btmorph Documentation

Release 0.9a

Ben Torben-Nielsen

June 30, 2016

Contents

CHAPTER 1

btmorph

1.1 Introduction

Small Python library containing a data structure and tools to represent and analyze neuronal morphologies stored in the *de facto* standard SWC format [1] . See *Design requirements* below.

1.2 Installation

1.2.1 Prerequisites

You will need Python (2.7), Numpy / Scipy and Matplotlib to run the code. IPython is highly recommended because it provides an interactive environment (like Matlab) for Python.

- Linux: Most linux distributions contain versions of these packages in their package manager that can readily be installed.
- Windows and Mac: There are "scientific python" bundles that are shipped with all the aforementioned packages; most famously from Anaconda or Enthough. Alternatively check the Ipython installation.

1.2.2 Proper installation

Note: The following instructions are for Linux and Max OSX systems and only use command line tools. Please follow the appropriate manuals for Windows systems or tools with graphical interfaces.

Check out the git repository and adjust your `$PYTHONPATH`.

```
git clone https://bitbucket.org/btorb/btmorph.git
cd btmorph
export PYTHONPATH=$(pwd):$PYTHONPATH
```

The above commands will temporarily set your `$PYTHONPATH`. Add the appropriate path in your `.bashrc` to make add the package permanently.

Test the installation by running the tests (see *Unit testing*):

[1] Cannon et al. *An online archive of reconstructed hippocampal neurons.*, J. Neurosci. methods (pubmed http://www.ncbi.nlm.nih.gov/pubmed/9821633).

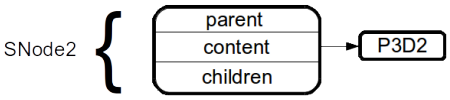
```
nosetests -v --nocapture tests/structs_test.py
nosetests -v --nocapture tests/stats_test.py
```

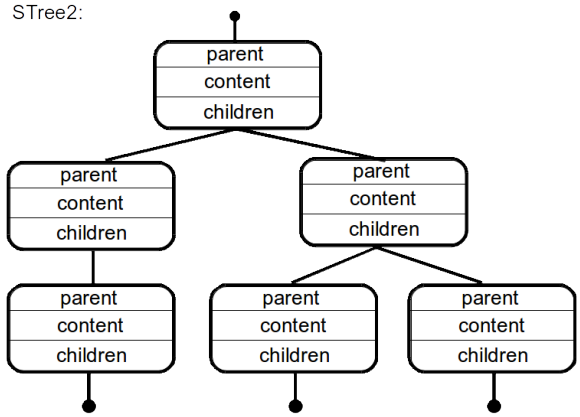
1.3 Data representation

Neurons not only look like the branching of trees, their structure is, mathematically speaking a tree structure because they can be represented as graphs without cycles. More precisely, when disregarding the soma, a neuron is a binary tree. That is a tree with at most two children at any node. As such, a **tree data structure** provides an intuitive representation of a morphology and can be easily probed to calculate morphometric features.

The tree is implemented as a linked list data structure (STree2). Each item in the list/tree is a node (SNode2) and contains pointers to its parent (`get_parent`) and its children (`get_children`). Each node can store *something* in its designated `content` container. By design, the content is a Python dict and in this library it has at least one key: `'p3d'`, a P3D2 object. Obviously, this tree data structure resembles strongly the structure of an SWC file.

Schematically, it looks like this:





1.4 Design requirements

A small set of library containing an efficient data structure and routines to quickly analyze morphometric features of neuronal morphologies.

The internal representation is based on a tree data-structure (rather than an adjacency matrix as in the TREES toolbox).

Atomic functions are provided to allow usage in scripting and enable the user to built more complex morphometrics on top of the provided functionality. The code is serial (i.e., non-parallel) because single neuron morphometrics are

fast to compute. When analyzing a batch of morphologies a parallel wrapper can be written (e.g., using Pythons's multiprocessing module or more fancy using MPI).

The input is a digital representation of a neuronal morphology in the SWC format. This is the current *de facto* format also used on the curated NeuroMorpho.org website.org database. It is expected to use the standardized SWC-format that follows the three-point soma description (see here). Analysis is based on the whole neuron but subtrees can be selectively analyzed based on the value of the SWC-type field.

Morphometrics can be either scalar (= one value per morphology) or vector / distributed (= a distribution of values per morphology). For vector morphometrics, the features can be measures either a branching point, terminal points or both. Other 'points' specified in the SWC file are only used for the internal representation of the geometry.

Simple wrappers are provided to analyze single neurons, populations thereof and compare two populations.

1.4.1 Morphometric features

- Scalar: (one per morphological structure under scrutiny)
  - total size: total length of the neurite
  - # stems
  - # branch points
  - # terminal points
  - width (without translation; absolute coordinates; potential extension along the first 3 principal components)
  - height
  - depth
  - max degree (of neurites sprouting at the soma)
  - max order (of neurites sprouting at the soma)
  - partition asymmetry (can/cannot be measured at the soma?)
- Vector: (for each point, bifurcation point or terminal point):
  - segment path length (incoming)
  - segment euclidean length (incoming)
  - contraction (euclidean / path; incoming)
  - order
  - degree
  - partition asymmetry
  - fractal dimension (of path between soma and PoI)
  - *Clouds*: save x,y,z coordinates for post-hoc histograms analysis or other scalar (e.g., moments) or vector properties (e.g., PCA)

1.4.2 Visualization

(simple, using matplotlib):

- Dendrogram
- 2D/3D plot as wires and/or with diameters

- Three 2D projections for improved visual inspection

1.5 Quick example

In the top directory of the package (`btmorph`) open `ipython --pylab` and issue the command below.

Note: In `ipython` you can use the magic function `%paste` to paste a whole code block. Copy the code below and type `%paste` at the `ipython` prompt.

```
import btmorph
import numpy
import matplotlib.pyplot as plt

swc_tree= btmorph.STree2()
swc_tree.read_SWC_tree_from_file("examples/data/v_e_moto1.CNG.swc")

stats = btmorph.BTStats(swc_tree)

# get the total length
total_length = stats.total_length()
print "total_length = %f" % total_length

# get the max degree, i.e., degree of the soma
max_degree = stats.degree_of_node(swc_tree.get_root())

# generate and save the dendrogram
btmorph.plot_dendrogram("examples/data/v_e_moto1.CNG.swc")
plt.savefig('examplar_dendrogram.pdf')
```

References

1.6 Citation

If you use this software, please cite the following peer-reviewed news item published in the Neuroinformatics journal.

B. Torben-Nielsen, An efficient and extendable Python library to analyze neuronal morphologies. Neuroinformatics, 2014, online first (<a href "http://link.springer.com/article/10.1007/s12021-014-9232-7">here)

CHAPTER 2

Tutorial

This is a brief, hands-on tutorial explaining how to use btmorph to load SWC files, analyse them by computing morphometric measures and compare morphologies to one another. A short description is also provided on how to use btmorph in further scripting.

