L'Equipe

1 Quick Start

- Install the libraries GSL, NETCDF, and if wanted ATLAS and MPI. Edit the Makefile with the correct links to your installations.
  (The current version with which lequipe is working are gsl-1.14, netcdf-3.6.3, atlas-3.8.3 and openmpi-1.4.2)
- Compile LEquipe with the make command on the machine where you are going to run the simulations.
- Write your matlab/python/... script that generates the netcdf files with all parameters and variables to run the simulation.
- Run the simulation.
- Read in the results with your matlab/python/... script.
- Consider to delete unnecessary data (.nc) files after the simulation is done and to put the data you want to keep into a tar archive. It is not necessary, or actually possible, to compress the netcdf files since all data is stored in the binary format.

2 Example: Spike Train

Let us calculate the spike train of a neural network using different neuron types. After compiling the main program LEquipe with the make command, open the matlab script LEquipe_train.m in the folder scripts. The script should look like this:

```
clear all
addpath('../code/')
set(0, 'defaultaxesfontsize', 20);

%%%%%%%%%%%%% define the parameters of the network here %%%%%%%%%%

neuronType = 1; %neuron type

N = 200;        %number of neurons
K = 50;         %number of synapses per neuron
J0 = -1;        %coupling strength
f = 5;          %network-averaged firing rate in Hz
tauM = 10;      %membrane time constant

rap = 1;        %AP onset rapidness in case of rapid theta neurons
tauS = tauM/2;  %synaptic time constant in case of cLIF or twoDlinear

%%%%%%%%%%%%%%%%%%%%%%%%%% end of input %%%%%%%%%%%%%%%%%%%%%%%%%%

%% set the given neuron parameters
ParaNet.N = N;
ParaNet.NeuronType = neuronType;
ParaNet.rapidness = rap;
ParaNet.tauM = tauM;

TwoDlinear.alpha = 1;
TwoDlinear.beta = 0;
TwoDlinear.gamma = 0;
TwoDlinear.delta = 1;
TwoDlinear.Cw = 0;
TwoDlinear.tauS = tauS;
ParaNet.twoDlinear = TwoDlinear;
```

```
%% set the random graph with K synapses per neuron on average
rand('twister', 1);
[ParaTopo.post ParaTopo.row_length] = random_graph(K, N);

%% set synapstic coupling strength (sqrt(K) scaling for the balanced state)
ParaTopo.J = J0/sqrt(K);

%% set the parameters of the simulation
ParaSim.rateWnt = f;        % this is the wanted firing rate
% the external currents that yield the wanted firing rate can be well
% approximated by the balance equation f = -I0/(J0*tauM)
% then with the balanced state scaling we end up with Iext = sqrt(K)*I0
ParaNet.Iext = -J0*f/1000*tauM*sqrt(K);

ParaSim.SW = 100;           % number of spikes per neuron during warmup
ParaSim.TC = 1;             % time duration of the calculation in seconds

ParaSim.train = 1:N;        % neurons, whose spike times are saved

%% write all parameters to netcdf files to directory data/ and get the hashes
directory = '../data/';
if ~exist(directory, 'dir')
    disp(['creating new directory: ' directory]);
    mkdir(directory)
end

[HashNet, FileNet] = writeNet(ParaNet, directory);
[HashTopo, FileTopo] = writeTopo(ParaTopo, directory);
[HashSim, FileSim] = writeSim(ParaSim, directory);
HashDataOut = DataHash([HashNet, HashTopo, HashSim]);
FileOut = [directory, 'DataOut-', HashDataOut, '.nc'];

%% run the C++ simulation
system(['../LEquipe ', FileNet, ' ', FileTopo, ' ', FileSim, ' ', FileOut]);

%% read the output file and plot the results
Data = readDataOut(FileOut);

figure;
plot(Data.trainTime, Data.trainNeuron, '.', 'markersize', 7);
xlabel('time (s)');
ylabel('neurons')
xlim([0 ParaSim.TC])
```

Press F5 and run the skript in matlab. This will create a directory called data and write the three netcdf files with the neuron description, the topology description and the simulation description. You can look at the binary content of these files with ncdump.

```
creating new directory: ../data/
writing neuron netcdf file: ../data/ParaNeurons-238560aa7913a48c7b277911375edf05.nc
writing topology netcdf file: ../data/ParaTopology-e0472435972de7d62357cbd3a12d597b.nc
writing simulation netcdf file: ../data/ParaSimulation-37020414514f2266560cff4617dee626.nc
```

Afterwards, the C++ LEquipe is called by the matlab script. It will tell us, that the simulation runs on one processor. Then it will output some not so important details and will read in the three netcdf files where all the parameters for the simulation are specified.

```
running on 1 processor(s) ...

********************** CPU 1 of 1 reporting for duty **********************
running on esperine.local in /Users/mik/MPI/code/cpp/lequipe/Tutorial/scripts
```

```
the 4 command line arguments were:
../data/ParaNeurons-238560aa7913a48c7b277911375edf05.nc
../data/ParaTopology-e0472435972de7d62357cbd3a12d597b.nc
../data/ParaSimulation-37020414514f2266560cff4617dee626.nc
../data/DataOut-5e87c799f7dcc1c9da6942aa0b791810.nc
```

```
the used data types are:
int:            size = 4 Bytes,         range = -2147483648 to 2147483647
unsigned:       size = 4 Bytes,         range = 0 to 4294967295
long long:      size = 8 Bytes,         range = -9223372036854775808 to 9223372036854775807
reell:          size = 8 Bytes,         range = 2.22507e-308 to 1.79769e+308,   precision =
```

```
reading ../data/ParaNeurons-238560aa7913a48c7b277911375edf05.nc ...
reading ../data/ParaTopology-e0472435972de7d62357cbd3a12d597b.nc ...
reading ../data/ParaSimulation-37020414514f2266560cff4617dee626.nc ...

*** Parameter import successful!

CPU time for initialization: 0s
```

