SpikeNet Documentation

Yifan Gu
Complex System Group, School of Physics, USYD
yigu8115@gmail.com

June 27, 2016

1 Overview

SpikeNet is a software developed for simulating spiking neuronal networks, of which the design provides the following four features.

Configurability SpikeNet supports any user-defined structure of synaptic connectivity topologies, coupling strengths and conduction delays. It can be easily extended by developers to support any variations of integrate-and-fire neuron and synapse models. For the models that are currently available, see Neuron and synapse models.

Performance Simulation of spiking neuronal network quickly becomes computationally intensive if the number of neurons N in the network exceeds a few thousand. To achieve superior performance, various measures have been taken at both algorithmic and implementation level. Notably, at the algorithmic level, kinetic models are exclusively chosen for their algorithmic efficiency (e.g., [2, 4, 6, 7]). Furthermore, for such kinetic models, computational cost of the simulation can be dramatically reduced with the commonly used mathematical abstraction that synapses can be simplified into a few groups, within which their dynamics (or more specifically, the time constants) are identical [1]. At the implementation level, C++, a programming language renowned for its high-performance computing, is used.

User-friendly interface In SpikeNet, although, C++ is used for heavy-duty computation, its user-interface is written in a high-level programming language (Matlab) for user-friendliness. This means SpikeNet does not require non-developer users to be familiar with C++. In practise, the typical work-flow of SpikeNet user is as following.

1. In Matlab, use SpikeNet functions to generate the input files (plain text) that define the simulation case.
2. Evoke the SpikeNet C++ simulator, which reads the input files, runs the simulation and generates the output files.

3. In Matlab, use SpikeNet functions to parse the output files (plain text) into Matlab data for post-processing.

Please see Matlab user interface for more detailed descriptions of the workflow. Developer users will be interested to know that the data format of the aforementioned input and output files follows pre-defined protocols (see Output protocols and Input protocols). This design essentially makes the SpikeNet C++ simulator a stand-alone software. Based on these protocols, the C++ simulator can easily interface with any other high-level programming languages if Matlab is not prefered.

Scalability The design of the SpikeNet C++ simulator readily supports parallel computing used Message Passing Interface (MPI). The simulator has two central C++ classes, representing populations of neurons and synaptic connectivities among these populations, respectively. For example, for a recurrent network consisting of an excitatory neuron population (E) and an inhibitory one (I), two neuron population objects will be created together with up to four synaptic connectivity objects including $E \leftarrow E$, $I \leftarrow E$, $E \leftarrow I$ and $I \leftarrow I$. These six objects can be created and simulated in parallel and the amount of message passing among them at each simulation time step is minimal.

2 Tech Stack

- C++11 standard is required (GCC 4.2.1 or later; Intel C++ 12.0 or later).
- Matlab is optional but highly recommended.
- Portable Batch System (PBS) is optional but highly recommended.
- Message Passing Interface (MPI) implementation is optional.

3 Quickstart

Following are the steps to set up the SpikeNet C++ simulator.

- Ask for read permission from one of the contributors with admin rights.
- Make a new directory: `mkdir tmp; cd tmp`
- Clone SpikeNet: `git clone git@github.com:BrainDynamicsUSYD/SpikeNet.git`
- Go into the directory: `cd SpikeNet`
- Build the C++ simulator: `make; make clean; cd ..; ls`

Now you should see the "simulator" in the current directory, with which you can run simulations by creating input files according to Input protocols. However, it will be much easier to work with the Matlab user interface. If you do not have access to

Matlab, try to contact the contributors to request interfaces with other high-level programming languages (Python for example). Following are the steps to use the Matlab user interface.

- Make a new directory for storing data: `mkdir tmp_data`
- Start Matlab: `matlab -nodisplay`
- Set up the environment for Matlab: `cd SpikeNet; addpath(genpath(cd));`
- Generate the example input files: `cd ../tmp_data; main_demo;`
- Quit Matlab: `quit`
- Run the simulator with the input files: `cd tmp_data; ../simulator *ygin;`
- Start Matlab: `cd ..; matlab -nodisplay`
- Set up the environment for Matlab: `cd SpikeNet; addpath(genpath(cd));`
- Parse the output files, run some basic post-processing and visualization: `cd ../tmp_data; PostProcessYG()`
- Load the simulation result: `d = dir(`*RYG.mat'); R = load(d(1).name)` (You may need to correct the single quotes in Matlab if you are directly copying the code from here.)

