Luntzel, Jon
CMPSC 292F
John Gilbert
10 June 2021

Digital Current through the Drosophila Connectome

I aimed to simulate a small neuronal pathway found in the drosophila brain using tools in Laplacian theory. Motivated by combining actual results from electron microscopy with cable and graph theory and articulating that with the methods discussed in class, I aimed to acquire a small set of neurons and decompose them into graphs of vertices and edges that retained information about the morphology and each formed a connected component. Then, to explore the pathway as a resistive network, they were condensed (stripped of excess vertices and edges) and an induced current on the network was considered primarily to observe how electrical properties such as voltage and current along the graphs changed over time. Many assumptions were made, such as that the membrane resistance of neurons was infinite, that is, no current was lost to the extracellular space primarily to preserve Kirchoff's law within the mesh and ignore current lost to surroundings.

Outline

- Identify and retrieve a small set of neurons with which to study
- Retain relevant details for computation, remove excess information for speed and clarity
- Model them as a resistor network

The Flywire (see tools cited) database provides a digital electron microscopic reconstruction of a female drosophila brain, and anatomical tracing performed by an undergraduate student coupled with software for automatically determining presynaptic and

postsynaptic partners allowed me to decide a region of interest and compile a set of candidate neurons with which to conduct simulations. Specifically after narrowing the candidates to the anterior visual pathway (AVP), I randomly selected the left posterior side of the brain and anchored myself to one neuron for the ME to AOTU sub-pathway, then found there was one postsynaptic partner from the AOTU to the BU, and finally 20 postsynaptic neurons in the BU to EB leg of the pathway. In the simulations I stratify three datasets: one is a 'toy' graph that is small and manually built (1), one is a set of three neurons in the AVP (2), and the other is the set described above: it is the same as the set of three neurons, but also contains the 20 neurons in the third portion of the pathway (3). After constraining myself to a specific pathway and selecting biologically relevant neurons to focus on, I downloaded them as .h5 or .ply files and prepared them for analysis.

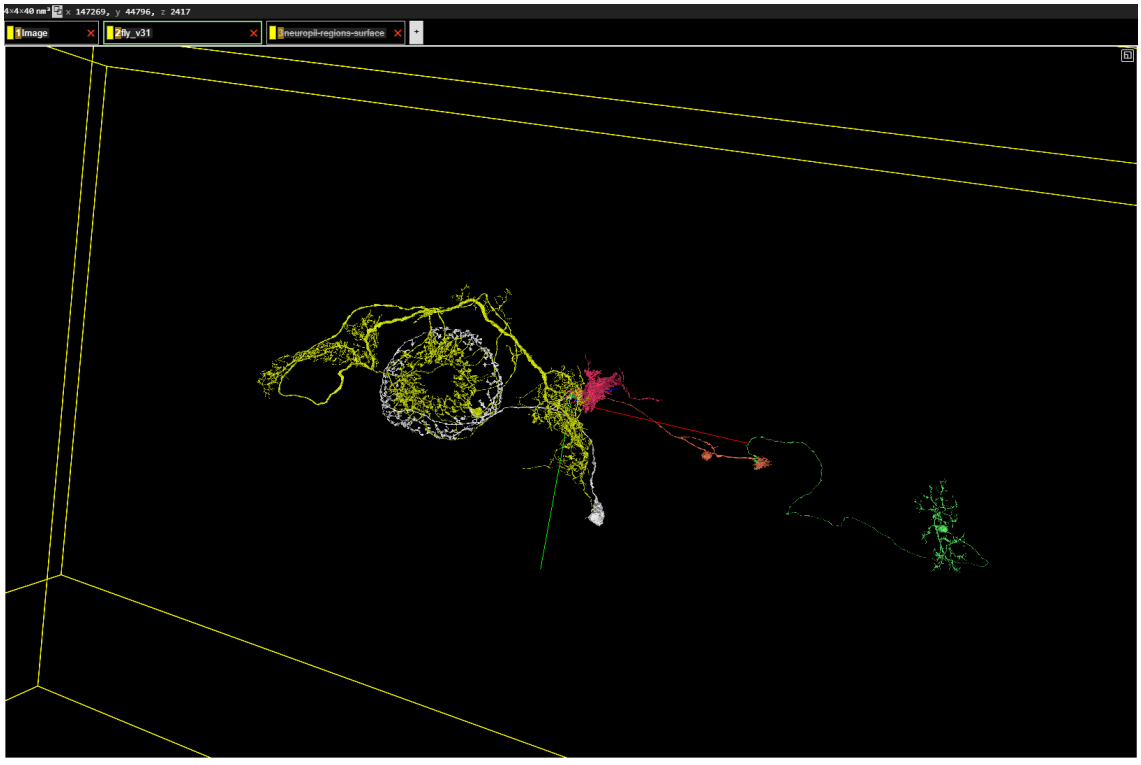


Figure 1: The selected neurons. From right to left: ME, AOTU, BU, EB. Note the single green and orange neuron for segments 1 and 2 of the AVP, as well as the ring-like structure marking the ellipsoid body.

