

SpikeNet Documentation

Yifan Gu

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

June 29, 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 preferred.

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:

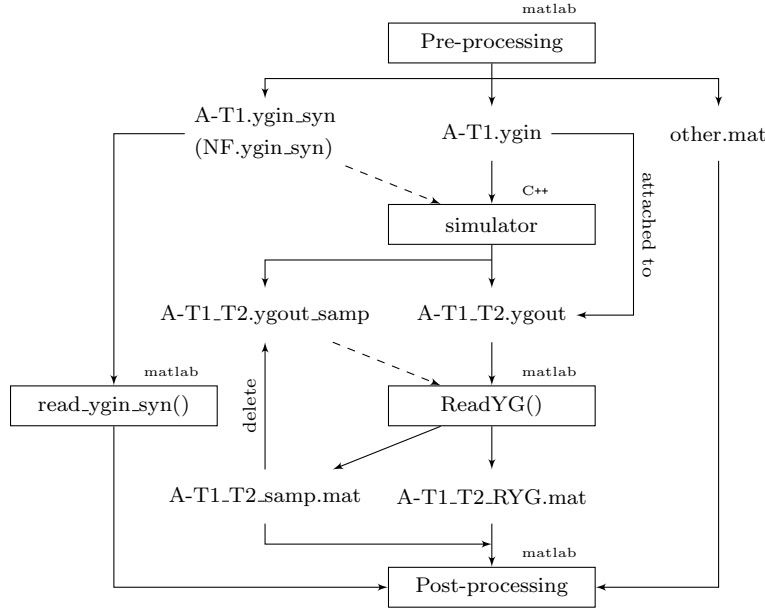
```

tmp
├─ SpikeNet
├─ simulator
├─ tmp.data
├─ my_functions
├─ PBS.out
└─ all_in_one.sh

```

4 Workflow

The typical workflow of SpikeNet is as shown in the following flowchart.



Although the C++ simulator accepts input files with any names, A-T1 is the recommended and default naming format. A is a 4-digit PBS array ID number. T1 is a timestamp identifying when the ygin file was generated. Similarly, T2 is a timestamp identifying when the ygout file was generated. Note that T2 allows multiple simulations to be run for the same ygin file. The synaptic connectivity definition file ygin_syn is separated from ygin for better memory management, since the connectivity definitions (protocol INIT006) are the single most memory-intensive component. Due to the same reason, the sampled time series data from simulation (protocol

SAMP001) are stored separately in `ygout_samp` and `samp.mat`. Non-default (NF) `ygin_syn` file path should be given in `ygin` using the provided Matlab interface. The dashed lines mean that the C++ simulator and the `ReadYG()` matlab function will automatically look for those auxiliary input files based on the information contained in the main input files.

5 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 ($i = 1, \dots, N_\alpha$) from population α evolves according to

$$C_m \frac{dV_i^\alpha}{dt} = -g_L(V_i^\alpha - V_L) + I_{i,syn}^\alpha(t) + I_{i,app}^\alpha(t), \text{ if } V_i^\alpha < V_{th}. \quad (5.1)$$

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

$$I_{i,syn}^\alpha(t) = \sum_{\beta=1}^P I_i^{\alpha\beta}(t) \quad (5.2)$$

$$= \sum_{\beta=1}^P [-g_i^{\alpha\beta}(V_i^\alpha - V_{rev}^\beta)] \quad (5.3)$$

$$= \sum_{\beta=1}^P \left\{ - \left[\sum_{j=1}^{N_\beta} a_{ij}^{\alpha\beta} g_{ij}^{\alpha\beta} s_{ij}^{\alpha\beta}(t) \right] (V_i^\alpha - V_{rev}^\beta) \right\} \quad (5.4)$$

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 instantaneous value of synaptic conductance in terms of the fraction of open channels, described by

$$\frac{ds_{ij}^{\alpha\beta}}{dt} = -\frac{s_{ij}^{\alpha\beta}}{\tau_d^\beta} + \sum_{t_j^\beta} h^\beta(t - t_j^\beta - d_{ij}^{\alpha\beta})(1 - s_{ij}^{\alpha\beta}) \quad (5.5)$$

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

$$h(t) = \begin{cases} 1/\tau_r, & \text{if } 0 \leq t \leq \tau_r \\ 0, & \text{otherwise} \end{cases} \quad (5.6)$$

All numerical values are in consistent 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].

