make_pathpatch(vertices: Sequence[Tuple[float, float]],
                                                                **patch_kwargs)
→
mpl.patches.PathPatch
```

Utility for building a matplotlib pathpatch from an array of vertices

#### Parameters

**vertices** [Defines the path. May be closed or open]

**\*\*patch\_kwargs** [passed directly to pathpatch constructor]

## neuron\_morphology.snap\_polygons.types

### Module Contents

#### Functions

<code>ensure_polygon(candidate: PolyType) → Polygon</code>	Convert from one of many polygon representations to Polygon
<code>ensure_linestring(candidate: LineType) → LineString</code>	Convert from one of many line representations to LineString
<code>ensure_path(candidate: PathType, num_dims: int = 2) → NicePathType</code>	Ensure that an input path, which might be a “x,y,x,y” string, is
<code>split_pathstring(pathstring: str, num_dims: int = 2, sep: str = ‘,’) → NicePathType</code>	Converts a pathstring (“x,y,x,y...” ) to a num_points X num_dims

neuron\_morphology.snap\_polygons.types.NicePathType

neuron\_morphology.snap\_polygons.types.PathType

neuron\_morphology.snap\_polygons.types.PathsType

neuron\_morphology.snap\_polygons.types.PolyType

neuron\_morphology.snap\_polygons.types.LineType

neuron\_morphology.snap\_polygons.types.TransformType

neuron\_morphology.snap\_polygons.types.MultiPolygonResolverType

neuron\_morphology.snap\_polygons.types.MultiSurfaceResolvertype

neuron\_morphology.snap\_polygons.types.ensure\_polygon(*candidate: PolyType*) → Polygon

Convert from one of many polygon representations to Polygon

neuron\_morphology.snap\_polygons.types.ensure\_linestring(*candidate: LineType*) → LineString

Convert from one of many line representations to LineString

neuron\_morphology.snap\_polygons.types.ensure\_path(*candidate: PathType, num\_dims: int = 2*) → NicePathType

Ensure that an input path, which might be a “x,y,x,y” string, is represented as a list of lists instead.

#### Parameters

**candidate** [input coordinate sequence]  
**num\_dims** [how manu elements define a coordinate]

#### Returns

**Contents of inputs, with each coordinate a list of float**

`neuron_morphology.snap_polygons.types.split_pathstring` (*pathstring: str, num\_dims: int = 2, sep: str = ', ' → NicePathType*)

Converts a pathstring (“x,y,x,y...”) to a num\_points X num\_dims list of lists of float

#### Parameters

**pathstring** [input coordinate sequence]  
**num\_dims** [how manu elements define a coordinate]  
**sep** [character separating elements]

#### Returns

**Contents of pathstring, with each coordinate a list of float**

`neuron_morphology.transforms`

#### Subpackages

`neuron_morphology.transforms.affine_transformer`

#### Submodules

`neuron_morphology.transforms.affine_transformer._schemas`

#### Module Contents

#### Classes

<i>AffineDictSchema</i>	affine_dict: keys and values corresponding to the following
<i>ApplyAffineSchema</i>	Arg Schema for apply_affine_transform module
<i>OutputParameters</i>	mm.Schema class with support for making fields default to

#### Functions

`validate_input_affine(data)`

`neuron_morphology.transforms.affine_transformer._schemas.validate_input_affine(data)`

```

class neuron_morphology.transforms.affine_transformer._schemas.AffineDictSchema (only=None,
ex-
clude=(),
many=False,
con-
text=None,
load_only=(),
dump_only=(),
par-
tial=False,
un-
known=None)

```

Bases: `argschema.schemas.DefaultSchema`

**affine\_dict:** keys and values corresponding to the following

```
[[tvr_00 tvr_01 tvr_02 tvr_09] [tvr_03 tvr_04 tvr_05 tvr_10] [tvr_06 tvr_07 tvr_08 tvr_11] [0 0 0 1]]
```

**tvr\_00**

**tvr\_01**

**tvr\_02**

**tvr\_03**

**tvr\_04**

**tvr\_05**

**tvr\_06**

**tvr\_07**

**tvr\_08**

**tvr\_09**

**tvr\_10**

**tvr\_11**

```

class neuron_morphology.transforms.affine_transformer._schemas.ApplyAffineSchema (only=None,
ex-
clude=(),
many=False,
con-
text=None,
load_only=(),
dump_only=(),
par-
tial=False,
un-
known=None)

```

Bases: `argschema.ArgSchema`

Arg Schema for `apply_affine_transform` module

**affine\_dict**

**affine\_list**

**input\_swc**

```
    output_swc
    validate_schema_input (self, data, **kwargs)
class neuron_morphology.transforms.affine_transformer._schemas.OutputParameters (only=None,
ex-
clude=(),
many=False,
con-
text=None,
load_only=(),
dump_only=(),
par-
tial=False,
un-
known=None)

Bases: argschema.schemas.DefaultSchema
mm.Schema class with support for making fields default to values defined by that field's arguments.

inputs
transformed_swc
```

```
neuron_morphology.transforms.affine_transformer.apply_affine_transform
```

Module Contents

Functions

---

<code>main()</code>
---------------------

---

```
neuron_morphology.transforms.affine_transformer.apply_affine_transform.main()
```

```
neuron_morphology.transforms.pia_wm_streamlines
```

Submodules

```
neuron_morphology.transforms.pia_wm_streamlines._schemas
```

Module Contents

Classes

<code>PiaWmStreamlineSchema</code>	Arg Schema for run_pia_wm_streamlines
<code>OutputParameters</code>	mm.Schema class with support for making fields default to

---

```

class neuron_morphology.transforms.pia_wm_streamlines._schemas.PiaWmStreamlineSchema (only=None,
ex-
clude=(),
many=False,
con-
text=None,
load_only=(),
dump_only=(),
par-
tial=False,
un-
known=

```

Bases: `argschema.ArgSchema`

Arg Schema for `run_pia_wm_streamlines`

`pia_path_str`

`wm_path_str`

`soma_path_str`

`resolution`

`pia_fixed_value`

`wm_fixed_value`

`mesh_res`

`output_dir`

```

class neuron_morphology.transforms.pia_wm_streamlines._schemas.OutputParameters (only=None,
ex-
clude=(),
many=False,
con-
text=None,
load_only=(),
dump_only=(),
par-
tial=False,
un-
known=None)

```

Bases: `argschema.schemas.DefaultSchema`

`mm.Schema` class with support for making fields default to values defined by that field's arguments.

`inputs`

`depth_field_file`

`gradient_field_file`

`translation`

`neuron_morphology.transforms.pia_wm_streamlines.calculate_pia_wm_streamlines`

## Module Contents

## Functions

---

```
convert_path_str_to_list(path_str: str, resolution: float = 1.0) → List[Tuple[float, float]]
```

---

```
run_streamlines(pia_path_str: str, wm_path_str: str, resolution: float, soma_path_str: Optional[str] = None, mesh_res: int = 20, pia_fixed_value: float = 1.0, wm_fixed_value: float = 0.0)
```

---

```
main()
```

---

```
neuron_morphology.transforms.pia_wm_streamlines.calculate_pia_wm_streamlines.convert_path_s
```

```
neuron_morphology.transforms.pia_wm_streamlines.calculate_pia_wm_streamlines.run_streamline
```

```
neuron_morphology.transforms.pia_wm_streamlines.calculate_pia_wm_streamlines.main()
```



`neuron_morphology.transforms.scale_correction`

## Submodules

`neuron_morphology.transforms.scale_correction._schemas`

## Module Contents

### Classes

<i><code>InputParameters</code></i>	The base marshmallow schema used by ArgSchemaParser to identify
<i><code>OutputParameters</code></i>	mm.Schema class with support for making fields default to

```
class neuron_morphology.transforms.scale_correction._schemas.InputParameters (only=None,
ex-
clude=(),
many=False,
con-
text=None,
load_only=(),
dump_only=(),
par-
tial=False,
un-
known=None)
```

Bases: `argschema.schemas.ArgSchema`

The base marshmallow schema used by ArgSchemaParser to identify input\_json and output\_json files and the log\_level

**swc\_path**

**marker\_path**

**soma\_depth**

**cut\_thickness**

```
class neuron_morphology.transforms.scale_correction._schemas.OutputParameters (only=None,
ex-
clude=(),
many=False,
con-
text=None,
load_only=(),
dump_only=(),
par-
tial=False,
un-
known=None)
```

Bases: `argschema.schemas.DefaultSchema`

mm.Schema class with support for making fields default to values defined by that field's arguments.

```
inputs
scale_correction
scale_transform
```

```
neuron_morphology.transforms.scale_correction.compute_scale_correction
```

## Module Contents

### Functions

---

<code>estimate_scale_correction(morphology:</code>	Estimate a scale factor to correct the reconstructed mor-
<code>Morphology, soma_depth: float, soma_marker_z: float,</code>	phology
<code>cut_thickness: Optional[float] = 350)</code>	

---

<code>get_soma_marker_from_marker_file(marker_path:</code>
<code>str)</code>

---

<code>run_scale_correction(morphology: Morphol-</code>
<code>ogy, soma_marker_z: float, soma_depth: float,</code>
<code>cut_thickness: float)</code>

---

<code>collect_inputs(args: Dict[str, Any]) → Dict[str,</code>
<code>Any]</code>

---

#### Parameters

<code>main()</code>
---------------------

---

```
neuron_morphology.transforms.scale_correction.compute_scale_correction.estimate_scale_corre
```

Estimate a scale factor to correct the reconstructed morphology for slice shrinkage

Prior to reconstruction, the slice shrinks due to evaporation. This is most notable in the z axis, which is the slice thickness.

To correct for shrinkage we compare soma depth within the slice obtained soon after cutting the slice to the fixed\_soma\_depth obtained during the reconstruction. Then the scale correction is estimated as:  $scale = soma\_depth / fixed\_soma\_depth$ . This is sensible as long as the z span of the corrected reconstruction is contained within the slice thickness. Thus we also estimate the maximum scale correction as:  $scale\_max = cut\_thickness / z\_span$ , and take the smaller of scale and scale\_max

#### Parameters

**morphology:** Morphology object

**soma\_depth:** recorded depth of the soma when it was sliced

**soma\_marker\_z:** soma marker z value from revised marker file (z is on the slice surface for the marker file)

**cut\_thickness:** thickness of the cut slice

### Returns

**scale factor correction**

`neuron_morphology.transforms.scale_correction.compute_scale_correction.get_soma_marker_from`

`neuron_morphology.transforms.scale_correction.compute_scale_correction.run_scale_correction`

`neuron_morphology.transforms.scale_correction.compute_scale_correction.collect_inputs` (*args: Dict[str, Any]*)  
 →  
*Dict[str, Any]*

### Parameters

**args:** dict of InputParameters

### Returns

**dict with string keys:** morphology: Morphology object soma\_marker\_z: z value from the marker file soma\_depth: soma depth cut\_thickness: slice thickness

`neuron_morphology.transforms.scale_correction.compute_scale_correction.main()`

`neuron_morphology.transforms.tilt_correction`

### Submodules

`neuron_morphology.transforms.tilt_correction._schemas`

### Module Contents

### Classes

<i>InputParameters</i>	The base marshmallow schema used by ArgSchema-Parser to identify
<i>OutputParameters</i>	mm.Schema class with support for making fields default to

## Functions

---

`validate_input_affine(data)`

---

`neuron_morphology.transforms.tilt_correction._schemas.validate_input_affine(data)`

**class** neuron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters (*only=None, exclude=(), many=False, con-text=None, load\_only=(), dump\_only=(), par-tial=False, un-known=None*)

Bases: argschema.schemas.ArgSchema

The base marshmallow schema used by ArgSchemaParser to identify input\_json and output\_json files and the log\_level

**swc\_path**

**marker\_path**

**slice\_image\_flip**

**ccf\_soma\_location**

**slice\_transform\_list**

**slice\_transform\_dict**

**ccf\_path**

**validate\_schema\_input** (*self, data, \*\*kwargs*)

**class** neuron\_morphology.transforms.tilt\_correction.\_schemas.OutputParameters (*only=None, exclude=(), many=False, con-text=None, load\_only=(), dump\_only=(), par-tial=False, un-known=None*)

Bases: argschema.schemas.DefaultSchema

mm.Schema class with support for making fields default to values defined by that field's arguments.

**inputs**

**tilt\_correction**

**tilt\_transform\_dict**

`neuron_morphology.transforms.tilt_correction.compute_tilt_correction`

## Module Contents

### Functions

<code>get_tilt_correction</code>	(morphology: Morphology, soma_voxel: List[int], slice_angle_matrix: float, closest_path)	Find the tilt angle between the slice plane and the nearest streamline
<code>find_closest_path</code>	(soma_voxel: List[int], ccf_path: Union[str, IO], n_sublists: int = 2)	Finds closest path to soma_voxel
<code>determine_slice_flip</code>	(morphology: Morphology, soma_marker: Dict, slice_image_flip: bool)	Determines whether the tilt correction should be positive or negative
<code>read_soma_marker</code>	(marker_path: str)	
<code>run_tilt_correction</code>	(morphology: Morphology, soma_marker: Dict, ccf_soma_location: Dict, slice_transform: aff.AffineTransform, slice_image_flip: bool, ccf_path: Union[str, IO])	
<code>main</code>	()	

`neuron_morphology.transforms.tilt_correction.compute_tilt_correction.CCF_SHAPE = [1320, 800]`

`neuron_morphology.transforms.tilt_correction.compute_tilt_correction.CCF_RESOLUTION = 10`

`neuron_morphology.transforms.tilt_correction.compute_tilt_correction.get_tilt_correction` (morphology: Morphology, soma\_voxel: List[int], slice\_angle\_matrix: float, closest\_path: Union[str, IO], n\_sublists: int = 2)

Find the tilt angle between the slice plane and the nearest streamline

#### Parameters

**morphology:** Morphology object

**soma\_voxel:** soma voxel in ccf {'x': , 'y': , 'z': }

**slice\_angle\_matrix:** 4 x 4 affine matrix of the slice plane relative to ccf

**closest\_path:** 3 x N array of voxel coordinates in closest streamline, only first (wm end) and last (pia end) coordinates are used. In future the tilt correction may be refined to use the entire path.

#### Returns

**tilt angle correction (radians)**

```
neuron_morphology.transforms.tilt_correction.compute_tilt_correction.find_closest_path(soma_voxel: List[Union[IO], n_sublists: int) = 2)
```

Finds closest path to soma\_voxel

#### Parameters

**soma\_voxel:** List containing soma voxel ccf coordinates

**ccf\_path:** str, FilePath, or File-like object openable by H5PY

**n\_sublists:** will separate path\_ids into n sublists to load in at a time. Higher values decrease memory usage but increase processing time. n = 1 uses about 16GB

#### Returns

**closest\_path:** array of voxel coordinates of the closest streamline

```
neuron_morphology.transforms.tilt_correction.compute_tilt_correction.determine_slice_flip(morphology: Morphology object, soma_marker: dict, slice_image_flip: bool) = 1)
```

Determines whether the tilt correction should be positive or negative

#### Parameters

**morphology:** Morphology object

**soma\_marker:** soma marker dictionary from reconstruction marker file

**slice\_image\_flip:** indicates whether the image was flipped relative to the slice (e.g the z axis of the image is opposite to the z axis in the slice)

#### Returns

**flip\_toggle** -1 or 1 to be multiplied against tilt correction

```
neuron_morphology.transforms.tilt_correction.compute_tilt_correction.read_soma_marker(marker_file: str)
```

neuron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction.**run\_tilt\_correction**(m  
M  
ph  
og  
so  
D  
cc  
D  
sl  
af  
sl  
bo  
cc  
U  
IC

neuron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction.**main**()

neuron\_morphology.transforms.upright\_angle

Submodules

neuron\_morphology.transforms.upright\_angle.\_schemas

Module Contents

Classes

<i>InputParameters</i>	The base marshmallow schema used by ArgSchema-Parser to identify
<i>OutputParameters</i>	mm.Schema class with support for making fields default to

Functions

<i>validate_neighbors</i> (num)
---------------------------------

neuron\_morphology.transforms.upright\_angle.\_schemas.**validate\_neighbors** (num)

```
class neuron_morphology.transforms.upright_angle._schemas.InputParameters (only=None,
                                                                    ex-
                                                                    clude=(),
                                                                    many=False,
                                                                    con-
                                                                    text=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False,
                                                                    un-
                                                                    known=None)

