Programming spiking neural networks with Scilab

Romain Brette
Ecole Normale Supérieure, Paris
`brette@di.ens.fr`

September 12, 2006

The last version of this document can be found at
`http://www.di.ens.fr/~brette/ScilabNeuron.pdf`.

This is a short notice documenting the Scilab files that implement the first two benchmarks described in the following review paper:

Simulation of networks of spiking neurons: A review of tools and strategies (2006). Brette, Rudolph, Carnevale, Hines, Beeman, Bower, Diesmann, Goodman, Harris, Zirpe, Natschlger, Pecevski, Ermentrout, Djurfeldt, Lansner, Rochel, Vibert, Alvarez, Muller, Davison, El Boustani and Destexhe. Journal of Computational Neuroscience.

These benchmarks are random networks of integrate-and-fire neurons. The reader is advised to read this notice along with the code (CUBA, COBAEuler or COBARungeKutta).

1 About Scilab

Scilab is a free vector-based scientific software (resembling Matlab) which can downloaded at `http://www.scilab.org`. It features an interpreted programming language, plotting tools and many mathematical functions. Scilab can be programmed as a standard imperative language, using loops and conditions. However, using loops is usually very slow and Scilab was designed to be efficient when dealing with vectors. The key to efficient (and elegant) Scilab programming is to replace loops by vector-based operations. Documentation and tutorials are available on the web site. There is an exhaustive on-line help system that can be queried with `help name_of_command`.

2 Data structures for spiking neural networks

Basic concepts and algorithms are described in the review paper mentioned in the introduction. Here I am only considering clock-driven simulation with no transmission delay.

2.1 State matrix

If there is no transmission delay, then spike times need not be stored. Each neuron can be described by a number of state variables. One special state variable is the membrane potential. Thus the state of a neuron can be stored in a row vector with p elements, where the first element is the membrane potential. Then the state of a network of N neurons can be stored in an $N \times p$ matrix, with each row being the state vector of a neuron. For example, the following command creates the state matrix of a network with N neurons and 3 state variables per neuron:

```
S=zeros(N,3);
```

2.2 Weight matrix

The synaptic weights can be described by an $N \times N$ matrix, where the value at row j and column i is the synaptic weight of connection $i \rightarrow j$. However, not all neuron pairs are connected, so that most entries of the weight matrix are null, i.e., the matrix is *sparse*. Scilab has special built-in functions and structures for sparse matrices. Only non-zero elements of a sparse matrix are stored. A nice feature of Scilab is that it deals with sparse matrices in a transparent way, i.e., once the matrix is defined as sparse, one can use all standard matrix operations as if it was a usual matrix. The following command creates a sparse matrix of 0s and 1s with 2% of its elements with value 1:

```
W=bool2s(sprand(N,N,0.02,'uniform')>0);
```

Even when it is sparse, the weight matrix can be very large, which can result in an error message from Scilab. The default memory allocated to the variables is quite low, and it can be increased by the command `stacksize`.

2.3 Spike times

An easy way to store spike times is to use a matrix with two columns, the first one corresponding to the neuron number, the second one to the spike timing. The following commands initialize the matrix and insert a spike defined by the time t and neuron number i.

```
spiketimes=[];
spiketimes=[spiketimes;i,t];
```

3 Programming a network simulation

A step of a clock-driven algorithm consists in 1) state updates and 2) propagation of spikes.

3.1 State updates

For linear models (current-based models), the state of all neurons in the network are simply updated by the matrix operation `S=S*A`, where S is the state matrix and A is the update matrix. This is the same as doing `S(i,:)=S(i,:)*A` for each neuron i, where `S(i,:)` is the row-vector of states of neuron i. For non-linear models, one uses an integration method such as Euler or Runge-Kutta (examples are in files COBAEuler and COBARungeKutta).

3.2 Propagation of spikes

The following command:

```
spikes=find(S(:,1)>Vt);
```

returns a index vector of all neurons whose membrane potential is greater than the threshold Vt. Then `W(:,spikes)` is the submatrix of the weight matrix made of all synapses $i \to j$ such that neuron i is emitting a spike. Then `sum(W(:,spikes),'c')` is a column-vector containing the total synaptic input received by each neuron. If synaptic interactions act on the second state variable, then the following command updates the state of all neurons receiving a spike:

```
S(:,2)=S(:,2)+sum(W(:,spikes),'c');
```

In general, one would have to distinguish between excitatory and inhibitory synapses, which act on distinct variables (see sample code).

After spiking, the membrane potential is reset: `S(spikes,1)=Vr;`.

Spike times are inserted as follows:

```
spiketimes=[spiketimes;spikes',ones(spikes')*t];
```

Note that `spikes'` is a column-vector containing the indexes of all spiking neurons and `ones(spikes')*t` is a column-vector with the same size as `spikes'` and all entries with value t.

4 Concluding remarks

The time-consuming operation here is the propagation of spikes, more precisely the command `S(:,2)=S(:,2)+sum(W(:,spikes),'c')`. Simulations with Scilab are typically much slower than with a compiled language such as C but still acceptable for medium-scale network simulations, all the more as writing a program can be very fast once one masters the concepts of vector-based programming.

Although Scilab programs are interpreted, simulation time is acceptable because for clock-driven algorithms, all operations can be vectorized. Event-driven algorithms can also be programmed with Scilab, but as they cannot be fully vectorized, their performance is very limited compared to compiled languages.