6 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.

```

1 function main_demo()
2 % Use coherent units (msec+mV+nF+muS+nA) unless otherwise stated
3
4
5 %%% Seed the Matlab random number generator
6 seed = 1;
7
8 %%% Open new (uniquely time-stamped) input files for the SpikeNet C++
9 % simulator.
10 % FID is the main input file, FID.syn is the input file with the
11 % synaptic connectivity definitions (could be very large).
12 [FID, FID.syn] = new_ygin_files_and_randseed(seed);
13 % If no FID.syn is needed, use FID = new_ygin_files_and_randseed(seed,0)
14
15 %%% Define some basic parameters
16 % Time step (ms)
17 dt = 0.1; % 0.1 is usually small enough
18 % Total number of simulation steps
19 step_tot = 10^2;
20 % Create two neuron populations with 50 neurons in the 1st population
21 % and 10 neurons in the 2nd population.
22 N = [40, 10];
23 % Write the above basic parameters to the input file
24 writeBasicPara(FID, dt, step_tot, N);
25
26
27 %%% Define non-parameters for the neuron model
28 % For demo purpose, the values used as following are the still the
29 % default ones.
30 % If default values are to be used, there is no need to re-define them.
31 Cm = 0.25; % (nF) membrane capacitance
32 tau_ref = 2.0; % (ms) absolute refractory time
33 V_rt = -60.0; % (mV) reset potential
34 V_lk = -70.0; % (mV) leaky reversal potential
35 V_th = -50.0; % (mV) firing threshold
36 g_lk = 0.0167; % (muS) leaky conductance (note that Cm/gL=15 ms)
37 for pop_ind = 1:2
38     writePopPara(FID, pop_ind, 'Cm', Cm, 'tau_ref', tau_ref, 'V_rt', V_rt, ...
39                 'V_lk', V_lk, 'V_th', V_th, 'g_lk', g_lk);
40 end

```

```

41
42 %%% Add spike-frequency adaptation to the 1st population
43 pop = 1;
44 writeSpikeFreqAdpt(FID, pop);
45
46
47 %%% Define the initial condition
48 p_fire = [0.1 0.1]; % initial firing probabilities for both populations
49 % set initial V distribution to be [V_rt, V_rt + (V_th-V_rt)*r_V0]
50 r_V0 = [0.5 0.5];
51 writeInitCond(FID, r_V0, p_fire)
52
53 %%% Add external Poissonian spike trains into the 1st population
54 pop = 1;
55 type_ext = 1; % 1 for AMPA-like synapses
56 g_ext = 2*10^-3; % synaptic coupling strength (mS)
57 N_ext = 1000; % No. of independent external connections onto each neuron
58 rate_ext = 2*ones(1, step_tot); % Poisson rate for each time step (Hz)
59 N_start = 1;
60 N_end = 10; % only add to neuron No.1-10.
61 writeExtSpikeSettings(FID, pop, type_ext, g_ext, N_ext, rate_ext,...
62     N_start, N_end );
63
64
65 %%% Add external currents (Gaussian white noise) to the 2nd population
66 pop = 2;
67 I_ext_mean = 0.5*ones(1,N(2)); % defined for each neuron (nA)
68 I_ext_std = 0.2*ones(1,N(2)); % defined for each neuron (nA)
69 writeExtCurrentSettings(FID, pop, I_ext_mean, I_ext_std)
70
71 %%% Add external conductances (Gaussian white noise) to the 2nd population
72 % The default reversal potential is V_ext = 0.0 mV.
73 pop = 2;
74 g_ext_mean = 50*10^-3*ones(1,N(2)); % defined for each neuron (mS)
75 g_ext_std = 0.0*ones(1,N(2)); % defined for each neuron (mS)
76 writeExtConductanceSettings(FID, pop, g_ext_mean, g_ext_std)
77
78
79 %%% Define runaway killer
80 % The computational cost of the simulation is directly proportional to
81 % the average firing rate of the network. Very often, your parameters
82 % may lead to biologically unrealistically high firing rate or even
83 % runaway activity (the highest rate possible ~1/tau_ref). To save the
84 % computational resources for more interesting simulation cases, it is
85 % advised to define the runaway killer.
86 % Note that the simulator will still output all the relevant data before
87 % the kill.
88 min_ms = 500; % the min simulation duration that should be guaranteed
89 runaway_Hz = 40; % the threshold above which the simu should be killed
90 Hz_ms = 200; % the window length over which the firing rate is averaged
91 pop = 1; % the population to be monitored by the runaway killer
92 writeRunawayKiller(FID, pop, min_ms, runaway_Hz, Hz_ms);
93
94
95 %%% Record the basic statistics for the 1st neuron population
96 pop = 1;
97 writePopStatsRecord(FID, pop);
98

