

CSIM : A Neural *C*ircuit *SIM*ulator

Version 1.0

User Manual

©2002 The IGI LSM Group

www.lsm.tugraz.at

February 11, 2004

This document is part of CSIM Release 1.0

Copyright 2002 The IGI LSM group

CSIM is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](http://www.gnu.org/copyleft/gpl.html) as published by the Free Software Foundation; either version 2, or (at your option) any later version.

CSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

To get a copy of the GNU General Public License point your browser to <http://www.gnu.org/copyleft/gpl.html>.

The IGI LSM group
Institute for Theoretical Computer Science
Graz University of Technology
Inffeldgasse 16/b, A-8010 Graz, AUSTRIA
lsm@igi.tu-graz.ac.at, www.lsm.tugraz.at

Contents

1	Preliminaries	5
1.1	What is CSIM ?	5
1.2	About this Manual	5
1.3	Features of the current version	5
1.4	Getting and Installing CSIM	6
2	A short Tutorial	7
2.1	Creating Objects	7
2.2	Object Handles	7
2.3	Setting Fields or Parameters	8
2.4	Making connections	8
2.5	Recording	9
2.6	Setting up the input	9
2.7	Running the simulation	10
2.8	Plotting the recorded traces	10
3	Input and Output	11
3.1	Input Signals	11
3.1.1	Single-stimulus input	11
3.1.2	Multi-stimulus input	12
3.2	Output	12
3.2.1	Getting results of the last simulation	12
3.2.2	Getting the results of a multi-stimulus simulation	13
4	Distributed Simulation	13
5	Additional Topics	13
5.1	How the network is simulated	13
5.2	Global fields	14
5.3	Event driven simulation	15
6	CSIM Commands	15
6.1	Multiple Networks	15
6.1.1	access	16
6.2	Setting up a network simulation	16

6.2.1	create	16
6.2.2	set	16
6.2.3	connect	17
6.3	Running the network simulation	17
6.3.1	reset	17
6.3.2	Single-stimulus simulate	17
6.3.3	Multi-stimulus simulate	18
6.4	Initializing PVM	18
6.4.1	pvm	18
6.5	Saving, loading and deleting networks	18
6.5.1	export	18
6.5.2	import	18
6.5.3	destroy	19
6.6	Displaying/Getting information	19
6.6.1	get	19
6.6.2	list	19
6.6.3	version	20
7	CSIM Classes	21
7.1	AChannel_Hoffman97	21
7.2	AChannel_Korngreen02	21
7.3	ActiveCaChannel	21
7.4	ActiveChannel	21
7.5	AHP_Channel	22
7.6	AlphaSpikeFilter	22
7.7	AnalogFeedbackNeuron	22
7.8	AnalogInputNeuron	23
7.9	AnalogTeacher	23
7.10	ArmModel	23
7.11	CaChannel_Yamada98	24
7.12	CaGate_Yamada98	24
7.13	CALChannel_Destexhe98	24
7.14	CbNeuronSt	25
7.15	CbNeuron	25
7.16	CountSpikeFilter	27

7.17	DiscretizationPreprocessor	27
7.18	DynamicSpikingSynapse	27
7.19	DynamicStdSynapse	28
7.20	ExpSpikeFilter	29
7.21	ExtInputNeuron	29
7.22	ExtOutLifNeuron	29
7.23	ExtOutLinearNeuron	30
7.24	ExtOutSigmoidalNeuron	30
7.25	GaussianAnalogFilter	31
7.26	GVD_cT_Gate	31
7.27	GVD_Gate	31
7.28	HChannel_Stuart98	32
7.29	HH_K_Channel	32
7.30	HH_Na_Channel	32
7.31	HHNeuron	32
7.32	HVACChannel_Brown93	33
7.33	IfbNeuron	33
7.34	Izhi_Neuron	34
7.35	KCACChannel_Mainen96	36
7.36	KChannel_Korngreen02	36
7.37	LifBurstNeuron	36
7.38	LifNeuron	37
7.39	linear_classification	38
7.40	LinearNeuron	38
7.41	LinearPreprocessor	39
7.42	linear_regression	39
7.43	MChannel_Mainen96	39
7.44	MChannel_Wang98	39
7.45	Mean.Std_Preprocessor	40
7.46	MexRecorder	40
7.47	NPChannel_McCormick02	40
7.48	PCAPreprocessor	40
7.49	Readout	41
7.50	Recorder	41
7.51	SICChannel_Maciokas02	42

7.52 SigmoidalNeuron	42
7.53 SpikingInputNeuron	43
7.54 SpikingTeacher	43
7.55 StaticAnalogSynapse	43
7.56 StaticSpikingSynapse	43
7.57 StaticStdpSynapse	44
7.58 TriangularAnalogFilter	44
7.59 UserAnalogFilter	45

1 Preliminaries

1.1 What is CSIM ?

CSIM is a tool for simulating heterogeneous networks composed of different model neurons and synapses. This simulator is written in C++ with an MEX interface to Matlab. It is intended to simulate networks containing up to 10.000 neurons and up to the order of a few millions of synapses (the actual size depends of course on the amount of RAM available on your machine).

1.2 About this Manual

This manual is intended to describe how to use CSIM from the (Matlab) users point of view. It *does not* try to explain (or give an introduction to) the type of models which can be simulated with CSIM. Regarding neural modeling we refer the reader to [Dayan and Abbott, 2001] and [Gerstner and Kistler, 2002]. Furthermore Matlab programming knowledge is assumed.

This manual is also available in [HTML format](#).

1.3 Features of the current version

Different levels of modeling Different neuron models: leaky-integrate-and-fire neurons, compartmental based neurons, sigmoidal neurons. Different synapse models: static synapses and a certain model of dynamic synapses are available for spiking as well as for sigmoidal neurons. Spike time dependent synaptic plasticity is also implemented.

Easy to use Matlab interface Since CSIM is incorporated into Matlab it is not necessary to learn any other script language to set up the simulation. This is all done with Matlab scripts. Furthermore the results of a simulation are directly returned as Matlab arrays and hence any plotting and analysis tools available in Matlab can easily be applied.

Object oriented design We adopted an object oriented design for CSIM which is similar to the approaches taken in [GENESIS](#) and [NEURON](#). That is there are objects (e.g. a `LifNeuron` object implements the standard leaky-integrate-and-fire model) which are

interconnected by means of some signal channels. The creation of objects, the connection of objects and the setting of parameters of the objects is controlled at the level of Matlab scripts whereas the actual simulation is done in the fast C++ core.

Fast C++ core Since CSIM is implemented in C++ and is not as general as GENESIS or NEURON simulations are performed quite fast. We also implemented some ideas from event driven simulators like [SpikeNet](#) which result on an average speedup of 3 (assuming an average firing rate of the neurons of 20Hz and short synaptic time constants) compared to a standard fixed time step simulation scheme.

Runs on Windows and Unix (Linux) CSIM is developed on Linux (SuSE 8.0 with gcc 2.95.3) but it is known also to run under other Linux distributions like Mandrake 8.0 and RedHat 7.2 as well as Windows 98 (we have no experience with Windows XP yet, but it should also run there) and should in principle run on any platform for which Matlab is available.

External Interface There is an external interface which allows CSIM to communicate with external programs. In this way one can for example control the miniature robot [Khepera](#) with CSIM. This feature is not available in the Windows version.

1.4 Getting and Installing CSIM

CSIM is distributed under the [GNU General Public License](#) and can be downloaded from <http://www.igi.tugraz.at/csim>.

To install CSIM perform the following steps:

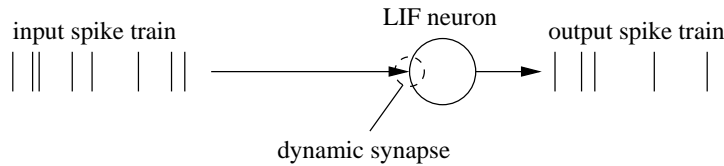
1. Download CSIM from www.igi.tugraz.at/csim
2. Unzip the file `csim-VER.zip` where VER stands for the version you have downloaded.
This will create a subdirectory `lsm` and `lsm/csim`
3. Start Matlab and change into the directory `lsm`
4. Run the Matlab script `install.m`.
5. Add the path `lsm` to the Matlab search path; e. g.
 - `addpath('/home/jack/lsm')}` or
 - `addpath('C:\Work\Neuroscience\lsm')`.
6. Change into the directory `lsm/csim/demos` and play around with them.
7. Have fun using CSIM !

