Documentation T2N

T2N is an extension of the TREES toolbox providing an interface between Matlab and the compartmental modeling environment NEURON.

T2N allows an easy generation of real and synthetic morphology single-cell and network models. All mechanisms, point processes (PPs), connections, morphologies and NEURON settings are directly set in a well-defined Matlab structure. For easy location-specific settings and manipulations T2N uses the TREES nodes which are automatically translated into NEURON sections and segments. Furthermore, T2N speeds up simulation time by automatic distributed computing of the simulations (e.g. a series of different current steps, or series of different ion channel blocks) and by using parallel Neuron (e.g. for large-scale networks). Finally, Matlab and TREES provide convenient ways to analyze the structured simulation output of T2N, thus making T2N a valuable tool for extensive *in silico* structure-function analyses.

# 1. Table of content

[1. Table of content 2](#_Toc495680515)

[2. Installation notes 4](#_Toc495680516)

[3. Tutorials 5](#_Toc495680517)

[4. The T2N main function 6](#_Toc495680518)

[4.1 Definition of simulation parameters 6](#_Toc495680519)

[4.2 The output of T2N 20](#_Toc495680520)

[5. Protocol functions 22](#_Toc495680521)

[5.1 t2n\_APprop – Calculating action potential properties 22](#_Toc495680522)

[5.2 t2n\_bAP – Perform a backpropagating AP simulation 23](#_Toc495680523)

[5.3 t2n\_blockchannel – Specifically reduce conductances 24](#_Toc495680524)

[5.4 t2n\_changemech – Manipulate mechanism parameters 25](#_Toc495680525)

[5.5 t2n\_currSteps – Applying current clamps 25](#_Toc495680526)

[5.6 t2n\_findCurr – find the current to hold a certain voltage 26](#_Toc495680527)

[5.7 t2n\_findFreq – Find current to fire certain # of APs 27](#_Toc495680528)

[5.8 t2n\_findSubthreshWeight – Find synaptic weights for cells to reach subthreshold potentials 28](#_Toc495680529)

[5.9 t2n\_FIplot – Plot the FI-curve of cells 29](#_Toc495680530)

[5.10 t2n\_IVplot – Plot the I-V relationship of cells 30](#_Toc495680531)

[5.11 t2n\_passTests – Get passive properties on each cell 31](#_Toc495680532)

[5.12 t2n\_plotbAP – Show backpropagating AP properties 32](#_Toc495680533)

[5.13 t2n\_plotCurrSteps – Plot current clamp simulations 33](#_Toc495680534)

[5.14 t2n\_plotdV – Make phase plots 34](#_Toc495680535)

[5.15 t2n\_plotVoltSteps – Plot voltage clamp simulations 35](#_Toc495680536)

[5.16 t2n\_Q10pas – Adjust passive model to temperature 35](#_Toc495680537)

[5.17 t2n\_resonance – Perform a resonance test 36](#_Toc495680538)

[5.18 t2n\_setionconcentration – Change ion concentrations according to literature 37](#_Toc495680539)

[5.19 t2n\_voltSteps – Applying voltage clamp protocols 38](#_Toc495680540)

[6. Auxiliary functions 39](#_Toc495680541)

[6.1 t2n\_as – Copy neuron setting to other instances 39](#_Toc495680542)

[6.2 t2n\_catName – concatenate name of simulation 40](#_Toc495680543)

[6.3 t2n\_catStruct – concatenate Matlab structures 40](#_Toc495680544)

[6.4 t2n\_checkinput – Sanity checks on neuron/tree struct 41](#_Toc495680545)

[6.5 t2n\_get – Return/Use recorded parameter 42](#_Toc495680546)

[6.6 t2n\_getGIDs – Return information for parallel NEURON 43](#_Toc495680547)

[6.7 t2n\_getMech – Visualize NEURON parameters values 44](#_Toc495680548)

[6.8 t2n\_getref – Get reference ID to neuron instance 44](#_Toc495680549)

[6.9 t2n\_getSynNum – Calculate synapse numbers 45](#_Toc495680550)

[6.10 t2n\_initModelfolders – Start a new model with T2N 45](#_Toc495680551)

[6.11 t2n\_makeNseg – Translate TREES nodes into NEURON sections/segments 46](#_Toc495680552)

[6.12 t2n\_plotChannel – Visualize ion channel kinetics 47](#_Toc495680553)

[6.13 t2n\_plotRaster – Generate a spike raster plot 48](#_Toc495680554)

[6.14 t2n\_plotTrees – Visualize morphologies for publication 48](#_Toc495680555)

[6.15 t2n\_pnode – Get all parameters at a specific node 49](#_Toc495680556)

[6.16 t2n\_poissonSpikeGen – Generate Poisson spike trains 50](#_Toc495680557)

[6.17 t2n\_renameNrnmech – rename compiled dll (Windows) 50](#_Toc495680558)

[6.18 t2n\_testComp – Generate simple compartment 51](#_Toc495680559)

[6.19 t2n\_writeTrees – Translate morphologies into hoc 51](#_Toc495680560)

# 2. Installation notes

T2N was developed on Windows but is compatible with Mac OS and Linux. T2N is not functional on Matlab 2009a or older and for some features it is recommended to use 2016a or higher. Similarly, we recommend using NEURON v7.5 or higher (<http://www.neuron.yale.edu/neuron/download>). Note, on Windows, please install NEURON with the “Set DOS environment” option enabled especially if the parallel NEURON feature should be used.

Please execute the “t2n\_runthisAfterUnzip.m” script file in the T2N main folder to have T2N and all subdirectories added to the Matlab search path and to initialize the masterplotter submodule (requires git to be installed beforehand)!

T2N requires the TREES toolbox, so be sure that this toolbox is installed (<http://www.treestoolbox.org/download.html>) and added to the Matlab search path!

Note for Mac/Linux users: Matlab has to be run from the terminal for T2N to be properly working! For parallel NEURON to be working on Mac/Linux machines the OpenMPI software needs to be installed (<https://www.open-mpi.org/software/ompi/> ). Here is the terminal command for installing under Linux:

sudo apt-get install openmpi-bin openmpi-common openssh-client openssh-server libopenmpi1.3 libopenmpi-dbg

Note for Linux users: If you experience problems during NEURON installation or the compilation of mod files, execute the following commands in the terminal in order to be sure that those packages necessary for compiling are installed:

sudo apt-get update

sudo apt-get install g++ libxext-dev libncurses5-dev

T2N was written by Marcel Beining and available at <https://github.com/MarcelBeining/T2N>.

Bug reports or suggestions for improvement can be issued at <https://github.com/MarcelBeining/T2N/issues>.

# 3. Tutorials

To get into T2N quickly, there is a series of well commented example scripts which can be found in the “Tutorials” subfolder. The scripts are available in the convenient Matlab live script format (.mlx - introduced with Matlab 2016a) or as standard .m file.

# 4. The T2N main function

## 4.1 Definition of simulation parameters

All NEURON parameters are defined within 2 structures:

“trees” comprises all morphologies which should be used during the simulation run. The data format is the general TREES toolbox structure of trees. Trees of other formats can be loaded with TREES toolbox via the function “load\_tree” (but see (Cuntz, Forstner, Borst, & Häusser, 2011) or the online manual).

“neuron” is the main structure. It has up to seven fields each defining biophysical or experimental parameters.

Neuron simulations can be executed with Matlab using T2N:

[out, minterf, tree] = t2n(neuron, tree, options, exchfolder);

The (partly optional, see gray color) five input and four output parameters of T2N are described in more detail in the following.

