Biological Neural Networks (BNNs) Toolbox for MATLAB: User Guide

Amir Reza Saffari Azar Alamdari

Version 1.0

http://www.ymer.org

August 2004

Contact Addresses

Email:	amir@ymer.org
Website:	http://www.ymer.org
Alt. Website:	http://ee.sut.ac.ir/faculty/saffari/main.htm
Community:	http://groups.yahoo.com/group/computational-neuronscience
Address:	No. A3, 2nd floor, Cheshm-Andaz, Moalem Str., Vali-Asr, Tabriz, Iran

Contents

Chapter 1

Introduction

1.1 What is BNN Toolbox?

Biological Neural Network (BNN) Toolbox is MATLAB-based software to simulate network of biological realistic neurons, as an abstract model of brain and Central Nervous System[1]. This software enables user to create and simulate various BNN models easily, using built-in library models, and just in a few lines of code. User can also create custom models and add them to the library, using library templates. A set of very descriptive examples are available to give a quick introduction to the toolbox and to reduce the coding time for beginners. In addition this toolbox only covers spiking models of neurons and biologically plausible network components. To simulate firing rate models (also known as ANN[2]), there exists very well designed packages, such as Neural Network Toolbox of MATLAB.

This toolbox uses powerful MATLAB programming language. MATLAB is a popular computation platform with highly specialized and powerful toolboxes for most scientific and engineering fields. To use all benefits of this toolbox, user must have an essential knowledge of programming in MATLAB or other similar popular languages, such as C/C++ or PASCAL. But a very small experience in defining variables and function usage is also suffices to use this package. User also can gain this by tracking example codes. Note that this constraint will be removed in next versions using a user friendly GUI system.

The main idea behind the first release is to present the essential concepts of a general biological nervous system simulator in MATLAB environment and gather feedback from users to improve next releases. So it is appreciated very much for any kind of comments, bug reports, and discussions. These feedbacks are believed to be the essential keys to keep this toolbox improving and evolving in the future.

This toolbox is created and developed by personal interests of the author in

[1] CNS

[2] Artificial Neural Networks

modeling brain and CNS and is provided to other users as an open source free software under GNU GPL[3]. Feel free to copy, modify, and distribute it according to the license. To obtain the latest version of GPL, please refer to http://www.gnu.org. Note that MATLAB itself is a commercial software provided by The MathWorks, Inc. http://www.mathworks.com.

1.2 Overview and Features

As the name of the toolbox implies, the main goal of this software is to provide users a set of integrated tools to create models of biological neural networks and simulate them easily, without the need of extensive coding. Users can create and simulate a huge network of spiking neurons in less than 10 lines of code (or even in one line, if they give all arguments to the main function) using predefined library functions. It is also possible to create and add new models to the library easily, using template library items provided for this reason.

Since programming in MATLAB is now very popular, users also would have the benefits of other toolboxes to extend their code and models easily. The followings are a list of features available in this release.

1.2.1 Current Features

- Users can create special architectures for their network connections using library tools. Current architectures in the library are:

  1. Fully connected
  2. Multilayer feedforward
  3. Multilayer feedforward with general feedback
  4. Multilayer feedforward with lateral connections
  5. Multilayer feedforward with local feedback

  Any other kind of user defined architecture is also possible. Each network connection consists of synaptic weight and transmission delay features. External inputs to the network can be defined to be either analog currents or time encoded spikes. In addition it is possible to restrict each neuron to inhibitory, excitatory, or hybrid neurons (its behavior can change in the simulation process).

- The spiking neuron models provided in the library are:

  1. Linear leaky integrate-and-fire model

[3] General Public License

2. Nonlinear quadratic integrate-and-fire model
3. Hodgkin-Huxley model
4. FitzHugh-Nagumo model
5. Morris-Lecar model
6. Canonical phase model
7. Izhikevich model

Users can define other special neuron models easily. It is also possible to define different functions for threshold and reset behavior of a neuron (if needed).

- Some predefined Post-Synaptic-Potential (PSP) behavior is available in the library:

  1. Two different versions of α-function
  2. Simple synaptic weight summation for spike inputs

  Again user defined synapses are possible.

- Users can define adaptation mechanisms for synaptic weights, transmission delays, and spike detection thresholds and any other parameter in the network.

- The simulation process of the network is based on a event driven system, meaning that the simulation stops for special computations such as calling reset or adaptation functions whenever an action potential is generated in the network. As a result the simulation speed depends highly on the rate of the spike generation rather than complexity of the models or the number of neurons in the network.

- For neuron models based on differential equations, simulation system uses MATLAB's powerful ODE solvers. Users have the chance to select various solvers provided for stiff and non-stiff problems. For continuous time equation models (such as SRM), a special function will be written to solve the model in next release.

- It is possible to provide static or dynamic external inputs to the network. Static inputs do not change over the simulation process of the model, but dynamic inputs may change according to the time or system state variables.

- During the simulation process, it is possible to call some user defined functions for specific tasks not included in the toolbox.

- Stop conditions for the simulation are very flexible. The default condition is the simulation time limitation, but one can define a special function for this reason.
- Some illustrative examples are included in the toolbox for users to explain how to write their code for simulating specific models. Users can change these examples according to their own purposes to obtain a fast code generation system.

Since this is the first release, it contains the basic concepts about how this toolbox would be in future releases. As mentioned earlier, the author would be very grateful for any kind of words from users to keep this toolbox better and alive in next versions. You can help the development process by providing comments, suggesting new models, reporting bugs, and anything you think about this toolbox. The following list is the features to be added to the toolbox in the next releases.

1.2.2 Future Outlook

- Next releases will have a very user friendly GUI[4] to improve the model creation and simulation processes. Because of time considerations this feature is not present in this release.
- There will be a Simulink blockset for simulating BNNs using MATLAB Simulink. This will highly increase the flexibilities of this toolbox with linking it to other Simulink libraries.
- There will be a variant of this toolbox for those users without MATLAB software. So they will be able to run this toolbox as a stand-alone software.
- Next versions will be much faster than this version in simulating models.
- The library will have more models for neurons, synapses, and architectures. If you have a model and wishing it to be included in next releases, please contact the author.
- The adaptation library will be added, containing various learning mechanisms for different parts of the system.
- The possibility to simulate continuous time equations, such as SRM, will be added.

[4] Graphical User Interface

- Modular architecture will be added to provide a tool for simulating modular and hierarchical network models.
- It is possible to add some specific tools for simulating compartmental models of neurons and networks. Since there are other highly specialized packages for this reason, such as NEURON or GENESIS, this function is not covered in this release, but might be included if users wish for such a feature.
- The programming style for essential definitions of this software will be changed to OOP[5] style using Classes and Objects programming of MATLAB.
- It would be provided to export the simulation results in different file formats, such as Excel.

Next release will be within 3 months from this one.

1.3 Installation Guide

1.3.1 Requirements

Here is the check list for the essential requirements to run this toolbox.

- MATLAB version 7.0 or 6.x. See Notes section below for more information.
- The latest version of the Biological Neural Networks Toolbox for MATLAB.

Notes:

Since MATLAB system is a script interpreter, you need to have MATLAB running on your computer to execute this toolbox (and of course any other MATLAB program). Because of this feature, most of the MATLAB programs are OS independent, if one does not use features provided for a specific OS. So there is no restriction about the system specifications including hardware and operating system except that it must have a MATLAB installed on it and also be capable of running MATLAB properly.

The toolbox is originally developed using MATLAB 7.0, Release 14 and is tested successfully with MATLAB 6.x, Release 13 under Microsoft Windows 2000 Professional.

You need also to download the latest version of this toolbox and related documentation from http://www.ymer.org, from the alternative website http://ee.sut.ac.ir/faculty/saffari/res-bnntoolbox.htm, or from the computational neuroscience community http://groups.yahoo.com/group/computational-neuroscience.