Remark concerning the compilation of CSIM If you do not use Windows or Linux as an operating system `install.m` will try to compile CSIM by calling `mex`. We do not know whether this works on other systems than Windows 98 with MS Visual C++ 6.0 and Linux with gcc 2.95.2. Since CSIM is written in C++ the success of compiling CSIM also depends on the compiler you are using. The [External Interfaces/API](#) section of the Mathworks web site and the [Tech-Note 1601](#) contain valuable information about compiler requirements for compiling C++ MEX files.

Note however that the compiled MEX-files (`csim.mexglx` and `csim.dll`) which already come with CSIM should run on any Linux or Windows operationg system respectively.

2 A short Tutorial

In this section we will introduce CSIM by means of a simple example. We will use CSIM to simulate a model where a [leaky-integrate-and-fire](#) (Sec. 7.38) neuron (LIF neuron) is driven by a Poisson spike train which is transmitted by a [dynamic synapse](#) (Sec. 7.18).



2.1 Creating Objects

Each element/entity of the simple model we will implement, will be simulated by a corresponding *object* in CSIM . The following table shows the correspondence between the elements of the model and the *class* of the object used to simulate the element

element of model	class of CSIM object
input spike train	SpikingInputNeuron
dynamic synapse	DynamicSpikingSynapse
LIF neuron	LifNeuron

Hence, for the simulation to run one must [create objects](#) (Sec. 6.2.1) of the given class:

```
>> i=csim('create','SpikingInputNeuron');
>> s=csim('create','DynamicSpikingSynapse');
>> n=csim('create','LifNeuron');
```

2.2 Object Handles

The values `i`, `s`, and `n` returned by these [create commands](#) (Sec. 6.2.1) are the *indices* or *handles* to the created objects. The handles are the only means to further access or modify existing objects.

2.3 Setting Fields or Parameters

Each of the created objects has several *fields* which usually correspond to a parameter of the model which is implemented by the class of the object. To display for example all [fields of the LifNeuron](#) (Sec. 7.38) object we can use the command

```
>> csim('get',n);
```

This will yield the following output

```
0 : LifNeuron
      Cm = 3e-08 (F)
      Rm = 1e+06 (Ohm)
Vresting = -0.06 (V)
Vreset = -0.06 (V)
Vinit = -0.06 (V)
Trefract = 0.003 (sec)
Inoise = 0 (A)
Iinject = 0 (A)
Vthresh = -0.045 (V)
      Vm : 0 (V)
      type = 0
      nSpikes : 0
      nIncoming : 0
      nOutgoing : 0
```

Some of the fields are parameters of the model (*Cm*, *Rm*, ...) which can be modified (denoted by the '='), others are internal state variables (*Vm* in this case) which can not be changed (denoted by the ':') and yet other provide auxiliary information (*nSpikes*, *nIncoming*, ...). See the [Class Reference](#) (Sec. 7) for details about fields of a particular object.

Suppose we want to change the absolute refractory period of the LIF neuron *n* to be of length 2ms and add a noisy current of 50 nA. This can be done with the command

```
>> csim('set',n,'Trefract',0.002,'Inoise',50e-9);
```

For the dynamic synapse *s* we choose parameters such that it will show a depressing behaviour:

```
>> csim('set',s,'W',10,'U',0.1,'D',1,'F',0.05);
>> csim('set',s,'u0',0.1,'r0',1);
```

2.4 Making connections

So far we have generated three independent objects and set their fields to the desired values. Now we have to connect the objects to implement our simple model. We have to connect the input *i* to the synapse *s* and the synapse to the neuron *n*:

```
>> csim('connect',n,s); % synapse to neuron
>> csim('connect',s,i); % input to synapse
```


Note the the `connect` command (Sec. 6.2.3) uses the convention that the signal destination is the first argument. Alternatively one can use the three argument form of the command:

```
>> csim('connect',n,i,s); % input to neuron via synapse
```

2.5 Recording

The model is now fully implemented. In addition we want to record traces of some quantities of interest. Suppose we want record the membrane potential and the spikes of neuron `n` as well as the postsynaptic responses (the field `psr`) of the synapse. To do so we have to create a `Recorder` object (Sec. 7.50) and tell this object which fields from which objects to record. The recorder is created with the command

```
>> r=csim('create','Recorder');
```

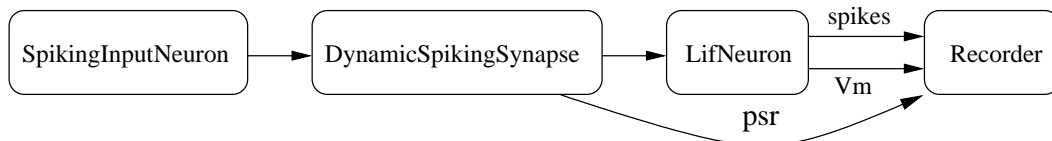
and the following commands tell the recorder `r` to record the field `psr` of synapse `s` as its first trace and the field `Vm` of neuron `n` as its second trace:

```
>> csim('connect',r,s,'psr');
>> csim('connect',r,n,'Vm');
```

Using the special field `spikes` the same syntax will be used to record the spikes of neuron `n` as the third trace of recorder `r`.

```
>> csim('connect',r,n,'spikes');
```

The following figure summarizes which objects we have created and how they are connected:



2.6 Setting up the input

Before we are ready to run the simulation we have to define the spike train which should be emitted by the input neuron `i`. In CSIM time varying *input signals* (Sec. 3.1) (analog or spiking) – also called the *stimulus* – are not considered to be properties/attributes of some objects but are always explicitly specified. In the case of our example we will define a spike train with randomly drawn spike times (for details see Section 3.1):

```
>> S.spiking = 1; % 1 ... spike times, 0 ... analog data
>> S.dt = NaN; % resolution for analog data
>> S.idx = i; % index/handle of receiving object
>> S.data = sort(rand(1,10)); % 10 random spikes in the interval 0 to 1 sec
```

2.7 Running the simulation

Now we are ready to run the simulation. The command

```
>> Tsim=1;
>> csim('simulate',Tsim,S);
```

simulates the simple model for 1 sec starting at time $t = 0$ ¹ with the stimulus S.

2.8 Plotting the recorded traces

The traces recorded by the recorder can be obtained by the command

```
>> t=csim('get',r,'traces')
```

which yields the output

```
t =
    channel: [1x3 struct]
```

A closer look at e.g. the first channel reveals that `t.channel` is a struct array with a similar structure as we have seen above for the input signal:

```
>> t.channel(1)

ans =

    idx: 1
  fieldName: 'psr'
    data: [1x2000 double]
   spiking: 0
      dt: 5.0000e-04
```

The output indicates that `t.channel(1).data` holds the `psr` trace (`t.channel(1).fieldName`) with a resolution of 0.5 ms (`t.channel(1).dt`). See the [Recorder class documentation](#) (Sec. 7.50) for details about the structure of the trace output. Similarly `t.channel(2)` holds the the Vm trace and `t.channel(3)` holds the output spike times. Hence the following command will plot the `psr` trace

```
>> plot(t.channel(1).dt:t.channel(1).dt:Tsim,t.channel(1).data);
```

