NeuronRK package

1 Introduction

This package contains a library of C++ functions for the dynamical simulation of networks of Hodgkin-Huxley-type of neurons. There is a broad variety of such models for many different types of cells. They all contain the core components of the Hodgkin-Huxley model (sodium, potassium, and leak currents) and extend this basic formalism by including different sorts of other ionic currents, levels of compartmentalization, *etc.* Complexity levels of these models reflect different degrees of physiological relevance in such a way that they naturally build up on top of each other, progressing from simple to more detailed. This makes the object-oriented programming an ideal formalism for implementation of such type of models.

2 Design

The two fundamental classes used in the package are NETWORK and NEURON. NETWORK is used as an enveloping object to keep all the information about elements of the network, time, simulation status *etc.* It contains an array of NEURONS, a counter to track the number of elements in the array, and the **cycle** method which is used to call the corresponding **cycle** methods in each element of the array. The **cycle** method executes one step of the dynamical simulation. In each NEURON, the corresponding **cycle** method computes all ionic and synaptic currents, calculates the right-hand side of the dynamic equations, and updates the dynamic variables.

The class called NEURON contains dynamic variables of not just one neuron but those of a *population* of similar neurons (for instance, a population of pyra-

midal cells). NETWORK may contain several such populations. The dynamic variables are organized as a $n \times m$ array, where n is the number of cells and m is the number of variables. We assume that all neurons within one population (i) are of the same type, i.e., obey the same dynamic equations, (ii) are connected all-to-all by chemical synapses (we assume that synaptic conductances are the same across population; zero conductance is used for uncoupled cells), (iii) are connected all-to-all by electrical (gap junction) synapses (these conductances are also assumed to be the same), and (iv) can have autapses. Whenever we say that two populations of neurons are connected by a synapse, we assume all-to-all connections between their corresponding cells with the same value of synaptic conductance. To track the connections between populations, we use an array of external synaptic inputs (pointers to other NEURONS) and the corresponding arrays of external synaptic and external gap junction conductances. This design allows important simplifications that are critical for effective simulations of large networks and dramatically improve the computation time as explained below.

Let Ω be the set of populations (NEURONS) in the NETWORK. Consider a NEURON $\alpha \in \Omega$ and let $i \in \alpha$ be one cell in α. The sum of synaptic inputs to i contains two components: $I_{syn,i} = I^{int}_{syn,i} + I^{ext}_{syn,i}$, where $I^{int}_{syn,i}$ is the sum of synaptic inputs to i from within population and $I^{ext}_{syn,i}$ is the sum of synaptic inputs to i from other populations. Specifically,

$$I^{int}_{syn,i} = \sum_{j \in \alpha, j \neq i} g_{j \to i} s_j (V_i - V_{rev,j}) + g_{i \to i} s_i (V_i - V_{rev,i}),$$

where $g_{j \to i}$, s_j, V_i, and $V_{rev,j}$ are the synaptic conductance between cells j and i, the synaptic gating variable, the membrane potential and the reversal potential, respectively. We assume that the synaptic conductances are the same across population, i.e., $g_{j \to i} = g_\alpha$. The reversal potential $V_{rev,i} = V_{rev,j} = V_{rev,\alpha}$ is also independent of i or j. Thus, we have

$$I^{int}_{syn,i} = g_\alpha \left(\sum_{j \in \alpha} s_j - s_i \right) (V_i - V_\alpha) + g_{self,\alpha} s_i (V_i - V_\alpha) =$$

$$= g_\alpha (s_\alpha - s_i)(V_i - V_\alpha) + g_{self,\alpha} s_i (V_i - V_\alpha),$$

where $s_\alpha = \sum_{j \in \alpha} s_j$ and $g_{self,\alpha}$ is the conductance of the autapse. Similarly, the

sum of external synaptic currents is

$$I_{syn,i}^{ext} = \sum_{\beta\in\Omega}\sum_{j\in\beta} g_{j\to i}s_j(V_i - V_{rev,j}) = \sum_{\beta\in\Omega} g_{\beta\to\alpha}(V_i - V_{rev,\beta})\left(\sum_{j\in\beta} s_j\right) =$$