The mesh structures are a collection of vertices given by a list of 3-dimensional coordinates where their index is the vertex number, and a list of tuples that describe the edges by grouping each connected pair as a tuple in the list. If we left the information as-given, we would effectively be simulating how a current travels along the outer surface of neurons, and nevertheless the size of the mesh files as well as their holes, disconnected components, and small perturbations throughout would already be sufficient for looking to store the data in a different way. To resolve the main problem of wanting to simulate how current flows through the axoplasm of a neuron rather than its membrane, skeletonization methods beginning initially with meshparty (tools cited) were used. A skeleton is a wire or frame which captures the most essential morphology of a three-dimensional object and also generally significantly reduces the number of edges and vertices, of which acquiring took the majority of time. After obtaining skeletons was technically feasible, two critical problems arose: how should one obtain and store information about the neuron's rough diameter throughout as a proxy for resistance, and how should one ensure that the resulting skeleton is a single connected component to work with one graph. Considering the radius problem I thought that taking a vector normal to the mesh and rotating it normal to its tangent direction, and then computing the area of the ellipsoid-like irregular structure, to find a radius which preserves the same cross-sectional area, may work well, but I desperately wanted to avoid diving even more into the mesh files and operating on them.

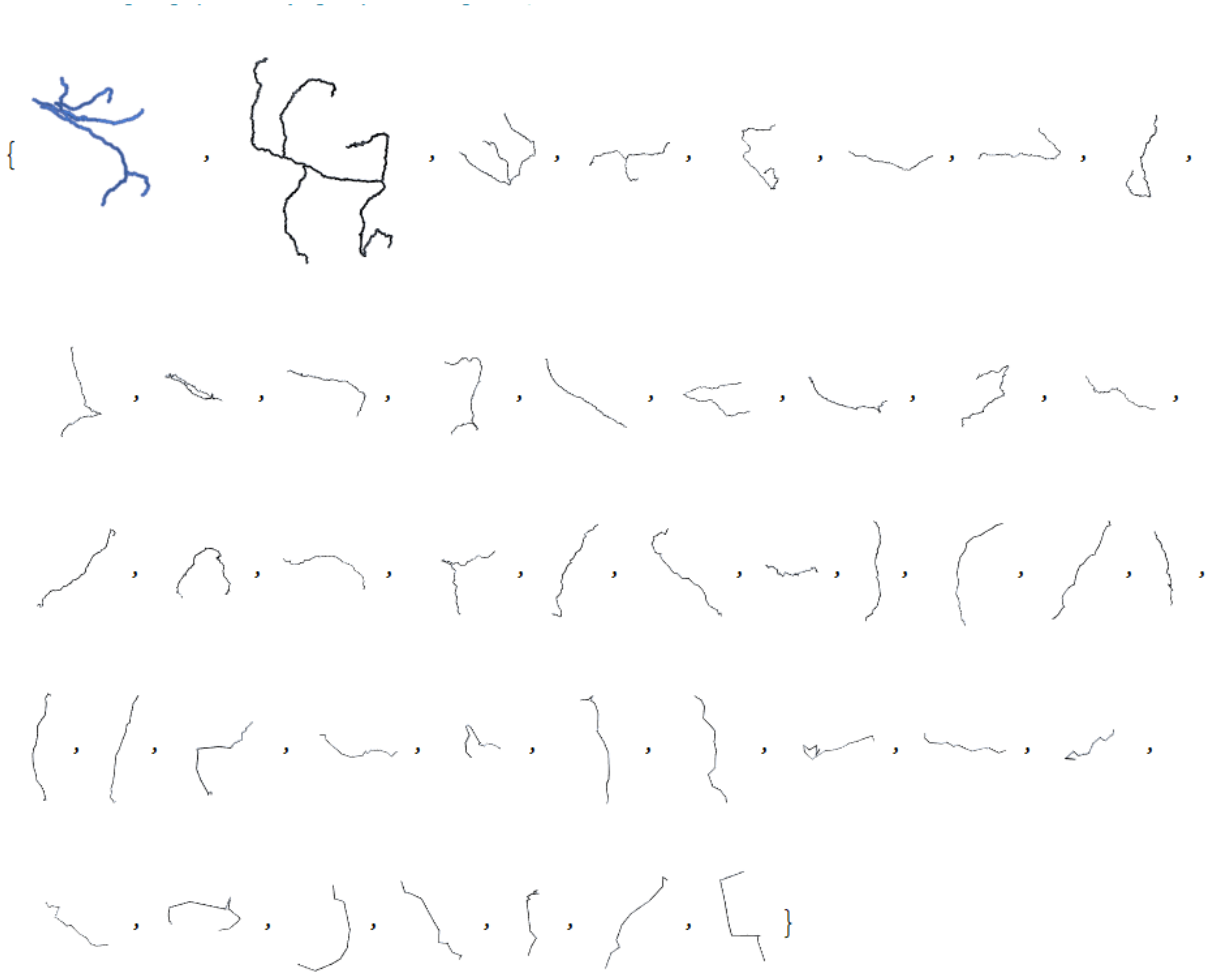


Figure 2: An example of a skeleton that was organized into its 45 connected components.

The first thorny problem of approximating diameter throughout the neuron seemed to me much more difficult and also less critical than the second, so I began to work with the second issue. To address it I first considered manually connecting the connected components using meshparty, but instead sought for software that automatically outputs a single connected component to lessen the workload. To be concise, I found one package which achieved this aim (figure 3), but found that switching methods and using Skeletor in conjunction with NAVis (tools cited) ultimately served to kill two birds with one stone.

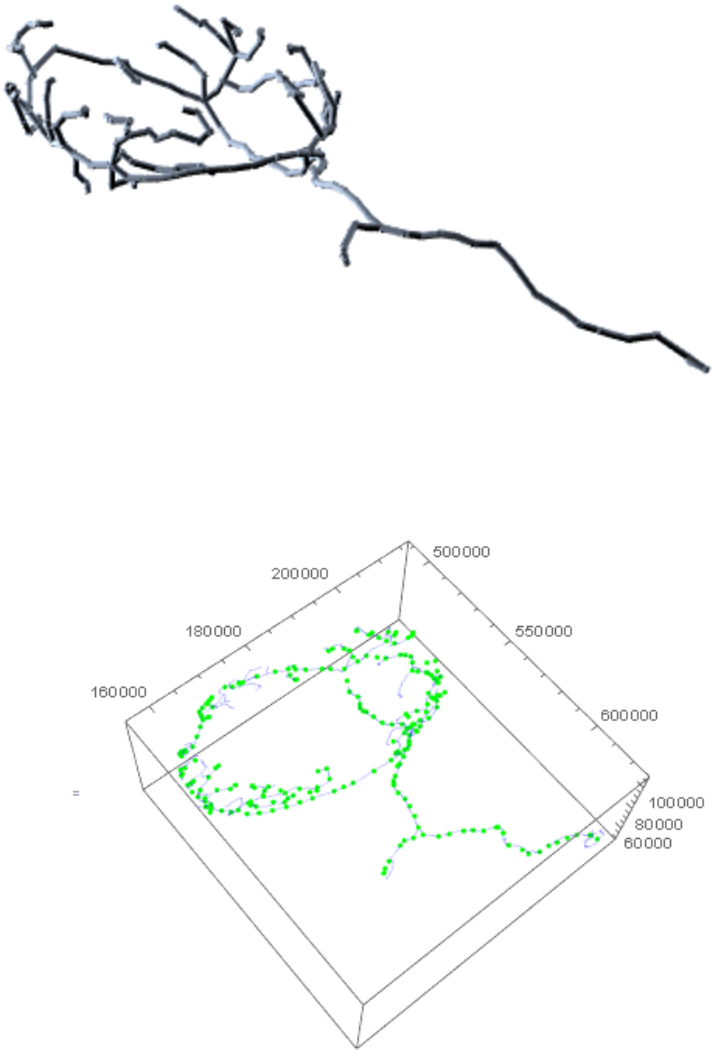


Figure 3: A skeletonization of a ‘ring neuron’, found in the third segment using a disbanded solution (a) and an example skeleton with edges in faded blue and vertices in green (b).