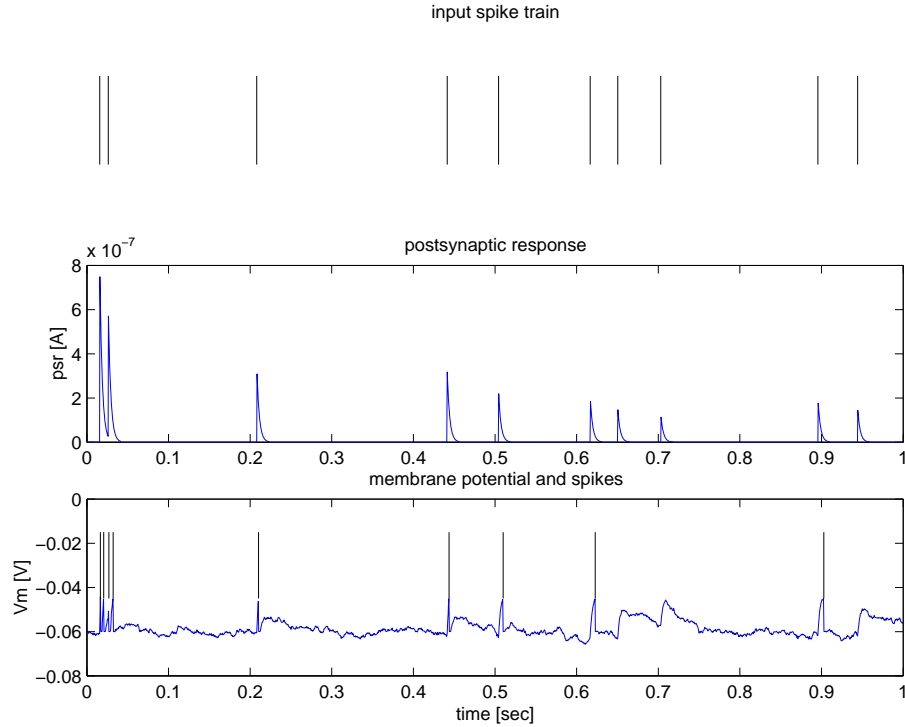
and the command

```
>> stem(t.channel(3).data,ones(size(t.channel(3).data)));
```

¹In general the simulation will be continued at the time where the last `simulate` command (Sec. 6.3.2) stopped. However the first `simulate` command – as in the case of the example – starts at time $t = 0$. Simulation time can be reset to time $t = 0$ with the `reset` command (Sec. 6.3.1) `csim('reset');`.

will draw the output spike train.

Using another set of plot commands one can easily create the following figure which shows the input, the postsynaptic response and the output of the neuron (the voltage and the spikes)



The full code of this example is contained as the file `first_model.m` in the demo directory of the [CSIM package](#).

3 Input and Output

3.1 Input Signals

3.1.1 Single-stimulus input

When running a network simulation via `csim('simulate',...)`; one can specify *input signals* like in the command

```
>> csim('simulate',Tsim,S);
```

where `S` is a struct array with the following fields:

- `S(i).idx` : array of handles of objects which should receives this signal
- `S(i).spiking` : binary flag (0/1) which determines if `S(i).data` should be interpreted as spike times or as an analog signal

- `S(i).dt` : time discretization; for analog signals (`S(i).spiking=0`) only
- `S(i).data` : signal data: vector of the analog values (`S(i).spiking=0`) or spike times (`S(i).spiking=1`)

Note that `csim('simulate',...)` accepts an arbitrary number of such struct arrays. See [documentation of the simulate command](#) (Sec. 6.3.2).

3.1.2 Multi-stimulus input

When running a network simulation via `csim('simulate',...)`; one can specify a *multi-stimulus* input `S` like in the command

```
>> R = csim('simulate',Tsim,S);
```

where `S` is a $n \times 1$ struct array of the form `S(i).channel` which describes the i -th stimulus. `S(i).channel` has the structure as described above.

If such multi-stimulus input is specified CSIM will simulate the current network with each of the n stimuli where at the beginning of each stimuli the network is reset to $t = 0$. The return argument `R` contains the recorded information of all n simulations (see [Section 3.2.2](#)).

3.2 Output

If one simulates a network then one also wants to record the quantities of interest. In our [introductory example](#) (Sec. 2) we were interested in the membrane potential of the leaky-integrate and fire neuron. CSIM can record any field of any object by means of [Recorder](#) (Sec. 7.50) objects. The following code fragment shows how to set up a Recorder to record the membrane potential (`Vm`) of a [LifNeuron](#) (Sec. 7.38) object with handle `n`.

```
>> rec = csim('create','Recorder');
>> csim('connect',rec,n,'Vm');
```

Note that one Recorder object can record an arbitrary number of fields from arbitrary objects.

3.2.1 Getting results of the last simulation

After a network simulation via a command like

```
>> csim('simulate',Tsim,InputSignal);
```

the recorded data of the *the last simulation* Recorder `rec` can be obtained by the command

```
>> R=csim('get',rec,'traces');
```

The exact structure of `R` for the recorder `rec` depends on the value of the [field commonChannels of the Recorder](#) (Sec. 7.50) object.

`commonChannels = 0` In this case (default) `R` is a struct array with the only field `channel` which is in turn a struct array with a similar structure as an [input signal](#) (Sec. 3.1). That is

- `R.channel(j).idx` : handle of the object from which field the data was recorded
- `R.channel(j).spiking` : binary flag (0/1) which determines if `data` should be interpreted as spike times or as an analog signal
- `R.channel(j).dt` : time discretization; for analog signals only
- `R.channel(j).data` : signal data : vector of the analog values or spike times. Note that the data always starts at time $t = 0$.
- `R.channel(j).fieldName` : name of the recorded field

`commonChannels = 1` In this case `R` has two fields:

- `R.data` : A double array where `R.data(j,s)` is the s -th recorded value of the j -th field recorded. Note that the data always starts at time $t = 0$. Remark: The number of values which are recorded depends on the [field dt of the Recorder object](#) (Sec. 7.50).
- `R.info` : A struct array where `R.info(j).idx` is the handle of the object from which the field `R.info(j).fieldName` is recorded.

3.2.2 Getting the results of a multi-stimulus simulation

Reminder: The command

```
>> R=csim('get',rec,'traces');
```

returns only the results of the last simulation. In the case of a multi-stimulus simulation with stimulus array `S` of length n this command returns only the result of the simulation with stimulus `S(n)`.

To get the results of all n simulations one has to use

```
>> R=csim('simulate',Tsim,S);
```

In that case `R` contains the results of all n simulations. `R{r}(s).channel` contains the recorded data of the r -th recorder during the s -th simulation ($1 \leq s \leq n$). `R{r}(s).channel` is of the format as described in [Section 3.2.1](#).

4 Distributed Simulation

5 Additional Topics

5.1 How the network is simulated

CSIM employs a fixed time step simulation scheme for the integration of the differential equations involved (represented by objects). Currently all objects use the exponential Euler

method with the same integration time step (see [Section 5.2](#)).

During each time step a certain method (**advance**) is called for each object part of the simulation. There is a fixed schedule which depends a) on the class of the object and b) on the time of creation. Here is the order of the classes.

1. Neuron
2. Synapse
3. Recorder

This means that all objects of classes derived from a generic Neuron are advanced first followed by objects of classes derived from a generic Synapse and so on.

Note that for example a more detailed neuron model (like the [CbNeuron](#) (Sec. 7.15)) has to advance all its child objects like ion channels.

After a `csim('simulate',...)` command returns the network is kept in its current state at this time t . A further `csim('simulate',...)` command continues the simulation at this time t . Use `csim('reset');` to reset the simulation to time $t = 0$.

Note that recorded traces you get by the appropriate [get command](#) (Sec. 6.6.1) always start at time $t = 0$.

5.2 Global fields

There are a few fields which determine the global (i.e. not object specific) behavior of CSIM which can be modified via the [set command](#) (Sec. 6.2.2):

```
>> csim('set', <fieldname>, <value>);
```

Read/writeable fields

dt The integration time step used by all objects

randSeed The seed value for the random number generator

verboseLevel How much stuff should be written to the console window (no effect yet!).

nThreads Number of threads to use for simulation (no effect yet!). The multi-threaded version of CSIM is currently under development.

spikeOutput Flag: 1 ... always output spikes as last cell element, 0 do not output spikes (obsolete).

The following fields are read-only:

t The current virtual (simulation) time

step The current time step; i.e. $\text{step} = t / dt$;

5.3 Event driven simulation

Since in a typical simulation the firing rate of the neurons is about 30 Hz and the time constant of a synaptic current is typically about 3ms most of the time a synapse is “idle”. This observation encouraged us to implement some ideas of event driven simulators like [SpikeNet²](#): We decided to cut off the very far tail of the exponential decaying synaptic responses (after 5 time constants) and remove those synapses from the list of active synapses (which are advanced every time step). A synapse is re-activated if it receives a further spike. For large networks with lots of synapses these idea results in an average speed-up of 2 to 3.