[5] Object Oriented Programming

1.3.2 How to install?

The toolbox files are compressed into a zip file format, so after downloading it you will need a decompressor program to unzip the files. There is also an extra decompressor MATLAB program on the website for those unable to unzip the toolbox.

Unpack the toolbox files into any directory you wish and then run the install.m file in the root directory of the toolbox. This program will add the toolbox directories into the MATLAB search path, enabling MATLAB to find toolbox functions. You can change the toolbox directory any time later, but be sure to run the installation file again or add the new paths manually.

Now you are ready to start working with the toolbox.

1.4 Support

The author would do his best to provide a good email supporting system for any kind of help regarding this toolbox. Please state your questions clearly and if possible with the related programs.

THIS SOFTWARE COMES WITH NO WARRANTY WHATSOEVER. THE AUTHOR IS NOT RESPONSIBLE FOR ANY DAMAGE CAUSED BY THE (MIS)USE OF THIS SOFTWARE!

Chapter 2

Basics of BNN

A Biological Neural Network or simply BNN is an artificial abstract model of different parts of the brain or nervous system, featuring essential properties of these systems using biologically realistic models. Neurons in the CNS communicate using short duration pulses, called action potentials or spikes, which the basic features of these signals, such as amplitude and temporal properties, are not different for a population of neurons in a certain part of the CNS. So the basic feature of a BNN is to use spiking neurons instead of traditional firing rate models (also known as sigmoidal neurons) used in ANNs. Because of this spiking behaviour, in most cases Spiking Neural Networks (SNNs) term is used instead of BNNs for these models.

The basic components of a BNN model is described in next section using a very simple and illustrative example explaining the essential features of the toolbox. It is recommended for beginners in computational neuroscience to read some reference books regarding modeling aspects of neuronal systems, since this manual is not going to repeat them here. It is supposed throughout this guide that the user has the eseential knowledge of spiking neurons

2.1 Components of a BNN

Let's start this section with a simple example. Suppose we want to simulate a network of three fully connected integrat-and-fire neurons using this toolbox. The source code for this example is available in the examples directory of the toolbox, under example1.m. Most of the functions in this example use default values for their input arguments. This will simplify the introduction process. Note that the full description of the commands and their input arguments will be given in next chapters. To open this example in a m-file editor, execute edit example1 in command window or run Open option from File menu and select example1.m.

The first three lines of the example are just clean up tasks, it clears command window, current figure, and variables in the workspace. If you don't want to lose

these information, comment these lines with inserting % at the beginning of the commands.

2.1.1 Architecture

The first step in creating a BNN model is to define the architecture of the system. Architecture of a BNN refers to the basic structure and topology of connections in a network of spiking neurons. It also defines how the external inputs are connected to the neurons.

Each connection has two basic parameters: synaptic weight and transmission delay. Synaptic weight defines the efficiency or the strength of the input to a specific neuron from other neurons or external sources. The zero value for a synaptic weight is equal to disconnecting the source from a neuron. Because of biochemical nature of signal transmission in neurons, the process of communication has a temporal delay, defined by transmission delay of each connection.

Neurons in CNS are divided into two different types: Inhibitory and Excitatory. The architecture object also stores the type of each neuron in the network. In addition, the architecture object enables user to specify the properties of external inputs to be as spiking and analog stimulation currents.

The following command defines the network architecture to be a fully connected structure of 3 excitatory neurons with one external analog input current:

```
NetArch = newarchfc(3 , 1 , -1);
```

2.1.2 Neuron Model

The next step is to define the model of neurons in the network. The basic part of this object is the Differential Equations (DEs) of the neuron behavior. Because all neurons in this toolbox are spiking models, it is necessary to define the spike detection (also known as threshold or event) and reset functions. The spike detection function determines when a specific neuron fires a spike according to its state variables. The reset function is responsible for the behavior of the neuron right after the transmission of the spike.

The next line of the example describes the neurons of the BNN to be linear leaky integrate-and-fire:

```
NeuronModel = newneuronif;
```

2.1.3 Synapse Model

The process of interaction of neurons in the network with each other and also the effect of external spiking inputs on neurons is described by the synapse model.

Upon receiving a spike from a pre-synaptic source, a Post-Synaptic-Potential (PSP) is triggered on the destination neuron. The synapse model consists of functions to describe the PSP behavior.

The following command creates a synapse model using the default values:

```
SynapseModel = newsynapse;
```

2.1.4 Adaptation Model

Adaptation model is used to describe the plasticity and learning mechanisms available for a specific BNN. Synaptic weight, transmission delay, threshold value, and model parameters plasticity functions can be defined using adaptation mechanism.

The next line described the adaptation object with default values:

```
AdaptModel = newadapt;
```

2.1.5 Simulation Parameters

To simulate a BNN properly, we have to specify simulation parameters, such as start and stop time, initial conditions, DE solver type, user defined custom function, and stop function. The main simulator program starts the simulation by solving the DEs from the start time and specified initial conditions. It stops the solver when a spike is generated, and calls adaptation and user defined custom functions. Then the simulation starts again. This process repeats until some stopping criteria is satisfied, such as stop time or user defined stop command.

In next command we specify stop time to be $20ms$ and leave other arguments to get default values:

```
SimParam = newsim(20);
```

2.1.6 Input/Output

The last component of the BNN model is the Input/Output object. This module specifies several features of input and output system of BNN, such as which values external inputs have at any time, saving options, and also some storage fields for the simulation results and output spike times.

For our simulation example, we are going to have a $1\mu A$ external input current:

```
IOMode = newiomode('none' , [] , 1);
```

2.1.7 BNN Model

Now we have all the required objects to construct a BNN model. The next line of code gathers the previous modules into a single BNN object:

```
network = newbnn(NetArch , NeuronModel , SynapseModel , AdaptModel , SimParam , IOMode);
```

2.1.8 Simulating BNN

The final step is to simulate the BNN model, using following command:

```
network = simbnn(network);
```

Now we can run this program with either executing the example1 in command window or selecting Run from Debug menu of the m-file editor. Note the messages appearing in the command window with each successful execution of program commands. With each spike generation the current time of simulation is also shown. After the completion of simulation, the membrane potentials are displayed in a figure.

2.2 Summary

Summarizing the example presented above, to create and simulate a BNN model properly, some basic components must be specified first, and then combined together to build a BNN object. This object can be simulated and the results can be plotted using provided tools.

In next chapters each component will be described in details separately. In addition available library items will be presented. Defining custom models and adding them to the library will be explained using function templates.

Chapter 3

Architecture Model

This chapter contains the description of the Architecture component and its variants. The most important factor for an architecture object is the topology of connections. Throughout the rest of the toolbox, the following standard will be used:

For any connection from input source j to destination i, a_{ij} will represent the synaptic weight or transmission delay of that connection.

Using this standard, connection topology of the network can be represented using four matrixes: two for interneuron connection weights and delays, and two for inputs/neurons connection weights and delays. If we have N neurons and M external inputs in the network, then the size of the first two matrixes is $N \times N$ and the last two is $N \times M$. These matrixes will be used by the simulation programs later. Note that a zero value for the synaptic weight of a specific connection would result in a disconnection between source and destination. Note that all timings in this toolbox are in *msec.* scale.

3.1 NEWARCH Function

The main function for the architecture component generation is the NEWARCH. The following lists describe the input arguments, output object, and default values for this function. Note that the input arguments must be entered only in the specified order. This is a very important issue when you want to use the default value for an argument. Let's state this problem with an example: suppose there is function $y = f(a, b)$ with two input arguments. If you enter $y = f(2, 3)$, the function will take $a = 2$ and $b = 3$. If you enter $y = f(2)$, the function will take $a = 2$ and will insert the default value for b. If you enter nothing, $y = f$, then the function will assume both default values for the a and b.

