SpikeNet Documentation

Yifan Gu*[1], James Henderson[1], Yuxi Liu[1], and Guozhang Chen[1]

[1]School of Physics and ARC Centre of Excellence for Integrative Brain Function, University of Sydney, NSW 2006, Australia

April 3, 2017

1 Overview

SpikeNet is a software developed for simulating spiking neuronal networks, of which the design provides the following four main 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, 8, 9]). 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.

*yigu8115@gmail.com

1. In Matlab, use SpikeNet functions to generate the input files 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 into Matlab data for post-processing.

Please see Matlab user interface for more detailed descriptions of the workflow. 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. Developer users will be interested to know that SpikeNet supports HDF5-based input/output file format which follows pre-defined protocols (see Input/Output protocols).

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. Additionally, HDF5 is supported for handling large input/output data files.

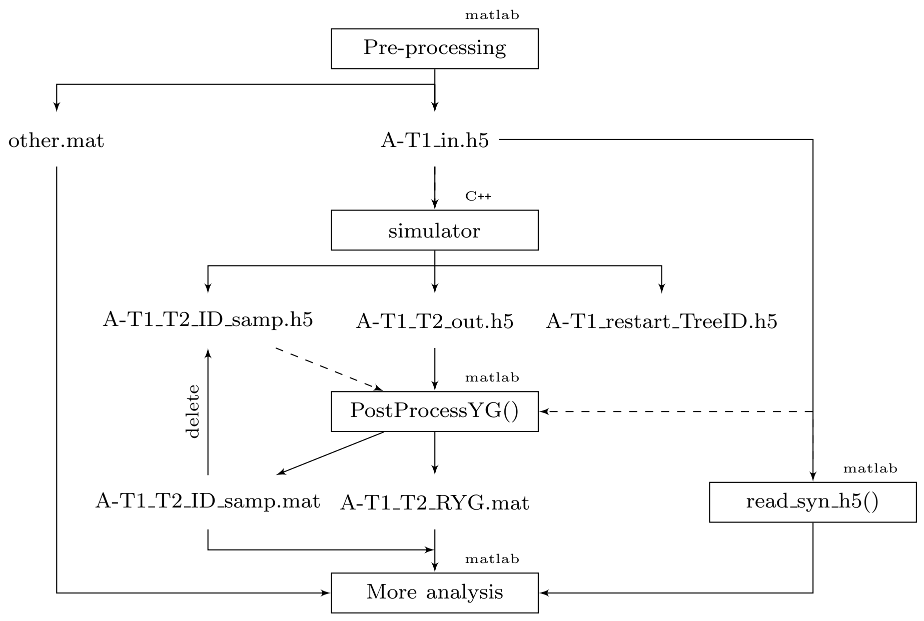
Here are some basic tips on git.

- To get the most recent version of SpikeNet: `git pull origin master`
- Build the new simulator: `make; make clean`
- 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
|  |__cpp_sourses
|  |__matlab_interfaces
|  |__makefile
|  |__other_scripts
|__simulator
|__tmp_data
|__my_functions
|__PBS_out
|__all_in_one.sh
```

2 Workflow

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



Notes:

- 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 in.h5 file was generated.
- Similarly, T2 is a timestamp identifying when the out.h5 file was generated. Note that T2 allows multiple simulations to be run for the same in.h5 file.
- The restart_TreeID.h5 files allow the users to directly modify any aspect of a simulation and restart it from there. The TreeID is automatically generated to ensure that the users can make as many different modifications and restart the simulation as many times as desired.
- For technical reasons, the time series data sampled from each neuron population or synapse group, identified by an ID number, during simulation are stored separately in samp.h5 files.
- The dashed lines mean that the C++ simulator and the PostProcessYG() matlab function will automatically look for those auxiliary input files based on the information contained in the main input files.

3 Documentation for the C++ simulator

To view the documentation

- Make sure doxygen and graphviz are installed
- Go to the SpikeNet directory
- Generate the documentation: `doxygen`
- Open the documentation: `open docu*/html/index.html`

4 Neuron and synapse models

4.1 Default model

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 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 [6].

4.2 Currently available models

1. Exponential leaky integrate-and-fire neuron model
2. Double-exponential synapse model: see [5]
3. Inhibitory plasticity: see [9]
4. Spike frequency adaptation: see [7]

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.