Bases: argschema.schemas.ArgSchema

The base marshmallow schema used by ArgSchemaParser to identify input_json and output_json files and the
log_level

gradient_path
node
step
neighbors
swc_path
validate_schema_input (self, data, **kwargs)

class neuron_morphology.transforms.upright_angle._schemas.OutputParameters (only=None,
                                                                    ex-
                                                                    clude=(),
                                                                    many=False,
                                                                    con-
                                                                    text=None,
                                                                    load_only=(),
                                                                    dump_only=(),
                                                                    par-
                                                                    tial=False,
                                                                    un-
                                                                    known=None)

Bases: argschema.schemas.DefaultSchema

mm.Schema class with support for making fields default to values defined by that field's arguments.

inputs
upright_angle
upright_transform_dict
```

```
neuron_morphology.transforms.upright_angle.compute_angle
```

## Module Contents

## Functions



---

```

get_upright_angle(gradient:      xr.DataArray,  Calculate the upright angle at a position, e.g.  soma,
point: Optional[List[float]] = None, n_win: int = 2) →  given a vector field
float


---


calculate_transform(gradient_field:
xr.DataArray, morph:  Morphology, node:  Op-
tional[List[float]] = None)


---


run_upright_angle(gradient_path: str, swc_path:
str, node: Optional[List[float]] = None)


---


main()


---



```

```

neuron_morphology.transforms.upright_angle.compute_angle.get_upright_angle (gradient:
                                                                    xr.DataArray,
                                                                    point:
                                                                    Op-
                                                                    tional[List[float]]
                                                                    =
                                                                    None,
                                                                    n_win:
                                                                    int
                                                                    =
                                                                    2)
                                                                    →
                                                                    float

```

Calculate the upright angle at a position, e.g. soma, given a vector field

#### Parameters

**gradient:** xarray of the the vector field

**point:** list [x,y,z] coordinates

**n\_win:** number of grid points to define the interpolation window

#### Returns

**angle**

```

neuron_morphology.transforms.upright_angle.compute_angle.calculate_transform (gradient_field:
                                                                    xr.DataArray,
                                                                    morph:
                                                                    Mor-
                                                                    phol-
                                                                    ogy,
                                                                    node:
                                                                    Op-
                                                                    tional[List[float]]
                                                                    =
                                                                    None)

```

```
neuron_morphology.transforms.upright_angle.compute_angle.run_upright_angle(gradient_path:
                                                                    str,
                                                                    swc_path:
                                                                    str,
                                                                    node:
                                                                    Optional[List[float]]
                                                                    =
                                                                    None)

neuron_morphology.transforms.upright_angle.compute_angle.main()
```

## Submodules

`neuron_morphology.transforms.affine_transform`

## Module Contents

### Classes

<code>AffineTransform</code>	Handles transformations to a pia/wm aligned coordinate frame.
------------------------------	---

### Functions

<code>affine_from_transform_translation</code> (transform: Any, translation: Optional[Any] = None, translate_first: bool = False)	Create affine from linear transformation and translation.
<code>rotation_from_angle</code> (angle: float, axis: int = 2)	Create an affine matrix from a rotation about a specific axis.
<code>affine_from_translation</code> (translation: Any)	Create an affine translation.
<code>affine_from_transform</code> (transform: Any)	Create affine transformation.

```
class neuron_morphology.transforms.affine_transform.AffineTransform(affine:
                                                                    Op-
                                                                    tional[Any]
                                                                    = None)
```

Bases: `neuron_morphology.transforms.transform_base.TransformBase`

Handles transformations to a pia/wm aligned coordinate frame.

```
classmethod from_dict (cls, affine_dict: Dict[str, float])
    Create an AffineTransform from a dict with keys and values.
```

#### Parameters

**affine\_dict**: keys and values corresponding to the following

```
[[tvr_00 tvr_01 tvr_02 tvr_09] [tvr_03 tvr_04 tvr_05 tvr_10] [tvr_06 tvr_07 tvr_08
tvr_11] [0 0 0 1]]
```

#### Returns

**AffineTransform** object

**classmethod from\_list** (*cls*, *affine\_list*: List[float])

Create an Affine Transform from a list

**Parameters**

**affine\_list**: list of tvr values corresponding to:

[[tvr\_00 tvr\_01 tvr\_02 tvr\_09] [tvr\_03 tvr\_04 tvr\_05 tvr\_10] [tvr\_06 tvr\_07 tvr\_08  
tvr\_11] [0 0 0 1]]

**Returns**

**AffineTransform** object

**to\_dict** (*self*)

Create dictionary defining the transformation.

**Returns**

**Dict with keys and values corresponding to the following:**

[[tvr\_00 tvr\_01 tvr\_02 tvr\_09] [tvr\_03 tvr\_04 tvr\_05 tvr\_10] [tvr\_06 tvr\_07 tvr\_08  
tvr\_11] [0 0 0 1]]

**to\_list** (*self*)

Create a list defining the transformation.

**Returns**

**List with values corresponding to the following:**

[[tvr\_00 tvr\_01 tvr\_02 tvr\_09] [tvr\_03 tvr\_04 tvr\_05 tvr\_10] [tvr\_06 tvr\_07 tvr\_08  
tvr\_11] [0 0 0 1]]

**transform** (*self*, *vector*: Any)

Apply this transform to (3,) point or (n,3) array-like of points.

**Parameters**

**vector**: a (3,) array-like point or a (n,3) array-like array of points to be transformed

**Returns**

**numpy.ndarray** with same shape as input

**\_get\_scaling\_factor** (*self*)

Calculate the scaling factor from the affine matrix.

**Returns**

**Scaling factor: 3rd root of the determinant.**

**transform\_morphology** (*self*, *morphology*: Morphology, *clone*: bool = False, *scale\_radius*: bool = True)

Apply this transform to all nodes in a morphology.

**Parameters**

**morphology**: a Morphology loaded from an swc file

**clone**: make a new object if True

**scale\_radius**: apply radius scaling if True

**Returns**

**A Morphology**

```
neuron_morphology.transforms.affine_transform.affine_from_transform_translation(transform: Optional[Any]
=
None,
trans-
la-
tion:
Optional[Any]
=
None,
trans-
late_first:
bool
=
False)
```

Create affine from linear transformation and translation.

Affine transformation of vector  $x \rightarrow Ax + b$  in 3D: [A, b

0, 0, 0, 1]

A is a 3x3 linear transformation b is a 3x1 translation

#### Parameters

**transform:** linear transformation (3, 3) array-like

**translation:** linear translation (3,) array-like

**translate\_first:** apply the translation before the transform

#### Returns

(4, 4) numpy.ndarray affine matrix

```
neuron_morphology.transforms.affine_transform.rotation_from_angle(angle: float,
axis: int =
2)
```

Create an affine matrix from a rotation about a specific axis.

#### Parameters

**angle:** rotation angle in radians

**axis:** axis to rotate about, 0=x, 1=y, 2=z (default z axis)

#### Returns

(3, 3) numpy.ndarray rotation matrix

```
neuron_morphology.transforms.affine_transform.affine_from_translation(translation: Any)
```

Create an affine translation.

#### Parameters

**translation:** array-like vector of x, y, and z translations

#### Returns

(4, 4) numpy.ndarray affine matrix

`neuron_morphology.transforms.affine_transform.affine_from_transform` (*transform: Any*)

Create affine transformation.

#### Parameters

**transformation:** (3, 3) row major array-like transformation

#### Returns

(4, 4) `numpy.ndarray` affine matrix

## `neuron_morphology.transforms.geometry`

Some handy utilities for working with vector geometries

## Module Contents

### Functions

<code>get_ccw_vertices_from_two_lines</code> ( <i>line1: List[Tuple], line2: List[Tuple]</i> )	Convenience method two do both <code>get_vertices_from_two_lines()</code>
<code>prune_two_lines</code> ( <i>line1: List[Tuple], line2: List[Tuple]</i> )	check the boundary to avoid intersections with side lines
<code>get_vertices_from_two_lines</code> ( <i>line1: List[Tuple], line2: List[Tuple]</i> )	Generates circular vertices from two lines
<code>get_ccw_vertices</code> ( <i>vertices: List[Tuple]</i> )	Generates counter clockwise vertices from vertices describing

`neuron_morphology.transforms.geometry.get_ccw_vertices_from_two_lines` (*line1: List[Tuple], line2: List[Tuple]*)

Convenience method two do both `get_vertices_from_two_lines()` and `get_ccw_vertices()`

`neuron_morphology.transforms.geometry.prune_two_lines` (*line1: List[Tuple], line2: List[Tuple]*)

check the boundary to avoid intersections with side lines

#### Parameters

**line1, line2:** List of coordinates describing two lines

#### Returns

**line1, line2:** boundary pruned if needed

`neuron_morphology.transforms.geometry.get_vertices_from_two_lines` (*line1: List[Tuple], line2: List[Tuple]*)

Generates circular vertices from two lines

#### Parameters

**line1, line2:** List of coordinates describing two lines

#### Returns

vertices of the simple polygon created from line 1 and 2

(first vertex = last vertex)

1-2-3-4

5-6-7-8 -> [1-2-3-4-8-7-6-5-1]

`neuron_morphology.transforms.geometry.get_ccw_vertices` (*vertices: List[Tuple]*)

Generates counter clockwise vertices from vertices describing a simple polygon

Method: Simplification of the shoelace formula, which calculates area of a simple polygon by integrating the area under each line segment of the polygon. If the total area is positive, the vertices were traversed in clockwise order, and if it is negative, they were traversed in counterclockwise order.

#### Parameters

**vertices:** vertices describing a convex polygon (`vertices[0] = vertices[-1]`)

#### Returns

vertices in counter clockwise order

`neuron_morphology.transforms.streamline`

## Module Contents

### Functions

<code>solve_laplace_2d</code> ( <i>V: fem.FunctionSpace, domain, bcs: List[fem.bcs.DirichletBCMetaClass]</i> )	Solves the laplace equation with boundary conditions bcs on V
<code>compute_gradient</code> ( <i>uh, W, bcs=[]</i> )	
<code>generate_laplace_field</code> ( <i>top_line: List[Tuple], bottom_line: List[Tuple], mesh_res: float = 20, top_value: float = 1.0, bottom_value: float = 0.0, eps_bounds: float = 1e-08</i> )	Solve Laplace equation inside a polygon bounded

`neuron_morphology.transforms.streamline.solve_laplace_2d` (*V: fem.FunctionSpace, domain, bcs: List[fem.bcs.DirichletBCMetaClass]*)

Solves the laplace equation with boundary conditions bcs on V

#### Parameters

**V:** Fenics FunctionSpace object created from a mesh

**bcs:** List of Fenics DirichletBC Boundary Conditions

`neuron_morphology.transforms.streamline.compute_gradient` (*uh, W, bcs=[]*)

```
neuron_morphology.transforms.streamline.generate_laplace_field(top_line:
                                                                List[Tuple],
                                                                bottom_line:
                                                                List[Tuple],
                                                                mesh_res: float
                                                                = 20, top_value:
                                                                float = 1.0,
                                                                bottom_value:
                                                                float = 0.0,
                                                                eps_bounds:
                                                                float = 1e-08)
```

Solve Laplace equation inside a polygon bounded by two lines (top\_line and bottom\_line) and the artificial straight side lines connecting the ends of the two lines. Apply Dirichlet BC on top\_line and bottom\_line and zero Neuman BC on the side lines.

**Demo of fenics:** [https://github.com/hplgit/fenics-tutorial/blob/master/pub/python/vol1/ft01\\_poisson.py](https://github.com/hplgit/fenics-tutorial/blob/master/pub/python/vol1/ft01_poisson.py)

If top\_value and bottom\_value defaults are used, value\_field will be the normalized distance to the top\_line

#### Parameters

**top\_line:** line that will have top\_value boundary condition  
**bottom\_line:** line that will have bottom\_value boundary condition  
**mesh\_res:** resolution of the mesh  
**top\_value:** value for top Dirichlet Boundary  
**bottom\_value:** value for bottom Dirichlet Boundary

#### Returns

**u:** returns value at input point e.g.  $0.5 = u((0.5, 0.5))$   
**grad\_u:** returns gradient at input point e.g.  $[0, 1] = u((0.5, 0.5))$   
**mesh\_coords:** coordinates of each vertex in the mesh  
**mesh\_values:** values at each vertex in the mesh  
**gradient\_mesh:** gradient at each vertex in the mesh

`neuron_morphology.transforms.transform_base`

## Module Contents

### Classes

---

*TransformBase*

Abstract base class for implementing swc transforms.

---

**class** neuron\_morphology.transforms.transform\_base.TransformBase

Bases: abc.ABC

Abstract base class for implementing swc transforms. Each child class should implement these methods.

**transform\_morphology** (*self*)

Apply this transform to all nodes in a morphology.

#### Returns

A Morphology

**transform** (*self*)  
Apply this transform to (3,) point or (3,n) array-like of points.

**Returns**  
**numpy.ndarray with same shape as input**

`neuron_morphology.validation`

Submodules

`neuron_morphology.validation.bits_validation`

Module Contents

Functions

<a href="#"><code>validate_independent_axon_has_more_than_three_nodes(morphology)</code></a>	This function checks if an independent (parent is -1) axon has more than three nodes
<a href="#"><code>validate_types_three_four_traceable_back_to_soma(morphology)</code></a>	This function checks if types 3,4 are traceable back to soma
<a href="#"><code>validate(morphology)</code></a>	

`neuron_morphology.validation.bits_validation.validate_independent_axon_has_more_than_four_nodes(morphology)`  
This function checks if an independent (parent is -1) axon has more than three nodes

`neuron_morphology.validation.bits_validation.validate_types_three_four_traceable_back_to_soma(morphology)`  
This function checks if types 3,4 are traceable back to soma

`neuron_morphology.validation.bits_validation.validate(morphology)`

`neuron_morphology.validation.marker_validation`

Module Contents

Functions

<a href="#"><code>validate_coordinates_corresponding_to_dendrite_markers(marker_file, morphology)</code></a>	This function checks whether the coordinates for each dendrite marker in the file correspond to the coordinates for each dendrite marker in the morphology
<a href="#"><code>validate_coordinates_corresponding_to_axon_markers(marker_file, morphology)</code></a>	This function checks whether the coordinates for each axon marker in the file correspond to the coordinates for each axon marker in the morphology
<a href="#"><code>validate_expected_name(marker_file)</code></a>	This function checks whether the markers have the expected types
<a href="#"><code>validate_type_thirty_count(marker_file)</code></a>	This function checks whether there is exactly one type 30 in the file
<a href="#"><code>validate_no_reconstruction_count(marker_file)</code></a>	This function checks whether there is exactly one type 20 in the file
<a href="#"><code>validate(marker_file, morphology)</code></a>	



`neuron_morphology.validation.marker_validation.validate_coordinates_corresponding_to_dendrite`

This function checks whether the coordinates for each dendrite marker corresponds to a tip of a dendrite type in the related morphology

`neuron_morphology.validation.marker_validation.validate_coordinates_corresponding_to_axon_t`

This function checks whether the coordinates for each axon marker corresponds to a tip of a axon type in the related morphology

`neuron_morphology.validation.marker_validation.validate_expected_name(marker_file)`

This function checks whether the markers have the expected types

`neuron_morphology.validation.marker_validation.validate_type_thirty_count(marker_file)`

This function checks whether there is exactly one type 30 in the file

`neuron_morphology.validation.marker_validation.validate_no_reconstruction_count(marker_file)`

This function checks whether there is exactly one type 20 in the file

`neuron_morphology.validation.marker_validation.validate(marker_file, morphology)`

**`neuron_morphology.validation.morphology_statistics`**

## Module Contents

### Functions

<code>count_number_of_independent_axons(morphology)</code>	This functions counts the number of independent axons (parent is -1)
--	--

<code>morphology_statistics(morphology)</code>
--

`neuron_morphology.validation.morphology_statistics.count_number_of_independent_axons(morphology)`

This functions counts the number of independent axons (parent is -1)

`neuron_morphology.validation.morphology_statistics.morphology_statistics(morphology)`

**`neuron_morphology.validation.radius_validation`**

## Module Contents

### Functions

<code>validate_radius_threshold(morphology)</code>	This function validates the radius for types 1, 3, and 4
<code>validate_extreme_taper(morphology)</code>	This function checks whether there is an extreme taper.
<code>validate_radius_has_negative_slope_dendrite(morphology, dendrite)</code>	This function checks whether the radius for dendrite nodes decreases

Continued on next page

Table 68 – continued from previous page

<code>slope_linear_regression_branch_order_avg_radius</code>	Use linear regression to find the slope of the best fit line
<code>validate_constrictions(morphology)</code>	This function checks if the radius of basal dendrite and apical dendrite
<code>validate(morphology)</code>	

`neuron_morphology.validation.radius_validation.validate_radius_threshold(morphology)`  
This function validates the radius for types 1, 3, and 4

`neuron_morphology.validation.radius_validation.validate_extreme_taper(morphology)`  
This function checks whether there is an extreme taper. Extreme taper occurs when for each segment, the average radius of the first two nodes is more than two times the average radius of the last two nodes.

Note: This tests is limited to segments of at least 8 nodes.

`neuron_morphology.validation.radius_validation.validate_radius_has_negative_slope_dendrite`

This function checks whether the radius for dendrite nodes decreases when you are going away from the soma.

`neuron_morphology.validation.radius_validation.slope_linear_regression_branch_order_avg_ra`

Use linear regression to find the slope of the best fit line

`neuron_morphology.validation.radius_validation.validate_constrictions(morphology)`  
This function checks if the radius of basal dendrite and apical dendrite nodes is smaller 2.0px

`neuron_morphology.validation.radius_validation.validate(morphology)`

## `neuron_morphology.validation.report`

### Module Contents

#### Classes

---

#### *Report*

---

**class** `neuron_morphology.validation.report.Report`

Bases: `object`

**add\_swc\_results** (*self*, *swc\_file*, *results*)

**add\_marker\_results** (*self*, *marker\_file*, *results*)

**add\_swc\_stats** (*self*, *swc\_file*, *stats*)

This function creates a report for swc statistics

**to\_json** (*self*)

**has\_results** (*self*)

`neuron_morphology.validation.resample_validation`

### Module Contents

## Functions

---