6 CSIM Commands

The common form of all CSIM commands is one of the following two

```
csim(command,...);  
csim(netIdx,command,...);
```

where `netIdx` is the index of the network to apply the command and `command` is one of the following strings:

- `'create'` (Sec. 6.2.1) : Create an object of a certain class
- `'set'` (Sec. 6.2.2) : Set fields of an object
- `'connect'` (Sec. 6.2.3) : Connect objects
- `'get'` (Sec. 6.6.1) : Get field values of an object
- `'access'` (Sec. 6.1.1) : Set default network
- `'reset'` (Sec. 6.3.1) : Reset the simulation to $t = 0.0$
- `'simulate'` (Sec. 6.3.2) : Simulate the network
- `'export'` (Sec. 6.5.1) : Export the network for saving it
- `'import'` (Sec. 6.5.2) : Import a loaded network
- `'destroy'` (Sec. 6.5.3) : Destroy/delete the current network
- `'list'` (Sec. 6.6.2) or `'ls'` (Sec. 6.6.2) List various items
- `'version'` (Sec. 6.6.3) : Print out a version string
- `'pvm'` (Sec. 6.4.1) : Initialize distributed simulation

A detailed description of each of the commands follows.

6.1 Multiple Networks

CSIM is capable of holding more than one network at a time. There are two ways of specifying on which network a command should operate.

- For each command of the form `csim(command,...);` the form `csim(netIdx,command,...);` exists which allows you to explicitly specify the network to operate on.
- If `csim(command,...);` is used the *default network* is used. The default network is set using the command `csim('access',netIdx);`.

²<http://www.cnl.salk.edu/~arno/spikenet/>

6.1.1 access

- `idx=csim('access',netIdx)`; makes the network specified by `netIdx` the default network and returns that value.
- `csim('access',netIdx)`; makes the network specified by `netIdx` the default network and prints that value.
- `idx=csim('access')`; returns the index of the default network.
- `csim('access')`; prints the index of the default

6.2 Setting up a network simulation

6.2.1 create

- `idx=csim('create','Network')`; creates a new network and returns the *index* or *handle* for the created network. The new network is the new default network; see command [access](#) (Sec. 6.1.1).
- `idx=csim('create',class)`; creates an object of the class specified by the string `class` and returns a `uint32` *index* or *handle* for the created object. This index/handle is the only means to access the created object.
- `idx=csim('create',class,N)`; creates `N` objects of the class specified by the string `class` and returns a `uint32` vector of indices/handles for the created objects.

The handles returned by `idx=csim('create',...)`; are used to identify the objects later in 'set', 'get' and 'connect' calls.

See the [Class Reference](#) (Sec. 7) for a full description of all available classes.

6.2.2 set

- `csim('set',field,value)`; sets the [network field](#) (Sec. 5.2) specified by the string `field` to `value`.
- `csim('set',idx,field,value)`; sets the field specified by the string `field` of the object with index/handle `idx` to `value`. `idx` can also be a vector of indices/handles of objects of the same class.
If `idx` is a vector and `value` is a scalar then the fields of all objects are set to `value`.
If `idx` and `value` are vectors of the same size then the field `field` of all objects `idx(i)` is set to `value(i)`.
- `csim('set',idx,f1,v1,f2,v2,...,fn,vn)`; sets the fields specified by the strings `f1`, `f2`, ..., `fn` of the object with index/handle `idx` to the values `v1`, `v2`, ... `vn`. `idx` and `v1`, `v2`, ... `vn` can also be vectors.

6.2.3 connect

- `csim('connect',dstIdx,srcIdx)`; sets up a signal flow from the source `srcIdx` object (e.g. a synapse) to the destination object `dstIdx` (e.g. the postsynaptic neuron) where `dstIdx` and `srcIdx` are indices/handles to objects.

`dstIdx` and `srcIdx` can also be vectors. In that case for each `i=1:length(srcIdx)`; a signal flow `dstIdx(i) ← srcIdx(i)` is set up.

- `csim('connect',dstIdx,srcIdx,viaIdx)`; for each `i=1:length(srcIdx)`; a signal flow of the form `dstIdx(i) ← viaIdx(i) ← srcIdx(i)` is set up where `dstIdx`, `srcIdx`, and `viaIdx` are vectors of the same length of indices/handles to objects.
- `csim('connect',recIdx,objIdx,fieldName)`; connects the object with handle `objIdx` to the recorder with handle `recIdx`. The recorder `recIdx` will then record a trace of the values of the field `fieldName` of the objects specified by the vector of handles `objIdx`.

6.3 Running the network simulation

6.3.1 reset

`csim('reset')`; resets the simulation time t back to $t = 0.0$.

6.3.2 Single-stimulus simulate

- `csim('simulate',Tsim,InputSignals)`; runs the network simulation for `Tsim` seconds starting at the time where the last simulate command stopped with [input signals InputSignals](#) (Sec. 3.1).
- `csim('simulate',Tsim,I1,I2,...,In)`; same as above but with input signals `I1` to `In`. There is no special meaning in the ordering of the input signals. It is equivalent to concatenate `I1` to `In` into one struct array and pass this array as the single input signal.
- `R=csim('simulate',Tsim,InputSignals)`;
same as `csim('simulate',Tsim,InputSignals)`; but in addition returns the cell array `R` which holds the [output \(traces\) of all Recorder objects](#) (Sec. 3.2).
- `R=csim('simulate',Tsim,I1,I2,...,In)`;
same as `csim('simulate',Tsim,I1,I2,...,In)`; but in addition returns the cell array `R` which holds the [output \(traces\) of all Recorder objects](#) (Sec. 3.2).
- `R=csim('simulate',Tsim,I1,I2,...,In)`;
same as `csim('simulate',Tsim,I1,I2,...,In)`; but in addition returns the cell array `R` which holds the [output \(traces\) of all Recorder objects](#) (Sec. 3.2).

6.3.3 Multi-stimulus simulate

`R=csim('simulate',Tsim,S);` runs n network simulations for `Tsim` sec. The network is reset to $t = 0$ before each simulation.

The n simulations are determined by the $n \times 1$ struct array `S` where `S(i).channel` specifies the stimulus for the i -th simulation (see [Section 3.1.2](#)).

`R` contains the reordred values of all n simulations (see [Section 3.2.2](#)).

If the [distributed simulation](#) (Sec. 4) is on (see command `pvm` (Sec. 6.4.1)), the n simulations will be carried out in parallel.

6.4 Initializing PVM

6.4.1 pvm

6.5 Saving, loading and deleting networks

6.5.1 export

The command

```
net = csim('export');
```

produces a struct array `net` which contains all information about the current network simulation which is needed to set up the simulation. Using `save file.mat net` you can save the whole network simulation.

NOTE: The struct array `net` returned by `net=csim('export');` is only intended to be saved to disk for later use by `csim('import',net);` and does not have a human readable form.

6.5.2 import

The command

```
csim('import',net);
```

allows you to set up a network simulation from the struct array `net` which has previously been generated by `net = csim('export');`. In combination with the matlab command `load` this CSIM command serves the purpose of loading a simulation from disk.

If the current network (either explicitly specified or default network) does not contain any objects the `net` will be imported into that network. If there are already objects a new network will be created and `net` will be imported to that.

NOTE: The struct array `net` returned by `net=csim('export');` *does not* contain information about the current state of the simulation at time t . Hence it is only possible to start simulation based on such an array at time $t = 0.0$.

6.5.3 destroy

`csim('destroy');` all the networks.

6.6 Displaying/Getting information

6.6.1 get

- `csim('get');` displays the values of the [global fields](#) (Sec. 5.2)
- `v=csim('get',field);` return the value of the [global field](#) (Sec. 5.2) specified by the string `field`
- `csim('get',idx);` displays the values of all fields of the object specified by the array `index/handle idx`
- `v=csim('get',idx,field);` returns a double array `v` where `v(:,i)` contains the values of the field `field` of the object with `index/handle idx(i)`.
- `o=csim('get',idx,'struct');` return a struct array describing the object specified by the `index/handle idx(1)`:
 - `o.className` : String containing the name of class of object
 - `o.spiking` : Double value which is 1 if the object is a spike emitting object (0 otherwise)
 - `o.fields` : Cell array of strings containing the field names of the object
 - and all fields with its values.
- `[incomming,outgoing]=csim('get',idx,'connections');` returns the indices/handles of the incomming (i.e. input delivering) and outgoing (i.e. output receiving) objects of the object specified by the `index/handle idx`. This is currently only implemented for Synapses and Neurons.
- `R=csim('get',recorder_idx,'traces');` returns the traces of the fields recorded by the [Recorder](#) (Sec. 7.50) object with handle `recorder_idx` during the *last simulation*. See the [Recorder documentation](#) (Sec. 7.50) for details about the format of `R`.

