Documentation M2N

Neuron simulations can be executed with matlab using m2n:

out = m2n(tree,params,neuron,options)

# Input parameters:

The four input parameters of m2n are described in the following:

### tree

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

**Non-**artificial cells have a struct according to treestoolbox trees

Artificial cells have a struct with field ‘artificial’ which contains the name of the object class (e.g. IntFire1 or NetStim) and optionally a struct ‘params’ with all parameters that need to be specified (fieldnames are parameter names and field value is parameter value).

### params

*.*access 1 by 2 vector defining cell and node as the standard accessed section. If not defined, the first node of the first non-artificial cell is used

*.*accuracy 0 no change

1 increases nseg in axon and soma by 3 for simulations with fast spiking dynamics or range variables

2 increases nseg everywhere by 3 for simulations with range variables or fast dynamics

*.*custom n by 2 cell with definitions of loading n additional hoc files {‘hocfilename.hoc’,’start/mid/end’}

.dt Time step in ms

.exchfolder relative path where to store temporary files in

.openNeuron boolean if neuron should be opened or only bckgrd

If Neuron gets opened, m2n cannot handle Neuron errors since they are not returned to Matlab!

.path root folder for neuron

.morphfolder relative path where to save cell templates in

.neuronpath absolute path to the NEURON nrniv.exe

.nrnmech if another dll in folder lib\_mech should be loaded then nrnmech.dll, tell it here. 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

.nseg number of segments per section or ‘dlambda’ for using the dlambda rule

.tstart Starting time of simulation in ms, also starting time for recording vectors etc

.tstop End time point of simulation in ms

.v\_init voltage which will be initialized at all cells/DMs

.prerun preruntime in ms to let system settle (afterwards, t is set to tstart)

.skiprun Boolean which determines if NEURON should run or only initialize everything (e.g. for debugging or custom running code)

## neuron

The neuron variable is a struct which contains the parameter set for the simulation (biophys model, stimulations, recordings, etc). It can have length 1 or length x (in the latter, x simulations are run in parallel [on different cores] each with a different parameter set but the same morphologies defined by **tree**)

.mech each cell has a struct containing

1. The name of the region at first level (be sure to use same region names as tree has!)
2. The name of the mechanism to insert at 2nd level
3. A struct with parameter name as field and value as value like “.gnabar=0.05”

This looks like:

neuron.mech{t}.regname.mechname.parname1 = value1

neuron.mech{t}.regname.mechname.parname2 = value2

or like:

neuron.mech{t}.regname.mechname = struct(‘parname1’,value1,’parname2’,value2)

If no parameter twins are specified, the initial parameters of the mechanism code are used

If a mechanism should be introduced to all sections/regions, use “all” as regname (e.g. “neuron.mech{t}.all.mechname1.parname1 = value1” )

If you want to modify range variables more detailed than for each region, you can use “range” as regname. The structure then comprises pairs with the name of the range variable as field name and a nx1 vector with the range variables, where n MUST be the number of nodes in the tree (if you want to put the same value in all segments, use the “all” feature). If nodes should keep their standard parameter, use NaN at these indices.

Example:

neuron.mech{t}.range.mechname = struct(‘alpha’,avec,’beta’,bvec)

with avec and bvec of size n x 1 where tree has n nodes

Note: It is intended to separate the range feature from region-specific insertion, in order not to get confused with nodes specified that are not in this region. It was also separated from the “all” feature to be able to quickly modify range variables without changing the general initialization and settings of the mechanism.

CAUTION: If nseg << # TT nodes per section, it might happen that different TT node values should be written in the same segment. 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.

CAUTION2: If nseg >> # TT nodes per section, the segments which do not have a node near to it are obviously not modified! This might cause confusion if you have a less realistic tree with only very few nodes with huge interdistance and you want to change a parameter in the whole TT section. Either make more regions and simply use the normal mech specification for a region (see above) or resample the tree to a smaller internode distance to avoid segments with no TT node in the surrounding.