```
validate_distance_between_connected_nodes(morphology)
validate(morphology)
```

---

```
neuron_morphology.validation.resample_validation.validate_distance_between_connected_nodes
neuron_morphology.validation.resample_validation.validate (morphology)
```

```
neuron_morphology.validation.result
```

## Module Contents

### Classes

---

```
NodeValidationError
MarkerValidationError
```

---

```
class neuron_morphology.validation.result.NodeValidationError (message,  
                                                                node_ids, level)
```

```
    Bases: object
```

```
    message
```

```
    node_ids
```

```
    level
```

```
    __repr__ (self)  
        Return repr(self).
```

```
class neuron_morphology.validation.result.MarkerValidationError (message,  
                                                                marker, level)
```

```
    Bases: object
```

```
    message
```

```
    marker
```

```
    level
```

```
    __repr__ (self)  
        Return repr(self).
```

```
exception neuron_morphology.validation.result.InvalidMorphology (validation_errors)  
    Bases: ValueError
```

```
    Inappropriate argument value (of correct type).
```

```
    validation_errors
```

```
    __str__ (self)  
        Return str(self).
```

```
exception neuron_morphology.validation.result.InvalidMarkerFile (validation_errors)  
    Bases: ValueError
```

```
    Inappropriate argument value (of correct type).
```

**validation\_errors**

```
__str__(self)
    Return str(self).
```

**neuron\_morphology.validation.structure\_validation****Module Contents****Functions**


---

```
validate_children_nodes_appear_before_parent_nodes(morphology)
validate(morphology)
```

---

```
neuron_morphology.validation.structure_validation.validate_children_nodes_appear_before_pa
```

```
neuron_morphology.validation.structure_validation.validate (morphology)
```

**neuron\_morphology.validation.type\_validation****Module Contents****Functions**

<i>validate_count_node_parent</i> (morphology, node_type, parent_type, expected_count)	This function validates the number of nodes that have a specific type of parent
<i>validate_number_of_soma_nodes</i> (morphology)	This function validates the number of type 1 nodes
<i>validate_expected_types</i> (morphology)	This function validates the expected types of the nodes
<i>valid_dendrite_parent</i> (morphology, node, valid_parent_type)	
<i>validate_node_parent</i> (morphology)	This function validates the type of parent node for a specific type of child node
<i>validate_immediate_children_of_soma_cannot_branch</i> (morphology)	This function validates that immediate children of soma cannot branch
<i>validate_multiple_axon_initiation_points</i> (morphology)	This function validates that the parent of axon (either type 1 or 3) only happens once
<i>validate</i> (morphology)	

```
neuron_morphology.validation.type_validation.valid_types
```

```
neuron_morphology.validation.type_validation.validate_count_node_parent (morphology,
                                                                    node_type,
                                                                    parent_type,
                                                                    expected_count)

    This function validates the number of nodes that have a specific type of parent
```

```
neuron_morphology.validation.type_validation.validate_number_of_soma_nodes (morphology)
    This function validates the number of type 1 nodes
```

`neuron_morphology.validation.type_validation.validate_expected_types` (*morphology*)

This function validates the expected types of the nodes

`neuron_morphology.validation.type_validation.valid_dendrite_parent` (*morphology*,  
*node*,  
*valid\_parent\_type*)

`neuron_morphology.validation.type_validation.validate_node_parent` (*morphology*)

This function validates the type of parent node for a specific type of child node

`neuron_morphology.validation.type_validation.validate_immediate_children_of_soma_cannot_branch`

This function validates that immediate children of soma cannot branch

`neuron_morphology.validation.type_validation.validate_multiple_axon_initiation_points` (*morphology*)

This function validates that the parent of axon (either type 1 or 3) only happens once

`neuron_morphology.validation.type_validation.validate` (*morphology*)

**`neuron_morphology.validation.validate_reconstruction`**

## Module Contents

### Functions

<code>parse_arguments</code> (args)	This function parses command line arguments
<code>main</code> ()	

`neuron_morphology.validation.validate_reconstruction.logger`

`neuron_morphology.validation.validate_reconstruction.parse_arguments` (*args*)

This function parses command line arguments

`neuron_morphology.validation.validate_reconstruction.main` ()

## Package Contents

### Functions

<code>validate_morphology</code> ( <i>morphology</i> )
<code>validate_marker</code> ( <i>marker</i> , <i>morphology</i> )

`neuron_morphology.validation.swc_validators`

`neuron_morphology.validation.marker_validators`

`neuron_morphology.validation.validate_morphology` (*morphology*)

`neuron_morphology.validation.validate_marker` (*marker*, *morphology*)

**`neuron_morphology.vis`**

## Submodules

`neuron_morphology.vis.morphovis`

## Module Contents

### Functions

---

<code>plot_morphology_xy(morphology, ax)</code>
<code>plot_morphology_zy(morphology, ax)</code>
<code>plot_cortical_boundary(pia_coords, wm_coords, ax)</code>
<code>plot_soma(soma_center, ax)</code>
<code>plot_depth_field(depth_field, ax)</code>
<code>plot_gradient_field(gradient_field, ax)</code>

---

```
neuron_morphology.vis.morphovis.plot_morphology_xy(morphology, ax)
neuron_morphology.vis.morphovis.plot_morphology_zy(morphology, ax)
neuron_morphology.vis.morphovis.plot_cortical_boundary(pia_coords, wm_coords,
                                                         ax)
neuron_morphology.vis.morphovis.plot_soma(soma_center, ax)
neuron_morphology.vis.morphovis.plot_depth_field(depth_field, ax)
neuron_morphology.vis.morphovis.plot_gradient_field(gradient_field, ax)
```

## 4.1.2 Submodules

`neuron_morphology.constants`

### Module Contents

```
neuron_morphology.constants.SOMA = 1
neuron_morphology.constants.AXON = 2
neuron_morphology.constants.BASAL_DENDRITE = 3
neuron_morphology.constants.APICAL_DENDRITE = 4
neuron_morphology.constants.CUT_DENDRITE = 10
neuron_morphology.constants.NO_RECONSTRUCTION = 20
neuron_morphology.constants.TYPE_30 = 30
neuron_morphology.constants.SPACING = [0.1144, 0.1144, 0.28]
```

`neuron_morphology.lims_apical_queries`

### Module Contents

## Functions

---

<code>convert_coords_str</code>	<code>(coords_str: str, resolution=None)</code>	Convert a comma seperated string of coordinate pairs
---------------------------------	---	--

---

<code>get_data</code>	<code>(query)</code>
-----------------------	----------------------

---

<code>get_all_intact_apical</code>	<code>()</code>
------------------------------------	-----------------

---

<code>get_benchmark_apical</code>	<code>()</code>
-----------------------------------	-----------------

---

`neuron_morphology.lims_apical_queries.convert_coords_str` (`coords_str: str, resolution=None`)

Convert a comma seperated string of coordinate pairs

`neuron_morphology.lims_apical_queries.get_data` (`query`)

`neuron_morphology.lims_apical_queries.get_all_intact_apical` (`()`)

`neuron_morphology.lims_apical_queries.get_benchmark_apical` (`()`)

`neuron_morphology.marker`

## Module Contents

### Classes

---

<code>Marker</code>	Simple dictionary class for handling reconstruction marker objects.
---------------------	---

---

## Functions

---

<code>read_marker_file</code>	<code>(file_name)</code>	read in a marker file and return a list of dictionaries
-------------------------------	--------------------------	---

---

**class** `neuron_morphology.marker.Marker` (`*args, **kwargs`)

Bases: `dict`

Simple dictionary class for handling reconstruction marker objects.

`neuron_morphology.marker.read_marker_file` (`file_name`)

read in a marker file and return a list of dictionaries

`neuron_morphology.morphology`

## Module Contents

### Classes

---

<code>Morphology</code>
-------------------------

---

**class** `neuron_morphology.morphology.Morphology` (`nodes, node_id_cb, parent_id_cb`)

Bases: `allensdk.core.simple_tree.SimpleTree`

`__len__ (self)`

`validate (self, strict=False)`

Validate the neuron morphology in [bits, radius, resample, type, structure]

`children_of (self, node)`

`parent_of (self, node)`

`get_children_of_node_by_types (self, node, node_types)`

`get_children (self, node, node_types=None)`

`node_by_id (self, node_id)`

`get_soma (self)`

Return one soma node labeled with SOMA If the input SWC file does not have any node labeled with SOMA, it will return None

**Parameters**

**morphology:** Morphology object

**Returns**

**Soma node object**

`get_root (self)`

Return the first found root node If the input SWC file does not have any root node, it will return None

**Parameters**

**morphology:** Morphology object

**Returns**

**Root node object**

`get_roots (self)`

`get_root_id (self)`

`get_roots_for_nodes (self, nodes)`

`get_roots_for_analysis (self, root=None, node_types=None)`

Returns a list of all trees to be analyzed, based on the supplied root. These trees are the list of all children of the root, if root is not None, and the root node of all trees in the morphology if root is None.

**Parameters**

**morphology:** Morphology object

**root:** dict

This is the node from which to count branches under. When root=None, all separate trees in the morphology are returned.

**node\_types:** list (AXON, BASAL\_DENDRITE, APICAL\_DENDRITE)

Type to restrict search to

**Returns**

**Array of Node objects**

`get_number_of_trees (self, nodes=None)`

`get_tree_list (self)`



```

get_root_for_tree (self, tree_number)
get_node_by_types (self, node_types=None)
has_type (self, node_type)
get_non_soma_nodes (self)
get_max_id (self)
is_soma_child (self, node)
get_segment_list (self, node_types=None)
_build_segment (self, end_node)
is_node_at_beginning_of_segment (self, node)
is_node_at_end_of_segment (self, node)
get_segment_length (self, segment)
get_branch_order_for_node (self, node)
get_branch_order_for_segment (self, segment)
_create_compartment_dictionary (self)
get_compartments (self, nodes=None, node_types=None)
get_compartment_for_node (self, node, node_types=None)
get_compartment_length (self, compartment)
get_compartment_surface_area (self, compartment: Sequence[Dict])
    Calculate the surface area of a single compartment. Treats the compartment as a circular conic frustum
    and calculates its lateral surface area. This is:

$$\pi * (r_1 + r_2) * \sqrt{(r_2 - r_1)^2 + L^2}$$

    Parameters
        compartment [two-long sequence. Each element is a node and must have] 3d position
            data ("x", "y", "z") and a "radius"
    Returns
        The surface area of the sides of the compartment

get_compartment_volume (self, compartment: Sequence[Dict])
    Calculate the volume of a single compartment. Treats the compartment as a circular conic frustum and
    calculates its volume as:

$$\pi * L * (r_1^2 + r_1 * r_2 + r_2^2) / 3$$

    Parameters
        compartment [two-long sequence. Each element is a node and must have] 3d position
            data ("x", "y", "z") and a "radius"
    Returns
        The volume of the compartment

