***flycode* Documentation**

**by Dustin Garner**

This document details all of the functions contained within the [*flycode* library](https://github.com/sungsoo-ucsb/GarnerKindLai_MeTu) ([*EM-Analyses/Python/flycode*](https://github.com/sungsoo-ucsb/GarnerKindLai_MeTu/tree/main/EM-Analyses/Python/flycode)), written by Dustin Garner. It contains much of the functionality required to make figures found in the [*Garner, 2024*](https://www.nature.com/articles/s41586-024-07967-z) paper published by the Kim Lab and Wernet Lab, specifically most of the extended data and supplemental information.

In order to regenerate any of those figures, there is a Jupyter Notebook *EM-Analyses/Python/figure\_generation.ipynb* that has detailed and organized information on how to generate each figure. Simply run the first cell that contains relevant imports and variable declarations, and then run any other cell that contains the figures you wish. Generated figures will be stored within a folder, *flycode/Generated-Figures* (this folder is created if it does not already exist).

Many of these functions read information from files in the folder *flycode/Readable*. Most of those files I included, but some were too large for uploading to Github. “Codex Labels 783.csv” can be downloaded [here](https://codex.flywire.ai/api/download) (Community Labels (Raw)), renamed, and placed in the *Readable* folder to run *proofreading.all\_codex\_names()*. “Full783.feather” can be downloaded [here](https://zenodo.org/records/10676866) for *path\_weights.get\_full\_df()* (watch out, the file is ~10GB). “NeuprintToken.txt” is a text file that contains a Neuprint Token, which can be generated [here](https://connectome-neuprint.github.io/neuprint-python/docs/quickstart.html#client-and-authorization-token), which is used for *neuprint\_reading.lobula\_counts()* and *specific.full\_comparison()*.

The library mainly utilizes the flywire module of the [*fafbseg* library](https://fafbseg-py.readthedocs.io/en/latest/source/intro.html). The default dataset is set to “production” in *\_\_init\_\_.py* when importing the *flycode* library. All analyses in the paper were used during the standard FlyWire materialization version 783. In order to standardize, standard queries should be performed through the *flywire\_functions* module instead of the *fafbseg* library. The most useful functions are as follows: *flywire\_functions.locs\_to\_segments()* takes in an array of FlyWire [x, y, z] coordinates and converts them to an array of Version 783 neuron IDs in which those coordinates are found. *flywire\_functions.fetch\_synapses()* takes in a neuron ID or an array of neuron IDs and returns a dataframe of connectivity information. It simply runs [*fafbseg.flywire.get\_synapses()*](https://fafbseg-py.readthedocs.io/en/latest/source/generated/fafbseg.flywire.synapses.get_synapses.html) with Version 783 neurons.

The most important thing to know about this library is how neuron types are stored. There is a spreadsheet in *flycode/Readable* called “Neuron Spreadsheet.xlsx”, which has neuron types and all of the coordinates of those neurons. This information is stored by having a neuron type in column A and a coordinate in column B. Rows with empty column B are not parsed. Then *readfiles.import\_coords()* goes through the spreadsheet and returns a dictionary of each neuron type with a list of all the relevant coordinates. The variable *mapping.neur\_coords* stores this information. Then whenever you want the IDs of these neurons, you can for example run *mapping.ids\_from\_types([“MeTu1\_R”, “MeTu2\_R”])* to retrieve the relevant neuron IDs of those types. Converting coordinates to IDs is costly, so every time this conversion occurs (via *mapping.add\_types()*, and ultimately via *fafbseg.flywire.locs\_to\_segments()*), neuron IDs are cached in the variable *mapping.neur\_ids* and can be accessed far faster later.

From there, connectivity charts can be made of these neuron types. The class *mapping.ConnectionMap* contains information comparing presynaptic and postsynaptic types and generates connectivity plots. In order to initialize it, pass in a list of presynaptic types, postsynaptic types, and the region in which you want the connectivity. For example, a connectivity plot between MeTu2 subtypes and their relevant TuBu types in the right AOTU can be made by running *metu\_to\_tubu = mapping.ConnectionMap([“MeTu2a\_R”, “MeTu2b\_R”], [“TuBu01\_R”, “TuBu06\_R”], “AOTU\_R”)* and then *metu\_to\_tubu.make\_connectivity\_plots(plot\_name=“MeTu2\_R to TuBu\_PC\_R in AOTU”)*.

From here on I will detail every module, including its variables, classes, and functions.

# 

# annotations.py

This module has a couple functions that help determine the extent to which certain annotators participated. It was mainly for testing whether FlyWire annotations lined up with the lab’s names. Both functions are now deprecated because the annotations are out of date with materialization version 783. No functions from this module were used for the paper.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

client caveclient.frameworkclient.CAVEclientFull

The CaveClient, used to retrieve the annotation table.

version numpy.int64

The version of the client.

user\_ids dict

Keys are Flywire user IDs, and values are a list containing the Flywire user’s name and

lab.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

**annotations.test\_types(neur\_types)**

Prints whether neuron types from Neuron Spreadsheet match up with FlyWire annotated neuron types.

Parameters

neur\_types list-like

Neuron types in Neuron Spreadsheet.

Returns

None

**annotations.compare\_annotations(neur\_types)**

Makes and exports a spreadsheet to compare neuron type annotations. This function is now deprecated, and proofreading.add\_codex\_names(neur\_types) replaced it.

Parameters

neur\_types list-like

Neuron types in Neuron Spreadsheet.

Returns

label\_df pandas.DataFrame

A DataFrame containing the all annotations of the neurons with the given types.

# figures.py

This module is a pipeline for figure generation such that all figures for the paper can maintain a consistent style. It also includes details for exporting figures.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

border number = 0

The borderaxespad size used in MatPlotLib legend generation.

text\_pad number = 0

The handletextpad size used in MatPlotLib legend generation.

label\_pad number = 2

The labelpad size used for MatPlotLib bar plot axis labels.

font\_size number = 6

The fontsize used in MatPlotLib figure generation.

xtick\_rotation number = 45

The xtick rotation angle (degrees) used in MatPlotLib figure generation.

ha str = ‘right’

The ha of xticks used in MatPlotLib figure generation.

edge\_color str = ‘white’

The edgecolor used in MatPlotLib legend generation.

rotation\_mode str = ‘anchor’

The xtick rotation mode used in MatPlotLib figure generation.

loc str = ‘lower left’

The loc used in MatPlotLib legend generation.

bbox tuple = (1.04, 0)

The bbox\_to\_anchor used in MatPlotLib legend generation.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Enums

**figures.FileType**

Contains values for different file types to export. Includes PDF, PNG, and SVG.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### figures.verify\_folder\_path(folder\_path)

Makes a directory path with the given list of directory names in the Generated-Figures directory if the given path does not already exist. Returns the (now valid) directory path.

Parameters

folder\_path list-like

A list of subdirectory names to nest within the Generated-Figures directory.

Returns

directory str

The verified folder path.

### figures.get\_folder\_path(figure\_name, folder\_path=[], file\_type=FileType.PDF)

Makes an exportable figure path within the given directory path with the given file type.

Parameters

figure\_name str

The desired name of the figure.

folder\_path list-like

A list of subdirectory names where the figure should be placed. The default is an empty list, which would put the figure directly into the Generated-Figures folder.

file\_type <enum ‘FileType’>

The desired file type of the figure, based on the options within the FileType enum.

Returns

file\_path str

The verified figure path where the figure can be generated.

### figures.save\_fig(fig, plot\_name, folder\_path=[], file\_type=FileType.PDF, dpi=300, transparent=False)

Saves a MatPlotLib figure at the given location with the given specifics.

Parameters

fig matplotlib.pyplot.figure

The figure to save.

plot\_name str

The desired name of the figure.

folder\_path list-like

A list of subdirectory names where the figure should be placed. The default is an empty list, which would put the figure directly into the Generated-Figures folder.

file\_type <enum ‘FileType’>

The desired file type of the figure, based on the options within the FileType enum.

dpi int

The desired dots per inch of the figure.

transparent bool

Whether the figure should have a transparent background.

Returns

None

### figures.add\_legend(legend=None, marker\_scale=1.2)

Makes a legend for a figure based on specific variables for formatting consistency.

Parameters

legend matplotlib.pyplot.legend

The legend to be converted if one has already been created for a figure. Otherwise, if None, creates a blank legend for the figure.

Returns

leg matplotlib.pyplot.legend

The legend with consistent formatting.

### figures.create\_strip\_plot(data, x\_label, y\_label, order, hue, palette, plot\_name, folder\_path=[], dodge=True, save\_figure=True, fig\_size=(2.0, 1.25), show\_means=False, mean\_type=’.’, size=2.5, lim\_range=None)

Makes a strip plot with the given parameters.

Parameters

data pandas.DataFrame

The data from which to make the strip plot.

x\_label str

The name of the data on the x-axis.

y\_label str

The name of the data on the y-axis.

order list-like

The order of the data columns.

hue str

What column to base the different colors on.

palette list-like

The list of colors.

plot\_name str

The name of the plot to be exported.

folder\_path list-like

The folder path the figure will be saved to. The default is [].

dodge bool

Whether the hue strips are separated. The default is True.

save\_figure bool

Whether the figure will be saved. The default is True.

fig\_size tuple

The size of the figure. The default is (2.0, 1.25).

show\_means bool

Whether to show the means. The default is False.

mean\_type str

The type of mean line to have. The default is ".".

size float

Radius of the markers. The default is 2.5.

lim\_range list-like

The range by which to limit the plot values. The default is None.