We recommend to use IPython. In a terminal type, change to the `examples` directory and type `ipython --pylab -i`. Then, you can either type the command in the listings below or directly copy them. Copy and pasting of the code snippets can be done by copying them and typing into the ipython prompt `%paste` (Magic functions).

Note: Make use of "magic functions" in IPython. Copy the code from this pages and type `%paste` in the IPython session. All code will be pasted with correct layout and directly executed.

2.1 Analyzing morphometric data

This tutorial assumes you are in the `examples` directory of the btmorph package (ath the top level, issue `cd examples`). Several exemplar SWC files are contained in the package [1].

```
import btmorph
import numpy
import matplotlib.pyplot as plt

swc_tree = btmorph.STree2()
swc_tree.read_SWC_tree_from_file("data/v_e_moto1.CNG.swc")
stats = btmorph.BTStats(swc_tree)

""" get the total length, a scalar morphometric feature """
total_length = stats.total_length()
print 'Total neurite length=%f' % total_length

""" get the topological measure of leaf node in the tree"""
no_terminals = stats.no_terminals()
print 'Number of terminals=%f' % no_terminals
```

In case you do this tutorial in one Ipython session (`Ipython --pylab`), you don't need to load all libraries all the time and you can just continue to copy and paste code into the prompt. Therefore, loading of the library is omitted in the code listing below.

Now probe a vector morphometric, for instance the segment length. Clearly, summing all segments lengths together should give us the total segment length as before.

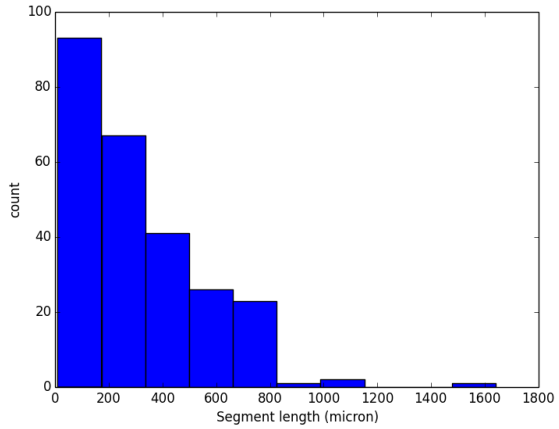
[1] v_e_moto1 is downloaded from here and originates from a study linked on pubmed.

```
bif_nodes = stats._bif_points
term_nodes = stats._end_points
all_nodes = bif_nodes + term_nodes
total_length = 0
all_segment_lengths = []
for node in all_nodes :
    all_segment_lengths.append( stats.get_segment_pathlength(node)  )
    total_length = total_length + all_segment_lengths[-1]
print 'total_length=', total_length
```

Now you can plot a histogram of the segment length distribution:

```
plt.hist(all_segment_lengths)
plt.xlabel('Segment length (micron)')
plt.ylabel('count')
```

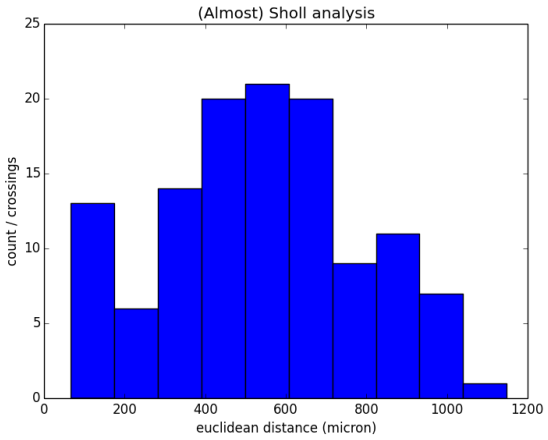
which should produce an image as illustrated below:



As a slightly more complicated example, we can also check the path length and Euclidean distance at which bifurcations occur. Plotting the number of bifurcations as a function of euclidean distance is roughly the same as the *Sholl analysis*.

```
bif_path_lengths = []
bif_euclidean_lengths = []
bif_contractions = []
for node in stats._bif_points :
    bif_path_lengths.append(stats.get_pathlength_to_root(node))
    bif_euclidean_lengths.append(stats.get_Euclidean_length_to_root(node))
    bif_contractions.append( bif_euclidean_lengths[-1] / bif_path_lengths[-1]  )
plt.hist(bif_euclidean_lengths)
plt.title('(Almost) Sholl analysis')
plt.xlabel('euclidean distance (micron)')
plt.ylabel('count / crossings')
```

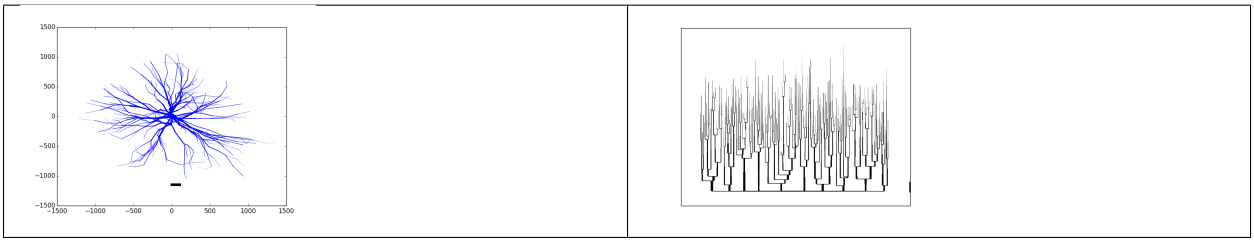
which produces the following image:



Clearly, in the above figure we can distinguish the bimodal distribution introduced by some the basal and oblique dendrites on the one hand, and the distal apical dendrites on the other.

Finally, to visually inspect both morphologies we could plot them:

```
plt.figure()
btmorph.plot_2D_SWC("data/v_e_moto1.CNG.swc")
plt.figure()
btmorph.plot_dendrogram("data/v_e_moto1.CNG.swc")
```



2.2 Potential extensions

There are also hooks in `btmorph` to access other features.

- `stats._all_nodes`: list with all nodes in the tree
- `stats._bif_points`: list with bifurcating nodes in the tree
- `stats._end_points`: list with terminal (=leaf) nodes in the tree
- `stats._tree`: STree2 structure. Can be used to compute various graph-theoretical features.

For instance, it is straight-forward to save a cloud on which measurement related to the spatial distribution of points (for instance, the moments of the point cloud) can be measured.:

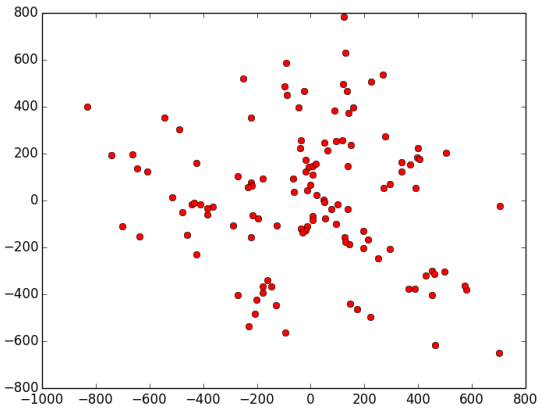
```
bx,by,bz = [],[],[]
for node in stats._bif_points :
    n = node.get_content()['p3d']
    bx.append(n.xyz[0])
    by.append(n.xyz[1])
    bz.append(n.xyz[2])
bif_cloud = [bx,by,bz]
# save as txt...
np.savetxt('bif_cloud.txt',bif_cloud)
```

```
#... or as pickle
import pickle
pickle.dump(bif_cloud,open('bif_cloud.pkl','w'))
```

Note that in this example only bifurcation points are considered. Through the `STree.get_nodes()` or `stats._all_points` all points can be retrieved.