### 4.1.1 The structure “neuron” to define the biophysical model and the experiment

“neuron” is a Matlab structure of length x (alternatively a cell array of length x, each containing a Matlab structure) which contains the parameter set(s) for the NEURON simulation (biophysical model, stimulations, recordings, etc). All x simulations are run in parallel as T2N distributes them on the available CPU cores. Each of these simulations can have a different parameter set defined by “neuron” but have access to the same morphology templates defined by “tree”.

Each Matlab structure in the variable neuron may contain the following fields (sorted by priority):

params defines general NEURON parameters, such as all simulation time variables, folder names and loading of custom hoc during simulation files.

This structure may contain the following fields (sorted by priority):

tstart This defines the starting time of the simulation in ms. This also defines the starting time for recording vectors etc. Default is 0 ms.

tstop This defines the end time point of simulation in ms. Default is 200 ms.

dt This parameter defines the simulation time step in ms. It is ignored (and a warning given) if the variable time step method (cvode) is activated. If not defined, NEURON uses its standard step of 0.025 ms.

nseg This can be a number defining the number of segments per section. Alternatively, this can be a string with ‘dlambda’ for using the dlambda rule to define nseg (caution with Ra/cm ranged values then! See ranged variable feature), or it can be ‘eachX’, where X is a number defining the distance in microns between segment nodes.

dlambda & freq (optional if nseg = ‘dlambda’) see <https://www.neuron.yale.edu/neuron/static/docs/d_lambda/d_lambda.html> Standard values are 0.1 and 300 Hz, respectively.

v\_init This defines the voltage in mV at which all compartments will be initialized. If not defined, NEURON uses its standard value of -65 mV.

celsius This parameter defines temperature in Celsius. This influences temperature-dependent mechanisms (e.g. ion channels). If not defined, NEURON uses its standard temperature of +6.3°C!

cvode This Boolean defines, if the variable time step method is used. This method speed up simulation time, if there are long time periods where no spiking occurs (see NEURON documentation).

parallel This variable enables the parallel NEURON feature and the number defines the number of cores that are used. This feature is beneficial for large-scale network simulations.

nrnmech By default, if there exists a “nrnmech.dll” file (containing the compiled mod files) in the current working directory of NEURON, it loads it. If another dll should be loaded from the standard folder lib\_mech, the file’s name can be defined here. This can also be a cell array of strings, if more than one dll has to be loaded. Be aware that one and the same mechanism can only be loaded once and an error will occur in NEURON if this is not considered! This feature does not work on Linux/Mac

prerun This defines a prerun time in ms during which the simulation runs in 10 ms time steps to let the system settle. For instance, an entry of 200 means that NEURON starts at -200 ms and lets the system settle until 0 ms. Then, the time is set to tstart and the normal simulation starts.

skiprun This Boolean allows to write the NEURON hoc files without running them. This can be used if a custom code defined in neuron.custom comprises a run command.

accuracy This parameter allows to increase the number of NEURON segments (nseg), which makes simulation spatially more accurate but slower. Possible values are:

0 no change (default)

1 increases nseg only in regions that contain the keywords “axon” and/or “soma” by a factor of 3. For simulations with fast spiking dynamics or with strongly varying range variables in the somatic/axonic region

2 increases nseg everywhere by a factor of 3. For simulations with general fast spiking dynamics or with generally strongly varying range variables

use\_local\_dt This Boolean is only important, if the variable time step (cvode) is used and several cells run in one simulation. Default is 0. If value is 1, each cell runs on an individual variable time step. This also transforms the T2N output time vector into a cell array containing an individual time vector for each cell.

q10 This Boolean defines if the passive properties (conductance, Ra and cm) are adjusted when the temperature is changed from 24°C) using the Q10 method. The Q10 values are from Trevelyan et al 2002 obtained from L2/3 pyramidal cells. Default is 0.

mech This field is used to define all distributed mechanisms (e.g. ion channels). The data structure is a n-by-1 cell array where n is the number of used cells (usually the number of trees in tree). Each cell entry is a Matlab structure containing

1. The name of the region at the 1st level (be sure to use the same region names as defined in the tree!)
2. The name of the mechanism to insert at the 2nd level
3. Optionally, a Matlab structure defining parameter values of the mechanism (parameter name as field name and parameter value as field value).

To define a mechanism for tree x including two mechanism parameters, this looks like:

neuron.mech{x}.regname.mechname.parname1 = value1;

neuron.mech{x}.regname.mechname.parname2 = value2;

or in a more compact form:

neuron.mech{x}.regname.mechname = struct(‘parname1’,value1,’parname2’,value2);

If no parameters are specified, the initial parameters of the mechanism are used (as defined in the mod file)

If a mechanism should be introduced to all sections/regions, use “all” as the region name, e.g.

neuron.mech{t}.all.mechname = struct(‘parname1’,value1);

If an ion mechanism is inserted (e.g. na\_ion) parameters to set can be *ion*i,*ion*o*,ion*i0,*ion*o0,e*ion* ,where *ion* is the corresponding ion (na,ca,k etc) Note, that the initial out- and inside concentrations cai0 and cao0 are GLOBAL variables, which means you cannot put different initial values at different locations. If you should have different concentrations, use a buffer model which writes cai / cao and do a prerun (see neuron.params.prerun)

**The ranged variable feature**

Distributed mechanisms normally contain range variables that may be different in different NEURON regions or even in different NEURON sections and segments. If you want to define such range variables on a more detailed level than for each region, you can use “range” as the region name in T2N. The structure then comprises parameter pairs with the name of the range variable as the field name and an n-by-1 vector with the range values, where n MUST be the number of nodes in the corresponding tree. If you want to put the same value in all segments, use the “all” feature (see above), which is computationally much more efficient. If you only want to define range values for some nodes whereas all other nodes should keep their standard (meaning defined by “all” or their region name, or even the standard value from the mod file) value , use NaN at these indices.

Example for defining ranged values:

neuron.mech{x}.range.mechname = struct(‘alpha’,avec,’beta’,bvec);

with avec/bvec of size n-by-1 where n is the # of nodes of tree x.

CAUTION#1: It is really a bad idea to set Ra or cm values with the ranged variable when dlambda rule for nseg calculation should be applied, because ranged values can only be set AFTER the dlambda was executed but then wrong Ra/cm values had been assumed during calculation.

CAUTION#2: If “nseg << # Tree nodes per section”, it might happen that different ranged values should be written to the same NEURON segment node. In that case an average is calculated from the values! This might also mean, that a node with “NaN” value is given another value because it is in the same segment as a specified node (NaNs are ignored when averaged).

CAUTION#3: If “nseg >> # Tree nodes per section”, the segments which do not have a corresponding Tree node are obviously not modified! This might cause confusion if you have a less realistic tree with only very few nodes and a huge inter-node-distance and you define ranged values. Either make more regions and simply use the normal T2N mechanism specification for a region (see above) or resample the tree to a smaller internode distance to avoid segments with no corresponding Tree node.

CAUTION#4: If you have many, many ranged parameters to set, this results in huge hoc files and might produce the infamous NEURON error “procedure too big” where there is no simple workaround…

pp This field is used to define all NEURON Point Processes (e.g. synapses or electrodes). Again, the data structure is an n‑by‑1 cell array where n is the number of used trees. Each cell entry is a Matlab structure containing

1. The name of the Point Process at the 1st level
2. A structure with information about the location and (optionally) parameter values of the Point Process. The location is set with the field “node” containing a scalar/vector of the node(s) where the Point Process(es) will be incorporated.