3.1.1 Syntax

NetArch = newarch(neuron_num , input_num , input_type , neuron_type , input_weight , neuron_weight , input_delay , neuron_delay)

3.1.2 Input Arguments

Input Name	Description
neuron_num	Number of neurons in the network: N. Must be an integer greater than zero. Default is a random number between 1 and 11.
input_num	Number of inputs in the network: M. Must be an integer than zero. Default is a random number between 1 and 11.
input_type	A vector of size M, each element corresponds to one of inputs. There are two different types of inputs: spiking and analog current. For spiking insert $+1$ and for analog current insert -1. Default is spiking for all inputs.
neuron_type	A vector of size N, each element corresponds to one of neurons. There are three different types of neurons: excitatory, inhibitory, and hybrid (can be both excitatory and inhibitory at the same time). For excitatory insert $+1$, for inhibitory insert -1, and for hybrid insert 0. Default is randomly chosen excitatory/inhibitory neurons.
input_weight	A matrix of size $N \times M$, each element a_{ij} corresponds to synaptic weight of connection from jth input to ith neuron. Default is a random matrix drawn from a uniform distribution.
neuron_weight	A matrix of size $N \times N$, each element a_{ij} corresponds to synaptic weight of connection from jth neuron to ith neuron. Default is a random matrix drawn from a uniform distribution.
input_delay	A matrix of size $N \times M$, each element a_{ij} corresponds to transmission delay of connection from jth input to ith neuron. Note that the time scale is $msec$. Default is a zero for all connections.
neuron_delay	A matrix of size $N \times N$, each element a_{ij} corresponds to transmission delay of connection from jth neuron to ith neuron. Note that the

3.1.3 Output Object

The output object is a structure variable containing the fields listed below. To access a particular field use *ObjectName.FieldName* style. For example if the object name is NetArch, the NetArch.NeuronNum will refer to number of neurons in this object. All of other architecture functions use this structure as output argument.

Field Name	**Description**
NeuronNum	Number of neurons in the network.
InputNum	Number of inputs in the network.
InputType	Input types vector.
NeuronType	Neuron types vector.
InputWeight	Input connections weight matrix.
NeuronWeight	Interneuron connections weight matrix.
InputDelay	Input connections delay matrix.
NeuronDelay	Interneuron connections delay matrix.

3.1.4 Examples

1. A network architecture with following properties: 4 neurons: 1 excitatory, 2 inhibitory, 1 hybrid. 2 inputs: 1 spiking and 1 analog current. Weights of all connections are equal to 1. All delays are $0.5msec$.

   ```
   NetArch = newarch(4 , 2 , [1 ; -1] , [1 ; -1 ; -1 ; 0] , ones(4 , 2) , ones(4 , 4) , 0.5*ones(4 , 2) , 0.5*ones(4 , 4));
   ```

2. A network architecture with 10 neurons and 5 inputs. All other values are set to default.

   ```
   NetArch = newarch(10 , 5);
   ```

3.2 NEWARCHFC Function

This function creates a fully connected architecture, i.e. all neurons and inputs are connected together.

3.2.1 Syntax

NetArch = newarchfc(neuron_num , input_num , input_type , neuron_type , weight_type , delay_type)

3.2.2 Input Arguments

Input Name	Description
neuron_num	Number of neurons in the network: N. Must be an integer. Default is a random number between 1 and 11.
input_num	Number of inputs in the network: M. Must be an integer. Default is a random number between 1 and 11.
input_type	A vector of size M, each element corresponds to one of inputs. There are two different types of inputs: spiking and analog current. For spiking insert $+1$ and for analog current insert -1. Default is spiking for all inputs.
neuron_type	A vector of size N, each element corresponds to one of neurons. There are three different types of neurons: excitatory, inhibitory, and hybrid (can be both excitatory and inhibitory at the same time). For excitatory insert $+1$, for inhibitory insert -1, and for hybrid insert 0. Default is randomly chosen excitatory/inhibitory neurons.
weight_type	A string indicating how to initialize the weight matrixes: 'random' for a random uniform distribution, 'normal' for a random normal distribution, 'one' for all weights equal to 1, and 'zero' for all weights equal to zero. Default is 'random'.
weight_type	A string indicating how to initialize the delay matrixes: 'random' for a random uniform distribution, 'one' for all delays equal to 1, and 'zero' for all delays equal to zero. Default is 'zero'.

3.2.3 Output Object

The same as NEWARCH.

3.2.4 Examples

1. A fully connected network architecture with following properties: 4 neurons: 1 excitatory, 2 inhibitory, 1 hybrid. 2 inputs: 1 spiking and 1 analog current. Weights of all connections are drawn from random uniform distribution. All delays are equal to one

   NetArch = newarchfc(4 , 2 , [1 ; -1] , [1 ; -1 ; -1 ; 0] , 'random' , 'one');

3.3 NEWARCHFF Function

This function creates a feedforward multilayer architecture, i.e. neurons are divided into groups called layers. The layers have an order which outputs of layer number n is connected only to inputs of layer number $n + 1$. Inputs are only connected to layer number 1.

3.3.1 Syntax

NetArch = newarchff(neuron_num , input_num , input_type , neuron_type , weight_type , delay_type)

3.3.2 Input Arguments

Input Name	Description
neuron_num	Number of neurons in each layer: $[N_1, N_2, ..., N_k]$ for a network with k layer and N_i neurons in layer number i. Default is a single layer network with random number of neurons between 1 and 11.
input_num	Number of inputs in the network: M. Must be an integer. Default is a random number between 1 and 11.
input_type	A vector of size M, each element corresponds to one of inputs. There are two different types of inputs: spiking and analog current. For spiking insert $+1$ and for analog current insert -1. Default is spiking for all inputs.
neuron_type	A vector of size N (total number of neurons in layers), each element corresponds to one of neurons. There are three different types of neurons: excitatory, inhibitory, and hybrid (can be both excitatory and inhibitory at the same time). For excitatory insert $+1$, for inhibitory insert -1, and for hybrid insert 0. Default is randomly chosen excitatory/inhibitory neurons.
weight_type	A string indicating how to initialize the weight matrixes: 'random' for a random uniform distribution, 'normal' for a random normal distribution, 'one' for all weights equal to 1, and 'zero' for all weights equal to zero. Default is 'random'.
weight_type	A string indicating how to initialize the delay matrixes: 'random' for a random uniform distribution, 'one' for all delays equal to 1, and 'zero' for all delays equal to zero. Default is 'zero'.

3.3.3 Output Object

The same as NEWARCH.

3.3.4 Examples

1. A feedforward network architecture with following properties: 3 layer: with 1 excitatory and 2 inhibitory in layer 1, 4 excitatory in layer 2, and 2 hybrid in layer 3. 2 inputs: 1 spiking and 1 analog current. Weights of all connections are drawn from random uniform distribution. All delays are equal to one

   NetArch = newarchff([3 ; 4 ; 2] , 2 , [1 ; -1] , [1 ; -1 ; -1 ; 1 ; 1 ; 1 ; 1 ; 0 ; 0] , 'random' , 'one');

3.4 NEWARCHGF Function

This function creates a feedforward multilayer architecture with general feedback, i.e. the same as feedforward except that outputs of the last layer are connected to inputs of first layer.

3.4.1 Syntax

NetArch = newarchgf(neuron_num , input_num , input_type , neuron_type , weight_type , delay_type)

3.4.2 Input Arguments

The same as NEWARCHFF.

3.4.3 Output Object

The same as NEWARCH.

3.4.4 Examples

The same as NEWARCHFF.