The cloud data can now be loaded and plotted (and serve for further analysis)

```
import pickle
bc = pickle.load(open('bif_cloud.pkl'))
for i in range(len(bc[0])) :
    plt.plot(bc[0][i],bc[1][i],'ro')
```



2.3 Comparison of morphologies

Validation of morphologies boils down -in the simplest one-dimensional case and in a statistical sense- to the comparison of data vectors. The idea is visually illustrated below. The method outlined here can be easily extended to conditional data, that is, N-dimensional data capturing relations between data point using adequate statistical tools.

2.3.1 One-to-one validation

Two neurons are compared to each other. On a one to one basis there is little statistical ground to compare the scalar properties with each other. However, the vector features (for instance, segment lengths) can be compared. In this example we do the fairly senseless thing of showing the difference between a hippocampal granule cell and a spinal cord motor neuron (used before).

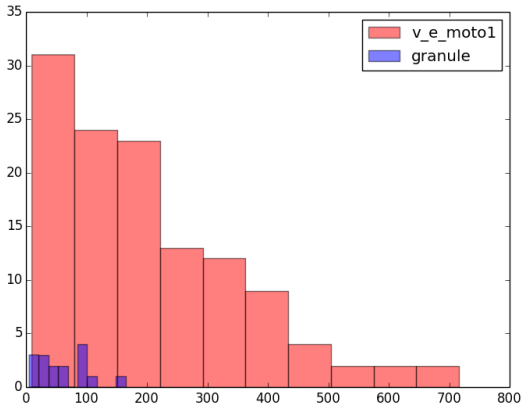
```
import btmorph
import numpy
import matplotlib.pyplot as plt

v1_tree = btmorph.STree2()
v1_tree.read_SWC_tree_from_file("data/v_e_moto1.CNG.swc")
v1_stats = btmorph.BTStats(v1_tree)

granule_tree = btmorph.STree2()
granule_tree.read_SWC_tree_from_file("data/1220882a.CNG.swc")
granule_stats = btmorph.BTStats(granule_tree)
```

```
v1_bif_nodes = v1_stats._bif_points
granule_bif_nodes = granule_stats._bif_points

v1_bif_segment_lengths = []
granule_bif_segment_lengths = []

for node in v1_bif_nodes:
    v1_bif_segment_lengths.append( v1_stats.get_segment_pathlength(node)  )
for node in granule_bif_nodes:
    granule_bif_segment_lengths.append( granule_stats.get_segment_pathlength(node)  )
```

And compare the two vectors (visually and by performing the Kruskal-Wallis H-test):

```
import scipy
import scipy.stats
hist(v1_bif_segment_lengths,color='r',alpha=0.5,label="v_e_moto1")
hist(granule_bif_segment_lengths,color='b',alpha=0.5,label="granule")
legend(loc=0)
res = scipy.stats.ks_2samp(v1_bif_segment_lengths,granule_bif_segment_lengths)
print 'K-S=%f, p_value=%f' % (res[0], res[1])
```

A figure will be generated and the output will appear: `K-S=0.609631, p_value=0.000023`



According to the manual: "if the K-S statistic is small or the p-value is high, then we cannot reject the hypothesis that the distributions of the two samples are the same."

2.3.2 Many-to-many validation

The comparison of two population can be done in exactly the same way as described above. The scalar properties of each neuron in the population make up a vector of values. Hence, the vector of one population can be compared against the vector associated with another population. In the case of vector features, all features can be appended to one vector per population.

2.4 Wrappers for btmorph

We provide basic wrappers that perform standard, of-the-shelf analysis of neurons. Two wrappers are available.

- `btmorph.perform_2D_analysis`. Collects morphometric features of birufcatiuon and terminal points and stores the results in files. For each of these points the path length to the soma, euclidean distance from the soma, degree, order, partition asymmetry and segment length are recorded. Hence, one can correlate, for instance, the segment length with the centrifugal order (= two-dimensional). Higher order correlation can be used at will as well. (See API)
- `btmorph.perform_1D_population_analysis`. Collects all morphometric features of one population in vectors and writes the result to files. (see API)

References

CHAPTER 3

Visualization

Visualization is highly customizable in btmorph because you can access the data directly and subsequently use all available Python tools to visualize the data. Visualizing a neurite boils down to looping over all nodes in the tree and plotting a line segment in between the current node and its parent. As such, end users can implement their own visualization flexibly and efficiently.

In addition to the "DIY" method btmorph also contains several wrappers to quickly generate some standard visualizations of neurites. Below we illustrate the plotting capabilities included in btmorph. Please consult the API for more details about the visualization wrappers. All wrapper use standard Matplotlib functionality.

Note: As in the previous tutorial, we assume the user is in the `examples` directory of the btmorph package. The exemplar neurons are from the Kawaguchi archive and downloaded from NeuroMorpho.org. Using the btmorph function *filter_and_save_SWC* we obtained "filtered" versions that only contain the basal and apical trees. These data files can be found in the folder *examples/data*.

3.1 2D plotting

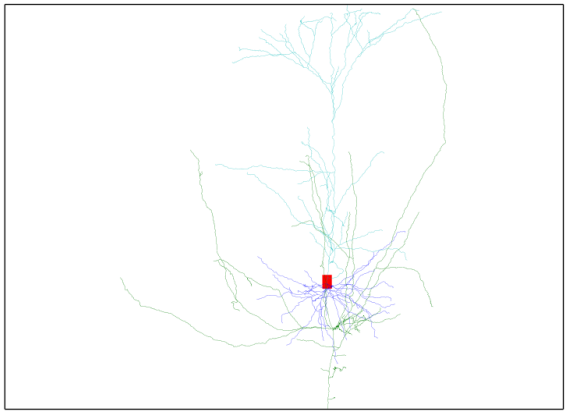
Basic 2D plotting is provided and entails a straightforward projection onto the XY plane. First, we set up some of the files for later use.

```
full_fn = "data/CTh5080306F.CNG.swc"
filtered_fn = "data/CTh5080306F.CNG_filtered.swc"
basal_fn = "data/CTh5080306F.CNG_filtered_basal.swc"
apical_fn = "data/CTh5080306F.CNG_filtered_apical.swc"
```

The color scheme used for 2D plotting can be adjusted. Currently only two schemes are in use: a default one and the one used by NeuroMorpho.org. Execute the following command:

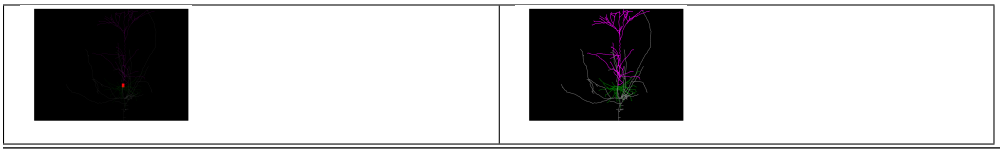
```
import btmorph
btmorph.plot_2D_SWC(full_fn,show_axis=False,color_scheme='default',depth='Y')
```

With this command axes are not drawn (*show_axis=False*), the default color scheme is used (*color_scheme='default'*) and the Y axis, which is used to set the depth (i.e., from superficial to deep), corresponds to the Y-axis used in the SWC file (*depth='Y'*). The color scheme and depth argument are redundant here because they are the defaults; calling the same function without them would produce the same figure as illustrated below.