For those who have access to a high-performance computing cluster with PBS, SpikeNet also provides bash script that fully automates the above Matlab → C++ → Matlab workflow for PBS job array submission.

- Go to the tmp directory
- Make a copy of the script: `cp SpikeNet/*sh.bak all_in_one.sh`
- Change it to executable: `chmod +x all_in_one.sh`
- Edit the following variables in the bash script accordingly:
  - MATLAB_SOURCE_PATH_2='your_path'
  - MATLAB_PRE_PROCESS_FUNC='your_functions'
- Make a directory for PBS output: `mkdir PBSout`
- Submit the job: `qsub -t 1-X -q queue_name all_in_one.sh`

For MPI jobs with SpikeNet, please contact Yifan Gu for more technical details.

Here are some basic tips on git.

- To get the most recent version of SpikeNet: `git pull origin master`

- Avoid directly modifying any existing files in SpikeNet unless you are a developer. Create another directory alongside SpikeNet and put all of your files there. In summary, your project folder should look like:

```
tmp
├── SpikeNet
├── simulator
├── tmp_data
├── my_functions
├── PBS_out
└── all_in_one.sh
```

4 Neuron and synapse models

A description of the neuron and synapse models can also be found in a publication that used SpikeNet [3]. The default neuron model in SpikeNet is leaky integrate-and-fire neuron. The membrane potential V_i^α of the i-th neuron ($i = 1, \cdots, N_\alpha$) from population α evolves according to

$$C_m \frac{dV_i^\alpha}{dt} = -g_L(V_i^\alpha - V_L) + I_{i,syn}^\alpha(t) + I_{i,app}^\alpha(t), \text{ if } V_i^\alpha < V_{th}. \tag{4.1}$$

When neurons reach the threshold V_{th}, a spike is emitted and they are reset to V_{rt} for an absolute refractory period τ_{ref}. The spike times t_i^α are recorded.

The default synapse model in SpikeNet is conductance-based. The synaptic currents received by a neuron are given by

$$I_{i,syn}^\alpha(t) = \sum_{\beta=1}^{P} I_i^{\alpha\beta}(t) \tag{4.2}$$

$$= \sum_{\beta=1}^{P} [-g_i^{\alpha\beta}(V_i^\alpha - V_{rev}^\beta)] \tag{4.3}$$

$$= \sum_{\beta=1}^{P} \{-[\sum_{j=1}^{N_\beta} a_{ij}^{\alpha\beta} g_{ij}^{\alpha\beta} s_{ij}^{\alpha\beta}(t)](V_i^\alpha - V_{rev}^\beta)\} \tag{4.4}$$

where V_{rev}^β is the reversal potential of the corresponding current $I_i^{\alpha\beta}$ induced by pre-synaptic population β. $a_{ij}^{\alpha\beta}$ is a binary variable which determines the existence of synapse from the j-th neuron in population β to the i-th neuron in population α, while $g_{ij}^{\alpha\beta}$ reflects the (maximal) strength of the synaptic conductance. The gating variable $s_{ij}^{\alpha\beta}(t)$ models the instananeous value of synaptic conductance in terms of the fraction of open channels, described by

$$\frac{ds_{ij}^{\alpha\beta}}{dt} = -\frac{s_{ij}^{\alpha\beta}}{\tau_d^\beta} + \sum_{t_j^\beta} h^\beta(t - t_j^\beta - d_{ij}^{\alpha\beta})(1 - s_{ij}^{\alpha\beta}) \tag{4.5}$$

where h models the concentration time-course of the channel-opening neurotransmitters, arrived with a conduction delay $d_{ij}^{\alpha\beta}$ after the pre-synaptic spike time t_j^{β}. The $(1 - s_{ij}^{\alpha\beta})$ term introduces saturation effect. Following the simplification in [1], a rectangular pulse with unitary area is used for h

$$h(t) = \begin{cases} 1/\tau_r, & \text{if } 0 \leq t \leq \tau_r \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

All numerical values are in consistant units unless mentioned otherwise (ms for time, mV for voltage, nA for current, nF for capacitance and μS for conductance). Default numerical integration is performed using Euler method and the suggested time-step is 0.1 ms [5].

