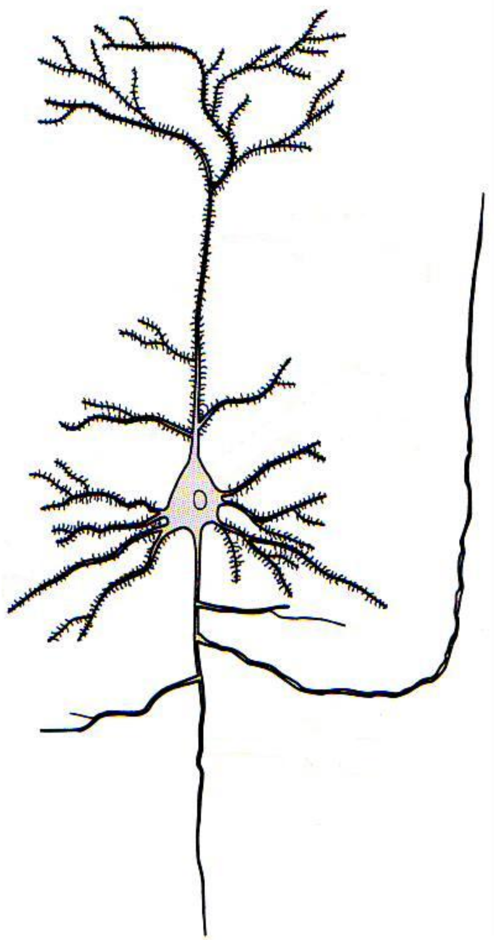
Simulating neural computation and information processing with BRIAN2

www.briansimulator.org

Marcel Stimberg
Computational Neuroscience of Sensory Systems Group – Institut de la Vision
marcel.stimberg@inserm.fr

Plan for today



- Introduction to modelling with Brian

Interactive tutorial ("live coding"):

- The jupyter notebook
- **Part 1:** Modelling neurons
- **Part 2:** Modelling synapses
- **Part 3:** Case study: coincidence dectors

Course material

Updated material will be uploaded here:

github.com/brian-team/brian-material/tree/master/2018-TD-Brian-Sorbonne

To download everything in a single ZIP file (includes material from other courses as well):
github.com/brian-team/brian-material/archive/master.zip

To download individual jupyter notebook files, make sure to switch to “raw” view

BRIAN^2

Who is Brian

- Simulator for spiking neuronal networks, written in Python
- Started by Dan Goodman and Romain Brette at ENS Paris in 2007
- "A simulator should not only save the time of processors, but also the time of scientists"
- Does not provide a library of fixed models but allows for a flexible definition of (almost) arbitrary models
- Focusses on "medium-sized" neuronal networks ("a few" to ~100000 neurons), simulations on standard PCs, not supercomputers
- Tool for research and teaching
- Free-and-open-source

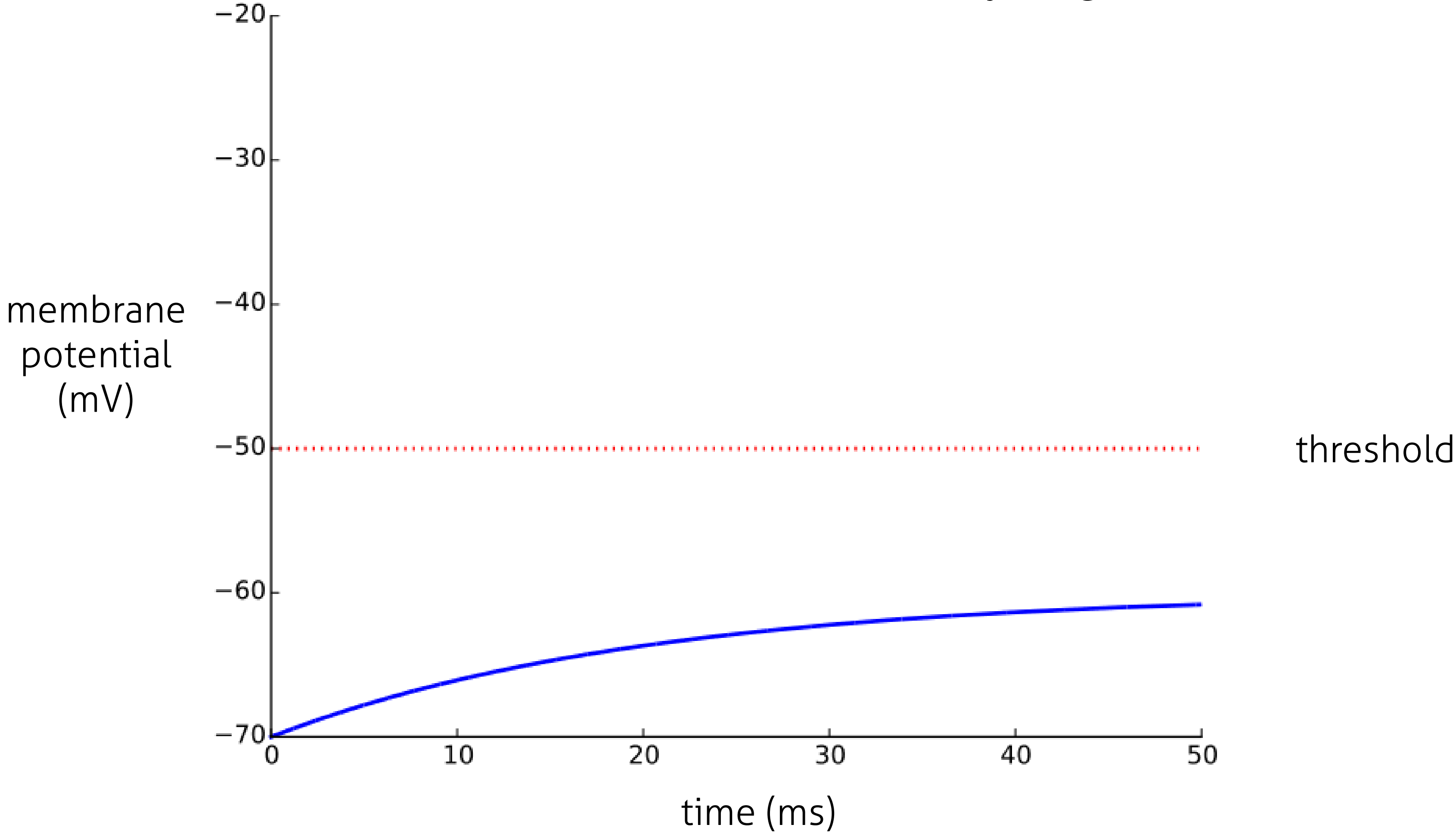
Brian's philosophy:

Mathematical model descriptions

Stimberg M, Goodman DF, Benichoux V, and Brette R (2014)
Equation-Oriented Specification of Neural Models for Simulation
Frontiers in Neuroinformatics

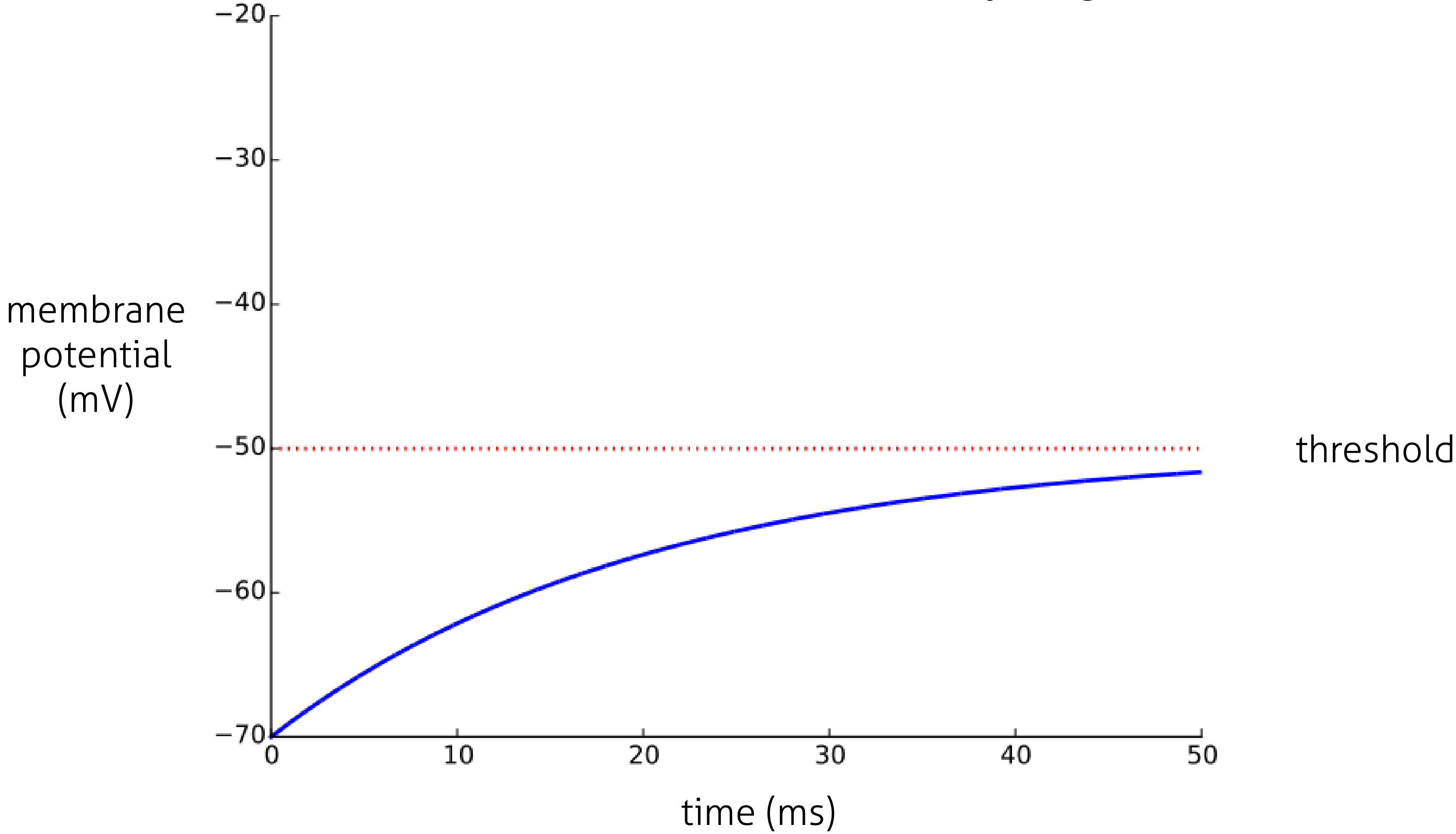
Modelling neural activity

“Leaky integrate-and-fire neuron”

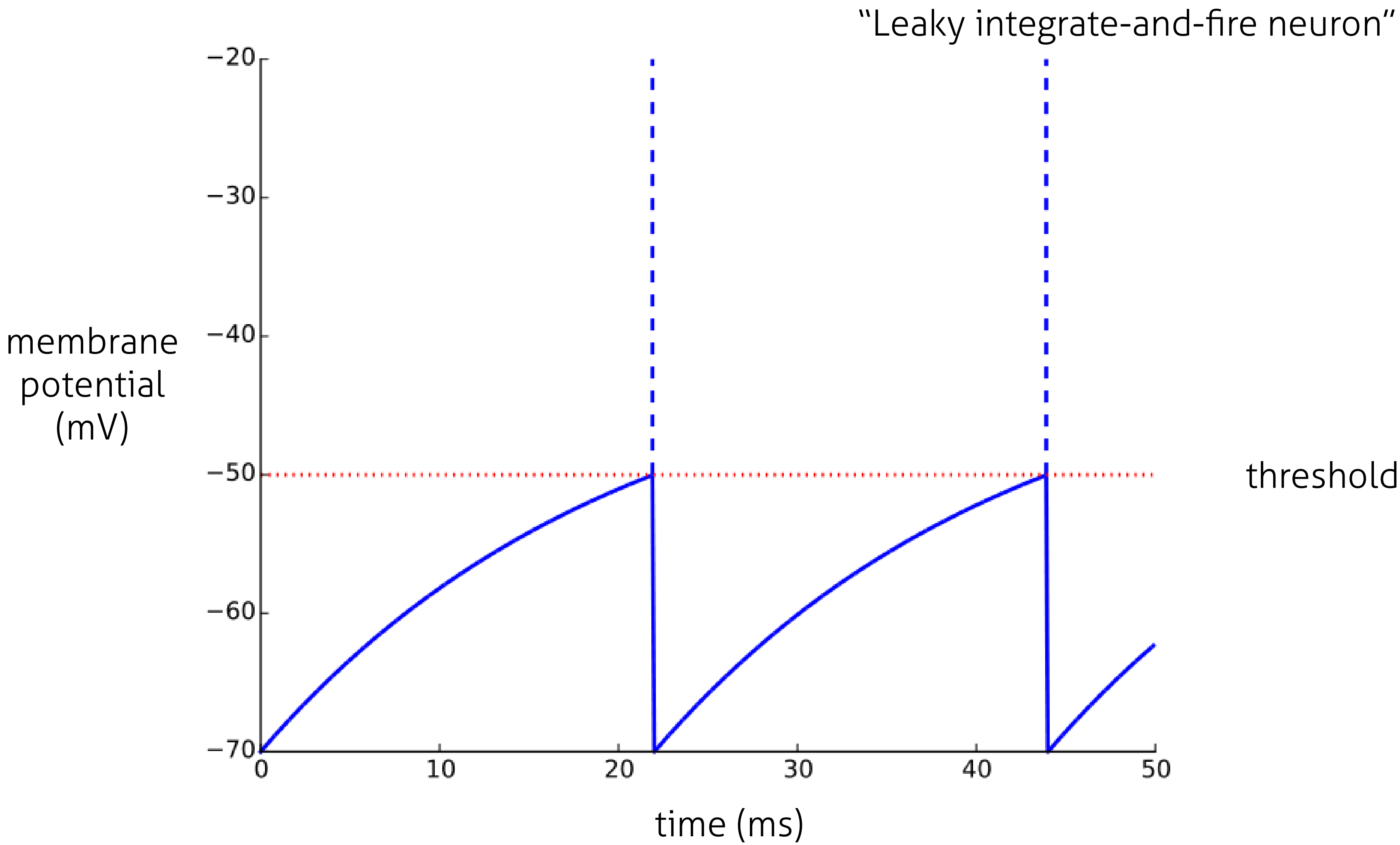


Modelling neural activity

“Leaky integrate-and-fire neuron”



Modelling neural activity



Modelling neural activity

Mathematical description:

$$
\begin{aligned}
Cm &= 200\,\mathrm{pF}\\
g_L &= 10\,\mathrm{nS}\\
E_L &= -70\,\mathrm{mV}\\
v_r &= E_L\\
v_{\mathrm{th}} &= -50\,\mathrm{mV}\\
v_{\mathrm{r}} &= E_L\\
\frac{\mathrm{d}v}{\mathrm{d}t} &= \frac{1}{Cm}\left(I + g_L\left(E_L - v\right)\right)
\end{aligned}
$$

v has units of volt,
I has units of amp(ère)