Examples

neuron.pp{1}.Exp2Syn = struct(‘node’,[50,100,150],’tau1’,0.2,’tau2’,3,’e’,0);

You can define multiple groups of the same Point Process class:

neuron.pp{1}.Exp2Syn(2) = struct(‘node’,[50,100,150],’tau1’,0.2,’tau2’,3,’e’,-70);

introduces inhibitory (negative reversal potential e) Exp2Syn synapses additional to the excitatory ones.

Generally, it is recommended to also add a unique string (tag) for each placed point process, especially in cases when you are placing several instances of the same point process at the same node and you want to record from them or add NetCons to them later, because otherwise you and t2n cannot distinguish between them! You can easily add a tag to your point process definition in form of a cell array of strings or a string array:

neuron.pp{1}.Exp2Syn(1) = struct(‘node’,[50,100,150],’tag’,{‘syn1’,’syn2’,’syn3’})

Here, 3 Exp2Syn point processes are added to the 3 specified nodes, so 3 tags are necessary. See the structure con and record for more information on how to use tags thereafter.

In the same way, NEURON’s standard electrodes are inserted. Amplitudes and durations can be given as defined in the NEURON documentation, however a T2N special feature for IClamp, VClamp and SEClamp is the possibility to simply define the time(s) (as many as wanted, but see limitation below) at which the current or voltage changes as well as the amplitude(s) to what value the current or voltage switches. For this, the two following fields have to be defined in addition to “node”:

‘times’: value vector containing the times at which the amplitude should change

‘amp’: value vector containing the amplitudes at each of the defined time points ([nA or mV])

Example

pp{1}.IClamp =

struct(‘node’,1,’times’,[0 50 250 300],’amp’,[0,-0.5,0.6,0]);

makes a 200 ms long hyperpolarization starting at 50 ms, directly followed by a 50 ms long depolarization.

If way too many values are given, NEURON returns an error because the procedure becomes too big. In that case, use the play feature to play the values into the electrode point process.

con This field is to define connections between cells, using the NetCon class in NEURON. The data structure is an n‑by-1 structure where n is the number of defined connection groups. As defining connections at least requires 5 values (from which node / point process located at which cell should a connection be made to which point process at which cell, and what is the trigger) these definition (as in NEURON, too) are the most complex ones in T2N, but follow an easy scheme.

The minimal required fields in each connection group are ‘source’ and ‘target’. Optional fields are ‘threshold’, ‘delay’ and ‘weight’ containing the respective value, otherwise their default values are: a threshold of 10, a delay of 1 ms and a weight of 0.

The ‘source’ field should contain a structure with fields

1. ‘cell’ defining the index of the source cell
2. ‘node’ defining the index of the TREE node (can be omitted for an artificial cell)
3. ‘watch’ defining the variable to be “watched”. When connecting a NetStim or IntFire as source, you can use ‘on’ as the ‘watch’.
4. If the source is a point process an additional field ‘pp’ defining the class name of the source Point Process is required.

Similarly, the ‘target’ field should contain a structure with fields

1. ‘cell’ defining the index of the source cell
2. ‘node’ defining the index of the TREE node
3. If the target is a point process an additional field ‘pp’ defining the class name of the target Point Process is required, e.g. ‘Exp2Syn’
4. If you have several groups of a Point Process class (e.g. Exp2Syn) defined at the same node, a connection is established to all of them unless you define the index to the PP group that should be connected to with the additional field ‘ppg’, e.g. “…’target’,struct(‘cell’,1,’pp’,’Exp2Syn’,’node’,50,’ppg’,2)…” to refer to the inhibitory synapse from our PP example.

Example:

neuron.con(1) = struct('source',struct('cell',2,'watch','on'),'target',struct('cell',1,'pp', 'Exp2Syn','node',50),'delay',0,'threshold',0.5,'weight',0.005);

makes a connection between cell 2 (which would be an artificial cell in that case) and the Exp2Syn Point Process at node 50 in cell 1 with a delay of 0 ms, a threshold of 0.5 and a weight of 0.005.

**Important hint:** If you want to place several Point Process instances of the same Point Process group at one and the same node (e.g. if synaptic density is higher than the number of nodes in that region), then you need to use the tag feature (see pp) in order to be able to target each in a specific manner. If you do, you can quite easily target point processes:

neuron.con(1) = struct('source',struct('cell',2,'watch','on'),'target',struct('cell',1,'pp', 'Exp2Syn',’tag’,{‘syn1’,’syn3’});

would only make a connection to the 2 excitatory Exp2Syn instances tagged with ‘syn1’ and ‘syn3’ from our example on page 14.

record This field is used to define all parameters that should be recorded and returned by T2N. Again, the data structure is an n‑by‑1 cell array where n is the number of used trees. Each cell entry is a Matlab structure containing

1. The name of the PointProcess class (or ‘cell’ if the parameter is from a compartment) in which the recorded parameter is located at the 1st level
2. The field ‘node’ (defining the Tree node(s) where to record) and ‘record’ (defining the parameter(s) to record) both at the 2nd level
3. Optionally, a field ‘Dt’ or ‘tvec’ at the 2nd level can be set, which leads to resampling of the recorded output to different time intervals (field Dt in ms) or to specific time points (field tvec, which contains an index pointing to one of the tvecs defined in neuron.tvecs). Of note, NEURON supports this with static dt but not with variable dt (cvode). However, t2n implements also cvode compatibility by resampling recorded values to the nearest previous time values, e.g. 3.1 is the nearest time point to 4 if the simulation time points with cvode had been [0.0, 3.0, 3.1, 4.1, 10.0].

Examples:

neuron.record{1}.cell = struct('record',’v',’node’,1);

records the voltage of cell 1 (a real morphology) at node 1

neuron.record{2}.cell = struct('record','on')

records the activity (i.e. of the ‘on’ variable) of cell 2 (a NetStim in that case)

neuron.record{1}.Exp2Syn = struct('node',50,'record','i')

records the current i of the Exp2Syn at node 50 of cell 1.

neuron.record{1}.cell = struct('record',’g\_pas',’node’,10,’Dt’,3);

records the passive conductance of cell 1 (a real morphology) at node 10 but returns only the values at a 3 ms interval.

neuron.record{1}.Exp2Syn = struct('node',50,'record','i',’tvec’,2) neuron.tvecs = { [0 5 20 100], 1:100 };

records the current i of the Exp2Syn at node 50 at cell 1 but returns only those values at time 1, 2, 3, …. , 100 ms. If tvec would equal 1, only the values at time 0, 5, 20, 100 ms would be returned.

Entries in field node and record can be multiple (e.g. [1:10] for ‘node’ or {‘v’,’i’} for ‘record)

If you record from a Point Process, be sure that it is defined at that node, otherwise nothing will be recorded. If you record from an artificial cell (i.e. point neuron), the field ‘node’ is ignored, as it has no tree nodes.

**Important hint:** If you want to place several Point Process instances of the same Point Process group at one and the same node (e.g. if synaptic density is higher than the number of nodes in that region), then you need to use the tag feature (see pp) in order to be able to target each in a specific manner. If you do, you can quite easily target point processes:

neuron.record{1}.Exp2Syn = struct(’tag’,{‘syn1’,’syn3’},'record','i');

would only record the current *i* from the 2 excitatory Exp2Syn instances tagged with ‘syn1’ and ‘syn3’ from our example on page 14.

tvecs This field is used to define different time points/vectors at which a recorded parameters can be resampled before being returned to Matlab. This is very helpful to drastically reduce loading times, when many parameters are recorded, but the time resolution is not important. The data structure is an n‑by‑1 cell array where n is the number of time vectors that are defined. See neuron.record for examples how to use this.

