%G%

Documentation for "ntscable" version %I%

by:

JC Wathey
Computational Neurobiology Laboratory
Salk Institute
PO Box 85800
San Diego CA 92138

(619) 453-4100 ext 565

WHAT THE PROGRAM DOES

This program translates a digitized morphological description of a neuron into files which can be used directly by the simulation programs "CABLE" and "NEURON", by Michael Hines of Duke University (Hines, 1989). In its original incarnation, ntscable could only read data files in the syntax of the Neuron Tracing System (Eutectic Electronics) and could only produce output in CABLE syntax; hence the name "ntscable". The latest version can also read data files generated by the digitizing systems of Rodney Douglas and Rocky Nevin. All of these digitizing systems are similar in concept. Since the Eutectic NTS system is the most widely used, it is described in some detail under INPUT FILE SYNTAX, below. Special considerations for using Douglas and Nevin data files are also described there.

The ASCII data files generated by these digitizing systems list each point as a separate line of text, including coordinates of the point and the thickness (i.e., diameter) of the neurite at that point. While reading the file, ntscable constructs in dynamic memory a complete representation of the neuron, exactly as it was digitized. It then divides the branches into "segments" as defined in the documentation for CABLE. The total number of such segments can be specified by the user and may be less than, greater than or exactly equal to the number of digitized segments in the input file. Diameters of the segments are calculated from the thickness values read from the input file. Regardless of the number of segments used in the output file, the diameter calculation algorithm guarantees that the membrane area of each segment is exactly the same (within roundoff error, which is negligible in this case) as the area of the corresponding length of neurite in the original digitized neuron. This means, of course, that all the translated neurites combined have the same total membrane area as in the original digitized neuron. The translation algorithm is explained in greater detail under TRANSLATION ALGORITHM, below.

If the translation is successful, ntscable produces an output file which contains a "geometry()" subroutine in either CABLE or NEURON syntax. This can be used by CABLE or NEURON with little or no modification by the user. The program can also create files which can be used to verify the correctness of the translation and to make schematic diagrams of the structure of the neuron.

INPUT FILE SYNTAX

ntscable determines the syntax of the input file by searching for a standard "header" line near the beginning of the file. Each digitizing system uses a unique header line, as listed in the file read.c. If the program cannot find any of these in the input file, it gives an error message and aborts.

I. Eutectic NTS syntax

NTS was designed for use with serial sections of stained neurons. Portions of neurites are traced from each section, and the whole neuron is reconstructed by aligning the data from adjacent sections. This process of alignment is called "merging" in the NTS manual. Your morphological data must be completely merged before you try to translate it with ntscable. In our lab we have only used data from wholemounts of hippocampal slices (prepared and digitized by David Amaral and colleagues). Normally the entire neuron is contained within the slice, so merging data from serial sections is unnecessary.

NTS stores 3 spatial coordinates (x,y,z) and a thickness value (i.e., neurite diameter) for each digitized data point. It also categorizes each data point as belonging to one of 21 different "point types". You should consult the NTS manual for a detailed explanation, but the point types are listed and briefly described below:

MTO	middle tree origin
TTO	top tree origin
BTO	bottom tree origin
CP	continuation point
FS	fiber swelling
SB	spine base
BP	branch point
NE	natural end
ES	end swelling
MAE	middle artificial end
TAE	top artificial end
BAE	bottom artificial end
SOS	soma outline start
SCP	soma continuation point
SOE	soma outline end
OS	outline start
OCP	outline continuation point
OE	outline end
DS	dot start
DCP	dot continuation point
DE	dot end

Points of type SOS, SCP and SOE define the traced outline of the cell body. The root of a primary neurite (i.e., one that is attached to the soma) is identified by MTO. Most points along the length of the neurite are of type CP; FS and SB are essentially the same as CP, but also specify the location of a varicosity or spine, respectively. A bifurcation of the neurite is indicated by BP; a split into 3 or more branches from the same point can be indicated by 2 or more successive points of type BP, all with the same x, y and z coordinates. The normal end of a fiber is indicated by NE or, if it ends in a presynaptic bouton, by ES. If the fiber cannot be traced to its end because of artifacts, incomplete filling, or because it is severed at one face of the section, then the end is indicated by MAE, TAE or BAE. TTO and BTO indicate the beginning of a severed neurite at the top or bottom face of the section, respectively; a fully merged neuron will not contain points of these types. Points of type OS, OCP, OE, DS, DCP, and DE are used to digitize histological features other than the neuron (laminar boundaries, locations of other nearby cell bodies, etc.).

