# NeuroField Reference

Peter Drysdale Felix Fung Romesh Abeysuriya

August 9, 2012

**Abstract**

`NeuroField` is a computer program (accompanied with helper scripts) that solves the neural field model of Robinson et al. This document is a reference of `NeuroField` for both users and developers.

# Contents

# 1 Users guide

`NeuroField` , written by Peter Drysdale with contributions from James Roberts, Felix Fung and Romesh Abeysuriya, is a `C++` program (accompanied with helper scripts) that solves the neural field model of Robinson et al.:

$$D_{ab}V_{ab}(\mathbf{r},t) = \nu_{ab}\phi_{ab}(\mathbf{r},t),$$
$$Q_a(\mathbf{r},t) = S_a\big[\sum_b V_{ab}(\mathbf{r},t)\big],$$
$$\mathcal{D}_{ab}\phi_{ab}(\mathbf{r},t) = Q_b(\mathbf{r},t-\tau_{ab}).$$

`NeuroField` generalizes the neural field theory by allowing users to:

1. Specify an arbitrary number of populations and connections between populations;
2. Specify the parameters for any objects, including populations, dendritic responses, firing responses, propagators, synapses, and stimulus pattern;
3. Choose alternative wave propagation types, i.e. choose different forms of $\mathcal{D}_{ab}$;
4. Uses plastic synapses, i.e. $\nu_{ab} = \nu_{ab}(\mathbf{r},t)$.
5. Use different firing responses, i.e. change $S_a$.

This users guide covers the obtaining and setting up (Sec. 1.1), configuring (Sec. 1.4) and launching of `NeuroField` (Sec. 1.3).

Within this documentation, specific terminology as appeared in the computer is in `typwriter font` . Commands are denoted as

```
Command to put in computer
```

## 1.1 Obtaining and setting up NeuroField

The code for `NeuroField` is managed by a version control system called `subversion` , which provides a single place to obtain the latest copy of the code, as well as storing the entire history of the program. To access the repository, contact Romesh Abeysuriya (`r.abeysuriya@physics.usyd.edu.au`) or Sue Yang (`xue.yang@sydney.edu.au`).

To set up the latest version of `NeuroField` in the current directory within the School of Physics, execute

```
svn co
   http://silliac.physics.usyd.edu.au:18080/svn/neurofield/trunk
   neurofield --username=<your SVN username>
```

You can also use these steps to obtain a copy of `NeuroField` from a computer which is not connected to the School of Physics network (eg. personal laptops at home). You will require a version of `SVN` higher than 1.6. However, you will need to have your IP address/network domain registered for remote access, please contact Sebastian Juraszek to request this (ideally by logging a helpdesk request at `http://physics.usyd.edu.au/itsupport` - only available within the School of Physics).

## 1.2 Directory layout

The canonical directory is `neurofield/trunk` . Within this directory, the user can find:

| | |
|---|---|
| `*.h, *.cpp` | C++ source code. |
| `Configs/` | Stores configuration files for `NeuroField` . |
| `Documentation/` | `Documentation/doc.tex` is the LaTeX file for generating this document. Running `make doc` produces this document in `pdf` format, and `make clean` removes all the files LaTeX produced (excluding `doc.pdf` and `doc.tex` ). |
| `Helper_scripts/` | Stores helper scripts, including plotting routines and other post-processing of data procedures. |
| `Launch` | A launcher script that handles compilation and launching of `NeuroField` , capable of automating parameter sweeps and submitting jobs in `yossarian` . |
| `Output/` | When using the launcher script to sweep over parameters, the launcher script produces this directory, which stores all output files `neurofield.*` in independent subdirectories. |
| `Release/` | All compiled files, including the object files and the `NeuroField` executable is stored here. This directory will be deleted by `make clean` , so user data should not be stored here. |
| `Test/` | Directory for unit testing and is irrelevant for users. |
| `neurofield.*` | Output files generated by `NeuroField` . |

## 1.3 Launching NeuroField

The `Launch` script is used to compile and launch the `NeuroField` program. Users are encouraged to use this script rather than calling the `NeuroField` executable directly. To launch `NeuroField` , do:

1. Edit `Makefile` : Identify the platform to run NeuroField, and comment/uncomment the appropriate `COMP` directions. Generally, this step is not needed, but users are encouraged to check.
2. To execute with only one set of parameters, edit your configuration file in `./Configs` , then run

```
./Launch Configs/config_file
```