**Caution:** For these resampled recordings, the time vector out.t in the output structure is not valid, of course. Use the time vectors you defined in neuron.tvecs for plots and analyses.

play This field is used to define all value sets that should be played into a NEURON variable during the simulation (see play feature in the NEURON documentation). Again, the data structure is an n‑by‑1 cell array where n is the number of used trees. Each cell entry is a Matlab structure containing

1. The name of the PointProcess class (or ‘cell’ if the parameter is within a compartment) to which the defined values will be transferred at the 1st level
2. The fields ‘play’ (defining the parameter(s) to be overwritten), ‘times’ (defining the times in ms at which a change should happen), ‘value’ (comprising the value(s) to be played at the specified time(s)) and ‘node’ (defining the node(s) where this parameter should be overwritten) at the 2nd level
3. Optionally, a field ‘cont’ with value 1 at the 2nd level introduces a linear interpolation to calculate the values between specified time points (if no step wise change is desired).

Examples:

Manipulating g\_pas of cell 3 at node 1 linearly within 100 ms:

neuron.play{3}.cell = struct(‘node’ ,1, ’play’, ’g\_pas’, ’times’, [0 50 100], ’value’, [0.001 0.01 0.001], ’cont’,1);

To overwrite the activity of cell 2 (an artificial NetStim) so that it is active from 100 to 160 ms, write

neuron.play{2}.cell = struct('play', 'on' ,’times’ ,[0 100 160], ’value’, [0 1 0]);

To shut down the current i of the Exp2Syn Point Process at node 50 of cell 1 after 200 ms write

neuron.play{1}.Exp2Syn = struct('node',50,'play','i',’times’,[200],’value’,[0]);

Entries in field ‘node’ and ‘play’ can be multiple (e.g. [1:10] for ‘node’ or {‘v’,’i’} for ‘play’)

Be sure that the point process you want to play something into is defined at that node, otherwise nothing will happen. If you play into an artificial cell, the field ‘node’ is ignored.

APCount This field is to define action potential counting for specific cells. Alternatively, one can record the voltage and count the spikes in Matlab. As with mech and record, the data structure is an n‑by‑1 cell array where n is the number of used trees. Each cell may contain an m‑by‑2 array where m is the number of AP counting sites. The values in each cell row should be: [target node, voltage threshold]

This method does also work with artificial cells. In that case, T2N uses a NetCon to record their activities.

custom This field can be used to execute custom written hoc code at specific phases of the simulation run. The data structure is an n‑by‑2 cell where n is the number of defined custom code executions. The values in each cell row should be {‘*Code*’,’*phase*’}

‘Code’ can directly be a string with Hoc -code (normal Matlab formatting applies, such as \n for new line) or a string with a hoc file name that will be executed. This hoc file needs to be located in a folder “lib\_custom” in the main folder of the model. If the string has no “.hoc” ending it is always executed as NEURON code, so be sure to add the ending “.hoc” if you refer to a hoc file!

‘phase’ defines the phase during the simulation. Valid string entries are:

‘start’ (before parameter initialization)

‘mid’ (after initialization but before run)

‘end’ (after simulation run)

### 4.1.2 The cell array “tree” to define morphologies and point-neurons

This is a 1-by-n or n-by-1 cell array where n is the number of defined tree morphologies.

Normal morphologies have a structure according to TREES toolbox trees. It is recommended to use the “repair\_tree” function of the TREES toolbox first in order to avoid T2N errors due to 0-length segments, trifurcations etc.

There is also the possibility to define artificial cells in the “tree” array. “Artificial” in this context does not mean synthetic morphologies, but point neurons such as Integrate-and-Fire or NetStim objects. Instead of the normal TREES toolbox structure, these cells are defined by a structure with the field ‘artificial’ and a value corresponding to the name of the object class (e.g. IntFire1 or NetStim, see NEURON documentation for a list of all available objects). Optionally, object-related parameters can be specified with fieldname/fieldvalue pairs (fieldnames are parameter names and field value is parameter value). For example:

tree{2} = struct(‘artificial’,’NetStim’,’params’,‘noise’,0,’interval’,10,’number’,5);

creates a NetStim that fires 5 times at a (non-noisy) interval of 10 ms as a second tree.

### 4.1.3 The string “options” to define a few T2N options

“options” is a string with which the T2N’s behavior can be changed. The following arguments exist and are simply concatenated to a string:

-w opens a waiting bar (provided that Matlab GUI is active) showing the progress of the simulation(s)

-d Debug mode (NEURON is opened and more parameters are printed to the Matlab Output)

-q quiet mode -> suppress output and do not open NEURON. Has priority over –d

-o opens all NEURON instances instead of running them in the background

-m let T2N recompile the .mod files. Useful if a .mod file was modified. For safety of compiled dlls, this option does not work when an explicit name of a dll was given via neuron.params.nrnmech!

-ncX The number X defines the number of cores that should be used to distribute simulations on. If not defined, T2N automatically assigns simulations to each physical core it can detect.

The following options/features have not been tested with newer versions of T2N and might return errors!

-cl cluster mode -> files are prepared to be executed on a server cluster. Cluster parameters need to be provided to t2n in a Matlab struct as fifth input argument.

### 4.1.4 The string “exchfolder” to define the relative location of temporary files

This string will be the name of the exchange folder between T2N and NEURON within the model folder. If not provided, the default name is t2n\_exchange.

## 4.2 The output of T2N

### 4.2.1 The structure “out“containing all recorded values

out contains the recordings which had been defined via neuron.record. If only one simulation was run, out is a structure, if n simulations were run, out is an n‑by‑1 cell array with each cell containing the corresponding output structure, which is:

t This field contains the time vector of the simulation. As mentioned, for variable time steps with the use\_local\_dt feature enabled, this field is an n-by-1 cell of vectors with n being the number of used trees.

record This field contains all recorded parameters. Its data structure is an n-by-1 cell array with the recordings of all n simulated neurons and containing:

1. The name(s) of the PointProcess class (or ‘cell’ if the parameter was recorded from a compartment) in which the recorded parameter was located at the 1st level
2. The name(s) of the recorded parameter at the 2nd level
3. A) If no tags were used, an n-by-1 cell array at the 3rd level with n being the highest node where the corresponding parameter was recorded from. B) If tags were used to define the recorded point processes (see pp page 14), then the 3rd level does not contain a cell array but a structure with field names according to the used tags. This is necessary to distinguish point processes placed at the same node.

Example

To access the recorded voltage vector at node 12 of cell 3 write

out.record{3}.cell.v{12}

To access the recorded current vector of an Exp2Syn tagged with “tagA”

out.record{2}.Exp2Syn.i.tagA

APtimes This field, an n-by-1 cell array, which contains the spiking times of each of the n trees that had been recorded with neuron.APCount

### 4.2.2 The optional cell “minterf”

”minterf“ is an n-by-1 cell each containing a matrix for one of the n trees that had been used. The matrix is of size m-by-3 and gives to each Tree node (first column) the corresponding NEURON sections (second column) and segment locations (third column). Note that nodes, which define the end of a section, are at the same time the starting position of the attached section. This is why they correspond to location 0 of one and to location 1 of the other section at the same time. This interface matrix is also saved in the morphology folder in the form of a .dat and .mat file.