6.6.2 list

- `csim('list');` and `csim('list','classes');` print a list of all available classes
- `csim('list','classes','-fields');` prints a list of all available classes with information about the fields of each class. Read/writeable fields are marked by a '=' between the field name and its description, whereas a ':' denotes a read-only field.
- `csim('list','objects');` prints a list of all created objects

- `csim('list','objects','-fields');` prints a list of all created objects and the values of its fields. Read/writeable fields are marked by a '=' between the field name and its value, whereas a ':' denotes a read-only field.
- `csim('list','networks');` prints a list of all networks
- `csim('list','networks','-fields');` prints a list of all networks and the values of its fields. Read/writeable fields are marked by a '=' between the field name and its value, whereas a ':' denotes a read-only field.

Not that instead of 'list' one can use 'ls' as a shortcut.

6.6.3 version

- `csim('version')` prints a version string
- `v=csim('version');` returns the version string

7 CSIM Classes

7.1 AChannel_Hoffman97

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.2 AChannel_Korngreen02

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.3 ActiveCaChannel

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.4 ActiveChannel

The Model

Generic active channel with arbitray number of ion gates . $g(t, V_m)$ is computed as

$$g(t, V_m) = \bar{g} \prod_i P_i(t, V_m)$$

where \bar{g} is the maximal conductance, and $P_i(t, V_m) \in [0, 1]$ is fraction of i -type gates currently open.

Note that no specific gates are modeled. Instead arbitrary [ion gate objects](#) can be connected to the channel.

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;
Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.5 AHP_Channel

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;
u : Constant step increase in n for each spike;
Ts : Time constant for the deactivation of the current;
Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

n : Fraction of the open conductance;
g (S) : The conductance $g(t, V_m)$ of the channel

7.6 AlphaSpikeFilter

Read/writable Fields

m_tau1 : Time constant 1.
m_tau2 : Time constant 2.

Readonly Fields

nChannels : Number of Input channels to filter.
m_dt : Time step length.
nValuesPerChannel : Number of values stored for each channel in lastValues.

7.7 AnalogFeedbackNeuron

Read/writable Fields

feedback : Feedback-Mode: 0 = external input, 1 = feedback
Vresting (V) : The resting membrane voltage.
Inoise (A^2) : The noise of analog neurons
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The current output (potential) of this neuron
VmOut : The value which will actually be propagated to the outgoing synapses.
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.8 AnalogInputNeuron

Read/writable Fields

Vresting (V) : The resting membrane voltage.
Inoise (A^2) : The noise of analog neurons
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The current output (potential) of this neuron
VmOut : The value which will actually be propagated to the outgoing synapses.
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.9 AnalogTeacher

7.10 ArmModel

Read/writable Fields

mintheta1 : Baseline parameter for theta1.
mintheta2 : Baseline parameter for theta2.
minU1 : Baseline parameter for U1.
minU2 : Baseline parameter for U2.
model_DT : The time-step of simulation.
Xo : The X coordinate of origin.
Yo : The Y coordinate of origin.
m1 : The mass of 1st link.
m2 : The mass of second link.
lc1 : The distance of central point of link 1.
lc2 : The distance of central point of link 2.
l1 : Length of first link.
l2 : Length of second link.
I1 : The moment of inertia of link1.
I2 : The moment of inertia of link2.
MFACT : The factor by which the external noise will be proportional to the input magnitude.
PERT_TIME : The time at which the perturbation will start.
DURATION : The duration of perturbation.
inputFileNr : The file-number for input of Xdest, Ydest, Theta1, Theta2 (file-name is 'T_{ixj}.mat', where ix_j is this number).

Readonly Fields

t1 : Angle theta 1.
t2 : Angle theta 2.
w1 : Angular velocity 1 (updated every time step).
w2 : Angular velocity 2 (updated every time step).
u1 : Torque 1 (updated every time step).
u2 : Torque 2 (updated every time step).
nIncoming : Number of inputs
nOutputChannels : Number of output channels

7.11 CaChannel_Yamada98

Read/writable Fields

u (*Mol*) : Constant step increase in the calcium concentration
Ts (*Sec*) : Time constant for the deactivation of the calcium concentration
Ca (*Mol*) : Interior calcium concentration
Gbar (*S*) : The maximum conductance of the channel;
Erev (*V*) : The reversal potential E_{rev} of the channel

Readonly Fields

g (*S*) : The conductance $g(t, V_m)$ of the channel

7.12 CaGate_Yamada98

Read/writable Fields

Ts : Scale of the time constant $\tau(Ca)$
k (1) : The exponent of the gate
p : The state variable.

Readonly Fields

P (1) : The output $P(t, V) = p(t, V)^k$ of the gate.

7.13 CALChannel_Destexhe98

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$
Gbar (*S*) : The maximum conductance of the channel;
Erev (*V*) : The reversal potential E_{rev} of the channel

Readonly Fields

$g(S)$: The conductance $g(t, V_m)$ of the channel

7.14 CbNeuronSt

Missing!

Read/writable Fields

STempHeight (*Volt*) : Height

Vthresh (*V*) : If V_m exceeds V_{thresh} a spike is emitted.

Vreset (*V*) : The voltage to reset V_m to after a spike.

doReset (*flag*) : Flag which determines whether V_m should be reset after a spike

Trefract (*sec*) : Length of the absolute refractory period.

type : Type (e.g. inhibitory or excitatory) of the neuron

Cm (*F*) : The membrane capacity C_m

Rm (*Ohm*) : The membrane resistance R_m

Vresting (*V*) : The resting membrane voltage.

Vinit (*V*) : Initial condition for V_m at time $t = 0$.

VmScale (*V*) : Defines the difference between $V_{resting}$ and the V_{thresh} for the calculation of the iongate tables and the ionbuffer E_{rev} .

Inoise (W^2) : Variance of the noise to be added each integration time constant.

Iinject (*A*) : Constant current to be injected into the CB neuron.

Readonly Fields

Em (*V*) : The reversal potential of the leakage channel

Vm (*V*) : The membrane voltage

STempIdx :

nStepsInRefr : If positive then this counter tells us how many time steps we are still in the absolute refractory period.

C1 : Internal constants for the exponential Euler integration of V_m .

nIncoming : Number of incoming synapses

nOutgoing : Number of outgoing synapses

nBuffers : Number of ion buffers

nChannels : Number of channels

7.15 CbNeuron

The Model

The membrane voltage V_m is governed by

$$C_m \frac{dV_m}{dt} = -\frac{V_m - E_m}{R_m} - \sum_{c=1}^{N_c} g_c(t)(V_m - E_{rev}^c) + \sum_{s=1}^{N_s} I_s(t) + I_{inject}$$

with the following meanings of symbols

- C_m membrane capacity (Farad)
- E_m reversal potential of the leak current (Volts)
- R_m membrane resistance (Ohm)
- N_c total number of channels (active + synaptic)
- $g_c(t)$ current conductance of channel c (Siemens)
- $E_{rev}^c(t)$ reversal potential of channel c (Volts)
- N_s total number of current supplying synapses
- $I_s(t)$ current supplied by synapse s (Ampere)
- I_{inject} injected current (Ampere)

At time $t = 0$ V_m is set to V_{init} .

The value of E_m is calculated to compensate for ionic currents such that V_m actually has a resting value of $V_{resting}$.

Spiking and resetting the membrane voltage

If the membrane voltage V_m exceeds the threshold V_{thresh} the CbNeuron sends a spike to all its outgoing synapses.

The membrane voltage is reset and clamped during the absolute refractory period of length $T_{refract}$ to V_{reset} if the flag `doReset=1`. This is similar to a LIF neuron (see [LifNeuron](#)).

If the flag `doReset=0` the membrane voltage is not reset and the above equation is also applied during the absolute refractory period but the event of threshold crossing is transmitted as a spike to outgoing synapses. This is useful if one includes channels which produce a *real* action potential (see [HH_K_Channel](#) and [HH_Na_Channel](#)) but one still just wants to communicate the spikes as events in time.