Then it will generate the neural network according to the parameters and start the actual simulation. First the external currents that were provided are adapted to yield the wanted average firing rate of 5Hz. Then a warmup of on average 100 spikes per neuron is done. After some preprocessing where internal arrays are allocated and some checks done, the main simulation is done and the spike train recorded. Then there is some postprocessing of internal variables and finally the results including the spike train are saved to the netcdf file DataOut....nc.

```
*************************** seting up the network ***************************

CPU time for network setup: 0s

************************* starting the simulations *************************
adapt the external currents to yield the wanted average firing rate
rateWnt = 5 Hz with precision pR = 0.01 and SR = 2000 or TR = 0 ms ...
        1*Iext yielded f = 5.27702 Hz
        0.947504*Iext yielded f = 5.04323 Hz
the external currents are set to 0.947504*Iext

warmup with SW = 20000 or TW = 0 ms ...
spikes: 20000   time: 19.888s    -> avg. rate: 5.02817 Hz

preprocessing ...

on the reference trajectory, gonna calculate:
         * the spike train of 200 neurons

simulation with SC = 0 or TC = 1000 ms ...
        10% of TC done ... 113 spikes @ t = 0.100248s
        20% of TC done ... 203 spikes @ t = 0.20086s
        30% of TC done ... 306 spikes @ t = 0.300026s
        40% of TC done ... 416 spikes @ t = 0.400065s
        50% of TC done ... 509 spikes @ t = 0.500204s
        60% of TC done ... 609 spikes @ t = 0.601312s
        70% of TC done ... 705 spikes @ t = 0.703119s
        80% of TC done ... 804 spikes @ t = 0.80028s
        90% of TC done ... 897 spikes @ t = 0.901178s
        100% of TC done ... 1014 spikes @ t = 1.00023s
        spikes: 1014    time: 1.00023s   -> avg. rate: 5.06882 Hz

postprocessing ...
```

```
CPU time for simulation: 0s

************************** saving the results ***************************

writing results to: ../data/DataOut-5e87c799f7dcc1c9da6942aa0b791810.nc
*** Parameter export successful

CPU time for saving results: 0s

******************************** the end ********************************

CPU time overall: 0s
```

After the C++ program has terminated, the matlab script will now read in the netcdf file with the results and plot the spike train.

```
reading result netcdf file: ../data/DataOut-5e87c799f7dcc1c9da6942aa0b791810.nc
reading spike train
```

1. The figure that finally pops up should look like in Fig. 1a. If you change the AP onset rapidness $rap = 10$, the result will look like in Fig. 1b. Just by eye, one can see in comparison that the spike pattern of the neurons with the faster AP onset rapidness looks more asynchronous. If you want to see this more clearly, change the connectivity to $K = 150$ and rerun the simulations for both $rap = 1$ and $rap = 10$. The results are shown in Fig. 1c and 1d. There exists a critical connectivity for network of neurons with low AP onset rapidness at which a transition from an asynchronous to a synchronous state occurs.

2. In the next step we want to change the neuron model to the leaky integrate-and-fire (LIF) model $neuronType = 2$ (Fig. 1e) and the correlated leaky integrate-and-fire model with synaptic time constant $\tau_s = \tau_m/2$: $neuronType = 12$ (Fig. 1f, blue dots). The latter is basically implemented twice, namely again in the twoDlinear model $neuronType = 10$ (Fig. 1f, green circles). In this case a whole structure of parameters need to be specified, as the twoDlinear model is the generic model class for any two-dimensional neuron model with linear voltage dependence. The main difference in the simulation is that in the cLIF case ($neuronType = 11 \dots 14$), the neurons' spike times can be computed with a closed form expression, whereas in the twoDlinear case ($neuronType = 10$) the next spike time is found numerically with a root finding algorithm. The fact that the two generated spike trains coincide, confirms that the simulation of both neuron models is very precise.

3. If you set the firing rate to $f = 12.5$, the spike patterns of the two neuron models deviate after $8s$ warmup and $1.3s$ simulation (Fig. 1g). This is a nice demonstration of the sensitivity on initial conditions in the chaotic regime. The two spike patterns deviate because the next spike time calculation is implemented differently and the small difference on the order of the machine precision builds up and becomes macroscopically visible after some time. We will see that this is the chaotic regime below when calculating the Lyapunov exponents of the system. Can you find a parameter set for which the spike patterns deviate in the stable regime?

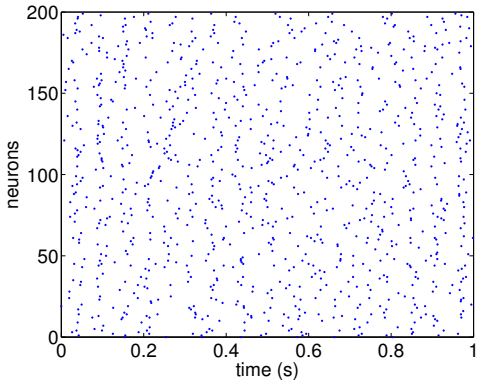
4. If we extend the synaptic time constant $tauS = tauM * 20$, the spike patterns change their shape because the input current variance changes ... (Fig. 1h).

3 Example: Lyapunov Spectrum

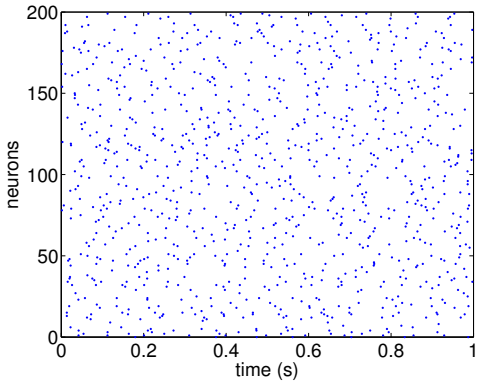
After, we have become a little bit familiar with LEquipe, let's calculate a Lyapunov spectrum. There are a couple of issues that need to be thought of. One of them is the convergence. Since the Lyapunov exponents are asymptotic quantities, the time should go to infinity. This is of course impossible in a numeric calculation. Nevertheless, we can stop the numerical calculation when we have good reasons that we think the Lyapunov spectrum has converged to its asymptotic shape. For every new system, we study, we should therefore do a convergence check, which will be explained here. Another feature is the convergence precision, which tests the largest and the smallest Lyapunov exponent for its convergence and ends the calculation when a certain precision is reach. This precision test start after the minimum number of spikes SC or time TC of the main calculation is done. More about that below.