A slight variation is shown below and uses the *color_scheme='neuromorpho'*. But because thin neurites are hardly visible we can better draw a wire plot in which all neurites have the same diameter.

```
btmorph.plot_2D_SWC(full_fn,show_axis=False,color_scheme='neuromorpho')
btmorph.plot_2D_SWC(full_fn,show_axis=False,color_scheme='neuromorpho',show_radius=False)
```

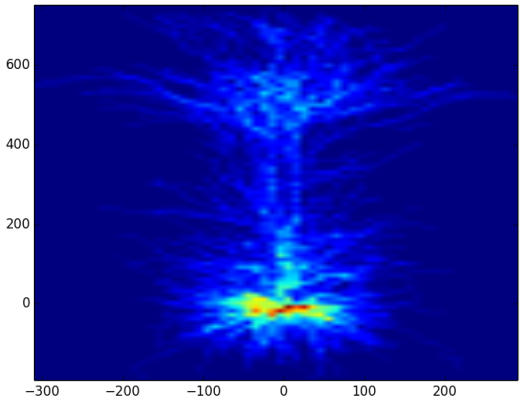


Note: Plots can be either saved by using the interactive command in Ipython: *savefig('name.extension')* (and see the matplotlib documentation for more info about the *savefig* command). Another option in btmorph is to provide the *outN* argument. The value of this argument has to be a string of the form `'name.extension'`; either the full path can be included in the name or a relative path to save the figure in the current working directory.

A nice way to visualize variation in a set of morphologies is by plotting a spatial density of the neurites. In btmorph this can be achieved by the population_2D_density_projections.

```
import btmorph
btmorph.population_density_projection(destination='data/pyr_pop/', \
   filter='*.swc', outN="data/pyr_pop/density.png", precision=[10, 10, 10],depth='Y')
```

and results in the following image:

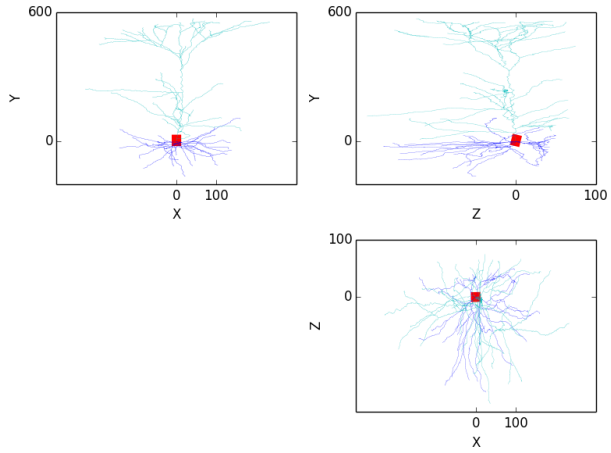


3.2 Pseudo 3D plotting

A 2D projection does not tell much about the spatial embedding of a neuron outside the projected plane. But 3D figures can become easily cluttered if many neurites are contained in one file. Therefore, we include two options for "pseudo 3D" plotting; that is, a plot containing three 2D projections that shows the neuron along three orthogonal axes.

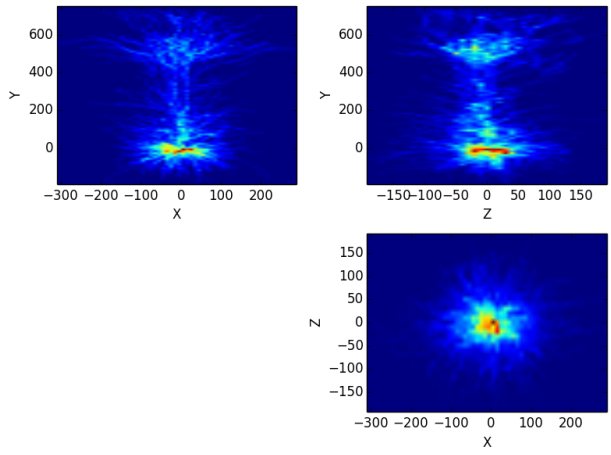
To plot a plain-vanilla pseudo 3D figure, execute the following command:

```
import btmorph
btmorph.true_2D_projections(filtered_fn,depth="Y",bar=[100,1000,100])
```



A similar pseudo 3D version is also available for the density plot.

```
import btmorph
btmorph.population_2D_density_projections(destination='data/pyr_pop/', \
    filter='*.swc', outN="data/pyr_pop/density.png", precision=[10, 10, 10])
```



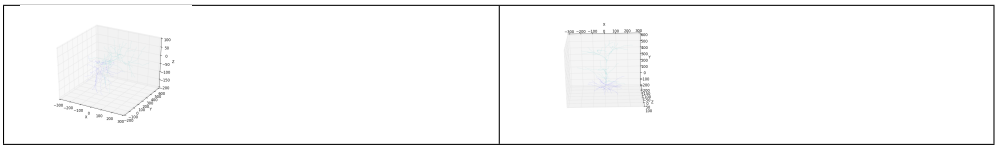
3.3 3D plotting

Note: 3D plotting is currently under construction. The final API can change in the near future.

A 3D plot can still be convenient especially when you can interactively adjust the point of view using `ipython --pylab -i`.

Invoke the following code to generate a plot as illustrated below (left). A rotated version is plotted on the right.

```
btmorph.plot_3D_SWC(filtered_fn)
```



3.4 Animation

A simple animation is provided to visually inspect a morphology in 3D. The animation rotates a morphology around the longest axis (generally the Y-axis in NeuroMorpho.org)

```
btmorph.animate_SWC_rotation(full_fn,color_scheme='default',out_n=full_fn+"_anim")
```

tjak

CHAPTER 4

Validation & testing

4.1 Comparison with L-Measure

Btmorph is compared to the "golden standard" L-Measure through the publicly available data at NeuroMorpo.org.

In most cases the results obtained with the btmorph library are similar; there are some slight differences that reflect slight implementation details and some measures are interpreted differently; implementation details of L-Measure can be found (here) and the meaning of the morphometrics displayed on NeuroMorpho.org are explained here.

We explain the similarities and differences by means of an exemplar analysis performed on one morphology: *v_e_moto1* (from here).