Implementation

The exponential Euler method is used for numerical integration.

Read/writable Fields

Vthresh (V) : If V_m exceeds V_{thresh} a spike is emitted.

Vreset (V) : The voltage to reset V_m to after a spike.

doReset (flag) : Flag which determines whether V_m should be reset after a spike

Trefract (sec) : Length of the absolute refractory period.

type : Type (e.g. inhibitory or excitatory) of the neuron

Cm (F) : The membrane capacity C_m

Rm (Ohm) : The membrane resistance R_m

Vresting (V) : The resting membrane voltage.

Vinit (V) : Initial condition for V_m at time $t = 0$.
VmScale (V) : Defines the difference between $V_{resting}$ and the V_{thresh} for the calculation of the iongate tables and the ionbuffer E_{rev} .
Inoise (W^2) : Variance of the noise to be added each integration time constant.
Iinject (A) : Constant current to be injected into the CB neuron.

Readonly Fields

Em (V) : The reversal potential of the leakage channel
Vm (V) : The membrane voltage
nStepsInRefr : If positive then this counter tells us how many time steps we are still in the absolute refractory period.
C1 : Internal constants for the exponential Euler integration of V_m .
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses
nBuffers : Number of ion buffers
nChannels : Number of channels

7.16 CountSpikeFilter

Read/writable Fields

time_window : Length of time window.

Readonly Fields

nChannels : Number of Input channels to filter.
m_dt : Time step length.
nValuesPerChannel : Number of values stored for each channel in lastValues.

7.17 DiscretizationPreprocessor

Readonly Fields

nInputRows : Number of rows for input vectors.
nOutputRows : Number of rows for output vectors.

7.18 DynamicSpikingSynapse

The time varying state $x(t)$ of the synapse is increased by $W \cdot r \cdot u$ when a presynaptic ** spike hits the synapse and decays exponentially (time constant ** τ) otherwise. u and r model the current ** state of facilitation and depression.

Read/writable Fields

U : The use parameter of the dynamic synapse
D (sec) : The time constant of the depression of the dynamic synapse
F (sec) : The time constant of the facilitation of the dynamic synapse
u0 : Value of the time varying facilitation state variable u for the first spike
r0 : Value of the time varying depression state variable r for the first spike
tau (sec) : The synaptic time constant τ
W : The weight (scaling factor, strenght, maximal amplitude) of the synapse
delay (sec) : The synaptic transmission delay

Readonly Fields

u : The time varying state variable u for facilitation
r : The time varying state variable u for depression
psr : The psr (postsynaptic response) is the result of whatever computation is going on in a synapse.
steps2cutoff :

7.19 DynamicSdpSynapse

Read/writable Fields

U : The use parameter of the dynamic synapse
D (sec) : The time constant of the depression of the dynamic synapse
F (sec) : The time constant of the facilitation of the dynamic synapse
u0 : Value of the time varying facilitation state variable u for the first spike
r0 : Value of the time varying depression state variable r for the first spike
back_delay (sec) : The minimum value of $\Delta t = t_{post} - t_{pre}$ which should be considered for STDP.
tauspost :
tauspre :
taupos :
tauneg :
dw :
STDPgap :
activeSTDP :
useFroemkeDanSTDP :
Wex : The maximal/minimal weight of the synapse
Aneg :
Apos :
mupos :
muneg :
tau (sec) : The synaptic time constant τ
W : The weight (scaling factor, strenght, maximal amplitude) of the synapse
delay (sec) : The synaptic transmission delay

Readonly Fields

u : The time varying state variable u for facilitation
r : The time varying state variable u for depression
psr : The psr (postsynaptic response) is the result of whatever computation is going on in a synapse.
steps2cutoff :

7.20 ExpSpikeFilter

Read/writable Fields

m_tau1 : Time constant.

Readonly Fields

nChannels : Number of Input channels to filter.
m_dt : Time step length.
nValuesPerChannel : Number of values stored for each channel in lastValues.

7.21 ExtInputNeuron

Read/writable Fields

Vresting (V) : The resting membrane voltage.
Inoise (A^2) : The noise of analog neurons
type : Type (e.g. inhibitory or excitatory) of the neuron
myIndex : ID of external input (is equal index of array)
beReal : Flag if external input is used or normal input

Readonly Fields

Vm (V) : The current output (potential) of this neuron
VmOut : The vlaue wich will actualle be propagated to the outgoing synapses.
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.22 ExtOutLifNeuron

Read/writable Fields

myIndex : ID of external input (is equal index of array)
beReal : Flag if external input is used or normal input
Cm (F) : The membrane capacity C_m
Rm (Ohm) : The membrane resistance R_m
Vthresh (V) : If V_m exceeds V_{thresh} a spike is emmited.
Vresting (V) : The resting membrane voltage.

Vreset (V) : The voltage to reset V_m to after a spike.
Vinit (V) : The initial condition for V_m at time $t = 0$.
Trefract (sec) : The length of the absolute refractory period.
Inoise (A) : The standard deviation of the noise to be added each integration time constant.
Iinject (A) : A constant current to be injected into the LIF neuron.
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The membrane voltage V_m
Isyn : synaptic input current
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.23 ExtOutLinearNeuron

Read/writable Fields

Iinject (W) : external current injected into neuron
Vinit (V) : initial 'membrane voltage'
Vresting (V) : The resting membrane voltage.
Inoise (A^2) : The noise of analog neurons
type : Type (e.g. inhibitory or excitatory) of the neuron
myIndex : ID of external input (is equal index of array)
beReal : Flag if external input is used or normal input

Readonly Fields

Vm (V) : The current output (potential) of this neuron
VmOut : The value which will actually be propagated to the outgoing synapses.
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.24 ExtOutSigmoidalNeuron

Read/writable Fields

thresh (W) : $Itot = \text{logsig}(\text{beta} * (Itot - \text{thresh}))$
beta (1) : $Itot = \text{logsig}(\text{beta} * (Itot - \text{thresh}))$
tau_m : time constant
A_max (1) : the output of a sig neuron is scaled to (0, W_{max})
Iinject (W) : external current injected into neuron
Vm_init (V) : initial 'membrane voltage'
Vresting (V) : The resting membrane voltage.
Inoise (A^2) : The noise of analog neurons
type : Type (e.g. inhibitory or excitatory) of the neuron
myIndex : ID of external input (is equal index of array)
beReal : Flag if external input is used or normal input

Readonly Fields

Vm (*V*) : The current output (potential) of this neuron
VmOut : The value which will actually be propagated to the outgoing synapses.
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.25 GaussianAnalogFilter

Read/writable Fields

m_std_dev : Standard deviation of the filter mask.
nKernelLength : The length of the kernel.

Readonly Fields

nChannels : Number of Input channels to filter.

7.26 GVD_cT_Gate

Read/writable Fields

Vh (*Volt*) : Inflection point of the sigmoidal function $p(V)$
Vc (*Volt*) : Slope of the sigmoidal function $p(V)$
Ts : Voltage independent time constant τ
k (1) : The exponent of the gate
p : The state variable.

Readonly Fields

P (1) : The output $P(t, V) = p(t, V)^k$ of the gate.

7.27 GVD_Gate

Read/writable Fields

Vh (*Volt*) : Inflection point of the sigmoidal function $p(V)$
Vc (*Volt*) : Determines the slope of the sigmoidal function $p(V)$
Ts : Scale of the time constant $\tau(V)$
Te : Additional exponential factor of the time constant $\tau(V)$
k (1) : The exponent of the gate
p : The state variable.

Readonly Fields

P (1) : The output $P(t, V) = p(t, V)^k$ of the gate.

7.28 HChannel_Stuart98

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.29 HH_K_Channel

Uses the gate [HH_n_Gate](#)

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.30 HH_Na_Channel

Uses the gates [HH_h_Gate](#) and [HH_m_Gate](#)

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.31 HHNeuron

The model is based on a [CbNeuron](#) and includes the [HH_K_Channel](#) and [HH_Na_Channel](#) for action potential generation.

