# Neurolucida v1.4 to Matlab v6.0 Converter

February 14, 2004

Convert.m reads the ASCII data produced by Neurolucida and outputs arrays and vectors of the imaged cell's information for later use by the algorithum. Convert.m will parse the output file looking for certain flags to identify various parts of the cell and its associated coordinates. Here is an example of the data:

```
; V3 text file written for MicroBrightField products.
(Sections)

("CellBody"
(Color Blue)
(CellBody)
( 11.35 -2.98 0.00 0.10) ; 1, 1
( 9.64 -5.58 0.00 0.10) ; 1, 2
( 10.04 -8.46 0.00 0.10) ; 1, 3
( 13.55 -8.75 0.00 0.10) ; 1, 4
( 14.86 -8.56 0.00 0.10) ; 1, 5
( 15.06 -4.90 0.00 0.10) ; 1, 6
( 13.86 -2.98 0.00 0.10) ; 1, 7
( 11.65 -3.17 0.00 0.10) ; 1, 8
) ; End of contour

( (Color Yellow) ; [10,1]
(Dendrite)
( 14.66 -8.75 0.00 0.10) ; Root
( 15.86 -8.75 0.00 0.10) ; R, 1
( 16.57 -9.71 0.00 0.10) ; R, 2
( 19.88 -10.67 0.00 0.10) ; R, 3
( 20.58 -11.15 0.00 0.10) ; R, 4
(
( 20.58 -11.15 0.00 0.10) ; R-1, 1
( 20.58 -11.83 0.00 0.10) ; R-1, 2
( 20.58 -12.60 0.00 0.10) ; R-1, 3
( 20.58 -13.27 0.00 0.10) ; R-1, 4
( 20.58 -14.52 0.00 0.10) ; R-1, 5
Normal
—
( 20.58 -11.15 0.00 0.10) ; R-2, 1
( 20.88 -10.96 0.00 0.10) ; R-2, 2
( 21.89 -10.96 0.00 0.10) ; R-2, 3
```

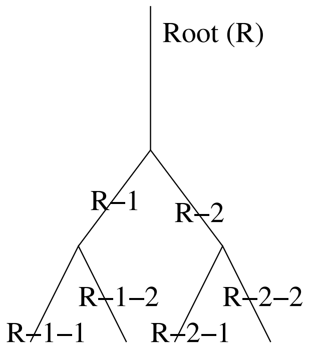

Figure 1: Neurolucida's labeling method for the dendritic tree

```
( 23.39 -10.67 0.00 0.10) ; R-2, 4
( 25.10 -10.67 0.00 0.10) ; R-2, 5
Normal
) ; End of split
) ; End of tree
```

Each file is broken up onto sections, i.e. soma, apical dendrite, basal dendtrite, etc. Within each of these sections are the points that make up the given structure. The first three points are the (x,y,z) coordinate points. The fourth number is the width of the given structure. Finally, each line is tagged to identify it. Every structure that branches from the soma receives the "Root" or "R" designation. Then its children are given a desgination of either "1" or "2". Then these two branches will give rise to more branches whose designation is either a "1" or "2" also. By this way, a branch from the tree can be identified by a string of 1's and 2's that identify it in the hierarchy as shown in **??**. It should be noted that Neurolucida is capable of producing trifurcating nodes. However, in most instances, a trifurcating node is usually a bifurcation node on closer inspection.

Data sets are ended with several different phrases depending on how you end your splits. You can end a split normally, or incompletely. Finally, the end of a tree is specified. Once the end of a tree is reached, and new trees are given the same numbering system. Therefore, branch numbering is unique only within its own tree and not to the entire set of trees.

Convert starts by opening the specified file for reading and opening other files for writing. In order to show the progress of convert, the branches are graphically shown as they are being processed. The coordinate information of the various points that make up the morphology of the cell are stored into files. These files are initialized along with the Neurolucida ASCII file. Convert also calls on a host of functions to break up some of the work that needs to be done. The first is getword. Getword simply takes an entire line from the ASCII file and takes any alphabetic letters and meshes them into one word it returns to convert. Getword is used to identify the flags written into the Neurolucida ASCII file. For example, in the above example of a file, for the line "( (Color Yellow) ; [10,1]", getword would return "ColorYellow". It takes any letter found on the line and concatenates them together. Convert then uses the combination of getting an entire line and sending it to getword looking for words like "cellbody", "dendrite", etc., to indicate what part of the cell the file is describing.