5 Matlab user interface

See main_demo.m for a list of the basic case-building matlab functions and their usages. Each such function implements one particular input protocol.

```
function main_demo()
% Use coherent units (msec+mV+nF+miuS+nA) unless otherwise stated


%%%% Seed the Matlab random number generator
seed = 1;

%%%% Open new (uniquely time-stamped) input files for the SpikeNet C++
% simulator.
% FID is the main input file, FID_syn is the input file with the
% synaptic connectivity definitions (could be very large).
[FID, FID_syn] = new_ygin_files_and_randseed(seed);
% If no FID_syn is needed, use FID = new_ygin_files_and_randseed(seed,0)

%%%% Define some basic parameters
% Time step (ms)
dt = 0.1; % 0.1 is usually small enough
% Total number of simulation steps
step_tot = 10^2;
% Create two neuron populations with 50 neurons in the 1st population
% and 10 neurons in the 2nd population.
N = [40, 10];
% Write the above basic parameters to the input file
writeBasicPara(FID, dt, step_tot, N);


%%%% Define non-parameters for the neuorn model
% For demo purpose, the values used as following are the still the
% default ones.
% If default values are to be used, there is no need to re-define them.
Cm = 0.25; % (nF) membrane capacitance
tau_ref = 2.0; % (ms) absolute refractory time
V_rt = -60.0;   % (mV) reset potential
V_lk = -70.0;   % (mV) leaky reversal potential
```

```
V_th = -50.0;   % (mV) firing threshold
g_lk = 0.0167;   % (miuS) leaky conductance (note that Cm/gL=15 ms)
for pop_ind = 1:2
    writePopPara(FID, pop_ind,'Cm',Cm,'tau_ref',tau_ref,'V_rt',V_rt,...
        'V_lk',V_lk,'V_th',V_th,'g_lk',g_lk);
end

%%%% Add spike-frequency adaptation to the 1st population
pop = 1;
writeSpikeFreqAdpt(FID, pop);


%%%% Define the initial condition
p_fire = [0.1 0.1]; % initial firing probabilities for both populations
% set initial V distribution to be [V_rt, V_rt + (V_th-V_rt)*r_V0]
r_V0 = [0.5 0.5];
writeInitCond(FID, r_V0, p_fire)

%%%% Add external Poissonian spike trains into the 1st population
pop = 1;
type_ext = 1; % 1 for AMPA-like syanpses
g_ext = 2*10^-3; % synaptic coupling strength (miuS)
N_ext = 1000; % No. of independent external connections onto each neuron
rate_ext = 2*ones(1, step_tot); % Poisson rate for each time step (Hz)
N_start = 1;
N_end = 10; % only add to neuron No.1-10.
writeExtSpikeSettings(FID, pop, type_ext, g_ext,  N_ext, rate_ext,...
    N_start, N_end );


%%%% Add external currents (Gaussian white noise) to the 2nd population
pop = 2;
I_ext_mean = 0.5*ones(1,N(2)); % defined for each neuron (nA)
I_ext_std = 0.2*ones(1,N(2)); % defined for each neuron (nA)
writeExtCurrentSettings(FID, pop, I_ext_mean, I_ext_std)

%%%% Define runaway killer
% The computational cost of the simulation is directly proportional to
% the avearage firing rate of the network. Very often, your parameters
% may lead to biologically unrealistically high firing rate or even
% runaway activity (the highest rate possible ~1/tau_ref). To save the
% computational resources for more interesting simulation cases, it is
% advised to define the runawary killer.
% Note that the simulator will still output all the relevant data before
% the kill.
min_ms = 500; % the min simulation duration that should be guaranteed
runaway_Hz = 40; % the threshold above which the simu should be killed
Hz_ms = 200; % the window length over which the firing rate is averaged
pop = 1; % the population to be monitored by the runaway killer
writeRunawayKiller(FID, pop, min_ms, runaway_Hz, Hz_ms);


%%%% Record the basic statistics for the 1st neuron population
pop = 1;
writePopStatsRecord(FID, pop);

%%%%%%%%%%%%%%%%%%% Chemical Connections %%%%%%%%%%%%%%%%%%%%%%%
% type(1:AMAP, 2:GABAa, 3:NMDA)
```

```
% Define AMAP-like excitatory synaptic connection wtihin the 1st pop
pop_pre = 1;
pop_post = 1;
syn_type = 1; % 1 for AMPA-like synapse
g_EE = 10*10^-3; % synaptic coupling strength (miuS)
% random adjacency matrix with a wiring probability of 0.2
A = rand(N(1), N(1)) < 0.2;
[I_EE, J_EE] = find(A);
K_EE = g_EE*ones(size(I_EE)); % identical coupling strength
D_EE = rand(size(I_EE))*5; % uniformly random conduction delay [0, 5] ms
writeChemicalConnection(FID_syn, syn_type,  pop_pre, pop_post, ...
    I_EE, J_EE, K_EE, D_EE);

% Define AMAP-like excitatory synaptic connection from pop 1 to pop 2
pop_pre = 1;
pop_post = 2;
