SBFSEM-tools tutorial

Sara Patterson

University of Washington

September 18, 2017

SBFSEM-tools

Goal: basic analysis support requiring minimal setup and background in computers, math, etc.

Current capabilities:

- Import relevant Tulip data (single neurons and networks)
- Count up synapses and condense synapses spanning multiple slices
- Summarize synapses with statistics and plots
- Sync with Blender images and cell fills
- Basic network analysis

Next:

- Compare neurons
- Additional network analysis

Changes

15Sept2017:

- Matlab's webread has replaced the Tulip import system as the default. Support for Tulip import remains, however, as it can be useful for certain situations.

Some changes since the first release:

- All display units are now in microns
- Stratification histogram (use the Add Cell Skeleton checkbox while the Z-axis histogram is active)
- Export current figure to a new window `[Menu -> Export -> ]`
- Plot soma mosaic with `vissoma.m`

Install

First download or clone SBFSEM-tools.
Make sure SBSFEM-tools is added to your MATLAB path like so:

```
addpath(genpath('C:\...\sbfsem-tools'));
```

If you already have JSONLab installed, make sure

```
which loadjson
```

returns the version in sbfsem-tools. Otherwise, you might get some errors.

Basic Use: Neuron

The Neuron class is the basic representation of a cell exported from Viking. To create a Neuron:

```
% cellName = Neuron(cellID, 'source');
c207 = Neuron(207, 'temporal');
```

Note: Source can be 'inferior', 'temporal' or 'rc1'. Also, abbreviating to 'i', 't' and 'r' works as well.
Open the Neuron class in the user interface

```
NeuronApp(c207);
```

To update a neuron in the workspace:

```
c207.update();
```

Cell Info Panel

This entire panel is designed with a future directory class in mind. As of now, you don't need to save each Neuron, so only set these if you have a specific reason for doing so.

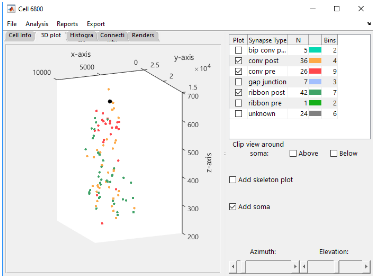
- If known, the cell type and subtype will be helpful for connectivity analysis.
- The other properties will eventually be used for cell queries but don't have much use yet.

After changing any of the attributes on the Cell Info Panel, make sure to press the `[Add to cell data]` button. This will make sure your changes are reflected next time you open the UI.

3d plot

Components

You can add and remove each synapse type, the soma node and the skeleton independently using the checkboxes. Rotate the plot with the elevation and azimuth sliders.

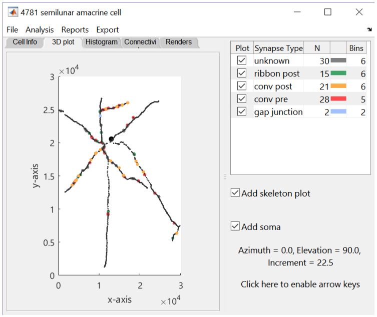

Cell 6800
File Analysis Reports Export
Cell Info 3D plot Histogra Connecti Renders
x-axis
y-axis
z-axis
Clip view around soma: Above Below
Azimuth:
Elevation:

Plot	Synapse Type	N		Bins
☐	bip conv p...	5	🟩	2
✓	conv post	36	🟧	4
✓	conv pre	26	🟥	9
☐	gap junction	7	🟦	3
✓	ribbon post	42	🟩	7
☐	ribbon pre	1	🟩	2
☐	unknown	24	⬛	6

☐ Add skeleton plot

✓ Add soma

AII amacrine cell synapses


4781 semilunar amacrine cell
File Analysis Reports Export
Cell Info 3D plot Histogram Connectivi Renders
y-axis
x-axis
×10^4

Plot	Synapse Type	N		Bins
✓	unknown	30	⬛	6
✓	ribbon post	15	🟩	6
✓	conv post	21	🟧	6
✓	conv pre	28	🟥	5
✓	gap junction	2	🟦	2

✓ Add skeleton plot

✓ Add soma

Azimuth = 0.0, Elevation = 90.0, Increment = 22.5

Click here to enable arrow keys

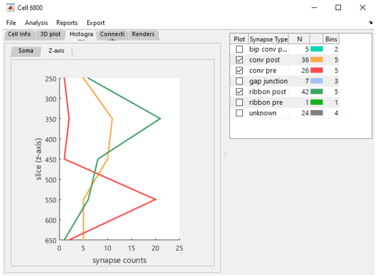
Semilunar synapses & skeleton

Histograms

There are two histograms. These plot synapse count as a function of:

- Distance from soma
- Section number (z-axis)

Use the Synapse Table to edit the number of bins. Add the cell skeleton to see dendrite stratification.