So, let's get started and calculate the Lyapunov spectrum with the following matlab script.

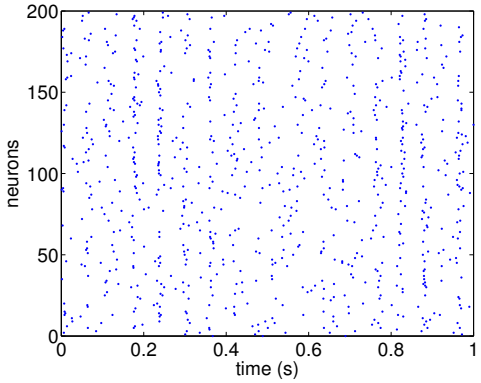
```
clear all
addpath('../code/')
set(0, 'defaultaxesfontsize', 20);
```



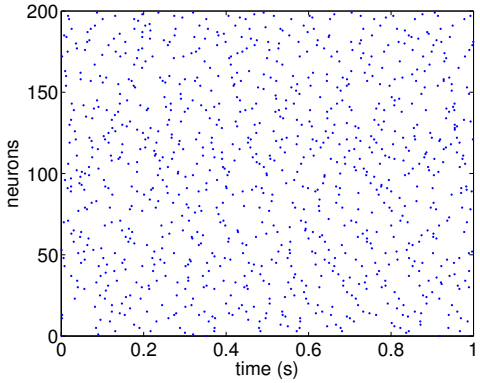
(a) Theta neurons $r = 1$.



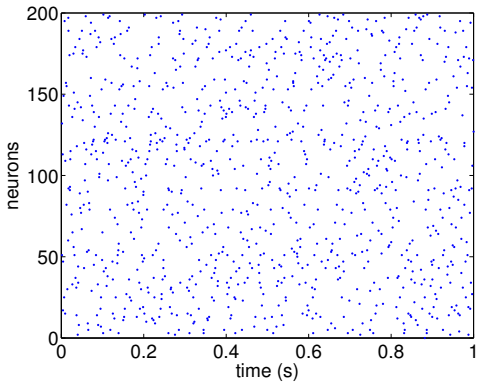
(b) Rapid theta neurons $r = 10$.



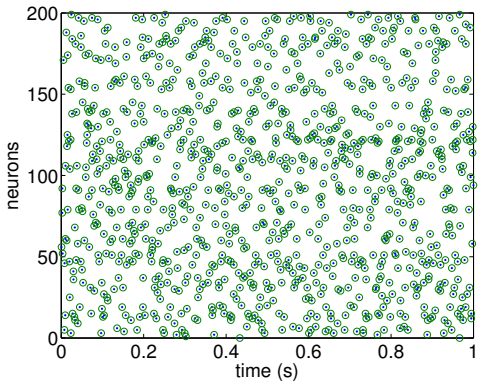
(c) Theta neurons $r = 1$ for $K = 150$.



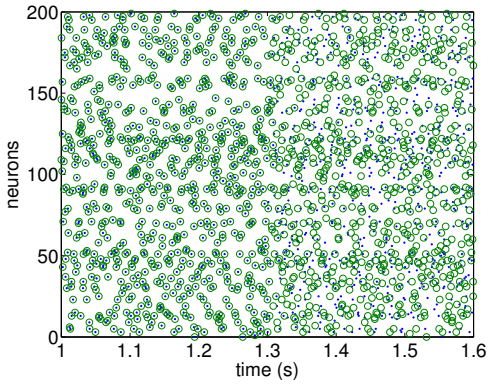
(d) Rapid theta neurons $r = 10$ for $K = 150$.



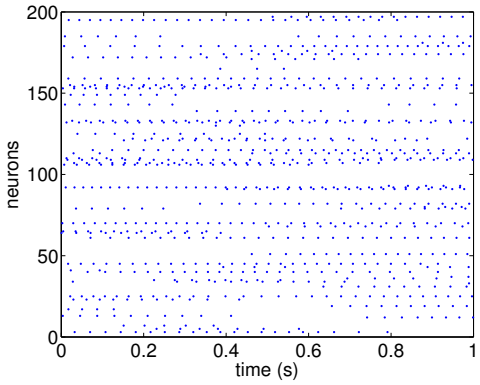
(e) Leaky integrate-and-fire neurons for $K = 150$.



(f) Correlated LIF neurons for $K = 150$.



(g) Correlated LIF neurons for $K = 150$ and $f = 12.5$.



(h) Correlated LIF neurons for $\tau_s = 20\tau_m$.