Returns

None

### figures.strip\_plot(\*args, \*\*kwargs)

An alias for create\_strip\_plot(). It was the previous name for the function until the name was changed for consistency.

### figures.create\_bar\_graph(data, x\_label, y\_label, x\_ticks, y\_ticks, colors, fig\_size, plot\_name, save\_figure, foler\_path=[], color\_axis=0)

Makes a bar graph with the given parameters.

Parameters

data dict

Keys are sets of sets of bars, one for each within a bar cluster. Values are 2D np arrays: one dimension for bars stacked on top of one another, the second for values of each bar.

x\_label str

The name of the data on the x-axis.

y\_label str

The name of the data on the y-axis.

x\_ticks str

The names of clusters of bars.

y\_ticks str

The spacing of number values on the y-axis.

colors list-like

Values are the colors that each bar corresponds to.

fig\_size tuple

The size of the figure.

plot\_name str

The name of the plot to be exported.

save\_figure bool

Whether the figure will be saved. The default is True.

folder\_path list-like

The folder path the figure will be saved to. The default is [].  
color\_axis int

Along which axis the color scheme occurs. The default is 0.

Returns

None

# flywire\_functions.py

This module is a mediator between the codebase and the fafbseg module to ensure that all functions are run consistently with the same parameters when generating figures and running analyses. That specifically refers to having the materialization version 783 so all neurons are within the same snapshot.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

client caveclient.frameworkclient.CAVEclientFull

The CaveClient, used to retrieve the annotation table.

version numpy.int64

The version of the client.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### flywire\_functions.locs\_to\_segments(coords)

Retrieves a list of neuron IDs based on coordinates inside the neurons. Performs the same function as fafbseg.flywire.locs\_to\_segments() with the timestamp set to “mat\_783.”

Parameters

coords numpy.ndarray

An array of shape (X, 3), where X is any number of coordinates in [x, y, z] form. The coordinates are in FlyWire units, not nanometers.

Returns

segments numpy.ndarray

An array of neuron IDs in the same order as the coordinates.

### flywire\_functions.fetch\_synapses(\*args, \*\*kwargs)

Performs fafbseg.flywire.get\_synapses() with version “mat\_783.” (Note: the fafbseg function used to be fafbseg.flywire.fetch\_synapses() but has been deprecated, hence the naming). Optionally pass in a “region” keyword argument to limit the search to a given region.

Parameters

region str

The region by which to limit the search. This should be a region within the Regions.xlsx spreadsheet or by extension reduction.region\_dict. If none is given, the DataFrame is not limited by region.

\*args

Any arguments for fafbseg.flywire.get\_synapses().

\*\*kwargs

Any keyword arguments for fafbseg.flywire.get\_synapses().

Returns

syn\_df pandas.DataFrame

A DataFrame containing the synapses of the given neuron IDs.

### flywire\_functions.fetch\_cave\_synapses(neur\_id)

Retrieves synapse information of a given neuron directly from CaveClient. This is the same method fafbseg module uses, so it is more direct. This method was used to test the accuracy of fafbseg.flywire.get\_synapses(), but was not used in the paper.

Parameters

neur\_id np.int64

The neuron of which synapse information is desired.

Returns

df pandas.DataFrame

A DataFrame containing the synapses of the given neuron ID.

### flywire\_functions.fetch\_boxed\_synapses(region, further\_limit=True)

Retrieves all synapses within a certain region bounding box. It is a more comprehensive search of all synapses, and was not used for the paper.

Parameters

region str

The region by which to limit the search. This should be a region within the “Boxes” sheet of the Regions.xlsx spreadsheet.

further\_limit

If true, also limits the synapses to be within the bounds of the same region in the “Include”/“Exclude” sheets of the Regions.xlsx spreadsheet.

Returns

df pandas.DataFrame

A DataFrame containing the synapses within the given region.

### flywire\_functions.get\_region\_neurons(region, min\_syns=1)

Retrieves all of the neurons within a given region.

Parameters

region str

The region by which to limit the search. This should be a region within the “Boxes” sheet of the Regions.xlsx spreadsheet.

min\_syns int

The minimum number of synapses that a neuron needs to have within the region in order to be included in the final list.

Returns

limited\_neurs np.array

An array containing all the neurons in the given region.

# maintenance.py

This module contains a couple sanity checks. These were performed when checking data quality for the paper.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### maintenance.find\_undone\_input(min\_syns=4)

Uses the MeTu input table to determine which neurons have not been classified based on the minimum number of synapses. This was useful when identifying all neurons upstream of proofread neurons down to 4 synapses.

Parameters

min\_syns int

The minimum number of synapses required between an upstream neuron and a MeTu neuron to determine whether the upstream neuron has been identified or not.

Returns

pre\_ids np.array

The upstream neurons that have not been identified.

### maintenance.check\_joined\_concurrence(types)

Out of all types, prints which neurons did not concur between the joined\_avp\_table.xlsx and “Neuron Spreadsheet.xlsx” spreadsheets. This was useful in the early stages of the paper when the former was used more, but afterwards all analyses were based on Neuron Spreadsheet.

Parameters

types list-like

A list of neuron types to compare within the two spreadsheets.

Returns

None

### maintenance.check\_proof\_concurrence(neur\_id)

Checks if MeTu\_proofreading.xlsx neurons match with those from

“Neuron Spreadsheet.xlsx”.

Returns

None

### maintenance.check\_neur\_connections(types, dend\_region, axon\_region)

Checks whether all neurons of the given types have dendrites in dend\_region and axons in axon\_region.

Parameters

types list-like

Neuron types from Neuron Spreadsheet.

dend\_region str

The region the neurons should have dendrites in.

axon\_region str

The region the neurons should have axons in.

Returns

issues np.array

Neurons that do not have dendrites in dend\_region or axons in axon\_region.

### maintenance.check\_duplicates()

Checks whether there are any neuron duplicates among the types found in Neuron Spreadsheet.

Returns

None

# mapping.py

This module contains most of the functionality used for analyzing connectivity between neurons and making charts. It has the class used for making numerous connectivity diagrams throughout the paper.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

neur\_coords dict

Keys are all of the neuron types that are usable for analysis. Values are arrays of coordinates. Each coordinate is a single neuron of the given type. They are all derived from the Neuron Spreadsheet. While there are no neuron repeats within specific neuron types, there may be broad types that contain the same neuron types as more specific types, for example MeTu3 has the same neurons as MeTu3a, MeTu3b, and MeTu3c.

prev\_ids dict

This was a dictionary used for checking whether neuron IDs in the Neuron Spreadsheet match with ones that are being queried. It is no longer useful given that all *flywire\_functions.py* queries now use the standard Version 783.

neur\_ids dict

This is a cache of neuron IDs that have been queried for, based on the types in the variable *neur\_coords*. Whenever the function *add\_types* is performed, it converts coordinates from *neur\_coords* and stores them here, so *fw.locs\_to\_segments* only has to be performed once for each type (it is costly).

new\_colors dict

This stores custom colormaps for plots. Keys are strings, which are color names. Values are *matplotlib.colors.ListedColormap* so they can be used as ranges.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Classes

### mapping.ConnectionMap

This is a class that stores presynaptic and postsynaptic neuron types for the ability to create figures with varying parameters.

**Methods**

#### \_\_init\_\_(self, pre\_types, post\_types, region, min\_synapses=5, exclude=[], font\_size=6)

Initializes the class.

Parameters

pre\_types list-like

This is a list of neuron types that will be used as presynaptic partners in connectivity plots.

post\_types list-like

This is a list of neuron types that will be used as presynaptic partners in connectivity plots.

region str

The region to limit the connectivity plots.

min\_synapses int

For the minimum weight connectivity plot, any neurons with fewer than this number of synapses in the given region have a weight of 0. This is to prevent colorbar skewing. The default is 5.

exclude list-like

Any outlier neuron IDs in this list are excluded to not skew the plot. They are shown to have a weight of 0 in the plots.

font\_size int

The font size of the figures. The default is 6.

#### get\_major\_label(self, type\_list, starting\_point=-0.5)

Gets the major label for connection maps.

Parameters

type\_list list-like

A list of neuron types in the label.

starting\_point float

The starting location of the label. The default is -0.5.

Returns

label\_position np.array

The positions of each label.

#### get\_minor\_label(self, type\_list, starting\_point=-0.5)

Gets the minor label for connection maps.

Parameters

type\_list list-like

A list of neuron types in the label.

starting\_point float

The starting location of the label. The default is -0.5.

Returns

label\_position np.array

The positions of each label.

#### get\_type\_ label(self, type\_list, starting\_point=-0.5)

Gets the major label for type maps.

Parameters

type\_list list-like

A list of neuron types in the label.

Returns

label\_position np.array

The positions of each label.

#### get\_minor\_type\_ label(self, type\_list, starting\_point=-0.5)

Gets the major label for type maps.

Parameters

type\_list list-like

A list of neuron types in the label.

starting\_point float

The starting location of the label. The default is -0.5.

Returns

label\_position np.array

The positions of each label.

#### plot\_connectivity(self, conn\_map, plot\_name=“”, plot\_folder=“”, cmap\_color=”Purples”, fig\_size=(1.5, 1.5), save\_figure=True)

Makes an exportable connectivity matrix figure.

Parameters

conn\_map np.array

A 2D connectivity matrix. Generally *self.syn\_map*, *self.weight\_map*, or *self.weight\_map\_minimized*.

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

cmap\_color str or matplotlib.colors.ListedColormap

The desired colormap of the plot (may be a value in *mapping.new\_colors*).

fig\_size tuple

The desired size of the plot (float, float).

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

Returns

fig matplotlib.pyplot.figure

The generated connectivity figure.