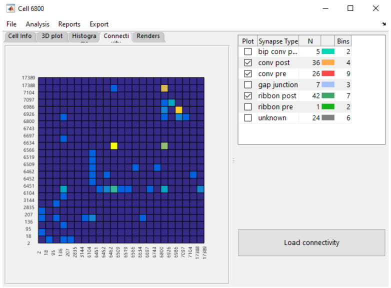
Plot	Synapse Type	N		Bins
☐	bip conv p...	5	🟩	2
✓	conv post	36	🟧	5
✓	conv pre	26	🟥	5
☐	gap junction	7	🟦	3
✓	ribbon post	42	🟩	5
☐	ribbon pre	1	🟩	1
☐	unknown	24	⬛	4

Z-axis synapse distribution for a putative AII AC. This synapse asymmetry isn't news but is still good to see.

Connectivity

To add connectivity data, save a network map in Tulip as a JSON file, as described above. Then in Matlab:

```
c207.addConnectivity('c207hops.json');
```



Plot	Synapse Type	N		Bins
☐	bip conv p...	5	🟩	2
☑	conv post	36	🟧	4
☑	conv pre	26	🟥	9
☐	gap junction	7	🟦	3
☑	ribbon post	42	🟩	7
☐	ribbon pre	1	🟩	2
☐	unknown	24	⬛	6

The connectivity matrix is weighted by the number of unique synapses between two cells (dark blue for 0 synapses). Directed synapses will only register a contact from the pre → post-synaptic neuron.

Connectivity for the AII AC.

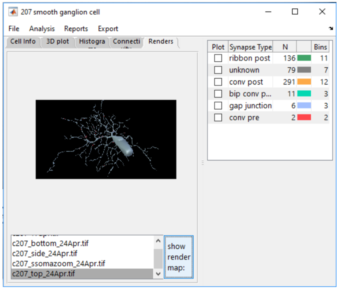
Connectivity

The network data is split into edges and nodes, each with their own table. The tables can be exported to Excel as `.csv` or as a text file from the UI menu bar.

To print a easily readable table to the command line use the networkTable function

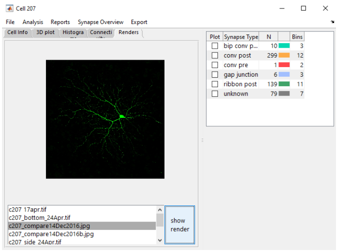
```
networkTable(c207);
```

Blender Renders and Cell Fills



Plot	Synapse Type	N		Bins
☐	ribbon post	136	🟩	11
☐	unknown	79	⬛	7
☐	conv post	291	🟧	12
☐	bip conv p...	11	🟩	3
☐	gap junction	6	🟦	3
☐	conv pre	2	🟥	2

```
c207_bottom_24Apr.tif
c207_side_24Apr.tif
c207_ssomazoom_24Apr.tif
c207_top_24Apr.tif
```



Plot	Synapse Type	N		Bins
☐	bip conv p...	10	🟩	3
☐	conv post	299	🟧	12
☐	conv pre	1	🟥	2
☐	gap junction	6	🟦	3
☐	ribbon post	139	🟩	11
☐	unknown	79	⬛	7

```
c207 17apr.tif
c207_bottom_24Apr.tif
c207_compare14Dec2016.jpg
c207_compare14Dec2016b.jpg
c207 side 24Apr.tif
```

For now, set the `renderDir` in `getFilepaths.m` to the file your images are saved into. The UI find the images if their filename includes the letter ‘c’ followed by cell number (like ‘c207’). I hope to improve this at some point.

This isn’t limited to renders and could include whatever images and diagrams are helpful. For example, I can compare my ON-smooth cell reconstructions and cell fills.