When $v > v_{\mathrm{th}}$:

$v \leftarrow v_r$

Modelling neural activity

Mathematical description:

Constants

$$Cm = 200\,\mathrm{pF}$$
$$g_L = 10\,\mathrm{nS}$$
$$E_L = -70\,\mathrm{mV}$$
$$v_r = E_L$$
$$v_{\mathrm{th}} = -50\,\mathrm{mV}$$
$$v_{\mathrm{r}} = E_L$$
$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{1}{Cm}\left(I + g_L\left(E_L - v\right)\right)$$

v has units of volt,
I has units of amp(ère)

When $v > v_{\mathrm{th}}$:

$$v \leftarrow v_r$$

Modelling neural activity

Mathematical description:

$$
\begin{aligned}
Cm &= 200\,\mathrm{pF} \\
g_L &= 10\,\mathrm{nS} \\
E_L &= -70\,\mathrm{mV} \\
v_r &= E_L \\
v_{\mathrm{th}} &= -50\,\mathrm{mV} \\
v_{\mathrm{r}} &= E_L
\end{aligned}
$$

Constants

Continuous dynamics

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{1}{Cm}\left(I + g_L\left(E_L - v\right)\right)$$

v has units of volt,
I has units of amp(ère)

When $v > v_{\mathrm{th}}$:

$$v \leftarrow v_r$$

Modelling neural activity

Mathematical description:

$$Cm = 200\,\mathrm{pF}$$
$$g_L = 10\,\mathrm{nS}$$
$$E_L = -70\,\mathrm{mV}$$
$$v_r = E_L$$
$$v_{\mathrm{th}} = -50\,\mathrm{mV}$$
$$v_{\mathrm{r}} = E_L$$

Constants

Continuous dynamics

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{1}{Cm}\left(I + g_L\left(E_L - v\right)\right)$$

v has units of volt,
I has units of amp(ère)

Physical units

When $v > v_{\mathrm{th}}$:

$$v \leftarrow v_r$$

Modelling neural activity

Mathematical description:

$$Cm = 200\,\mathrm{pF}$$
$$g_L = 10\,\mathrm{nS}$$
$$E_L = -70\,\mathrm{mV}$$
$$v_r = E_L$$
$$v_{\mathrm{th}} = -50\,\mathrm{mV}$$
$$v_{\mathrm{r}} = E_L$$

Constants

Continuous dynamics

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{1}{Cm}\left(I + g_L\left(E_L - v\right)\right)$$

v has units of volt,
I has units of amp(ère)

Physical units

When $v > v_{\mathrm{th}}$:

$$v \leftarrow v_r$$

Discontinuous state change

Modelling neural activity

Brian description:

```
Cm = 200*pF
g_L = 10*nS
E_L = -70*mV
v_r = E_L
v_th = -50*mV

G = NeuronGroup(N,
                '''dv/dt = 1/Cm * (I + g_L * (E_L - v)) : volt
                   I : amp''',
                threshold='v > v_th',
                reset='v = v_r')
```

Modelling neural activity

Brian description:

Constants

```
Cm = 200*pF
g_L = 10*nS
E_L = -70*mV
v_r = E_L
v_th = -50*mV

G = NeuronGroup(N,
                '''dv/dt = 1/Cm * (I + g_L * (E_L - v)) : volt
                   I : amp''',
                threshold='v > v_th',
                reset='v = v_r')
```

Modelling neural activity

Brian description:

Constants

```
Cm = 200*pF
g_L = 10*nS
E_L = -70*mV
v_r = E_L
v_th = -50*mV

G = NeuronGroup(N,
                '''dv/dt = 1/Cm * (I + g_L * (E_L - v)) : volt
                   I : amp''',
                threshold='v > v_th',
                reset='v = v_r')
```

Continuous dynamics

Modelling neural activity

Brian description:

```
Cm = 200*pF
g_L = 10*nS
E_L = -70*mV
v_r = E_L
v_th = -50*mV

G = NeuronGroup(N,
                '''dv/dt = 1/Cm * (I + g_L * (E_L - v)) : volt
                   I : amp''',
                threshold='v > v_th',
                reset='v = v_r')
```

Constants

Continuous dynamics

Physical units

Modelling neural activity

Brian description:

```
Cm = 200*pF
g_L = 10*nS
E_L = -70*mV
v_r = E_L
v_th = -50*mV

G = NeuronGroup(N,
                '''dv/dt = 1/Cm * (I + g_L * (E_L - v)) : volt
                   I : amp''',
                threshold='v > v_th',
                reset='v = v_r')
```

Constants

Continuous dynamics

Physical units

Discontinuous state change

Part 1: Modelling neurons

[see jupyter notebook]

Part 2: Modelling synapses

[see jupyter notebook]