Skeletor provides a handful of mesh algorithms, but the wavefront algorithm in particular was ideal for the project because it centers the skeleton well within the neuronal mesh it derives from. In particular, this simplifies calculating and storing radius information, which ultimately provides the edge weights of the resistive network. Second, Skeletor has a radii attribute for skeleton

objects which pairs each vertex in the skeleton with a radius value. The method for computing radius is quite interesting and can be found in its Github. The program however did not assure the skeleton would be a single connected component, so since the meshes were not proper from the beginning, the result was still a bundle of disconnected graphs. NAVis, a different program, has a 'heal fragmented neuron' function which effectively sews the mesh together by taking shortest paths between fragments and creating edges. In order to communicate between Skeletor and NAVis I saved and loaded swc files which provided a common ground between the two skeleton classes and could to use them in conjunction with each other. The work of making sure I had retained morphological details and a path connecting the ends of the neurons had been completed.

Next, the computational modeling portion began with a toy (1) graph to quickly compute over and check methods with antecedent to the realistic AVP and more interesting domain. It had qualities such as occasional positions with degree greater than 1, and the vertices were located relatively uniformly, but there was enough variation in distance to see deviations in the resistance and to provide a weighted graph.

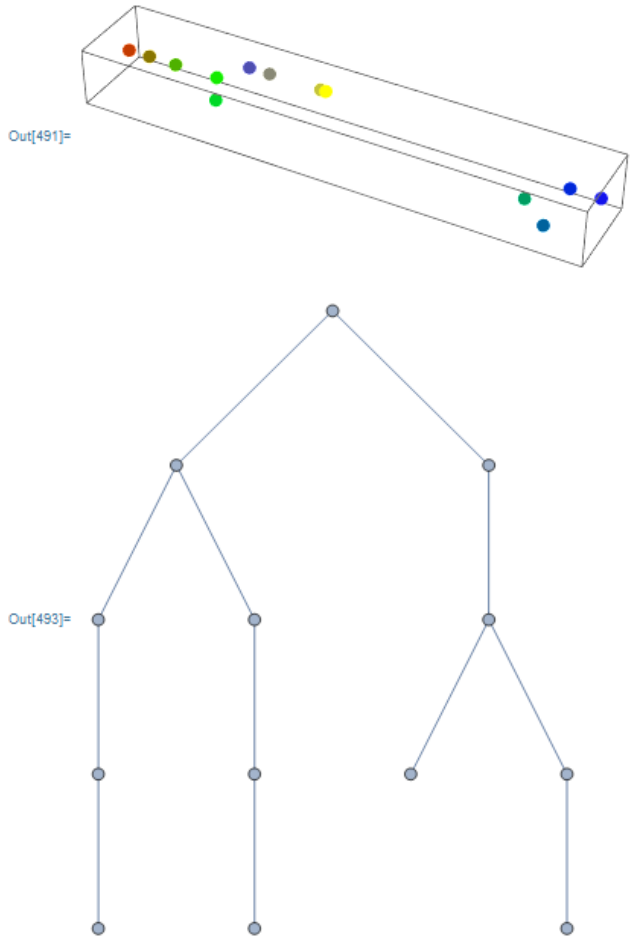


Figure 4: A ‘toy graph’. The upper image color-codes vertices by their order in the vertex list and plots them by 3d-coordinate, while the lower image is a flattened representation of the graph.

To put a neuronal pathway and Laplacian work surrounding resistor networks in tandem, cable theory provided a sensical bridge and so I started to define resistances across edge weights according to cable theory. In a physical system, we know that a longer cable will resist flow more, and regardless of its geometry, if the cross-sectional area is larger, it will conduct more easily. The general formula for a section of cable is:

(rho * length) / area

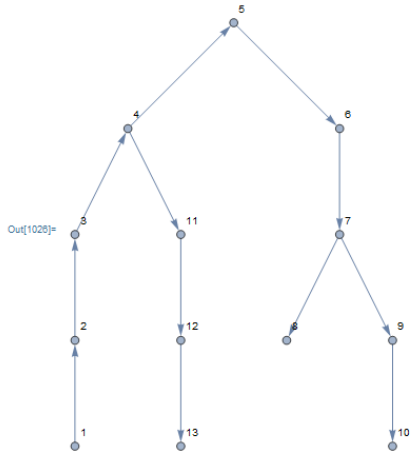
Where rho is the specific electrical resistance in ohms*metres, leaving ohms. We approximate the neuron as cylindrical hence area is 2*pi*r^2, and I assumed fictitious radius values for the toy graph at each vertex. To compute edge resistance, I simply took the average of the vertex radii in the edge and applied the formula above using vertex positions for distance. The specific electrical resistance was found in various literature across animals (and different anatomical regions) such as sea slug, giant squid, and barnacle to be 40-76 ohms-meters, so I decided to set rho=50. Unfortunately I could not find support for drosophila neurons. In order to set the baseline voltage of a neuron, one paper found that ventral lateral neurons had a resting potential of -49 mV, which I took straightforwardly from. Finally, basics about current divider circuits alongside Spielman's notes allowed me to simulate an induced current on a graph and update the corresponding voltages and currents to determine the outgoing current at leaf nodes.

```
Do[vcurs = levels[[m]];
 Do[outs = outies[Ut, vcurs[[j]]]; chn = childs[Ut, outs]; (*get children of a given node*)
  edgereconstruct = ArrayReshape[Riffle[ConstantArray[vcurs[[j]], Length[chn]], chn], {Length[chn], 2}]; (*rebuild edges*)
  edgeinds = {};
  Do[AppendTo[edgeinds, Position[gedges2, edgereconstruct[[i]]]], {i, Length[edgereconstruct]}];
  edgeinds = Flatten[edgeinds]; (*rebuild edge indices*)
  rtot = Sum[R[[edgeinds[[i]]]][[edgeinds[[i]]]], {i, Length[edgeinds]}];
  Do[cr[[edgeinds[[i]]]] = R[[edgeinds[[i]]]][[edgeinds[[i]]]] / rtot, {i, Length[edgeinds]}], {j, Length[vcurs]}], {m, 1, Length[levels]}]
Print["Current ratio: ", cr]

(*current at each vertex, itot*)
Do[vcurs = levels[[m]];
 Do[ins = innies[Ut, vcurs[[j]]]; pts = parents[Ut, innies[Ut, vcurs[[j]]]] ;
  inedges = ArrayReshape[Riffle[pts, ConstantArray[vcurs[[j]], Length[pts]]], {Length[pts], 2}];
  edgeinds = {};
  Do[AppendTo[edgeinds, Position[ge, inedges[[i]]]], {i, Length[inedges]}];
  edgeinds = Flatten[edgeinds]; (*rebuild edge indices*)
  itot[[vcurs[[j]]]] = Sum[itot[[inedges[[k]][[1]]]] * cr[[edgeinds[[k]]]], {k, Length[edgeinds]}], {j, Length[vcurs]}] , {m, 2, Length[levels]}]
```

Figure 5: Snippet of code which describes the step-function for updating the graph. It first computes current ratios. Then, it finds vertices at a given level, locates their incoming edges, and updates current based on the incoming edges. Voltage is a straightforward calculation by Ohm's law.

The process relies on converting undirected graphs into directed acyclic graphs, and being able to sort it into a level graph. The levels are currently computed inefficiently and I did not run a complete analysis on the three large neurons, however a promising approach is to use the Cuthill-McKee algorithm. The Matlab function symrcm(S) might work, and there exists a program to connect Mathematica and Matlab. An example analysis on two toy graphs are displayed in figures 6 and 7:



Incidence matrix:

$$\begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Resistance matrix:

$$\begin{pmatrix} 0.00141391 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0.00129154 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0.00227133 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0.00205431 & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0.0168105 & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0.00397139 & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0.000786795 & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.00642948 & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.0220298 & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.00307743 & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.0031476 \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix} \quad \begin{matrix} 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0.000181242 \end{matrix}$$

Current ratio: {1., 1., 1., 0.400314, 1., 1., 0.109031, 0.890969, 1., 0.599686, 1., 1.}

Current/Vertex: {x, 1. x, 1. x, 1. x, 0.400314 x, 0.400314 x, 0.400314 x, 0.0436465 x, 0.356668 x, 0.356668 x, 0.599686 x, 0.599686 x, 0.599686 x}

Incoming currents: {x}

Outgoing currents: {0.0436465 x, 0.356668 x, 0.599686 x}

Total Incoming & Outgoing: x, 1. x

Figure 6: Results on a toy graph displayed above. Graphs are converted into directed graphs according to the order of vertex pairs in the edge list. Shown is the incidence matrix, resistance matrix (using arbitrary radii), and the set of current ratios for each edge. Given an input current of x, listed is the incoming and outgoing currents. Note Spielman uses edge currents, which are straightforward to obtain.