syn_type = 1; % 1 for AMPA-like synapse
g_IE = 10*10^-3; % synaptic coupling strength (miuS)
% random adjaceny matrix with a wiring probability of 0.5
A = rand(N(1), N(2)) < 0.5;
[I_IE, J_IE] = find(A);
K_IE = g_IE*ones(size(I_IE)); % identical coupling strength
D_IE = rand(size(I_IE))*5; % uniformly random conduction delay [0, 5] ms
writeChemicalConnection(FID_syn, syn_type,  pop_pre, pop_post, ...
    I_IE, J_IE, K_IE, D_IE);

% Define GABA-like excitatory synaptic connection from pop 2 to pop 1
pop_pre = 2;
pop_post = 1;
syn_type = 2; % 2 for GABA-like synapse
g_EI = 10*10^-3; % synaptic coupling strength (miuS)
% random adjaceny matrix with a wiring probability of 0.5
A = rand(N(2), N(1)) < 0.5;
[I_EI, J_EI] = find(A);
K_EI = g_EI*ones(size(I_EI)); % identical coupling strength
D_EI = rand(size(I_EI))*5; % uniformly random conduction delay [0, 5] ms
writeChemicalConnection(FID_syn, syn_type,  pop_pre, pop_post, ...
    I_EI, J_EI, K_EI, D_EI);

% Define GABA-like excitatory synaptic connection within pop 2
pop_pre = 2;
pop_post = 2;
syn_type = 2; % 2 for GABA-like synapse
g_II = 20*10^-3; % synaptic coupling strength (miuS)
% random adjaceny matrix with a wiring probability of 0.5
A = rand(N(2), N(2)) < 0.5;
[I_II, J_II] = find(A);
K_II = g_II*ones(size(I_II)); % identical coupling strength
D_II = rand(size(I_II))*5; % uniformly random conduction delay [0, 5] ms
writeChemicalConnection(FID_syn, syn_type,  pop_pre, pop_post, ...
    I_II, J_II, K_II, D_II);
%%%%%%%%%%%%%%%%%%% Chemical Connections Done %%%%%%%%%%%%%%%%%%%%%


%%%% Use an existing synapse connectivity definition file instead of
% generating a new one:
% writeSynFilename(FID, 'path/to/existing/XXX.ygin_syn')
% In this case, you should use
```

```
% "FID = new_ygin_files_and_randseed(seed, 0)"
% to avoid creating an empty ygin_syn file.



%%%% Define non-default synapse parameters
writeSynPara(FID, 'tau_decay_GABA', 3);
% "help writeSynPara" to see all the parameter names and default values

%%%% Add short-term depression to the synapses with the 1st population
pop_pre = 1;
pop_post = 1;
step_start = round(step_tot/3); % Turn on STD at this time step
writeSTD(FID, pop_pre, pop_post, step_start);


%%%% Add inhibitory STDP to the synapses from pop 2 to pop 1
pop_pre = 2;
pop_post = 1;
% Turn on inhibitory STDP at a certain time step
step_start = round(step_tot/3*2);
writeInhSTDP(FID, pop_pre, pop_post, step_start);

%%%% Record the basic statistics for the excitatory synapses within
% the 1st neuron population
pop_pre = 1;
pop_post = 1;
syn_type = 1; % 1 for AMPA-like synapse
writeSynStatsRecord(FID, pop_pre, pop_post, syn_type);

%%%% Sample detailed time serious data from the 1st population
pop = 1;
sample_neuron = 1:10:N(1); % sample every 10th neuron
sample_steps = 1:2:step_tot; % sample every 2nd time step
sample_data_type = logical([1,1,1,1,0,0,1,0]);
% The logical vector above corresponds to
% [V,I_leak,I_AMPA,I_GABA,I_NMDA,I_GJ,I_ext, I_K]
writeNeuronSampling(FID, pop, sample_data_type, ...
    sample_neuron, sample_steps)

%%%% Optional: record explanatory variables (scalars only)
% They will also be used in pro-processsing for auto-generated comments
discard_transient = 0; % transient period data to be discarded (ms)
writeExplVar(FID, 'discard_transient', discard_transient, ...
    'g_EE', g_EE, ...
    'g_EI', g_EI, ...
    'g_IE', g_IE, ...
    'g_II', g_II, ...
    'g_ext', g_ext, ...
    'N_ext', N_ext, ...
    'rate_ext', rate_ext);


%%%% Additional comments to be passed to the post-processing stage
comment1 = 'This is a demo.';
comment2 = ['If you have any question regarding SpikeNet,'...
    'please contact Yifan Gu (yigu8115@gmail.com).'];
writeExplVar(FID, 'comment1', comment1, 'comment2', comment2);
```

```
209
210 %%%% append this matlab file to the input file for future reference
211 appendThisMatlabFile(FID)
212
213 end
214
215
216
217 function appendThisMatlabFile(FID)
218 breaker = ['>',repmat('#',1,80)];
219 fprintf(FID, '%s\n', breaker);
220 fprintf(FID, '%s\n', '> MATLAB script generating this file: ');
221 fprintf(FID, '%s\n', breaker);
222 Fself = fopen([mfilename('fullpath'),'.m'],'r');
223 while ~feof(Fself)
224     tline = fgetl(Fself);
225     fprintf(FID, '%s\n', tline);
226 end
227 fprintf(FID, '%s\n', breaker);
228 fprintf(FID, '%s\n', breaker);
229 fprintf(FID, '%s\n', breaker);
230 end
```