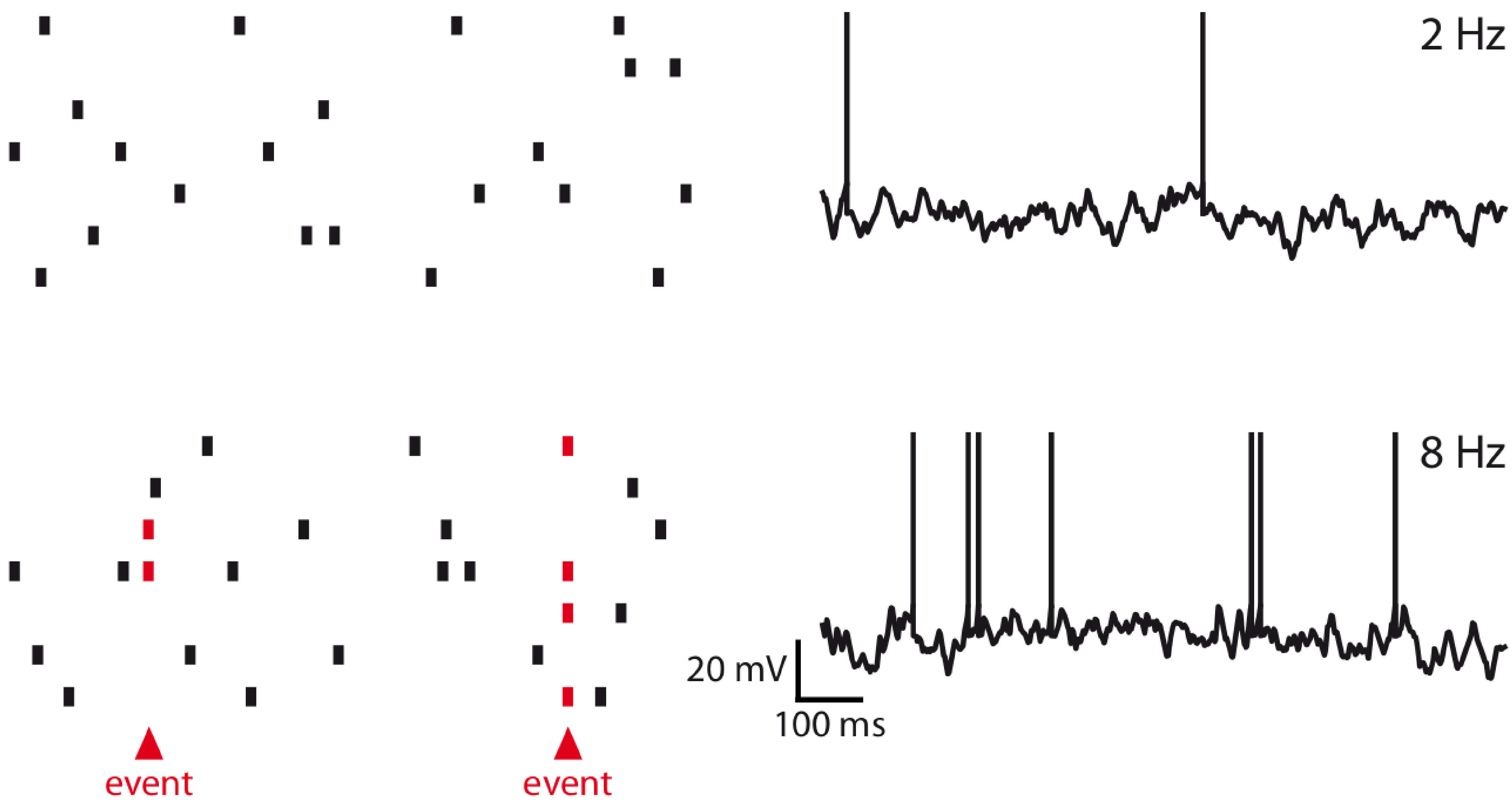
Part 3: Case Study

The Journal of Neuroscience, November 23, 2011 • 31(47):17193–17206 • 17193

Cellular/Molecular

Sensitivity of Noisy Neurons to Coincident Inputs

Cyrille Rossant,[1,2] Sara Leijon,[3,4] Anna K. Magnusson,[3,4] and Romain Brette[1,2]
[1]Laboratoire Psychologie de la Perception, CNRS, Université Paris Descartes, F-75006 Paris, France, [2]Equipe Audition, Département d'Etudes Cognitives, Ecole Normale Supérieure, 75005 Paris, France, and [3]Center for Hearing and Communication Research and [4]Department of Clinical Science, Intervention and Technology, Karolinska Institutet, 17176 Stockholm, Sweden


2 Hz
8 Hz
20 mV
100 ms
event
event

[see jupyter notebook]

Further material about Brian 2

Website

briansimulator.org

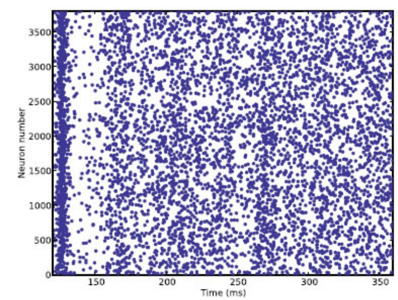
The Brian spiking neural network simulator

Posts Comments

Brian is a free, open source simulator for spiking neural networks. It is written in the Python programming language and is available on almost all platforms. We believe that a simulator should not only save the time of processors, but also the time of scientists. Brian is therefore designed to be easy to learn and use, highly flexible and easily extensible.

