# NeuroField User Manual

Felix Fung Romesh Abeysuriya

January 25, 2015

`NeuroField` is a `C++` program (accompanied with helper scripts) that solves the neural field model of Robinson et al., where each of the simultaneous equations are handled by an object:

$$P = \nu_{ab}\phi_{ab}, \qquad \texttt{Couple}$$
$$D_{ab}V_{ab} = P, \qquad \texttt{Dendrite}$$
$$Q_a = S_a\big[\sum_b V_{ab}\big], \qquad \texttt{QResponse}$$
$$\mathcal{D}_{ab}\phi_{ab} = Q_b. \qquad \texttt{Propag}$$

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

1. Specify an arbitrary population model: an arbitrary number of populations and connections them may be specified;
2. Choose different types of populations, including neural or stimulus populations. For each neural population, the type of firing response, dendritic response type may be specified.
3. Choose different types of connections, including the type of axonal propagation, and synaptic coupling.
4. For each object, specify the parameter values.

This users guide covers the obtaining and setting up (Sec. 1), configuring (Sec. 3) and launching of `NeuroField` (Sec. 2), as well as postprocessing (Sec. 4) and tips and tricks (Sec. 5).

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

```
Command to put in computer
```

# Contents



# 1 Obtaining and setting up NeuroField

## 1.1 Obtaining NeuroField

The code for `NeuroField` is managed by version control system `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>
```

To obtain a copy of `NeuroField` from a computer which is not connected to the School of Physics network (e.g. personal laptops at home), you will require a version of `SVN` higher than 1.6. However, you may need to have your IP address/network domain registered for remote access. If you are unable to access the repository remotely, 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/` | Contains documentations `Documentation/user.pdf` and `Documentation/developer.pdf`. Running `make doc` generates these documentations. |
| `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. |
| `Scrap/` | Directory containing development scraps and is irrelevant for users. |
| `neurofield.*` | Output files generated by `NeuroField`. See Sec. 4. |

## 1.3 Compiling NeuroField

You can compile NeuroField simply by running

```
make
```

from the root directory containing all of the source files. The compiler command is specified in `Makefile` and should be edited if you wish to use a different compiler, or if your compiler does not support some of the compiler flags.

If you are compiling on Windows, the suggested route is to cross-compile using MinGW on a Unix-like system, and then copy across any missing DLLs from the Unix system into the same directory as the executable file on the Windows machine. Compiling with the Visual C++ compiler has not been tested.

# 2 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]

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.

   `-i` specify an configuration file name.

   `-o` specify an output file name.

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

5. In case the script is executed on computer cluster `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.

[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‘`.

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 generates `./Documentation/user.pdf` and `./Documentation/developer.pdf` . `make clean` deletes the files (excluding the `.pdf` files) created by LaTeX.

# 3 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/example.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. 3.1).
4. Specify all populations (Sec. 3.2).
5. Specify all propagators (Sec. 3.3).
6. Specify all couples (Sec. 3.4).
7. Specify all output requests (Sec. 3.5).

General rules on the entries within a configuration file:

1. The structure of a configuration file is: 1) comment 2) global information 3) object specification 4) output specification.

2. Object specification involves first specifying the object type. The syntax is the object identifier, followed by its type, then a hyphen:

```
Object 1: Type -
```

   Then the object parameters are specified, following this pattern:

```
Object parameter: value
```

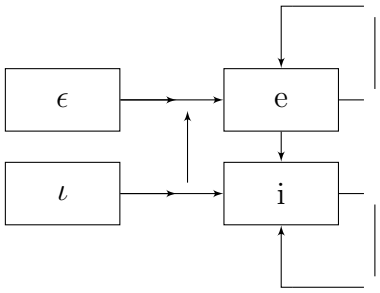
3. Most parameters are essential. Failure to provide these parameters would result in `NeuroField` terminating with an error message. A minority of the parameters are optional.

4. The ordering of the parameters are important. Wrong parameter ordering results in `NeuroField` terminating with an error message.

5. The configuration file is white-space independent, e.g., there can be either no spaces, many spaces, or new lines between parameters.

6. For readability, users are encouraged to arrange parameter entries for different objects (via new lines and indentations) and aligning corresponding parameters between different objects.

7. Tip for `vi` users: `./helper_scripts/neurofield.vim` implements syntax highlighting for configuration files in `vi` . See comments within for installation instructions.




| 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.

## 3.1 Global information

- 
```
Example config file of cortical model with excitatory and inhibitory
   neurons
```

```
Example config file of cortical model with excitatory and inhibitory
   neurons
Time: 1 Deltat: 1e-4
Nodes: 4 Longside: 2

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
 Length: .5
 Stimulus: Const - Onset: 0 Mean: 5