Morphometric feature	NeuroMorpho.org	btmorph
Soma Surface	45216 μm2	45238 μm2 [1]
# Stems	10	10
# Bifurcations	122	122
# Branches	254	254 [2]
Overall Width	1804.67 μm	2588.0 μm [3]
Overall Height	2259.98 μm	2089.0 μm [3]
Overall Depth	1701.72 μm	2306.0 μm [3]
Average Diameter	2.2 μm	2.2 μm [4]
Total Length	78849.1 μm	78849.1 μm
Total Surface	512417 μm2	512417 μm2
Total Volume	390413 μm3	390412 μm3
Max Euclidean Distance	765.73 μm	1531 μm [5]
Max Path Distance	873.56 μm	1817 μm [6]
Max Branch Order	3.69	3.83 [7]
Average Contraction	0.94	0.9359 [8]
Total Fragmentation	559	599 [9]
Partition Asymmetry	0.43	0.43 [10]
Average Rall's Ratio	1.25	1.25
Average Bifurcation Angle Local	46.83°	46.83°
Average Bifurcation Angle Remote	45.74°	45.7 °

[1] In accordance with the three-point soma format, the somatic surface is computed as $A = 4 \times \pi \times r^2$.

[2] Computed by *stats.no_bifurcations()* + *stats.no_terminals()*

[3] We compute the raw, untranslated extend in X,Y and Z dimension. This is different from aligning the axis with the first three principal components and including 95% of the data as is done in L-Measure and NeuroMorpho.org.

[4] Computed by *np.mean(stats.get_diameters())*

[5] Unclear how the NeuroMorpho.org value is generated. We compute the euclidean distance between each terminal point and the soma. A visual inspection shows that our value is correct.

[6] See [5]

[7] This is actually not the maximum as listed on the NeuroMorpho website but the average of either all points, or the bifurcation points.

```
eds = []
pls = []
for node in stats._end_points:
    eds.append(stats.get_segment_Euclidean_length(node))
    pls.append(stats.get_segment_pathlength(node))
mean(array(eds)/array(pls))

pas = []
for node in stats._bif_points:
    pas.append(stats.partition_asymmetry(node))
mean(pas)
```

4.2 Unit testing

Unit-testing refers to testing of elementary pieces of code in a computer program (Wikipedia). Testing is done using the Python testing framework, called nose tests. In these tests, we compare the outcome of our library to similar outcomes generated by L-Measure that are accessible through the NeuroMorpho.org website. Note that there are some differences in design and definition of the features as listed *Comparison with L-Measure*.

Unit-tests of this library are provided in the `tests` directory and can be run by

```
nosetests -v tests/stats_test.py
nosetests -v --nocapture tests/structs_test.py
```

Note: Run the unit-tests after change to the code to ensure a) backward compatibility and b) correctness of the results.

[8] Computed as follows:
[9] The fragmentation can be computed by `len(stats._all_nodes)-3`, where 3 is subtracted to discount the three soma points.
[10] Computed as follows:

CHAPTER 5

API Documentation

5.1 Data structures

File contains:

- P3D2
- SNode2
- STree2

B. Torben-Nielsen (from legacy code). Daniele Linaro contributed the iterators in STree2.

class btmorph.btstructs2.**P3D2**(*xyz*, *radius*, *type=7*)
Bases: object

Basic container to represent and store 3D information

Constructor.

xyz [numpy.array] 3D location

radius : float type : int

Type asscoiated with the segment according to SWC standards

class btmorph.btstructs2.**SNode2**(*index*)
Bases: object

Simple Node for use with a simple Tree (STree)

By design, the "content" should be a dictionary. (2013-03-08)

Constructor.

index [int] Index, unique name of the SNode2

add_child(*child_node*)
add a child to the children list of a given node

node : SNode2

children
Return the children nodes of this one (if any)

children [list SNode2] In case of a leaf an empty list is returned

content
Return the content dict of a SNode2

parent [SNode2] In case of the root, None is returned.Otherwise a SNode2 is returned

get_children()
Return the children nodes of this one (if any)

children [list SNode2] In case of a leaf an empty list is returned

get_content()
Return the content dict of a SNode2

parent [SNode2] In case of the root, None is returned.Otherwise a SNode2 is returned

get_index()
Return the index of this node

index : int

get_parent()
Return the parent node of this one.

parent [SNode2] In case of the root, None is returned.Otherwise a SNode2 is returned

index
Return the index of this node

index : int

make_empty()
Clear the node. Unclear why I ever implemented this. Probably to cover up some failed garbage collection

parent
Return the parent node of this one.

parent [SNode2] In case of the root, None is returned.Otherwise a SNode2 is returned

remove_child(*child*)
Remove a child node from the list of children of a specific node

node [SNode2] If the child doesn't exist, you get into problems.

set_children(*children*)
Set the children nodes of this one

children: list SNode2

set_content(*content*)
Set the content of a node. The content must be a dict

content [dict] dict with content. For use in btmorph at least a 'p3d' entry should be present

set_index(*index*)
Set the unqiue name of a node

index : int

set_parent(*parent*)
Set the parent node of a given other node

node : SNode2

class btmorph.btstructs2.**STree2**
Bases: object

Simple tree for use with a simple Node (SNode2).

While the class is designed to contain binary trees (for neuronal morphologies) the number of children is not limited. As such, this is a generic implementation of a tree structure as a linked list.

Default constructor. No arguments are passed.

add_node_with_parent(*node*, *parent*)
Add a node to the tree under a specific parent node

node [SNode2] node to be added

parent [SNode2] parent node of the newly added node

degree_of_node(*node*)
Get the degree of a given node. The degree is defined as the number of leaf nodes in the subtree rooted at this node.

node [SNode2] Node of which the degree is to be computed.

degree : int

get_node_in_subtree(*index*, *fake_root*)
Get a node with a specific name in a the subtree rooted at fake_root. The name is always an integer

index [int] Name of the node to be found

fake_root: **SNode2** Root node of the subtree in which the node with a given index is searched for

node [SNode2] Node with the specific index

get_node_with_index(*index*)
Get a node with a specific name. The name is always an integer

index [int] Name of the node to be found

node [SNode2] Node with the specific index

get_nodes()
Obtain a list of all nodes int the tree

all_nodes : list of SNode2

get_root()
Obtain the root node

root : SNode2

get_sub_tree(*fake_root*)
Obtain the subtree starting from the given node

fake_root [SNode2] Node which becomes the new root of the subtree

sub_tree [STree2] New tree with the node from the first argument as root node

is_leaf(*node*)
Check whether a node is a leaf node, i.e., a node without children

is_leaf [boolean] True is the queried node is a leaf, False otherwise

is_root(*node*)
Check whether a node is the root node

is_root [boolean] True is the queried node is the root, False otherwise

order_of_node(*node*)
Get the order of a given node. The order or centrifugal order is defined as 0 for the root and increased with

any bifurcation. Hence, a node with 2 branch points on the shortest path between that node and the root has order 2.

node [SNode2] Node of which the order is to be computed.

order : int

path_between_nodes (*from_node*, *to_node*)
Find the path between two nodes. The from_node needs to be of higher order than the to_node. In case there is no path between the nodes, the path from the from_node to the soma is given.

from_node : SNode2 to_node : SNode2

path_to_root (*node*)
Find and return the path between a node and the root.

node [SNode2] Node at which the path starts

path [list of SNode2] list of SNode2 with the provided node and the root as first and last entry, respectively.

read_SWC_tree_from_file (*file_n, types=[1, 2, 3, 4, 5, 6, 7, 8, 9]*)
Non-specific for a "tree data structure" Read and load a morphology from an SWC file and parse it into an STree2 object.