To get an idea of what writing a simulation in Brian looks like, the following code defines a randomly connected network of integrate and fire neurons with exponential inhibitory and excitatory currents, runs the simulation and makes the raster plot on the right.

```
from brian2 import *
eqs = '''
dv/dt  = (ge+gi-(v+49*mV))/(20*ms) : volt
dge/dt = -ge/(5*ms)                : volt
dgi/dt = -gi/(10*ms)               : volt
'''
P = NeuronGroup(4000, eqs, threshold='v>-50*mV', reset='v=-60*mV')
P.v = -60*mV
Pe = P[:3200]
Pi = P[3200:]
Ce = Synapses(Pe, P, on_pre='ge+=1.62*mV')
Ce.connect(p=0.02)
Ci = Synapses(Pi, P, on_pre='gi-=9*mV')
Ci.connect(p=0.02)
M = SpikeMonitor(P)
run(1*second)
plot(M.t/ms, M.i, '.')
show()
```



Interactive demo

We also have an interactive demo (running on mybinder.org) that lets you modify parameters and even change the simulation code, running from the web browser without installing anything.

Getting started

Once you've decided you want to use Brian, you can click the links on the left hand side to install Brian, go through the tutorials, look at example code and read the full documentation. If you have problems, we have an email support list.

Other software and tools

We have produced the following tools in addition to the main Brian simulator:

- brian2tools: for simple plotting and analysis with Brian

You can also download the older version of Brian (1.4) and the toolboxes we designed for it:

- Brian Hears (compatible with Brian 2): for modelling the auditory system
- Model fitting toolbox: for automatically fitting models to electrophysiological data

How to cite Brian

If you use Brian for your published research, we suggest that you cite one of our introductory articles:

1. Goodman DF and Brette R (2009). The Brian simulator. *Front Neurosci* doi:10.3389/neuro.01.026.2009
2. Stimberg M, Goodman DFM, Benichoux V, Brette R (2014). Equation-oriented specification of neural models for simulations. Frontiers Neuroinf, doi: 10.3389/fninf.2014.00006.

You can also download our logo for posters and presentations.

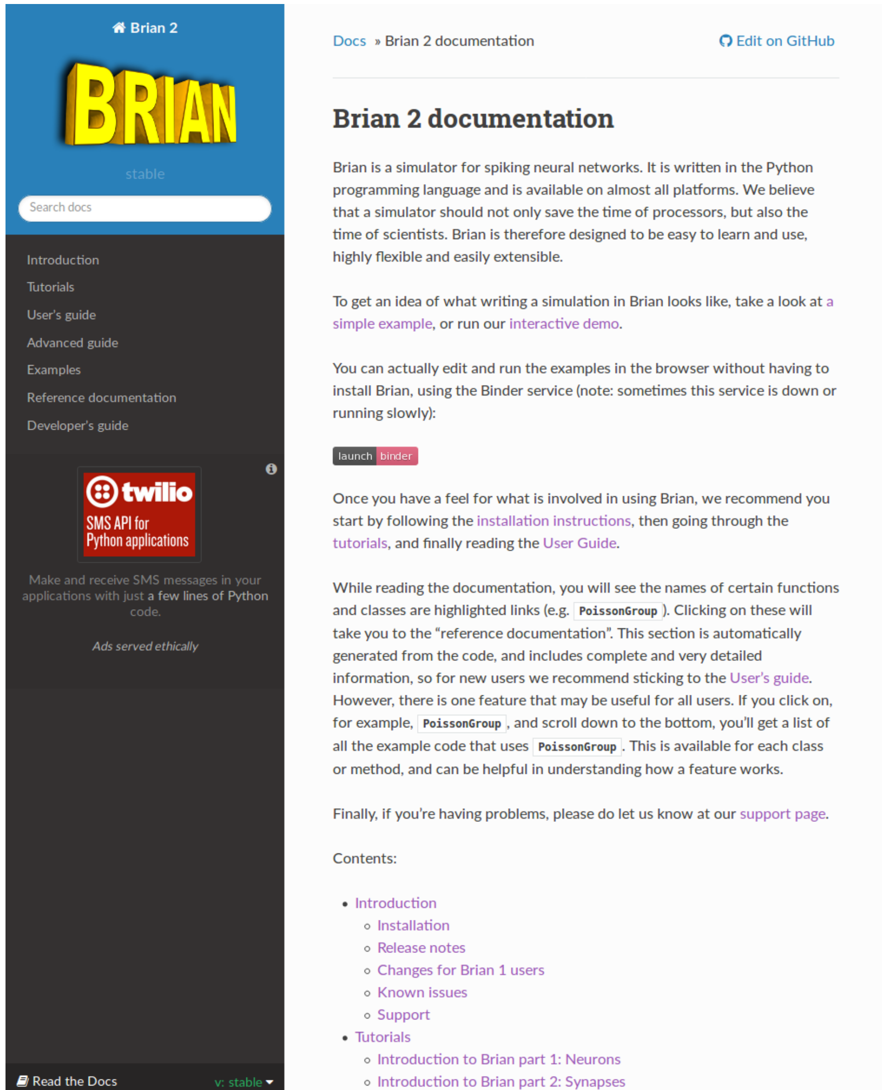
The Brian team

Brian is being developed by:

- Romain Brette (l'Institut de la Vision, Paris)

Documentation

brian2.readthedocs.org



Brian 2

BRIAN

stable

Search docs

- Introduction
- Tutorials
- User's guide
- Advanced guide
- Examples
- Reference documentation
- Developer's guide

Read the Docs v: stable

Docs » Brian 2 documentation Edit on GitHub

Brian 2 documentation

Brian is a simulator for spiking neural networks. It is written in the Python programming language and is available on almost all platforms. We believe that a simulator should not only save the time of processors, but also the time of scientists. Brian is therefore designed to be easy to learn and use, highly flexible and easily extensible.

To get an idea of what writing a simulation in Brian looks like, take a look at a simple example, or run our interactive demo.

You can actually edit and run the examples in the browser without having to install Brian, using the Binder service (note: sometimes this service is down or running slowly):

launch binder

Once you have a feel for what is involved in using Brian, we recommend you start by following the installation instructions, then going through the tutorials, and finally reading the User Guide.

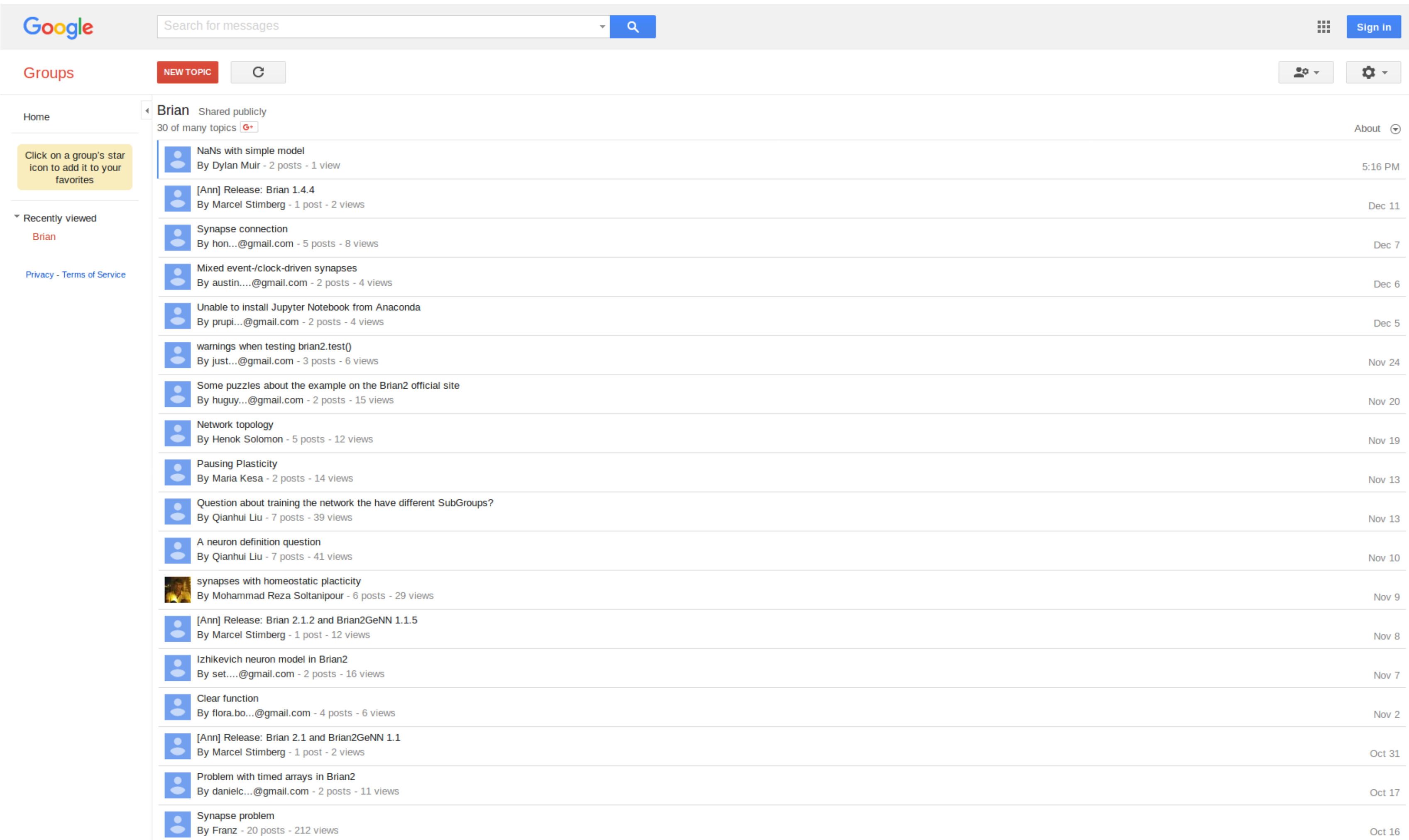
While reading the documentation, you will see the names of certain functions and classes are highlighted links (e.g. `PoissonGroup`). Clicking on these will take you to the "reference documentation". This section is automatically generated from the code, and includes complete and very detailed information, so for new users we recommend sticking to the User's guide. However, there is one feature that may be useful for all users. If you click on, for example, `PoissonGroup`, and scroll down to the bottom, you'll get a list of all the example code that uses `PoissonGroup`. This is available for each class or method, and can be helpful in understanding how a feature works.