An NTS data file lists each point as a separate line of text, including a point number, point type, coordinates and thickness (diameter). See example.nts for a simplified example of such a file. When ntscable reads the file, it first skips over any lines which do not match the standard header line. This allows you to insert any number of comment lines at the top of the file. Any such comments will be included as part of the automatically generated comment at the top of the translated file.

Brief comments may also be appended to the end of a data line, as shown in example.nts, but these will not appear in the translated file.

The current version of ntscable does not recognize points of type TTO and BTO. If your file contains these, it means your data are not fully merged. You must complete the merger before translation. Points of type OS, OCP, OE, DS, DCP, and DE are not part of the neuron; ntscable ignores these. Points of type FS and SB are treated the same as CP. Similarly, ES is treated exactly the same as NE. I have plans for a future version of ntscable which will construct "prefab" spines at every SB point, but for the present spines must be added manually after translation.

If the file was constructed by merging several files from serial sections, then it may contain multiple separate tracings of the soma (one for each section that contained part of the soma). If this is the case, ntscable will use only the largest soma outline (i.e., the one that encloses the largest area) for calculating the length and diameter of the soma. The details of this calculation are explained under TRANSLATION ALGORITHM, below.

II. Nevin syntax

The Nevin syntax (originated in John Miller's lab at UC Berkeley) differs from the Eutectic syntax in two respects: (1) it uses fewer point types, and (2) the soma is digitizing not by tracing its outline but instead by digitizing it as a "branch".

Nevin point types		Eutectic equivalent
C	continuation point	CP
S	soma continuation point	CP
B	branch point	BP

T	termination point	NE
F	fiducial point	DS,OS,DCP,OCP,DE,OE
f	fiducial point	DS,OS,DCP,OCP,DE,OE
c	fiducial point	DS,OS,DCP,OCP,DE,OE

The soma is digitized by moving the cursor along one arbitrarily chosen direction through the center of the soma and storing the diameter (measured perpendicular to the direction of cursor movement) at frequent intervals. This is identical to the procedure used to digitize a dendritic branch, except that the points are tagged with point type 'S', rather than 'C'.

The Nevin files I have seen are of unipolar invertebrate neurons. In most of them the file begins with the soma points, and the single primary neurite then starts with a point of type 'C'. In such cases the soma was digitized along a line that ends at the root of the primary neurite. Alternatively, the file may begin with the neurites and end with the soma points. In yet another style the soma is digitized along a line that does not end at the root of the neurite. Instead, the root of the neurite is indicated as a point of type 'B' in the middle of the list of 'S' points, which is terminated with a point of type 'T'. ntscable will correctly translate files organized in any of these ways. See TRANSLATION ALGORITHM, below, for an explanation of how the soma length and diameter are calculated from the list of soma points.

The fiducial points are analogous to Eutectic "dot" and "outline" points and are ignored by ntscable. The lower case 'f' and 'c' points sometimes (but not always) occur at the beginning and end, respectively, of a list of 'F' points. It appears that sometimes a lowercase 'f' point is in reality a neurite point (judging from the particular values of x, y, z and diameter). I recommend checking these points carefully before translation. If any of them was meant to be a data point, its point type must be changed before translation.

III. Douglas syntax

Rodney Douglas' digitizing system produces data files in two different formats, called "2D" and "3D". The 3D file contains a complete, 3-dimensional representation of the neuron.
The 2D file is derived from the 3D file and contains only branch length, diameter and connectivity information. If both file types are available for a given cell, then the 3D file is preferable. ntscable is, however, capable of translating both formats.

Douglas 3D syntax

The Douglas 3D format is similar to the Eutectic format, with a point type identifier and an x, y, z and diameter value for each digitized point.