get_compartment_midpoint (self, compartment)
get_leaf_nodes (self, node_types=None)

```

**get\_branching\_nodes** (*self*, *node\_types=None*)  
**clone** (*self*)  
**build\_intermediate\_nodes** (*self*, *make\_intermediates\_cb*, *set\_parent\_id\_cb*)  
**\_insert\_between** (*self*, *new\_node*, *parent\_id*, *child\_id*, *set\_parent\_id\_cb*)  
**\_make\_and\_insert\_intermediate** (*self*, *make\_intermediates\_cb*, *set\_parent\_id\_cb*, *child*)  
**breadth\_first\_traversal** (*self*, *visit*, *neighbor\_cb=None*, *start\_id=None*)  
Apply a function to each node of a connected graph in breadth-first order

#### Parameters

**visit** [callable] Will be applied to each node. Signature must be visit(node). Return is ignored.  
**neighbor\_cb** [callable, optional] Will be used during traversal to find the next nodes to be visited. Signature must be neighbor\_cb(node id) -> list of node\_ids. Defaults to self.child\_ids.  
**start\_id** [hashable, optional] Begin the traversal from this node. Defaults to self.get\_root\_id().

#### Notes

assumes rooted, acyclic

**depth\_first\_traversal** (*self*, *visit*, *neighbor\_cb=None*, *start\_id=None*)  
Apply a function to each node of a connected graph in depth-first order

#### Parameters

**visit** [callable] Will be applied to each node. Signature must be visit(node). Return is ignored.  
**neighbor\_cb** [callable, optional] Will be used during traversal to find the next nodes to be visited. Signature must be neighbor\_cb(node\_id) -> list of node\_ids. Defaults to self.child\_ids.  
**start\_id** [hashable, optional] Begin the traversal from this node. Defaults to self.get\_root\_id().

#### Notes

assumes rooted, acyclic

**swap\_nodes\_edges** (*self*, *merge\_cb=None*, *parent\_id\_cb=None*, *make\_root\_cb=None*,  
*start\_id=None*)  
Build a new tree whose nodes are the edges of this tree and vice-versa

#### Parameters

**merge\_cb** [callable, optional]  
**parent\_id\_cb** [callable, optional]  
**make\_root\_cb** [callable, optional]  
**start\_id** [hashable, optional]

## Notes

assumes rooted, acyclic

**`_get_edge_and_merge`** (*self*, *merge\_cb*, *new\_nodes*, *node*)

Used by `swap_nodes_edges`

**`static _get_node_attributes`** (*attributes*, *nodes*)

**`get_dimensions`** (*self*, *node\_types=None*)

**`static euclidean_distance`** (*node1*, *node2*)

**`static midpoint`** (*node1*, *node2*)

`neuron_morphology.morphology_builder`

## Module Contents

### Classes

---

*MorphologyBuilder*

---

**`class`** `neuron_morphology.morphology_builder.MorphologyBuilder`

**`next_id`**

**`parent_id`**

**`active_node_id`**

**`up`** (*self*, *by=1*)

Terminate a branch. Set the active node to the previous active node's ancestor.

#### Parameters

**`by`** [how far (up the tree) to set the new active node. Default is the] parent of the current node (1). 2 would correspond to the

**`root`** (*self*, *x=0*, *y=0*, *z=0*, *node\_type=SOMA*, *radius=1*)

Add a new root node (parent -1) to this reconstruction. This will be the new active node.

**`child`** (*self*, *x*, *y*, *z*, *node\_type*, *radius=1*)

Add a child node to the current active node. This will become the new active node.

**`_add_node`** (*self*, *x*, *y*, *z*, *node\_type*, *radius*)

Add a new node to this morphology.

#### Parameters

**`x`** [x coordinate of this node's position]

**`y`** [y coordinate of this node's position]

**`z`** [z coordinate of this node's position]

**`node_type`** [one of AXON, SOMA, APICAL\_DENDRITE, BASAL\_DENDRITE]

**`radius`** [describe the size of this node]

**axon** (*self*, *x=None*, *y=None*, *z=None*, *radius=1*)

Convenience for creating an axon node. Will not create a root.

**apical\_dendrite** (*self*, *x=None*, *y=None*, *z=None*, *radius=1*)

Convenience for creating an apical dendrite node. Will not create a root.

**basal\_dendrite** (*self*, *x=None*, *y=None*, *z=None*, *radius=1*)

Convenience for creating a basal dendrite node. Will not create a root.

**build** (*self*)

Construct a Morphology object using this builder. This is a non- destructive operation. The Morphology will be validated at this stage.

`neuron_morphology.swc_io`

## Module Contents

### Functions

<code>read_swc(path, columns=SWC_COLUMNS, sep=''</code>	Read an swc file into a pandas dataframe
<code>', casts=COLUMN_CASTS)</code>	
<code>write_swc(data, path, comments=None, sep=''</code>	Write an swc file from a pandas dataframe
<code>', columns=SWC_COLUMNS, casts=COLUMN_CASTS)</code>	
<code>apply_casts(df, casts)</code>	
<code>morphology_from_swc(swc_path)</code>	
<code>morphology_to_swc(morphology, swc_path, com-</code>	Write an swc file from a morphology object
<code>ments=None)</code>	

`neuron_morphology.swc_io.SWC_COLUMNS = ['id', 'type', 'x', 'y', 'z', 'radius', 'parent']`

`neuron_morphology.swc_io.COLUMN_CASTS`

`neuron_morphology.swc_io.read_swc(path, columns=SWC_COLUMNS, sep=''`  
`casts=COLUMN_CASTS)`

Read an swc file into a pandas dataframe

`neuron_morphology.swc_io.write_swc(data, path, comments=None, sep=''`  
`columns=SWC_COLUMNS, casts=COLUMN_CASTS)`

Write an swc file from a pandas dataframe

`neuron_morphology.swc_io.apply_casts(df, casts)`

`neuron_morphology.swc_io.morphology_from_swc(swc_path)`

`neuron_morphology.swc_io.morphology_to_swc(morphology, swc_path, comments=None)`  
Write an swc file from a morphology object

## 4.1.3 Package Contents

`neuron_morphology.version_path`

`neuron_morphology.__version__`

*neuron morphology* is an open-source Python package for working with single-neuron morphological reconstruction data, such as those in the [Allen Cell Types Database](#). It provides tools for transforming, analyzing, and visualizing

these data. To get started, take a look at the [installation instructions](#) and the [usage guides](#).

To report a bug or request a feature, see [the issues page](#).



---

## Python Module Index

---

### n

neuron\_morphology, 7

neuron\_morphology.constants, 106

neuron\_morphology.feature\_extractor, 7

neuron\_morphology.feature\_extractor.\_\_main\_\_, 7

neuron\_morphology.feature\_extractor.\_schemas, 9

neuron\_morphology.feature\_extractor.data, 11

neuron\_morphology.feature\_extractor.feature\_extraction\_run, 12

neuron\_morphology.feature\_extractor.feature\_extractor, 12

neuron\_morphology.feature\_extractor.feature\_specialization, 13

neuron\_morphology.feature\_extractor.feature\_writer, 16

neuron\_morphology.feature\_extractor.mark, 19

neuron\_morphology.feature\_extractor.marked\_feature, 23

neuron\_morphology.feature\_extractor.run\_feature\_extraction, 26

neuron\_morphology.feature\_extractor.utilities, 28

neuron\_morphology.features, 28

neuron\_morphology.features.branching, 28

neuron\_morphology.features.branching.bifurcations, 28

neuron\_morphology.features.default\_features, 43

neuron\_morphology.features.dimension, 43

neuron\_morphology.features.intrinsic, 44

neuron\_morphology.features.layer, 31

neuron\_morphology.features.layer.layer\_histogram, 62

neuron\_morphology.features.layer.layered\_point\_depths, 31

neuron\_morphology.features.layer.reference\_layer\_depths, 36

neuron\_morphology.features.path, 46

neuron\_morphology.features.size, 48

neuron\_morphology.features.soma, 51

neuron\_morphology.features.statistics, 37

neuron\_morphology.features.statistics.coordinates, 37

neuron\_morphology.features.statistics.moments, 40

neuron\_morphology.features.statistics.moments\_along, 41

neuron\_morphology.features.statistics.overlap, 41

neuron\_morphology.layered\_point\_depths, 52

neuron\_morphology.layered\_point\_depths.\_\_main\_\_, 52

neuron\_morphology.layered\_point\_depths.\_schemas, 56

neuron\_morphology.lims\_apical\_queries, 106

neuron\_morphology.marker, 107

neuron\_morphology.morphology, 107

neuron\_morphology.morphology\_builder, 111

neuron\_morphology.pipeline, 58

neuron\_morphology.pipeline.\_schemas, 58

neuron\_morphology.pipeline.post\_data\_to\_s3, 60

neuron\_morphology.snap\_polygons, 61

neuron\_morphology.snap\_polygons.\_\_main\_\_, 61

neuron\_morphology.snap\_polygons.\_from\_lims, 62

neuron\_morphology.snap\_polygons.\_schemas,

65 neuron\_morphology.validation.marker\_validation,  
 neuron\_morphology.snap\_polygons.bounding\_box, 100  
 67 neuron\_morphology.validation.morphology\_statistics,  
 neuron\_morphology.snap\_polygons.cortex\_surfaces, 101  
 68 neuron\_morphology.validation.radius\_validation,  
 neuron\_morphology.snap\_polygons.geometries, 101  
 70 neuron\_morphology.validation.report, 102  
 neuron\_morphology.snap\_polygons.image\_output, 102  
 76 neuron\_morphology.validation.resample\_validation,  
 neuron\_morphology.snap\_polygons.types, neuron\_morphology.validation.result, 103  
 79 neuron\_morphology.validation.structure\_validation,  
 neuron\_morphology.swc\_io, 112 104  
 neuron\_morphology.transforms, 80 neuron\_morphology.validation.type\_validation,  
 neuron\_morphology.transforms.affine\_transform, 104  
 94 neuron\_morphology.validation.validate\_reconstruction,  
 neuron\_morphology.transforms.affine\_transformer, 105  
 80 neuron\_morphology.vis, 105  
 neuron\_morphology.transforms.affine\_transformer.morphology.vis.morphovis, 106  
 80  
 neuron\_morphology.transforms.affine\_transformer.apply\_affine\_transform,  
 82  
 neuron\_morphology.transforms.geometry,  
 97  
 neuron\_morphology.transforms.pia\_wm\_streamlines,  
 82  
 neuron\_morphology.transforms.pia\_wm\_streamlines.\_schemas,  
 82  
 neuron\_morphology.transforms.pia\_wm\_streamlines.calculate\_pia\_wm\_streamlines,  
 83  
 neuron\_morphology.transforms.scale\_correction,  
 85  
 neuron\_morphology.transforms.scale\_correction.\_schemas,  
 85  
 neuron\_morphology.transforms.scale\_correction.compute\_scale\_correction,  
 86  
 neuron\_morphology.transforms.streamline,  
 98  
 neuron\_morphology.transforms.tilt\_correction,  
 87  
 neuron\_morphology.transforms.tilt\_correction.\_schemas,  
 87  
 neuron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction,  
 89  
 neuron\_morphology.transforms.transform\_base,  
 99  
 neuron\_morphology.transforms.upright\_angle,  
 91  
 neuron\_morphology.transforms.upright\_angle.\_schemas,  
 91  
 neuron\_morphology.transforms.upright\_angle.compute\_angle,  
 92  
 neuron\_morphology.validation, 100  
 neuron\_morphology.validation.bits\_validation,  
 100



## Symbols

`__call__()` (`neuron_morphology.feature_extractor.marked_feature.MarkedFeature` method), 24  
`__hash__()` (`neuron_morphology.feature_extractor.data.Data` method), 11  
`__hash__()` (`neuron_morphology.feature_extractor.marked_feature.MarkedFeature` method), 24  
`__len__()` (`neuron_morphology.morphology.Morphology` method), 107  
`__name__` (`neuron_morphology.feature_extractor.marked_feature.MarkedFeature` attribute), 24  
`__repr__()` (`neuron_morphology.feature_extractor.marked_feature.MarkedFeature` method), 24  
`__repr__()` (`neuron_morphology.snap_polygons.bounding_box.BoundingBox` method), 67  
`__repr__()` (`neuron_morphology.validation.result.MarkerValidationError` method), 103  
`__repr__()` (`neuron_morphology.validation.result.NodeValidationError` method), 103  
`__slots__` (`neuron_morphology.feature_extractor.marked_feature.MarkedFeature` attribute), 24  
`__slots__` (`neuron_morphology.snap_polygons.bounding_box.BoundingBox` attribute), 67  
`__str__()` (`neuron_morphology.validation.result.InvalidMarkerFile` method), 104  
`__str__()` (`neuron_morphology.validation.result.InvalidMorphologyBuilder` method), 103  
`__version__` (in module `neuron_morphology`), 112  
`_add_node()` (`neuron_morphology.morphology_builder.MorphologyBuilder` method), 111  
`_build_segment()` (`neuron_morphology.morphology.Morphology` method), 109  
`_calculate_max_path_distance()` (in module `neuron_morphology.features.path`), 46  
`_calculate_mean_contraction()` (in module `neuron_morphology.features.path`), 47  
`_create_compartment_dictionary()` (`neuron_morphology.morphology.Morphology` method), 109  
`_draw_geometries()` (`neuron_morphology.snap_polygons.image_outputter.ImageOutputter` method), 77  
`_get_edge_and_merge()` (`neuron_morphology.morphology.Morphology` method), 111  
`_get_node_attributes()` (`neuron_morphology.morphology.Morphology` static method), 111  
`_get_scaling_factor()` (`neuron_morphology.transforms.affine_transform.AffineTransform` method), 95  
`_insert_between()` (`neuron_morphology.morphology.Morphology` method), 110  
`_make_and_insert_intermediate()` (`neuron_morphology.morphology.Morphology` method), 110  
`_register_many()` (`neuron_morphology.snap_polygons.geometries.Geometries` method), 72  
`active_node_id` (`neuron_morphology.morphology_builder.MorphologyBuilder` attribute), 111  
`add_layer_histogram()` (in module `neuron_morphology.feature_extractor.feature_writer`), 18  
`add_mark()` (`neuron_morphology.feature_extractor.marked_feature.MarkedFeature` method), 24  
`add_marker_results()` (`neuron_morphology.validation.report.Report` method), 102  
`add_run()` (`neuron_morphology.feature_extractor.feature_writer.FeatureWriter` method), 16  
`add_swc_results()` (`neuron_morphology.validation.report.Report` method), 102

[add\\_swc\\_stats\(\)](#) (neuron\_morphology.validation.report.Report method), 102  
[affine\\_dict\(neuron\\_morphology.transforms.affine\\_transformer.schemas.ApplyAffineSchema in neuron\\_morphology.feature\\_extractor.feature\\_specialization\), attribute\), 81](#)  
[affine\\_from\\_transform\(\)](#) (in module neuron\_morphology.transforms.affine\_transform), 14  
[affine\\_from\\_transform\\_translation\(\)](#) (in module neuron\_morphology.transforms.affine\_transform), 96  
[affine\\_from\\_translation\(\)](#) (in module neuron\_morphology.transforms.affine\_transform), 95  
[affine\\_list\(neuron\\_morphology.transforms.affine\\_transformer.schemas.ApplyAffineSchema attribute\), 81](#)  
[AffineDictSchema](#) (class in neuron\_morphology.transforms.affine\_transformer.schemas), 14  
[AffineTransform](#) (class in neuron\_morphology.transforms.affine\_transform), 80  
[AllNeuriteCompareSpec](#) (class in neuron\_morphology.feature\_extractor.feature\_specialization), 94  
[AllNeuriteSpec](#) (class in neuron\_morphology.feature\_extractor.feature\_specialization), 22  
[AllNeuriteTypes](#) (class in neuron\_morphology.feature\_extractor.mark), 38  
[APICAL\\_DENDRITE](#) (in module neuron\_morphology.constants), 106  
[apical\\_dendrite\(\)](#) (neuron\_morphology.morphology\_builder.MorphologyBuilder method), 112  
[ApicalDendriteCompareSpec](#) (class in neuron\_morphology.feature\_extractor.feature\_specialization), 15  
[ApicalDendriteSpec](#) (class in neuron\_morphology.feature\_extractor.feature\_specialization), 14  
[apply\\_casts\(\)](#) (in module neuron\_morphology.swc\_io), 112  
[ApplyAffineSchema](#) (class in neuron\_morphology.transforms.affine\_transformer.schemas), 81  
[aspect\\_ratio](#) (neuron\_morphology.snap\_polygons.bounding\_box.BoundingBox attribute), 67  
[AXON](#) (in module neuron\_morphology.constants), 106  
[axon\(\)](#) (neuron\_morphology.morphology\_builder.MorphologyBuilder method), 111  