#### plot\_type\_connectivity(self, type\_map, plot\_name=“”, plot\_folder=“”, cmap\_color=new\_colors[“Green”], fig\_size=(1.5, 1.5), save\_figure=True)

Makes an exportable type connectivity matrix figure.

Parameters

type\_map np.array

A 2D type connectivity matrix. Generally *self.type\_map*, or *self.type\_weight\_map*.

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

cmap\_color str or matplotlib.colors.ListedColormap

The desired colormap of the plot (may be a value in *mapping.new\_colors*).

fig\_size tuple

The desired size of the plot (float, float).

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

Returns

fig matplotlib.pyplot.figure

The generated connectivity figure.

#### make\_connectivity\_plots(self, plot\_name=“”, plot\_folder=“”, fig\_size=(1.5, 1.5), save\_figure=True)

Makes connectivity matrix figures out of *self.syn\_map*, *self.weight\_map*, and *self.weight\_map\_minimized* using *self.plot\_connectivity()*. Only saves *self.weight\_map\_minimized*, which was the plot type used in the paper.

Parameters

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

fig\_size tuple

The desired size of the plot (float, float).

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

Returns

None

#### make\_type\_plots(self, plot\_name=“”, plot\_folder=“”, fig\_size=(1.5, 1.5), save\_figure=True)

Makes type connectivity matrix figures out of *self.type\_map* and *self.type\_weight\_map*, using *self.plot\_type\_connectivity()*. Only saves *self.type\_weight\_map*, which was the plot type used in the paper.

Parameters

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

fig\_size tuple

The desired size of the plot (float, float).

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

Returns

None

#### to\_excel(self, file\_name)

Exports *self.syn\_map*, *self.weight\_map*, *self.weight\_map\_minimized*, *self.type\_map* and *self.type\_weight\_map* to Excel.

Parameters

tile\_name str

The desired name of the Excel file.

Returns

None

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### mapping.add\_types(types, output=False)

Converts any neuron types that are in *mapping.neur\_coords* to the neuron IDs and stored them in the cache *mapping.neur\_ids*.

Parameters

types list-like

The types that you want the IDs of.

output bool

If True, prints whether a type has no neurons in the spreadsheet or neurons that are not updated (the latter is obsolete due to using Version 783 for everything).

Returns

None

### mapping.add\_broad\_type(type\_name, constituents)

Puts a type in *mapping.neur\_ids* that consists of various subtypes. For instance, MeTu3\_L could comprise MeTu3a\_L, MeTu3b\_L, and MeTu3c\_L.

Parameters

type\_name str

The name that you want the broad type to be called.

constituents list-like

The subtypes that comprise the broad type.

Returns

None

### mapping.ids\_from\_types(types)

Retrieves an array of the neuron IDs of all the given types.

Parameters

types list-like

The types (from Neuron Spreadsheet) that you want the IDs of.

Returns

ids np.array

The root IDs of the neurons of the given types.

### mapping.copy\_types(types)

Similar to *mapping.ids\_from\_types()*, but rather than returning the IDs, copies them to the clipboard (so you can easily paste them in FlyWire).

Parameters

types list-like

The types (from Neuron Spreadsheet) that you want the IDs of.

Returns

None

### mapping.find\_neur\_type(neur\_id, search=[], unidentified\_type=“”)

Searches through *mapping.neur\_ids* and finds what type the given neuron is, if it has been added to the dict.

Parameters

neur\_id np.int64

The neuron ID you want the type of.

searches list-like

Which types to search for. If the list is not empty, the type will only be identified as one of the given search types. If the list is empty, the function will search through all added *mapping.neur\_ids* types and return the first time one in which it finds *neur\_id*.

unidentified\_type str

If the neuron is not found among the search types, this is the default value returned.

Returns

neur\_type str

Either the neuron’s type or *unidentified\_type* if not found.

### mapping.check\_existence\_in\_region(types, regions, min\_syns=10)

Prints whether any neurons of the given types do not have at least *min\_syns* in the given regions. This is useful for bulk checking neuron types that normally have high synapse counts in a certain region, but that may have some damaged individual neurons that have fewer synapses.

Parameters

types list-like

Neuron types from Neuron Spreadsheet.

regions list-like

Regions from the Regions spreadsheet.

min\_syns int

If a neuron has fewer synapses than this value in a region, that information is printed.

Returns

None

### mapping.find\_partners(neur\_ids, region, pre\_or\_post, min\_syns=5)

Takes in neuron IDs and returns a dictionary of all presynaptic or postsynaptic connections to other neurons, including the IDs of the connected neurons and how many connections there are.

Parameters

neur\_ids int or list-like

The neuron IDs whose partners are wanted.

regions str

The region with which to limit the search.

pre\_or\_post str

If “pre”, presynaptic partners of *neur\_ids* are returned. If “post”, postsynaptic partners are returned.

min\_syns int

If a partner has fewer than this amount of synapses to any of *neur\_ids*, it is not included in the returned dictionary.

Returns

neur\_dict dict

Keys are the IDs of the partners, sorted by decreasing number of synapses. Values are the total number of pre- or post-synapses each neuron has to neurons in *neur\_ids*.

### mapping.partner\_df(neur\_ids, region=”Connectome”, pre=True, post=True, autapses=False, min\_syns=5)

Retrieves a synapse dataframe via *flywire.get\_synapses()* that is limited by the parameters.

Parameters

neur\_ids int or list-like

The neuron IDs of which the connection DataFrame is wanted.

region str

The region with which to limit the search.

pre bool

If True, includes the IDs as presynaptic partners in the final DataFrame.

post bool

If True, includes the IDs as postsynaptic partners in the final DataFrame.

autapses bool

If True, includes autapses (default is set to False).

min\_syns int

If a partner has fewer than this amount of synapses to any of *neur\_ids*, it is not included in the returned dataframe.

Returns

syn\_df pandas.DataFrame

The resulting synapse dataframe.

### mapping.make\_syn\_maps(df, pre\_neurs, post\_neurs, min\_synapses=5, exclude=[])

Makes connection matrices out of a synapse dataframe.

Parameters

syn\_df pandas.DataFrame

A partner dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

pre\_neurs list-like

The presynaptic neuron IDs.

post\_neurs list-like

The postsynaptic neuron IDs.

min\_syns int

If a neuron has fewer than this amount of connections in the matrix, it is given a weight of 0 in the minimized weight map.

exclude list-like

Any neuron IDs included in this list are given a weight of 0 in the final maps.

Returns

syn\_map np.array

A two-dimensional map of the number of synapses between presynaptic neurons (rows) and postsynaptic neurons (columns).

weight\_map np.array

A two-dimensional map of the number of synapses between presynaptic neurons (rows) and postsynaptic neurons (columns) out of the total number of synapses the postsynaptic neurons have in the dataframe.

min\_map np.array

Same as *weight\_map*, but excluding neurons with fewer than *min\_syns* synapses to not skew the map towards outliers. This map was used within the research paper.

### mapping.make\_type\_maps(syn\_df, pre\_types, post\_types)

Makes type connection matrices out of a synapse dataframe. Rather than being based on individual neurons, these connection matrices are based on the total connections between different types.

Parameters

syn\_df pandas.DataFrame

A partner dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

pre\_types list-like

The presynaptic neuron typess.

post\_types list-like

The postsynaptic neuron types.

Returns

type\_map np.array

A two-dimensional map of the number of synapses between presynaptic neuron types (rows) and postsynaptic neuron types (columns).

weight\_map np.array

A weight map of the percent of connections from the presynaptic type to the given postsynaptic type. This map was used within the research paper.

### mapping.syn\_header(types)

Creates an array of types and their IDs (for Excel use).

Parameters

types list-like

A list of neuron types.

Returns

header np.array

Contains two rows: the second row is all of the IDs of the given neuron types and the first row is the corresponding neuron type to that ID. This can be placed with connection maps to be demonstrative when exporting to Excel.

### mapping.get\_total\_weight(pre\_types, post\_types, region=”Connectome”, rounded=2, width\_conversion=lambda x:x)

Gets the total weight of pre\_types on post\_types, ranging between 0.0 and 1.0. (For instance, if the pre\_types comprised all of the post\_types’ connections, the function would return 1.0).

Parameters

pre\_types list-like

The presynaptic types.

post\_types list-like

The postsynaptic types.

region str

The region in which the weight is calculated.

rounded int

How many decimal places to round the returned weight.

width\_conversion function

A linear conversion equation to determine how large bars should be in Fig. EDF12.

Returns

weight\_dict dict

Keys are the given presynaptic types, and values are the synaptic weight that the pre types have on the post types in the given region.

# meshes.py

This module allows for the generation of neuron meshes for 3D renders. It generates meshes in the .ply file format.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

link str

The link for generating FlyWire meshes.

cv cloudvolume.CloudVolume

CloudVolume has functionality for saving meshes.

patched bool

A cached value that demonstrates whether *navis.patch\_cloudvolume()* has been run. It adds crucial parameters to *cv.mesh.get()*. If not, it is run the first time a mesh is generated.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### meshes.get\_file\_path(file\_name)

Generates a file path in the Meshes directory with a .ply file extension.

Parameters

file\_name str

The desired file name of the mesh.

Returns

file\_path str

The final file path for saving a mesh based on the given *file\_name*.

### meshes.save\_neurs(neur\_ids, file\_name=“”, downsample=False)

Saves a single neuron or multiple neurons as a single mesh.

Parameters

neur\_ids int, str, or list-like

The neuron ID or a list of neuron IDs you want the mesh of.

file\_name str

The desired file name of the mesh.

downsample bool

Setting this to True reduces the mesh quality but improves performance. It is useful for large meshes. It uses a downsampling factor of 50.