Douglas 3D point types		Eutectic equivalent
mto	middle tree origin	MTO

dcp	continuation point	CP
bp	branch point	BP
ne	natural end	NE
mae	middle artificial end	MAE
tae	top artificial end	TAE
bae	bottom artificial end	BAE
mto	soma outline start	SOS
scp	soma continuation point	SCP
mae	soma outline end	SOE

Note that mto is used both for the origin of a neurite and for the beginning of the list of soma outline points; similarly mae is used both for an "artificial" end of a neurite branch and for the end of the list of soma points. The exact meaning of mto and mae is determined from the context in which they occur. The coordinates and diameters in the Douglas 3D files are in units of microns * 10 (i.e., the numbers in the file must be divided by 10 to yield microns).

Douglas 2D syntax

The 2D file is evidently derived from the 3D file by a translation program and is a reduced version of the 3D file. Rather than containing a complete list of all digitized points, 2D files contain only one point for each segment of neurite between adjacent branch points. For each of these points there is a point type identifier and a segment length and diameter. The diameter is the average of all the original digitized diameters which lay within the segment. The length is the sum of the distances between adjacent points that lay within the segment. Since there is only one point for each interval between branch points, there is no need for continuation points. The soma is listed as a single point with a corresponding length and diameter. There are no x, y, or z coordinates in the file. The point types in the 2D Douglas syntax are:

Douglas 2D point types		Eutectic equivalent
som	soma	SOS, SCP, SOE
bp	branch point	BP
ne	natural end	NE
mae	middle artificial end	MAE
tae	top artificial end	TAE
bae	bottom artificial end	BAE

As with Eutectic files, short comments may be appended to the end of a line in a Douglas 2D file. When adding such comments to a Douglas 2D file, be sure there is at least one space character between the original end of the line and the beginning of the comment.

When ntscable translates a Douglas 2D file, it synthesizes lists of x, y, and z coordinates for the soma and dendrites. The x coordinates are calculated by summing the appropriate segment lengths. The y coordinate is constant for all segments in a given branch, but differs from one branch to the next. The z coordinates are all 0. The synthesized coordinates are used for the lists of 3D points in the translation to NEURON syntax. If the neuron were drawn using these coordinates, the resulting picture would be essentially the same as the schematic diagram produce by ntscable (see SCHEMATIC DIAGRAMS, below).

TRANSLATION ALGORITHM (CABLE or MULTI syntax)

Each digitized data point from the input file is represented by ntscable as a TREE_POINT (a data structure defined in ntscable.h). Each TREE_POINT has associated with it, among other things, a neurite diameter and a length which gives the distance to the adjacent proximal TREE_POINT. The length may be zero (for example, in the case of the most proximal TREE_POINT of a primary dendrite), but normally it is positive. The neuronal membrane between two adjacent TREE_POINTs should be thought of as a cone truncated at the two TREE_POINTs by planes perpendicular to the axis of the cone. The diameters of the cut-off ends of the cone are, of course, the diameters associated with the corresponding TREE_POINTS, and the length of the conical segment is the length associated with the more distal of the two TREE_POINTs.

Given this representation, the simplest way to translate the data to CABLE segments would be to create one cylindrical CABLE segment for each pair of adjacent TREE_POINTs. The length of the segment would be the distance between the TREE_POINTs. The diameter would be the average of the diameters at the two TREE_POINTs. Averaging the diameters in this way guarantees that the membrane area of the cylindrical segment is the same as the area of the conical piece of membrane between the two TREE_POINTs.

Unfortunately there are some serious problems with this simple algorithm. The person who digitized the cell may have been forced to save many points at short intervals, just to follow the twists and turns of the dendrites. As a result, the data file may contain far more points than are required for an adequate compartmental simulation. In such a case the simple algorithm would produce a CABLE simulation that ran slower and used more memory than necessary; in extreme cases the simulation might not even fit into the computer's memory. At the other extreme, the distance between adjacent TREE_POINTS might be too large (this is typical of the reduced Douglas syntax). In such a case the corresponding CABLE segments would be too large to provide adequate numerical accuracy (in the spatial dimension) in the simulation. Furthermore, not all simulations of a given cell require the same degree of spatial accuracy. For example, a simulation of a small current injection into the soma requires less detail in the segmentation of the dendrites than does a simulation of a complex pattern of synaptic input scattered across the entire dendritic tree.