Figure 1: Examples of spike trains.

```
%%%%%%%%%%%%% define the parameters of the network here %%%%%%%%%%

neuronType = 1; %neuron type

N = 200;        %number of neurons
K = 50;         %number of synapses per neuron
J0 = -1;        %coupling strength
f = 5;          %network-averaged firing rate in Hz
tauM = 10;      %membrane time constant

rap = 1;        %AP onset rapidness in case of rapid theta neurons
tauS = tauM/2;  %synaptic time constant in case of cLIF or twoDlinear

%%%%%%%%%%%%%%%%%%%%%%%% end of input %%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set the given neuron parameters
ParaNet.N = N;
ParaNet.NeuronType = neuronType;
ParaNet.rapidness = rap;
ParaNet.tauM = tauM;

TwoDlinear.alpha = 1;
TwoDlinear.beta = 0;
TwoDlinear.gamma = 0;
TwoDlinear.delta = 1;
TwoDlinear.Cw = 0;
TwoDlinear.tauS = tauS;
ParaNet.twoDlinear = TwoDlinear;

%% set the random graph with K synapses per neuron on average
rand('twister', 1);
[ParaTopo.post ParaTopo.row_length] = random_graph(K, N);

%% set synapstic coupling strength (sqrt(K) scaling for the balanced state)
ParaTopo.J = J0/sqrt(K);

%% set the parameters of the simulation
ParaSim.rateWnt = f;        % this is the wanted firing rate
% the external currents that yield the wanted firing rate can be well
% approximated by the balance equation f = -I0/(J0*tauM)
% then with the balanced state scaling we end up with Iext = sqrt(K)*I0
ParaNet.Iext = -J0*f/1000*tauM*sqrt(K);

ParaSim.SW = 100;           % number of spikes per neuron during warmup

ParaSim.train = 1:N;        % neurons, whose spike times are saved

%Lyapunov exponent parameters
ParaSim.LyapunovExp = 1;    % number of Lyapunov exponents
ParaSim.SC = 10;            % avg. number of spikes per neuron in the calculation

%% write all parameters to netcdf files to directory data/ and get the hashes
directory = '../data/';
if ~exist(directory, 'dir')
    disp(['creating new directory: ' directory]);
    mkdir(directory)
end
```

```
[HashNet, FileNet] = writeNet(ParaNet, directory);
[HashTopo, FileTopo] = writeTopo(ParaTopo, directory);
[HashSim, FileSim] = writeSim(ParaSim, directory);
HashDataOut = DataHash([HashNet, HashTopo, HashSim]);
FileOut = [directory, 'DataOut-', HashDataOut, '.nc'];

%% run the C++ simulation
system(['../LEquipe ', FileNet, ' ', FileTopo, ' ', FileSim, ' ', FileOut]);

%% read the output file and plot the results
Data = readDataOut(FileOut);

disp(Data.LyapunovExponents(1));

figure;
subplot(2,2,1)
plot(Data.trainTime, Data.trainNeuron, '.', 'markersize', 5);
xlabel('time (s)');
ylabel('neurons')

subplot(2,2,2)
plot(1/ParaSim.LyapunovExp:1/ParaSim.LyapunovExp:1, Data.LyapunovExponents)
ylabel ('\lambda_i ( s ^{ -1})');
xlabel('i / N')
```

This script is essentially the same as the one above where we have calculated the spike trains, but with a new block to set up the calculation of the Lyapunov exponents.

1. The difference to the previous script that only calculated the spike trains are the following lines that make the C++ code also calculate the largest Lyapunov exponent during a simulation of 10 spike per neuron on average.

   ```
   %Lyapunov exponent parameters
   ParaSim.LyapunovExp = 1;
   ParaSim.SC = 10;
   ```

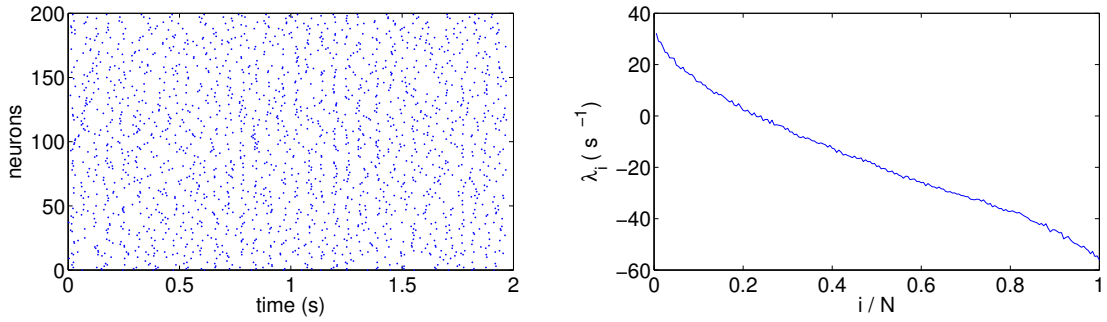
   The output of the program returns the largest Lyapunov exponent, in this case $32\,s^{-1}$. Setting $ParaSim.LyapunovExp$ to 2 would calculate the first two Lyapunov exponents and so on. This is interesting in case of a stable dynamics, e.g. for the LIF networks, where the first Lyapunov exponent is always zero, as the corresponding vector is tangent to the trajectory. Then the second exponent is called the 'largest' Lyapunov exponent.

2. If we want to calculate the whole Lyapunov spectrum, we have to replace the line $ParaSim.LyapunovExp = 1;$ with the following lines:

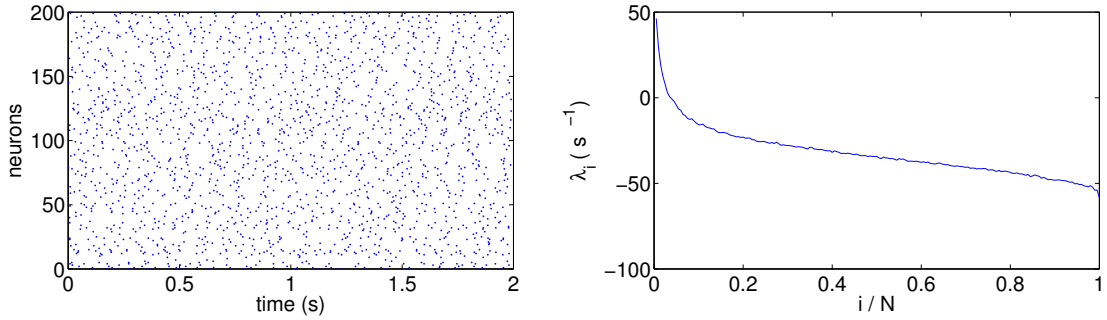
   ```
   %Lyapunov exponent parameters
   if ParaNet.NeuronType < 10
       ParaSim.LyapunovExp = ParaNet.N;
   else
       ParaSim.LyapunovExp = 2*ParaNet.N;
   end
   ```

   The if statement sets the number of Lyapunov epxonents to be calculated. In case of the phase neurons, there are N Lyapunov epxonents and in case of the 2-dimensional neuron models, there are $2N$ Lyapunov exponents making up the Lyapunov spectrum. The result is displayed in Fig. 2a. Calculate the whole Lypunov spectrum for rapid theta neurons with AP onset rapidness $rap = 10$ (result shown in Fig. 2b).