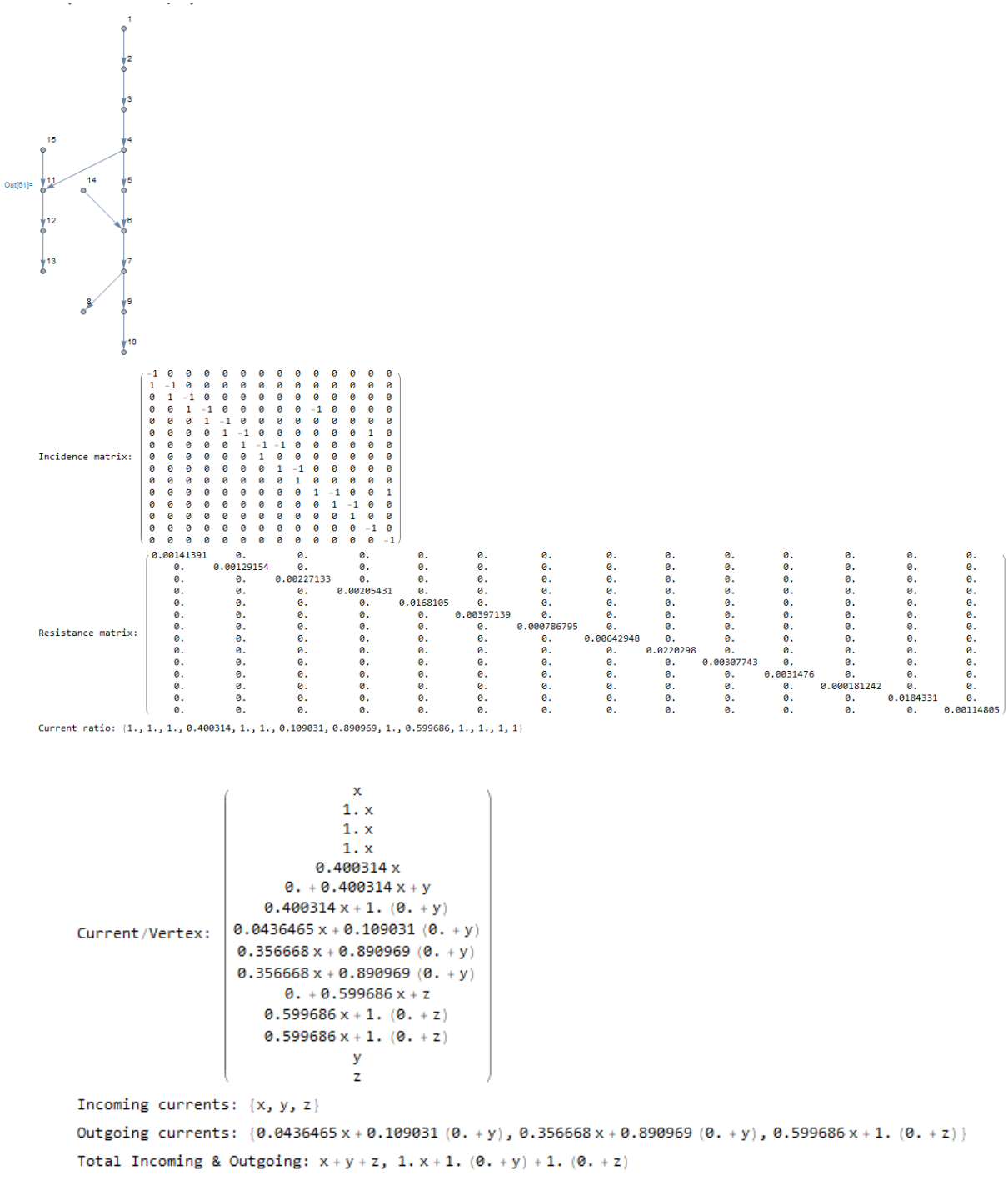


Figure 7: U and R on a graph with multiple roots, slightly modifying the toy graph previously shown. Below is the current per vertex where node 1, 14, and 15 receive x,y, and z current respectively. This method allows the user to specify an induced current, and in return they get a vector which will actually satisfy i = Inverse[R].Transpose[Ut].v.