Returns

None

### meshes.save\_individually(neur\_types, downsample=False)

Saves each neuron within the list of neur\_types individually. The neurons are saved as {root\_id}.ply in the Meshes subdirectory.

Parameters

neur\_types list-like

Neuron types from Neuron Spreadsheet.

downsample bool

Setting this to True reduces the mesh quality but improves performance. It is useful for large meshes. It uses a downsampling factor of 50.

Returns

None

### meshes.save\_types(neur\_types, downsample=False)

As opposed to *meshes.save\_individually()*, which takes in types and saves all the neurons of each type individually, this function saves each type as an aggregated mesh including all of the neurons within that type.

Parameters

neur\_types list-like

Neuron types from Neuron Spreadsheet.

downsample bool

Setting this to True reduces the mesh quality but improves performance. It is useful for large meshes. It uses a downsampling factor of 50.

Returns

None

### meshes.save\_types\_together(neur\_types, downsample=False)

This function is similar to *meshes.save\_types()*, but aggregates all of the neurons of all of the types into a single mesh..

Parameters

neur\_types list-like

The neuron types from Neuron Spreadsheet.

downsample bool

Setting this to True reduces the mesh quality but improves performance. It is useful for large meshes. It uses a downsampling factor of 50.

Returns

None

# metu\_comparison.py

This module contains many functions used for generating figures comparing MeTu neuron subtypes and neurotransmitters. Specifically, it was used in Figs EDF4, 9, 10, and 12. The module is often imported as *comparison*.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

font\_size int

The font size of figures. The default is 6.

dot\_size int

The dot size of figures. The default is 0.4.

dv\_table pandas.DataFrame

The dataframe that contains MeTu dendrite information. Some column name alterations and additions are performed to make figure creation easier.

spacing dict

Keys are column names in *dv\_table*, values are lists that include the ranges of y-values for certain figures.

colors dict

Keys are MeTu subtypes, keys are hex codes for figure colors.

metu\_types list

All of the MeTu subtypes.

nts list

All of the neurotransmitter types included in automated neurotransmitter prediction.

adjusted\_name int

The font size of figures. The default is 6.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Enums

**metu\_comparison.FormatType**

For generated scatter plots, figures can be generated with two different formats: ME\_POSITION and RETINOTOPY. ME\_POSITION has axis ticks while RETINOTOPY does not.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### metu\_comparison.compare\_metu(column, save\_figure=True, plot\_folder=“”)

Makes a strip plot comparing all MeTu subtypes based on their values in a column within *dv\_table*.

Parameters

column str

The relevant column in *dv\_table* of which a strip plot will be made.

save\_figure bool

Whether the figure will be saved. The default is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

Returns

None

### metu\_comparison.scatter\_plots(x, y, type\_palette={}, plot\_name=“”, plot\_folder=“”, save\_figure=True, fig\_size=(1.9, 1.5), format\_type=FormatType.ME\_POSITION)

Makes scatter plots comparing two columns of dv\_table for each MeTu subtype, as well as a plot with all subtypes together and their line of best fits together as two different subplots.

Parameters

x str

The column in *dv\_table* to be plotted on the x-axis.

y str

The column in *dv\_table* to be plotted on the y-axis.

type\_palette dict

Allows for specification of a color scheme, keys are MeTu subtypes and values are colors. Defaults to the variable *colors*.

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved. The default is True.

fig\_size tuple

The size of the figure. The default is (1.9, 1.25).

format\_type <enum ‘FormatType’>

If FormatType.ME\_POSITION, has x-ticks. If FormatType.RETINOTOPY, does not have x-ticks. The one used for figures was FormatType.ME\_POSITION, the default.

Returns

None

### metu\_comparison.nt\_by\_types(neur\_types, plot\_names=[“NTs”], plot\_folder=“”, region=“Connectome”, palette=[“#000000”], save\_figure=True, fig\_size=(1.6, 1.25), separate\_plots=False)

Makes a plot or multiple plots of the neurotransmitters for the neuron types.

Parameters

neur\_types list-like

Neuron types from Neuron Spreadsheet.

plot\_names list-like

The names of the generated plot(s).

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

region str

If included, limits the NT prediction to a given region.

palette list-like

The colors of the figure dots for the neurons in *neur\_types*.

save\_figure bool

Whether the figure will be saved.

fig\_size tuple

The size of the figure.

separate\_plots bool

If True, separates each neuron type into its own figure. Otherwise, plots them all together.

Returns

None

### metu\_comparison.get\_mt4e()

Retrieves putative MeTu4e from the spreadsheet *MeTu4a\_4e\_candidates.xlsx* as well as MeTu4a that are putatively not MeTu4e.

Returns

types dict

Keys are “Putative MeTu4e” and “Other MeTu4a”, values are numpy.arrays of the relevant neurons of those putative types.

### metu\_comparison.mt4e\_partner\_comparison(plot\_name, plot\_folder=“”, save\_figure=False)

Makes plots comparing

Parameters

plot\_name str

The name of the plot file if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved.

Returns

df pandas.DataFrame

A dataframe containing the partner information obtained by the function.

### metu\_comparison.mt4e\_dendrite\_comparison(plot\_name, plot\_folder=“”, save\_figure=False)

Makes plots comparing dendrites of MeTu4a and putative MeTu4e.

Parameters

plot\_name str

The name of the plot file if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved.

Returns

dorsal\_locs pandas.DataFrame

A dataframe containing the dendrite information obtained by the function.

# neuprint\_reading.py

This module contains functionality for figures that contain information from the Hemibrain dataset using the Neuprint library and the Neuprint Spreadsheet. Throughout the library, this module is imported as *neuread*.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

client Neuprint.Client

The client, which is necessary for fetching Neuprint information. This is initialized with *neuprint\_reading.initialize\_client()*.

neuprint\_neuron\_dict dict

Similar to *mapping.neur\_ids*, contains a list of Neuprint neuron IDs for each neuron type that is loaded with *neuprint\_reading.add\_types()*.

comparison\_data pandas.DataFrame

Contains information from the spreadsheet Neuprint Spreadsheet.

ratio\_pairs dict

Keys are values from the Ratio enum. Values are lists of lists, upon which each interior list contains two different neuron types that are compared for their ratios. These lists are used in *neuprint\_reading.ratio\_plot()*.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Enums

**neuprint\_reading.Ratio**

When creating ratio plots, these are the options for creating plots. They include METU\_TO\_TUBU, TUBU\_TO\_METU, RING\_TO\_TUBU, and TUBU\_TO\_RING. The ratios they reference are self-explanatory.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### neuprint\_reading.initialize\_client()

Initializes the variable *client* if this has not been done previously. This is necessary for running Neuprint functions.

Returns

None

### neuprint\_reading.add\_types(types)

Retrieves IDs of types and caches them in *neuprint\_neuron\_dict*.

Parameters

types list-like

A list of Neuprint neuron types.

Returns

None

### neuprint\_reading.get\_connections(pre\_types, post\_types)

Finds the sum of all postsynaptic connections to individual presynaptic neurons.

Parameters

pre\_types list-like

A list of Neuprint neuron types that will be presynaptic connections in the chart.

post\_types list-like

A list of Neuprint neuron types that will be aggregated as postsynaptic connections.

Returns

syn\_df pandas.DataFrame

The dataframe with information about the connections between presynaptic neurons and postsynaptic types.

### neuprint\_reading.hemi\_syn\_counts(types, region, outliers=[], y\_axis=“Synapse Count”)

Gives the counts of each neuron’s synapses within a given region.

Parameters

types list-like

A list of Neuprint neuron types.

region str

A Neuprint region.

outliers list-like

Outliers, any IDs included in the list will have 0 connections in the output.

y\_axis str

What the label of the y\_axis will be.

Returns

neur\_dict dict

Contains information about the ID, Neuron Type, Synapse Type, and *y\_axis* of each neuron in *types*.

### neuprint\_reading.lobula\_counts(just\_metu4, plot\_name=“”, plot\_folder=“”, save\_figure=True)

Creates a figure of the lobula synapse counts of all MeTu4 neurons or all MeTu neurons, with the labels being MC61 and MC64.

Parameters

just\_metu4 bool

If True, only the neurons identified as MeTu4 will be included. If False, contains all MeTu neurons. In the final plot, they will be labeled based on whether Neuprint labels them MC61 or MC64.

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

Returns

df pandas.DataFrame

The dataframe that contains information shown in the outputted plot.

### neuprint\_reading.ratio\_plot(ratio\_type, plot\_name, plot\_folder=“”, save\_figure=True)

Plots the ratios between two sets of neurons.

Parameters

ratio\_type <enum ‘Ratio’>

The type of ratio you want plotted. It may be METU\_TO\_TUBU, TUBU\_TO\_METU, RING\_TO\_TUBU, or TUBU\_TO\_RING.

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

Returns

ratio\_dict dict

A dictionary including the plotted information.

### neuprint\_reading.plot\_comparison(compare\_types, datasets={“FAFB”: “LR”, “Hemibrain”: “R”}, plot\_name=“Neuron Counts, plot\_folder=“”, y\_ticks=np.arange(0,3,1), save\_figure=True, fig\_size=(2.0, 1.25)

Compares neuron counts between hemispheres and datasets.

Parameters

compare\_types list-like

The neuron types you want compared (without hemisphere noted).

datasets dict

Keys are datasets you want to include in the plot (“FAFB”, “Hemibrain”, “FIB-SEM”), and values are which hemispheres you want included (“LR”, “L”, “R”). Both Hemibrain and FIB-SEM only have the right hemisphere.