[AxonCompareSpec](#) (class in neuron\_morphology.feature\_extractor.feature\_specialization), 15  
[BasalDendriteCompareSpec](#) (class in neuron\_morphology.feature\_extractor.feature\_specialization), 14  
[BASAL\\_DENDRITE](#) (in module neuron\_morphology.constants), 106  
[basal\\_dendrite\(\)](#) (neuron\_morphology.morphology\_builder.MorphologyBuilder method), 112  
[BasalDendriteSpec](#) (class in neuron\_morphology.feature\_extractor.feature\_specialization), 14  
[BIFURCATION](#) (neuron\_morphology.features.statistics.coordinates.COORDINATES attribute), 38  
[BifurcationFeatures](#) (class in neuron\_morphology.feature\_extractor.mark), 22  
[BifurcationSpec](#) (class in neuron\_morphology.features.statistics.coordinates), 38  
[BranchingEdges](#) (neuron\_morphology.features.layer.layer\_histogram.LayerHistogram attribute), 32  
[BothEmpty](#) (neuron\_morphology.features.layer.layer\_histogram.EarthMover attribute), 32  
[BothPresent](#) (neuron\_morphology.features.layer.layer\_histogram.EarthMover attribute), 32  
[boundaries](#) (neuron\_morphology.feature\_extractor.schemas.Reference attribute), 9  
[BoundingBox](#) (class in neuron\_morphology.snap\_polygons.bounding\_box), 67  
[BranchingBox](#) (neuron\_morphology.layered\_point\_depths.schemas.Layer attribute), 57  
[breadth\\_first\\_traversal\(\)](#) (neuron\_morphology.morphology.Morphology method), 110  
[build\(\)](#) (neuron\_morphology.morphology\_builder.MorphologyBuilder method), 112  
[build\\_intermediate\\_nodes\(\)](#) (neuron\_morphology.morphology.Morphology method), 110  
[build\\_output\\_table\(\)](#) (neuron\_morphology.feature\_extractor.feature\_writer.FeatureWriter method), 17  
[calculate\\_branches\\_from\\_root\(\)](#) (in module

`neuron_morphology.features.intrinsic)`, 45  
`calculate_coordinate_overlap()`  
     (in module `neuron_morphology.features.statistics.overlap`), 42  
`calculate_coordinate_overlap_from_min_max()`  
     (in module `neuron_morphology.features.statistics.overlap`), 42  
`calculate_max_branch_order_from_root()`  
     (in module `neuron_morphology.features.intrinsic`), 45  
`calculate_max_path_distance()` (in module `neuron_morphology.features.path`), 46  
`calculate_mean_contraction()` (in module `neuron_morphology.features.path`), 48  
`calculate_mean_fragmentation_from_root()`  
     (in module `neuron_morphology.features.intrinsic`), 45  
`calculate_outer_bifs()` (in module `neuron_morphology.features.branching.bifurcations`), 29  
`calculate_relative_soma_depth()` (in module `neuron_morphology.features.soma`), 51  
`calculate_soma_features()` (in module `neuron_morphology.features.soma`), 51  
`calculate_soma_surface()` (in module `neuron_morphology.features.soma`), 51  
`calculate_stem_exit_and_distance()` (in module `neuron_morphology.features.soma`), 51  
`calculate_transform()` (in module `neuron_morphology.transforms.upright_angle.compute_angle`), 93  
`ccf_path(neuron_morphology.transforms.tilt_correction._schemas.InputParameters attribute)`, 88  
`CCF_RESOLUTION` (in module `neuron_morphology.transforms.tilt_correction.compute_tilt_correction`), 89  
`CCF_SHAPE` (in module `neuron_morphology.transforms.tilt_correction.compute_tilt_correction`), 89  
`ccf_soma_location` (in module `neuron_morphology.transforms.tilt_correction._schemas.InputParameters attribute`), 88  
`ccf_soma_xyz` (in module `neuron_morphology.pipeline._schemas.InputParameters attribute`), 60  
`cell_depth(neuron_morphology.pipeline._schemas.InputParameters attribute)`, 60  
`check(neuron_morphology.feature_extractor.feature_writer.FeatureFromMorphology attribute)`, 17  
`check_nodes_have_key()` (in module `neuron_morphology.feature_extractor.mark`), 23  
`child()` (`neuron_morphology.morphology_builder.MorphologyBuilder` method), 111  
`child_ids_by_type()` (in module `neuron_morphology.features.intrinsic`), 44  
`children_of()` (`neuron_morphology.morphology.Morphology` method), 108  
`clear_overlaps()` (in module `neuron_morphology.snap_polygons.geometries`), 74  
`clone()` (`neuron_morphology.morphology.Morphology` method), 110  
`close_bounds` (`neuron_morphology.snap_polygons.geometries.Geometries` attribute), 72  
`closest_from_stack()` (in module `neuron_morphology.snap_polygons.geometries`), 74  
`collect_inputs()` (in module `neuron_morphology.transforms.scale_correction.compute_scale_correction`), 87  
`COLUMN_CASTS` (in module `neuron_morphology.swc_io`), 112  
`COMPARTMENT` (`neuron_morphology.features.statistics.coordinates.COORD_TYPE` attribute), 38  
`CompartmentFeatures` (class in `neuron_morphology.feature_extractor.mark`), 22  
`CompartmentSpec` (class in `neuron_morphology.features.statistics.coordinates`), 38  
`compute_gradient()` (in module `neuron_morphology.transforms.streamline`), 94  
`ConditionFn` (in module `neuron_morphology.snap_polygons.cortex_surfaces`), 62  
`ConfigSchema` (`neuron_morphology.snap_polygons._from_lims.FromLimsSource` attribute), 65  
`containing_layer()` (in module `neuron_morphology.layered_point_depths._main_`), 104  
`convert_coords_str()` (in module `neuron_morphology.lims_apical_queries`), 107  
`convert_path_str_to_list()` (in module `neuron_morphology.transforms.pia_wm_streamlines.calculate_pia_volume`), 84  
`convex_hull()` (`neuron_morphology.snap_polygons.geometries.Geometries` method), 73  
`COORD_TYPE` (class in `neuron_morphology.features.statistics.coordinates`), 38

COORD\_TYPE\_SPECIALIZATIONS (in module *neuron\_morphology.features.statistics.coordinates*), 39

coordinates (*neuron\_morphology.snap\_polygons.bounding\_box.BoundingBox* module *neuron\_morphology.features.layer.reference\_layer\_depths*), 67

copy () (*neuron\_morphology.snap\_polygons.bounding\_box.BoundingBox* method), 68

count\_number\_of\_independent\_axons () (in module *neuron\_morphology.validation.morphology\_statistics*), 101

counts (*neuron\_morphology.features.layer.layer\_histogram.LayerHistogram* attribute), 32

credentials\_file (in *neuron\_morphology.pipeline.\_schemas.S3LandingBucket* attribute), 59

cut () (*neuron\_morphology.snap\_polygons.geometries.Geometries* method), 73

CUT\_DENDRITE (in module *neuron\_morphology.constants*), 106

cut\_thickness (in *neuron\_morphology.pipeline.\_schemas.InputParameters* attribute), 60

cut\_thickness (in *neuron\_morphology.transforms.scale\_correction.\_schemas.InputParameters* attribute), 85

**D**

Data (class in *neuron\_morphology.feature\_extractor.data*), 11

database (*neuron\_morphology.snap\_polygons.\_from\_lims.PostgresInputConfigSchema* attribute), 64

deepcopy () (*neuron\_morphology.feature\_extractor.marked\_feature\_extractor.feature\_extractor* method), 24

DEFAULT\_COLOR\_CYCLE (in *neuron\_morphology.snap\_polygons.image\_outputter.ImageOutputter* attribute), 77

DEFAULT\_FEATURE\_FORMATTERS (in module *neuron\_morphology.feature\_extractor.feature\_writer*), 19

default\_features (in module *neuron\_morphology.features.default\_features*), 43

DEFAULT\_HUMAN\_ME\_MET\_REFERENCE\_LAYER\_DEPTHS (in module *neuron\_morphology.features.layer.reference\_layer\_depths*), 37

DEFAULT\_HUMAN\_MTG\_REFERENCE\_LAYER\_DEPTHS (in module *neuron\_morphology.features.layer.reference\_layer\_depths*), 37

DEFAULT\_MOUSE\_ME\_MET\_REFERENCE\_LAYER\_DEPTHS (in module *neuron\_morphology.features.layer.reference\_layer\_depths*), 37

DEFAULT\_MOUSE\_REFERENCE\_LAYER\_DEPTHS (in module *neuron\_morphology.features.layer.reference\_layer\_depths*), 37

DEFAULT\_MULTIPOLYGON\_RESOLVER (in module *neuron\_morphology.snap\_polygons.geometries.Geometries* attribute), 72

DEFAULT\_MULTISURFACE\_RESOLVER (in module *neuron\_morphology.snap\_polygons.geometries.Geometries* attribute), 72

DendriteCompareSpec (class in *neuron\_morphology.feature\_extractor.feature\_specialization*), 15

DendriteSpec (class in *neuron\_morphology.feature\_extractor.feature\_specialization*), 14

depth (*neuron\_morphology.layered\_point\_depths.\_schemas.InputParameters* attribute), 58

depth\_field\_file (in *neuron\_morphology.transforms.pia\_wm\_streamlines.\_schemas.OutputParameters* attribute), 83

depth\_field\_path (in *neuron\_morphology.transforms.scale\_correction.\_schemas.InputParameters* attribute), 85

depth\_first\_traversal () (in *neuron\_morphology.morphology.Morphology* method), 110

DepthField (class in *neuron\_morphology.layered\_point\_depths.\_schemas*), 57

destination\_bucket (in *neuron\_morphology.pipeline.\_schemas.InputParameters* attribute), 60

determine\_slice\_flip () (in module *neuron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction*), 90

DF\_COLS (*neuron\_morphology.features.layer.layered\_point\_depths.LayeredPointDepthField* attribute), 36

dimension () (in module *neuron\_morphology.features.dimension*), 43

downsample (*neuron\_morphology.snap\_polygons.\_schemas.ImageOutputter* attribute), 65

downsample (*neuron\_morphology.snap\_polygons.\_schemas.OutputImageOutputter* attribute), 66

draw\_after () (in *neuron\_morphology.snap\_polygons.image\_outputter.ImageOutputter* method), 78

draw\_before () (in *neuron\_morphology.snap\_polygons.image\_outputter.ImageOutputter* method), 77

## E

early\_branch\_path() (in module neuron\_morphology.features.path), 47  
 earth\_movers\_distance() (in module neuron\_morphology.features.layer.layer\_histogram), 32  
 earth\_movers\_distance\_formatter (in module neuron\_morphology.feature\_extractor.feature\_writer), 19  
 EarthMoversDistanceInterpretation (class in neuron\_morphology.features.layer.layer\_histogram), 32  
 EarthMoversDistanceResult (class in neuron\_morphology.features.layer.layer\_histogram), 32  
 ensure() (neuron\_morphology.feature\_extractor.marked\_feature.MarkFeature class method), 24  
 ensure\_layers() (in module neuron\_morphology.features.layer.layer\_histogram), 32  
 ensure\_linestring() (in module neuron\_morphology.snap\_polygons.types), 79  
 ensure\_node\_types() (in module neuron\_morphology.features.layer.layer\_histogram), 32  
 ensure\_path() (in module neuron\_morphology.snap\_polygons.types), 79  
 ensure\_polygon() (in module neuron\_morphology.snap\_polygons.types), 79  
 ensure\_tuple() (in module neuron\_morphology.features.layer.layer\_histogram), 31  
 estimate\_scale\_correction() (in module neuron\_morphology.transforms.scale\_correction.compute\_scale\_correction), 86  
 euclidean\_distance() (neuron\_morphology.morphology.Morphology static method), 111  
 extent (neuron\_morphology.snap\_polygons.bounding\_box.BoundingBox attribute), 67  
 extract() (neuron\_morphology.feature\_extractor.feature\_extraction\_run.FeatureExtractionRun method), 12  
 extract() (neuron\_morphology.feature\_extractor.feature\_extractor.FeatureExtractor method), 13  
 extract\_multiple() (in module neuron\_morphology.feature\_extractor.\_\_main\_\_), 8  
 factory() (neuron\_morphology.feature\_extractor.mark.Mark class method), 20  
 Feature (in module neuron\_morphology.feature\_extractor.marked\_feature), 25  
 feature\_set (neuron\_morphology.feature\_extractor.\_schemas.InputParameter attribute), 10  
 FeatureExtractionRun (class in neuron\_morphology.feature\_extractor.feature\_extraction\_run), 12  
 FeatureExtractor (class in neuron\_morphology.feature\_extractor.feature\_extractor), 13  
 FeatureFn (in module neuron\_morphology.feature\_extractor.marked\_feature), 24  
 FeatureFormatter (class in neuron\_morphology.feature\_extractor.feature\_writer), 17  
 FeatureOutputCheck (in module neuron\_morphology.feature\_extractor.feature\_writer), 17  
 FeatureOutputHandler (in module neuron\_morphology.feature\_extractor.feature\_writer), 17  
 FeatureSpecialization (class in neuron\_morphology.feature\_extractor.feature\_specialization), 13  
 FeatureWriter (class in neuron\_morphology.feature\_extractor.feature\_writer), 16  
 fill\_gaps() (neuron\_morphology.snap\_polygons.geometries.Geometries method), 73  
 find\_closest\_path() (in module neuron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction), 89  
 find\_transition() (in module neuron\_morphology.snap\_polygons.cortex\_surfaces), 69  
 find\_vertical\_surfaces() (in module neuron\_morphology.snap\_polygons.geometries), 75  
 first\_met() (in module neuron\_morphology.snap\_polygons.cortex\_surfaces), 70  
 fname\_suffix() (in module neuron\_morphology.snap\_polygons.image\_outputter), 78  
 focal\_plane\_image\_series\_id (neuron\_morphology.snap\_polygons.\_from\_lims.FromLimsSchema attribute), 64  
 from\_gsv() (neuron\_morphology.features.layer.layered\_point\_depths.LayeredPointDepth class method), 36  
 from\_dataframe() (neuron\_morphology.features.layer.layered\_point\_depths.LayeredPointDepth class method), 36

## F

factory() (neuron\_morphology.feature\_extractor.feature\_specialization.FeatureSpecialization class method), 14  
 from\_dataframe() (neuron\_morphology.features.layer.layered\_point\_depths.LayeredPointDepth class method), 36



`neuron_morphology.features.layer.layered_point_depths.LayeredPointDepth.coordinates()`  
`class method`), 36 (in module `neuron_morphology.features.statistics.coordinates`),  
`from_dict()` (`neuron_morphology.transforms.affine_transform.AffineTransform`), 39  
`class method`), 94  
`from_hdf5()` (`neuron_morphology.features.layer.layered_point_depths.LayeredPointDepth`) (neuron\_morphology.morphology.Morphology  
`class method`), 36  
`from_list()` (`neuron_morphology.transforms.affine_transform.AffineTransform`) (neuron\_morphology.morphology.Morphology  
`class method`), 94  
`FromLimsSchema` (class in neuron\_morphology.snap\_polygons.\_from\_lims),  
64  
`FromLimsSource` (class in neuron\_morphology.snap\_polygons.\_from\_lims),  
65  
`Fs` (in module `neuron_morphology.feature_extractor.feature_specialization`), neuron\_morphology.morphology.Morphology  
13  
13  
**G**  
`generate_laplace_field()` (in module `neuron_morphology.transforms.streamline`), 98  
`Geometric` (class in neuron\_morphology.feature\_extractor.mark),  
20  
`Geometries` (class in neuron\_morphology.snap\_polygons.geometries),  
72  
`get_all_intact_apical()` (in module `neuron_morphology.lims_apical_queries`), 107  
`get_benchmark_apical()` (in module `neuron_morphology.lims_apical_queries`), 107  
`get_bifurcation_coordinates()`  
(in module `neuron_morphology.features.statistics.coordinates`),  
39  
`get_branch_order_for_node()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_branch_order_for_segment()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_branching_nodes()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_ccw_vertices()` (in module `neuron_morphology.transforms.geometry`), 98  
`get_ccw_vertices_from_two_lines()`  
(in module `neuron_morphology.transforms.geometry`), 97  
`get_children()` (neuron\_morphology.morphology.Morphology  
method), 108  
`get_children_of_node_by_types()` (neuron\_morphology.morphology.Morphology  
method), 108  
`get_compartment_length()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_compartment_midpoint()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_compartment_surface_area()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_compartment_volume()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_compartments()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_coordinates()` (in module `neuron_morphology.features.statistics.coordinates`),  