3.5 NEWARCHLC Function

This function creates a feedforward multilayer architecture with lateral connection for each layer, i.e. the same as feedforward except that outputs of each layer are connected to inputs of the same layer.

3.5.1 Syntax

NetArch = newarchlc(neuron_num , input_num , input_type , neuron_type , weight_type , delay_type)

3.5.2 Input Arguments

The same as NEWARCHFF.

3.5.3 Output Object

The same as NEWARCH.

3.5.4 Examples

The same as NEWARCHFF.

3.6 NEWARCHLF Function

This function creates a feedforward multilayer architecture with local feedbacks for each layer, i.e. the same as feedforward except that outputs of the each layer are connected to inputs of the previous layer.

3.6.1 Syntax

NetArch = newarchlf(neuron_num , input_num , input_type , neuron_type , weight_type , delay_type)

3.6.2 Input Arguments

The same as NEWARCHFF.

3.6.3 Output Object

The same as NEWARCH.

3.6.4 Examples

The same as NEWARCHFF.

3.7 CHECKARCH Function

This function checks an architecture object for errors and returns a message and a flag indicating the condition of the object.

3.7.1 Syntax

[CheckMessage , CheckFlag] = checkarch(NetArch)

3.7.2 Input Arguments

An architecture object.

3.7.3 Outputs

Output Name	Description
CheckMessage	A string containing the error condition of the object. 'OK' for no error situation
CheckFlag	An integer for associated condition. 1 for no error situation.

Chapter 4

Neuron Model

This chapter contains the description of the Neuron component and its variants. Three essential parts of a neuron model are:

1. Differential Equations (DEs) of the state variables of the model.
2. Spike detection procedure.
3. Reset procedure.

Eq(4.1) gives the general and complete structure of a neuron model:

$$\frac{d\mathbf{x}}{dt} = F(t, \mathbf{x}) + I(t, \mathbf{x}) \tag{4.1}$$

which $\mathbf{x}$ is the state vector, $F(t, \mathbf{x})$ is the internal dynamics, and $I(t, \mathbf{x})$ is the total external or internal dendritic current. The last term will be described in next chapter under the general definition of synapse.

Please refer to chapter 9 for more information about adding custom neuron models to the library and also the structure of functions. This chapter continues with description of basic neuron component and library items available for this version.

4.1 NEWNEURON Function

This is the main function for custom neuron model component definition.

4.1.1 Syntax

NeuronModel = newneuron(neuron_fun , model_type , state_num , model_param , spike_det_fun , spike_det_fun_param , reset_fun , reset_fun_param)

4.1.2 Input Arguments

Input Name	Description
neuron_fun	Function name containing neuron's DEs. Must be a string. Default value is 'int_fire_lin' (linear integrate-and-fire neuron).
model_type	Model type. Always must be 'CDE' for this version. Default value is 'CDE'.
state_num	Number of state variables in the DEs. Default is 1.
model_param	Parameters for the model. Must be a vector. Enter 'def' for default values.
spike_det_fun	Function name for spike detection algorithm. Default value is 'threshold_fun'.
spike_det_fun_param	Parameters for the spike detection function. Enter 'def' for default values.
reset_fun	Function name for reset algorithm. Enter 'none' if there is no need for reset. Default value is 'reset_fun'.
reset_fun_param	Parameters for the reset function. Enter 'def' for default values.

4.1.3 Output Object

The output object is a structure variable containing the fields listed below. All of other neuron functions use this structure as output argument.

Field Name	Description
Model.NeuronFun	Function name containing neuron's DEs.
Model.ModelType	Model type.
Model.StateNum	Number of state variables in the DEs.
Model.ModelParam	Parameters for the model.
SpikeDet.SpikeDetFun	Function name for spike detection algorithm.
SpikeDet.SpikeDetFunParam	Parameters for the spike detection function.
Reset.ResetFun	Function name for reset algorithm.
Reset.ResetFunParam	Parameters for the reset function.

4.1.4 Examples

1. A Hodgkin-Huxley neuron with standard parameters and a spike detection function using a threshold of $10mV$:

   ```
   NeuronModel = newneuron('hodgkin_huxley' , 'CDE' , 4 , 'def' , 'hh_threshold_fun' , 10 , 'none' , 'def');
   ```

2. A leaky linear integrate-and-fire neuron with reset potential set to $1mV$. All other values are set to default.

   ```
   NeuronModel = newneuron('int_fire_lin' , 'CDE' , 1 , 'def' , 'threshold_fun' , 'def' , 'reset_fun' , 1);
   ```

4.2 NEWNEURONIF Function

This function creates an integrate-and-fire neuron object. There are two different type of this model in the library:

1. Leaky linear integrate-and-fire, see Eq(4.1).
2. Quadratic nonlinear integrate-and-fire, see Eq(4.2).

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \tag{4.2}$$

$$\tau_m \frac{du}{dt} = a_0(u - u_r)(u - u_c) + RI(t) \tag{4.3}$$

$$\tau_m = RC \tag{4.4}$$

Spike detection and after spike reset is described by ($\hat{t}$ is the most recent spike time):

$$\text{if} \quad u \geq \theta \quad \text{then} \quad \hat{t} = t, \quad u(t^+) = u_r \tag{4.5}$$

Default values for model and threshold function parameters are listed bellow. Note that you must preserve the order of this table whenever you try to load other parameters into the model. Also the default scale for each parameter is the same as what is displayed in this table.

$C = 1\mu F$	$R = 10k\Omega$	
$a_0 = 1$	$u_r = 0mV$	$u_c = 0.5mV$
$\theta = 1mV$		

4.2.1 Syntax

NeuronModel = newneuronif(if_model , model_param , spike_det_fun_param , reset_fun_param)

4.2.2 Input Arguments

Input Name	Description
if_model	Integrate-and-fire model name: can be 'linear' or 'quadratic'. Default value is 'linear'.
model_param	Parameters for the model. Enter 'def' for default values.
spike_det_fun	Function name for spike detection algorithm. Default value is 'threshold_fun'.
spike_det_fun_param	Parameters for the spike detection function. Enter 'def' for default values.

4.2.3 Output Object

The same as NEWNEURON.

4.2.4 Examples

1. A quadratic model with following parameters: $R = 20k\Omega, C = 4\mu F, a_0 = 2, u_r = 0.5mV, u_c = 1mV$

   NeuronModel = newneuronif('quadratic' , [4 , 20 , 2 , 0.5 , 1]);

4.3 NEWNEURONHH Function

This function creates a Hodgkin-Huxley (HH) neuron model with following equations:

$$C\frac{du}{dt} = I - [g_{Na}m^3h(u - E_{Na}) + g_K n^4(u - E_K) + g_L(u - E_L)] \quad (4.6)$$

$$\frac{dm}{dt} = \alpha_m(1 - m) - \beta_m m \quad (4.7)$$

$$\frac{dn}{dt} = \alpha_n(1 - n) - \beta_n n \quad (4.8)$$

$$\frac{dh}{dt} = \alpha_h(1 - h) - \beta_h h \quad (4.9)$$

Original HH model does not have any spike detection and reset function. But because the simulator program has to detect the spike generation event, we added a threshold function to this model. The default value for this threshold is $0mV$.

Default values for model parameters are listed bellow. Note that you must preserve the order of this table whenever you try to load other parameters into the model. Also the default scale for each parameter is the same as what is displayed in this table.

$g_{Na} = 120$	$g_K = 36$	$g_L = 0.3$
$E_{Na} = 50mV$	$E_K = -77mV$	$E_L = -54.4mV$
$C = 1\mu F$		

4.3.1 Syntax

NeuronModel = newneuronhh(model_param , spike_det_fun_param)

4.3.2 Input Arguments

Input Name	Description
model_param	Parameters for the model. Enter 'def' for default values.
spike_det_fun_param	Parameters for the spike detection function. Enter 'def' for default values.