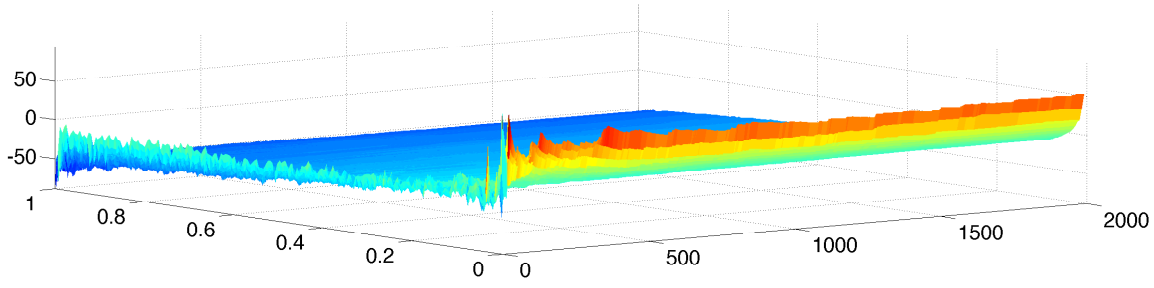
3. So far, we haven't considered the convergence criteria. Fortunately SC was long enough, such that the Lyapunov spectra were already pretty well converged. Let's take a look at how the exponents converge over time to their asymptotic shape. In order to do that, add $ParaSim.LyapunovExpConvergence = 0;$ to the Lyapunov exponent parameters after the end of the previously described if statement. Additionally, add the following lines to the end of the script



(a) Theta neuron $r = 1$



(b) rapid theta neuron $r = 10$



(c) rapid theta neuron $r = 10$

Figure 2: Some example Lyapunov spectra

```
subplot(2,2,3:4)
Data.LEtimes(1) = [];
Data.LEconvergence(1,:) = [];
time = repmat(Data.LEtimes, 1, size(Data.LEconvergence, 2));
index = repmat(1/size(Data.LEconvergence, 2):1/size(Data.LEconvergence, 2):1,
size(Data.LEconvergence, 1), 1);
surf(time, index, Data.LEconvergence);
shading flat;
zlim([min(min(Data.LEconvergence)) max(max(Data.LEconvergence))]);
```

There is a large peak at the very beginning of the calculation and therefore the convergence of the Lyapunov spectrum is hardly visible.
In the first couple of calculations, the Gram–Schmidt–vectors need to align with the right subspaces and do not really reflect the properties for the Lyapunov exponents. It is therefore wise to do a warmup of the Gram–Schmidt–vectors before the calculation of the Lyapunov spectrum is started. To do so, add the line $ParaSim.SWONS = 10$; to the Lyapunov exponent parameters. The result can be seen in Fig. 2c. Also, try out the LIF spectrum with and without the warmup.
This seems to achieve a pretty good convergence. We can set the precision of the largest and the smallest Lyapunov exponents with $ParaSim.pLE = 0.001$; Add this to the Lyapunov exponent parameters and rerun the script. What changes and what does that mean?

4. In order to speed up the simulation, we can skip a couple of reorthonormalization procedures. These are expensive as the computation time scales with N^3. As long as the Jacobian has a moderate condition number, the orthonormal system does not need to be reorthonormalized. In case of the sparse random graphs, we can estimate the number of spikes after which the product of the single spike Jacobians is not sparse but full. This is approximately after N/K spikes. Add the line $ParaSim.ONstep = 10 * N/K$; to the Lyapunov exponent parameters. This tells the C++ code to start with a reorthonormalization steps size of 40. It will then estimate the condition number and reduce the step size until a moderate condition number is achieved (the error of the orthonormalization procedure scales linear with the condition number). After running the script, there will be this additional check of the condition number

```
optimize ON step size (maximal conditional number = 42)...
        the average condition number with ON steps = 40 is 183.501
        the average condition number with ON steps = 20 is 121.352
        the average condition number with ON steps = 10 is 3.28798
        setting ON step size to 10
```

In this procedure, the orthonormalization step size is adjusted and set to 10. This should lead to a considerable speedup of the simulation. Compare the computation times with and without the larger step size. Does the Lyapunov spectrum change?

5. Another convergence criteria you might want to check is different starting orthonormal systems. The seed for the random number generator to create the starting orthonormal system can be controlled with $ParaSim.seedONS = seed$;. You should should also check, whether different initial conditions of the trajectory ($ParaNet.seedInit = seed$;) or different realizations of the random graph ($rand('twister', seed)$; before $random\_graph(K, N)$; is called) change the Lyapunov spectrum.

4 Example: Covariant Lyapunov vectors

The information that is being calculated during the calculation of the asymptotic Lyapunov exponents can be used to calculate the covariant Lyapunov vectors. Consult the following references for more details:

1. F. Ginelli, P. Poggi, A. Turchi, H. Chaté, R. Livi, and A. Politi, Phys. Rev. Lett. 99, 130601 (2007)
2. Hong-liu Yang and Günter Radons, Phys. Rev. E 82, 046204 (2010)

The covariant Lyapunov vectors can be calculated with the following script.