   For example,

```
./Launch Configs/cortex.conf
```

   The output will be stored in the current directory.[1]
3. To ease parameter exploration, the launcher script is capable of doing parameter sweeps. The launcher script supports sweeping over an arbitrary number of objects, parameters and runs. For example, if the user wants to launch `NeuroField` 3 times, with parameters varying according to the following table,

| Object(s) | Parameter | 1st run | 2nd run | 3rd run |
|---|---|---|---|---|
| `Propag 1` , `Propag 2` | `gamma` | 10 | 20 | 30 |
| `Couple 1` | `nu` | 1 | 2 | 3 |

   simply list the above table entries into the argument list:

```
./Launch Configs/cortex.conf 'Propag 1' 'Propag 2' gamma 10 20
   30 'Couple 1' nu 1 2 3
```

   In terms of syntactic restriction, each row in the table can have an arbitary number of objects, but only one parameter; every row must have the same number of runs.[2]

[1] Tips for `vi` users: you can launch `NeuroField` in `vi` with `:!./Launch % [optional params]`

[2] Tips: the UNIX `seq` program allows shortening parameter value listing from `10 20 30 40 50 60` into `‘seq 10 10 60‘` , e.g. `Launch Configs/cortex.conf 'Propag 1' gamma ‘seq 10 10 60‘` .

4. Switches are accepted for choosing options. To turn on the switches, put them in the argument list to the launcher script. The following switches are accepted:

   `--restart` run `NeuroField` in restart mode.

   `-d` specify an alternative dump file name.

   `-i` specify an configuration file name.

   `-o` specify an output dump file name.

   `-s` specify silent mode, so that no dump files are generated. This switch supercedes the `-d` switch.

   `-v` specify verbose mode, so that `NeuroField` outputs every timestep.

   `-h` prints out a list of these switches.

5. In case the script is executed on `yossarian` , the launcher script tries to find a file called `pbs` . If no such file is found, the user would be prompted for a job name, expected computational time and email to generate `pbs` . This file is then submitted to the PBS system.

6. To clean up the directory, delete or store your `neurofield.*` output files and the `Output/` directory and subdirectories. Then run

   ```
   make clean
   ```

   to delete `./Release/` and the files LaTeXproduced in `Documentation/` .

7. This documentation can be generated by running

   ```
   make doc
   ```

   which produces `./Documentation.pdf` . `make clean` deletes the files (excluding `./Documentation.pdf` ) created by LaTeX.

## 1.4 Writing a configuration file

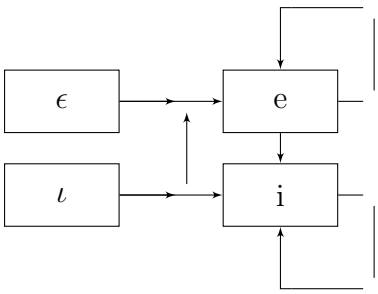
`NeuroField` allows an arbitrary number of populations and connections between them, with all objects taking arbitrary parameter values. These are all configured via a configuration file. This section documents the specifications of configuration files, where we use `Configs/cortex.conf` as an illustrative example.

To write a configuration file, a user can follow these steps:

1. Determine your population model by drawing a schematic diagram, thereby constructing a connection matrix. An example is shown in Fig. 1.
2. Look up existing configuration files in `Configs/` . By checking the comment located at the top of a configuration file, and also the connection matrix, a user should find the most suitable existing file to construct his own. This is less tedious (and less error prone) than writing a new one from scratch.
3. Specify the global parameters and connectivity matrix (Sec. 1.4.1).
4. Specify all populations (Sec. 1.4.2).
5. Specify all propagators (Sec. 1.4.3).
6. Specify all couples (Sec. 1.4.4).
7. Specify all output requests (Sec. 1.4.5).

In general, the entries in configuration file follow the following rules:

1. Each entry generally follows a `Pamameter: value` pattern.
2. Most parameters are essential. Failure to provide these parameters would result in `NeuroField` terminating with an error message. A minority of the parameters are optional.




| From: | $\epsilon$ | $\iota$ | e | i |
|---|---|---|---|---|
| To $\epsilon$: | 0 | 0 | 0 | 0 |
| To $\iota$: | 0 | 0 | 0 | 0 |
| To e: | 1 | 0 | 2 | 3 |
| To i: | 0 | 4 | 5 | 6 |

Figure 1: Left: schematic diagram of a purely cortical population model comprising excitatory and inhibitory populations, as well as two stimulus populations; each arrow indicates a connection between populations, so that each stimulus connects to a cortical population, and each cortical population connects to all cortical populations. Right: connection matrix indicating the connections between populations; zero indicates no connection, and a connection is indicated by a nonzero number, ordered top to bottom, left to right.