4.3.3 Output Object

The same as NEWNEURON.

4.3.4 Examples

1. A HH model with following parameters: $g_{Na} = 110, g_K = 40, g_L = 0.5, E_{Na} = 60mV, E_K = -80mV, E_L = -60mV, C = 2\mu F$

   ```
   NeuronModel = newneuronhh([110 , 40 , 0.5 , 60 , -80 , -60 , 2]);
   ```

4.4 NEWNEURONFN Function

This function creates a FitzHugh-Nagumo (FN) neuron model with following equations:

$$\epsilon \frac{dv}{dt} = -v(v-\alpha)(v-1) - w + I \tag{4.10}$$

$$\frac{dw}{dt} = v - \gamma w \tag{4.11}$$

Like HH model, original FN model does not have any spike detection and reset function. But because the simulator program has to detect the spike generation event, we added a threshold function to this model. The default value for this threshold is 0.8.

Default values for model parameters are listed bellow. Note that you must preserve the order of this table whenever you try to load other parameters into the model.

$\alpha = 0.1$	$\epsilon = 0.01$	$\gamma = 0.5$

4.4.1 Syntax

```
NeuronModel = newneuronfn(model_param , spike_det_fun_param)
```

4.4.2 Input Arguments

Input Name	Description
model_param	Parameters for the model. Enter 'def' for default values.
spike_det_fun_param	Parameters for the spike detection function. Enter 'def' for default values.

4.4.3 Output Object

The same as NEWNEURON.

4.4.4 Examples

1. A FN model with following parameters: $\alpha = 0.05, \epsilon = 0.02, \gamma = 0.6$

```
NeuronModel = newneuronfn([0.05 , 0.02 , 0.6]);
```

4.5 NEWNEURONML Function

This function creates a Morris-Lecar (ML) neuron model with following equations:

$$\frac{dv}{dt} = -g_{Ca}m_\infty(v)(v - E_{Ca}) - g_K w(v - E_K) - g_L(v - E_L) + I \quad (4.12)$$

$$\frac{dw}{dt} = \varphi\frac{[w_\infty(v) - w]}{\tau_w(v)} \quad (4.13)$$

$$m_\infty(v) = 0.5(1 + tanh(\frac{v - v_1}{v_2})) \quad (4.14)$$

$$w_\infty(v) = 0.5(1 + tanh(\frac{v - v_3}{v_4})) \quad (4.15)$$

$$\tau_w(v) = \frac{1}{cosh(\frac{v - v_3}{2v_4})} \quad (4.16)$$

Like HH model, original ML model does not have any spike detection and reset function. But because the simulator program has to detect the spike generation event, we added a threshold function to this model. The default value for this threshold is $40mV$.

Default values for model parameters are listed bellow. Note that you must preserve the order of this table whenever you try to load other parameters into the model.

$g_{Ca} = 4.4$	$g_K = 8$	$g_L = 2$
$E_{Ca} = 120mV$	$E_K = -84mV$	$E_L = -60mV$
$v_1 = -1.2$	$v_2 = 18$	$v_3 = 2$
$v_4 = 30$	$\varphi = 0.04$	

4.5.1 Syntax

NeuronModel = newneuronml(model_param , spike_det_fun_param)

4.5.2 Input Arguments

Input Name	Description
model_param	Parameters for the model. Enter 'def' for default values.
spike_det_fun_param	Parameters for the spike detection function. Enter 'def' for default values.

4.5.3 Output Object

The same as NEWNEURON.

4.5.4 Examples

1. A ML model with default parameters and threshold value of $30mV$:

```
NeuronModel = newneuronml('def' , 30);
```

4.6 NEWNEURONCANON Function

This function creates a canonical phase neuron model with following equations:

$$\frac{d\varphi}{dt} = 1 - cos(\varphi) + (1 + cos(\varphi))I \tag{4.17}$$

There is no adjustable parameter for this model. Spike detection is described by:

$$\text{if} \quad \varphi \geq \pi \quad \text{then} \quad \hat{t} = t \tag{4.18}$$

4.6.1 Syntax

NeuronModel = newneuroncanon

4.6.2 Input Arguments

No input argument.

4.6.3 Output Object

The same as NEWNEURON.

4.6.4 Examples

1. A canonical phase model:

```
NeuronModel = newneuroncanon;
```

4.7 NEWNEURONIZ Function

This function creates an Izhikevich neuron model with following equations:

$$\frac{dv}{vt} = 0.04v^2 + 5v + 140 - u + I \tag{4.19}$$

$$\frac{du}{vt} = a(bv - u) \tag{4.20}$$

Spike detection and after spike reset is described by:

$$\text{if} \quad u \geq \theta \quad \text{then} \quad \hat{t} = t, \quad v(t^+) = c, u(t^+) = u(t^-) + d \tag{4.21}$$

Default values for model and reset function parameters are listed bellow. Note that you must preserve the order of this table whenever you try to load other parameters into the model.

$a = 0.02$	$b = 0.2$
$c = -65$	$d = 8$
$\theta = 30$	

4.7.1 Syntax

```
NeuronModel = newneuroniz(model_param , spike_det_fun_param , reset_fun_param)
```

4.7.2 Input Arguments

Input Name	Description
model_param	Parameters for the model. Enter 'def' for default values.
spike_det_fun_param	Parameters for the spike detection function. Enter 'def' for default values.
reset_fun_param	Parameters for the reset function. Enter 'def' for default values.

4.7.3 Output Object

The same as NEWNEURON.

4.7.4 Examples

1. An Izhikevich model with following properties: $a = 0.01, b = 0.3, c = -70, d = 6.5, \theta = 20$.

   ```
   NeuronModel = newneuroniz([0.01 , 0.3] , 20 , [-70 , 6.5]);
   ```

4.8 CHECKNEURON Function

This function checks a neuron object for errors and returns a message and a flag indicating the condition of the object.

4.8.1 Syntax

```
[CheckMessage , CheckFlag] = checkneuron(NeuronModel)
```

4.8.2 Input Arguments

A neuron object.

4.8.3 Outputs

Output Name	Description
CheckMessage	A string containing the error condition of the object. 'OK' for no error situation
CheckFlag	An integer for associated condition. 1 for no error situation.

Chapter 5

Synapse Model

This chapter contains the description of the Synapse component and its variants. Two essential parts of a synapse model are:

1. External PSPs.
2. Internal PSPs.

Eq(5.1) gives the general structure of a synapse model:

$$I(t, \mathbf{x}) = I_{ext}^{analog} + I_{ext}^{spike} + I_{int}^{spike} \tag{5.1}$$

which $I(t, \mathbf{x})$ is the total input current to a neuron, I_{ext}^{analog} is the external analog current injected to a neuron, I_{ext}^{spike} is the PSP due to external spiking inputs, and I_{int}^{spike} is the PSP because of spikes from other neurons in the network. Since the current injection is not related to synapses, the synapse object only has the properties of the effects of spiking inputs. There are three different PSP functions in the library:

1. α-function: $I^{spike} = \frac{1}{\tau_m} \sum_k \sum_j w_k (exp(-\frac{t-t_k^j-d_k}{\tau_m}))$, default: $\tau_m = 10msec.$
2. α-function2: $I^{spike} = \frac{1}{\tau_s-\tau_m} \sum_k \sum_j w_k (exp(-\frac{t-t_k^j-d_k}{\tau_s}) - exp(-\frac{t-t_k^j-d_k}{\tau_m}))$, default: $\tau_s = 10msec, \tau_m = 5msec.$
3. Delta function: $I^{spike} = \sum_k \sum_j w_k \delta(t - t_k^j - d_k)$, default: $ValidPeriod = 2msec.$

which I^{spike} is the spike related input current for a neuron, w_k and d_k are synaptic weight and transmission delay for kth connection, respectively, t_k^j is the jth input spike time from kth synapse, τ_s and τ_m are time constants, $\delta(t)$ is the Dirac delta function. Since realizing delta function is not possible in computations, this function is replaced with a square pulse function given bellow:

$$P(t) = \begin{cases} 1 & 0 \leq t \leq ValidPeriod \\ \\ 0 & otherwise \end{cases}$$

Please refer to chapter 9 for more information about adding custom synapse models to the library and also the structure of functions. This chapter continues with description of basic synapse component and library items available for this version.