### 4.2.3 The optional structure “tree”

”tree“ contains the putatively modified tree cell array after execution of T2N. This might be important, e.g. if the trees had not been sorted beforehand with “sort\_tree“, because then, the node indices might not correspond to the previous tree locations anymore. It is generally recommended to sort and repair tree morphologies before using T2N.

# 5. Protocol functions

Protocol function avoid the need to write T2N code of NEURON configurations for most standard experimental approaches, such as current or voltage clamps, blocking channels, analyzing resonance, etc.. Also they include functions analyze the output of such simulations and to make convenient plots from them. In the following, each function is explained in detail.

## 5.1 t2n\_APprop – Calculating action potential properties

[props, fig, figname] = t2n\_APprop(targetfolder\_data,neuron,currstep,tree,dataset,askfile)

This function analyzes and optionally plots various action potential (AP) characteristics at a current injection step previously simulated with t2n\_currsteps

**INPUTS**

targetfolder\_data destination of the temporary results file

neuron t2n neuron structure with already defined mechanisms

currstep current injection amplitude at which AP properties

should be analyzed [nA]

tree (optional) tree cell array with morphologies to

calculate dependencies on tree surfaces

dataset (optional, so far only compatible with GC model) dataset of

experimental data which should be loaded

askfile (optional) boolean if it should be ask for which

simulation should be loaded and analyzed

**OUTPUTS**

props structure with AP properties. If this is the only

output defined, figures are not plotted

fig figure handles to figures

figname name of each figure

## 5.2 t2n\_bAP – Perform a backpropagating AP simulation

t2n\_bAP(neuron, tree, amp, targetfolder\_data, simplicity)

This function performs an experiment on backpropagating action potentials (bAPs) by "zapping" the cell(s) with a short high current and recording the membrane potential at all locations of the cell(s). The results are saved in a mat file named Exp\_bap and the specification in neuron.experiment and can then be analyzed with t2n\_plotbAP.

**INPUTS**

neuron t2n neuron structure with already defined mechanisms

tree tree cell array with morphologies

amp amplitude [nA] of zap. Default is 1.3 nA

targetfolder\_data destination of temporary results file

simplicity 0 (DEFAULT) measure voltage over whole dendritic tree

1 only record from every second node

2 only record voltages from the first primary dendrite

## 5.3 t2n\_blockchannel – Specifically reduce conductances

neuron = t2n\_blockchannel(neuron, channels, amount, regions, specify)

This function manipulates the neuron structure to reduce specific channel conductances defined by "channels" in specific regions of the cell(s) defined by "regions" by a certain amount defined by "amount"

**INPUTS**

neuron t2n neuron structure with already defined mechanisms

channels string or cell array of strings which channels should

be blocked (i.e. its conductance reduced)

amount amount of blockade (0-100) [%]. This could also be used to increase conductances if a negative percentage is given, however it is recommended to rather use t2n\_changemech.

regions allows to select only specific regions of cell for blockade. Can be string or cell array of strings. The region name has to be defined in the tree regions, too.

specify allows to select only specific conductances of a channel, eg. gabkbar\_BK

**OUTPUTS**

neuron manipulated t2n neuron structure

## 5.4 t2n\_changemech – Manipulate mechanism parameters

strct = t2n\_changemech(strct, change, mode)

This function changes parameters in the T2N neuron mech structure according to argument "change".

**INPUTS**

strct neuron structure containing ion channel densities or alternatively only the mech field of the neuron structure

change structure with field names according to mechanism parameter names that should be changed and values either describing the factor by which it should be changed or an absolute value

mode scalar defining how parameters should be changed:

1 or 'relative': (DEFAULT) values in change are relative factors (e.g. 0.5 for 50% decrease)

2 or 'absolute': values are absolute values that overwrite the values in strct

**OUTPUTS**

strct updated neuron or mechanism structure

**NOTE**

Cell arrays of neuron structure are not supported.

## 5.5 t2n\_currSteps – Applying current clamps

t2n\_currSteps(neuron, tree, targetfolder\_data, ostruct)

This function performs one or multiple current clamp steps in the cells given by "tree" and "neuron" and saves the results in a mat file named according to neuron.experiment for later analysis, e.g. using t2n\_plotCurrSteps or t2n\_FIplot.

**INPUTS**

neuron t2n neuron structure with already defined mechanisms

tree tree cell array with morphologies

targetfolder\_data destination of temporary results file

ostruct structure with fields defining the curr step simulation

amp vector with amplitudes [nA]

delay time point at which current injection starts [ms]

duration time period of current injection [ms]

holding\_voltage (optional) potential at which cell is held before current injection [mV]

spikeThresh (optional) threshold above which spikes are

detected (default is -15 mV)

## 5.6 t2n\_findCurr – find the current to hold a certain voltage

[amp, Vrest] = t2n\_findCurr(neuron,tree,desv,amp,options)

This function finds the current necessary to keep the neurons defined in "neuron" and "tree" at a desired voltage (0.2 mV precision) or alternatively finds the spiking threshold with 0.5 pA precision.

**INPUTS**

neuron t2n neuron structure with already defined mechanisms (see documentation)

tree tree cell array with morphologies (see documentation)

desv desired voltage (mV) or 'spike' if spiking threshold is searched for

amp (optional) starting values of the current amplitudes (one for each neuron defined in tree)

options (optional) options for starting T2N (see t2n)

**OUTPUTS**

amp current amplitude for each neuron to reach the desired voltage/spike

Vrest resting potential of each cell

## 5.7 t2n\_findFreq – Find current to fire certain # of APs

[amp] = t2n\_findFreq(neuron, tree, desNum, options)

This function finds the current necessary to let each neuron spike a certain amount of spikes with an IClamp protocol previously defined in "neuron"

**INPUTS**

neuron t2n neuron structure with already defined mechanisms (see documentation)

tree tree cell array with morphologies (see documentation)

desNum desired number of spikes during the IClamp protocol

options (optional) options for starting T2N

**OUTPUT**

amp current amplitude [nA] for each neuron to reach the desired number of spikes

## 5.8 t2n\_findSubthreshWeight – Find synaptic weights for cells to reach subthreshold potentials

[weight] = t2n\_findSubthreshWeight(neuron, tree, weight, freq, tim)

This function finds the synaptic weight of Exp2Syn synapses necessary to have the neurons defined in "neuron" and "tree" at a sub-spiking threshold level.

**INPUTS**

neuron T2N neuron structure (already containing all synapses for which the weight should be searched.

tree TREES toolbox tree cell array

weight starting weight for the synapses

freq frequency of the netstim input

tim (optional) time [ms] after which simulation stops (default five

rounds of synaptic activation)

**OUTPUT**

weight synaptic weight for each neuron to reach subthreshold state

## 5.9 t2n\_FIplot – Plot the FI-curve of cells

fig = t2n\_FIplot(targetfolder\_data, neuron, ostruct, targetfolder\_results)

This function uses the simulation data generated by t2n\_currSteps and plots the spike frequency / number of APs at each current step.