By maintaining edge resistances in a diagonal matrix, and keeping track of voltage and current in vectors, I could maintain information about the network and apply an induced current, then calculate the outgoing current (or alternatively, voltage) at leaf nodes. The same thing can be computed on (2) and (3) simply by wiring neurons by the closest pair of vertices and treating the entire pathway as a single connected component, which is easy given that each neuron graph has 1 leaf node (Figure 7). I find this project exciting, and in particular would like to further consolidate between cable theory and Laplacians. Given the ability to compute the amount of time a signal takes to travel under reasonable assumptions, it would be interesting to be able to output vertex voltages, edge currents, and edge resistances at any given moment in time. In actuality, the current will flow through the network more slowly in some regions and quickly in others, and it will be interesting to assume the speed of electricity with no loss of current to the environment then see the network state at any given point in time, as opposed to computing vertex values in discrete steps whereby incoming edges arrive simultaneously. There is a lot of room for Laplacian-related analysis, and I think it will be a useful goal for applying linear algebra in the future.

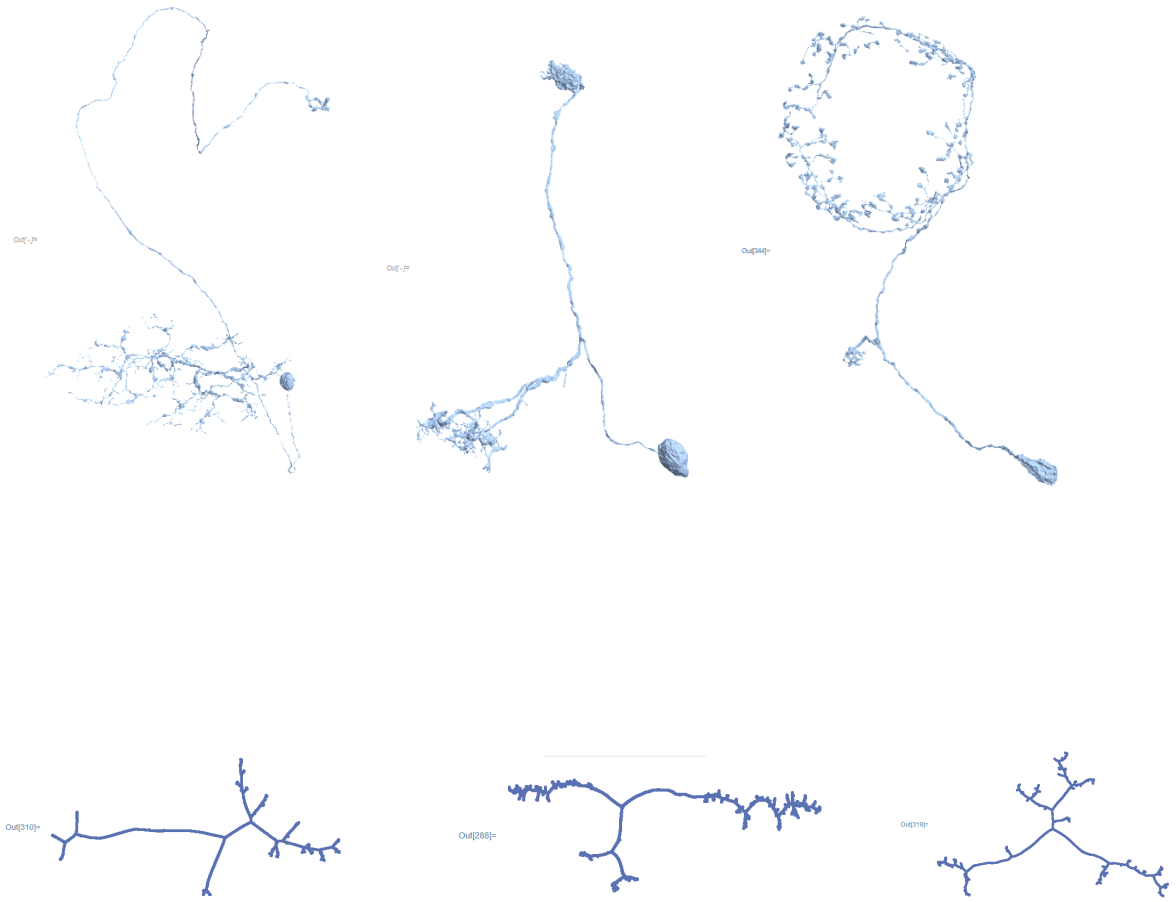


Figure 7: Mesh objects of three neurons (2) in the AVP. In sequence from left-right (a), as well as resulting graphs (vertices and edges) with ~15000 vertices and edges for neurons 1 and 2, and ~30000 for neuron 3. Number of root nodes from left to right: 3123, 3315, 6513. 1 leaf node/neuron.