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

y\_ticks numpy.array

A range of values to include as the y-ticks, generally given by *numpy.arange()*.

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

fig\_size tuple

The desired size of the figure.

Returns

neuron\_counts dict

The count of each neuron type, with their datasets and hemispheres.

# path\_weights.py

This module is used to determine broad connectivity patterns between sets of neurons that may be connected through multiple intermediates.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

all\_tb\_types list

A list of all TuBu types without hemisphere information.

tb\_l list

A list of all TuBu types in the left hemisphere.

tb\_r list

A list of all TuBu types in the right hemisphere.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### path\_weights.get\_full\_df()

Imports the spreadsheet with all neurons and connections. Full783.feather must be in the Readable folder.

Returns

full\_df pandas.DataFrame

The dataframe with all synaptic connections.

### path\_weights.get\_cx\_df(full\_df=pandas.DataFrame())

Gets a dataframe with all synapses in which CX non-interneurons are postsynaptic and the synapses are not within the CX itself (exempting TuBu presynapses which are sometimes labeled as located within the CX while they are really in the bulb).

Parameters

full\_df pandas.DataFrame

The dataframe retrieved from *path\_weights.get\_full\_df()*. If no dataframe is passed, it is created with that function.

Returns

cx\_df pandas.DataFrame

The dataframe as described above.

### path\_weights.get\_layer2(neur, cx\_df, min\_weight=0.01)

Gets all of the relevant presynaptic neurons from a CX neuron. Layer2 refers to neurons that are presynaptic to CX neurons.

Parameters

neur numpy.int64

The CX neuron you would like the presynaptic partners of (from among Layer2 neurons in the CX\_Neurons spreadsheet).

cx\_df pandas.DataFrame

The dataframe retrieved from *path\_weights.get\_cx\_df()*.

min\_weight float

The minimum weight that the neuron is required to supply to *neur* to be included.

Returns

weight\_dict dict

A dictionary where keys are the presynaptic neurons and values are their weights to *neur*.

### path\_weights.get\_layer2\_df(full\_df)

Gets a dataframe with all synapses in which Layer2 neurons are postsynaptic. Layer2 refers to neurons that are presynaptic to CX neurons.

Parameters

full\_df pandas.DataFrame

The dataframe retrieved from *path\_weights.get\_full\_df()*.

Returns

cx\_df pandas.DataFrame

The dataframe as described above.

### path\_weights.get\_layer3(neur, cx\_df, min\_weight=0.01)

Gets all of the relevant presynaptic neurons from a Layer2 neuron. Layer2 refers to neurons that are presynaptic to CX neurons and Layer3 refers to neurons that are presynaptic of Layer2 neurons.

Parameters

neur numpy.int64

The Layer2 neuron you would like the presynaptic partners of (from among Layer3 neurons in the CX\_Neurons spreadsheet).

layer2\_df pandas.DataFrame

The dataframe retrieved from *path\_weights.get\_layer2\_df()*.

min\_weight float

The minimum weight that the neuron is required to supply to *neur* to be included.

Returns

weight\_dict dict

A dictionary where keys are the presynaptic neurons and values are their weights to *neur*.

### path\_weights.get\_renderable\_neurs(neur\_type, min\_weight=0.01)

Gets the path of a specific neuron type, and tells which neurons should be rendered.

Parameters

neur\_type str

A neuron type (does not include hemisphere suffix).

min\_weight float

The minimum weight to include when querying.

Returns

neur\_dict dict

Contains the neurons, including what layer and hemisphere they are in.

### path\_weights.make\_weight\_strip\_plot(plot\_name=“”, plot\_folder=“”, save\_figure=True)

Makes a weight strip plot of CX neuron types.

Parameters

plot\_name str

The name of the plot.

plot\_folder str

The folder to save the plot in.

save\_figure bool

Whether to save the figure at the given *plot\_name* in the *plot\_folder*.

Returns

weight\_df pandas.DataFrame

The resulting dataframe that is plotted.

# proofreading.py

This module is used to create spreadsheets that demonstrate the proofreading edit history of FlyWire. *proofreading.all\_codex\_names()* is used to generate the spreadsheet “Codex Naming History.xlsx” and *proofreading.full\_percent\_spread()* is used to generate the spreadsheet “FlyWire Consortium Edit Record.xlsx.”

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

client caveclient.frameworkclient.CAVEclientFull

The CaveClient, used to retrieve the change log table.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### proofreading.empty\_change\_df()

Makes an empty dataframe that can be combined with the edits of other neurons, specifically for neurons that have issues accessing their data.

Returns

df pandas.DataFrame

The dataframe as described above.

### proofreading.get\_change\_log(ids)

Retrieves the change log of the given IDs.

Parameters

ids list-like

Neuron IDs from which to retrieve the change log.

Returns

proof\_dict dict

Keys are neuron IDs, values are dataframes with their edit histories.

### proofreading.individual\_change\_logs(ids)

Similar to *proofreading.get\_change\_log()*, but queries the IDs individually, so if there is a problem neuron it creates a blank dataframe instead.

Parameters

ids list-like

Neuron IDs from which to retrieve the change log.

Returns

proof\_dict dict

Keys are neuron IDs, values are dataframes with their edit histories.

### proofreading.condense\_change\_log(change\_log)

Condenses a dictionary of change dataframes into a single dataframe.

Parameters

change\_logs dict

A change log retrieved from *client.chunkedgraph.get\_tabular\_change\_log()* or the above change log functions.

Returns

change\_df pandas.DataFrame

The concatenated dataframe.

### proofreading.proof\_types(neur\_types)

Returns a condensed change log based on neuron types.

Parameters

neur\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet).

Returns

full\_df pandas.DataFrame

A dataframe of all edits of the neurons of the given types.

### proofreading.proof\_counts(full\_df)

Gives the edit counts of each user.

Parameters

full\_df pandas.DataFrame

A dataframe given by *proofreading.proof\_types()*.

Returns

proof\_df pandas.DataFrame

A dataframe with the edit counts of each person.

### proofreading.proof\_counts\_types(neur\_types)

Retrieves the edit counts of each user given the neuron types.

Parameters

neur\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet).

Returns

proof\_df pandas.DataFrame

A dataframe with the edit counts of each person for the neurons of the given types.

### proofreading.neuron\_counts(neur\_types)

Shows the total edits each person performed on each neuron.

Parameters

neur\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet).

Returns

user\_df pandas.DataFrame

A dataframe showing the total number of edits that each person performed among those neurons. They are included if they contributed >=10% of the edits of the neuron.

### proofreading.full\_percent\_spread(neur\_types)

Shows how much each lab contributed to certain neurons.

Parameters

neur\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet).

Returns

edit\_df pandas.DataFrame

Breaks down how much each lab contributed to editing each neuron out of neur\_types. The labs are included if editors in the lab in total contributed >=10% of the edits of the neuron.

### proofreading.all\_codex\_names(neur\_types, keep\_our\_names=True)

Makes a spreadsheet of all relevant named neurons in Codex, and compares how they were annotated over time. Given that some annotations that were performed by our lab were not recorded by FlyWire, this function accounts for those neurons and the time that they were actually annotated.

Parameters

neur\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet).

keep\_our\_names bool

If True, includes annotations made by people in the Kim Lab and Wernet Lab (Dustin Garner, Emil Kind, Ben Gorko, and Lucy Houghton). If False, excludes annotations by those labs.

Returns

label\_df pandas.DataFrame

The dataframe including the annotation information of each neuron over time.

# readfiles.py

This module imports various files that are used throughout the library. Readable files are stored in the *Readable* directory.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### readfiles.import\_file(file\_name, sheet\_name=0, usecols=None, file\_type=“xlsx”, dtype=None)

Imports a file from the *Readable* directory.

Parameters

file\_name str

The name of the file in the *Readable* directory (do not include the file extension).

sheet\_name str or int

If the file is a spreadsheet, the name or index of the sheet to retrieve. The default is 0 (the first sheet).

usecols str, list-like

If the file is a spreadsheet, the name or names of columns to retrieve. The default is None (imports all columns).

file\_type str

The file extension. The default is “xlsx.” Imports spreadsheets when given “xlsx,” “csv,” or “feather,” and imports a text file as a string when given “txt.”

dtype str

If the file is a spreadsheet, the data type that it imports column as. The default is None (imports the columns without altering the data type).

Returns

file pandas.Dataframe, str

The file as a dataframe or a str, depending on *file\_type*.

### readfiles.import\_input()

Imports “MeTu\_input.xlsx” and renames the columns to align with past versions that had different column names.

Returns

input\_df pandas.Dataframe

The MeTu input file as a dataframe.

### readfiles.import\_proofreading(just\_connectivity=False)

Imports “MeTu\_proofreading.xlsx,” depending on which version is wanted.

Parameters

just\_connectivity bool

Whether sheet 0 (False) or sheet 1 (True) is wanted. Sheet 0 is default.

Returns

proof\_df pandas.Dataframe

The MeTu proofreading file as a dataframe.

### readfiles.import\_joined()

Imports “joined\_avp\_table.xlsx.”.

Returns

joined\_df pandas.Dataframe

The joined\_avp\_file as a dataframe.

### readfiles.import\_regions()

Imports “regions.xlsx” with their include and exclude points.

Returns

regions\_dict dict

Makes a dictionary of the coordinate areas included and excluded from each region.

### readfiles.import\_coords()

Imports “Neuron Spreadsheet.xlsx” coordinates and previous IDs.