5.1 NEWSYNAPSE Function

This is the main function for custom synapse model component definition.

5.1.1 Syntax

SynapseModel = newsynapse(ext_psp_fun , ext_psp_fun_param , int_psp_fun , int_psp_fun_param)

5.1.2 Input Arguments

Input Name	Description
ext_psp_fun	Function name for external PSP. Must be a string. Available library PSP functions are: 'alpha_fun_ext', 'alpha_fun2_ext', 'delta_fun_ext'. Default value is 'alpha_fun_ext' (external $\alpha-$function).
ext_psp_fun_param	Parameters for the external PSP function. Enter 'def' for default values.
int_psp_fun	Function name for internal PSP. Must be a string. Available library PSP functions are: 'alpha_fun_int', 'alpha_fun2_int', 'delta_fun_int'. Default value is 'alpha_fun_int' (internal $\alpha-$function).
int_psp_fun_param	Parameters for the internal PSP function. Enter 'def' for default values.

5.1.3 Output Object

The output object is a structure variable containing the fields listed below.

Field Name	Description
External.ExtPSPFun	Function name for external PSP.
External.ExtPSPFunParam	Parameters for external PSP function.
Internal.IntPSPFun	Function name for internal PSP.
Internal.IntPSPFunParam	Parameters for internal PSP function.

5.1.4 Examples

1. A synapse model with following properties: external PSP is delta function with $ValidPeriod = 1msec.$, internal PSP is α-fucntion2 with $\tau_s = 20msec, \tau_m = 10msec.$

```
SynapseModel = newsynapse('delta_fun_ext' , 1 , 'alpha_fun_int' , [20 , 10]);
```

5.2 CHECKSYNAPSE Function

This function checks a synapse object for errors and returns a message and a flag indicating the condition of the object.

5.2.1 Syntax

```
[CheckMessage , CheckFlag] = checksynapse(SynapseModel)
```

5.2.2 Input Arguments

A synapse object.

5.2.3 Outputs

Output Name	Description
CheckMessage	A string containing the error condition of the object. 'OK' for no error situation
CheckFlag	An integer for associated condition. 1 for no error situation.

Chapter 6

Adaptation Model

This chapter contains the description of the Adaptation component. Four essential parts of a adaptation model are:

1. Synaptic weight adaptation mechanism.
2. Transmission delay adaptation mechanism.
3. Threshold adaptation mechanism.
4. Model parameters adaptation mechanism.

Since this is the first release, there is no adaptation model in the library and adaptation items will be added in next release.

Please refer to chapter 9 for more information about adding custom adaptation models to the library and also the structure of functions. This chapter continues with description of basic synapse component and library items available for this version.

6.1 NEWADAPT Function

This is the main function for custom adaptation model component definition.

6.1.1 Syntax

AdaptModel = newadapt(adapt_fun_weight , adapt_fun_weight_param , adapt_fun_delay , adapt_fun_delay_param , adapt_fun_threshold , adapt_fun_threhsold_param , adapt_fun_model , adapt_fun_model_param)

6.1.2 Input Arguments

Input Name	Description
adapt_fun_weight	Function name for weight adaptation mechanism. Must be a string. Default value is 'none' (no adaptation).
adapt_fun_weight_param	Parameters for the weight adaptation function. Enter 'def' for default values.
adapt_fun_delay	Function name for delay adaptation mechanism. Must be a string. Default value is 'none' (no adaptation).
adapt_fun_delay_param	Parameters for the delay adaptation function. Enter 'def' for default values.
adapt_fun_threshold	Function name for threshold adaptation mechanism. Must be a string. Default value is 'none' (no adaptation).
adapt_fun_threshold_param	Parameters for the threshold adaptation function. Enter 'def' for default values.
adapt_fun_model	Function name for model adaptation mechanism. Must be a string. Default value is 'none' (no adaptation).
adapt_fun_model_param	Parameters for the model adaptation function. Enter 'def' for default values.

6.1.3 Output Object

The output object is a structure variable containing the fields listed below.

Field Name	Description
Weight.AdaptFun	Function name for weight adaptation mechanism.
Weight.AdaptFunParam	Parameters for weight adaptation function.
Delay.AdaptFun	Function name for delay adaptation mechanism.
Delay.AdaptFunParam	Parameters for delay adaptation function.
Threshold.AdaptFun	Function name for threshold adaptation mechanism.
Threshold.AdaptFunParam	Parameters for threshold adaptation function.
Model.AdaptFun	Function name for model adaptation mechanism.
Model.AdaptFunParam	Parameters for model adaptation function.

6.1.4 Examples

1. An adaptation model with default values.

```
AdaptModel = newadapt;
```

6.2 CHECKADAPT Function

This function checks an adaptation object for errors and returns a message and a flag indicating the condition of the object.

6.2.1 Syntax

```
[CheckMessage , CheckFlag] = checkadapt(AdaptModel)
```

6.2.2 Input Arguments

An adaptation object.

6.2.3 Outputs

Output Name	Description
CheckMessage	A string containing the error condition of the object. 'OK' for no error situation
CheckFlag	An integer for associated condition. 1 for no error situation.

Chapter 7

Simulation Model

This chapter contains the description of the Simulation component. The simulation object contains information needed for a solver to compute the solution of the BNN model.

7.1 NEWSIM Function

This is the main function for custom simulation model component definition.

7.1.1 Syntax

SimParam = newsim(stop_time , start_time , initial_cond , solver_type , solver_option , stop_fun , user_fun)

7.1.2 Input Arguments

Input Name	Description
stop_time	Simulation stop time. Must be greater than zero and start time. Default value is 0 (no simulation).
start_time	Simulation start time. Must be greater than zero and less than stop time. Default value is 0 (no start).
initial_cond	Initial condition of state variable. A vector with the same size of the total state variables of the network. Default value is 0.
solver_type	MATLAB solver type selection. Must be one of 'ode45', 'ode23', 'ode113', 'ode15s', 'ode23s', 'ode23t', or 'ode23tb'. Refer to MATLAB help for more information about the solvers. Default value is 'ode45'.
solver_option	MATLAB solver option. There is no option in this release. Must be [] for this version.
stop_fun	User defined stop function (used to define stopping criteria rather than stop time). Must be a string. Default value is 'none' (no stop function).
user_fun	User defined function (used to define functions not included in the toolbox). Must be a string. Default value is 'none' (no user function).

7.1.3 Output Object

The output object is a structure variable containing the fields listed below.

Field Name	Description
StopTime	Simulation stop time.
StartTime	Simulation start time.
InitialCond	Simulation initial conditions.
Solver	Solver function.
SolverOption	Solver options.
StopFun	Stop function.
UserFun	User function.

7.1.4 Examples

1. A simulation model with following properties: start time = 0, stop time = 20, initial condition = 1 (note that when you enter a single value for initial condition, the function will automatically generalize it for all other states).

   ```
   SimParam = newsim(20 , 0 , 1);
   ```

7.2 CHECKSIM Function

This function checks a simulation object for errors and returns a message and a flag indicating the condition of the object.