**INPUTS**

targetfolder\_data folder which was given to t2n\_currSteps, where the data of the simulation lies

neuron t2n neuron structure (see documentation)

ostruct structure with fields defining some output

figurewidth width of figures to be created

figureheight height of figures to be created

savename prefix filename of figures when saved

color (optional) 1x3 RGB vector for the plotted line color

handles (optional) handles to the previously created figure in case they should be merged

targetfolder\_results folder where pdfs from figures should be saved. If not provided, figures will only be plotted

**OUTPUT**

fig figure handles to the output figures

## 5.10 t2n\_IVplot – Plot the I-V relationship of cells

fig = t2n\_IVplot(targetfolder\_data, neuron, ostruct)

This function uses the simulation data generated by t2n\_voltSteps and plots the IV relationship at each voltage step.

**INPUTS**

targetfolder\_data folder which was given to t2n\_voltSteps, where the data of the simulation lies

neuron t2n neuron structure (see documentation)

ostruct structure with fields defining some output

single (optional) boolean to determine if one line for each cell is plotted

subtract\_hv (optional) boolean to determine if the current measured at the baseline voltage step is subtracted from the other currents

handles (optional) handles to the previously created figure in case they should be merged

**OUTPUT**

fig figure handles to the output figures

## 5.11 t2n\_passTests – Get passive properties on each cell

[Rin, tau, cap, Vrest] = t2n\_passTests(neuron, tree, targetfolder\_results, ostruct)

This function applies protocols to extract passive properties (input resistance, membrane time constant, capacitance and resting membrane potential) from each cell.

**INPUTS**

neuron t2n neuron structure (see documentation)

tree tree cell array with morphologies (see documentation)

targetfolder\_results folder where pdfs from figures should be saved. If not provided, figures will only be plotted

ostruct structure with fields defining some parameters passtest Exact passive test that will be applied. These were extracted from several papers: 'Mongiat', 'Mongiat2', 'Mehranfahrd', 'SH' and a standard protocol 'Std'

figureheight (optional) height of figure in cm

figurewidth (optional) width of figure in cm

recordnode (optional) node index at which the current/voltage will be recorded. Default is root.

stimnode (optional) node index at which the current/voltage will be injected. Default is root.

**OUTPUTS**

Rin input resistance [MOhm] of all cells

tau membrane time constant [ms] of all cells

cap capacitance [pF] of all cells

Vrest resting membrane potential [mV] of all cells

## 5.12 t2n\_plotbAP – Show backpropagating AP properties

[bAPdistHM, mveloc\_dend, mveloc\_farax, mveloc\_nearax, fig] = t2n\_plotbAP(targetfolder\_data, neuron, ostruct, targetfolder\_results)

This function uses the resulting data from t2n\_bAP and plots the backpropagating spike amplitude as well as the delay versus distance to the root. Furthermore maps of the amplitude onto each tree are generated. Figures will only be plotted if no output is defined or fig is included in the output.

**INPUTS**

targetfolder\_data folder given to t2n\_currSteps, where the simulation data lies

neuron t2n neuron structure (see documentation)

ostruct structure with fields defining some output

figurewidth width of figure to be created

figureheigth height of figure to be created

dist string defining how distance to root is measured: 'Eucl.' for Euclidean distance or 'PL' for path length

relamp boolean for normalization of values

plotData boolean if experimental data from rat DGCs should be added to the graph

targetfolder\_results folder where pdfs from figures should be saved.

**OUTPUTS**

bAPdistHM distance [um] at which the bAP amplitude reached half-max.

mveloc\_dend mean AP velocity at the dendrite [um/ms]

mveloc\_farax mean AP velocity at the distal axon [um/ms]

mveloc\_nearax mean AP velocity at the proximal axon [um/ms]

fig figure handles

## 5.13 t2n\_plotCurrSteps – Plot current clamp simulations

t2n\_plotCurrSteps(targetfolder\_data, neuron, steps)

This function plots one or multiple current steps, which had been previously simulated and saved with t2n\_currSteps. The exact loaded simulation is defined by neuron.experiment.

**INPUTS**

targetfolder\_data destination of temporary results file

neuron t2n neuron structure with already defined mechanisms

steps (optional) restrict plot on specific current steps [nA]

## 5.14 t2n\_plotdV – Make phase plots

[maxdv, fig] = t2n\_plotdV(targetfolder\_data, neuron, ostruct , targetfolder\_results)

This function creates a phase plot (dV/dt) of the current steps that were simulated with t2n\_currSteps.

**INPUTS**

targetfolder\_data folder which was given to t2n\_currSteps, where the simulated data lies

neuron t2n neuron structure (see documentation)

ostruct structure with fields defining some output

figurewidth width of figure to be created

figureheigth height of figure to be created

savename prefix filename of figures when saved

ampprop amplitude [nA] for which an extra figure will be made and the maximal dV is calculated.

targetfolder\_results folder where pdfs from figures should be saved. If not provided, figures will only be plotted

**OUTPUTS**

maxdv maximal rate of change of the membrane voltages for each cell (mV/ms)

fig figure handles to the plotted figure

## 5.15 t2n\_plotVoltSteps – Plot voltage clamp simulations

fig = t2n\_plotVoltSteps(targetfolder\_data, neuron,steps, subtract\_hv)

This function plots one or multiple voltage steps, which had been previously simulated and saved with t2n\_voltSteps. The exact loaded simulation is defined by neuron.experiment.

**INPUTS**

targetfolder\_data destination of temporary results file that was provided to t2n\_voltSteps

neuron t2n neuron structure with already defined mechanisms

steps (optional) make 2nd plot where only these steps [nA] are plotted

subtract\_hv Boolean if the current at the holding potential (before the step) should be subtracted. Default is 0

**OUTPUT**

fig figure handles to the plotted figures

## 5.16 t2n\_Q10pas – Adjust passive model to temperature

neuron = t2n\_Q10pas(neuron, celsius)

This function adjusts the passive parameters g and Ra to a given temperature (assuming that g and Ra were defined for 24°C before). Hence, do not use it multiple times!

**INPUTS**

neuron t2n neuron structure with already defined mechanisms

celsius the temperature [C] to which the passive parameters will be adjusted to

**OUTPUT**

neuron the modified neuron structure

## 5.17 t2n\_resonance – Perform a resonance test

[imp, freq] = t2n\_resonance(neuron, tree, amp, holding\_voltage, doerrorbar)

Perform a resonance test (impedance measurement using oscillating current

injections) on each cell and plot the result.

**INPUTS**

neuron t2n neuron structure with already defined mechanisms

tree tree cell array with morphologies (see documentation)

amp amplitude [nA] of the oscillating current injection

holding\_voltage holding potential [mV] before current injection

doerrorbar (optional) Boolean if error bar should be added to plot

**OUTPUTS**

imp matrix with impedances [MOhm] of each cell at all frequencies

freq frequency vector [Hz] same size as imp

## 5.18 t2n\_setionconcentration – Change ion concentrations according to literature

neuron = t2n\_setionconcentration(neuron, str)

This function modifies the Nernst potential or the extra- and intracellular ion concentrations (which then leads to different Nernst potential) according to specific experiments in literature. Of course, the list can be extended by anyone.

**INPUTS**

neuron neuron structure containing ion channel densities

str string defining the experimental conditions from literature.