40  
`get_coordinates()` (neuron\_morphology.features.statistics.coordinates.COORD\_TYPE  
method), 38  
`get_credentials()` (in module `neuron_morphology.pipeline.post_data_to_s3`),  
60  
`get_data()` (in module `neuron_morphology.lims_apical_queries`), 107  
`get_dict()` (`neuron_morphology.snap_polygons._from_lims.FromLimsSchema`),  
65  
`get_dimensions()` (neuron\_morphology.morphology.Morphology  
method), 111  
`get_inputs_from_lims()` (in module `neuron_morphology.snap_polygons._from_lims`),  
64  
`get_leaf_nodes()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_max_id()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_morphology()` (in module `neuron_morphology.feature_extractor.data`),  
11  
`get_node_by_types()` (neuron\_morphology.morphology.Morphology  
method), 109  
`get_node_coordinates()` (in module `neuron_morphology.morphology.Morphology`), 109

`ron_morphology.features.statistics.coordinates)`, `get_vertices_from_two_lines()` (in module  
40 `neuron_morphology.transforms.geometry`), 97  
`get_node_intersections()` (in module `neuron_morphology.layered_point_depths.__main__`),  
54 `ron_morphology.feature_extractor._schemas.InputParameters`  
attribute), 11  
`get_non_soma_nodes()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 109 10  
`get_number_of_trees()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108 `gradient_field_file` (in `neuron_morphology.transforms.pia_wm_streamlines._schemas.OutputParameters`), 83  
`get_root()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108 `gradient_field_path` (in `neuron_morphology.layered_point_depths._schemas.DepthField`), 57  
`get_root_for_tree()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108 `gradient_path` (in `neuron_morphology.transforms.upright_angle._schemas.InputParameters`), 92  
`get_root_id()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108  
**H**  
`get_roots()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108 `handler` (`neuron_morphology.feature_extractor.feature_writer.FeatureWriter`), 17  
`get_roots_for_analysis()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108 `has_results()` (`neuron_morphology.validation.report.Report`), 102  
`get_roots_for_nodes()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108 `has_subkey()` (in module `neuron_morphology.feature_extractor.feature_writer`), 17  
`get_segment_length()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 109 `has_type()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 109 `heavy_output_path` (in `neuron_morphology.feature_extractor._schemas.InputParameters`), 10  
`get_segment_list()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 109 `height` (`neuron_morphology.snap_polygons.bounding_box.BoundingBox`), 67  
`get_snapped_polys()` (in module `neuron_morphology.snap_polygons.geometries`),  
75 `histogram_earth_movers_distance()` (in module `neuron_morphology.features.layer.layer_histogram`), 33  
`get_soma()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108  
`get_soma_marker_from_marker_file()` (`neuron_morphology.transforms.scale_correction.compute_scale_correction`), 87  
(in module `neuron_morphology.transforms.scale_correction.compute_scale_correction`), 87  
`get_tilt_correction()` (in module `neuron_morphology.transforms.tilt_correction.compute_tilt_correction`),  
89  
**I**  
`get_tip_coordinates()` (in module `neuron_morphology.features.statistics.coordinates`),  
39 `identifier` (`neuron_morphology.feature_extractor._schemas.ReconstructionSchema`), 10  
`get_tree_list()` (`neuron_morphology.morphology.Morphology` (class in `neuron_morphology.feature_extractor._schemas`),  
method), 108 `Image` (class in `neuron_morphology.snap_polygons._schemas`), 65  
`get_upright_angle()` (in module `neuron_morphology.transforms.upright_angle.compute_angle`),  
93 `image_output_root` (in `neuron_morphology.snap_polygons._from_lims.FromLimsSchema`), 64

ImageOutputter (class in neuron\_morphology.snap\_polygons.image\_outputter), 20  
 InvalidMarkerFile, 103  
 InvalidMorphology, 103  
 images (neuron\_morphology.snap\_polygons.\_schemas.InputParameters.beginning\_of\_segment() (neuron\_morphology.morphology.Morphology attribute), 66  
 images (neuron\_morphology.snap\_polygons.\_schemas.OutputParameters.end\_of\_segment() (neuron\_morphology.morphology.Morphology attribute), 67  
 is\_node\_at\_end\_of\_segment() (neuron\_morphology.morphology.Morphology method), 109  
 input\_path (neuron\_morphology.snap\_polygons.\_schemas.ImageOutputter.get\_image\_id() (neuron\_morphology.morphology.Morphology attribute), 65  
 input\_path (neuron\_morphology.snap\_polygons.\_schemas.InputParameters.get\_image\_id() (neuron\_morphology.morphology.Morphology attribute), 66  
 input\_sw (neuron\_morphology.transforms.affine\_transformer.\_schemas.ApplyAffineSchema.apply() (neuron\_morphology.transforms.affine\_transformer.\_schemas.ApplyAffineSchema attribute), 81  
 is\_valid() (neuron\_morphology.feature\_extractor.\_schemas.ReferenceLayerDepth attribute), 9  
 InputParameters (class in neuron\_morphology.feature\_extractor.\_schemas), 10  
 item (in module neuron\_morphology.feature\_extractor.run\_feature\_extraction), 26  
 InputParameters (class in neuron\_morphology.layered\_point\_depths.\_schemas), 57  
 InputParameters (class in neuron\_morphology.pipeline.\_schemas), 59  
 InputParameters (class in neuron\_morphology.snap\_polygons.\_schemas), 66  
 key (neuron\_morphology.feature\_extractor.\_schemas.ReferenceLayerDepth attribute), 9  
 known\_feature\_sets (in module neuron\_morphology.feature\_extractor.run\_feature\_extraction), 26  
 InputParameters (class in neuron\_morphology.transforms.scale\_correction.\_schemas), 85  
 kwargs (neuron\_morphology.feature\_extractor.feature\_specialization.AllInputParameters attribute), 15  
 InputParameters (class in neuron\_morphology.transforms.tilt\_correction.\_schemas), 88  
 kwargs (neuron\_morphology.feature\_extractor.feature\_specialization.AllInputParameters attribute), 15  
 InputParameters (class in neuron\_morphology.transforms.upright\_angle.\_schemas), 91  
 kwargs (neuron\_morphology.feature\_extractor.feature\_specialization.AllInputParameters attribute), 14  
 inputs (neuron\_morphology.feature\_extractor.\_schemas.OutputParameters attribute), 15  
 inputs (neuron\_morphology.layered\_point\_depths.\_schemas.OutputParameters attribute), 58  
 inputs (neuron\_morphology.snap\_polygons.\_schemas.OutputParameters attribute), 66  
 inputs (neuron\_morphology.transforms.affine\_transformer.\_schemas.ApplyAffineSchema attribute), 82  
 inputs (neuron\_morphology.transforms.pia\_wm\_streamlines.\_schemas.OutputParameters attribute), 83  
 inputs (neuron\_morphology.transforms.scale\_correction.\_schemas.OutputParameters attribute), 85  
 inputs (neuron\_morphology.transforms.tilt\_correction.\_schemas.OutputParameters attribute), 88  
 inputs (neuron\_morphology.transforms.upright\_angle.\_schemas.OutputParameters attribute), 92  
 interpretation (neuron\_morphology.features.layer.layer\_histogram Earth Moving Distance Result attribute), 32  
 Intrinsic (class in neuron\_morphology.feature\_extractor.mark), 39  
 kwargs (neuron\_morphology.features.statistics.coordinates.TipSpec attribute), 39



## L

Layer (class in neuron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction), 91  
 layer\_order (neuron\_morphology.snap\_polygons.\_schemas.InputParameters (in module neuron\_morphology.transforms.upright\_angle.compute\_angle), attribute), 66  
 layer\_polygons (neuron\_morphology.snap\_polygons.\_schemas.InputParameters (in module neuron\_morphology.validation.validate\_reconstruction), attribute), 66  
 layered\_point\_depths\_path (neuron\_morphology.feature\_extractor.\_schemas.Reconstruction (in module neuron\_morphology.snap\_polygons.image\_outputter), attribute), 10  
 LayeredPointDepths (class in neuron\_morphology.features.layer.layered\_point\_depths), 36  
 LayerHistogram (class in neuron\_morphology.features.layer.layer\_histogram), 32  
 layers (neuron\_morphology.layered\_point\_depths.\_schemas.InputParameters (class in neuron\_morphology.feature\_extractor.mark), attribute), 58  
 level (neuron\_morphology.validation.result.MarkerValidationError (in module neuron\_morphology.validation.result), attribute), 103  
 level (neuron\_morphology.validation.result.NodeValidationError (in module neuron\_morphology.validation.result), attribute), 103  
 LineType (in module neuron\_morphology.snap\_polygons.types), 79  
 logger (in module neuron\_morphology.validation.validate\_reconstruction), 105

## M

M (in module neuron\_morphology.feature\_extractor.marked\_feature), 24  
 main () (in module neuron\_morphology.feature\_extractor.\_\_main\_\_), 8  
 main () (in module neuron\_morphology.layered\_point\_depths.\_\_main\_\_), 56  
 main () (in module neuron\_morphology.pipeline.post\_data\_to\_s3), 61  
 main () (in module neuron\_morphology.snap\_polygons.\_\_main\_\_), 61  
 main () (in module neuron\_morphology.transforms.affine\_transformer.apply\_affine\_transform), 82  
 main () (in module neuron\_morphology.transforms.pia\_wm\_streamlines.calculate\_pia\_wm\_streamlines), 84  
 main () (in module neuron\_morphology.transforms.scale\_correction.compute\_scale\_correction), 87  
 marker\_file (neuron\_morphology.pipeline.\_schemas.InputParameters (in module neuron\_morphology.pipeline), attribute), 60  
 marker\_path (neuron\_morphology.transforms.scale\_correction.\_schemas.InputParameters (in module neuron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters), attribute), 85  
 marker\_path (neuron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters (in module neuron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters), attribute), 88  
 marker\_validators (in module neuron\_morphology.validation), 105  
 MarkerValidationError (class in neuron\_morphology.validation.result), 103  
 marks (neuron\_morphology.feature\_extractor.feature\_specialization.AllNeurons (in module neuron\_morphology.feature\_extractor.feature\_specialization), attribute), 15  
 marks (neuron\_morphology.feature\_extractor.feature\_specialization.AllNeurons (in module neuron\_morphology.feature\_extractor.feature\_specialization), attribute), 15  
 marks (neuron\_morphology.feature\_extractor.feature\_specialization.ApicalNeurons (in module neuron\_morphology.feature\_extractor.feature\_specialization), attribute), 15  
 marks (neuron\_morphology.feature\_extractor.feature\_specialization.ApicalNeurons (in module neuron\_morphology.feature\_extractor.feature\_specialization), attribute), 14  
 marks (neuron\_morphology.feature\_extractor.feature\_specialization.AxonNeurons (in module neuron\_morphology.feature\_extractor.feature\_specialization), attribute), 15  
 marks (neuron\_morphology.feature\_extractor.feature\_specialization.AxonNeurons (in module neuron\_morphology.feature\_extractor.feature\_specialization), attribute), 14  
 marks (neuron\_morphology.feature\_extractor.feature\_specialization.BasalNeurons (in module neuron\_morphology.feature\_extractor.feature\_specialization), attribute), 15

marks (neuron\_morphology.feature\_extractor.feature\_specialization.BasalDendriteSpec attribute), 14

marks (neuron\_morphology.feature\_extractor.feature\_specialization.DendriteDynamicsSpec attribute), 15

marks (neuron\_morphology.feature\_extractor.feature\_specialization.DendriteSpec attribute), 15

marks (neuron\_morphology.feature\_extractor.feature\_specialization.FeatureSpec attribute), 14

marks (neuron\_morphology.features.statistics.coordinates.BifurcationSpec attribute), 38

marks (neuron\_morphology.features.statistics.coordinates.CompartmentSpec attribute), 39

marks (neuron\_morphology.features.statistics.coordinates.ModeSpec attribute), 38

marks (neuron\_morphology.features.statistics.coordinates.TipSpec attribute), 39

max\_branch\_order() (in module neuron\_morphology.features.intrinsic), 46

max\_euclidean\_distance() (in module neuron\_morphology.features.size), 50

max\_iter(neuron\_morphology.layered\_point\_depths.\_schemas.InputParameters attribute), 58

max\_path\_distance() (in module neuron\_morphology.features.path), 47

mean\_bifurcation\_angle\_local() (in module neuron\_morphology.features.branching.bifurcations), 30

mean\_bifurcation\_angle\_remote() (in module neuron\_morphology.features.branching.bifurcations), 30

mean\_contraction() (in module neuron\_morphology.features.path), 48

mean\_diameter() (in module neuron\_morphology.features.size), 49

mean\_fragmentation() (in module neuron\_morphology.features.intrinsic), 45

mean\_parent\_daughter\_ratio() (in module neuron\_morphology.features.size), 50

mesh\_res(neuron\_morphology.transforms.pia\_wm\_streamlines.\_schemas.PiaWmStreamlineSchema attribute), 83

message(neuron\_morphology.validation.result.MarkerValidationError attribute), 103

message(neuron\_morphology.validation.result.NodeValidationError attribute), 103

midpoint() (neuron\_morphology.morphology.Morphology static method), 111

moments() (in module neuron\_morphology.features.statistics.moments), 40

moments\_along\_max\_distance\_projection() (in module neuron\_morphology.features.statistics.moments\_along\_max\_distance\_projection), 40

Morphology (class in neuron\_morphology), 107

morphology\_from\_swc() (in module neuron\_morphology.swc\_io), 112

morphology\_statistics() (in module neuron\_morphology.validation.morphology\_statistics), 101

Mr (in module neuron\_morphology.feature\_extractor.mark), 20

multipolygon\_error\_threshold (neuron\_morphology.snap\_polygons.\_schemas.InputParameters attribute), 66

MultiSurfaceResolvertype (in module neuron\_morphology.snap\_polygons.types), 79

name(neuron\_morphology.feature\_extractor.feature\_specialization.AllNeur attribute), 15

name(neuron\_morphology.feature\_extractor.feature\_specialization.AllNeur attribute), 15

name(neuron\_morphology.feature\_extractor.feature\_specialization.Apical attribute), 15

name(neuron\_morphology.feature\_extractor.feature\_specialization.Apical attribute), 14

name(neuron\_morphology.feature\_extractor.feature\_specialization.AxonC attribute), 15

name(neuron\_morphology.feature\_extractor.feature\_specialization.AxonSp attribute), 14

name(neuron\_morphology.feature\_extractor.feature\_specialization.BasalD attribute), 15

name(neuron\_morphology.feature\_extractor.feature\_specialization.BasalD attribute), 14

name(neuron\_morphology.feature\_extractor.feature\_specialization.Dendri attribute), 15

name(neuron\_morphology.feature\_extractor.feature\_specialization.Dendri attribute), 14

name(neuron\_morphology.feature\_extractor.feature\_specialization.Feature attribute), 14

name(neuron\_morphology.feature\_extractor.feature\_writer.FeatureFormat attribute), 17

name(neuron\_morphology.features.statistics.coordinates.BifurcationSpec attribute), 38

name(neuron\_morphology.features.statistics.coordinates.CompartmentSpe attribute), 39