7.2.1 Syntax

```
[CheckMessage , CheckFlag] = checksim(SimParam)
```

7.2.2 Input Arguments

A simulation object.

7.2.3 Outputs

Output Name	Description
CheckMessage	A string containing the error condition of the object. 'OK' for no error situation
CheckFlag	An integer for associated condition. 1 for no error situation.

Chapter 8

Input/Output Model

This chapter contains the description of the Input/Output component. This modules defines how the model will interact with inputs and outputs.

8.1 NEWIOMODE Function

This is the main function for custom input/output model component definition.

8.1.1 Syntax

IOMode = newiomode(input_fun_name , input_spike_time , analog_input , auto_save , save_file_name , file_format)

8.1.2 Input Arguments

Input Name	Description
input_fun_name	Input function name. This function must provide external analog current and spike times of input channels. Must be a string. Default value is 'none'.
input_spike_time	Spike times of inputs. This value will be used if input function name was 'none'. Must be a cell array of the same size of the number of spiking inputs. Each element of this cell array is a vector containing spike times of that input. Note that if there was an input function name, the spike time outputs of this function will be saved in this field. Default value is 0.
analog_input	Analog current input values. This value will be used if input function name was 'none'. Must be an array of the same size of the number of analog inputs. Each element of this array corresponds to the analog current value of that input. Note that if there was an input function name, the analog current outputs of this function will be saved in this field. Default value is 0.
auto_save	Auto save option. If you want to save the results automatically when simulation completed, set this variable value to 1, otherwise 0. You can specify the file name and file format using next input arguments. Default value is 0.
save_file_name	Save file name for auto save option. Must be a string. This value can also have the relative or absolute address. Default value is 'bnnet'.
file_format	Save file format for auto save option. Must be a string. Only 'MAT' is allowed in this version.

8.1.3 Output Object

The output object is a structure variable containing the fields listed below.

Field Name	Description
Input.InputFunName	Input function name.
Input.InputSpikeTime	Spike times of inputs.
Input.AnalogInput	Analog current input values.
Output.SpikeTimes	Neurons spike times. A cell array of the same size of the number of neurons.
Output.States	States values of network. Each column of this matrix corresponds to one of the states.
Output.Time	Time vector. A vector containing the time values of computations.
Output.AutoSave	Auto save option.
Output.SaveFileName	Auto save file name.
Output.FileFormat	Auto save file format.

8.1.4 Examples

1. An input/output model with following properties: 2 spiking input with spike times = $\{[2, 4, 8]; [3, 10]\} msec$, 1 analog input = $2\mu A$, auto save on, and save file to the 'results' directory with 'network' file name.

```
IOMode = newiomode('none' , {[2 4 8] ; [3 10]} , 2 , 1 , 'results/network');
```

8.2 CHECKIOMODE Function

This function checks an input/output object for errors and returns a message and a flag indicating the condition of the object.

8.2.1 Syntax

[CheckMessage , CheckFlag] = checkiomode(IOMode)

8.2.2 Input Arguments

An input/output object.

8.2.3 Outputs

Output Name	Description
CheckMessage	A string containing the error condition of the object. 'OK' for no error situation
CheckFlag	An integer for associated condition. 1 for no error situation.

Chapter 9

BNN Model

This chapter contains the description of the BNN model. Every BNN model must has the following components:

1. An Architecture model object.
2. A Neuron model object.
3. A Synapse model object.
4. An Adaptation model object.
5. A Simulation model object.
6. An Input/Output model object.

9.1 NEWBNN Function

This is the main function for custom BNN model definition.

9.1.1 Syntax

net = newbnn(NetArch , NeuronModel , SynapseModel , AdaptModel , SimParam , IOMode)

9.1.2 Input Arguments

Input Name	Description
NetArch	Architecture model object. Default is a default architecture object.
NeuronModel	Neuron model object. Default is a default neuron object.
SynapseModel	Synapse model object. Default is a default synapse object.
AdaptModel	Adaptation model object. Default is a default adaptation object.
SimParam	Simulation model object. Default is a default simulation object.
IOMode	Input/Output model object. Default is a default input/output object.

9.1.3 Output Object

The output object is a structure variable containing the fields listed below.

Field Name	Description
Architecture	Architecture model.
Neurons	Neuron model.
Synapses	Synapse model.
Adaptation	Adaptation model.
Simulation	Simulation model.
InputOutput	Input/Output model.

9.1.4 Examples

1. A BNN model with default objects.

```
net = newbnn;
```

9.2 CHECKBNN Function

This function checks a BNN model for errors and returns a message and a flag indicating the condition of the model. It uses previous object check functions.

9.2.1 Syntax

```
[CheckMessage , CheckFlag] = checkbnn(net)
```

9.2.2 Input Arguments

A BNN model.

9.2.3 Outputs

Output Name	Description
CheckMessage	A string containing the error condition of the object. 'OK' for no error situation
CheckFlag	An integer for associated condition. 1 for no error situation.

Chapter 10

Simulate a BNN

This chapter explains how to simulate a BNN model build with previous functions.

10.1 SIMBNN Function

This is the main function to simulate a BNN model.

10.1.1 Syntax

net = simbnn(net)

10.1.2 Input Arguments

Input Name	Description
net	A BNN model.

10.1.3 Output Object

The output object is the same as the input BNN model. The output fields would have the results of the simulation.

10.1.4 Examples

1. Simulate a BNN model.

   ```
   net = simbnn(net);
   ```

Chapter 11

Tools

This chapter contains some additional functions used for accessing and plotting the results easily.

11.1 GETSTATES Function

This function returns solved state values of a particular neuron.

11.1.1 Syntax

states = getstates(net , neuron_no)

11.1.2 Input Arguments

Input Name	Description
net	A BNN model (after computations).
neuron_no	Neuron number in the network. If you want to get state values for all neurons, don't insert this argument

11.1.3 Output Object

State values in a vector for a single neuron or in a matrix for all neurons with each column for a neuron.

11.1.4 Examples

1. Get state values for 2nd neuron.

   ```
   states = getstates(net , 2);
   ```

2. Get state values for all neurons.

   ```
   states = getstates(net);
   ```

11.2 GETSPIKES Function

This function returns spike times of a particular neuron.

11.2.1 Syntax

```
spike_time = getspikes(net , neuron_no)
```

11.2.2 Input Arguments

Input Name	Description
net	A BNN model (after computations).
neuron_no	Neuron number in the network. If you want to get spike time for all neurons, don't insert this argument

11.2.3 Output Object

Spike times in a vector for a single neuron or in a cell array for all neurons with each elemnt for a neuron.

11.2.4 Examples

1. Get spike times for 2nd neuron.

   ```
   spike_time = getspikes(net , 2);
   ```

2. Get spike times for all neurons.

   ```
   spike_time = getspikes(net);
   ```

11.3 GETTIME Function

This function returns computation time vector.

11.3.1 Syntax

time = gettime(net)

11.3.2 Input Arguments

Input Name	Description
net	A BNN model (after computations).

11.3.3 Output Object

Time vector of computation.

11.3.4 Examples

1. Get time values.

   ```
   time = gettime(net);
   ```

11.4 PLOTSTATES Function

This function plots solved state values of a particular neuron.

11.4.1 Syntax

plotstates(net , neuron_no)

11.4.2 Input Arguments

Input Name	Description
net	A BNN model (after computations).
neuron_no	Neuron number in the network. If you want to plot state values for all neurons, don't insert this argument

11.4.3 Output Object

A figure.

11.4.4 Examples

1. Plot state values for 2nd neuron.

   ```
   plotstates(net , 2);
   ```

2. Plot state values for all neurons.

   ```
   plotstates(net);
   ```

11.5 PLOTSPIKES Function

This function plots spike times of a particular neuron.