Read/writable Fields

Vthresh (V) : If V_m exceeds V_{thresh} a spike is emitted.
Vreset (V) : The voltage to reset V_m to after a spike.
doReset ($flag$) : Flag which determines whether V_m should be reset after a spike
Trefract (sec) : Length of the absolute refractory period.
type : Type (e.g. inhibitory or excitatory) of the neuron
Cm (F) : The membrane capacity C_m
Rm (Ohm) : The membrane resistance R_m
Vresting (V) : The resting membrane voltage.
Vinit (V) : Initial condition for V_m at time $t = 0$.
VmScale (V) : Defines the difference between $V_{resting}$ and the V_{thresh} for the calculation of the iongate tables and the ionbuffer E_{rev} .
Inoise (W^2) : Variance of the noise to be added each integration time constant.
Iinject (A) : Constant current to be injected into the CB neuron.

Readonly Fields

Em (V) : The reversal potential of the leakage channel
Vm (V) : The membrane voltage
nStepsInRefr : If positive then this counter tells us how many time steps we are still in the absolute refractory period.
C1 : Internal constants for the exponential Euler integration of V_m .
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses
nBuffers : Number of ion buffers
nChannels : Number of channels

7.32 HVACChannel_Brown93

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$
Gbar (S) : The maximum conductance of the channel;
Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.33 IfbNeuron

see Izhikevich E. "Which model to use for cortical spiking neurons?"

Model

A leaky-integrate-and-fire-or burst neuron model is implemented where the membrane potential V_m of a neuron is given by

$$\tau_m \frac{dV_m}{dt} = -(V_m - V_{resting}) + R_m \cdot (I_{syn}(t) + I_{inject} + I_{noise})$$

where $\tau_m = C_m \cdot R_m$ is the membrane time constant, R_m is the membrane resistance, $I_{syn}(t)$ is the current supplied by the synapses, I_{inject} is a non-specific background current and I_{noise} is a Gaussian random variable with zero mean and a given variance noise.

At time $t = 0$ V_m is set to V_{init} . If V_m exceeds the threshold voltage V_{thresh} it is reset to V_{reset} and hold there for the length $T_{refract}$ of the absolute refractory period.

Implementation

The exponential Euler method is used for numerical integration.

Read/writable Fields

Cm (F) : The membrane capacity C_m
Rm (Ohm) : The membrane resistance R_m
Vthresh (V) : If V_m exceeds V_{thresh} a spike is emitted.
Vresting (V) : The resting membrane voltage.
Vreset (V) : The voltage to reset V_m to after a spike.
Vinit (V) : The initial condition for V_m at time $t = 0$.
Trefract (sec) : The length of the absolute refractory period.
Inoise (A) : The standard deviation of the noise to be added each integration time constant.
Iinject (A) : A constant current to be injected into the LIF neuron.
h : internal parameter h - deinactivation of Ca-channels
tau_p : tau_p time constant for recovery from inactivation
tau_m : tau_m time constant for inactivation
Vh : Vh constant for onset of Ca-channel activation.
gh : gh constant for relative impact of Ca-current to leak current (0.5-2)
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The membrane voltage V_m
Isyn : synaptic input current
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.34 Izhi_Neuron

see Izhikevich E. "Simple model of Spiking Neurons"

Model

A canonical neuron model is implemented where the membrane potential V_m of a neuron is given by

, R_m is the membrane resistance, $I_{syn}(t)$ is the current supplied by the synapses, I_{inject} is a non-specific background current and I_{noise} is a Gaussian random variable with zero mean and a given variance noise.

At time $t = 0$ V_m is set to V_{init} . If V_m exceeds the threshold voltage V_{thresh} it is reset to V_{reset} and hold there for the length $T_{refract}$ of the absolute refractory period.

Implementation

The simple Euler method is used for numerical integration.

Read/writable Fields

Cm (F) : The membrane capacity C_m
Rm (Ohm) : The membrane resistance R_m
Vthresh (V) : If V_m exceeds V_{thresh} a spike is emitted.
Vresting (V) : The resting membrane voltage.
Vreset (V) : The voltage to reset V_m to after a spike.
Vinit (V) : The initial condition for V_m at time $t = 0$.
Trefract (sec) : The length of the absolute refractory period.
Inoise (A) : The standard deviation of the noise to be added each integration time constant.
Iinject (A) : A constant current to be injected into the LIF neuron.
a : A constant (0.02, 01) describing the coupling of variable u to V_m .
b : A constant controlling sensitivity of u.
c : A constant controlling reset of V_m ($1000 \cdot V_{reset}$).
d : A constant controlling reset of u.
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The membrane voltage V_m
Vb : The membrane voltage in millivolt.
Vint : The membrane voltage in millivolt.
u : internal variable
ub : The internal variable u adapted to millivolt and millisecond.
Isyn : synaptic input current
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.35 KCACHannel_Mainen96

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$
Gbar (S) : The maximum conductance of the channel;
Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.36 KChannel_Korngreen02

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$
Gbar (S) : The maximum conductance of the channel;
Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.37 LifBurstNeuron

that produces bursts dependent on two additional variables analog to the implementation of dynamical synapses: see Kaske&Bertschinger "Traveling wave patterns..."

Model

A nonstandard leaky-integrate-and-fire neuron model is implemented where the membrane potential V_m of a neuron is given by

$$\tau_m \frac{dV_m}{dt} = -(V_m - V_{resting}) + R_m \cdot (I_{syn}(t) + I_{inject} + I_{noise})$$

where $\tau_m = C_m \cdot R_m$ is the membrane time constant, R_m is the membrane resistance, $I_{syn}(t)$ is the current supplied by the synapses, I_{inject} is a non-specific background current and I_{noise} is a Gaussian random variable with zero mean and a given variance noise. V_{reset} is modulated by u and r to produce spiking dependent on the previous spike pattern At time $t = 0$ V_m is set to V_{init} . If V_m exceeds the threshold voltage V_{thresh} it is reset to V_{reset} and hold there for the length $T_{refract}$ of the absolute refractory period.

Implementation

The exponential Euler method is used for numerical integration.

Read/writable Fields

Cm (F) : The membrane capacity C_m
Rm (Ohm) : The membrane resistance R_m
Vthresh (V) : If V_m exceeds V_{thresh} a spike is emitted.
Vresting (V) : The resting membrane voltage.
Vreset (V) : The default voltage to reset V_m to after a spike.
VBthresh (V) : spike pattern dependent threshold V_m to after a spike.
Vinit (V) : The initial condition for V_m at time $t = 0$.
Trefract (sec) : The length of the absolute refractory period.
Inoise (A) : The standard deviation of the noise to be added each integration time constant.
Iinject (A) : A constant current to be injected into the LIF neuron.
uB : internal parameter uB - recovery from spike facilitation with time constant FB
rB : internal parameter rB - recovery from spike depression with time constant DB
FB : FB time constant for recovery from facilitation of spike generation.
DB : DB time constant for recovery from depression of spike generation.
UB : UB constant for onset of facilitation of spike generation.
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The membrane voltage V_m
Isyn : synaptic input current
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.38 LIFNeuron

Model

A standard leaky-integrate-and-fire neuron model is implemented where the membrane potential V_m of a neuron is given by

$$\tau_m \frac{dV_m}{dt} = -(V_m - V_{resting}) + R_m \cdot (I_{syn}(t) + I_{inject} + I_{noise})$$

where $\tau_m = C_m \cdot R_m$ is the membrane time constant, R_m is the membrane resistance, $I_{syn}(t)$ is the current supplied by the synapses, I_{inject} is a non-specific background current and I_{noise} is a Gaussian random variable with zero mean and a given variance noise.

At time $t = 0$ V_m is set to V_{init} . If V_m exceeds the threshold voltage V_{thresh} it is reset to V_{reset} and hold there for the length $T_{refract}$ of the absolute refractory period.

Implementation

The exponential Euler method is used for numerical integration.

Read/writable Fields

Cm (F) : The membrane capacity C_m
Rm (Ohm) : The membrane resistance R_m
Vthresh (V) : If V_m exceeds V_{thresh} a spike is emitted.
Vresting (V) : The resting membrane voltage.
Vreset (V) : The voltage to reset V_m to after a spike.
Vinit (V) : The initial condition for V_m at time $t = 0$.
Trefract (sec) : The length of the absolute refractory period.
Inoise (A) : The standard deviation of the noise to be added each integration time constant.
Iinject (A) : A constant current to be injected into the LIF neuron.
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The membrane voltage V_m
Isyn : synaptic input current
nIncoming : Number of incoming synapses
nOutgoing : Number of outgoing synapses

7.39 linear_classification

Read/writable Fields

nClasses : Number of Classes.
range_low : Lower bound of algorithms range.
range_high : Upper bound of algorithms range.