3. The ordering of the parameters are important. Wrong parameter ordering results in `NeuroField` terminating with an error message.
4. With the exception of keywords, the configuration file is white-space independent, e.g., there can be either no spaces, many spaces, or new lines between the colon of `Integration steps:` `10000` and 10000, but you cannot have two spaces between `Integration` and `steps` . For consistency and readability, users are encouraged to follow the existing white space scheme when reasonable.
5. Tip for `vi` users: `./helper_scripts/neurofield.vim` implements syntax highlighting for configuration files in `vi` . See comments within for installation instructions.

### 1.4.1 Global information

- ```
  Purely cortical model with excitatory and inhibitory neurons
  ```

  Any text before the entry, `Integration steps:` , is disregarded by `NeuroField` and serves as comment.

- ```
  Time: 10 Deltat: .0001
  ```

  `Time` is the simulation duration in seconds.

  `Deltat` is the time increment for each time step.

- ```
  Nodes: 4 Longside: 2
  ```

  `Nodes` is the number of grid points in the spatial dimension per population of neurons. The code has been explicitly designed to have equal number of neurons per population.

  `Longside` is an optional parameter, specifying the longside of the rectangular grid. If it is not supplied, it is assumed to be a square.

  Both spatial dimensions have periodic boundary conditions, so that populations have the topology of a torus.

- ```
  Glutamate dynamics - fast Lambda: 3e-4 fast tGlu: 5e-3
                       slow Lambda: 3e-4 slow tGlu: 5e-3
  ```

```
Purely cortical model with excitatory and inhibitory neurons
Time: 10 Deltat: .0001
Nodes: 4
Glutamate dynamics - fast Lambda: 3e-4 fast tGlu: 5e-3
                     slow Lambda: 3e-4 slow tGlu: 5e-3

Connection matrix:
From: 1 2 3 4
To 1: 0 0 0 0
To 2: 0 0 0 0
To 3: 1 0 2 3
To 4: 0 4 5 6

Population 1: Stimulation
 Stimulus: Mode: Const - Onset: 0.0002 Mean: 5

Population 2: Stimulation
 Stimulus: Mode: Const - Onset: 0.0002 Mean: 5

Population 3: Excitatory neurons
 Q: 8.87145
 Firing: Sigmoid - Theta: 0.013 Sigma: 0.0038 Qmax: 340.0
  Dendrite 1: V: 0.0 alpha: 83 beta: 769
  Dendrite 2: alpha: 83 beta: 769
  Dendrite 3: alpha: 83 beta: 769

Population 4: Inhibitory neurons
 Q: 8.87145
 Firing: Sigmoid - Theta: 0.013 Sigma: 0.0038 Qmax: 340.0
  Dendrite 4: V: 0.0 alpha: 83 beta: 769
  Dendrite 5: alpha: 83 beta: 769
  Dendrite 6: alpha: 83 beta: 769

Propag 1: Map  - Tau: 0
Propag 2: Map  - Tau: 0
Propag 3: Wave - Tau: 0 Deltax: 0.0035 Range: 0.08 gamma: 116
Propag 4: Wave - Tau: 0 Deltax: 0.0035 Range: 0.08 gamma: 116
Propag 5: Map  - Tau: 0
Propag 6: Map  - Tau: 0
Couple 1: Map  - nu: 0.00015
Couple 2: Map  - nu: 0.00015
Couple 3: Map  - nu: 0.0015
Couple 4: Map  - nu: 0.0015
Couple 5: Map  - nu:-0.0018
Couple 6: Map  - nu:-0.0018

Output: Node: 1
Population:
Propag: 2
Couple:
```

Figure 2: An example configuration file, which can be found in `Configs/cortex.conf` .

Parameters governing glutamate dynamics, which is currently useful only for `CaDP` couples.

`Lambda` is the glutamate release concentration per presynaptic spike, in moles.

`tGlu` is the decay timescale of glutamate in seconds.

- ```
  Connection matrix:
  From: 1 2 3 4
  To 1: 0 0 0 0
  To 2: 0 0 0 0
  To 3: 1 0 2 3
  To 4: 0 4 5 6
  ```

We specify an arbitrarily sized square connection matrix, where each entry is the connection from the column population to the row population.

Zero indicates no connection.

A nonzero number indicates connection. This number must be indexed from top to bottom, left to right.