Returns

neur\_types dict

Keys are neuron types and values are arrays of coordinates of each neuron of those types.

prev\_ids dict

Keys are neuron types and values are arrays IDs found in the Neuron Spreadsheet. Previously, this was used to determine whether IDs were up-to-date, but now is obsolete due to using the standardized Materialization Version 783.

### readfiles.import\_neuprint\_data()

Imports the Hemibrain neurons from the Comparison sheet of “Neuron Spreadsheet.xlsx.”

Returns

df pandas.DataFrame

The Hemibrain neurons from the spreadsheet.

### readfiles.import\_colors()

Imports the color scales from “Colors.xlsx” as *matplotlib.pyplot.ListedColormap* for use in connectivity charts and synapse density functions.

Returns

new\_colors dict

Keys are the color names and values are colormaps usable in figures.

# reduction.py

This module includes functions that take in synapse dataframes given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()* and reduce their information in various useful ways.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

region\_dict dict

Region information given by *readfiles.import\_regions()*.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### reduction.remove\_autapses(df)

Removes autapses from a synapse dataframe. This is no longer necessary to run because *fafbseg.flywire.get\_synapses()* has a *clean* parameter that removes autapses and is by default set to true.

Parameters

df pandas.DataFrame

A synapse dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

Returns

df pandas.DataFrame

The input dataframe with autapses removed.

### reduction.remove\_min\_synapses(df, min\_syns=1, pre\_or\_post=“post”)

Removes neurons from a synapse dataframe that have fewer than min\_syns connections to the partner neurons.

Parameters

df pandas.DataFrame

A synapse dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

min\_syns int

Neurons with fewer than this number of connections to the partner neurons are removed.

pre\_or\_post str

Should be “pre” or “post”. This represents from which perspective this function is run. For example, if this parameter is “pre”, than any postsynaptic neurons with fewer than *min\_syns* connections in the dataframe are removed.

Returns

df pandas.DataFrame

The input dataframe with less connected neurons removed.

### reduction.remove\_min\_weight(df, min\_weight=0.05, pre\_or\_post=“post”)

Removes neurons from a synapse dataframe that have fewer than min\_weight connection weight to the partner neurons.

Parameters

df pandas.DataFrame

A synapse dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

min\_weight float

Neurons with less than this weight to the partner neurons are removed.

pre\_or\_post str

Should be “pre” or “post”. This represents from which perspective this function is run. For example, if this parameter is “pre”, than any postsynaptic neurons with less than *min\_weight* connection weight in the dataframe are removed.

Returns

df pandas.DataFrame

The input dataframe with less connected neurons removed.

### reduction.lim\_vol(df, include=numpy.array([]), exclude=numpy.array([]), coord\_names=[f“post\_{x}” for x in “xyz”])

Takes in a synapse dataframe and returns one with its synapses limited to a convex volume made up of *include* points, excluding a convex volume made up of *exclude* points.

Parameters

df pandas.DataFrame

A synapse dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

include numpy.array

An array of coordinates that make up the volume that is wanted to be included in the final dataframe.

exclude numpy.array

An array of coordinates that make up the volume that is wanted to be excluded from the final dataframe. This overrides *include*. For instance, the medulla volume excludes a portion that overlaps with the lobula.

coord\_names list-like

Should have three values, and they refer to how the coordinates are named in the dataframe. Sometimes this is “post\_x”, etc. and sometimes it is just “x”, etc.

Returns

df pandas.DataFrame

The input dataframe only with synapses in the given volume.

### reduction.lim\_region(df, region, coord\_names=[f“post\_{x}” for x in “xyz”])

Limits a synapse dataframe to synapses within a given region in the Regions spreadsheet.

Parameters

df pandas.DataFrame

A synapse dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

region str

A region found in the “Include” sheet of the Regions spreadsheet.

coord\_names list-like

Should have three values, and they refer to how the coordinates are named in the dataframe. Sometimes this is “post\_x”, etc. and sometimes it is just “x”, etc.

Returns

df pandas.DataFrame

The input dataframe only with synapses in the given region.

### reduction.remove\_region(df, region, coord\_names=[f“post\_{x}” for x in “xyz”])

Limits a synapse dataframe to synapses outside a given region in the Regions spreadsheet.

Parameters

df pandas.DataFrame

A synapse dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

region str

A region found in the “Include” sheet of the Regions spreadsheet.

coord\_names list-like

Should have three values, and they refer to how the coordinates are named in the dataframe. Sometimes this is “post\_x”, etc. and sometimes it is just “x”, etc.

Returns

df pandas.DataFrame

The input dataframe only with synapses outside the given region.

### reduction.coords\_to\_region(coords, region)

Limits a coordinate array to the given region.

Parameters

coords list-like

An array of coordinates.

region str

A region found in the “Include” sheet of the Regions spreadsheet.

Returns

lim\_coords numpy.array

The given coordinates limited to the given region.

### reduction.lines\_to\_region(coords1, coords2, region)

Takes in two same-sized sets of coordinates and returns coordinates that are both within the region. I am uncertain what the use of this function was, it was not used in the paper.

Parameters

coords1 list-like

An array of coordinates.

coords2 list-like

An array of coordinates, must be the same size as *coords1*.

region str

A region found in the “Include” sheet of the Regions spreadsheet.

Returns

lim\_coords1 numpy.array

Coordinates from *coords1* that were in the given *region* as long as the matching-indexed coordinate from *coords2* is also in *region*.

lim\_coords2 numpy.array

Coordinates from *coords2* that were in the given *region* as long as the matching-indexed coordinate from *coords1* is also in *region*.

# specific.py

This module includes functions that generate specific plots found throughout the paper that do not fit in any other broad category.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

pie\_fig\_size tuple

The size of pie charts generated by *specific.make\_pie\_chart()* and *specific.make\_bihem\_pie\_charts()*.

aotu\_subregions dict

Keys are AOTU subregions and values are lists of the MeTu and TuBu subtypes found within them.

bulb\_subregions dict

Keys are Bulb subregions and values are lists of the TuBu and Ring subtypes found within them.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### specific.sm17\_map\_by\_dv\_axis(plot\_name=“”, plot\_folder=“”, save\_figure=True)

Plots MeTu3b and MeTu3c synaptic counts with respect to their offset from the medulla centroid along the D-V axis (Fig. EDF7a).

Parameters

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

type\_dict dict

The plotted data.

### specific.fafb\_syn\_counts(neur\_types, region, plot\_name=“Synapse Counts”, plot\_folder=“”, y\_axis=“Synapse Count”, save\_figure=True)

Compares regional synaptic counts in the FAFB dataset.

Parameters

neur\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet).

region str

The region in Regions spreadsheet to limit the plot.

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

y\_axis str

What to name the y-axis.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

full\_df pandas.DataFrame

The plotted data.

### specific.full\_comparison(neur\_types, region, plot\_name=“Synapse Counts”, plot\_folder=“”, neu\_outliers=[], y\_axis=“Synapse Count”, save\_figure=True)

Compares regional synaptic counts between the FAFB and Hemibrain datasets. This utilizes *specific.fafb\_syn\_counts()* and *neuprint\_reading.hemi\_syn\_counts()* to make Fig. EDF9h.

Parameters

neur\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet).

region str

The region in Regions spreadsheet to limit the plot.

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

neu\_outliers list-like

Hemibrain outlier neurons to exclude.

y\_axis str

What to name the y-axis.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

full\_df pandas.DataFrame

The plotted data.

### specific.mt3\_pre\_connections(pre\_types, plot\_name=“”, plot\_folder=“”, save\_figure=True)

Makes a strip plot of MeTu3 subtypes and their presynaptic connections (Fig. EDF7b).

Parameters

pre\_types list-like

Neuron types in *mapping.neur\_coords* (Neuron Spreadsheet) presynaptic to MeTu3.

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

neur\_df pandas.DataFrame

The plotted data.

### specific.tutu\_comparison(plot\_name=“TuTu Synapse Counts”, plot\_folder=“”, save\_figure=True)

Makes a bar graph comparing the percentages of TuTu connections that are MeTu, TuBu, or other on the ipsilateral and contralateral sides (Fig. EDF4f before labels were added).

Parameters

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

conn\_dict dict

The plotted data.

### specific.get\_region\_avg\_counts(bihem\_type, region, side, relevant\_subregions={})

Retrieves the average number of pre and post connections of ipsilateral or contralateral side from a given bihemispheric neuron type in a certain region.

Parameters

bihem\_type str

The type of bihemispheric neuron without a hemisphere (“TuTuB\_a”, “TuTuB\_b”, or “AOTU046”).

region str

The region from Regions spreadsheet to limit the search.

side str

Either “ipsi” or “contra” for the side of the desired connections.

relevant\_subregions dict

A dict that contains the names of subregions as keys with lists of neurons within those regions as values. The variables *aotu\_subregions* and *bulb\_subregions* are examples.

Returns

syn\_counts dict

The average number of pre and post connections.

### specific.make\_pie\_chart(syn\_counts, bihem\_type, region, side, folder\_path=[])

Makes pie charts based on the return dict of *specific.get\_region\_avg\_counts()*.

Parameters

syn\_counts dict

A dict given by *specific.get\_region\_avg\_counts()*.

bihem\_type str

The type of bihemispheric neuron without a hemisphere (“TuTuB\_a”, “TuTuB\_b”, or “AOTU046”).

region str

The region from Regions spreadsheet to limit the search.

side str

Either “ipsi” or “contra” for the side of the desired connections.

folder\_path list-like

The folder path by which to store the pie charts.

Returns

None

### specific.make\_bihem\_pie\_charts(plot\_folder=“”)

Makes all relevant bihemispheric pie charts with the correct relative sizing used in Fig. EDF4b and EDF4di-ii.