```matlab
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] = new_ygin_files_and_randseedHDF5(seed);

%%%% 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
writeBasicParaHDF5(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
    writePopParaHDF5(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;
writeSpikeFreqAdptHDF5(FID, pop);


% % Use Adam 2016 synapse model
% model_choice = 2;
% writeSynapseModelChoiceHDF5(FID, model_choice)


%%%% 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];
writeInitCondHDF5(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)
neurons_recv = zeros(1, N(pop));
neurons_recv(1:10) = 1; % only neurons #1-10 receive such external input
writeExtSpikeSettingsHDF5(FID, pop, type_ext, g_ext,  N_ext, rate_ext,...
    neurons_recv );


%%%% 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)
writeExtCurrentSettingsHDF5(FID, pop, I_ext_mean, I_ext_std)

%%%% Add external conductances (Gaussian white noise) to the 2nd population
% The default reveral potential is V_ext = 0.0 mV.
pop = 2;
g_ext_mean = 50*10^-3*ones(1,N(2)); % defined for each neuron (muS)
g_ext_std = 0.0*ones(1,N(2)); % defined for each neuron (muS)
writeExtConductanceSettingsHDF5(FID, pop, g_ext_mean, g_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
writeRunawayKillerHDF5(FID, pop, min_ms, runaway_Hz, Hz_ms);


%%%% Record the basic statistics for the 1st neuron population
pop = 1;
writePopStatsRecordHDF5(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
writeChemicalConnectionHDF5(FID, 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
writeChemicalConnectionHDF5(FID, 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
writeChemicalConnectionHDF5(FID, 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
writeChemicalConnectionHDF5(FID, 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:
% (The feature needs to be written)


%%%% Define non-default synapse parameters
writeSynParaHDF5(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
writeSTDHDF5(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);
writeInhSTDPHDF5(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
writeSynStatsRecordHDF5(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 = [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]
writeNeuronSamplingHDF5(FID, pop, sample_data_type, ...
    sample_neuron, sample_steps)

%%%% Optional: record explanatory variables (scalars only)
```

```matlab
% They will also be used in pro-processsing for auto-generated comments
discard_transient = 0; % transient period data to be discarded (ms)
comment1 = 'This is a demo.';
comment2 = ['If you have any question regarding SpikeNet,'...
    'please contact Yifan Gu (yigu8115@gmail.com).'];
writeExplVarHDF5(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,...
    'comment1', comment1, 'comment2', comment2);

%%%% append this matlab file to the input file for future reference
appendThisMatlabFileHDF5(FID)

end


function appendThisMatlabFileHDF5(FID)

% need to coopy and past the following code into the file to be appended!
text = fileread([mfilename('fullpath'),'.m']);
hdf5write(FID,'/config/MATLAB/config.m',text,'WriteMode','append');

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/Output protocols

```
/
└── config
    ├── Net
    │   ├── INIT001
    │   │   └── N
    │   ├── INIT002
    │   │   └── dt, step_tot
    │   └── SEED001
    │       └── seed
    ├── pops
    │   ├── pop0
    │   │   ├── PARA001
    │   │   │   └── para_str_ascii
    │   │   ├── INIT004
    │   │   │   └── mean, std
    │   │   ├── INIT011
    │   │   │   └── r_V0, p_fire
    │   │   ├── SAMP103
    │   │   │   └── time_start, time_end
    │   │   └── ...
    │   ├── pop1
    │   │   └── ...
    │   └── ...
    └── syns
        ├── n_syns
        ├── syn0
        │   ├── INIT006
        │   │   └── type, i_pre, j_post, I, J, K, D
        │   ├── INIT005
        │   │   └── pop_ind, type_ext, K_ext, Num_ext, neurons,
        │   │       rate_ext_t
        │   ├── SEED002
        │   │   └── seed
        │   ├── SAMP104
        │   │   └── time_start, time_end
        │   └── ...
        ├── syn1
        │   └── ...
        └── ...
```

6.1 Notes on RNG control

To fully control the RNG in C++ simulator, you need to pass the seed via this two functions.

1. writeExtSpikeSettingsHDF5(..., Seed)
2. writePopSeedHDF5(PopID, Seed)

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] Adam Keane and Pulin Gong. Propagating waves can explain irregular neural dynamics. *The Journal of Neuroscience*, 35(4):1591–1605, 2015.

[6] 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.

[7] DV Madison and RA Nicoll. Control of the repetitive discharge of rat CA1 pyramidal neurones in vitro. *The Journal of Physiology*, 354:319, 1984.

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

[9] 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.