$$= V_i\sum_{\beta\in\Omega} g_{\beta\to\alpha}s_\beta - \sum_{\beta\in\Omega} g_{\beta\to\alpha}s_\beta V_{rev,\beta},$$

where the synaptic conductance $g_{j\to i}$ is the same for all connections between these two populations ($g_{j\to i} = g_{\beta\to\alpha}$) and the reversal potential $V_{rev,j}$ depends only on β ($V_{rev,j} = V_{rev,\beta}$). The sum of gap junction currents is $I_{elc,i} = I_{elc,i}^{int} + I_{elc,i}^{ext}$, where

$$I_{elc,i}^{int} = \sum_{j\in\alpha} e_{j\to i}(V_j - V_i) = e_\alpha\left(\sum_{j\in\alpha} V_j - n_\alpha V_i\right) = e_\alpha V_\alpha - e_\alpha n_\alpha V_i,$$

$$I_{elc,i}^{ext} = \sum_{\beta\in\Omega}\sum_{j\in\beta} e_{j\to i}(V_j - V_i) = \sum_{\beta\in\Omega} e_{\beta\to\alpha}\left(\sum_{j\in\beta} V_j - n_\beta V_i\right) =$$

$$= \sum_{\beta\in\Omega} e_{\beta\to\alpha}V_\beta - V_i\sum_{\beta\in\Omega} e_{\beta\to\alpha}n_\beta,$$

where e_α is the gap junction conductance within population, $e_{j\to i}$ is the gap junction conductance between cells j and i (as before, $e_{j\to i} = e_{\beta\to\alpha}$), $V_\alpha = \sum_{j\in\alpha} V_j$, and n_β is the number of cells in β.

The computations are organized such a way that we first compute the values of s_α and V_α for each population and then compute $\sum_{\beta\in\Omega} g_{\beta\to\alpha}s_\beta$, $\sum_{\beta\in\Omega} g_{\beta\to\alpha}s_\beta V_{rev,\beta}$, $\sum_{\beta\in\Omega} e_{\beta\to\alpha}V_\beta$, and $\sum_{\beta\in\Omega} e_{\beta\to\alpha}n_\beta$ for all populations that are connected. These computations are executed in the **cycle** method of the class NETWORK. As a result, the computation of the synaptic current, which otherwise involves combinations of all pairs of cells in the network, takes linear time with respect to the total size of the network and results in a dramatic speedup of the overall simulation time.

Every NEURON contains a method called **f**. Its puprose is to compute the right-hand side of the ODE for every cell in the population (the dynamical equations reside here). Different cell models have different dynamic equations and, consequently, different dynamic variables and different **f** methods. One of the advantages of using the object-oriented paradigm is that **f** methods can be overridden during class inheritance, while other methods which are not related to

specific details of the model can be inherited from the enveloping class NEURON without overriding.

One of the "housekeeping" methods, which is inherited without overriding, is **RungeKutta**. It calls the **f** method for every cell in the population and updates its dynamic variables. **RungeKutta** is called from the **cycle** method of each NEURON, which, in turn, is called by the **cycle** method of the NETWORK. Another housekeeping method called **track_spikes** is used to detect spike times. It is paired with the **cycle** method and updates the array of spike times for each cell in the population. Note that we define the time of the spike as a midpoint between two time points, at which the membrane potential of a cell crosses the +10 mV threshold.

Additionally, NEURON contains the **report** method, which prints out the values of the dynamic variables in a tab-delimited text file. To reduce the size of the output, this method is executed only if the membrane potential changes by more than 0.1 mV compared to the previously reported value. At that, even if a change in the membrane potential was detected in only one cell, **report** is executed for all cells in the network to keep the rectangular shape of the output matrix.

Now the computation of a spike time response curve is nothing but a loop that repetetively applies a short DC current pulse to trigger exactly one spike in the presynaptic cell and measures the effect of this spike on the spike time of the postsynaptic cell by using the **track_spikes** method. This loop is executed in the method called **getprc**, which logically belongs to the class NETWORK since it involved different types of cells (for instance, an O-cell and an I-cell, or a an O-cell and a PING network which consists of an E-cell and an I-cell).