Current cases: 'Krueppel' from Krueppel et al. (2011) Neuron

'Mongiat'&'Mongiat2' from Mongiat et al (2009) PlOS One

'Riazanski'&'SH07' from Schmidt-Hieber et al (2007) JNS; and Riazanski et al (2001) J. of Physiology

'SH08' from Schmidt-Hieber et al (2008) Journal of Physiology

'Stocca' from Stocca et al (2008) Journal of Physiology, (used for the adult mice)

**OUTPUT**

neuron updated neuron structure

## 5.19 t2n\_voltSteps – Applying voltage clamp protocols

[currVec, out] = t2n\_voltSteps(neuron,tree,vstepsModel,dur,holding\_voltage,targetfolder\_data)

This function performs one or multiple voltage steps (in the squared pulse format, i.e. voltage step is surrounded by step to the baseline potential) in the cells given by "tree" and "neuron" and saves the results in a mat file named according to neuron.experiment for later analysis.

**INPUTS**

neuron t2n neuron structure with already defined mechanisms

tree tree cell array with morphologies

vstepsModel vector with voltages [mV] that will be held

dur duration time period of voltage step [ms]. Can be a 1x3 vector which then includes the pre and post step duration at which the cell is held at the holding potential, or a scalar with only the step duration and pre/post being set to 100 ms

holding\_voltage potential [mV] at which cell is held before and after the voltage step

targetfolder\_data destination of temporary results file

**OUTPUTS**

currVec cell array of vectors containing the simulation time vector and the measured current during the simulation for each cell (first dim) and voltage step (second dim)

out the direct output structure of t2n, necessary for some other t2n functions

# 6. Auxiliary functions

Auxiliary functions simplify handling T2N, Matlab and NEURON. They help setting or returning specific values or are useful subfunctions of the T2N main function. In the following, each function is explained in detail.

## 6.1 t2n\_as – Copy neuron setting to other instances

This function completes the neuron specifications of one simulation with the neuron specifications of another simulation. This is helpful e.g. if you do many similar simulations in parallel where you just want to change one parameter.

neuron = t2n\_as(x, neuron)

**INPUTS**

“*x*” defines the simulation number with which the current neuron specification should be completed

“*neuron*” is the so far defined neuron structure. If this argument is left out, T2N\_as creates a neuron structure where all possible fields are pointing to simulation x.

**OUTPUT**

“neuron“ is the completed neuron file

**Usage example**

loc = [1 10 50 100]; % specify node locations

neuron{1} = ……. % full specification of all parameters for the first simulation/ location

for s = 2:numel(loc) % go through all other locations

neuron{s}.pp{1}.IClamp = struct(‘node’,loc(s)); % change node where IClamp is % placed each time

neuron{s} = T2N\_as(1, neuron{s}): % define all other parameters as in simulation 1

end

**Note**

This function has no advantage for computation speed in Matlab, as one could simply write neuron{s} = neuron{1}. However, T2N\_as speeds up hoc writing timing enormously, as T2N is simply told to let NEURON execute the relevant hoc files from the first simulation folder instead of writing separate hoc files for each simulation.

## 6.2 t2n\_catName – concatenate name of simulation

outname = t2n\_catName(varargin)

This function concatenates strings to a full file, e.g. for file names. If the first input is a path (i.e. containing separators / or \) than the function concatenates the strings to a full file path.

**INPUTS**

varargin any number of strings

**OUTPUT**

outname concatenated full name

## 6.3 t2n\_catStruct – concatenate Matlab structures

strct = t2n\_catStruct(varargin)

This function concatenates two structures which have the same fields.

**INPUTS**

varargin several structures with same field names

**OUTPUT**

strct concatenated structure

## 6.4 t2n\_checkinput – Sanity checks on neuron/tree struct

[neuron, tree, usestreesof, nocell, exchfolder] = t2n\_checkinput(neuron, tree, options)

This sub function of the t2n main function checks the neuron and tree structure for correct definition of the used morphologies and returns information about it.

**INPUTS**

neuron t2n neuron structure with already defined mechanisms (see documentation)

tree tree cell array with morphologies (see documentation)

options (optional) option string of t2n

**OUTPUTS**

tree corrected tree cell array

neuron corrected neuron structure

usestreesof points to the neuron entry/instance from which the tree definitions are taken from

nocel Boolean if neuron input was a structure or cell array

exchfolder name for exchfolder that was possibly found in the neuron structure

## 6.5 t2n\_get – Return/Use recorded parameter

[vec, tvecout ] = t2n\_get(out, par, arg, typ)

This function can be used to obtain a recorded parameter during a specific time point/period of the simulation.

**INPUTS**

out the output structure of the t2n main function

par string of the parameter to obtain

arg can be either

- a scalar defining time point [ms] at which the parameter should be returned

- a 1x2 vector with the start and end time point [ms] between which the parameter should be returned

- the string name of a function that should be applied on the recorded parameter values at each node

typ string that specifies if the recorded parameter is from the cell ('cell',default) or from a point process (e.g. 'IClamp')

**OUTPUTS**

vec cell array or vector returning the desired parameter at simulation (first level) and node (second level)

tvecout time vector [ms] if arg was no string

**NOTE**

This function is currently not supporting NEURON's local\_dt feature.

## 6.6 t2n\_getGIDs – Return information for parallel NEURON

[GIDs, neuron, mindelay] = t2n\_getGIDs(neuron, tree, thesetrees)

This function prepares the neuron structure to be used for the parallel NEURON environment. Also it returns the global IDs, which are necessary for the parallel environment.

**INPUTS**

neuron t2n neuron structure with already defined mechanisms

tree tree cell array with morphologies

thesetrees (optional) if not all trees of 'tree' are used, this is the index array to the used ones

**OUTPUTS**

GIDs structure with information on all needed global IDs that are initialized in parallel NEURON

neuron complemented neuron structure ready for parallel NEURON

mindelay minimum delay and time step of parallel network (calculated from minimal delay between synapses)

## 6.7 t2n\_getMech – Visualize/return NEURON mech values

[varVec] = t2n\_getMech(neuron,tree, var)

This function maps the values of a given NEURON variable onto each morphology or returns these values. This is useful, e.g. for validating ion channel density distribution of a biophysical model. It can also read the standard value of the parameter from the mod file corresponding to the mechanism in order to return the complete parameter distribution as set by NEURON.

**INPUTS**

neuron t2n neuron structure or cell array of structure (see documentation)

tree TREES toolbox morphologies

var string with valid NEURON variable name, such as 'g\_pas'

**OUTPUT**

varVec (optional) m-by-n cell array with mapped values for m neuron instances and n trees. If defined, values will not be visualized

## 6.8 t2n\_getPP – Visualize/return NEURON point processes

[varVec] = t2n\_getPP(neuron,tree, pp)

This function maps the locations of one or more given NEURON point processes onto each morphology or returns their node indices. This is useful, e.g. for validating synapse distribution of a biophysical model.

**INPUTS**

neuron t2n neuron structure or cell array of structure (see documentation)

tree TREES toolbox morphologies

pp string or cell array of strings with valid NEURON point process names, such as 'Exp2Syn'

**OUTPUT**

varVec (optional) m-by-n cell array with o-by-p logical vector for m neuron instances, n trees, o tree nodes and p pp names. If defined, values will not be visualized

## 6.9 t2n\_getref – Get reference ID to neuron instance

n = t2n\_getref(n, neuron, field)

This function returns the neuron instance that is referenced in neuron instance 'n' at field 'field'. If the neuron instance has own definitions at 'field', n itsself is returned. The function t2n\_as produces such references.

**INPUTS**

n index of the neuron instance, which reference is searched for

neuron t2n neuron structure with already defined mechanisms (see doc.)

field the field for which reference is searched for, e.g. params, mech, pp etc.