### 1.4.2 Population data

This section contains population information sections. There are two types of neural populations: ordinary populations and stimulus populations:

**Stimulus populations**

`NeuroField` identifies stimulus populations as populations which have no dendrites, i.e., the row for that population contains no nonzero elements. Each stimulus population information section is as follows.

- ```
  Population 1: Stimulation
  ```

  The identifier `Population 1` is required for cross-checking.
  The descriptor `Stimulation` is not parsed by `NeuroField` , but it is strongly recommended for human referencing.

- ```
  Stimulus: Mode: Const - Onset: 0.0002
  ```

  The identifier `Stimulus` is required for cross-checking.
  `Onset` specifies the time onset before the stimulus begins. Users should allow at least two timesteps for the transient state before onset.
  `Mode` specifies the stimulus mode, which includes, for example:
  Pulse stimulus pattern:

  ```
  Mode: Const - Amplitude: 20 Width: 0.02 Repetition: 1
  ```

  White noise stimulus pattern (spatially uncorrelated), where Ranseed is an optional parameter for the random number generator's initial seed defaulting to -98716872:

  ```
  Mode: White - Ranseed:-98716872 Amplitude: 20
  ```

  A "special" stimulus pattern is the composite pattern. At onset (encouraged to be set to zero), it superimposes an arbitrary number of stimuli:

  ```
  Stimulus: Mode: Composite - Onset: 0 Number of Stimuli: 2
   Mode: MNS - Onset: 0.5
    N20 width: .01 N20 height: 7 P25 width: .01 P25 height: 18
   Mode: Pulse - Onset: 0.53 Amplitude: 18 Width: 4e-3
  ```
  ```

**Ordinary populations**

Any non-stimulus population is an ordinary population.

- ```
  Population 3: Excitory neurons
  ```

  The identifier `Population 3` is required for cross-checking.
  The descriptor `Excitatory neurons` is not parsed by `NeuroField`, but it is strongly recommended for human referencing.

- ```
  Q: 8.87145
  ```

  The initial firing rate.

- ```
  Firing: Sigmoid - Theta: 0.013 Sigma: 0.0038 Qmax: 340.0
  ```

  Specify the sigmoidal firing response of the population.
  `Sigma` is sometimes known as $\tilde{\sigma}$. It is already scaled by $\pi/\sqrt{3}$.
  Alternatively, you can specify a linear firing response by using

  ```
  Firing: Linear - Gradient: 1 Intercept: 1
  ```

- ```
  Dendrite 1: V: Steady alpha: 83 beta: 769
  ```

  The identifier `Dendrite 1`, where 1 is the presynaptic connection index, is required for cross-checking. Users should find that these indices are simply ordered as 1, 2, 3, 4, ...
  `V` is the initial depolarization contribution from presynaptic activity. It can be a numerical value with units of V, or set at `Steady`, so that `NeuroField` calculates the initial value by $V_{ab} = \nu_{ab}\phi_{ab}$.
  `alpha` and `beta` are the parameters for the depolarization response.

### 1.4.3 Propagation data

```
Propag 1:
```

This identifier is required for cross-checking.

A propagator type is required at this point. Choices are `Map`, `Wave`, and `Harmonic`.

**Map**

```
Map - Tau: 0
```

This propagator is the mapping propagator where spatial spreading is negligible. Its form is given by

$$\phi_{ab}(\mathbf{r}, t) = Q_b(\mathbf{r}, t - \tau_{ab}).$$

Its only one parameter, `Tau`, is the the delay term. It is specified below.

**Wave**

This propagator is the wave equation propagator governed by the equation

$$\left[\frac{1}{\gamma_{ab}^2}\frac{d^2}{dt^2} + \frac{2}{\gamma_{ab}}\frac{d}{dt} + 1\right]\phi_{ab}(\mathbf{r}, t) = Q_b(\mathbf{r}, t - \tau_{ab}).$$

Its input is given by

```
Wave - phi: Steady Deltax: 0.0035 Tau: 0 Range: 0.08 gamma: 116
```

`phi` is the initial value for $\phi_{ab}$ in the wave equation. Inputting `Steady` gives $\phi_{ab} = Q_b$.

`Deltax` is the length of a node in mm. This must satisfy the Courant condition,

$$\Delta t/\Delta x < \sqrt{2}/r_e\gamma_e.$$

`Range` is $r_{ab}$ in the wave equation.

The final parameter can be `gamma` or `velocity` in the wave equation.

In case there is only one node, this degenerates into a `Harmonic` propagator.

**Harmonic** This is a harmonic oscillator implementation of the damped wave equation. If there is no spatial variation, use `Harmonic` instead of `WaveEqn` . The input form is given by

```
Harmonic - phi: Steady Tau: 0 gamma: 116
```

**Tau**

The axonal time delay between populations. If it is spatially homogeneous, then it is a number with units of seconds. If it is spatially inhomogeneous, then input $n$ numbers, where $n =$ `Nodes` .

### 1.4.4 Coupling data

- ```
  Couple 1:
  ```

  Identifier for cross-checking.

- A couple type is required at this point. Choices are `Map` , `Modcouple` , `CaDP` , and `STP` .

  **Map**

  Nonplastic synaptic coupling wtih a single constant parameter `nu` ,

  ```
  Map - nu: 0.0012
  ```

  `nu` is the synaptic coupling parameter. It corresponds to the product of the mean synaptic strength $s_{ab}$ and $N_{ab}$, the mean number of connections from cells of type $b$ to cells of type $a$.

  **CaDP**

  Calcium dependent plasticity according to Fung and Robinson.