Finally, if you're having problems, please do let us know at our support page.

Contents:

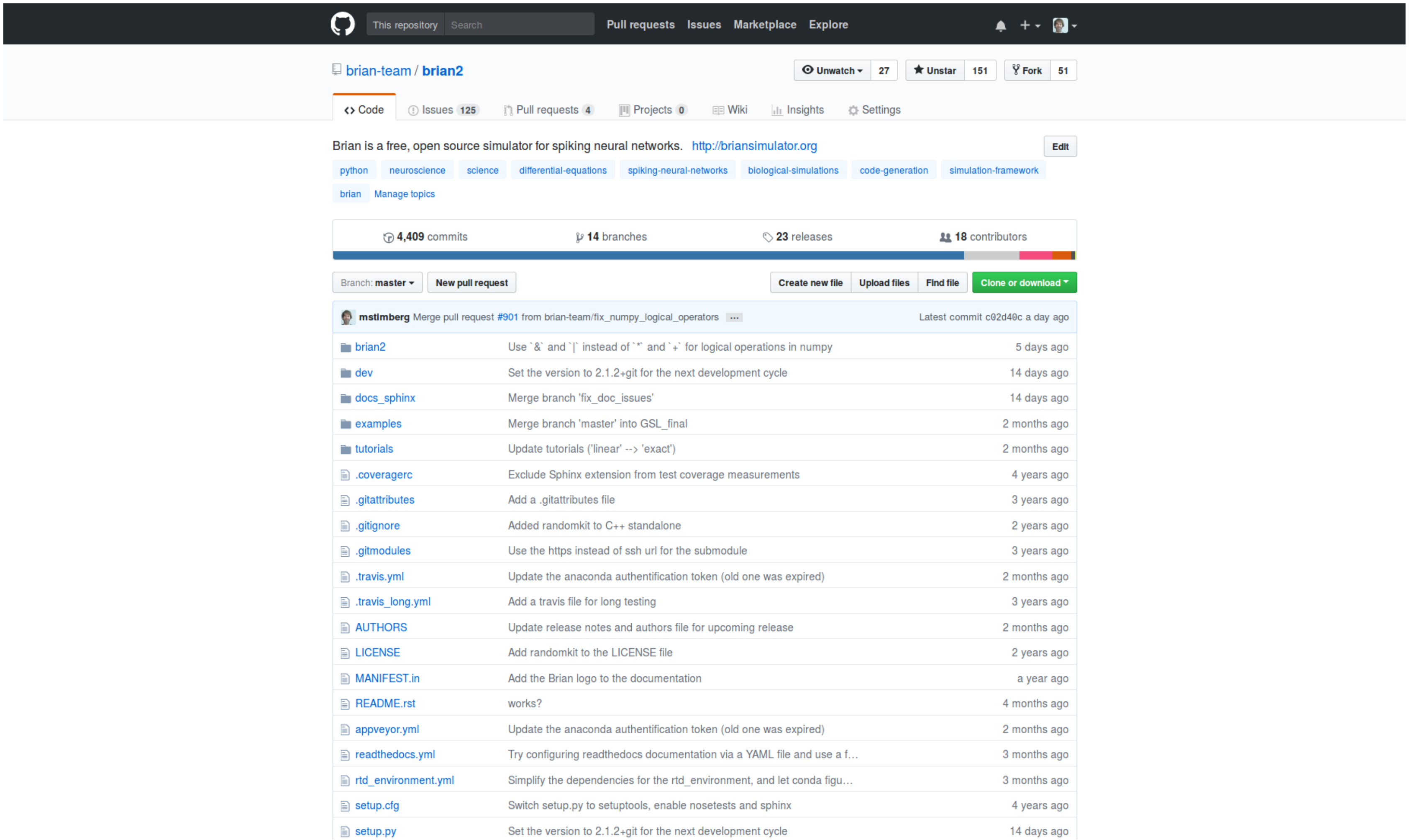
Mailing list

briansupport@googlegroups.com
groups.google.com/forum/#!forum/briansupport



Code repository

github.com/brian-team/brian2



Installation

Recommended way:

- Use Anaconda distribution
- Add the "conda-forge" channel

```
conda config --add channels conda-forge
```

- Install brian2

```
conda install brian2
```

More infos and alternative installation:
brian2.readthedocs.org