```

```

99 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Chemical Connections %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
100 % type(1:AMAP, 2:GABAA, 3:NMDA)
101 % Define AMAP-like excitatory synaptic connection within the 1st pop
102 pop_pre = 1;
103 pop_post = 1;
104 syn_type = 1; % 1 for AMPA-like synapse
105 g_EE = 10*10^-3; % synaptic coupling strength (mS)
106 % random adjacency matrix with a wiring probability of 0.2
107 A = rand(N(1), N(1)) < 0.2;
108 [I_EE, J_EE] = find(A);
109 K_EE = g_EE*ones(size(I_EE)); % identical coupling strength
110 D_EE = rand(size(I_EE))*5; % uniformly random conduction delay [0, 5] ms
111 writeChemicalConnection(FID_syn, syn_type, pop_pre, pop_post, ...
112     I_EE, J_EE, K_EE, D_EE);
113
114 % Define AMAP-like excitatory synaptic connection from pop 1 to pop 2
115 pop_pre = 1;
116 pop_post = 2;
117 syn_type = 1; % 1 for AMPA-like synapse
118 g_IE = 10*10^-3; % synaptic coupling strength (mS)
119 % random adjacency matrix with a wiring probability of 0.5
120 A = rand(N(1), N(2)) < 0.5;
121 [I_IE, J_IE] = find(A);
122 K_IE = g_IE*ones(size(I_IE)); % identical coupling strength
123 D_IE = rand(size(I_IE))*5; % uniformly random conduction delay [0, 5] ms
124 writeChemicalConnection(FID_syn, syn_type, pop_pre, pop_post, ...
125     I_IE, J_IE, K_IE, D_IE);
126
127 % Define GABA-like excitatory synaptic connection from pop 2 to pop 1
128 pop_pre = 2;
129 pop_post = 1;
130 syn_type = 2; % 2 for GABA-like synapse
131 g_EI = 10*10^-3; % synaptic coupling strength (mS)
132 % random adjacency matrix with a wiring probability of 0.5
133 A = rand(N(2), N(1)) < 0.5;
134 [I_EI, J_EI] = find(A);
135 K_EI = g_EI*ones(size(I_EI)); % identical coupling strength
136 D_EI = rand(size(I_EI))*5; % uniformly random conduction delay [0, 5] ms
137 writeChemicalConnection(FID_syn, syn_type, pop_pre, pop_post, ...
138     I_EI, J_EI, K_EI, D_EI);
139
140 % Define GABA-like excitatory synaptic connection within pop 2
141 pop_pre = 2;
142 pop_post = 2;
143 syn_type = 2; % 2 for GABA-like synapse
144 g_II = 20*10^-3; % synaptic coupling strength (mS)
145 % random adjacency matrix with a wiring probability of 0.5
146 A = rand(N(2), N(2)) < 0.5;
147 [I_II, J_II] = find(A);
148 K_II = g_II*ones(size(I_II)); % identical coupling strength
149 D_II = rand(size(I_II))*5; % uniformly random conduction delay [0, 5] ms
150 writeChemicalConnection(FID_syn, syn_type, pop_pre, pop_post, ...
151     I_II, J_II, K_II, D_II);
152 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Chemical Connections Done %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
153
154
155 %%% Use an existing synapse connectivity definition file instead of
156 % generating a new one:

```



```

157 % writeSynFilename(FID, 'path/to/existing/XXX.ygin-syn')
158 % In this case, you should use
159 % "FID = new_ygin_files_and_randseed(seed, 0)"
160 % to avoid creating an empty ygin-syn file.
161
162
163
164 %%% Define non-default synapse parameters
165 writeSynPara(FID, 'tau_decay_GABA', 3);
166 % "help writeSynPara" to see all the parameter names and default values
167
168 %%% Add short-term depression to the synapses with the 1st population
169 pop_pre = 1;
170 pop_post = 1;
171 step_start = round(step_tot/3); % Turn on STD at this time step
172 writeSTD(FID, pop_pre, pop_post, step_start);
173
174
175 %%% Add inhibitory STDP to the synapses from pop 2 to pop 1
176 pop_pre = 2;
177 pop_post = 1;
178 % Turn on inhibitory STDP at a certain time step
179 step_start = round(step_tot/3*2);
180 writeInhSTDP(FID, pop_pre, pop_post, step_start);
181
182 %%% Record the basic statistics for the excitatory synapses within
183 % the 1st neuron population
184 pop_pre = 1;
185 pop_post = 1;
186 syn_type = 1; % 1 for AMPA-like synapse
187 writeSynStatsRecord(FID, pop_pre, pop_post, syn_type);
188
189 %%% Sample detailed time series data from the 1st population
190 pop = 1;
191 sample_neuron = 1:10:N(1); % sample every 10th neuron
192 sample_steps = 1:2:step_tot; % sample every 2nd time step
193 sample_data_type = logical([1,1,1,1,0,0,1,0]);
194 % The logical vector above corresponds to
195 % [V, I_leak, I_AMPA, I_GABA, I_NMDA, I_GJ, I_ext, I_K]
196 writeNeuronSampling(FID, pop, sample_data_type, ...
197     sample_neuron, sample_steps)
198
199 %%% Optional: record explanatory variables (scalars only)
200 % They will also be used in post-processing for auto-generated comments
201 discard_transient = 0; % transient period data to be discarded (ms)
202 writeExplVar(FID, 'discard_transient', discard_transient, ...
203     'g_EE', g_EE, ...
204     'g_EI', g_EI, ...
205     'g_IE', g_IE, ...
206     'g_II', g_II, ...
207     'g_ext', g_ext, ...
208     'N_ext', N_ext, ...
209     'rate_ext', rate_ext);
210
211
212 %%% Additional comments to be passed to the post-processing stage
213 comment1 = 'This is a demo.';
214 comment2 = ['If you have any question regarding SpikeNet, '...

```

```

215     'please contact Yifan Gu (yigu8115@gmail.com).'];
216 writeExplVar(FID, 'comment1', comment1, 'comment2', comment2);
217
218 %%% append this matlab file to the input file for future reference
219 appendThisMatlabFile(FID)
220
221 end
222
223
224
225 function appendThisMatlabFile(FID)
226 breaker = ['>', repmat('#',1,80)];
227 fprintf(FID, '%s\n', breaker);
228 fprintf(FID, '%s\n', '> MATLAB script generating this file: ');
229 fprintf(FID, '%s\n', breaker);
230 Fself = fopen([mfilename('fullpath'), '.m'], 'r');
231 while ~feof(Fself)
232     tline = fgetl(Fself);
233     fprintf(FID, '%s\n', tline);
234 end
235 fprintf(FID, '%s\n', breaker);
236 fprintf(FID, '%s\n', breaker);
237 fprintf(FID, '%s\n', breaker);
238 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.

7 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 ( $\mu$ S), 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 > INIT012 # external Gaussian conductance
44     pop_ind,
45     mean_1 ( $\mu$ S), mean_2, ..., mean_N, (for each neuron)
46     std_1 ( $\mu$ S), std_2, ..., std_N,
47
48 > SYNFO01 # non-default synapse definition file name
49     path/to/file_name (no comma!)
50
51 > KILL001 # runaway killer setting
52     pop_ind, min_ms, runaway_Hz, Hz_ms,
53
54 > PARA001 # non-default neuron population parameter
55     pop_ind, number_of_parameters,
56     parameter_name1, value1,
57     parameter_name2, value2,
58     ...
59
60 > PARA002 # non-default synapse parameter
61     number_of_parameters,
62     parameter_name1, value1,