11.5.1 Syntax

```
plotspikes(net , neuron_no)
```

11.5.2 Input Arguments

Input Name	Description
net	A BNN model (after computations).
neuron_no	Neuron number in the network. If you want to plot spike times for all neurons, don't insert this argument

11.5.3 Output Object

A figure.

11.5.4 Examples

1. Plot spike times for 2nd neuron.

   ```
   plotspikes(net , 2);
   ```

2. Plot spike times for all neurons.

```
plotspikes(net);
```

11.6 PLOTSPIKES2D Function

This function plots spike times of all neurons with a dot. Vertical axis will be the neuron number and horizental axis will be time.

11.6.1 Syntax

```
plotspikes2d(net)
```

11.6.2 Input Arguments

Input Name	Description
net	A BNN model (after computations).

11.6.3 Output Object

A figure.

11.6.4 Examples

1. Plot spike times in 2D.

```
plotspikes2d(net);
```

Chapter 12

How to Add to Library

This chapter describes how to add models to the library using function templates. To understand this issue, it's necessary to illustrate the details of the simulator program and other associated functions first.

12.1 Details of Simulator

SIMBNN is the main simulator program. The process of simulation in this function has the following steps:

1. Checks the BNN model for errors using CHECKBNN function.
2. Initializes some internal variables. Sets the simulation current time to start time.
3. Creates event (spike detection) function for the MATLAB ODE solver.
4. Generalizes initial conditions. If the size of initial condition vector does not match with the total number of states in the BNN model, checks to see if its size is the same as the number of states for a single neuron. If the check result was true, it would repeat this initial condition vector for all other neurons. Otherwise, if the original initial condition was a single value, the function would make a vector for initial condition filled with this value for all states in the network.
5. Starts the computation loop, by calling the MATLAB ODE solver with following arguments: DEs function of the neuron model, current time, stop time, initial condition, and event function. The solver starts to integrate the DEs and stop whenever the computation time reaches to stop time or an event is detected using spike detection function. At this time, the solver stops the integration and returns states matrix, spike times, index of spiker neuron, and computation time vector.

6. The program stores new results in the proper output fields of BNN model and sets updates current time with the final value of the time vector.

7. It then calls the reset function for post-spike generation processes with state matrix and index of spiker neuron. This function must return new initial conditions for next steps of integration.

8. The program calls adaptation functions to perform learning mechanisms over adjustable parameters. The adaptation functions receive the BNN model as input and must return the modified BNN model.

9. If there was any user defined or stop function, it calls them with BNN model as input. User defined function must return a BNN model. Stop function must return the new stop flag condition: 0 for continue and 1 for stop.

10. The program checks current time. If current time was not greater than stop time and also stop flag was not equal to 1, then continues the simulation by going to step 5, otherwise it goes to next step.

11. If the auto save option was selected, it saves the BNN model, and then returns back the BNN model to the original program.

12.2 How to Add Neuron Model

To add a neuron model to the library, you can use the neuron_model_template.m program from the template directory. To have a backup from the template, it is recommended to copy this function with your model name and then modify its contents according to the instructions bellow. Each neuron model function has the DEs (or more exactly the ODEs) of that particular model. This function takes current time (t) and state vector (y) as input arguments and returns the derivative of state variables ($\frac{dy}{dt}$). The general structure of a neuron model is given by:

$$\frac{d\mathbf{y}}{dt} = F(t, \mathbf{y}) + I(t, \mathbf{y}) \tag{12.1}$$

which $\mathbf{y}$ is the state vector, $F(t, \mathbf{y})$ is the internal dynamics, and $I(t, \mathbf{y})$ is the total external or internal dendritic current.

The template function is the same as the leaky integrate-and-fire model discussed earlier, except that it has help sections about how to modify the template with your code. Follow these steps to create your model:

1. First of all take a copy of this file, rename it to 'my_model_name' (for example 'int_fire2'). Also change the name of function at the first line from 'neuron_model_template' to 'my_model_name'.

2. Put default parameters for your model at the specified section.

3. If you want to get other parameters rather than defaults from outside, change these codes according to your model. Note that you must preserve the parameters order here and outside to get the correct results. Another point is that you can specify different parameters for each neuron by setting model parameter in neuron object to a matrix containing parameters for each neuron in a row (i.e. each column corresponds to one of parameters and each row has the parameters for that neuron).

4. Put the DEs equations in the specified section. Use dotted arithmetic operations if you have different parameters for different neurons.

5. Save your file.

There is a line of code in this template for dendritic currents calculations:

```
I = external_input(t , y) + internal_input(t , y);
```

These functions will be explained in next sections.

12.3 How to Add Synapse Model

To add a synapse model to the library, you can use the synapse_model_template.m program from the template directory. To have a backup from the template, it is recommended to copy this function with your model name and then modify its contents according to the instructions bellow.

To gain more insight into the synapse models, let's explain the procedure of PSPs calculations. As stated before, the total dendritic input current to a neuron is a summation of external and internal inputs. Each neuron model function must have some functions (or a section in the same function) to calculate this value for all neurons in the network. There are two predefined functions associated with this problem in the toolbox: external_input and internal_input. These two functions has the same structure, except that an additional section is available in external_input for analog injection currents. The procedure of calculations in these files is:

1. Initialization of variables.

2. Separates the spiking inputs from analog inputs (external_input function only).

3. Checks for input file name. If there was such a function, then it calls this function with t and y as input arguments and gets analog input values

together with input spike times in the same structure as stated in InputOutput chapter. If there was not any input function, then it reads the input fields of BNN model.

4. If the size of the spike times cell array was equal to 1 and it does not match with the total number of spiking inputs, the function automatically generalizes this spike time to all other spiking inputs (external_input function only).

5. If the size of the analog input array was equal to 1 and it does not match with the total number of analog inputs, the function automatically generalizes this analog value to all other analog inputs (external_input function only).

6. The program then checks for PSP function name. If there was a PSP function, then it calls the PSP function with t, y, spike times, and transmission delay values as input arguments. The PSP function must return a matrix with a proper size. Each a_{ij} element of this matrix corresponds to the dendritic current value from jth source to ith neuron at that time.

7. The final step is the computation of weighted sum of total dendritic currents for all neurons.

As seen from above the main role of a PSP function is to get the spike times and to return the current value in an appropriate format. You can use any other functions rather than external_input and internal_input, if the previous steps are not suitable for you model.

To write a new PSP function you can use the template function. This function is the same as the α-function PSP model for internal spikes discussed earlier, except that it has help sections about how to modify the template with your code. Follow these steps to create your model:

1. First of all take a copy of this file, rename it to 'my_model_name' (for example 'int_fire2'). Also change the name of function at the first line from 'synapse_model_template' to 'my_model_name'.

2. Put default parameters for your model at the specified section.

3. If you want to get other parameters rather than defaults from outside, change these codes according to your model. Note that you must preserve the parameters order here and outside to get the correct results.

4. Put your PSP function in the specified section.

5. Save your file.

12.4 How to Add Adaptation Model

There is no template file for adaptation model, since there is no adaptation mechanism available in this release. But as a general rule writing an adaptation function is not a complicated task. This function must get the BNN model as its input. Then it should modify the parameters, such as weights, according to the learning algorithm. And at last stores them in the appropriate fields in the BNN model and returns it as output argument. Next release will have adaptation functions and templates.

Chapter 13

References

This chapter contains some useful references for models used in this toolbox.

1. Gerstner, W., Kistler, W. M. (2002) Spiking Neuron Model: Single Neuron, Populations, and Plasticity Cambridge University Press.

2. Maass, W., Bishop, C. M. (1998) Pulsed Neural Networks MIT Press.

3. Northrop, R. B. (2000) Introduction to Dynamic Modeling of Neuro-Sensory Systems CRC Press.