What's needed is an algorithm which lets the user choose the number of segments in the translation, and which can combine many TREE_POINTs into one segment (where possible) or divide the region between two TREE_POINTs into many segments (where necessary). In all cases, regardless of how the neurites are divided into segments, the algorithm should calculate the segment diameters such that the area of each segment is exactly equal to the area of the corresponding region of the original digitized neurite. The translation algorithm used by ntscable does all of these things. The user controls the spatial accuracy of the translation by specifying the maximum allowable segment length and by selecting one of three "accuracy levels". The accuracy level specifies the method to be used for grouping and dividing TREE_POINTS into segments, as explained in detail below. The user can specify the maximum allowable segment length (max_dx) in any of three different ways, as explained in detail in the section HOW TO RUN ntscable.

Most of the work of translation is done by the recursive function get_cable_branch() in the file xlate.c. This routine translates each branch in small chunks. Each such chunk is called a "translation interval", and may contain one, a few, or all of the TREE_POINTS in the branch being translated, depending on which of the accuracy levels defined below was chosen by the user. The ends of the translation interval always coincide exactly with TREE_POINTs. In all cases translation consists of dividing the translation interval into the smallest possible number of segments of equal length dx such that dx <= max_dx. The diameter of each segment is then calculated such that membrane area is conserved, as described above.

accuracy level	meaning
2	Specifies the most accurate method for dividing the branch into CABLE segments. The translation interval is the interval between each pair of adjacent TREE_POINTs. Note that if the largest distance between any two adjacent TREE_POINTS is less than or equal to max_dx, then this becomes the "simplest algorithm" described above.
1	The translation interval is the interval between adjacent branch points, if the branch gives off higher-order branches; otherwise it is the entire branch.
0	Specifies the least accurate method for dividing the branch into CABLE segments. The translation interval comprises the entire branch. If the branch gives off higher-order branches, then each branch point is moved to the distal end of the CABLE segment containing its true location. This adjustment of branch point locations is necessary because, in the CABLE simulation, dendritic branch points must coincide with the end of a segment. Note that if the number of segments in a branch is small, this adjustment could introduce significant errors in the structure of the simulated neuron. With accuracy levels 1 and 2, the branch point locations are exactly preserved

in translation, regardless of the number of segments used.

By default the program uses accuracy level 1, which is the best choice for most purposes. Accuracy level 0 is useful for quick and dirty simulations of highly branched neurons. It is also useful any time you need all the segments to be as nearly equal in length as possible. One such situation I have encountered involved placing synapses at random on the dendritic tree. A random number generator was used to pick the segment index numbers of those segments which would receive synapses. The segments had to be nearly equal in length to avoid bias in the spatial distribution of synapses. Accuracy level 2 guarantees a "literal" translation of the original digitized data: both membrane area and dendrite diameter are equal at corresponding points on the neurites of the original and the translation. For most neurons accuracy level 2 results in an unnecessarily large number of segments in the simulation. It might be necessary, however, for simulations in which the results were sensitive to changes in dendrite diameter over short distances.

Regardless of the input file syntax, the length and diameter of the soma are calculated from a common internal representation of the soma shape. This representation, which is called the soma_3D_list, consists of a list of x,y,z,diameter points taken at frequent intervals along one path through the soma. This is analogous to the way branches are digitized and is identical to the way the soma is digitized in the Nevin syntax. When translating Nevin files, ntscable calculates the soma length and diameter directly from the original digitized soma points. For the other input file formats, ntscable must first synthesize a soma_3D_list before the length and diameter of the soma can be calculated.