```

name (neuron_morphology.features.statistics.coordinates.NodeSpec
    attribute), 38
name (neuron_morphology.features.statistics.coordinates.TipSpec
    attribute), 39
name (neuron_morphology.layered_point_depths._schemas.Layer
    attribute), 57
name (neuron_morphology.pipeline._schemas.S3LandingBucket
    attribute), 59
name (neuron_morphology.snap_polygons._schemas.SimpleGeometry
    attribute), 65
names (neuron_morphology.feature_extractor._schemas.ReferenceLayerDepth
    attribute), 9
neighbors (neuron_morphology.transforms.upright_angle._schemas.InputParameters
    attribute), 92
nested_specialize() (in module neu- neuron_morphology.features.layer.layered_point_dept
ron_morphology.feature_extractor.marked_feature), (module), 36
25 neuron_morphology.features.layer.reference_layer_de
NEURITE_COMPARISON_SPECIALIZATIONS (module), 36
(in module neu- neuron_morphology.features.path (module),
ron_morphology.feature_extractor.feature_specialization), 46
15 neuron_morphology.features.size (module),
NEURITE_SPECIALIZATIONS (in module neu- 48
ron_morphology.feature_extractor.feature_specialization), neuron_morphology.features.soma (module),
15 51
NeuriteTypeComparison (class in neu- neuron_morphology.features.statistics
ron_morphology.feature_extractor.mark), (module), 37
22 neuron_morphology.features.statistics.coordinates
neuron_morphology (module), 7 (module), 37
neuron_morphology.constants (module), 106 neuron_morphology.features.statistics.moments
neuron_morphology.feature_extractor (module), 40
(module), 7 neuron_morphology.features.statistics.moments_along
neuron_morphology.feature_extractor.__main__ (module), 41
(module), 7 neuron_morphology.features.statistics.overlap
neuron_morphology.feature_extractor._schemas (module), 41
(module), 9 neuron_morphology.layered_point_depths
neuron_morphology.feature_extractor.data (module), 52
(module), 11 neuron_morphology.layered_point_depths.__main__
neuron_morphology.feature_extractor.feature_ext (module), 52
(module), 12 neuron_morphology.layered_point_depths._schemas
neuron_morphology.feature_extractor.feature_ext (module), 56
(module), 12 neuron_morphology.lims_apical_queries
neuron_morphology.feature_extractor.feature_spec (module), 106
(module), 13 neuron_morphology.marker (module), 107
neuron_morphology.feature_extractor.feature_neuron (module), 107
(module), 16 neuron_morphology.morphology_builder
neuron_morphology.feature_extractor.mark (module), 111
(module), 19 neuron_morphology.pipeline (module), 58
neuron_morphology.feature_extractor.marked_feat (module), 58
(module), 23 neuron_morphology.pipeline._schemas
neuron_morphology.feature_extractor.run_feature_extractor (module), 60
(module), 26 neuron_morphology.pipeline.post_data_to_s3
neuron_morphology.feature_extractor.utilites (module), 61
(module), 28 neuron_morphology.snap_polygons (module),
neuron_morphology.features (module), 28 neuron_morphology.snap_polygons.__main__

```

(module), 61  
 neuron\_morphology.snap\_polygons.\_from\_lines (module), 62  
 neuron\_morphology.snap\_polygons.\_schemas (module), 65  
 neuron\_morphology.snap\_polygons.bounding\_box (module), 67  
 neuron\_morphology.snap\_polygons.cortex\_surfaces (module), 68  
 neuron\_morphology.snap\_polygons.geometries (module), 70  
 neuron\_morphology.snap\_polygons.image\_outputter (module), 76  
 neuron\_morphology.snap\_polygons.types (module), 79  
 neuron\_morphology.swc\_io (module), 112  
 neuron\_morphology.transforms (module), 80  
 neuron\_morphology.transforms.affine\_transform (module), 94  
 neuron\_morphology.transforms.affine\_transformer (module), 80  
 neuron\_morphology.transforms.affine\_transformer\_1d (module), 80  
 neuron\_morphology.transforms.affine\_transformer\_2d (module), 82  
 neuron\_morphology.transforms.geometry (module), 97  
 neuron\_morphology.transforms.pia\_wm\_streamlines (module), 82  
 neuron\_morphology.transforms.pia\_wm\_streamlines\_1d (module), 82  
 neuron\_morphology.transforms.pia\_wm\_streamlines\_2d (module), 83  
 neuron\_morphology.transforms.scale\_correction (module), 85  
 neuron\_morphology.transforms.scale\_correction.attribute (module), 85  
 neuron\_morphology.transforms.scale\_correction.attribute\_1d (module), 86  
 neuron\_morphology.transforms.streamline (module), 98  
 neuron\_morphology.transforms.tilt\_correction (module), 87  
 neuron\_morphology.transforms.tilt\_correction.attribute (module), 87  
 neuron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction (module), 89  
 neuron\_morphology.transforms.transform\_base (module), 99  
 neuron\_morphology.transforms.upright\_angle (module), 91  
 neuron\_morphology.transforms.upright\_angle.\_schemas (module), 91  
 neuron\_morphology.transforms.upright\_angle.compute\_angle (module), 92  
 neuron\_morphology.validation (module), 100  
 neuron\_morphology.validation.bits\_validation (module), 100  
 neuron\_morphology.validation.marker\_validation (module), 100  
 neuron\_morphology.validation.morphology\_statistics (module), 101  
 neuron\_morphology.validation.radius\_validation (module), 101  
 neuron\_morphology.validation.report (module), 102  
 neuron\_morphology.validation.resample\_validation (module), 102  
 neuron\_morphology.validation.result (module), 103  
 neuron\_morphology.validation.structure\_validation (module), 104  
 neuron\_morphology.validation.type\_validation (module), 104  
 neuron\_morphology.validation.validate\_reconstruction (module), 105  
 neuron\_morphology.vis (module), 105  
 neuron\_morphology.validation.affine\_transformer (module), 106  
 neuron\_reconstruction\_id (neuron\_morphology.pipeline.\_schemas.InputParameters attribute), 60  
 next\_id (neuron\_morphology.morphology\_builder.MorphologyBuilder attribute), 61  
 NicePathType (in module neuron\_morphology.snap\_polygons.types), 79  
 NO\_RECONSTRUCTION (in module neuron\_morphology.constants), 106  
 NODE (neuron\_morphology.features.statistics.coordinates.COORD\_TYPE attribute), 38  
 node (neuron\_morphology.transforms.upright\_angle.\_schemas.InputParameters attribute), 91  
 node\_by\_id () (neuron\_morphology.morphology.Morphology method), 108  
 node\_ids (neuron\_morphology.validation.result.NodeValidationError attribute), 103  
 NodeSpec (class in neuron\_morphology.features.statistics.coordinates), 38  
 NodeValidationError (class in neuron\_morphology.validation.result), 103  
 normalized\_depth\_histogram () (in module neuron\_morphology.features.layer.layer\_histogram), 91  
 normalized\_depth\_histogram\_formatter (in module neuron\_morphology.validation.result), 103

[neuron\\_morphology.feature\\_extractor.feature\\_writer\)](#)[OutputParameters](#) (class in [neuron\\_morphology.layered\\_point\\_depths.\\_schemas](#)),  
[19](#)  
[normalized\\_depth\\_histogram\\_within\\_layer\(\)](#) [58](#)  
 (in module [neuron\\_morphology.features.layer.layer\\_histogram](#)), [OutputParameters](#) (class in [neuron\\_morphology.snap\\_polygons.\\_schemas](#)),  
[35](#) [66](#)  
[normalized\\_depth\\_histograms\\_across\\_layer\(\)](#) [OutputParameters](#) (class in [neuron\\_morphology.transforms.affine\\_transformer.\\_schemas](#)),  
 (in module [neuron\\_morphology.features.layer.layer\\_histogram](#)), [82](#)  
[34](#)  
[num\\_branches\(\)](#) (in module [neuron\\_morphology.features.intrinsic](#)), [OutputParameters](#) (class in [neuron\\_morphology.transforms.pia\\_wm\\_streamlines.\\_schemas](#)),  
[45](#) [83](#)  
[num\\_nodes\(\)](#) (in module [neuron\\_morphology.features.intrinsic](#)), [OutputParameters](#) (class in [neuron\\_morphology.transforms.scale\\_correction.\\_schemas](#)),  
[44](#) [85](#)  
[num\\_outer\\_bifurcations\(\)](#) (in module [neuron\\_morphology.features.branching.bifurcations](#)), [OutputParameters](#) (class in [neuron\\_morphology.transforms.tilt\\_correction.\\_schemas](#)),  
[29](#) [88](#)  
[num\\_processes](#) ([neuron\\_morphology.feature\\_extractor.\\_schemas.InputParameters](#) (class in [neuron\\_morphology.transforms.upright\\_angle.\\_schemas](#)),  
[attribute](#)), [11](#) [92](#)  
[num\\_tips\(\)](#) (in module [neuron\\_morphology.features.intrinsic](#)), [44](#)  
[numpy\\_array\\_formatter](#) (in module [neuron\\_morphology.feature\\_extractor.feature\\_writer](#)), [42](#)  
[19](#)  
[O](#)  
[overlap\(\)](#) (in module [neuron\\_morphology.features.statistics.overlap](#)),  
[OneEmpty](#) ([neuron\\_morphology.features.layer.layer\\_histogram](#) (class in [neuron\\_morphology.snap\\_polygons.\\_schemas](#)), [OutputImage](#)  
[attribute](#)), [32](#) [66](#)  
[only\\_marks](#) ([neuron\\_morphology.feature\\_extractor.\\_schemas.InputParameters](#) (class in [neuron\\_morphology.snap\\_polygons.\\_schemas](#)), [Image](#)  
[attribute](#)), [10](#) [65](#)  
[origin](#) ([neuron\\_morphology.snap\\_polygons.bounding\\_box.BoundingBox](#) (class in [neuron\\_morphology.snap\\_polygons.image\\_outputter.ImageOutputter](#)  
[attribute](#)), [67](#) [77](#)  
[output\\_dir](#) ([neuron\\_morphology.transforms.pia\\_wm\\_streamlines.\\_schemas.PiaWmStreamlineSchema](#)  
[attribute](#)), [83](#)  
[P](#)  
[output\\_path](#) ([neuron\\_morphology.layered\\_point\\_depths.\\_schemas.InputParameters](#) (class in [neuron\\_morphology.features.size](#)), [49](#)  
[attribute](#)), [58](#)  
[output\\_path](#) ([neuron\\_morphology.layered\\_point\\_depths.\\_schemas.OutputParameters](#) (class in [neuron\\_morphology.morphology\\_builder.MorphologyBuilder](#)  
[attribute](#)), [58](#) [111](#)  
[output\\_path](#) ([neuron\\_morphology.snap\\_polygons.\\_schemas.Image](#) (class in [neuron\\_morphology.morphology.Morphology](#)  
[attribute](#)), [65](#) [108](#)  
[output\\_path](#) ([neuron\\_morphology.snap\\_polygons.\\_schemas.OutputImage](#) (class in [neuron\\_morphology.morphology.Morphology](#)  
[attribute](#)), [66](#) [108](#)  
[output\\_path](#) ([neuron\\_morphology.snap\\_polygons.\\_schemas.OutputImage](#) (class in [neuron\\_morphology.morphology.Morphology](#)  
[attribute](#)), [66](#) [108](#)  
[output\\_swc](#) ([neuron\\_morphology.transforms.affine\\_transformer.\\_schemas.ApplyAffineSchema](#) (class in [neuron\\_morphology.transforms.affine\\_transformer.\\_schemas](#)),  
[attribute](#)), [81](#)  
[output\\_table\\_path](#) ([neuron\\_morphology.feature\\_extractor.\\_schemas.InputParameters](#) (class in [neuron\\_morphology.snap\\_polygons.\\_from\\_lims.PostgresInputParameters](#)  
[attribute](#)), [11](#) [64](#)  
[OutputImage](#) (class in [neuron\\_morphology.snap\\_polygons.\\_schemas](#)), [path](#) ([neuron\\_morphology.pipeline.\\_schemas.PathResolution](#)  
[66](#) [attribute](#)), [59](#)  
[OutputParameters](#) (class in [neuron\\_morphology.feature\\_extractor.\\_schemas](#)), [path](#) ([neuron\\_morphology.snap\\_polygons.\\_schemas.SimpleGeometry](#)  
[11](#) [attribute](#)), [65](#)  
[PathResolution](#) (class in [neuron\\_morphology.pipeline.\\_schemas](#)), [59](#)



[PathsType](#) (in module `neuron_morphology.snap_polygons.types`), 79  
[PathType](#) (in module `neuron_morphology.snap_polygons.types`), 79  
[Pia](#) (`neuron_morphology.pipeline._schemas.PrimaryBoundaries` attribute), 59  
[pia\\_fixed\\_value](#) (`neuron_morphology.transforms.pia_wm_streamlines._schemas.PiaWmStreamlineSchema` attribute), 83  
[pia\\_path\\_str](#) (`neuron_morphology.transforms.pia_wm_streamlines._schemas.PiaWmStreamlineSchema` attribute), 83  
[pia\\_side](#) (`neuron_morphology.features.layer.reference_layer_depths.ReferenceLayerDepths` attribute), 37  
[pia\\_sign](#) (`neuron_morphology.layered_point_depths._schemas.DepthField` attribute), 57  
[pia\\_surface](#) (`neuron_morphology.layered_point_depths._schemas.Layer` attribute), 57  
[pia\\_surface](#) (`neuron_morphology.snap_polygons._schemas.InputParameters` attribute), 66  
[PiaWmStreamlineSchema](#) (class in `neuron_morphology.transforms.pia_wm_streamlines._schemas`), 83  
[plot\\_cortical\\_boundary\(\)](#) (in module `neuron_morphology.vis.morphovis`), 106  
[plot\\_depth\\_field\(\)](#) (in module `neuron_morphology.vis.morphovis`), 106  
[plot\\_gradient\\_field\(\)](#) (in module `neuron_morphology.vis.morphovis`), 106  
[plot\\_morphology\\_xy\(\)](#) (in module `neuron_morphology.vis.morphovis`), 106  
[plot\\_morphology\\_zy\(\)](#) (in module `neuron_morphology.vis.morphovis`), 106  
[plot\\_soma\(\)](#) (in module `neuron_morphology.vis.morphovis`), 106  
[polygons](#) (`neuron_morphology.snap_polygons._schemas.OutputParameters` attribute), 67  
[PolyType](#) (in module `neuron_morphology.snap_polygons.types`), 79  
[port](#) (`neuron_morphology.snap_polygons._from_lims.PostgresInputConfigSchema` attribute), 64  
[post\\_object\\_to\\_s3\(\)](#) (in module `neuron_morphology.pipeline.post_data_to_s3`), 60  
[PostgresInputConfigSchema](#) (class in `neuron_morphology.snap_polygons._from_lims`), 64  
[primary\\_boundaries](#) (`neuron_morphology.pipeline._schemas.InputParameters` attribute), 60  
[PrimaryBoundaries](#) (class in `neuron_morphology.pipeline._schemas`), 59  
[process\\_earth\\_movers\\_distance\(\)](#) (in module `neuron_morphology.feature_extractor.feature_extractor`), 9  
[process\\_feature\(\)](#) (`neuron_morphology.feature_extractor.feature_extractor.FeatureWriter` method), 17  
[prune\\_two\\_lines\(\)](#) (in module `neuron_morphology.transforms.geometry`), 97  
[query\\_for\\_cortical\\_surfaces\(\)](#) (`neuron_morphology.snap_polygons._from_lims.ReferenceLayerDepths` method), 63  
[query\\_for\\_image\\_dims\(\)](#) (in module `neuron_morphology.snap_polygons._from_lims.ReferenceLayerDepths` method), 64  
[query\\_for\\_layer\\_images\(\)](#) (in module `neuron_morphology.snap_polygons._from_lims.ReferenceLayerDepths` method), 64  
[query\\_for\\_layer\\_polygons\(\)](#) (in module `neuron_morphology.snap_polygons._from_lims.ReferenceLayerDepths` method), 62  
[QueryEngineType](#) (in module `neuron_morphology.snap_polygons._from_lims.ReferenceLayerDepths` method), 62  
[rasterize\(\)](#) (in module `neuron_morphology.snap_polygons.geometries.Geometries` method), 74  
[rasterize\(\)](#) (`neuron_morphology.snap_polygons.geometries.Geometries` method), 72  
[read\(\)](#) (`neuron_morphology.features.layer.layered_point_depths.LayeredPointDepths` class method), 36  
[read\\_image\(\)](#) (in module `neuron_morphology.snap_polygons.image_outputter`), 78  
[read\\_jp2\(\)](#) (in module `neuron_morphology.snap_polygons.image_outputter`), 78  
[read\\_marker\\_file\(\)](#) (in module `neuron_morphology.marker`), 107  
[read\\_soma\\_marker\(\)](#) (in module `neuron_morphology.transforms.tilt_correction.compute_tilt_correction`), 90  
[read\\_swc\(\)](#) (in module `neuron_morphology.swc_io`), 112  
[read\\_with\\_ndimage\(\)](#) (in module `neuron_morphology.snap_polygons.image_outputter`), 78  
[Reconstruction](#) (class in `neuron_morphology.feature_extractor._schemas`), 9

reconstructions (neuron\_morphology.feature\_extractor.schemas.InputParameters.LayerAnnotations (class in neuron\_morphology.feature\_extractor.mark), 20  
 attribute), 10  
 reference\_layer\_depths (neuron\_morphology.feature\_extractor.schemas.GlobalParameters.LayeredPointDepths (class in neuron\_morphology.feature\_extractor.mark), 20  
 attribute), 10  
 ReferenceLayerDepths (class in neuron\_morphology.feature\_extractor.schemas), RequiresRadii (class in neuron\_morphology.feature\_extractor.mark), 9  
 22  
 ReferenceLayerDepths (class in neuron\_morphology.features.layer.reference\_layer\_depths), RequiresReferenceLayerDepths (class in neuron\_morphology.feature\_extractor.mark), 36  
 RequiresRegularPointSpacing (class in neuron\_morphology.feature\_extractor.mark), 23  
 region (neuron\_morphology.pipeline.\_schemas.S3LandingBucket RequiresRelativeSomaDepth (class in neuron\_morphology.feature\_extractor.mark), 59  
 attribute), 59  
 register\_features() (neuron\_morphology.feature\_extractor.feature\_extractor.FeatureExtractor method), 13  
 RequiresRoot (class in neuron\_morphology.feature\_extractor.mark), 22  
 register\_formatters() (neuron\_morphology.feature\_extractor.feature\_writer.FeatureWriter method), 17  
 RequiresSoma (class in neuron\_morphology.feature\_extractor.mark), 21  
 register\_polygon() (neuron\_morphology.snap\_polygons.geometries.Geometries resolution (neuron\_morphology.pipeline.\_schemas.PathResolution attribute), 59  
 method), 72  
 resolution (neuron\_morphology.transforms.pia\_wm\_streamlines.\_schemas.PathResolution attribute), 83  
 register\_polygons() (neuron\_morphology.snap\_polygons.geometries.Geometries resolve\_reference\_layer\_depths() (in module neuron\_morphology.feature\_extractor.run\_feature\_extraction), 72  
 method), 72  
 attribute), 32  
 register\_surface() (neuron\_morphology.snap\_polygons.geometries.Geometries results (neuron\_morphology.feature\_extractor.schemas.OutputParameters attribute), 11  
 method), 72  
 attribute), 32  
 register\_surfaces() (neuron\_morphology.snap\_polygons.geometries.Geometries root() (neuron\_morphology.morphology\_builder.MorphologyBuilder method), 111  
 method), 72  
 attribute), 32  
 RegistrableFeature (in module neuron\_morphology.feature\_extractor.feature\_extractor), rotation\_from\_angle() (in module neuron\_morphology.transforms.affine\_transform), 13  