Parameters

plot\_folder str

Where to save the pie charts.

Returns

None

### specific.get\_bihem\_weights(min\_weight=0.05)

Gets the average ipsilateral and contralateral connection weight between bihemispheric neurons and MeTu and TuBu neurons in the AOTU.

Parameters

min\_weight float

The lowest weight value that will be included in the returned dictionary.

Returns

weight\_dict dict

The weights of connections between neurons in the AOTU.

### specific.get\_bihem\_weight\_line\_width(min\_weight=0.05)

Gets the line width of arrows for Fig. EDF4a, using the formula y=8.611x+1.5494 (the formula can be any linear transformation, what matters is how thick the lines are relative to each other).

Parameters

min\_weight float

The lowest weight value that will be included in the returned dictionary.

Returns

line\_dict dict

The weights of connections between neurons in the AOTU transformed by the formula.

# synapse\_density.py

This module contains functions that generate synapse density figures. It is often imported simply as density.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Variables

regions dict

Boxed region information given found in the “Boxes” sheet of the Regions spreadsheet.

subregion\_types dict

Contains the “pre” and “post” neuron subtypes found in each subregion of the AOTU.

type\_colors dict

For each neuron subtype, has the (transparent) color map sheet name found in the Colors spreadsheet that is used to create the figures.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Enums

**synapse\_density.AotuRegion**

Contains the four AOTU subregions (POSTERIOR\_LATERAL, POSTERIOR\_CENTRAL, ANTERIOR, MEDIAL).

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### synapse\_density.rotate\_coord(coord, center, angle, scale=1)

Rotates the coordinate on the z-axis around a center point.

Parameters

coord numpy.array

An [x,y,z] coordinate that will be rotated around the z-axis.

center numpy.array

The [x,y,z] point around which *coord* will be rotated.

angle float

The angle of rotation (in degrees).

scale float

The new scale of the coordinate in reference to the center point.

Returns

coord numpy.array

The rotated and rescaled coordinate.

### synapse\_density.make\_box(xmin, xmax, ymin, ymax, zmin, zmax)

Gives the vertex coordinates of a box given the min and max values.

Parameters

xmin float

The minimum x value of the box.

xmax float

The maximum x value of the box.

ymin float

The minimum y value of the box.

ymax float

The maximum y value of the box.

zmin float

The minimum z value of the box.

zmax float

The maximum z value of the box.

Returns

coords numpy.array

The 8 corner coordinates of the box, stored as a 2D array.

### synapse\_density.transform\_box(box, region=“AOTU\_R”, angle=30, scale=1.3)

Performs transformations on a box and returns the transformed coordinates.

Parameters

box numpy.array

A box, for instance given by *synapse\_density.make\_box()*.

region str

The region, used for finding the center point of rotation.

angle float

The angle by which to rotate the box.

scale float

The scale by which to resize the box.

Returns

box numpy.array

The new 8 corner coordinates of the box, stored as a 2D array.

### synapse\_density.get\_steps(region=“AOTU\_R”, spacing=10)

Creates a set of equidistant steps along each axis depending on the region.

Parameters

region str

The region by which to find the steps (based on the boxes from the Regions spreadsheet).

spacing int

The distance apart each step should be.

Returns

x\_steps numpy.array

Equally spaced steps along the x-axis.

y\_steps numpy.array

Equally spaced steps along the y-axis.

z\_steps numpy.array

Equally spaced steps along the z-axis.

### synapse\_density.make\_map(df, x\_steps, y\_steps, z\_steps, spacing, plane, vert=False, reverse={True: False, False: False}, region=“AOTU\_R”, angle=30, scale=1.3)

Recursively makes a synapse density map given a synapse dataframe and a region. This is optimized by alternately cutting the dataset in half vertically and horizontally each recursion step and ceasing recursion as soon as a passed dataframe is empty.

Parameters

df pandas.DataFrame

Synapse dataframe given by *flywire\_functions.fetch\_synapses()* or *fafbseg.flywire.get\_synapses()*.

x\_steps numpy.array

Equally spaced steps along the x-axis, can be given by *synapse\_density.get\_steps()*.

y\_steps numpy.array

Equally spaced steps along the y-axis, can be given by *synapse\_density.get\_steps()*.

z\_steps numpy.array

Equally spaced steps along the z-axis, can be given by *synapse\_density.get\_steps()*.

spacing int

The distance apart each step should be.

plane str

“xy”, “zy”, or “xz” depending on which plane the map is viewed from.

vert bool

Used for alternating direction during recursion, should be False upon running the function.

region str

The region by which the search is performed.

angle float

The angle by which to rotate the box.

scale float

The scale by which to resize the box.

Returns

syn\_density numpy.array

A synapse density matrix given the step size and view plane.

### synapse\_density.xy\_map(df, steps, region=“AOTU\_R”, angle=30, scale=1.3)

Uses *synapse\_density.make\_map()* from the “xy” plane point of view.

### synapse\_density.zy\_map(df, steps, region=“AOTU\_R”, angle=30, scale=1.3)

Uses *synapse\_density.make\_map()* from the “zy” plane point of view.

### synapse\_density.xz\_map(df, steps, region=“AOTU\_R”, angle=30, scale=1.3)

Uses *synapse\_density.make\_map()* from the “xz” plane point of view.

### synapse\_density.get\_all\_maps(df, spacing=10, region=“AOTU\_R”, angle=30, scale=1.3)

Creates maps by *synapse\_density.make\_map()* along each axis.

Parameters

df pandas.DataFrame

Synapse dataframe given by *flywire\_functions.fetch\_synapses()* or *fafbseg.flywire.get\_synapses()*.

spacing int

The distance apart each step should be.

region str

The region by which the search is performed.

angle float

The angle by which to rotate the box.

scale float

The scale by which to resize the box.

Returns

syn\_densities tuple

All three synapse density maps in (“xy”, “zy”, “xz”) order.

### synapse\_density.maps\_from\_types(neur\_types, spacing=10, region=“AOTU\_R”, angle=30, scale=1.3)

Creates maps from specific neuron types.

Parameters

neur\_types list-like

Neuron types from the Neuron Spreadsheet..

spacing int

The distance apart each step should be.

region str

The region by which the search is performed.

angle float

The angle by which to rotate the box.

scale float

The scale by which to resize the box.

Returns

syn\_densities tuple

All three synapse density maps in (“xy”, “zy”, “xz”) order.

### synapse\_density.plot\_maps(maps, color=“TPurple”, blur=4, max\_value=0, plot\_name=“”, plot\_folder=“”, save\_figure=True)

Plots synapse density maps.

Parameters

maps tuple

Maps given by *synapse\_density.get\_all\_maps()*.

color str

A color within the Colors spreadsheet.

blur float

The sigma value of the Gaussian blur applied to the plot.

max\_value float

A value to be put in the corner of the plots for normalization.

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

None

### synapse\_density.plot\_region\_density(subregion, blur=10, plot\_folder=“”, save\_figure=True)

Makes MeTu and TuBu synapse density plots in a given AOTU subregion.

Parameters

subregion <enum ‘AotuRegion’>

A subregion from the enum AotuRegion.

blur float

The sigma value of the Gaussian blur applied to the plot.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

all\_maps dict

The maps generated for each neuron subtype.

### synapse\_density.plot\_type\_density(types, blur=4, plot\_name=“”, plot\_folder=“”, save\_figure=True)

Plots synapse density of a certain neuron type.

Parameters

types list-like

A list of neuron types from which to retrieve the synapse density.

blur float

The sigma value of the Gaussian blur applied to the plot.

color str

A color within the Colors spreadsheet.

plot\_name str

The name of the plot will be saved if *save\_figure* is True.

plot\_folder str

The directory where the plot will be saved if *save\_figure* is True.

save\_figure bool

Whether the figure will be saved. The default is True.

Returns

maps tuple

The maps generated by *synapse\_density.maps\_from\_types()*.

# utils.py

This module contains various helper functions that are used throughout the library.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Classes

### utils.TreeNode

This is a node for the binary tree SearchTree class.

**Methods**

#### \_\_init\_\_(self, value)

Initializes the node with *value*.

### utils.SearchTree

A binary search tree.

**Methods**

#### print\_all(self)

Prints all of the node values and their depths.

#### get\_values(self)

Returns a list of all values in ascending sorted order.

#### has\_values(self, value)

Checks whether *value* is contained within the tree.

#### insert(self, value)

Inserts *value* into the tree.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### utils.time\_elapsed(func)

A wrapper function that prints the time a function takes to run.

Parameters

func function

A function.

Returns

wrap function

A function that performs and returns what *func* does and prints the time elapsed.

### utils.attempt\_func(func, repeat\_times=15)

If a function fails to run, it is repeated *repeat\_times* - 1 times until it succeeds. This was used for *fafbseg.flywire.get\_synapses()* when there would sometimes be server errors when making queries.

Parameters

func function

A function.

repeat\_times int

The number of times the function should be attempted before giving up.

Returns

wrap function

A function that performs and returns what *func* does after attempting it *repeat\_times* times.

### utils.notify\_finished(func)

A wrapper function that makes a noise when the function is completed. This is useful for functions that take a long time.

Parameters

func function

A function.

Returns

wrap function

A function that performs and returns what *func* does.

### utils.copy\_ids(neur\_ids, make\_unique=False, output=True)

A function that copies neuron IDs to the clipboard, which is useful for pasting them directly into FlyWire.