For those formats in which the soma is digitized by tracing its outline, ntscable constructs the soma_3D_list along a path roughly parallel to the x-axis through the center of the soma outline. The points are ordered from left to right and occur at the x values found in the list of soma outline points. For each point in the soma_3D_list, the x, y, and z coordinates are calculated as the midpoint of the chord, perpendicular to the x-axis, that spans the soma outline and contains at least one soma outline point at one end. The corresponding diameter value is the length of that chord. The soma length is calculated as the arc length along the soma_3D_list. The soma diameter is calculated as the diameter of the cylinder having the same surface area (not including the cylinder end faces) as all the conical surface segments defined by the soma_3D_list.

For both the Eutectic and Douglas 3D formats, the soma outline is digitized roughly parallel to the xy plane. If there are large, spurious fluctuations in the z-coordinates of the soma outline, then the soma length and diameter, as calculated from the soma_3D_list, will be in error. The user may eliminate such errors by running ntscable with the -z option: this forces ntscable to replace the soma outline z coordinates with their average value.

For the Douglas 2D syntax, ntscable constructs a soma_3D_list

from the soma length and diameter read from the input file.
The coordinates of these synthesized points are:

x	y	z	diameter
0.0	D/2	0.0	D
L	D/2	0.0	D

where D and L are the original diameter and length values read from the input file. It may seem pointless to construct this soma_3D_list from a diameter and length, just so that the same diameter and length can be calculated from the soma_3D_list. For translations to CABLE syntax, where only diameter and length are used in the output file, it is indeed pointless. For translations to NEURON syntax, however, the shape of the soma must be specified in the output file as a list of x,y,z, diameter points, rather than as a length and diameter (see below).

In CABLE syntax the translated soma is always a single segment which is given the segment index number 0. It is modeled as a cylinder of length dx[0] and diameter diam[0]. As in the case of the neuritic segments, the ends of the cylinder are not included in the calculation of surface area. The area is therefore PI*dx*diam, which turns out to be the same as the area of a sphere of this diameter if dx[0] = diam[0].

TRANSLATION ALGORITHM (NEURON syntax)

The translation to NEURON syntax is much simpler. The output file includes a "create" statement which allocates 1 section each for the soma and all branches. The number of segments is determined as in the translation to CABLE syntax with accuracy level 0. The segment lengths and diameters are not explicitly specified in the translation. Instead the translation provides for each section a list of x,y,z and diameter data from which the NEURON simulator calculates segment diameters and total section length.

Important: The NEURON-syntax output files produced by ntscable are compatible only with versions of NEURON dated 9 August 91 or later.

SCHEMATIC DIAGRAMS

The program can also create schematic diagrams of the original digitized data or of the translated neuron. The diagrams are created as files with names of the form
inN.gph and outM.fig, where N is a number identifying a single primary neurite and M is the segment number of the most proximal segment in the file. Each of these is an ASCII data file which produces a schematic diagram of the branching pattern of the neuron, either as it was originally digitized (in* files) or after translation (out* files). In diagrams of this form, the horizontal line segments represent branches and are drawn to scale. The vertical line segments serve only to show connectivity; they do not represent portions of the physical tree. The Eutectic software can

also produce schematic diagrams analogous to these, although in a somewhat different style.

These diagrams are extremely useful (in some cases essential) for simulations involving many synaptic inputs or a diversity of voltage sensitive conductances in large, complex dendrites. The diagrams serve as "road maps" to help you find your way around the neuron. The .fig diagrams are more detailed than the .gph diagrams, can be edited with a graphical drawing program and can be used for publication-quality hardcopy. The .gph diagrams have the advantage that they can be displayed and printed on a wider variety of terminals and printers than can the .fig files.

The actual plotting of a .gph file is done via the standard UNIX graph(1) and plot(1) programs. The shell script "schematic" contains an example of a typical plotting command for use in a Sun Tektool or in an X-windows xterm in tek mode. This shell script can be easily modified to send the output to other devices supported by the Unix graph command. Each branch in the diagram is labeled with the corresponding data point numbers (for the in* files) or the corresponding CABLE segment index numbers (for the out* files).

