SpikeNet Documentation

Yifan Gu

Complex System Group, School of Physics, USYD

yigu8115@gmail.com

June 15, 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, 3, 5, 6]). 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_PRE_PROCESS_FUNC='your_function'
    ○ MATLAB_POST_PROCESS_FUNC='your_function'
    ○ MATLAB_SOURCE_PATH='SpikeNet'

- 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 your own files.

3

4 Matlab user interface

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

```matlab
1   function main_demo()
2   % Use coherent units (msec+mV+nF+miuS+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 neuorn 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;   % (miuS) 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)
```

```matlab
52
53  %%%% Add external Poissonian spike trains into the 1st population
54  pop = 1;
55  type_ext = 1; % 1 for AMPA-like syanpses
56  g_ext = 2*10^-3; % synaptic coupling strength (miuS)
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  %%%% Define runaway killer
72  % The computational cost of the simulation is directly proportional to
73  % the avearage firing rate of the network. Very often, your parameters
74  % may lead to biologically unrealistically high firing rate or even
75  % runaway activity (the highest rate possible ~1/tau_ref). To save the
76  % computational resources for more interesting simulation cases, it is
77  % advised to define the runawary killer.
78  % Note that the simulator will still output all the relevant data before
79  % the kill.
80  min_ms = 500; % the min simulation duration that should be guaranteed
81  runaway_Hz = 40; % the threshold above which the simu should be killed
82  Hz_ms = 200; % the window length over which the firing rate is averaged
83  pop = 1; % the population to be monitored by the runaway killer
84  writeRunawayKiller(FID, pop, min_ms, runaway_Hz, Hz_ms);
85
86
87  %%%% Record the basic statistics for the 1st neuron population
88  pop = 1;
89  writePopStatsRecord(FID, pop);
90
91  %%%%%%%%%%%%%%%%%% Chemical Connections %%%%%%%%%%%%%%%%%%%%%%%%
92  % type(1:AMAP, 2:GABAa, 3:NMDA)
93  % Define AMAP-like excitatory synaptic connection wtihin the 1st pop
94  pop_pre = 1;
95  pop_post = 1;
96  syn_type = 1; % 1 for AMPA-like synapse
97  g_EE = 10*10^-3; % synaptic coupling strength (miuS)
98  % random adjacency matrix with a wiring probability of 0.2
99  A = rand(N(1), N(1)) < 0.2;
100 [I_EE, J_EE] = find(A);
101 K_EE = g_EE*ones(size(I_EE)); % identical coupling strength
102 D_EE = rand(size(I_EE))*5; % uniformly random conduction delay [0, 5] ms
103 writeChemicalConnection(FID_syn, syn_type,  pop_pre, pop_post, ...
104     I_EE, J_EE, K_EE, D_EE);
105
106 % Define AMAP-like excitatory synaptic connection from pop 1 to pop 2
107 pop_pre = 1;
108 pop_post = 2;
109 syn_type = 1; % 1 for AMPA-like synapse
```

```matlab
110  g_IE = 10*10^-3; % synaptic coupling strength (miuS)
111  % random adjaceny matrix with a wiring probability of 0.5
112  A = rand(N(1), N(2)) < 0.5;
113  [I_IE, J_IE] = find(A);
114  K_IE = g_IE*ones(size(I_IE)); % identical coupling strength
115  D_IE = rand(size(I_IE))*5; % uniformly random conduction delay [0, 5] ms
116  writeChemicalConnection(FID_syn, syn_type, pop_pre, pop_post, ...
117      I_IE, J_IE, K_IE, D_IE);
118
119  % Define GABA-like excitatory synaptic connection from pop 2 to pop 1
120  pop_pre = 2;
121  pop_post = 1;
122  syn_type = 2; % 2 for GABA-like synapse
123  g_EI = 10*10^-3; % synaptic coupling strength (miuS)
124  % random adjaceny matrix with a wiring probability of 0.5
125  A = rand(N(2), N(1)) < 0.5;
126  [I_EI, J_EI] = find(A);
127  K_EI = g_EI*ones(size(I_EI)); % identical coupling strength
128  D_EI = rand(size(I_EI))*5; % uniformly random conduction delay [0, 5] ms
129  writeChemicalConnection(FID_syn, syn_type, pop_pre, pop_post, ...
130      I_EI, J_EI, K_EI, D_EI);
131
132  % Define GABA-like excitatory synaptic connection within pop 2
133  pop_pre = 2;
134  pop_post = 2;
135  syn_type = 2; % 2 for GABA-like synapse
136  g_II = 20*10^-3; % synaptic coupling strength (miuS)
137  % random adjaceny matrix with a wiring probability of 0.5
138  A = rand(N(2), N(2)) < 0.5;
139  [I_II, J_II] = find(A);
140  K_II = g_II*ones(size(I_II)); % identical coupling strength
141  D_II = rand(size(I_II))*5; % uniformly random conduction delay [0, 5] ms
142  writeChemicalConnection(FID_syn, syn_type, pop_pre, pop_post, ...
143      I_II, J_II, K_II, D_II);
144  %%%%%%%%%%%%%%%%%% Chemical Connections Done %%%%%%%%%%%%%%%%%%%%%%%%
145
146
147  %%%% Use an existing synapse connectivity definition file instead of
148  % generating a new one:
149  % writeSynFilename(FID, 'path/to/existing/XXX.ygin_syn')
150  % In this case, you should use
151  % "FID = new_ygin_files_and_randseed(seed, 0)"
152  % to avoid creating an empty ygin_syn file.
153
154
155
156  %%%% Define non-default synapse parameters
157  writeSynPara(FID, 'tau_decay_GABA', 3);
158  % "help writeSynPara" to see all the parameter names and default values