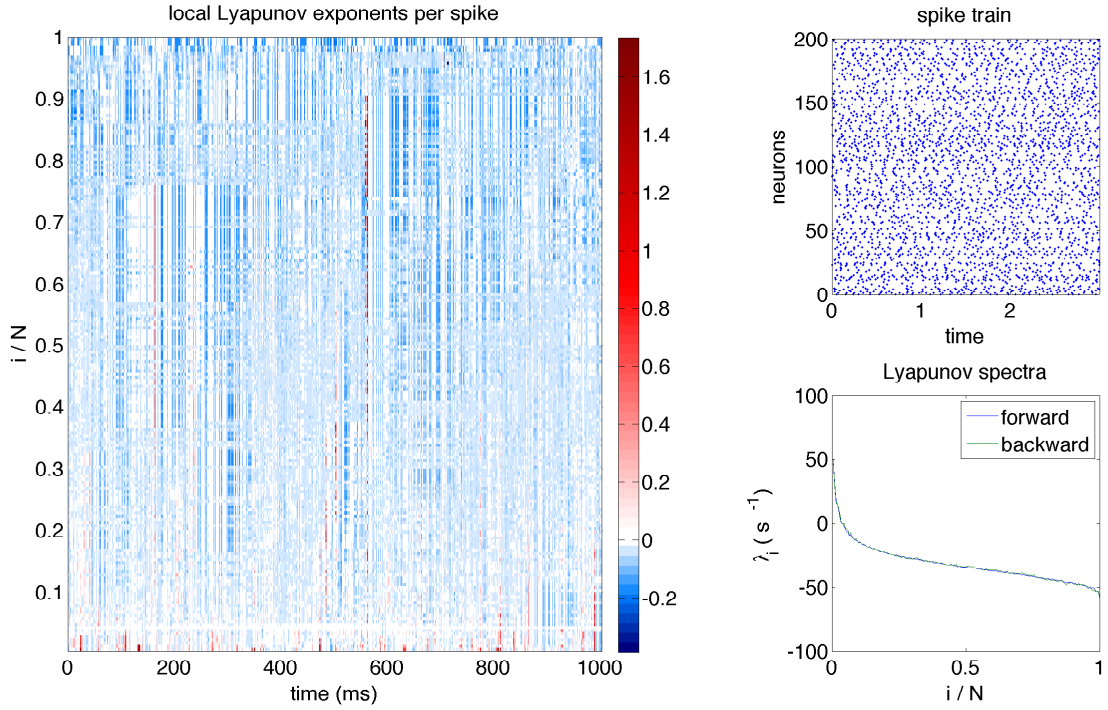
```
clear all
addpath('../code/')
set(0, 'defaultaxesfontsize', 20);
```

```
%%%%%%%%%%%%% define the parameters of the network here %%%%%%%%%%

neuronType = 1; %neuron type

N = 200;        %number of neurons
K = 50;         %number of synapses per neuron
J0 = -1;        %coupling strength
f = 5;          %network-averaged firing rate in Hz
tauM = 10;      %membrane time constant

rap = 10;        %AP onset rapidness in case of rapid theta neurons
tauS = tauM/2;  %synaptic time constant in case of cLIF or twoDlinear

%%%%%%%%%%%%%%%%%%%%%%%% end of input %%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% set the given neuron parameters
ParaNet.N = N;
ParaNet.NeuronType = neuronType;
ParaNet.rapidness = rap;
ParaNet.tauM = tauM;

TwoDlinear.alpha = 1;
TwoDlinear.beta = 0;
TwoDlinear.gamma = 0;
TwoDlinear.delta = 1;
TwoDlinear.Cw = 0;
TwoDlinear.tauS = tauS;
ParaNet.twoDlinear = TwoDlinear;

%% set the random graph with K synapses per neuron on average
rand('twister', 1);
[ParaTopo.post ParaTopo.row_length] = random_graph(K, N);

%% set synapstic coupling strength (sqrt(K) scaling for the balanced state)
ParaTopo.J = J0/sqrt(K);

%% set the parameters of the simulation
ParaSim.rateWnt = f;        % this is the wanted firing rate
% the external currents that yield the wanted firing rate can be well
% approximated by the balance equation f = -I0/(J0*tauM)
% then with the balanced state scaling we end up with Iext = sqrt(K)*I0
ParaNet.Iext = -J0*f/1000*tauM*sqrt(K);

ParaSim.SW = 100;           % number of spikes per neuron during warmup

ParaSim.train = 1:N;        % neurons, whose spike times are saved

%Lyapunov exponent parameters
if ParaNet.NeuronType < 10
    ParaSim.LyapunovExp = ParaNet.N;
else
    ParaSim.LyapunovExp = 2*ParaNet.N;
end
ParaSim.SC = 15;            % avg. number of spikes per neuron in the calculation
ParaSim.SWONS = 10;         % warmup of the ONSE
ParaSim.ONstep = 1;         % orthonormalization step size

%covariant Lyapunov vectors
```

```
ParaSim.CLV = 1;            % calculate the CLVs
ParaSim.SWCLV = 10;         % warmup of the CLVs (must be < ParaSim.SC)

%% write all parameters to netcdf files to directory data/ and get the hashes
directory = '../data/';
if ~exist(directory, 'dir')
    disp(['creating new directory: ' directory]);
    mkdir(directory)
end

[HashNet, FileNet] = writeNet(ParaNet, directory);
[HashTopo, FileTopo] = writeTopo(ParaTopo, directory);
[HashSim, FileSim] = writeSim(ParaSim, directory);
HashDataOut = DataHash([HashNet, HashTopo, HashSim]);
FileOut = [directory, 'DataOut-', HashDataOut, '.nc'];

%% run the C++ simulation
system(['../LEquipe ', FileNet, ' ', FileTopo, ' ', FileSim, ' ', FileOut]);

%% read the output file and plot the results
Data = readDataOut(FileOut);

figure;

subplot(2,3,3)
plot(Data.trainTime, Data.trainNeuron, '.', 'markersize', 5);
xlabel('time');
ylabel('neurons')
xlim([0 max(max(Data.trainTime))])
title('spike train');

subplot(2,3,6)
plot(1/ParaSim.LyapunovExp:1/ParaSim.LyapunovExp:1, Data.LyapunovExponents)
hold all;
plot(1/ParaSim.LyapunovExp:1/ParaSim.LyapunovExp:1, Data.LEclv)
ylabel ('\lambda_i ( s ^{ -1})');
xlabel('i / N')
title('Lyapunov spectra')
legend(['forward '; 'backward'], 'Location', 'Northeast');


subplot(2,3,[1:2 4:5])
time = repmat(Data.LEtimes(1:size(Data.localLE, 1)), 1, size(Data.localLE, 2));
index = repmat(1/size(Data.localLE, 2):1/size(Data.localLE, 2):1, size(Data.localLE, 1), 1);
pcolor(time, index, Data.localLE);
shading flat;
zlim([min(min(Data.localLE)) max(max(Data.localLE))]);
xlabel('time (ms)');
ylabel('i / N')
title('local Lyapunov exponents per spike')
colormap(bluewhitered);
colorbar;
```