  ```
  CaDP - nu: .4086e-4 Nu_max: .1e-3 Threshold: 1e-4 LTD: 5e-2
     LTP: 8e-2 B: 2e4 N: 10000 rho: 4200
  ```

  **STP**

  Short term plasticity.

### 1.4.5 Output data

`NeuroField` can output a time series for arbitrary fields on any particular nodes, into an output file named `neurofield.output` by default. A user chooses which objects to output, and on which nodes to output on. The object then outputs the field(s). For example, populations output $V_a$, propagators output $\phi_{ab}$, and couples output $\nu_{ab}$, and they may output additional fields if appropriate.

An example output is

```
                  Time   |           Propag.2.phi
                         |                       0
1.00000000000000e-04     |  8.87145000000000e+00
```
```

Each column is a time series with its name indicated in the first line. The first column is always time, and in this example, the second column is `Propag.2.phi` , indicating that it is $\phi_{ee}$ (when checked against connection matrix). The delimiter `|` indicates that the two columns are different fields, rather than different nodes of the same field. The node number is indicated in the second line.

To specify output in the configuration file, we start by

- ```
  Output: Node: 1 Start: 0 Interval: .0002
  ```

  We first specify which nodes to output. `Node: All` is a shorthand to output all nodes.

  `Start` is the time in seconds to start outputting. If this is not given, then output starts at time 0.

  `Interval` is the time interval in seconds between each output. If this is not given, then `NeuroField` outputs every timestep.

- ```
  Population: 3
  Propag: 2
  Couple: 2
  ```

  Specify an arbitrary number of objects to output.

## 1.5 Postprocessing

`NeuroField` produces 3 files:

| | |
|---|---|
| `neurofield.conf` | When using the launcher script, this file is created to store the running configuration file. |
| `neurofield.dump` | `NeuroField` dumps data here when it terminates. This file can be used in restart mode to continue simulation. Creation of this file can be suppressed by a `-s` switch. |
| `neurofield.output` | The result of the simulation is stored here for postprocessing. |
| `neurofield.pbs` | If `NeuroField` is run in `yossarian` , then this file stores the output of the queueing system. |

When the launcher script runs with only one set of parameters, all output files are also in the present working directory. However, if the launcher script sweeps over parameters, each parameter set has its own subdirectory inside `Output/` , and each set of `neurofield.*` files are stored in its subdirectory.

### 1.5.1 Quickplot

The data in `neurofield.output` may be plotted by `./Helper_script/quickplot.pl` or in `MatLab` via the matlab scripts within `./Helper_script/` .

To use `./Helper_scriptquickplot.pl` , execute

```
./Helper_script/quickplot.pl [output.file] [field] [node index]
```

where an example is

```
./Helper_script/quickplot.pl neurofield.output Propag.2.phi 1
```

or a shorthand to plot all fields and nodes is

```
./Helper_script/quickplot.pl all
```

All these commands launches `gnuplot` plotting sessions.

### 1.5.2 Matlab

A number of Matlab functions are provided to make it easy to manipulate NeuroField data from within Matlab. The functions are generally self-documenting with comments at the start of the file.

Essentially, an output file from NeuroField is read into a `nf` struct object in Matlab which simply contains all of the output from NeuroField in memory for easy access. Here is an example of a `nf` object:

```
     fields: {'Propag.1.phi'  'Propag.3.phi'}
      nodes: {[1]  [1]}
       data: {[300000x1 double]  [300000x1 double]}
       time: [300000x1 double]
     deltat: 1.0000e-04
    npoints: 300000
```

- `fields` stores a record of which traces from NeuroField are present in the output file
- `nodes` is a cell with the same size as `fields`, and records for each field present, the number of the node in the output.
- `data` is a matrix storing the actual values of the traces
- `time` is a vector of time values, so that you can plot any of the data traces directly again `nf.time`
- `deltat` stores the temporal sampling rate
- `npoints` stores the total number of points in the output. The total duration is `nf.deltat*nf.npoints` (or `nf.time(end)`)

There are two ways to create the `nf` object. You can read the output file directly after executing NeuroField elsewhere

```
nf = nf_read('neurofield.output')
```

or you can use the `nf_run` helper script to run the config file using NeuroField and automatically parse the output

```
nf = nf_run('neurofield.conf')
```

Using `nf_run` means you will need to add NeuroField to your shell search path.