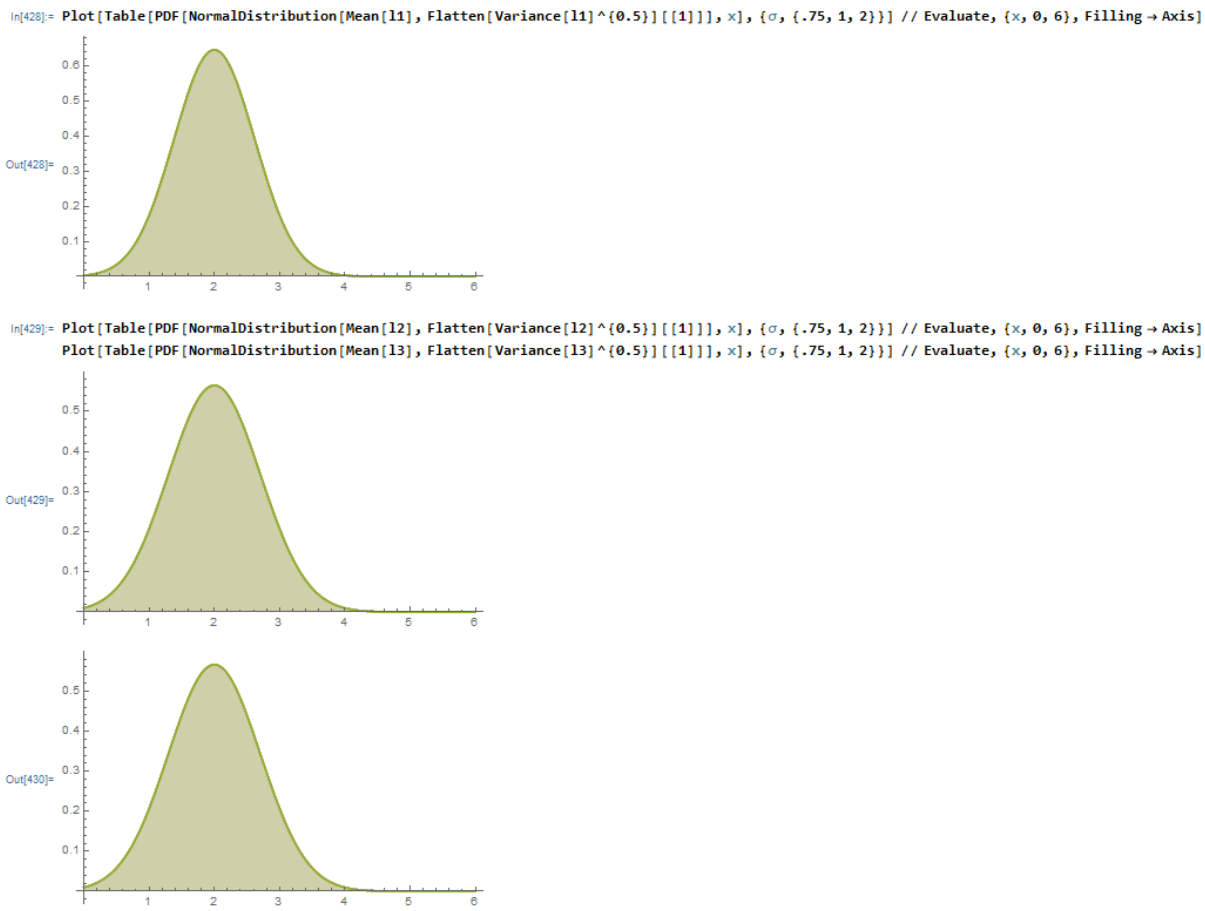


Figure 8: Distribution of degree of vertices in skeletons of three neurons in the AVP (2).

Tools Cited

Cloud Volume, fafbseg-py, FlyWire, Mathematica, Meshparty, NAVis, Skeletor

Works Cited

Omoto et al., 2017 J.J. Omoto, M.F. Keleş, B.M. Nguyen, C. Bolanos, J.K. Lovick, M.A. Frye, V. Hartenstein: Visual input to the Drosophila central complex by developmentally and functionally distinct neuronal populations, Curr. Biol., 27 (2017), pp. 1098-1110

Sheeba V, Gu H, Sharma VK, O'Dowd DK, Holmes TC. Circadian- and light-dependent regulation of resting membrane potential and spontaneous action potential firing of Drosophila circadian pacemaker neurons. *J Neurophysiol*. 2008. February;99(2):976–88. 10.1152/jn.00930.2007