(More need to be added on the available post-processing Matlab functions. SpikeNet has a variety of statistical analysis and visualization tools for spiking networks.)

++Add descriptions of CollectVectorYG and CollectCellYG.

6 Input protocols

The input files are plain text files. The default input filename is "filename.ygin". It is advised that synapse definitions always be given in a separate file. The default synapse definition filename is "filename.ygin_syn". Non-default synapse definition file path and name can be specified by "SYNF001" protocol. Following is the complete list of the input data protocols. Note that the protocols use three special characters, including ">" for starting a new protocol, "#" for commenting and "," for delimiting.

```
1
2  > INIT001 # number of neurons in each population
3      N1, N2, ...,
4
5  > INIT002 # time step length and total number of steps
6      dt (ms), step_tot,
7
8  > INIT003 # random initial distributions for V (deprecated)
9      p_fire_pop1 (range (−∞,1]), ..., p_fire_popN,
10
11 > INIT004 # external Gaussian currents
12     pop_ind,
13     mean_1 (nA), mean_2, ..., mean_N, (for each neuron)
14     std_1 (nA), std_2, ..., std_N,
```

```
15
16 > INIT005 # external Poissonian spikes
17     pop_ind, type_ext, K_ext (μS), Num_ext, ia, ib,
18     rate_1 (Hz), rate_2, ..., rate_step_tot
19
20 > INIT006 # chemical connection definition
21     type, pop_ind_pre, pop_ind_post,
22     I (row vector),
23     J (row vector),
24     K (row vector),
25     D (row vector),
26
27 > INIT007 # set perturbation (remove one spike)
28     pop_ind, step_perturb,
29
30 > INIT008 # add short-term depression
31     pop_ind_pre, pop_ind_post, STD_on_step,
32
33 > INIT009 # add inhibitory STDP
34     pop_ind_pre, pop_ind_post, STDP_on_step,
35
36 > INIT010 # add spike-frequency adaptation
37     pop_ind,
38
39 > INIT011 # random initial conditions for V and firing prob
40     r_V0_pop1 (range (0,1]), ...,r_V0_popN,
41     p_fire_pop1 (range (0,1]), ..., p_fire_popN,
42
43 > SYNF001 # non-default synapse definition file name
44     path/to/file_name (no comma!)
45
46 > KILL001 # runaway killer setting
47     pop_ind, min_ms, runaway_Hz, Hz_ms,
48
49 > PARA001 # non-default neuron population parameter
50     pop_ind, number_of_parameters,
51     parameter_name1, value1,
52     parameter_name2, value2,
53     ...
54
55 > PARA002 # non-default synapse parameter
56     number_of_parameters,
57     parameter_name1, value1,
58     parameter_name2, value2,
59     ...
60
61 > SAMP001 # neuron data sampling
62     pop_ind,
63     data_type (logical vector),
64     ind1, ind2, ..., indX, (sample neuron indices)
```

```
    1, 1, 0, 0, ..., (1-by-step_tot logical values)
    # Note that data_type specifies sample data types
    # and it must correspond to
    # [V,I_leak,I_AMPA,I_GABA,I_NMDA,I_GJ,I_ext]

> SAMP002 # synapse data sampling
    pop_ind_pre, pop_ind_post, syn_type,
    ind1, ..., indX, (post-synaptic sample neuron indices)
    1, 1, 0, 0, ..., (1-by-step_tot logical values)

> SAMP003 # neuron population statistical data record
    pop_ind,

> SAMP004 # synaptic currents  statistical data record
    pop_ind_pre, pop_ind_post, syn_type,

> SAMP005 # local field potential data record
    pop_ind, No_of_measures
    1, 0, 0, 1, ...,
    ...
    0, 0, 1, 1, ..., (No_of_measures-by-N logical values)
```