Reports

This is pretty limited... Right now you can generate two reports:

- Location IDs of unknown synapses
- An overview of all synapse types

The report name is auto-generated and will overwrite any existing reports of the same type for that neuron.
Let me know any other reports that would be of use.

Save

Neurons do not need to be saved, as the underlying data is replaced with each update. However, if you would like to save a neuron for offline work, here's some guidelines:

Save Cell Info

After adding information to the Cell Info Panel, make sure to press the button `[Add to cell info]`.
This saves cell attributes to the Neuron object.

Save Neuron

To save the Neuron object itself, go to `File->Save Cell` or save from the command line to the current directory.

Export

Configuration

These are found in the `defaults` folder.

1. **Directories** - to save time navigating to the correct directory each time, edit `getFilepaths.m`
2. **Cell types and subtypes** - go to `getCellTypes.m` and `getCellSubtypes.m`. Let me know if any are missing.
3. **Synapse colors** - go to `getStructureColors.m` to change the default colors for each synapse.

NeuronAnalysis Class

The NeuronAnalysis class helps keep population data organized by managing input parameters and results of common analyses. To create a new analysis, subclass NeuronAnalysis and edit the `doAnalysis` and `visualize` methods.

See `Tutorial.m` for information on these existing classes:

- **DendriticFieldHull** - uses convex hull to estimate dendritic field area, includes methods for removing axons prior to analysis.
- **PrimaryDendriteDiameter** - returns the median dendrite diameter at a given distance from the soma.

Mosaic Class

The Mosaic class is a first attempt at support for groups of neurons. This allows the most important properties of each neuron to be accessed and analyzed as a group.

Mosaic is the parent class that should not be used directly. Instead use Generic or one of the cell type specific subclasses (BipolarCells, Photoreceptors, HorizontalCells).

Mosaic

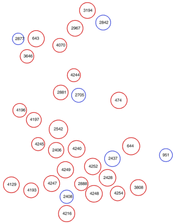
These core methods are common to all Mosaic subclasses, not just Generic.

```
% Create a Mosaic by passing a Neuron
bip = Generic(c142);
% Add a description of the mosaic
bip.describe('s-off bipolar cells');
% Add Neuron to the mosaic
bip.add(c1411);
% Remove a Neuron
bip.rmNeuron(1411); % by cell number
bip.rmRow(2); % by row number
% To view the neurons in the cmd line:
bip
% Mosaic is essentially matlab's table. To use the
      full range of table methods, cast to table:
T = mosaic2table(bip);
```

Mosaic Visualization

The Mosaic class grew from functions designed to view the photoreceptor mosaic. Accordingly, the visualization methods are most developed.

```
% Basic plot of cell 'somas'
% Plot to existing axis
bip.somaPlot('ax', axesHandle);
% Include cell ID labels
bip.somaPlot('lbl', true);
% Set the color and linewidth
bip.somaPlot('co', [1 0 0], 'lw', 1);
```



Cone mosaic with cellID labels

Additional cell-type specific parameters can be found in each Mosaic class' description.

Appendix

Here's some information and methods that are less essential:

- Links
- Tulip import method
- Will add more with next update...

Links

The SBFSEM-tools repository can be found on Github.
I included two open source matlab toolboxes: JSONLab and the GUI Layout Toolbox.
The software used for annotations is Viking, developed by Jamie Anderson and the Marc Lab at University of Utah.
Useful free programs involved in these analyses:

- Tulip supports graph visualization. The documentation for Tulip's Python package can be found here.
- Blender for 3D renders of neurons.

SBFSEM-tools was developed in the Neitz lab at University of Washington.

Import

Step One: Tulip

Open a cell in Tulip and then open the Python command line. It's on the bottom toolbar.
Set the file name* and file path:

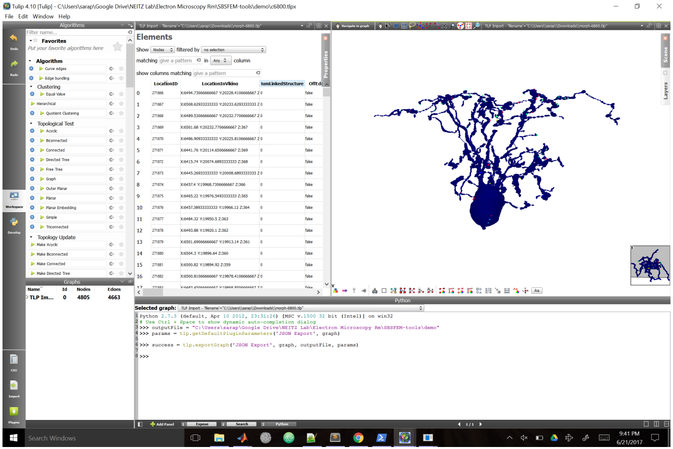
```
    outputFile = "C:\...\c207.json"
```

Then run these two lines:

```
    params = tlp.getDefaultPluginParameters('JSON
   Export', graph)
    success = tlp.exportGraph('JSON Export', graph
   , outputFile, params)
```

Import

Step One: Tulip



TLP Import - 'filename'="C:\\Users\\sarap\\Downloads\\morph-6800.tlp"

Elements

Show Nodes filtered by no selection

matching give a pattern in Any column

show columns matching give a pattern

	LocationID	LocationInViking	lumLinkedStructure	OffEd
0	271866	X:6494.73066666667 Y:20228.4106666667 Z	0	false
1	271867	X:6508.62933333333 Y:20233.6293333333 Z	0	false
2	271868	X:6489.53066666667 Y:20232.7706666667 Z	0	false
3	271869	X:6501.68 Y:20232.7706666667 Z:367	0	false
4	271870	X:6486.90933333333 Y:20225.8106666667 Z	0	false
5	271871	X:6441.76 Y:20114.6506666667 Z:369	0	false
6	271872	X:6415.74 Y:20074.6893333333 Z:368	0	false
7	271873	X:6445.26933333333 Y:20008.6893333333 Z	0	false
8	271874	X:6457.4 Y:19968.7306666667 Z:366	0	false
9	271875	X:6465.22 Y:19976.5493333333 Z:365	0	false
10	271876	X:6457.38933333333 Y:19966.12 Z:364	0	false
11	271877	X:6484.32 Y:19950.5 Z:363	0	false
12	271878	X:6493.88 Y:19920.1 Z:362	0	false
13	271879	X:6501.69066666667 Y:19913.14 Z:361	0	false
14	271880	X:6504.3 Y:19896.64 Z:360	0	false
15	271881	X:6500.82 Y:19894.92 Z:359	0	false
16	271882	X:6500.81066666667 Y:19878.4106666667 Z	0	false
17	271883	X:6483.45066666667 Y:19868.8506666667 Z	0	false

Name	Id	Nodes	Edges
TLP Im...	0	4805	4663

Selected graph: TLP Import - 'filename'="C:\\Users\\sarap\\Downloads\\morph-6800.tlp"

```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32
# Use Ctrl + Space to show dynamic auto-completion dialog
>>> outputFile = "C:\Users\sarap\Google Drive\NEITZ Lab\Electron Microscopy Rm\SBSFEM-tools\demo"
>>> params = tlp.getDefaultPluginParameters('JSON Export', graph)

>>> success = tlp.exportGraph('JSON Export', graph, outputFile, params)

>>>
```

Alternative Streamlined Import

Step One - Tulip

If you're comfortable with Python, you can skip opening Tulip's UI entirely.
First get the Tulip modules:

```
$ pip install tulip-python
```

This will allow you to export `.tlp` (or compressed `.tlp.gz`) files to JSON from a command line without Tulip's UI.

```
from tulip import tlp
graph = tlp.loadGraph("C:\...\morph-207.tlp")
outputFile = "C:\...\c207.json"
params = tlp.getDefaultPluginParameters('JSON
  Export', graph)
success = tlp.exportGraph('JSON Export', graph,
  outputFile, params)
```

Import

Step Two - MATLAB

Load in the JSON file and create a Neuron object:

```
c207 = Neuron('c207.json');
```

A dialog box will ask for the cell number and source (temporal, inferior, rc1). To avoid that, include them while creating the Neuron object:

```
% output = Neuron(filename, cellNumber, source);
c207 = Neuron('c207.json', 207, 'temporal');
```

To update the underlying data for an existing Neuron:

```
c207.updateData('c207.json');
```

Open up the user interface:

```
c207.openUI;
```