CAUTION3: If you have many, many parameters to set, this can produce the infamous error “procedure too big” where there is no simple workaround..

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)

.stim each cell has a m by 3 cell where m is the number of defined electrode sites

values in each row: {“electrodename”, “target node”, struct}

struct is a struct containing these fields:

‘times’: value vector containing the times of changing the amplitude

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

.pp each cell has a m by 3 cell where m is the number of Point Processes

{“PP name”, “target node”, struct}

**Multiple nodes can be input for “target node” but be aware to keep track of number of inserted PPs when you want to record from a specific one**

struct is optional where fields define parameter name of PP parameter and the value gives the value for this PP parameter (e.g. “struct(‘tau’, 0.3) “ )

.con  m by 8 cell where m is the number of defined connections

values in each row:

{sourcespec, sourceid, value to observe (default v) targetspec, targetid, threshold, delay, weight}

Sourcespec can be ‘cell’ , ’node’ or ‘pp’

When connecting a NetStim or IntFire as source, use ‘on’ as value to observe!

Targetspec can be ‘cell’ or ‘pp’

Id after cell is the treenumber of the artificial cell

Id after ‘node’ is treenumber.node , e.g. ‘2.1’.

Id after ‘pp’ is treenumber.ppnumber, e.g. ‘2.4’

Last 3 parameters are optional, if they are not specified, their default values are 10, 1, and 0 respectively.

.record each cell has a m by 2 or 3 cell where m is the number of defined recordings

values in each row: *{index, parameter of interest (‘v’,’i\_pas’ etc), rectype (optional)}*

*rectype* can be ‘node’ (default if empty), ‘pp’, ‘stim’, ‘artificial’

PP is for recording a parameter of a point process

Stim is the same as PP but for IClamp/VClamp/SEClamp electrodes

Artificial is for recording a parameter of an artificial cell

For ‘node’, index is the node index at which NEURON will record

For PP and Stim, index is the x-th PP or Stim which has been defined for that cell

For ‘Artificial’ index is not necessary (can be any number)

.play each cell has a m by 5 cell where m is the number of defined play sites

values in each row: {target node, parameter of interest, time vector, vector to play, continuing Boolean}

.APCount each cell has a m by 2 cell where m is the number of AP counting sites

values in each row: {target node, voltage threshold}

does also work with artificial cells. In this case, netcon is used for recording

## Options

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

-w waitbar

-d Debug mode (NEURON is opened and more parameters are printed)

-q quiet mode -> suppress output and do not open NEURON

-cl cluster mode -> files are prepared to be executed on a cluster. Cluster parameters need to be specified in params

-f force cluster to run neuron directly without qsub

# Output:

## Out

Out contains the recordings which had been defined. If only one simulation was run, out is a struct, if m simulations were run, out is a 1 by m cell with each containing the corresponding output structure.

.t the time vector of the simulation

.record m by 1 cell with the recordings of all m simulated neurons. Each cell is a structure with field names equal to the recorded parameter names. The field contains a cell n by 1 cell structure with n being the highest node where the corresponding parameter was recorded.

E.g. to access the vector of the recorded voltage at node 12 of cell 3 write

out.record{3}.v{12}

.APtimes m by 1 cell with the spiking times of each of the m simulated neurons

The m2n\_as function

**Completes the neuron specification of one simulation with neuron specification of another simulation. Helpful e.g. if you do very similar simulations in parallel where you just want to change one parameter.**

# Input

m2n\_as( as, neuron)

“as” defines the simulation # with which the current neuron specification should be completed with

“neuron” (optional) is the already defined neuron specification. M2n\_as completes all fields that have not been specified yet in neuron

# Output

“thisneuron“ is the completed neuron file

# Example

for s = 1:numsim

if s == 1

neuron{s} = ……. % full specification of all parameters

end

neuron{s}.stim = {s,’IClamp’}; % change node where IClamp is placed each time

neuron{s} = m2n\_as(1,neuron{s}): % define all other parameters as in sim 1

end

**Known bugs**

Server function not fixed in terms of relative folders!