The .fig files are in the syntax of the public domain program "fig". fig is a mouse-driven drawing program analogous to (but more limited than) MacDraw on the Apple Macintosh. Schematic diagrams in fig syntax can be edited on the screen using a special version of fig modified by Mike Hines and by myself. At present this version is available only for Sun workstations. Graphics output from CABLE can easily be combined with schematic diagrams, since CABLE can also produce output in fig syntax. The .fig files can be translated to postscript using the program f2ps. Note that f2ps can be used independently of fig (and therefore on computers other than Sun workstations) for getting hardcopy of schematic diagrams. Unlike the .gph diagrams, the schematic diagrams in fig format show the diameters of the neurites (as the widths of the horizontal lines) and the individual segments are delineated by narrow gaps of white space. Each branch is labeled with the CABLE segment index numbers of the segments it contains. The size of the characters in the label is determined by the vertical separation between branches. If the diagram contains many branches, the label characters may be too small to read. If this happens, ntscable will automatically enlarge the diagram by an integer factor large enough to make the labels legible. The enlargement is done by dividing the diagram into m-squared rectangular panels of equal size, where m is the magnification factor; a separate fig file is created for each panel. Each of these can then be printed on a separate page at the same size as the entire diagram would be if m=1. The user must assemble the panels to see the entire diagram. If a panel includes nothing but white space from the original diagram, then no fig file will be created for it. The panels are numbered 1 to m-squared; the panel number appears in the filename of the fig file and in the printout of the panel itself. The panels fit together as shown in the example below (m=3):

```
           -----------------
          |     |     |     |
          |     |     |     |
          |  1  |  2  |  3  |
          |     |     |     |
          |     |     |     |
           -----------------
          |     |     |     |
          |     |     |     |
          |  4  |  5  |  6  |
          |     |     |     |
          |     |     |     |
           -----------------
          |     |     |     |
          |     |     |     |
          |  7  |  8  |  9  |
          |     |     |     |
          |     |     |     |
           -----------------
```

The user may enlarge any diagram in this way, to any desired size, via the -M command line option. Similarly, the automatic enlargement just described can be suppressed via -M 1.

Note that any pre-existing in*.gph or out*.fig files will be deleted each time ntscable creates new diagrams; the files must be renamed or moved to a safe place if you want to keep them.

PROGRAM VERIFICATION

If executed with the -v option, ntscable creates the file "verify.txt". This file contains a text listing of the internal data structures used by ntscable to represent the digitized neuron. Its purpose is to verify the validity of the translation by showing in explicit detail the correspondence between the digitized data points from the input file and the CABLE segments in the output file. To understand fully the contents of verify.txt, you should first become familiar with the relevant data structures (defined in ntscable.h) and with the translation algorithm (in xlate.c). Unless you have reason to suspect a bug in the program, you can usually skip the creation of verify.txt.

HOW TO RUN ntscable

ntscable conforms to the "Standard Command Format" for UNIX programs [Thomas et al., 1986]. This means:

(1) All command line options and option arguments appear before any file name arguments on the command line.

(2) Each option contains a single dash followed by the option letter.

(3) An option may use an option argument which is a number, a word or a filename that further defines the action of the option.

(4) Options that do not require option arguments can be grouped together into a single program argument or specified as individual options, each prefaced with a

```
       dash.

    (5)  Option arguments are themselves NOT optional.  An
       option argument must be specified immediately following
       the option it modifies; white space between the option
       and its argument is permissible but not required.

    (6)  The '-' argument by itself is considered a file name
         that specifies the standard input.
```

In actuality ntscable tolerates a fair amount of sloppiness in the command line. For example, file names and options can be intermingled in almost any order and ntscable will still parse everything correctly. It does, however, require that an input file name (or '-') be specified, and that the input file name precede the output file name if both are specified.

```
usage:
      %     ntscable [options] [option_args] input_file [output_file]

options:
  -f                         create diagrams in fig syntax
  -g                         create diagrams in graph syntax
  -i                         create diagram of input file
  -o                         create diagram of output file
  -v                         create verify.txt
  -z                         average soma outline z coordinates
  -x <output_syntax>         neuron, cable or multi
  -a <translation_accuracy>  0, 1 or 2; 0=least, 2=most accurate
  -n <num_segments>          minimum number of segments
  -l <segment_length>        maximum segment length in microns
  -d <length_factor>         max seg length as multiple of diam

  -M <magnification>         magnify diagram by integer factor
  -O <feature_to_omit>       omit labels or diams from diagram
  -B <branch_diameters>     \
  -D <description_size>      |--Each of these options changes
  -G <gap_between_segments>  |  the size of one feature of the
  -L <label_size>            |  schematic diagram.  The argu-
  -S <scalebar_length>       |  ment is a floating point factor
  -T <max_tag_size>          |  by which the default size is
  -W <connection_width>      |  multiplied.
  -Z <tag_separation>       /
```