Readonly Fields

nInputRows : Number of rows for input vectors.

7.40 LinearNeuron

Read/writable Fields

Iinject (W) : external current injected into neuron
Vinit (V) : initial 'membrane voltage'
Vresting (V) : The resting membrane voltage.
Inoise (A^2) : The noise of analog neurons
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The current output (potential) of this neuron

VmOut : The value which will actually be propagated to the outgoing synapses.

nIncoming : Number of incoming synapses

nOutgoing : Number of outgoing synapses

7.41 LinearPreprocessor

Readonly Fields

nInputRows : Number of rows for input vectors.

nOutputRows : Number of rows for output vectors.

7.42 linear_regression

Read/writable Fields

range_low : Lower bound of algorithms range.

range_high : Upper bound of algorithms range.

Readonly Fields

nInputRows : Number of rows for input vectors.

7.43 MChannel_Mainen96

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.44 MChannel_Wang98

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

$g(S)$: The conductance $g(t, V_m)$ of the channel

7.45 Mean_Std_Preprocessor

Readonly Fields

nInputRows : Number of rows for input vectors.

nOutputRows : Number of rows for output vectors.

7.46 MexRecorder

Read/writable Fields

commonChannels : Flag: 1 ... output all channels in one matrix (WARNING: no spikes are returned yet), 0 ... output each recorded field as separate channel

dt (sec) : The timestep at which an recording should be done (no meaning if recording spikes).

Tprealloc (sec) : Provide your best guess how long the network will be simulated (in simulation time).

enabled : Flag: 0 ... recorder disabled, 1 ... recorder enabled

7.47 NPChannel_McCormick02

Read/writable Fields

Ts : Scale of the time constants of the gating variables $\tau(V)$

Gbar (S) : The maximum conductance of the channel;

Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

$g(S)$: The conductance $g(t, V_m)$ of the channel

7.48 PCAPreprocessor

Readonly Fields

nInputRows : Number of rows for input vectors.

nOutputRows : Number of rows for output vectors.

7.49 Readout

This class implements readouts within the simulation. Readouts take the responses of the liquid and calculate a new function with these inputs. Readouts can be trained to approximate some target function. Readouts can be connected to physical models or feedback neurons for closed-loop simulations. Therefore the interface of readouts is similar to that of synapses. The output of a readout can be recorded.

Read/writable Fields

enabled : Flag: 0 ... readout disabled, 1 ... readout enabled
offset : Offset of output values.
range : Range of output values.

Readonly Fields

output : Output variable of a readout.

7.50 Recorder

This class allows you to record traces of any fields of any object during the simulation. The recorded traces of an Recorder with handle **rec_idx** can be obtained via

```
R=csim('get',rec_idx,'traces');
```

The exact form of **R** depends on the flag **commonChannels** (see below). Note that the traces returned always start at time $t=0$ and are recorded at an interval of **dt**.

In addition a recorder can also record spikes from spike emitting objects. Via a command like

```
csim('connect',rec_idx,neuron_idx,'spikes');
```

the Recorder with handle **rec_idx** is set up to record the spikes from the spike emitting objects with handles **neuron_idx**.

commonChannels=0 In this case the Matlab array **R** is a struct array with the only field **channel** which is in turn a struct array with the following fields:

- **R.channel(j).idx** : handle of the object from which field the data was recorded
- **R.channel(j).spiking** : binary flag (0/1) which determines if **data** should be interpreted as spike times or as an analog signal
- **R.channel(j).dt** : time discretization; for analog signals only
- **R.channel(j).data** : signal data : vector of the analog values or spike times.

- `R.channel(j).fieldName` : name of the recorded field

commonChannels=1 In this case `**R` has two fields (WARNING: no spikes are returned):

- `R.data` : A double array where `R.data(j,s)` is the `s`-th recorded value of the `j`-th field.
- `R.info` : A struct array where `R.info(j).idx` is the handle of the object from which the field `R.info(j).fieldName` is recorded.

Read/writable Fields

commonChannels : Flag: 1 ... output all channels in one matrix (WARNING: no spikes are returned yet), 0 ... output each recorded field as separate channel
dt (sec) : The timestep at which an recording should be done (no meaning if recording spikes).
Tprealloc (sec) : Provide your best guess how long the network will be simulated (in simulation time).
enabled : Flag: 0 ... recorder disabled, 1 ... recorder enabled

7.51 SICChannel_Maciokas02

Read/writable Fields

Gbar (S) : The maximum conductance of the channel;
Erev (V) : The reversal potential E_{rev} of the channel

Readonly Fields

g (S) : The conductance $g(t, V_m)$ of the channel

7.52 SigmoidalNeuron

Read/writable Fields

thresh (W) : $Itot = \text{logsig}(\text{beta} * (Itot - \text{thresh}))$
beta (1) : $Itot = \text{logsig}(\text{beta} * (Itot - \text{thresh}))$
tau_m : time constant
A_max (1) : the output of a sig neuron is scaled to (0, W_max)
I_inject (W) : external current injected into neuron
Vm_init (V) : initial 'membrane voltage'
Vresting (V) : The resting membrane voltage.
Inoise (A^2) : The noise of analog neurons
type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

Vm (V) : The current output (potential) of this neuron

VmOut : The value which will actually be propagated to the outgoing synapses.

nIncoming : Number of incoming synapses

nOutgoing : Number of outgoing synapses

7.53 SpikingInputNeuron

Read/writable Fields

type : Type (e.g. inhibitory or excitatory) of the neuron

Readonly Fields

nIncoming : Number of incoming synapses

nOutgoing : Number of outgoing synapses

7.54 SpikingTeacher

7.55 StaticAnalogSynapse

Read/writable Fields

Inoise : The noise of our analog synapses

W : The weight (scaling factor, strength, maximal amplitude) of the synapse

delay (sec) : The synaptic transmission delay

Readonly Fields

psr : The psr (postsynaptic response) is the result of whatever computation is going on in a synapse.

7.56 StaticSpikingSynapse

We call a synapse a *static* synapse if the amplitude of each postsynaptic response is equal. Here we implement a synaptic response $x(t)$ of the form $x(t) = W \cdot \exp(-t/\tau)$ for each spike which hits the synapse at time t with an amplitude of W and a decay time constant of τ . The responses of all the spikes are added up linearly.

Read/writable Fields

tau (sec) : The synaptic time constant τ

W : The weight (scaling factor, strength, maximal amplitude) of the synapse

delay (sec) : The synaptic transmission delay

Readonly Fields

psr : The psr (postsynaptic response) is the result of whatever computation is going on in a synapse.
steps2cutoff :

7.57 StaticStdpSynapse

Read/writable Fields

back_delay (*sec*) : The minimum value of $\Delta = t_{post} - t_{pre}$ which should be considered for STDP.
tauspost :
tauspre :
taupos :
tauneg :
dw :
STDPgap :
activeSTDP :
useFroemkeDanSTDP :
Wex : The maximal/minimal weight of the synapse
Aneg :
Apos :
mupos :
muneg :
tau (*sec*) : The synaptic time constant τ
W : The weight (scaling factor, strenght, maximal amplitude) of the synapse
delay (*sec*) : The synaptic transmission delay

Readonly Fields

psr : The psr (postsynaptic response) is the result of whatever computation is going on in a synapse.
steps2cutoff :

7.58 TriangularAnalogFilter

Read/writable Fields

nKernelLength : The length of the kernel.

Readonly Fields

nChannels : Number of Input channels to filter.

7.59 UserAnalogFilter

Read/writable Fields

m_weights : User-defined weights.

nKernelLength : The length of the kernel.

Readonly Fields

nChannels : Number of Input channels to filter.

References

- [Dayan and Abbott, 2001] Dayan, P. and Abbott, L. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press. See also <http://people.brandeis.edu/~abbott/book/>.
- [Gerstner and Kistler, 2002] Gerstner, W. and Kistler, W. (2002). *Spiking Neuron Models*. Cambridge University Press. See also <http://diwww.epfl.ch/~gerstner/BUCH.html>.