# SBFSEM-tools Documentation

Sara Patterson, Neitz Lab

## Contents

## 1 Introduction

SBFSEM-tools is a Matlab toolbox developed for serial EM data and connectomics in the Neitz Lab at University of Washington. While SBFSEM-tools was built around the Viking annotation software, many aspects are quite general and could apply easily to other programs and imaging methods.
SBFSEM-tools imports annotation data through Viking's OData service and parses the results into Matlab data types. This happens behind the scenes so the average user can work with familiar objects (neuron, synapse, etc).
Other features include:

- Single neuron analysis: dendritic field area, dendrite diameter, soma size, stratification, synapse distribution
- Group analysis: density recovery profile, nearest neighbor, synapse statistics
- 3D volume rendering of polygon annotations and free-form traces over a stack of EM images. 2D projections of dendritic fields.
- Generate surfaces from IPL boundary markers, compensate for Z-axis misalignments.
- Misc UIs for visualizing EM images and annotations

## 2 Install

SBFSEM-tools is available on Github: sbfsem-tools. Clone or download the repository and make sure to add it to your MATLAB path.

## 3 Neuron

Most work revolves around the Neuron class, which is a basic representation of a neuron (called a Structure in Viking). A neuron is created with two inputs: the Cell ID and the volume name:

```
% Cell 6800 from NeitzTemporalMonkey
c6800 = Neuron(6800, 't');
% Cell 2795 from NeitzInferiorMonkey
c2795 = Neuron(2795, 'NeitzInferiorMonkey');
% Same cell but using the abbreviated volume name
c2795 = Neuron(2795, 'i');
```

The full volume names are: ‘NeitzTemporalMonkey’, ‘NeitzInferiorMonkey’ and ’MarcRC1’. These can be abbreviated to ’t’, ’i’ and ’r’, respectively.

### 3.1 Neuron Methods

If you are new to MATLAB, methods are class-specific functions. A list of all methods can be obtained by `methods(className)` (so, for example, `methods(‘Neuron’)`).

#### 3.1.1 update

*Description:* Updates the underlying data from OData. Useful when working with a neuron you are currently annotating.
*Syntax:* `obj.update();`

#### 3.1.2 graph

*Description:* Convert the neuron into an undirected or directed graph. For more information on what this enables, see MATLAB’s graph class documentation.
*Syntax:*

## 4 Ultrastructure

### 4.1 Mitochondria

## 5 Markers

### 5.1 IPL Boundary Markers

## 6 Views

Neurons can be passed to several UIs with the following syntax:

```
% VIEWNAME(NeuronObject);
c207 = Neuron(207, 't');
StratificationView(c207);
```

- **NodeView** - 3D scatter plot of each cell and synapse annotation associated with a cell.
- **StratificationView** - Z-axis histogram of dendrites and synapses.
- **SomaDistanceView** - proxmial-distal distribution of dendrites and synapses.

## 7 Renders

### 7.1 Closed Curve

The closed curve renders support cutouts and multiple annotations per section. Mixing closed curve and disc renders is the next goal.

```
c2542 = Neuron(2542, 'i');
lmcone = renderClosedCurve(c2542);
% Decrease the resolution to speed up the renders
lmcone = renderClosedCurve(c2542, 'resizeFactor', 0.5);
```

### 7.2 2D Projection

A brief outline of the steps involved in the 2D projection renders:

1. Query OData service for the XYZ coordinates and radius of each annotation (units = pixels).
2. Query OData service for the volume dimensions - use these to convert XYZ coordinates to microns.
3. Convert the annotations and links between each annotation into the nodes and edges of an undirected graph.
4. Segment dendrites using depth-first search.
5. Render each with MATLAB's built-in `cylinder` function.
6. Rotate each cylinder

## 7.3 VikingPlot

Editing the MATLAB figure generated by VikingPlot can be a good alternative to Blender. A few useful commands:

```
% Make sure the figure is the active window.
ax = gca; % Create a handle to the axis
ax.Children % Returns a list of child structures
% There should be PATCH and LIGHT objects.
% Use numerical indexing to generate handles to the objects
% NOTE: The numbering might be different on your figure
lightObj = ax.Children(1);
renderObj = ax.Children(2);
% A few useful properties to edit for PATCH objects:
renderObj.FaceColor = [0, 0.8, 0.3];
renderObj.FaceAlpha = 0.7;
```

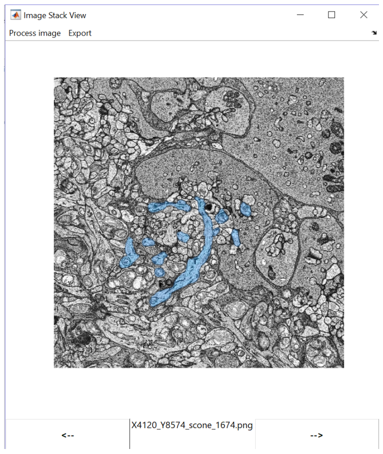
# 8 Images

## 8.1 ImageStackApp

The ImageStackApp was built to scan through a series of images exported using Viking's Export Frames option (although this will work with any folder of images, provided there is some numbering in the file names). Pass a folder name to browse through the images (using left and right arrow keys or buttons). If the filenames contain numbers (Viking's export frames appends a frame number automatically), these will be used to order the images.

```
ImageStackApp('C:\...\foldername');
```

To crop the set of images, use `Process Image` → `Crop` and draw a rectangle on the image. Use `Full size` to return to the original size. Both changes require moving to another image to apply.



## 8.2 ImageStack class

The ImageStackApp creates an ImageStack object. To create a GIF, create an instance of ImageStack alone and use the `stack2gif` function.

```
I = sbfsem.images.ImageStack('C:\...\foldername');
I.stack2gif();
```

# 9 Matlab 101

MATLAB tips are currently scattered throughout the documentation.. consolidate them here.