Several helper files are provided to manipulate the `nf` object. The three most important helpers are `nf_extract` and `nf_grid`. Often you want to extract a particular field from the `nf` object, for example, to examine the output from `Propag.3.phi`. To do this directly with the `nf` object, you would need to check the `fields` variable to find the index of the trace you wanted, and then extract it from the `data` field. In the previous example, `Propag.3.phi` is the second trace. These expressions are identical:

```
trace = nf.data{2};
trace = nf_extract(nf,'Propag.3.phi')
```

`nf_extract` quickly becomes useful when there are many different fields. It is not case sensitive (so `propag.3.phi` works as well). You can also specify using additional arguments to extract only a portion of the time series, and also to select a subset of nodes. Finally, you can also provide multiple traces to concatenate them into a single matrix. For example,

```
trace = nf_extract(nf,'propag.1.phi,propag.3.phi');
```

will create a 300000x2 matrix with both of the traces.

Finally, if you run NeuroField with multiple nodes, it typically solves the system of equations on a square grid. Therefore, if you have output for notes 1-400, this corresponds to a 20x20 grid.

`nf_grid` allows you to request a trace from the `nf` object and have it reshaped into a square grid. This lets you easily make surface plots of the data, or perform tasks that are spatially dependent.

# 2 Developers guide

`NeuroField` is coded in `ANSI C++`. This guide assumes readers have basic knowledge of `C++`, including object oriented programming, usage of template, and standard template library (STL).

`NeuroField` solves each equation within the Robinson et al. model with an object:

$$D_{ab}V_{ab}(\mathbf{r},t) = \nu_{ab}\phi_{ab}(\mathbf{r},t), \qquad \text{Dendrite}$$

$$Q_a(\mathbf{r},t) = S_a\big[\sum_b V_{ab}(\mathbf{r},t)\big], \qquad \text{QResponse}$$

$$\mathcal{D}_{ab}\phi_{ab}(\mathbf{r},t) = Q_b(\mathbf{r},t-\tau_{ab}). \qquad \text{Propag}$$

`Propag` and `Couple` is overloaded to give more sophisticated axonal propagation (see Sec. 2.7.1) and synaptic plasticity (see Sec. 2.7.2), respectively.

## 2.1 Coding style

Within the C++ source files, developers should stick to this conventions for consistency:

| | |
|---|---|
| Tabs: | two spaces. |
| Braces: | the K&R style is strongly encouraged. |
| Class names: | UpperCamelCase |
| Function names: | lowerCamelCase |
| Variable names: | lowercase |