Population 2: Stimulation
 Length: .5
 Stimulus: Const - Onset: 0 Mean: 5

Population 3: Excitatory neurons
 Length: .5
 Q: 8.87145
 Firing: Sigmoid - Theta: 13e-3 Sigma: 3.8e-3 Qmax: 340
  Dendrite 1: V: Steady alpha: 83 beta: 769
  Dendrite 2: V: Steady alpha: 83 beta: 769
  Dendrite 3: V: Steady alpha: 83 beta: 769

Population 4: Inhibitory neurons
 Length: .5
 Q: 8.87145
 Firing: Sigmoid - Theta: 13e-3 Sigma: 3.8e-3 Qmax: 340
  Dendrite 4: V: Steady alpha: 83 beta: 769
  Dendrite 5: V: Steady alpha: 83 beta: 769
  Dendrite 6: V: Steady alpha: 83 beta: 769

Propag 1: Map  - phi: Steady Tau: 0
Propag 2: Wave - phi: Steady Tau: 0 Range: 80e-3 gamma: 116
Propag 3: Map  - phi: Steady Tau: 0
Propag 4: Map  - phi: Steady Tau: 0
Propag 5: Wave - phi: Steady Tau: 0 Range: 80e-3 gamma: 116
Propag 6: Map  - phi: Steady Tau: 0

Couple 1: Map - nu: .15e-3
Couple 2: Map - nu: 1.5e-3
Couple 3: Map - nu:-1.8e-3
Couple 4: Map - nu: .15e-3
Couple 5: Map - nu: 1.5e-3
Couple 6: Map - nu:-1.8e-3

Output: Node: 1 2 Start: 0 Interval: 1e-4
Population: 4.V
Dendrite: 5
Propag: 1 4.phi
Couple: 3.nu
```

Any text before the entry, `Time:` , is disregarded by `NeuroField` and serves as comment, which is strongly recommended for all configuration files.

- ```
  Time: 10 Deltat: 1e-4
  ```

  `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.

- ```
  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.

## 3.2 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.

- ```
  Length: .5
  ```

The physical (1D) length of the population (which is a 2D sheet), in mitres. This is used in `Wave` propagators and the `Psd` of `White` stimulus.

- ```
  Stimulus: Const - Onset: 0
  ```

The identifier `Stimulus` is required for cross-checking.

This is followed by the type of stimulus, to be further elaborated below.

Optional parameter `Onset` specifies the time onset for the stimulus to begin. If unspecified, stimulus starts at time 0.

Either `Cease` or `Duration` can be an optional parameter to specify the end time of the stimulus. If unspecified, stimulus ends at 1000 seconds.

If optional parameter `Node` is specified, only the specified node indices will receive stimulation.

Possible stimulus patterns:

**Constant**

```
Const - Mean: 5
```

**Pulse**

```
Pulse - Amplitude: 1 Width: 2e-2 Frequency: 1 Pulses: 1
```

**White**

Gaussian noise, characterized by the mean and standard deviation:

```
White - Mean: 1 Std: 20 Ranseed: 10
```

Alternatively, the power spectral density (PSD) may be specified instead of the standard deviation. The advantage is that the PSD is invariant to change in `Deltat`, population `Length` and spatial `Nodes`. Given the PSD, `NeuroField` correctly calculates the standard deviation:

```
White - Mean: 1 Psd: 20 Ranseed: 10
```

In general, it is preferable to specify the noise using PSD rather than Std. The magnitude of nonlinear effects and the total power in the spectrum both depend on the PSD rather than the Std.

The random number generator may be specified in `Ranseed`. If a seed is not specified, an automatically-incremented seed will be used instead, so that multiple stimulus populations will have independent sequences. In general it is not necessary to set the seed manually unless different random numbers are required for otherwise identical runs.

**Superimposing stimuli**

To superimpose 2 or more stimuli, begin with the keyword `Superimpose` , followed by the number of stimuli. Then list all the stimulus patterns and their parameters, with each stimulus pattern preceded by the keyword `Stimulus` .

```
Stimulus: Superimpose: 2
 Stimulus: White - Mean: 1 Psd: 1
 Stimulus: Pulse - Onset: 0.5 Width: 2e-2 Frequency 1 Pulses: 1
```

**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: 13e-3 Sigma: 3.8e-3 Qmax: 340
  ```

  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 the number 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, ...

  Optional parameter `V` may be used to specify the initial depolarization contribution from presynaptic activity. If unspecified, or set to `Steady` , `NeuroField` calculates the initial value by $V_{ab} = \nu_{ab}\phi_{ab}$.

  `alpha` and `beta` are the parameters for the depolarization response.

## 3.3 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 - phi: Steady Tau: 0
```

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

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

Optional parameter, `Tau` , is the axonal delay term. If unspecified, it is taken as zero. Since all propagators contain this object, its description is given below.