1. There are only two new lines

```
%covariant Lyapunov vectors
ParaSim.CLV = 1;
ParaSim.SWCLV = 10;
```



(a) rapid theta neuron $r = 10$

Figure 3: Example local Lyapunov exponents

The first line after the comment turns on the covariant Lyapunov vector (CLV) calculation and the second one set the length of the warmup of the CLVs in spikes. When the C++ program is called it follows the normal procedure as before but stores the local projection matrices R in the Gram–Schmidt–orthonormalization. After this forward calculation is done, the projection matrices are used to calculate the Lyapunov vectors in a backward calculation. The principle idea is that backward in time, random vectors in the eigenbasis spanned by the forward Gram–Schmidt–basis align with the least converging directions forward in time (see the literature above for a detailed explanation). The information stored in the covariant Lyapunov vectors about the dynamics is manifold.
Here we calculated the local Lyapunov exponents per spike (Fig. 3a). The local Lyapunov exponents are the local diverging/converging rates per spike. It is per spike here, since $ONstep = 1$. This is smallest interval where stretching can occurr, since between spikes, all neurons phases just evolve with constant phase velocities. If you increase $ONstep$, then the Lyapunov vectors are not really a local property of those points in phase space anymore.
Can you see the zero Lyapunov exponent index whose vector aligns tangent to the trajectory? Why are there positive local exponents in the negtive part of the spectrum and vice versa? What are the correlations?
In Fig. 3a, there is also shown a comparison between the Lyapunov spectrum obtained in the standard procedure (forward) and the one obtained from the diverging/converging rates of the Lyapunov vectors (backward). These spectra should coincide, as a health check.

2. Other quantitites that can be calculated from the Lyapunov vectors are:

- participation ratios
- angles between the vectors and thus the angle between the stable and unstable manifold which is important to check whether the system is hyperbolic, see e.g. Masanobu Inubushi, Miki U. Kobayashi, Shin-ichi Takehiro, and Michio Yamada, Phys. Rev. E 85, 016331 (2012)

5 Example: Spike Statistics

One feature of LEquipe is to calculate the statistics of the firing patterns that are being simulated. Although most of them could in principle be calculated from the spike trains in, e.g., Matlab, there are internal functions for

their calculation in the C++ program. The firing statistics are obtained during the same simulation as, e.g., the Lyapunov exponent calculation. This means that if both the Lyapunov exponents and the firing rates are measured, they describe the exact same trajectory. This has the advantage that one can be sure that the measurement is for the same trajectory and one does not need to rely on a repition of simulations in which one might change by mistake one parameter. The disadvantage is that for the spike statistics of individual neurons, one needs to consider a lot more spikes in the network (say 1000 spikes per neuron on average) than for the calculation of network properties such as the Lyapunov exponents (say 10 spikes per neuron on average).

Run the following script to determine the individual firing rates of all neurons in the network.

```
clear all
addpath('../code/')
set(0, 'defaultaxesfontsize', 20);

%%%%%%%%%%%%% define the parameters of the network here %%%%%%%%%%

neuronType = 1; %neuron type

N = 200;        %number of neurons
K = 50;         %number of synapses per neuron
J0 = -1;        %coupling strength
f = 5;          %network-averaged firing rate in Hz
tauM = 10;      %membrane time constant

rap = 1;        %AP onset rapidness in case of rapid theta neurons
tauS = tauM/2;  %synaptic time constant in case of cLIF or twoDlinear

%%%%%%%%%%%%%%%%%%%%%%%%%% end of input %%%%%%%%%%%%%%%%%%%%%%%%%%

%% set the given neuron parameters
ParaNet.N = N;
ParaNet.NeuronType = neuronType;
ParaNet.rapidness = rap;
ParaNet.tauM = tauM;

TwoDlinear.alpha = 1;
TwoDlinear.beta = 0;
TwoDlinear.gamma = 0;
TwoDlinear.delta = 1;
TwoDlinear.Cw = 0;
TwoDlinear.tauS = tauS;
ParaNet.twoDlinear = TwoDlinear;

%% set the random graph with K synapses per neuron on average
rand('twister', 1);
[ParaTopo.post ParaTopo.row_length] = random_graph(K, N);

%% set synapstic coupling strength (sqrt(K) scaling for the balanced state)
ParaTopo.J = J0/sqrt(K);

%% set the parameters of the simulation
ParaSim.rateWnt = f;        % this is the wanted firing rate
% the external currents that yield the wanted firing rate can be well
% approximated by the balance equation f = -I0/(J0*tauM)
% then with the balanced state scaling we end up with Iext = sqrt(K)*I0
ParaNet.Iext = -J0*f/1000*tauM*sqrt(K);

ParaSim.SW = 100;           % number of spikes per neuron during warmup
ParaSim.SC = 10;            % average number of spikes per neuron
```

```
ParaSim.ISIneurons = 1:ParaNet.N;          % neurons whose spike statistics are calculated
ParaSim.ISIstats = 1;                      % kinda moments that are being calculated

%% write all parameters to netcdf files to directory data/ and get the hashes
directory = '../data/';
if ~exist(directory, 'dir')
    disp(['creating new directory: ' directory]);
    mkdir(directory)
end

[HashNet, FileNet] = writeNet(ParaNet, directory);
[HashTopo, FileTopo] = writeTopo(ParaTopo, directory);
[HashSim, FileSim] = writeSim(ParaSim, directory);
HashDataOut = DataHash([HashNet, HashTopo, HashSim]);
FileOut = [directory, 'DataOut-', HashDataOut, '.nc'];

%% run the C++ simulation
system(['../LEquipe ', FileNet, ' ', FileTopo, ' ', FileSim, ' ', FileOut]);

%% read the output file and plot the results
Data = readDataOut(FileOut);

figure;
plot(Data.rateNeurons, '.', 'markersize', 7);
xlabel('neuron index');
ylabel('firing rate (Hz)')
```

1. After completing the simulation, a plot appears with the neurons firing rates versus the neuron index. You can plot a histogram with 20 bins with the matlab command

