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, 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. 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.

T2N was developed on Matlab2014a and tested on versions from 2012a to 2015a. T2N is not functional on Matlab 2009a or older.

T2N was written by Marcel Beining and available at [www.treestoolbox.org](http://www.treestoolbox.org) or XXXXX.github.com. Bug reports and suggestions for improvement can be issued at github.

# 1. The T2N main function

## 1.1 Definition of simulation parameters

All NEURON parameters are defined within 3 structures:

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

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

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

Neuron simulations can be executed with Matlab using T2N:

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

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

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

This is a 1 by m cell array where m is the number of defined trees.

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 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, an additional field ‘params’ containing a structure with object-related parameters can be specified (fieldnames are parameter names and field value is parameter value). For example:

tree{2} = struct(‘artificial’,’NetStim’,’params’,struct(‘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.

### 1.1.2 The structure “params” to define general NEURON parameters

This structure may contain the following fields (sorted alphabetically):

access 1 by 2 vector with tree index and node index that defines the NEURON standard accessed section. If not defined, the first node (root) of the first non-artificial cell is used.

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

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).

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.

exchfolder This string provides the (relative) path to the folder where temporary simulation files are stored.

openNeuron This Boolean defines, if neuron should be opened or ran non-visibly in the background.

Note, if NEURON gets opened, T2N cannot handle/show NEURON errors since they are not returned to Matlab!

path This defines the (absolute) path to the model’s root folder and the working directory of NEURON. If not specified, the current Matlab working directory is used.

morphfolder This is the (relative) path to the folder where the NEURON morphology templates will be written to / opened from

neuronpath This is the (absolute) path to the NEURON executable “nrniv.exe”. This definition is mandatory!

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!

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, or it can be ‘eachX’, where X is a number defining the distance in microns between segment nodes.

numCores This number defines the number of cores that should be used to distribute simulations on. If not defined, T2N automatically assigns simulations to each physical core.

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.

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.

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.

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.

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.

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

“neuron” is a Matlab structure which contains the parameter set(s) for the NEURON simulation (biophysical model, stimulations, recordings, etc). It can have length 1 or length x. In the latter, 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 they share the same NEURON parameters (defined in “params”) and have access to the same morphology templates defined by “tree”. It is also possible to define x several parameter sets by a x-by-1 cell array instead of a structure array.

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

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 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 value, use NaN at these indices.

Example for defining ranged values:

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

with avec and bvec of size n-by-1 where n is the number of nodes of tree x

CAUTION#1: 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#2: 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#3: If you have many, many parameters to set, this 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 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 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 will be incorporated.

Examples

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, e.g. add

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

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

In the same way, NEURON’s standard electrodes are inserted. Amps and durations can be given as defined in the NEURON documentation, however a special feature for IClamp, VClamp and SEClamp is the possibility to simply define the time(s) (as many as wanted, but see limitation below) when the current or voltage changes and 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 of changing the amplitude

‘amp’: value vector containing the amplitudes at each time point ([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 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 m-by-1 structure where m 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 0 weight.

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
3. ‘watch’ defining the variable to be “watched”
4. If the source is a point process an additional field ‘pp’ defining the class name of the source Point Process is required. When connecting a NetStim or IntFire as source, use ‘on’ as the ‘watch’ variable!

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:

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.

If you run into the rare case that you have several Point Process instances of the same Point Process group at one and the same node and you need to target only a specific one, use the additional field “ipp” to reference to the specific Point Process. For example “….’target’,struct(‘node’,50,‘ipp’,2)…” makes the connection to the second defined Point Process of this Point Process group at node 50.

record This field is used to define all parameters that should be recorded and returned by T2N. Again, 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 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

Examples:

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

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.

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 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 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:

To increase and decrease g\_pas of cell 3 at node 1 linearly within 100 ms write

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 countings for specific cells. Alternatively, one can record the voltage and count the spikes in Matlab. The data structure is an n-by-2 cell where n 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)

### 1.1.4 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 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 or params.openNeuron

-m let T2N recompile the nrnmech.dll. 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 params.nrnmech!

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 cluster. Cluster parameters need to be specified in params

## 1.2 The output of T2N

### 1.2.1 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 a 1-by-n 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 a n-by-1 cell of vectors with n being the number of used cells.

record This field contains all recorded parameters. Its data structure is a 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 n-by-1 cell array at the 3rd level with n being the highest node where the corresponding parameter was recorded from.

Example

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

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

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

### 1.2.2 The optional cell “minterf”

”minterf“ is m-by-1 cell each containing a matrix for one of the m cells that had been used. The matrix is of size n-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.

### 1.2.3 The optional structure “params”

If not all parameters where defined in the beginning, some parameters were set to their default values. The output structure params contains the modified (i.e. complemented) parameter set in that case. This is normally only interesting for debugging.

### 1.2.4 The optional structure “tree”

Analog, to “params“ the output variable ”tree“ contains the putatively modified tree cell array. 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.

# 2. The t2n\_init\_modelfolder function

This function helps to initialize a standard folder hierarchy for using or creating a model with T2N. It also copies important hoc files with general functions to the (created) folder “lib\_genroutines”

*t2n\_init\_modelfolder(folder)*

**Input**

“*folder*” is a string with the absolute path to the desired model main folder. If this argument is left out, you will be asked to specify the folder with a dialog.

# 3. The t2n\_writetrees function

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 eac 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 the tree names have been specified using the field “tname” in params, then each real 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 “\_”.

tree = t2n\_writetrees(params,tree,savepath,options)

**Input**

”params“ should be the same structure as it will be used as input for the t2n main function, as it contains the information on the folder locations and (if used, the server access data)

“tree” is the TREE cell structure as it will be used for the t2n main function

“savepath” is optional and a 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).

“options” is optional and similar to the t2n main function input, as the same following arguments exist and are simply concatenated to a string:

-w opens a waiting bar showing the progress of the simulation(s)

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

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 cluster. Cluster parameters need to be specified in params

**Output**

“tree” is the TREE cell structure input with the added “NID” fields that contains the corresponding hoc file name.

# 4. The t2n\_as function

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)

**Input**

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

“*neuron*” is the so far defined neuron structure. T2N\_as completes all fields that have not been specified yet in this 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.

# 5. The mech\_tree function

This function is a new extension of the TREES toolbox which can either return (if output is defined) or map the values of a mechanism parameter at each node of each tree as had been specified in the T2N structure "neuron". 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 (params structure needed as 5th input). Supports only one neuron specification at a time.

*[tvec] = mech\_tree(tree, neuron, mech, par[, params])*

**Input**

“*tree*” is a tree structure or cell array of tree structures

“*neuron*” is the model description structure used in t2n

“*mech*” is a string with the name of the mechanism to be investigated

“*par*” is a string with the name of the parameter of mechanism mech to be investigated

“*params*” (optional) is the parameter structure used in t2n, which is then used to search for the mod file of the mechanism

**Output**

“*tvec*” vector or cell array of vectors returning the parameter values at each node and omitting making a figure plot

# 7. Example scripts

Example scripts can be found in the folder “Examples”

“simplet2ntest.m” includes several different simulations, such as single current injection, current injection with increasing amplitudes computed in parallel, and mapping of the backpropagating action potential onto the morphology at different time points

“constant\_current\_injection.m” simulates a long-lasting current injection at the dendritic tips

# 8. Known bugs

1. Server function (T2N option ‘-cl’) not fixed in terms of relative folders