```

```

63     parameter_name2, value2,
64     ...
65
66 > SAMP001 # neuron data sampling
67     pop_ind,
68     data_type (logical vector),
69     ind1, ind2, ..., indX, (sample neuron indices)
70     1, 1, 0, 0, ..., (1-by-step_tot logical values)
71     # Note that data_type specifies sample data types
72     # and it must correspond to
73     # [V,I_leak,I_AMPA,I_GABA,I_NMDA,I_GJ,I_ext]
74
75 > SAMP002 # synapse data sampling
76     pop_ind_pre, pop_ind_post, syn_type,
77     ind1, ..., indX, (post-synaptic sample neuron indices)
78     1, 1, 0, 0, ..., (1-by-step_tot logical values)
79
80 > SAMP003 # neuron population statistical data record
81     pop_ind,
82
83 > SAMP004 # synaptic currents statistical data record
84     pop_ind_pre, pop_ind_post, syn_type,
85
86 > SAMP005 # local field potential data record
87     pop_ind, No_of_measures
88     1, 0, 0, 1, ...,
89     ...
90     0, 0, 1, 1, ..., (No_of_measures-by-N logical values)

```

8 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.

```

1
2 > KILL002 # step at which runaway activity is killed
3     step_killed,
4
5 > POPD001 # spike history of neuron population
6     pop_ind,
7     spike_neuron_ind (row vector),
8     num_spikes_t (1-by-step_tot row vector),
9     num_ref_t (1-by-step_tot row vector),
10

```

```

11 > POPD002 # neuron parameters in the population
12     pop_ind, number_of_parameters,
13     parameter_name1, value1,
14     parameter_name2, value2,
15     ...
16
17 > POPD003 # neuron population statistical data
18     pop_ind,
19     V_mean_t1, V_mean_t2, ... (1-by-step_tot)
20     V_std_t1, V_std_t2, ... (1-by-step_tot)
21     I_input_mean_t1, I_input_mean_t2, ... (1-by-step_tot)
22     I_input_std_t1, I_input_std_t2, ... (1-by-step_tot)
23
24 > POPD004 # sampled neuron data (deprecated)
25     pop_ind, number_of_sample_neurons,
26     data_name1, ..., data_nameX,
27     data_1 (sampled_neurons-by-sampled_steps matrix),
28     data_2 (sampled_neurons-by-sampled_steps matrix),
29     ...
30     data_X (sampled_neurons-by-sampled_steps matrix),
31
32
33 > POPD005 # E-I ratio for each neuron
34     pop_ind,
35     EI_ratio_1, EI_ratio_2, ... (1-by-N),
36
37 > SAMF001 # sampled data file name
38     path/to/file_name (no comma!)
39
40 > POPD006 # sampled neuron data
41     pop_ind, number_of_neurons times number_of_steps
42     data_name1, ..., data_nameX,
43     step_1 (sampled_neurons-by-data_types matrix),
44     step_2 (sampled_neurons-by-data_types matrix),
45     ...
46     step_X (sampled_neurons-by-data_types matrix),
47
48 > POPD007 # local field potential data
49     pop_ind, No_of_measures,
50     LFP_1, LFP_2, ..., LFP_step_tot, (measure 1)
51     ...
52
53 > SYND001 # synaptic connection parameters
54     number_of_parameters,
55     parameter_name1, value1,
56     parameter_name2, value2,
57     ...
58
59 > SYND002 # sampled synapse data
60     pop_ind_pre, pop_ind_post, syn_type, no_of_sample_neurons,

```

```

61     data (sampled_neurons-by-sampled_steps matrix),
62
63 > SYND003 # synapse statistical data
64     pop_ind_pre, pop_ind_post, syn_type,
65     I_mean_t1, I_mean_t2, ... (1-by-step_tot)
66     I_std_t1, I_std_t2, ... (1-by-step_tot)

```

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.