 96  
 remove\_duplicates() (in module neuron\_morphology.snap\_polygons.cortex\_surfaces), round() (neuron\_morphology.snap\_polygons.bounding\_box.BoundingBox method), 68  
 70  
 Report (class in neuron\_morphology.validation.report), 102  
 required\_marks (neuron\_morphology.feature\_extractor.schemas.InputParameters.run\_feature\_extraction() (in module neuron\_morphology.feature\_extractor.run\_feature\_extraction), 27  
 attribute), 11  
 RequiresApical (class in neuron\_morphology.feature\_extractor.mark), run\_layered\_point\_depths() (in module neuron\_morphology.layered\_point\_depths.\_\_main\_\_), 21  
 56  
 RequiresAxon (class in neuron\_morphology.feature\_extractor.mark), run\_scale\_correction() (in module neuron\_morphology.transforms.scale\_correction.compute\_scale\_correction), 21  
 87  
 RequiresBasal (class in neuron\_morphology.feature\_extractor.mark), run\_snap\_polygons() (in module neuron\_morphology.snap\_polygons.\_\_main\_\_), 21  
 61  
 RequiresDendrite (class in neuron\_morphology.feature\_extractor.mark), run\_streamlines() (in module neuron\_morphology.transforms.affine\_transform), 21  
 96

ron\_morphology.transforms.pia\_wm\_streamlines.calculate\_pia\_wm\_streamlines),  
 84 slice\_image\_flip (neu-  
 run\_tilt\_correction() (in module neu- ron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters  
 ron\_morphology.transforms.tilt\_correction.compute\_tilt\_correction), 88  
 90 slice\_transform (neu-  
 run\_upright\_angle() (in module neu- ron\_morphology.pipeline.\_schemas.InputParameters  
 ron\_morphology.transforms.upright\_angle.compute\_angle), attribute), 60  
 93 slice\_transform\_dict (neu-  
 ron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters  
 attribute), 88  
**S**  
 S3LandingBucket (class in neu- slice\_transform\_list (neu-  
 ron\_morphology.pipeline.\_schemas), 59 ron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters  
 safe\_linemerge() (in module neu- attribute), 88  
 ron\_morphology.snap\_polygons.geometries), slope\_linear\_regression\_branch\_order\_avg\_radius()  
 72 (in module neu-  
 scale(neuron\_morphology.features.layer.reference\_layer\_depths.ReferenceLayerDepths  
 attribute), 37 102  
 scale\_correction (neu- solve\_laplace\_2d() (in module neu-  
 ron\_morphology.transforms.scale\_correction.\_schemas.OutputParameters),  
 attribute), 86 98  
 scale\_transform (neu- SOMA (in module neuron\_morphology.constants), 106  
 ron\_morphology.transforms.scale\_correction.\_schemas.OutputParameters  
 attribute), 86 attribute), 59  
 select\_features() (neu- soma\_depth(neuron\_morphology.transforms.scale\_correction.\_schemas  
 ron\_morphology.feature\_extractor.feature\_extraction\_run.FeatureExtractionRun  
 method), 12 attribute), 85  
 select\_largest\_subpolygon() (in module neu- soma\_origin(neuron\_morphology.layered\_point\_depths.\_schemas.Depth  
 ron\_morphology.snap\_polygons.geometries), attribute), 57  
 71 soma\_path\_str (neu-  
 select\_marks() (neu- attribute), 83  
 ron\_morphology.feature\_extractor.feature\_extraction\_run.FeatureExtractionRun  
 method), 12 106  
 sequential() (neu- SpecializationOption (in module neu-  
 ron\_morphology.features.layer.reference\_layer\_depths.ReferenceLayerDepths  
 class method), 37 14  
 serialize() (neuron\_morphology.feature\_extractor.feature\_extraction\_run.FeatureExtractionRun module neu-  
 method), 12 ron\_morphology.feature\_extractor.feature\_specialization),  
 setup\_data() (in module neu- 14  
 ron\_morphology.feature\_extractor.run\_feature\_extraction, specializationSets (in module neu-  
 27 ron\_morphology.feature\_extractor.feature\_specialization),  
 setup\_interpolator() (in module neu- 14  
 ron\_morphology.layered\_point\_depths.\_\_main\_\_)specialize() (in module neu-  
 53 ron\_morphology.feature\_extractor.marked\_feature),  
 setup\_layers() (in module neu- 25  
 ron\_morphology.layered\_point\_depths.\_\_main\_\_)specialize() (neu-  
 56 ron\_morphology.feature\_extractor.marked\_feature.MarkedFeature  
 method), 24  
 shared\_faces() (in module neu-  
 ron\_morphology.snap\_polygons.geometries), specimen\_id(neuron\_morphology.pipeline.\_schemas.InputParameters  
 75 attribute), 60  
 SimpleGeometry (class in neu- split\_pathstring() (in module neu-  
 ron\_morphology.snap\_polygons.\_schemas), ron\_morphology.snap\_polygons.types), 80  
 65 step(neuron\_morphology.transforms.upright\_angle.\_schemas.InputParameters  
 slice\_image\_flip (neu- attribute), 92  
 ron\_morphology.pipeline.\_schemas.InputParametersstep\_from\_node() (in module neu-



`ron_morphology.layered_point_depths.__main__().to_json()` (`neuron_morphology.snap_polygons.geometries.Geometries`  
 53 `method`), 73  
`step_size` (`neuron_morphology.layered_point_depths._schemas.InputParameters` `neuron_morphology.validation.report.Report`  
`attribute`), 58 `method`), 102  
`surface_distance_threshold` (`neuron_morphology.snap_polygons._schemas.InputParameters` `method`), 95  
`attribute`), 66 `total_length()` (`in module neuron_morphology.features.size`), 48  
`surfaces` (`neuron_morphology.snap_polygons._schemas.OutputParameters` `neuron_morphology.features.size`), 49  
`attribute`), 67 `total_surface_area()` (`in module neuron_morphology.features.size`), 49  
`swap_nodes_edges()` (`neuron_morphology.morphology.Morphology` `total_volume()` (`in module neuron_morphology.features.size`), 49  
`method`), 110  
`SWC_COLUMNS` (`in module neuron_morphology.swc_io`), `transform()` (`neuron_morphology.snap_polygons.bounding_box.Bounding`  
 112 `method`), 68  
`swc_file` (`neuron_morphology.pipeline._schemas.InputParameters` `transform()` (`neuron_morphology.snap_polygons.geometries.Geometries`  
`attribute`), 60 `method`), 73  
`swc_path` (`neuron_morphology.feature_extractor._schemas.Reconstruction` (`neuron_morphology.transforms.affine_transform.AffineTransform`  
`attribute`), 10 `method`), 95  
`swc_path` (`neuron_morphology.layered_point_depths._schemas.InputParameters` (`neuron_morphology.transforms.transform_base.TransformBase`  
`attribute`), 58 `method`), 100  
`swc_path` (`neuron_morphology.transforms.scale_correction._schemas.InputParameters` (`neuron_morphology.transforms.affine_transform.AffineTransform`  
`attribute`), 85 `method`), 95  
`swc_path` (`neuron_morphology.transforms.tilt_correction._schemas.InputParameters` `transform_morphology()` (`neuron_morphology.transforms.affine_transform.AffineTransform`  
`attribute`), 88 `method`), 99  
`swc_path` (`neuron_morphology.transforms.upright_angle._schemas.InputParameters` `transform_morphology()` (`neuron_morphology.transforms.affine_transform.AffineTransform`  
`attribute`), 92 `method`), 99  
`swc_validators` (`in module neuron_morphology.validation`), 105 `TransformBase` (`class in neuron_morphology.transforms.transform_base`),  
 99  
**T** `transformed_swc` (`neuron_morphology.transforms.affine_transformer._schemas.OutputParameters` `attribute`), 82  
`thickness` (`neuron_morphology.features.layer.reference_layer_depths.reference_layer_depths` `attribute`), 37  
`tilt_correction` (`neuron_morphology.transforms.tilt_correction._schemas.OutputParameters` `TransformType` (`in module neuron_morphology.snap_polygons.types`), 79  
`attribute`), 88 `translate_field()` (`in module neuron_morphology.layered_point_depths.__main__`),  
`tilt_transform_dict` (`neuron_morphology.transforms.tilt_correction._schemas.OutputParameters` `58`  
`attribute`), 88 `translation` (`neuron_morphology.transforms.pia_wm_streamlines._schemas.OutputParameters` `attribute`), 83  
`TIP` (`neuron_morphology.features.statistics.coordinates.COORD_TYPES` `attribute`), 38  
`TipFeatures` (`class in neuron_morphology.feature_extractor.mark`), `trim_coords()` (`in module neuron_morphology.snap_polygons.cortex_surfaces`),  
 22 `70`  
`TipSpec` (`class in neuron_morphology.features.statistics.coordinates`), `trim_to_close()` (`in module neuron_morphology.snap_polygons.cortex_surfaces`),  
 39 `69`  
`to_csv()` (`neuron_morphology.features.layer.layered_point_depths.layered_point_depths.__main__`),  
`method`), 36 `53`  
`to_dict()` (`neuron_morphology.transforms.affine_transformer._schemas.AffineTransform` `attribute`), 81  
`method`), 95  
`to_dict_human_readable()` (`neuron_morphology.features.layer.layer_histogram.EarthMoverDistanceResult` `tv_r_01` (`neuron_morphology.transforms.affine_transformer._schemas.AffineTransform` `attribute`), 81  
`method`), 32 `tv_r_02` (`neuron_morphology.transforms.affine_transformer._schemas.AffineTransform` `attribute`), 81

tvr\_03 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.resample\_validation),  
 attribute), 81  
 tvr\_04 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.resample\_validation),  
 attribute), 81 validate() (in module neuron\_morphology.validation.resample\_validation),  
 tvr\_05 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.resample\_validation),  
 attribute), 81 104  
 tvr\_06 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.type\_validation),  
 attribute), 81  
 tvr\_07 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.type\_validation),  
 attribute), 81 validate() (in module neuron\_morphology.validation.type\_validation),  
 tvr\_08 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.type\_validation),  
 attribute), 81 validate() (neuron\_morphology.feature\_extractor.mark.Mark  
 tvr\_09 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.type\_validation),  
 attribute), 81 validate() (neuron\_morphology.feature\_extractor.mark.RequiresApical  
 tvr\_10 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.type\_validation),  
 attribute), 81 validate() (neuron\_morphology.feature\_extractor.mark.RequiresAxon  
 tvr\_11 (neuron\_morphology.transforms.affine\_transformer.\_schemas.AffineDictSchema module neuron\_morphology.validation.type\_validation),  
 attribute), 81 validate() (neuron\_morphology.feature\_extractor.mark.RequiresBasal  
 TYPE\_30 (in module neuron\_morphology.constants), class method), 20  
 106 validate() (neuron\_morphology.feature\_extractor.mark.RequiresLayer  
 U validate() (neuron\_morphology.feature\_extractor.mark.RequiresLayer  
 unnest() (in module neuron\_morphology.feature\_extractor.utilities), class method), 23  
 28 validate() (neuron\_morphology.feature\_extractor.mark.RequiresRadii  
 up() (neuron\_morphology.morphology\_builder.MorphologyBuilder class method), 22  
 method), 111 validate() (neuron\_morphology.feature\_extractor.mark.RequiresRefer  
 update() (neuron\_morphology.snap\_polygons.bounding\_box.BoundingBox class method), 23  
 method), 67 validate() (neuron\_morphology.feature\_extractor.mark.RequiresRelati  
 upright\_angle (neuron\_morphology.transforms.upright\_angle.\_schemas.OutputPass method), 21  
 attribute), 92 validate() (neuron\_morphology.feature\_extractor.mark.RequiresRoot  
 upright\_transform\_dict (neuron\_morphology.transforms.upright\_angle.\_schemas.OutputPass method), 22  
 attribute), 92 validate() (neuron\_morphology.feature\_extractor.mark.RequiresSoma  
 user (neuron\_morphology.snap\_polygons.\_from\_lims.PostgresInputConfigSchema module neuron\_morphology.morphology.Morphology  
 attribute), 64 validate() (in module neuron\_morphology.morphology.Morphology  
 V validate() (in module neuron\_morphology.morphology.Morphology  
 valid\_dendrite\_parent() (in module neuron\_morphology.validation.type\_validation), method), 108  
 105 validate() (in module neuron\_morphology.validation.type\_validation),  
 valid\_types (in module neuron\_morphology.validation.type\_validation), validate\_constrictions() (in module neuron\_morphology.validation.radius\_validation),  
 104 102  
 validate() (in module neuron\_morphology.validation.type\_validation), validate\_coordinates\_corresponding\_to\_axon\_tip()  
 100 (in module neuron\_morphology.validation.type\_validation),  
 validate() (in module neuron\_morphology.validation.type\_validation), validate\_coordinates\_corresponding\_to\_axon\_tip()  
 101 (in module neuron\_morphology.validation.type\_validation),  
 validate() (in module neuron\_morphology.validation.type\_validation), validate\_coordinates\_corresponding\_to\_dendrite\_tip()  
 101 (in module neuron\_morphology.validation.type\_validation),  
 validate() (in module neuron\_morphology.validation.type\_validation), validate\_count\_node\_parent()  
 102 (in module neuron\_morphology.validation.type\_validation),  
 104

[validate\\_distance\\_between\\_connected\\_nodes\(\)](#) (in module *neuron\_morphology.validation.radius\_validation*), 102  
[validate\\_expected\\_name\(\)](#) (in module *neuron\_morphology.validation.marker\_validation*), 101  
[validate\\_expected\\_types\(\)](#) (in module *neuron\_morphology.validation.type\_validation*), 104  
[validate\\_extreme\\_taper\(\)](#) (in module *neuron\_morphology.validation.radius\_validation*), 102  
[validate\\_immediate\\_children\\_of\\_soma\\_cannot\\_branch\(\)](#) (in module *neuron\_morphology.validation.type\_validation*), 105  
[validate\\_independent\\_axon\\_has\\_more\\_than\\_four\\_nodes\(\)](#) (in module *neuron\_morphology.validation.bits\_validation*), 100  
[validate\\_input\\_affine\(\)](#) (in module *neuron\_morphology.transforms.affine\_transformer.\_schemas*), 80  
[validate\\_input\\_affine\(\)](#) (in module *neuron\_morphology.transforms.tilt\_correction.\_schemas*), 88  
[validate\\_marker\(\)](#) (in module *neuron\_morphology.validation*), 105  
[validate\\_morphology\(\)](#) (in module *neuron\_morphology.validation*), 105  
[validate\\_multiple\\_axon\\_initiation\\_points\(\)](#) (in module *neuron\_morphology.validation.type\_validation*), 105  
[validate\\_neighbors\(\)](#) (in module *neuron\_morphology.transforms.upright\_angle.\_schemas*), 91  
[validate\\_no\\_reconstruction\\_count\(\)](#) (in module *neuron\_morphology.validation.marker\_validation*), 101  
[validate\\_node\\_parent\(\)](#) (in module *neuron\_morphology.validation.type\_validation*), 105  
[validate\\_number\\_of\\_soma\\_nodes\(\)](#) (in module *neuron\_morphology.validation.type\_validation*), 104  
[validate\\_point\\_depths\\_path\(\)](#) (in module *neuron\_morphology.feature\_extractor.\_schemas*), 9  
[validate\\_radius\\_has\\_negative\\_slope\\_dendrite\(\)](#) (in module *neuron\_morphology.validation.radius\_validation*), 102  
[validate\\_radius\\_threshold\(\)](#) (in module *neuron\_morphology.validation.radius\_validation*), 102  
[validate\\_schema\\_input\(\)](#) (*neuron\_morphology.transforms.affine\_transformer.\_schemas.ApplyAffine* method), 82  
[validate\\_schema\\_input\(\)](#) (*neuron\_morphology.transforms.tilt\_correction.\_schemas.InputParameters* method), 88  
[validate\\_schema\\_input\(\)](#) (*neuron\_morphology.transforms.upright\_angle.\_schemas.InputParameters* method), 92  
[validate\\_table\\_extension\(\)](#) (*neuron\_morphology.feature\_extractor.feature\_writer.FeatureWriter* method), 17  
[validate\\_type\\_thirty\\_count\(\)](#) (in module *neuron\_morphology.validation.marker\_validation*), 101  
[validate\\_types\\_three\\_four\\_traceable\\_back\\_to\\_soma\(\)](#) (in module *neuron\_morphology.validation.bits\_validation*), 100  
[validation\\_errors](#) (*neuron\_morphology.validation.result.InvalidMarkerFile* attribute), 103  
[validation\\_errors](#) (*neuron\_morphology.validation.result.InvalidMorphology* attribute), 103  
[version\\_path](#) (in module *neuron\_morphology*), 112

## W

[well\\_known\\_marks](#) (in module *neuron\_morphology.feature\_extractor.run\_feature\_extraction*), 26  
[WELL\\_KNOWN\\_REFERENCE\\_LAYER\\_DEPTHS](#) (in module *neuron\_morphology.features.layer.reference\_layer\_depths*), 37  
[White\\_Matter](#) (*neuron\_morphology.pipeline.\_schemas.PrimaryBoundaries* attribute), 59  
[width](#) (*neuron\_morphology.snap\_polygons.bounding\_box.BoundingBox* attribute), 67  
[wm\\_fixed\\_value](#) (*neuron\_morphology.transforms.pia\_wm\_streamlines.\_schemas.PiaWMStreamlines* attribute), 83  
[wm\\_path\\_str](#) (*neuron\_morphology.transforms.pia\_wm\_streamlines.\_schemas.PiaWMStreamlines* attribute), 83  
[wm\\_side](#) (*neuron\_morphology.features.layer.reference\_layer\_depths.ReferenceLayerDepths* attribute), 37

`wm_surface(neuron_morphology.layered_point_depths._schemas.Layer`  
     *attribute*), 57  
`wm_surface(neuron_morphology.snap_polygons._schemas.InputParameters`  
     *attribute*), 66  
`working_scale` (neu-  
     *ron\_morphology.snap\_polygons.\_schemas.InputParameters*  
     *attribute*), 66  
`write()` (*neuron\_morphology.feature\_extractor.feature\_writer.FeatureWriter*  
     *method*), 17  
`write_figure()` (in module neu-  
     *ron\_morphology.snap\_polygons.image\_outputter*),  
     78  
`write_images()` (neu-  
     *ron\_morphology.snap\_polygons.image\_outputter.ImageOutputter*  
     *method*), 78  
`write_swc()` (in module *neuron\_morphology.swc\_io*),  
     112  
`write_table()` (neu-  
     *ron\_morphology.feature\_extractor.feature\_writer.FeatureWriter*  
     *method*), 17

## Z

`zip_files()` (in module neu-  
     *ron\_morphology.pipeline.post\_data\_to\_s3*),  
     60