7 Output protocols

The output files are plain text files. The default output filename is "filename-X.ygout", where X is a time-stamp that uniquely marks the output file so that multiple simulations can be run for the same input files. For data completeness, the corresponding input file will be attached to the output file. Following is the complete list of the output data protocols.

```

> KILL002 # step at which runaway activity is killed
    step_killed,

> POPD001 # spike history of neuron population
    pop_ind,
    spike_neuron_ind (row vector),
    num_spikes_t (1-by-step_tot row vector),
    num_ref_t (1-by-step_tot row vector),

> POPD002 # neuron parameters in the population
    pop_ind, number_of_parameters,
    parameter_name1, value1,
    parameter_name2, value2,
    ...

> POPD003 # neuron population statistical data
```

```
    pop_ind,
    V_mean_t1, V_mean_t2, ... (1-by-step_tot)
    V_std_t1, V_std_t2, ... (1-by-step_tot)
    I_input_mean_t1, I_input_mean_t2, ... (1-by-step_tot)
    I_input_std_t1, I_input_std_t2, ... (1-by-step_tot)

> POPD004 # sampled neuron data (deprecated)
    pop_ind, number_of_sample_neurons,
    data_name1, ..., data_nameX,
    data_1 (sampled_neurons-by-sampled_steps matrix),
    data_2 (sampled_neurons-by-sampled_steps matrix),
    ...
    data_X (sampled_neurons-by-sampled_steps matrix),


> POPD005 # E-I ratio for each neuron
    pop_ind,
    EI_ratio_1, EI_ratio_2, ... (1-by-N),

> SAMF001 # sampled data file name
    path/to/file_name (no comma!)

> POPD006 # sampled neuron data
    pop_ind, number_of_neurons times number_of_steps
    data_name1, ..., data_nameX,
    step_1 (sampled_neurons-by-data_types matrix),
    step_2 (sampled_neurons-by-data_types matrix),
    ...
    step_X (sampled_neurons-by-data_types matrix),

> POPD007 # local field potential data
    pop_ind,  No_of_measures,
    LFP_1, LFP_2, ..., LFP_step_tot, (measure 1)
    ...

> SYND001 # synaptic connection parameters
    number_of_parameters,
    parameter_name1, value1,
    parameter_name2, value2,
    ...

> SYND002 # sampled synapse  data
    pop_ind_pre, pop_ind_post, syn_type, no_of_sample_neurons,
    data (sampled_neurons-by-sampled_steps matrix),

> SYND003 # synapse statistical data
    pop_ind_pre, pop_ind_post, syn_type,
    I_mean_t1, I_mean_t2, ... (1-by-step_tot)
    I_std_t1, I_std_t2, ... (1-by-step_tot)
```

8 Publications

References

[1] Alain Destexhe, Zachary F Mainen, and Terrence J Sejnowski. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation*, 6(1):14–18, 1994.

[2] Alain Destexhe, Zachary F Mainen, and Terrence J Sejnowski. Kinetic models of synaptic transmission. In *Methods in neuronal modeling 2*, pages 1–25. 1998.

[3] Yifan Gu and Pulin Gong. The dynamics of memory retrieval in hierarchical networks. *Journal of Computational Neuroscience*, 40(3):247–268, 2016.

[4] Matthias H Hennig. Theoretical models of synaptic short term plasticity. *Neural Information Processing with Dynamical Synapses*, page 5, 2015.

[5] Ashok Litwin-Kumar and Brent Doiron. Slow dynamics and high variability in balanced cortical networks with clustered connections. *Nature neuroscience*, 15(11):1498–1505, 2012.

[6] Alessandro Treves. Mean-field analysis of neuronal spike dynamics. *Network: Computation in Neural Systems*, 4(3):259–284, 1993.

[7] TP Vogels, Henning Sprekeler, Friedemann Zenke, Claudia Clopath, and Wulfram Gerstner. Inhibitory plasticity balances excitation and inhibition in sensory pathways and memory networks. *Science*, 334(6062):1569–1573, 2011.