On the NeuroMorpho.org website, 5 types of somadescriptions are considered (http://neuromorpho.org/neuroMorpho/SomaFormat.html). The "3-point soma" is the standard and most files are converted to this format during a curation step. btmorph follows this default specificationand the *internal structure of btmorph implements the 3-point soma*.

However, two other options to describe the soma are still allowed and available, namely: - soma absent: btmorph adds a 3-point soma in between of [TO DEFINE/TODO] - multiple cylinder: [TO DEFINE/TODO]

file_n [str] name of the file to open

remove_node (*node*)
Remove a node from the tree

node [SNode2] node to be removed

root
Obtain the root node

root : SNode2

set_root (*node*)
Set the root node of the tree

node [SNode2] to-be-root node

write_SWC_tree_to_file (*file_n*)
Non-specific for a tree.

Used to write an SWC file from a morphology stored in this STree2. Output uses the 3-point soma standard.

file_n [str] name of the file to open

5.2 Morphometrics

class btmorph.btstats.**BTStats**(*tree*)

Bases: object

Compute morphometric features and statistics of a single morphology

Assume the "3 point" soma of the curated NeuroMorpho format. (website)

2.Torben-Nielsen (legacy code)

Constructor.

tree [STree2] Neuronal tree for which to compute morphometrics

approx_soma()

Scalar, global morphometric

By NeuroMorpho.org convention: soma surface ~ 4*pi*r^2, where r is the abs(y_value) of point 2 and 3 in the SWC file

surface [float] soma surface in micron squared

bifurcation_angle_vec(*node*, *where='local'*)

Vector, local morphometric

Only to be computed at branch points (_bif_points). Computes the angle between the two daughter branches in the plane defined by the parent and the two daughters.

cos alpha = $(\dot{ab})/(|a||b|)$

node : btmorph.btstructs2.SNode2 where : string

either "local" or "remote". "Local" uses the immediate daughter points while "remote" uses the point just before the next bifurcation or terminal point.

angle [float] Angle in degrees

bifurcation_rall_ratio_classic(*node*, *where='local'*)

Vector, local morphometric

The ratio $\frac{{d_1}^p+{d_2}^p}{D^p}$ computed with $p = 1.5$

node : btmorph.btstructs2.SNode2 where : string

either 'local or 'remote'. 'Local' uses the immediate daughter points while "remote" uses the point just before the next bifurcation or terminal point.

rr [float] Approximation of Rall's ratio

bifurcation_ralls_power_brute(*node*, *where='local'*, *min_v=0*, *max_v=5*, *steps=1000*)

Vector, local morphometric

Approximation of Rall's ratio. $D^p = {d_1}^p + {d_2}^p$, p is approximated by brute-force checking the interval [0,5] in 1000 steps (by default, but the exact search dimensions can be specified by keyworded arguments.

node : btmorph.btstructs2.SNode2 where : string

either 'local or 'remote'. 'Local' uses the immediate daughter points while "remote" uses the point just before the next bifurcation or terminal point.

rr [float] Approximation of Rall's power, p

bifurcation_ralls_power_fmin(*node*, *where='local'*)
Vector, local morphometric

Approximation of Rall's ratio using scipy.optimize.fmin. The error function is $F = {D_{d1}}^n + {D_{d2}}^n - {D_p}^n$

node : btmorph.btstructs2.SNode2 where : string

either "local" or "remote". "Local" uses the immediate daughter points while "remote" uses the point just before the next bifurcation or terminal point.

rr [float] Apprixiamation of Rall's ratio

bifurcation_sibling_ratio(*node*, *where='local'*)
Vector, local morphometric

Ratio between the diameters of two siblings.

node : btmorph.btstructs2.SNode2 where : string

Toggle 'local' or 'remote'

result [float] Ratio between the diameter of two siblings

degree_of_node(*node*)
Degree of a node. (The number of leaf node in the subtree mounted at the provided node)

node : btmorph.btstructs2.SNode2

degree [float] degree of the subtree rooted at node

frac_dim_lac(*vg=None*)
Compute both lacunarity and fractal dimension Calculates lacunarity based on standard fixed grid box counting method with coef. of variation See wikipedia for more information: http://en.wikipedia.org/wiki/Lacunarity#equation_1 Note: here we ignore orientations (all boxes start from (0,0,0)) and box sizes are always power of two Calculates fractal dimension of the given voxel grid by this formula: D = lim e -> 0 of (log(Ne)/log(e)) http://rsbweb.nih.gov/ij/plugins/fraclac/FLHelp/Glossary.htm#db

vg [btmorph.btstructs2.VoxelGrid] Ready to use voxel grid

lacunarity, fractal_dimension : tuple

fractal_dimension_box_counting_core(*vg*)
Calculates fractal dimension of the given voxel grid by this formula: D = lim e -> 0 of (log(Ne)/log(e)) http://rsbweb.nih.gov/ij/plugins/fraclac/FLHelp/Glossary.htm#db

fractal_dimension_lacunarity(*voxelSize*)
Calculate both lacunarity and fractal dimension of a tree. Faster than calling fractal_dim_box_counting and lacunarity_standard separately

voxelSize [number] Desired voxel size, affects resolution. Both measures use voxelization of the 3D tree for calculations

(lacunarity, fractal_dimension)

get_Euclidean_length_to_root(*from_node*)
euclidean length between the from_node and the root

from_node : btmorph.btstructs2.SNode2

length [float] length of the path between the soma and the provided node

get_diameters()
Vector, local morphometric

Get the diameters of all points in the morphology

get_pathlength_to_root(*from_node*)
Length of the path between from_node to the root. another branching point

from_node : btmorph.btstructs2.SNode2

length [float] length of the path between the soma and the provided node

get_points_of_interest()
Get lists containting the "points of interest", i.e., soma points, bifurcation points and end/terminal points.

soma_points : list bif_points : list end_points : list

get_segment_Euclidean_length(*to_node*)
Euclidean length to the incoming segment. Between this node and the soma or another branching point

from_node : btmorph.btstructs2.SNode2

length [float] Euclidean distance *to* provided node (from soma or first branch point with lower order)

get_segment_pathlength(*to_node*)
Vector, local morphometric.