## 2.2 Class diagram

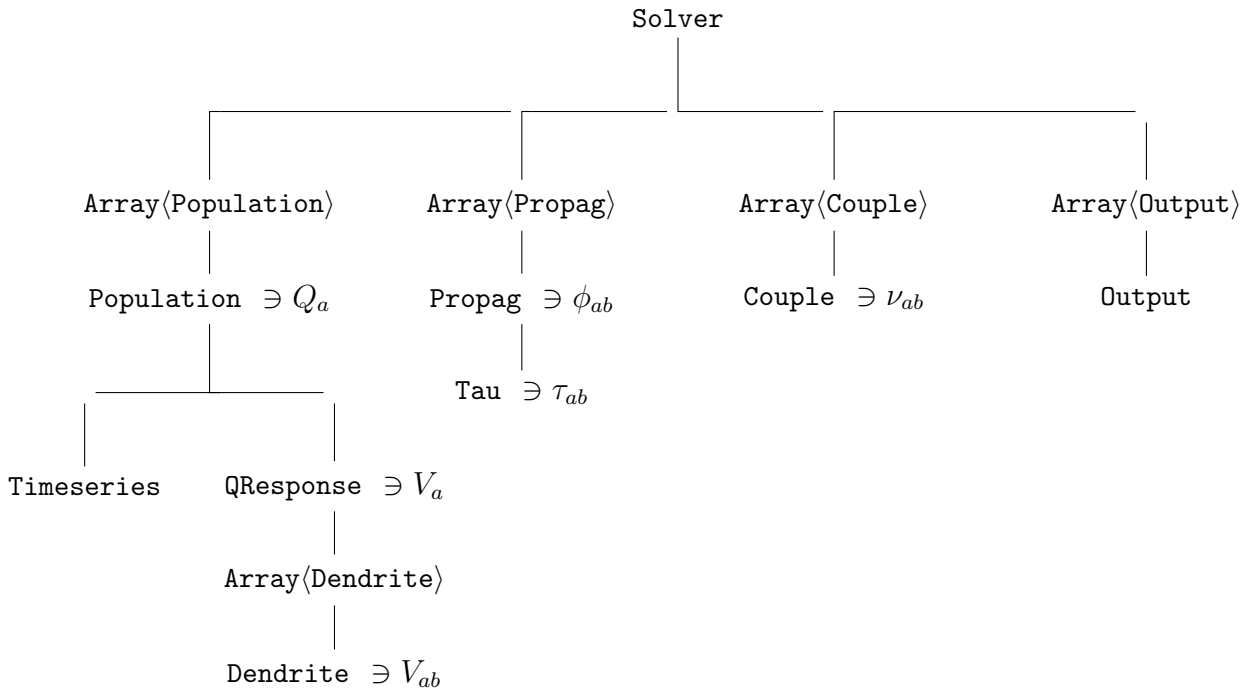



Figure 3: Schematic of the main class structures in `NeuroField`. Each line indicates that the bottom class is a member of the top class. The `a` $\ni b$ symbol indicates that the dynamical field $b(\mathbf{r},t)$ is a member of the class `a`. Inheritance structures are NOT illustrated.

## 2.3 Class NF

**ALL** classes that requires the configuration file should derive from the `NF` object. This abstract base class contains 3 member variables, and 4 interface methods:

Variable

| | |
|---|---|
| `nodes` | The number of nodes as specified from the configuration file. |
| `deltat` | The time increment per timestep in units of seconds. |
| `index` | The index associated with the object. |

Methods

| | |
|---|---|
| `init(Configf& configf)` | Initializes the object with the config file. |
| `dump(Dumpf& dumpf) const` | When the program terminates, all objects dump information into a dump file ( `dumpf` ) for later restart. `Dumpf` is an `ofstream` . |
| `restart(Restartf& restartf)` | Restarts the object in restart mode, *in addition* to `init()` . The developer should have dumped all relevant information in dump, then reads it here. |
| `step(void)` | At each timestep, this function is called. |

All `NF` classes automatically handles the `ofstream::<<` and `ifstream::>>` operators.

## 2.4 Class Array

`Array` is a container array to store objects that supports the `ofstream::<<` and `ifstream::>>` operators, as well as a `step(void)` function. This object typically is, but not necessarily has to be an `NF` object.

The `step(void)` function is equivalent to a `foreach(element).step()` in pseudocode. This function is encouraged over the use of `empty()` , `size()` , and `operator[]()` , which are discouraged to be used.

## 2.5 Class Population

Models a neural population, which may be either a stimulus or normal population. If it has any dendrites, i.e. it has presynaptic connections, then it is a normal population, and it is a stimulus if it does not have dendrites.

In the former case, it contains the `QResponse` class and have a soma potential; in the latter case it contains the `Timeseries` class, and does not have a soma potential. See Fig. 3 for reference.

In either cases, the `Population` class has a keyring storing the firing rate history, coded as a 2D array plus an integer key.

A population is "settled" after `Population::init()` is called, after which no dendrites can be added, and the firing rate history cannot grow.

## 2.6 Input/Output

Input via the configuration file is implemented in the `init()` function, via `Configf` , which provides the following functions:

1. next: go to the next keyword
2. param: go to the next keyword and reads in a variable. If the keyword is not found, barks and exits.
3. optional: same as param, but does not bark nor exit.
4. find: search a keyword and return the next variable as string.

In restart mode, `init()` is called, followed by `restart()`. `Restartf` is identical to `Configf`.

There are two modes of output. One is outputting the solution in `Population::output()`, `Propag::output()`, and `Couple::output()`, and the other is the dumping of data for restart implemented in `dump()`.

To output solutions, return a vector of pointers to `new Output`, specifying the name and field of the solution. The name must be of the format

```
label("Object.",index+1)+".field"
```

where `Object` and `field` should be appropriately named.

`dumpf()` provides a `Dumpf` object, which is essentially an `ofstream`. Dump all data here, so that it can be read in `restart()`.

## 2.7 Writing a new class

In extending `NeuroField` it is likely new classes needs to be written. When doing this, it would be useful to keep in mind:

1. When appropriate, the default constructor, copy constructor, and `operator=` should be made inaccessible by declaring them to be private.
2. If the configuration file is used, the `NF` class **MUST** be inherited.
3. The `init()`, `restart()`, `dump()`, `step()` functions should be overloaded appropriately. Keep in mind that `restart()` internally calls `init()` at the beginning.
4. `Population`, `Propag`, and `Couple` has the `output()` function to specify which field(s) to output. Overload this if appropriate.
5. When in doubt, read existing class implementations for examples.

### 2.7.1 Class Propag

The `Propag` class implements the axonal propagation as an identity map, i.e.

$$\mathcal{D}_{ab} = 1.$$

To introduce more sophisticated axonal propagation, this class is inherited and overloaded.

The `Propag` class provides a constant references to both presynaptic and postsynaptic populations.