159
160  %%%% Add short-term depression to the synapses with the 1st population
161  pop_pre = 1;
162  pop_post = 1;
163  step_start = round(step_tot/3); % Turn on STD at this time step
164  writeSTD(FID, pop_pre, pop_post, step_start);
165
166
167  %%%% Add inhibitory STDP to the synapses from pop 2 to pop 1
```

```matlab
168  pop_pre = 2;
169  pop_post = 1;
170  % Turn on inhibitory STDP at a certain time step
171  step_start = round(step_tot/3*2);
172  writeInhSTDP(FID, pop_pre, pop_post, step_start);
173
174  %%%% Record the basic statistics for the excitatory synapses within
175  % the 1st neuron population
176  pop_pre = 1;
177  pop_post = 1;
178  syn_type = 1; % 1 for AMPA-like synapse
179  writeSynStatsRecord(FID, pop_pre, pop_post, syn_type);
180
181  %%%% Sample detailed time serious data from the 1st population
182  pop = 1;
183  sample_neuron = 1:10:N(1); % sample every 10th neuron
184  sample_steps = 1:2:step_tot; % sample every 2nd time step
185  sample_data_type = logical([1,1,1,1,0,0,1,0]);
186  % The logical vector above corresponds to
187  % [V,I_leak,I_AMPA,I_GABA,I_NMDA,I_GJ,I_ext, I_K]
188  writeNeuronSampling(FID, pop, sample_data_type, ...
189       sample_neuron, sample_steps)
190
191  %%%% Record explanatory variables that are nacessary for post-processing
192  discard_transient = 0; % transient period data to be discarded (ms)
193  writeExplVar(FID, 'discard_transient', discard_transient, ...
194       'g_EE', g_EE, ...
195       'g_EI', g_EI, ...
196       'g_IE', g_IE, ...
197       'g_II', g_II, ...
198       'g_ext', g_ext, ...
199       'N_ext', N_ext, ...
200       'rate_ext', rate_ext);
201
202
203  %%%% Additional comments to be passed to the post-processing stage
204  comment1 = 'This is a demo.';
205  comment2 = ['If you have any question regarding SpikeNet,'...
206       'please contact Yifan Gu (yigu8115@gmail.com).'];
207  writeExplVar(FID, 'comment1', comment1, 'comment2', comment2);
208
209  %%%% append this matlab file to the input file for future reference
210  appendThisMatlabFile(FID)
211
212  end
213
214
215
216  function appendThisMatlabFile(FID)
217  breaker = ['>',repmat('#',1,80)];
218  fprintf(FID, '%s\n', breaker);
219  fprintf(FID, '%s\n', '> MATLAB script generating this file: ');
220  fprintf(FID, '%s\n', breaker);
221  Fself = fopen([mfilename('fullpath'),'.m'],'r');
222  while ~feof(Fself)
223       tline = fgetl(Fself);
224       fprintf(FID, '%s\n', tline);
225  end
```

```
226  fprintf(FID, '%s\n', breaker);
227  fprintf(FID, '%s\n', breaker);
228  fprintf(FID, '%s\n', breaker);
229  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.)

5 Neuron and synapse models

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

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

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

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

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

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

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

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

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

8

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

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 [4].

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)
65      1, 1, 0, 0, ..., (1-by-step_tot logical values)
66      # Note that data_type specifies sample data types
67      # and it must correspond to
68      # [V,I_leak,I_AMPA,I_GABA,I_NMDA,I_GJ,I_ext]
69
70  > SAMP002 # synapse data sampling
71      pop_ind_pre, pop_ind_post, syn_type,
72      ind1, ..., indX, (post-synaptic sample neuron indices)
73      1, 1, 0, 0, ..., (1-by-step_tot logical values)
74
75  > SAMP003 # neuron population statistical data record
76      pop_ind,
77
78  > SAMP004 # synaptic currents  statistical data record
79      pop_ind_pre, pop_ind_post, syn_type,
```

```
80
81  > SAMP005 # local field potential data record
82      pop_ind,
83      1, 0, 0, 1, ..., (1-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.

```
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,
50      LFP_1, LFP_2, ..., LFP_step_tot,
51
52  > SYND001 # synaptic connection parameters
53      number_of_parameters,
54      parameter_name1, value1,
55      parameter_name2, value2,
56      ...
57
58  > SYND002 # sampled synapse  data
59      pop_ind_pre, pop_ind_post, syn_type, no_of_sample_neurons,
60      data (sampled_neurons-by-sampled_steps matrix),
61
62  > SYND003 # synapse statistical data
63      pop_ind_pre, pop_ind_post, syn_type,
64      I_mean_t1, I_mean_t2, ... (1-by-step_tot)
65      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] Matthias H Hennig. Theoretical models of synaptic short term plasticity. *Neural Information Processing with Dynamical Synapses*, page 5, 2015.

[4] 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.

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

[6] 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.