Length of the incoming segment. Between this node and the soma or another branching point. A path is defined as a stretch between the soma and a bifurcation point, between bifurcation points, or in between of a bifurcation point and a terminal point

to_node [btmorph.btstructs2.SNode2] Node *to* which the measurement is taken

length [float] length of the incoming path in micron

global_horton_strahler()
Calculate Horton-Strahler number at the root See local_horton_strahler()

Horton-Strahler number at the root

lacunarity_box_counting_core(*vg*)
Calculate lacunarity based on standard fixed grid box counting method with coef. of variation See wikipedia for more information: http://en.wikipedia.org/wiki/Lacunarity#equation_1 Note: here we ignore orientations (all boxes start from (0,0,0)) and box sizes are always power of two

vg [btmorph.btstructs2.VoxelGrid] Ready to use voxel grid

lacunarity : float

local_horton_strahler(*node*)
We assign Horton-Strahler number to all nodes of a tree, in bottom-up order, as follows:

If the node is a leaf (has no children), its Strahler number is one. If the node has one child with Strahler number i, and all other children have Strahler numbers less than i, then the Strahler number of the node is i again. If the node has two or more children with Strahler number i, and no children with greater number, then the Strahler number of the node is i + 1. *If the node has only one child, the Strahler number of the node equals to the Strahler number of the child The Strahler number of a tree is the number of its root node.

See wikipedia for more information: http://en.wikipedia.org/wiki/Strahler_number

node [btmorph.btstructs2.SNode2] Node of interest

hs [int] The Horton-Strahler number (Strahler number) of the node

no_bifurcations()
Scalar, global morphometric

Count the number of bifurcations points in a complete moprhology

no_bifurcations [int] number of bifurcation

no_stems()
Scalar, global morphometric

Count the number of stems in a complete moprhology (except the three point soma from the Neuromoprho.org standard)

no_stems [int] number of stems

no_terminals()
Scalar, global morphometric

Count the number of temrinal points in a complete moprhology

no_terminals [int] number of terminals

order_of_node(*node*)
Order of a node. (Going centrifugally away from the soma, the order increases with 1 each time a bifurcation point is passed)

node : btmorph.btstructs2.SNode2

order [float] order of the subtree rooted at node

partition_asymmetry(*node*)
Vector, local morphometric

Compute the partition asymmetry for a given node.

node : btmorph.btstructs2.SNode2

partition_asymmetry [float] partition asymmetry of the subtree rooted at node (according to vanpelt and schierwagen 199x)

pca(*A*)

performs principal components analysis (PCA) on the n-by-p data matrix A Rows of A correspond to observations, columns to variables.

Returns [] coeff :

is a p-by-p matrix, each column containing coefficients for one principal component.

score :

the principal component scores; that is, the representation of A in the principal component space. Rows of SCORE correspond to observations, columns to components.

latent :

a vector containing the eigenvalues of the covariance matrix of A. source: http://glowingpython.blogspot.jp/2011/07/principal-component-analysis-with-numpy.html

total_dimension()
Scalar, global morphometric Overall dimension of the morphology

dx [float] x-dimension

dy [float] y-dimension

dz [float] z-dimension

total_dimensions_verbose()
Scalar, global morphometric

Overall dimension of the whole moprhology. (No translation of the moprhology according to arbitrary axes.)

dx [float] x-dimension

dy [float] y-dimension

dz [float] z-dimension

data [list] minX,maxX,minY,maxY,minZ,maxZ

total_length()
Scalar, global morphometric

Calculate the total length of a complete morphology

total_length [float] total length in micron

total_surface()
Scalar, global morphometric

Total neurite surface (at least, surface of all neurites excluding the soma. In accordance to the NeuroMorpho / L-Measure standard)

total_surface [float] total surface in micron squared

total_volume()
Scalar, global morphometric

Total neurite volume (at least, surface of all neurites excluding the soma. In accordance to the NeuroMorpho / L-Measure standard)

total_volume [float] total volume in micron cubed

5.3 Visualization

Basic visualization of neurite morphologies using matplotlib.

Usage is restricted to morphologies in the sWC format with the three-point soma standard

2. Torben-Nielsen

btmorph.btviz.**pca_project_tree**(*tree*)
Returns a tree which is a projection of the original tree on the plane of most variance

tree : btmorph.btstructs2.STree2 A tree

tree [btmorph.btstructs2.STree2] New flattened tree

btmorph.btviz.**plot_2D_SWC**(*file_name='P20-DEV139.CNG.swc', cs=None, synapses=None, syn_cs='ro', outN=None, align=True, offset=None, show_axis=False, depth='Y', filter=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], show_radius=True, bar_L=None, bar=[50, 50, 50], new_fig=True, color_scheme='default'*)
2D matplotlib plot of a neuronal moprhology. Projection can be in XY and XZ. The SWC has to be formatted with a "three point soma". Colors can be provided

file_name [string] File name of the SWC file to plots

cs [list of float] Raw values that will be plotted on the morphology according to a colormap

synapses [list of int] Indices of the compartments where synapses (= small circles) should be drawn

syn_c [string] Color of the synapses ('r','g', 'b', ...). String follows matplotlib conventions. You can include the marker style as well. Default *syn_c='ro'*

outN [string] File name of the output file. Extension of this file sets the file type

align [boolean] Translate the figure so that the soma is on the origin [0,0,0].

offset [list on float] List of length 3 with X,Y and Z shift of the morphology to be plotted. Not to be used in conjunction with the "align" option

show_axis [boolean] whether or not to draw the axis

filter [list] List of integers indicating the SWC types to be included (1:soma, 2:axon, 3:basal,4:apical,...). By default, all SWC types are included

show_radius [boolean] Default "True" plots the diameter; "False" will render a wire plot.

bar_L [float] Add a scale bar to the plot. *Currently only works with align=True*

bar [list of real] When the axis are shown (*show_axis=True*), ticks will be plotted according to this list. List contains 3 values for X, Y and Z ticks. Default [50,50,50]

depth [string] Default "Y" means that X represents the superficial to deep axis. Otherwise, use "Z" to conform the mathematical standard of having the Z axis.

new_fig [boolean] True if matplotlib has to plot in a new figure. False, if otherwise.

color_scheme [string] Set the color scheme used for background and neurite types. Default *default*. Other option *neuromorpho*

If the soma is not located at [0,0,0], the scale bar (*bar_L*) and the ticks (*bar*) might not work as expected

`btmorph.btviz.`**`plot_3D_SWC`**(*file_name='P20-DEV139.CNG.swc', cs=None, synapses=None, syn_cs=None, outN=None, offset=None, align=True, filter=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*)

3D matplotlib plot of a neuronal morphology. The SWC has to be formatted with a "three point soma". Colors can be provided and synapse location marked

file_name [string] File name of the SWC file to plots

cs [list of float] Raw values that will be plotted on the morphology according to a colormap

synapses [list of int] Indices of the compartments where synapses (= small circles) should be drawn

syn_cs [string] Color of the synapses ('r','g', 'b', ...)

outN [string] File name of the output file. Extension of this file sets the file type

offset [list on float] List of length 3 with X,Y and Z shift of the morphology to be plotted. Not to be used in conjunction with the "align" option

show_axis [boolean] whether or not to draw the axis

filter [list] List of integers indicating the SWC types to be included (1:soma, 2:axon, 3:basal,4:apical,...). By default, all SWC types are included

