Urd: a Matlab tool for simulating simple neural networks

Sid Visser

May 19, 2012

1 Introduction

Computational modeling of neural networks in the mammalian nervous system can provide a better understanding of some of the complex functions that we observe. Depending on the structure that is studied, these models can grow both in size and complexity. Clearly, if the model design is more complex, implementing the model on a computer will be a tedious and time consuming task.

Several software packages exist that offer an extended environment for simulating neurons and networks thereof, such as Neuron, GENESIS, PyNN, and NEST. These environments are typically very large programs that offer many specialized features to their users, for instance: subtle changes in ion-channel dynamics and very realistic 3d reconstructions of axonic and dendritic structures. A clear disadvantage arises here: if your model design is a simple network of neurons, implementing it in these dedicated packages can already be hard; especially if you are unfamiliar with the language and its features.

To overcome this problem, it seems natural to have a simulation environment available in Matlab as well, since this programming language is very accessible. Furthermore, Matlab is already accessible at most research institutes, so installation of new software could be avoided in this manner. A draft of this package, with the name Urd, is now available.

2 Program structure

The Matlab programming language is optimized for vectorized equations. While the ODEs for the neurons are easily vectorized (since all neurons obey similar dynamical systems), using a default Matlab ODE solver, such as ode45 or ode23s, is far from optimal. This is mainly because the neuron dynamics are stiff, sometimes discontinuous, and all neurons are in different phases relatively to each other. Furthermore, if axonic delays are introduced in the system the simulation times will increase significantly.

For that reason Urd is a Matlab interface for the Verdandi simulation environment. The latter one is a custom C++ simulation program specialized for simple neural networks. This offers the following program structure:

- The user implements a model in Matlab, using Urd,
- Upon execution of the program, Urd interprets the user input and writes this to the files `Inits.txt`, `CellIn.txt`, `SynIn.txt`, and `ConIn.txt`, then

- Urd automatically starts Verdandi, which reads the input files, performs the simulations, and writes simulation results to `Vm.txt` and `Spikes.txt`,
- These files can be read again for post-processing in Urd, or even Skuld[1] for a more in-depth analysis of the results.

Now that the program flow is described, attention will be given to the specific features of Urd.

3 Model structure

A model implemented in Urd consists of two main components: *populations* and *connectivities*. Population:

- A population is an ensemble of neurons of the same type (from a model point of view),
- Every population has a unique name,
- Every neuron in a population has a position,
- Parameters can be set for the whole population, as well as for individual cells,
- Initial conditions can be set for the whole population, as well as for individual cells.

Connectivity:

- A connectivity is a set of connections from neurons in population A to population B, with synapses of type S.
- Connections between A and B can be given as a matrix.
- Connections between A and B can be determined using distance dependent probability function.
- Each connection can have a unique weight and delay.
- Depending on the type S of synapses, dynamics have to be specified.

4 Urd syntax

Urd consists of two commands: `AddPopulation` and `AddConnectivity`, and each command requires a set of (possible multiple) *attributes* with corresponding *arguments*. For example:

```
command(Att1, Arg1, Att2, Arg2a, Arg2b, ..., Attn, Argn)
```

For each command, the required attributes and allowed arguments are given in the text and tables on the following pages.

Table 1: Allowed interpretations for *aNumber*

Code fragment	Notes	Description
`..., 'Att',` c `, ...`	c scalar	Attribute's value is set to c for all instances
`..., 'Att',` V `, ...`	V column vector	Attribute of object i is set to V_i
`..., 'Att', 'Uniform',` $[a, b]$ `, ...`	a, b scalar	Attribute's value is uniformly distributed $U([a, b])$
`..., 'Att', 'Normal',` $[\mu, \sigma]$ `, ...`	μ, σ scalar	Attribute's value is normal distributed $N(\mu, \sigma)$

4.1 *aNumber*

In some of the columns the argument *aNumber* for attribute `'Att'` is given, which allows the definitions given in Table 1.