For finite differences integration, `Stencil` provides a 9-point stencil. To use, initialize it with a `vector<double>`, then access the Moore grid with `nw`, `n`, ... . Loop over the stencil with the operator `++`.

To "register" your propagator, look for the `//PUT YOUR PROPAGATORS HERE` section in `solver.cpp`.

### 2.7.2 Class Couple

The `Couple` class manages $\nu_{ab}$, which is constant in space and time.

To introduce synaptic plasticity, derive from this class.

The `Couple` class provides a constant references to both presynaptic and postsynaptic populations. Glutamate concentration is also provided. `pos` is $+1$ or $-1$, depending on the sign of $\nu_{ab}$.

To "register" your couple, look for the `//PUT YOUR COUPLES HERE` section in `solver.cpp`.

### 2.7.3 Stimulus

To implement new stimulus pattern, use class `Timeseries`. `Timeseries::fire()` modifies a reference to firing rate across nodes. Keep in mind that the modification should be using `+=` rather than `=`, so that stimuli may be composited.

## 2.8 Program flow

Essentially, the program flow can be read from Fig. 3, so that objects take priority from top to bottom, left to right, both in terms of initialization and stepping through each timestep. A more detailed description is given below, and the reader is referred to the source code for complete description.

We use the semicolon to denote a succession of functions/procedures, and `a()` ⇒ `b()` symbol to denote function `b()` as content of function `a()` .

| | | |
|---|---|---|
| `main()` | ⇒ | Initialize the config file, dump file and output file; |
| | | `Solver::init()` ; `Solver::solve()` ; |
| `Solver::init()` | ⇒ | read in global parameters; Read in `CntMat` ; |
| | | Construct `Population` ; construct `Propag` ; construct `Couple` ; `Population::add2Dendrite()` ; |
| | | Read configurations for `Population` , `Propag` , `Couple` , and `Output` . |
| `Solver::solve()` | ⇒ | `for(...)` { `Solver::step()`; `Output::step()` ; } |
| `Solver::step()` | ⇒ | `Population::step()` ; `Propag::step()` ; `Couple::step()` ; |
| `Population::step()` | ⇒ | `QResponse::step()` if neural population; |
| | | `Timeseries::step()` if stimulus |
| `QResponse::step()` | ⇒ | `Dendrite::step()` ; sum over $V_{ab}$ |

- One integration step of the model implements the following stages: 1) Dendritic response 2) Afferent summation. 3) Firing response/stimulus response. 4) Wave equation integration step which includes Q delay processing 5) Coupling response.
- Most of the computational load comes from integrating wave equations and harmonic oscillators within the dendritic responses.
- Wave equations are integrated by explicit finite differences integration. A nine point spatial stencil is used to reduce high frequency spatial instabilities when driven by random noise. Other parts of code are unaffected by spatial geometry so this can be switched to irregular gridding easily.
- Harmonic oscillators with dendritic rseponse are integrated using a heavily strength reduced explicit direct integration assuming constant drive. This was more efficient than a constant drive RK4 algorithm which would not be fourth order in any case due to the constant drive. Rennie used a constant drive RK4 for his 1997 code.

## 2.9 Tools for solving differential equations

Classes `DE` and `Integrator` (currently RK4 is implemented) are used to solve generic systems of ODEs, where the dynamical variables are homogeneous fields. For inhomogeneous DEs and spatial dependency, a 9-point stencil is provided in `Stencil` .

**DE** To solve ODEs of homogeneous fields, inherit `DE` and define the purely virtual `rhs()` function. Instantiate a `DE` object and an integrator, then step through it with `integrator.step()` . `STP` is an example using classes `DE` and `RK4` .

**Stencil** Use `operator=` to apply a `Stencil` to a `vector<double>` , Then increment the stencil with `operator++` . The 9 stencil points can be read with `nw()` ... `se()` .

## 2.10 Submission to SVN

The SVN repository is a standard installation, so any tutorials on usage of SVN apply here. For developers, we suggest obtaining a copy of the entire repository with

```
svn co http://silliac.physics.usyd.edu.au:18080/svn/neurofield/
   neurofield --username=<your SVN username>
```

The following folders are present:

| | |
|---|---|
| `branches` | Folders for users to make changes to the program. |
| `tags` | Previous versions of the program. |
| `trunk` | Latest stable version of the program. |

To make changes to the program, it is recommended to copy the trunk into a new subfolder within branches

```
cp -r trunk ./branches/my_branch
```

You can then freely modify the code in the branch with the benefit of version control, and other users can obtain your code through the branch. If your modifications are accepted for the main program, your branch will then be merged into the trunk subject to appropriate testing.

## 2.11 TODO

1. check numerical entry of connection matrix
2. restart mode is crippled; rewrite the whole thing.