`btmorph.btviz.`**`plot_dendrogram`**(*file_name, transform='plain', shift=0, c='k', radius=True, rm=20000.0, ra=200, outN=None*)

Generate a dendrogram from an SWC file. The SWC has to be formatted with a "three point soma"

file_name [string] File name of the SWC file to plots

transform [string] Either 'plain' or 'lambda'. Plain means no transform while 'lambda' performs an elecotrtonic transform

shift [float] Offset in the x-direction

c [string] Color ('r','g', 'b', ...)

radius [boolean] Plot a wire (False) dendrogram or one with the thickness of the processes (True)

rm [float] Membrane resistance. Only needed when transform = 'lambda'

rm [float] Axial resistance. Only needed when transform = 'lambda'

outN [string] File name of the output file. Extension of this file sets the file type

`btmorph.btviz.`**`true_2D_projections`**(*file_name='P20-DEV139.CNG.swc'*, *align=True*, *outN=None*, *bar=None*, *depth='Z'*)

Three 2D projections

file_name [string] File name of the SWC file to plots

depth [string] Set which axis represents "depth". In experimental work, the Z-axis is depth (as in my PPNeurMorphC) but in NeuroMorpho the Y-axis is the depth. (Depth is used to denote the axis from deep to superficial)

align [boolean] Translate the figure so that the soma is on the origin [0,0,0].

outN [string] File name of the output file. Extension of this file sets the file type

bar [list of int or real] Three values to set the thicks and marks on the plot in X,Y and Z-dimension

depth [string] Set the axis representing the depth (= axis from superficial to deep). In most SWC files this is 'Y'. The other option is 'Z', that is more consistent with the usual Cartesian coordinate systems

`btmorph.population_density_plots.`**`population_2D_density_projections`**(*destination='.'*, *filter='*.swc'*, *outN=None*, *depth='Z'*, *limits=None*, *precision=[10, 10, 10]*)

Plot a pseudo-3D heat-map of all neurites in a population by means of three 2D plots. A population can be sepcified by setting the *destination* and *filter* arguments.

destination [string] Set the directory in which the population is located. Default "." sets the current working directory.

filter [string] Filter the file-names to constrain the population. Default "*.swc" will select all SWC files in one directory. The filter uses the glob module so make sure glob understands your filter!

outN [string] File name to save the figure to. Default is None and the figure will not be saved by default.

depth [string] Indicate the axis from superficial to deep. In most files this is "Y". Default, however, is set to the mathematical standard "Z".

precision [list of ints] Set the bin size of the heatmap. Default [10,10,10] (in micron)

`btmorph.population_density_plots.`**`population_density_projection`**(*destination='.'*, *filter='*.swc'*, *outN=None*, *depth='Z'*, *precision=[10, 10, 10]*)

Plot a 2D heat-map of all neurites in a population. A population can be sepcified by setting the *destination* and

filter arguments. Parameters ———- destination : string

Set the directory in which the population is located. Default "." sets the current working directory.

filter [string] Filter the file-names to constrain the population. Default "*.swc" will select all SWC files in one directory. The filter uses the glob module so make sure glob understands your filter!

outN [string] File name to save the figure to. Default is None and the figure will not be saved by default.

depth [string] Indicate the axis from superficial to deep. In most files this is "Y". Default, however, is set to the mathematical standard "Z".

precision [list of ints] Set the bin size of the heatmap. Default [10,10,10] (in micron)

Generate animation of morphologies using matplotlib.

Usage is restricted to morphologies in the sWC format with the three-point soma standard

btmorph.btviz_dynamic.**animate_SWC_rotation**(*file_name, offset=None, align=True, color_scheme='neuromorpho', filter=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], depth='Y', out_n='rotate_animation'*)

Rotate illustration of a neuronal morphology over 360 degrees.

file_n [string] Filename of SWC description. **Must be using the 3-point soma** description. If the file is not in this format, use `btmorph.btstructs2.STree2.read_SWC_tree_from_file()` followed by `btmorph.btstructs2.STree2.write_SWC_tree_to_file()`, which will convert the description to three-point soma.

offset [array_like] Move the structure by the specified [x,y,z]

color_scheme [string] Default or "neuromorpho"

filter_range [array_like] List of integers indicating the SWC types to be included (1:soma, 2:axon, 3:basal,4:apical,...). By default, all SWC types are included

depth [string] Specify which direction (axis) represents "depth". On NeuroMorpho.org this is the Y-axis (and is used here by default). In NeuroMaC, depth is on the Z-axis.

out_n [string] File name of the generated gif file. The ".gif" extension is added automatically.

5.4 Wrappers / Tools

btmorph.tools.analyze_1D_population.**perform_1D_population_analysis**(*destination, filter='*.swc', depth='Y', bar=[200, 200, 200], post_name=None, max_n=None*)

Wrapper function to perform a complete analysis of a population of neuronal morphologies stored in SWC format (and three-point soma).

Computes the following features:

- # bifurcations
- # terminals
- # stems

- •total length
- •total volume
- •total surface
- •max centrifugal order
- •inter bifurcation length
- •terminal segment length
- •euclidean distance between terminal tips and soma
- •path legth between terminal tips and soma

For each feature a list is created with all values collected from all morphologies.

These vectors are saved as python Pickle objects.

At the same time a histogram is generated to display the data.

destination [string] string with the location of where to find the SWC files.

filter [string] filter to select SWC files. Default is "*.swc". See glob documentation for more advanced filters

depth [string] Dimension that indicates "depth"/distance from top. Default is "Y"; NeuroMac generated files use "Z".

bar [array of float] Include a scale-bar with the specified dimensions

max_n [int] To perform the analysis on a subset of morphologies specify a number. Default is None and all morphologies will be taken into account.

post_name [string] string appended to the file name when saving. Default None

btmorph.tools.analyze_2D_per_neuron.**perform_2D_analysis**(*destination*, *filter='*.swc'*, *max_n=None*)

Wrapper function to perform an analysis of the vector features of one neuronal morphology (in the SWC format and with 3-point soma)

For both the terminal points and the branching points the following features are recorded

- •Order of the node
- •degree of the node
- •Euclidean distance to the soma
- •path length to the soma
- •pathlength of the segment (coming in to a node)
- •Euclidean distance of the segment (coming in the a node)
- •branch angle amplitude [branch points only]

Two text files are generated, for terminal and branching points. Each row corresponds to a node (point) and the six columns correspond to the features above.

destination [string] string with the location of where to find the SWC files.

btmorph.tools.filter_and_save_swc.**filter_and_save_SWC**(*destination, filter, types=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], prefix='_filtered'*)

Removes points from a SWC structure and saves the new SWC to a file.

Can be used to remove unwanted structures that are identifiable by the type-field in the SWC description. Specification of (standard) SWC type fields can be found here.

To select the basal dendrites only, use the argument *types=[1,3]*: 1 to select the soma and 3 for the basal dendrites themselves.

destination [string] string with the location of where to find the SWC files.

types [list of int] types that are to be remained in the SWC file.

CHAPTER 6

Indices and tables

- *genindex*
- *search*

.