4.2 `AddPopulation`

The main attributes for population command are given by:

Table 2: Attributes for `AddPopulation`

Attribute	Description	Argument(s)
`'Name'`	Name of the population	String
`'nCell'`	Number of neurons in population	Positive integer
`'Position'`	Position of each cell	`nCell` $\times$ d-array, $1 \leq d \leq 3$
`'Type'`	Type of neurons in population	`'LeakyIntegrate'`, or `'Izhikevich'`, or `'Poisson'`

Depending on the neuron type chosen, different attributes are required for the `AddPopulation` to execute properly, see Tables 3, 4, and 5. Precise definitions of these single compartment models and relevant parameter values are easily found in literature; they will not be treated here.

Table 3: Additional attributes for `AddPopulation(..., 'Type', 'LeakyIntegrate', ...)`

Attribute	Description	Argument(s)
`'C'`	Membrane capacitance	*aNumber*
`'VRest'`	Resting membrane potential (mV)	*aNumber*
`'VThreshold'`	Membrane potential at which a spike is generated and neuron is set to `'VReset'`	*aNumber*
`'VReset'`	Value at which membrane potential is reset after spike (mV)	*aNumber*
`'V'`	Initial values for membrane potentials (mV)	*aNumber*

4.3 `AddConnectivity`

This command's attributes are shown in Table 6.

[1] Skuld is another Matlab toolbox that offers a wide variety of tools for post-processing and analysis of simulation results of neural networks.

Table 4: Additional attributes for `AddPopulation(..., 'Type', 'Izhikevich', ...)`

Attribute	Description	Argument(s)
`'a'`	Parameter a	*aNumber*
`'b'`	Parameter b	*aNumber*
`'c'`	Parameter c	*aNumber*
`'d'`	Parameter d	*aNumber*
`'I'`	Constant injected current	*aNumber*
`'V'`	Initial values for membrane potentials (mV)	*aNumber*
`'w'`	Initial values for recovery variable	*aNumber*

Table 5: Additional attributes for `AddPopulation(..., 'Type', 'Poisson', ...)`

Attribute	Description	Argument(s)
`'Lambda'`	Rate of Poisson process (1/s)	*aNumber*

Table 6: Attributes for `AddConnectivity`

Attribute	Description	Argument(s)
`'From'`	Population(s) that make connections	String or cell with strings
`'To'`	Population(s) that reveice connections	String or cell with strings
`'Connections'`	Precisely specifies the connections	See text
`'Delay'`	Specifies the delays used in the network	See Table text
`'Type'`	Type of synapse	`'Delta'` or `'Exponential'`

The options for the attribute `'Connections'` are numerous, so they are provided with more detail in the following list:

- `'Connections', 'Matrix',` M `, ...`, with M a $n_t \times n_f$-matrix for n_t and n_f the number of neurons in `'To'` and `'From'` populations respectively:
  If $M_{ij} \neq 0$ a connection is made from neuron j in source population to neuron i in target population. The weight of this connection is given by M_{ij}.
- `'Connections', 'Generate', 'Probabiliy',` P `, 'Weight',` W `, ...`, for P, W either scalar or function handles depending on distance:
  All pairs of neurons in source and target populations have probability P (optionally dependent on distance between neurons) to make a connection with weight W (optionally dependent on distance).

For the delays, the following options are available:

- `'Delay', 'Matrix',` T `, ...`, with M a $n_t \times n_f$-matrix for n_t and n_f the number of neurons in `'To'` and `'From'` populations respectively (only allowed is `'Connections'` also is defined using a matrix):
  For a connection from neuron j to neuron i, the delay is set to $T_{i,j}$ (in ms).
- `'Delay', 'Function',` F `, ...`, with F either scalar constant or function handle depending on distance:
  The delay of a connection is given by F (optionally dependent on distance).

If the exponential synapse is chosen, additional attributes have to defined; see Table 7.

Table 7: Additional attributes for `AddPopulation(..., 'Type', 'Izhikevich', ...)`

Attribute	Description	Argument(s)
`'Tau'`	Time constant of decay (ms)	*aNumber*
`'E'`	Reversal potential of postsynaptic current (mV)	*aNumber*
`'g'`	Maximal conductance of postsynaptic current	*aNumber*