**OUTPUT**

n reference to the neuron instance comprising the definitions of 'field'

## 6.10 t2n\_getSynNum – Calculate synapse numbers

[nsyn,synids] = t2n\_getSynNum(tree, syn\_dens, regions)

This function calculates the number and location of synapses at a given morphology given a synaptic density 'syn\_dens'. The output ‘synids’ can then be used to place location-specific synapse mechanisms such as Exp2Syn.

**INPUTS**

tree one morphology structure (see documentation)

syn\_dens single scalar with the desired synaptic density [# per length unit of morphologies]. If multiple regions are given (see next input), this can also be a vector with one entry for each region.

regions (optional) regions for which the synapse number should be calculated for. These can be anything from the 'rnames' field of the tree structure. Default is all regions.

**OUTPUTS**

nsyn number of synapses at each node index of 'tree'

synids index to the node for each synapse to be implemented

## 6.11 t2n\_initModelfolders – Start a new model with T2N

t2n\_initModelfolders(folder)

This function initializes a new folder in which all necessary T2N files and folders are created in order to start a new compartmental model featuring T2N.

**INPUTS**

folder (optional) path to the folder which should be initialized/created. If not provided, you will be asked to specify the directory.

## 6.12 t2n\_makeNseg – Translate TREES nodes into NEURON sections/segments

minterf = t2n\_makeNseg(tree, minterf, par, mech)

This function calculates at which position of each cell there will be a segment node in NEURON. This is necessary for T2N to correctly translate TREES toolbox node information into NEURON section and segment information.

**INPUTS**

tree single TREES toolbox tree morphology

minterf Nx3 mapping matrix created by neuron\_template\_tree (in morphology folder)

par T2N neuron.params structure

mech T2N neuron.mech structure that belongs to the 'tree' morphology

**OUTPUT**

minterf updated mapping matrix with 4th column showing the real segment node locations

## 6.13 t2n\_plotChannel – Visualize ion channel kinetics

t2n\_plotChannel(neuron, mcondChannel, region, outputFolder, options)

This function plots the activation and inactivation dynamics of an voltage-dependent ion channel. The fits on the activation and inactivation time are mono-exponential, hence they should be handled with care in cases the channel has more complex kinetics.

INPUTS

neuron t2n neuron structure with already defined mechanisms

mcondChannel the name of the maximum conductance parameter of a channel, as it is in NEURON and NMODL (e.g. gbar\_hh)

region name of the region in ‘neuron’ from which specifications for the channel should be taken from. If not provided, t2n takes the first specification it can find within a region

outputFolder (optional) folder where pdfs should be saved to

options string with more options that can be concatenated:

-h set if channel is hyperpolarization-activated

-si or -sa set if channel inactivation or activation is very slow (dt and duration increased)

-fi or -fa set if channel inactivation or activation is very fast (dt and duration decreased)

## 6.14 t2n\_plotRaster – Generate a spike raster plot

t2n\_plotRaster(spikeMat, tvec)

This function plots the raster plot of one or multiple spike trains that were obtained from an APCount, created manually or generated e.g. with t2n\_poissonSpikeGen.

**INPUTS**

spikeMat logical matrix with Ones where a spike occurs. Alternatively it can be a vector with spiking times, or for multiple spike trains, a cell array with each cell comprising the spike time vectors

tVec corresponding time vector [ms]. Only necessary if spikeMat is a logical matrix

## 6.15 t2n\_plotTrees – Visualize morphologies for publication

h = t2n\_plotTrees(tree,targetfolder,col,ostruct)

This function plots each tree in a nice way and saves them as eps files, eg for using in Adobe Illustrator.

**INPUTS**

tree TREES toolbox tree cell array

targetfolder target folder for output files

col (optional) cell array with rgb color values for each tree

ostruct (optional) option structure with possible fields

'show' 1 = colored plotting but not good for putting in Illustrator 2 = noncolored plotting

'savename' prefix name for image files.

**OUTPUT**

H (optional) figure handles to the figures

## 6.16 t2n\_pnode – Get all parameters at a specific node

prop = t2n\_pnode(neuron, tree, node)

This function returns all properties (mechanisms and point processes) that have been set at a specific node. This is similar to psection() in NEURON and useful for validation of set parameters.

**INPUTS**

neuron t2n neuron structure (see documentation)

tree TREES toolbox tree structure or cell array of tree structures

node index (or indices if multiple trees) to the node of interest in the tree

**OUTPUT**

prop structure or cell array of structures with all properties at the specified node of the tree(s)

## 6.17 t2n\_poissonSpikeGen – Generate Poisson spike trains

[spikeMat, tVec] = t2n\_poissonSpikeGen(freq, par, nTrials)

This function creates Poisson spike trains at a frequency 'freq'. The output is a spike matrix and a a time vector, both which can be used to define a artificial NEURON VecStim that follows such a spike train.

**INPUTS**

freq desired frequency of the spiking [Hz]

par parameter structure of t2n neuron.params (see documentation)

nTrials number of independent spike traces to be generated (corresponds to the rows in the returned spike matrix)

**OUTPUTS**

spikeMat logical matrix with ones where a spike occurs, e.g. for input to a NEURON VecStim point process

tVec corresponding time vector [ms]

## 6.18 t2n\_renameNrnmech – rename compiled dll (Windows)

t2n\_renameNrnmech(newname, path)

This function renames the nrnmech dll file to the name specified in newname. Also deletes all .o and .c files created during dll compilation by NEURON

**INPUTS**

newname string with new name for dll file

path (optional) path to the main folder of the model (containing the folder lib\_mech) If not provided, the current Matlab working directory is used.

## 6.19 t2n\_testComp – Generate simple compartment

tree = t2n\_testComp

This function returns a single compartment of region "soma" in the TREES toolbox format, which can be used for testing purposes.

**OUTPUT**

tree The testing compartment tree structure

## 6.20 t2n\_writeTrees – Translate morphologies into hoc

tree = t2n\_writeTrees(tree, tname, savepath)

This function transforms TREES toolbox morphologies (swc, mtr, neu) into hoc code. In order to associate each TREES tree with its hoc file, the name of the hoc files is saved in each tree structure under the field “NID” (Neuron ID). This is why the TREES morphology has to be resaved by t2n\_writetrees. The name of the hoc file is dependent on the kind of the tree and parameters that are specified in the following order

1. If the tree is artificial (e.g. an IntFire object), the hoc file is saved only once for each object class with the name being the name of the object, e.g. “cell\_NetStim.hoc”.
2. If the tree is a real morphology …
   1. …and “tname” has been specified as an input argument then each tree’s hoc file has this name plus a counter suffix.
   2. …and the tree has a name specified in the TREES structure field “name”, then the tree’s hoc file has this name. Be sure that no trees have the same name!
   3. … and none of the above apply, the hoc files simply have the name “Tree” plus a counter suffix.

Note that characters such as “%”, “-“ and “.” cannot be used in NEURON (since they are reserved) and are automatically deleted (%) or replaced with “\_”.

**INPUTS**

tree tree cell array with morphologies (see documentation)

tname (optional) name after which all tree files will be named (with counting). If not provided, the tree.name will be used, or, if not given simply “Tree”+number

savepath (optional) string containing the absolute file location (folder + filename) of the TREE structure which needs to be resaved after the hoc files have been written. This prevents t2n\_writetrees from asking for it via a dialog and is especially useful, if the tree location and file name are known anyway from the input/output of load\_tree (see TREES manual).

**OUTPUT**

tree tree cell array with each tree containing a unique NEURON ID (NID) which is the name of the corresponding hoc file