   ```
   hist(Data.rateNeurons, 20)
   ```

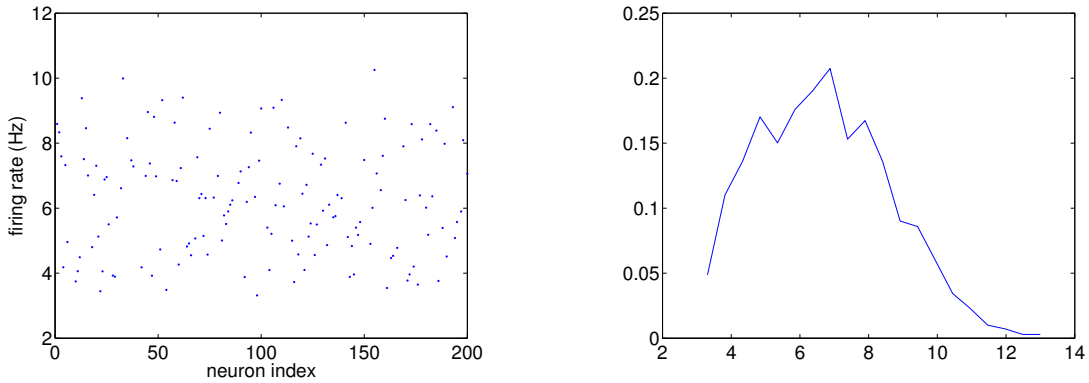
   To get some more data points for the histogram, increase the number of neurons to $N = 2000$;. The histogram with 20 bins can also automatically be calculated with the command $ParaSim.ISIbins = 20$; Insert that under $ParaSim.ISIstats = 1$; in the matlab script. The results are displayed in Fig. 4a.

2. The paramater $ParaSim.ISIstats$; defines the 'kinda' moments of the inter-spike-interval distribution of the individual neurons. It's called 'kinda' because it's related to the moments but the more relevant quantities with respect to neural networks are saved. With the moments $\mu_n = E(t_{\text{isi}}^n)$, where t_{isi} denote the inter-spike-interval of the neuron, and the standard deviation $\sigma = \sqrt{\mu_2 - \mu_1^2}$, the kinda moments are:

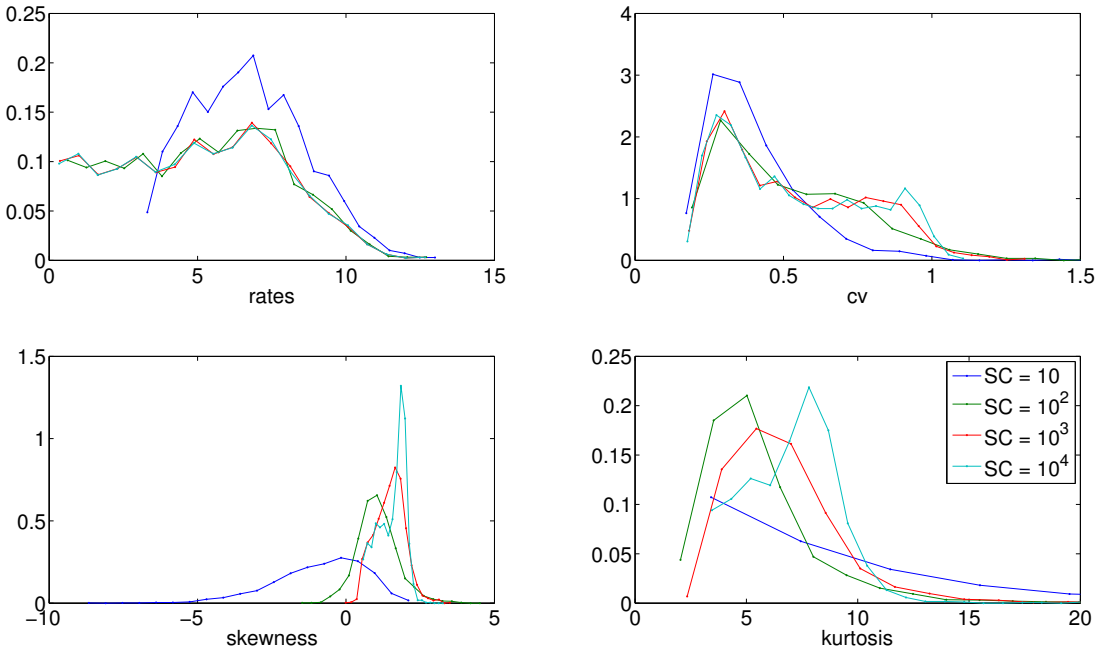
the average firing rate	$ParaSim.ISIstats = 1$	$\nu = 1/\mu_1$
the coefficient of variation	$ParaSim.ISIstats = 2$	$cv = \sigma/\mu_1$
the skewness (third standardized moment)	$ParaSim.ISIstats = 3$	$\gamma_1 = \mu_3/\mu_1^3$
the kurtosis (fourth standardized moment)	$ParaSim.ISIstats = 4$	$\beta_2 = \mu_4/\mu_1^4$

   Change $ParaSim.ISIstats = 4$; to calculate all four moments and plot them with

   ```
   figure;

   subplot(2,2,1)
   plot(Data.rateDistX, Data.rateDistY, '.-');
   xlabel('rates');

   subplot(2,2,2)
   plot(Data.cvDistX, Data.cvDistY, '.-');
   xlabel('cv');

   subplot(2,2,3)
   plot(Data.skewnessDistX, Data.skewnessDistY, '.-');
   xlabel('skewness');
   ```



(a) firing rates of all neurons and the histogram for a theta neuron network



(b) firing rates of all neurons and the histogram for a theta neuron network

Figure 4: Examples inter-spike-interval statistics

```
subplot(2,2,4)
plot(Data.kurtosisDistX, Data.kurtosisDistY, '.-');
xlabel('kurtosis');
```

3. Because the moments are only accurate for a large enough sample, in our case the number of spikes, 10 spikes per neuron on average for the calculation are not enough. The faster spiking neurons fire more often than 10 times and the statistics is fine. The neurons with a low firing rate, however, will only fire a few spikes and their statistics is bad. Only neurons that fire more than 6 spikes are considered in the calculation of the moments anyway. You should therefore increase the number of spikes per neuron *ParaSim.SC* to reproduce Fig. 4b. How far *SC* needs to be cranked up depends on the distribution of firing rates. The broader the firing rate distribution in the direction of low firing rates, the longer the simulation need to be.

6 Temporally changing external currents

7 Example: Phase Perturbations