Optional parameter `phi` may be used to specify the initial axonal firing rate. If unspecified, or set to `Steady` , `NeuroField` calculates the initial value by $\phi_{ab} = Q_b$.

**Wave**

This propagator is the wave equation propagator governed by the equation

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

`NeuroField` checks whether the Courant condition must be satisfied, i.e.

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

where $\Delta x$ is the population length per node.

The propagator input is given by

```
Wave - phi: Steady Tau: 0 Range: 80e-3 gamma: 116
```

Optional parameter `phi` may be used to specify the initial axonal firing rate. If unspecified, or set to `Steady` , `NeuroField` calculates the initial value by $\phi_{ab} = Q_b$.

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

`gamma` is the damping coefficient. Alternatively, `velocity` may be specified, and `gamma` is calculated via $\gamma_{ab} = \mathrm{v}_{ab}/r_{ab}$.

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, with no spatial variations:

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

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` .

## 3.4 Coupling data

- ```
  Couple 1:
  ```

  Identifier for cross-checking.

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

  **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 $a$.

  **Matrix**

  Coupling becomes connection matrix, where connection strength does *not* change with time. The format of the nu matrix is the same as the population connection matrix, each row is to the same node, each column is from the same node. When outputting, each specified outputting node output the indexed row.

  ```
  nu:
    13e-6 0
    0 13e-6
  ```

  **CaDP**

  Calcium dependent plasticity according to Fung and Robinson.

  ```
  CaDP - nu: 13e-6 nu_max: 80e-6 Dth: .25e-6 Pth: .45e-6 xyth: 1e-4
     x: 2.3e-2 y: 2e-2 B: 30e3 glu_0: 200e-6 gNMDA: 2e-3
  ```

  `Dth` and `Pth` are the calcium-plasticity thresholds; `xyth` , `x` and `y` are the plasticity rates; `B` , `glu_0` and `gNMDA` are NMDA receptor parameters.

  To use `CaDP` , glutamate dynamics must be specified for the postsynaptic population. In the end of the relevant population entry, append

  ```
  Glutamate dynamics - Lambda: 150e-6 tGlu: 30e-3
  ```

  `Lambda` is the glutamate concentration rise per presynaptic spike; `tGlu` is the decay timescale for glutamate dynamics.

  **BCM**

  Extends `CaDP` with metaplasticity according to Fung and Robinson. it has an additional parameter `t_BCM` , the timescale of metaplasticity.

  ```
  CaDP - nu: 13e-6 nu_max: 80e-6 Dth: .25e-6 Pth: .45e-6 xyth: 1e-4
     x: 2.3e-2 y: 2e-2 B: 30e3 glu_0: 200e-6 gNMDA: 2e-3 t_BCM: 7
  ```
  ```

### 3.5 Output data

`NeuroField` outputs field quantities (i.e. a neurodynamic quantity which takes a value for each node) with respect to nodes and time. By default, the output file is `neurofield.output`, which can be changed by launching the program with the `-o` switch.

- ```
  Output:
  ```

  Begin with the `Output` declaration.

- ```
  Node: 1 2
  ```

  Enumerate all nodes to be outputted. If outputting all nodes, use shorthand `All`. If no nodes are specified, no nodes will be outputted.

- ```
  Start: 0 Interval: 1e-4
  ```

  Optional parameters for the time to start output, and optional parameter for time interval between outputs.

  If undefined, defaults, to 0 and `Deltat`, respectively.

- ```
  Population: 4.V
  Dendrite: 5
  Propag: 1 4.phi
  Couple: 3.nu
  ```

  `NeuroField` allows the user to specify which objects to output, by entering the appropriate object indices after the labels. For each object, it has some intrinsic fields that will be outputted; for example, `Couple` outputs `nu`, whereas `CaDP` outputs `nu` and `Ca`.

  For each entry, a field name may be appended after the index with a dot, so that only that field of the object is outputted. If no field name is specified for that entry, then all fields of that object is outputted.

  If a specific field of an object is specified, but that field does not exist, `NeuroField` checks and returns an error.

# 4 Postprocessing

`NeuroField` produces 3 files:

| | |
|---|---|
| `neurofield.conf` | When using the launcher script, this file is created to store the running configuration file. |
| `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.

Example content in `neurofield.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.

It is also worth noting that traces will be written in the order that they are specified. For example, if you write `Population: 3 1` then the columns in the output file will be arranged in this order.

## 4.1 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.

## 4.2 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')
```

Several helper files are provided to manipulate the nf object. The two 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 300000×2 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 20×20 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.

One important task is computing the power spectrum as predicted by the linearized analytic equations. This can be achieved using `nf_spatial_spectrum` which takes in an `nf` object and computes the power spectrum integrated over $k$ taking into account volume conduction.

# 5 Tips and tricks