Parameters

neur\_ids list-like

A list of neuron IDs to copy.

make\_unique bool

Whether to make the neuron IDs unique. This is not necessary if pasting them to FlyWire because FlyWire automatically makes pasted neuron IDs unique.

output bool

If True, prints the number of IDs that were copied to the clipboard.

Returns

None

### utils.row\_to\_str(id\_row)

Takes a pandas row of IDs that are in numpy.int64 form and changes them to str. This is necessary when exporting them to Excel because Excel rounds large numbers, losing the ID.

Parameters

id\_row pandas.Series

A series of neuron IDs (numpy.int64).

Returns

new\_row numpy.array

The same series as “U50”.

### utils.syn\_df\_to\_str(syn\_df)

Takes a synapse dataframe and converts the “pre” and “post” columns to str for Excel.

Parameters

syn\_df pandas.DataFrame

A partner dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

Returns

syn\_df pandas.DataFrame

The same DataFrame with “pre” and “post” rows as str for exporting to Excel.

### utils.coord\_str\_to\_list(coord)

Takes in a coordinate string (normally from Excel) and returns them as a list.

Parameters

coord str

A coordinate in “(x, y, z)” form. It may not have the parentheses or spaces, but the commas are necessary.

Returns

coord\_list list

The same coordinate in [x, y, z] Python list form.

### utils.coord\_str\_to\_arr(coord)

Takes in a coordinate string (normally from Excel) and returns them as a numpy array.

Parameters

coord str

A coordinate in “(x, y, z)” form. It may not have the parentheses or spaces, but the commas are necessary.

Returns

coord\_list numpy.array

The same coordinate in [x, y, z] numpy array form.

### utils.coord\_list\_to\_str(coord)

Takes in a coordinate string as a numpy array and returns it as a string.

Parameters

coord numpy.array

An [x, y, z] coordinate array.

Returns

coord str

The same coordinate in “x, y, z” form.

### utils.coord\_column\_to\_array(column)

Takes in a pandas series of coordinate strings and returns them as an array of coordinate arrays, ready for *fafbseg.flywire.locs\_to\_segments()* or *flywire\_functions.locs\_to\_segments()*.

Parameters

column pandas.DataFrame

A series of coordinates in “(x, y, z)” form, generally taken from a spreadsheet. They may not have the parentheses or spaces, but the commas are necessary.

Returns

coord\_arr numpy.array

An array of [x, y, z] coordinate arrays.

### utils.basemodel\_list\_to\_df(basemodels)

Takes in a list of pydantic basemodels and returns a dataframe of them. For instance, if there is a list of three basemodels with “ID” and “NeurType” fields, then those fields will be columns in the dataframe and each basemodel will be a row.

Parameters

basemodels str

A list of any data class inheriting from *pydantic.BaseModel*.

Returns

data pandas.DataFrame

The fields of the basemodel in dataframe form.

### utils.write\_excel(df, file\_name)

Takes dataframe and writes it to Excel.

Parameters

df pandas.DataFrame

A dataframe to be exported.

file\_name str

The desired name of the file. It will be stored in the “Excel-Plots” folder.

Returns

None

### utils.usable\_coords(x\_coords, y\_coords, z\_coords, divide)

Takes in three lists of x, y, and z coordinate components and returns them as usable coordinates for importing into FlyWire.

Parameters

x\_coords list-like

An array of x coordinate components.

y\_coords list-like

An array of y coordinate components.

z\_coords list-like

An array of z coordinate components.

divide bool

Sometimes coordinates are found in nanometer physical form and must be converted to FlyWire’s units by dividing by (4, 4, 40). If True, performs this division.

Returns

coord\_array numpy.array

An array of “x, y, z” coordinates in str form for export to spreadsheets.

### utils.make\_importable\_syn\_coords(syn\_df, pre\_or\_post=“pre”, file\_name=“Importable Coordinates”, save\_file=True)

Takes a synapse dataframe and turns it into an importable coordinate spreadsheet for FlyWire.

Parameters

syn\_df pandas.DataFrame

A partner dataframe given by *fafbseg.flywire.get\_synapses()* or *flywire\_functions.fetch\_synapses()*.

pre\_or\_post str

“pre” or “post”. Whether the importable coordinates should be the presynaptic or postsynaptic sites. This does not change much because the sites are usually nanometers apart.

file\_name str

The desired name of the file.

save\_file bool

Whether to save the file.

Returns

coord\_df pandas.DataFrame

The dataframe that can be imported by FlyWire.

### utils.importable\_coords(coords, file\_name=“Importable Coordinates”, save\_file=True)

Makes a csv file of the coordinates that can be imported to FlyWire as annotations. FlyWire has a specific format for importing annotations, and this creates it. If *save\_file*, stores the file in the “Importable-Coords” folder.

Parameters

coords list-like

A list of coordinates.

file\_name str

The desired name of the file.

save\_file bool

Whether to save the file.

Returns

coord\_df pandas.DataFrame

The dataframe that can be imported by FlyWire.

### utils.importable\_coord\_lines(coords1, coords2, file\_name=“Importable Coordinates”, save\_file=True)

Makes a csv file of coordinate lines that can be imported to FlyWire as annotations (there is a line between each *coords1* and *coords2* entry). FlyWire has a specific format for importing annotations, and this creates it. If *save\_file*, stores the file in the “Importable-Coords” folder.

Parameters

coords1 list-like

A list of coordinates. Should be the same size as *coords2*.

coords2 list-like

A list of coordinates. Should be the same size as *coords1*.

file\_name str

The desired name of the file.

save\_file bool

Whether to save the file.

Returns

coord\_df pandas.DataFrame

The dataframe that can be imported by FlyWire.

### utils.count\_instances(neurons, min\_neurs=5)

Returns a dict of the count of neurons in the input array if that count is greater than *min\_neurs*.

Parameters

neurons list-like

A list of neuron IDs.

min\_neurs int

The minimum number of occurrences for a neuron to be included in the final dict.

Returns

neur\_dict dict

Keys are neurons and values are the number of instances of the values.

### utils.build\_tree(values)

Builds a binary search tree initialized with the given values.

Parameters

values list-like

The values (likely neuron IDs) with which to initialize the search tree.

Returns

tree utils.SearchTree

A binary search tree containing the values.

# volumes.py

This module includes functions that determine how neuron volumes have changed over time. F1-Score is the test used to calculate changes in neuron volumes and synapse counts, as seen in the flagship FlyWire paper, [*Dorkenwald, 2021*](https://www.nature.com/articles/s41592-021-01330-0).

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Functions

### volumes.make\_change\_log()

Creates the dataframe that was saved as “MeTu\_Change\_Log.xlsx” from “MeTu\_proofreading.xlsx”. Since this function takes a long time to run, running *readfiles.import\_file(“MeTu\_Change\_Log”)* gets the same information and is faster.

Returns

change\_log pandas.DataFrame

A dataframe detailing the changes.

### volumes.l2\_info(save\_file=True)

Gets the l2 info of neurons in “MeTu\_Change\_Log.xlsx”.

Parameters

save\_file bool

Whether the dataframe will be saved as an excel file.

Returns

l2 pandas.DataFrame

The retrieved l2 information.

### volumes.get\_f1\_score(true\_positive, false\_negative, false\_positive)

Calculates the F1-Score based on the given values. The F1 calculation equation is TP / (TP + (FP+FN)/2).

Parameters

true\_positive float

The number of items in both the pre-edited and the post-edited neurons.

false\_negative float

The number of items in the post-edited but not the pre-edited neurons.

false\_positive float

The number of items in the pre-edited but not the post-edited neurons.

Returns

f1\_score float

The F1-Score based on the given values.

### volumes.get\_f1\_nodes(before\_id, after\_id)

Gets the F1-Score by comparing the nodes of a neuron before and after editing.

Parameters

before\_id np.int64

The neuron ID from before editing.

after\_id np.int64

The neuron ID from after editing.

Returns

f1\_score float

The F1-Score of the skeletal nodes compared between the two IDs.

### volumes.get\_f1\_synapses(before\_id, after\_id)

Gets the F1-Score by comparing the synapse counts of a neuron before and after editing.

Parameters

before\_id np.int64

The neuron ID from before editing.

after\_id np.int64

The neuron ID from after editing.

Returns

f1\_score float

The F1-Score of the synapse counts compared between the two IDs.

### volumes.compare\_f1\_scores()

Gets the F1-Scores of the nodes and synapses of “MeTu\_Change\_Log.xlsx” neurons.

Returns

node\_f1\_scores dict

Arrays of F1-Scores of skeletal nodes for all neurons between the three rounds of proofreading.

syn\_f1\_scores dict

Arrays of F1-Scores of synapse counts for all neurons between the three rounds of proofreading.

### volumes.plot\_f1(f1\_scores, plot\_folder=“”)

Plots the skeletal node and synapse F1-Score charts. Used in EDF1biii.

Parameters

f1\_scores tuple

Items given by *volumes.compare\_f1\_scores()*.

plot\_folder str

The folder in which to save the plots.

Returns

None

### volumes.t\_test(round\_comparison)

Calculates p-values through paired-samples t-test on a round dict.

Parameters

round\_comparison dict

A dictionary containing the round keys (“before\_first”, “after\_first”, and “before\_second”) and values are arrays of numbers.

Returns

p-values dict

A dictionary containing the compared p-values.

### volumes.get\_round\_values()

Gets the round values from the proofreading “volume\_change” and “number\_of\_edits” spreadsheets.

Returns

p-values dict

A dictionary containing the compared p-values.

### volumes.get\_all\_t\_tests(f1\_scores)

prints p-scores from t-tests on each of the four proofreading round data.

Parameters

f1\_scores tuple

The return value of *volumes.compare\_f1\_scores()*.