It now should be noted that the coding has gone through different stages of intended use. Originally, convert was written in C to convert Neurolucida files into Neuron .hoc files. Hoc is a variation of C created in the early 80's. In order to ensure that eventually, the converter could be written into a .hoc file for a module

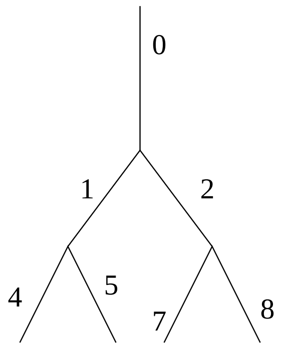

Figure 2: The new labels for the branches based off of the base 3 numbering system

in Neuron, only rudimentary commands were implemented. Some of those design aspects were carried over when writing the m-file. Then, it was decided to implement the converter within our own reverse engineering algorithms. Therefore, it is very possible to rewrite parts of convert using commands that would avoid some unnecessary Boolean switches.

Getting back to the data, once the cell body is found, a series of Boolean switches determine whether or not a character read from the file is a numeral, and if it is a numeral, then grouping numerals together to form numbers. The x and y coordinates are stored in two vectors which are fed into area.m once the cell body is finished. Area then calculates the are of the "circle". This number is doubled and multiplied by a constant to approximate the surface area of the soma. If we consider the soma to be roughly disc shaped surface, this is a good enough approximation.

Next, convert process the dendrites. The same algorithum is used to store the number for the dendrites with the addition of now also storing the radii of the points of the branches. Convert will now call upon getdendnum to identify the branch the point being processed belongs to. Getdendnum gets a line from the file and goes through it to find the branch id, i.e. "R-1-2-1". It then treats this id as if it were a base 3 number (121). For the root, it assigns a 0. It then converts this base 3 number into a base 10 number. Therefore, each branch is numbered with a unique number. The reasons for doing this are quite simple. It is easy to store such numbers as a vector. Second, it provides an easy way to reconstruct the relationship of the branches into a tree. For the tree above, once processed, it will now be Therefore, convert will generate a vector [0 1 2 4 5 7 8]. Given the number of any branch and the vector of numbers we can reconstruct the tree using the following equation: parent number = floor(child number/3). The branch numbers are stored in "numbers".

After each branch is processed, the x and y coordinates of its points are fed into getlength which calculates the distance of a given point from the starting point and then stored into lpoint. In addition, the length of each branch is kept in a running vector called len. It should be pointed out that truncated values are places into len. The reasons for the data in lpoint and truncating the values in len will be discussed later in the makeMK section. Finally, the radii of each point for each branch on a tree are stored in list. All the data are linked together similar to linked arrays. This is illustrated below Therefore, the first element of the vector number stores the value of the branch number and the first element of each other variable stores the information corresponding to that same branch.

A brief discussion is necessary about the use of cell arrays. The variables lpoint and list are cell arrays. This is necessary since matrices in matlab must have rows of the same length. However, in storing the values of the radii or the lengths of the points from the beginning of the branch, these data sets will vary depending on where the user clicks in Neurolucida. Some branches will have more points than others. Therefore, the data is stored as strings in cell arrays. Inorder to access these data sets, the cell array elements must be

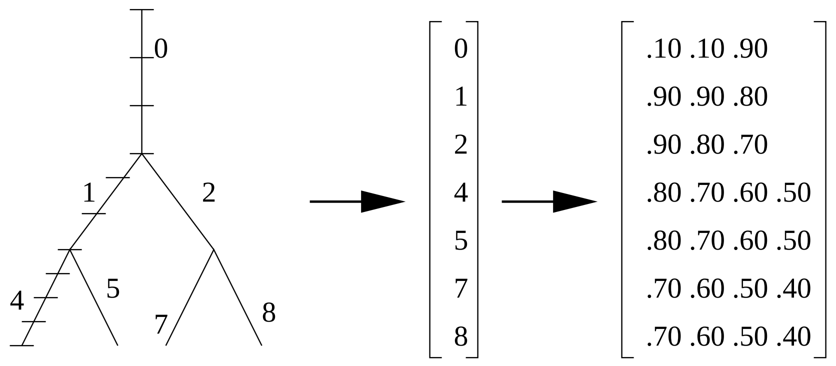

Figure 3: Illustrates how the information for the branches are stored in a set of matrix for matlab

accessed individually and converted back into numbers.