where input_file is the name of an existing ascii data file to be translated, and output_file is the name of the output file to be created. If output_file is omitted, the translation is written to the standard output. If '-' is given as the input filename, then the input to be translated is read from the standard input.

Typing the command "ntscable" with nothing else on the command line produces the "usage" and "options" messages shown above.

The -z option specifies that the z coordinates of the soma outline points are to be replaced by their average value.

The argument translation_accuracy (= 0, 1 or 2) of the -a option specifies the accuracy level as defined in TRANSLATION ALGORITHM. If the -a option is not used, then the accuracy level is set to 1 by default for CABLE translations. For translations to NEURON syntax, accuracy level is always 0, and any request on the command line for a higher accuracy level is ignored.

The options -n, -l and -d provide three alternative ways of specifying the maximum allowable segment length (max_dx), as follows:

-l The argument segment_length specifies max_dx explicitly, in microns.

-n The argument num_segments specifies the minimum number of CABLE segments into which the translated neurites are to be divided.

-d The argument length_factor specifies max_dx as a multiple of the average diameter of the translation interval.

For the -n option, the program calculates a max_dx which, when divided into the total length of all the neurites, would give exactly the specified number of segments. If the cell has more than one branch, it will be impossible to divide the cell into exactly that number of segments, because the lengths of the individual branches will not be integer multiples of max_dx. Instead, each branch is divided into the smallest possible number of equal-length segments shorter than max_dx. The total number of segments actually used is therefore larger than the number requested by the user. If the number of segments used is greater than the maximum number that CABLE can handle, then ntscable gives a warning message to this effect.

When max_dx is specified via -l or -n, its value is constant during translation and applies to the entire neuron. In contrast, when max_dx is specified via the -d option, its value changes during translation and is determined by the average diameter of the current translation interval. This lets you take advantage of the fact that a large diameter neurite is more nearly isopotential over a longer distance than is a smaller diameter neurite. For example, if the argument length_factor is 10, then a portion of the dendrite with an average diameter of 5 um will be divided into segments <= 50 um in length, while a 1 um diameter branch of the same neuron will be divided into segments <= 10 um in length.

The -l, -n and -d options may be used in any combination. The program will calculate max_dx by all of the specified methods, and the smallest value obtained will be used. If none of these options is used on the command line, then by default the program calculates max_dx as if the user had typed -n 1. This default, along with the default accuracy level of 1, will produce a translation in which branch lengths, areas and branch point locations are exactly preserved using the minimum number of segments.

The -i and -o options indicate that schematic diagrams are to be created for the input or output, respectively. The -f and -g options specify that these diagrams are to be created in fig or graph syntax, respectively. If either -i or -o or both are specified, but neither -f nor -g is specified, then the program will create diagrams both in fig and graph syntax. If either -f or -g or both are specified, but neither -i nor -o is specified, then the program will create diagrams for both the input and the output. If any diagrams are created, all old diagrams in the current directory are first deleted. If none of the -f, -g, -i or -o options is specified, then no diagrams are created, and no old diagram files are deleted.

The -x option selects the syntax to be used for the output file. The argument is the name of the simulator to be used (at present only "neuron", "cable" and "multi" are valid; "cable" is the default). Only the first letter of the argument is checked, and letter case is ignored, so -x n, -x N, -x c, -x C, -x m, and -x M are all valid. Specifying the "multi" argument causes the program to write the dx and diam variables as two-dimensional arrays for use with MULTI (the multicellular version of CABLE). The left (higher order) index is the neuron type index. In the output file, ntscable uses the HOC variable "type" for the value of this index. The user must assign an appropriate value to "type" by manually editing the output file. The "neuron" argument specifies translation to the syntax of NEURON (Michael Hines' rewritten version of CABLE). These files are compatible only with versions of NEURON dated 9 August 91 or later.