Once an entire tree is read in and stored, convert continues to move through the ASCII file until all trees are stored. The beginning of trees are marked by ”0”'s in the branch number vector, i.e. [0 1 2 4 5 0 1 2] describes two trees each begins at their respective zeros. This produces a set of data not quite ready to be sent to makeMK. The program makeMK needs the data first of all to be sorted in ascending order based on branch number. In Neurolucida, the ASCII file is written is the same way the user moves through the cell. It will go through one branch. When that branch forks, it goes down one and waits till all subsequent branches from that point are done before returning to the second branch. As a result, it produces a file where R-2 branch is not written on file till after the R-1-1-1 branch. The data sets are sorted together by their branch number. However, they are sorted by tree's only so that [0 1 4 5 2 7 8 0 1 2] becomes [0 1 2 4 5 7 8 0 1 2].

MakeMK2.m computes the mass and stiffness matrices for neuron. The program has three different layers. The most basic is makeMK which computes the mass and stiffness matrix for a wishbone (the node where three branches meet). MakeMK1, takes as its input an entire tree, cuts up those trees into wishbones sends them to makeMK for processing. This wishbone data is then pieced together into one matrix that describes the mass and stiffness of the entire tree. MakeMK2 takes in a cell, breaks it up into trees which it sends to makeMK1, and then reconstructs the mass and stiffness matrix for the entire cell. This is probably not a very efficient way to go about finding the mass and stiffness for the entire cell. But it should be noted that when the project was first started, it was decided to keep makeMK.m unchanged since that was verified to work. Therefore, makeMK1 and makeMK2 were programmed around makeMK.

It is extremely helpful to have ”Recovering the Passive Properties of Tapered Dendrites From Single and Dual Potential Recordings” by Steven J Cox near by as a reference to makeMK2 since the programming follows for the most part the algorithm in the paper.

In makeMK, we solve for the mass and stiffness matrix of a soma and a tapered fiber. Section 5 in Cox describes how to get these matrices from a soma with a tapered fiber. MakeMK requires the input of three branches of the wishbone and the space step. With this information, and using equations(5.4) and (5.5) in Cox, the mass and stiffness matrix can be calculated.

In makeMK1, the program is sent the space step, the radii of the points, the vector of branch numbers, the lengths of each branch, and distances of each point on each branch from the start of their own branch. In makeMK1, first, the cell array elements must be converted to numbers as the program uses them. Next, the radii must be extrapolated along each point of the space step. Since the user will click at uneven points along the dendrite, we must now smooth that out along regular intervals for makeMK. Therefore, all the points and their coordinates are interpolated. After they are interpolated, the number of points and the total number of points are calculated. This is important later for placement of the matrix from makeMK

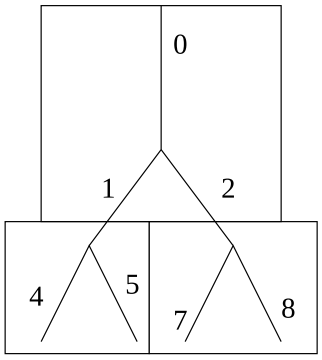

Figure 4: The breakdown of a tree by nodes. makeMK processes only a single node at a time.

into the new mass and stiffness matrices.

Once the makeMK matrix is computed, it has to be incorporated into the larger matrix. The following illustrates graphically, this process We can see from illustration, that for instances the wishbone where branches 7, 22, and 23 meet will be processed by makeMK as if it were branch 1,2, and 3. makeMK was designed to calculate the mass and stiffness matrices of this kind of cell. Therefore, in the matrix that is sent back, the information must be cut out and put into their proper location. MakeMK1 uses a branch's branch number to identify its location in the overall matrix. It excises the necessary data from the makeMK matrix and places them in their correct position in the makeMK1 matrix. Finally, in order to connect the different wishbones together, makeMK2 will break up the tree into halves. Looking at the tree, we can see that any branch can be either a child that is an ending, a parent to children and a child of a parent or a root. Therefore, in connecting the wishbones together within the overall matrix, makeMK2 takes halves of branches. Therefore, most branches, which are both a parent and a child, are processed twice - once as a parent and once as a child. Each time a branch is processed, half of the corresponding location in the matrix is filled in.

MakeMK2 now takes in as input an entire cell, breaks up the cell into trees, passes each tree to makeMK1, and then reconstructs the mass and stiffness matrix for the entire cell. MakeMK2 uses the list of branch numbers and identifies trees by "0"'s. Once a tree has been processed, it calculates the location for that tree within the final matrix. Its last job is then to connect all the trees within the matrix together. The first tree and its root are considered the root of roots. Every root is connected to the soma and the soma is isopotential. Therefore, the first point on every root should be the same. The final matrix reflects this property by making every root share the same properties for the first point.