The remaining options are for customizing the appearance of the fig-format schematic diagrams. Typing any of these on the command line also activates the -f option, whether or not the user types it explicitly.

The -M option enlarges the diagram by an integer factor which the user supplies as the argument to this option. Specifying a magnification of 0 causes the program to use the smallest possible magnification that will produce legible branch labels. If not specified on the command line, the magnification will be determined as if -M 0 had been specified. See SCHEMATIC DIAGRAMS, above, for more details.

The -O option specifies a feature of the diagram to be omitted. At present the only features that can be omitted are "labels" and "diameters". Typing "-O labels" will delete the branch labels (i.e., segment numbers) from the diagram; "-O diameters" causes all branches to be drawn with the same line thickness (i.e., diameter information is omitted). Only the first letter of the argument is checked, and letter case is ignored, so -O l, -O L, -O d, and -O D are all valid.

Each of the remaining options changes the size of one feature of the schematic diagram. The argument is a floating point factor by which the default size is multiplied.

The -B option changes the scale at which branch diameters and the soma diameter are drawn, relative to the separation between branches. The diameter dimension of the scale bar is adjusted so that the calibration remains correct. For example, if a particular branch has a diameter equal to 0.2 times the separation between branches, then, after the diagram is redrawn with -B 3, the diameter of that branch will be 0.6 times the separation between branches. A subtle point: changing the diameters will cause the soma to occupy relatively more or less of the vertical extent of the diagram than in the default case. This in turn causes the vertical separation between branches to shrink or enlarge slightly, since the entire diagram is always scaled to have the same absolute vertical extent on the page. For this reason using, for example, -B 3 will not make the diameters exactly 3 times larger in absolute size. They will be exactly 3 times larger relative to the branch separation, but, since the branch separation becomes slightly smaller, the absolute enlargement of the branch diameters will be slightly less than 3.

The -D option changes the size of the descriptive text at the bottom of the diagram.

The -G option changes the size of the gap between the ends of adjoining segments. The argument 0.0 may be given, in which case there will be no gap between segments.

The -L option changes the size of the branch label text.

The -S option changes the length of the scale bar.

The -W option changes the width of the vertical dashed connection lines.

The -T option changes the scale at which segment tags are drawn. Segment tags are simple geometrical figures (e.g., filled or open circles) which indicate something special about the segment (e.g., the presence of a synapse). Note: At this writing the segment tagging feature of ntscable is not fully implemented, so this and the next option have no effect.

The -Z option changes the offset used when multiple, overlapping tags are drawn on the same segment (see Note above).

HOW TO COMPILE ntscable

The necessary files are:

```
    makefile
    ntscable.h
    ntscable.c cmdline.c diagrams.c write.c read.c
    read_nts.c read_rd2.c read_rd3.c read_rn.c verify.c xlate.c
```

The following header files from my "toolbox" are also required; for compilation these and my toolbox archive file (libjw.a) should be in the directory specified by $T in the makefile:

```
    tf.h jwmath.h sets.h filtools.h figtools.h sccstools.h scf.h
```

Use the 'make' command to compile.

WISH LIST

A future version should be able to create spines at Eutectic SB points.

At this writing ntscable cannot produce .gph files of the translation, nor .fig files of the input, nor any kind of schematic diagrams of output files in NEURON syntax. I hope to add these capabilities in the future.

The convention for naming schematic diagram files should be changed to something more sensible.

BUGS

There are no known bugs in ntscable.

REFERENCES

Hines M (1989) A program for simulation of nerve equations with branching geometries. Int J Biomed Comput 24:55-68

Thomas R, Rogers L, Yates J (1986) Advanced programmer's guide to Unix system V. Osborne McGraw-Hill: Berkeley