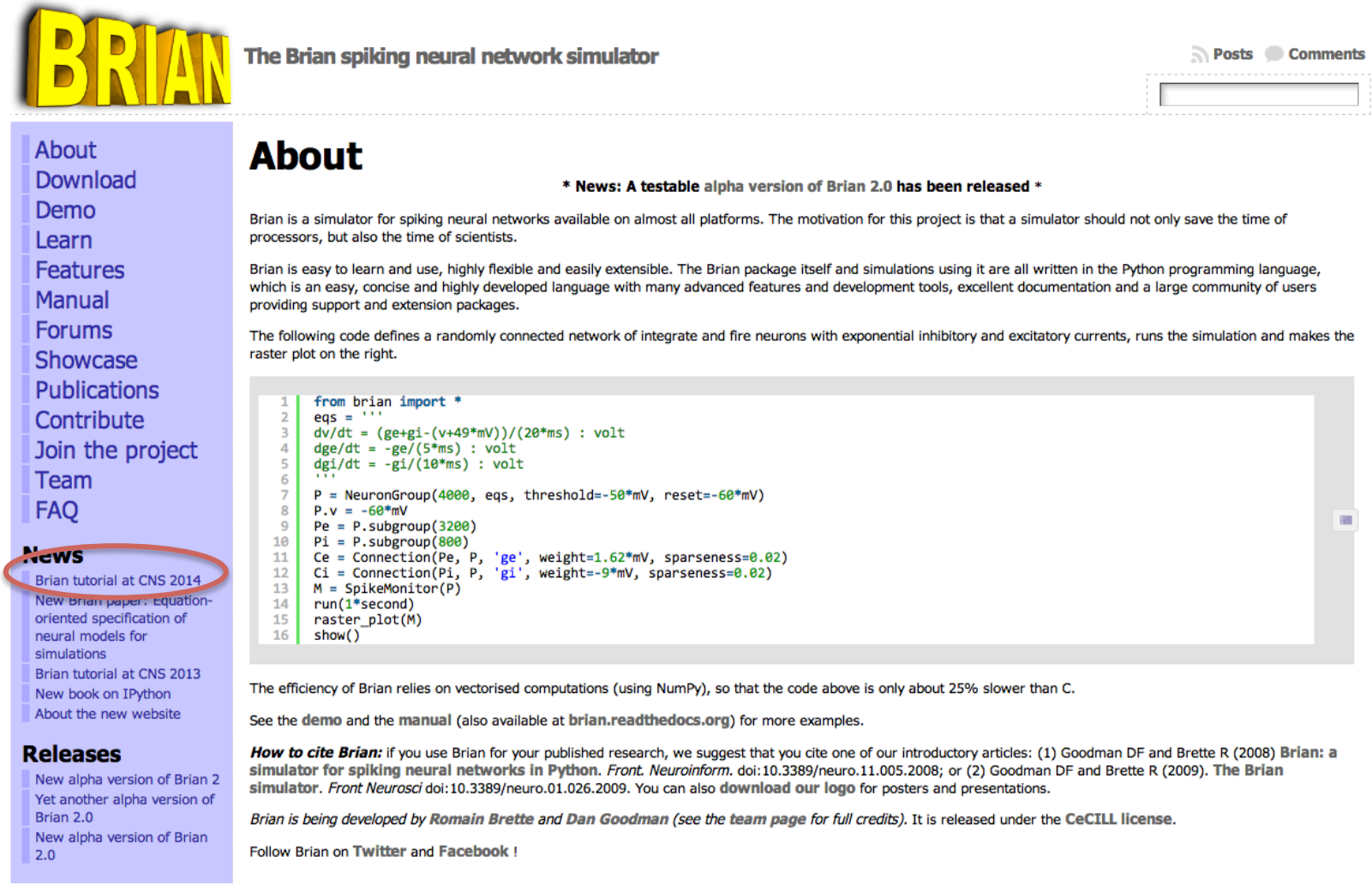

http://www.briansimulator.org

Romain Brette
Institut de la Vision, Paris
romain.brette@inserm.fr

Main developers of BRIAN 2: Dan Goodman & Marcel Stimberg

# Installing Brian 2

**briansimulator.org**



# The spirit of BRIAN

*"A simulator should not only save the time of processors, but also the time of scientists"*

*scientist*

*computer*

Writing code often takes more time than running it

Goals:

- Quick model coding
- Flexible

} models are defined by equations (rather than pre-defined)

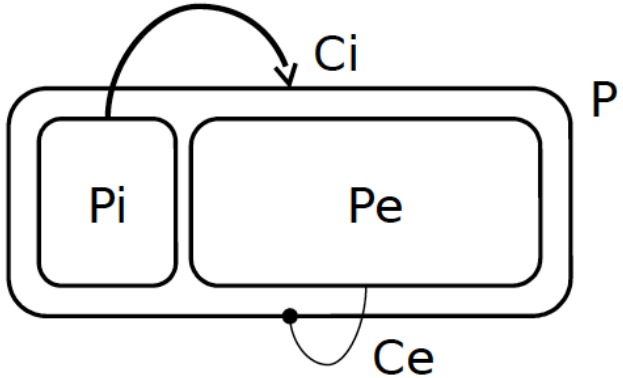
*Goodman, D. and R. Brette (2009). The Brian simulator. Front Neurosci doi:10.3389/neuro.01.026.2009.*

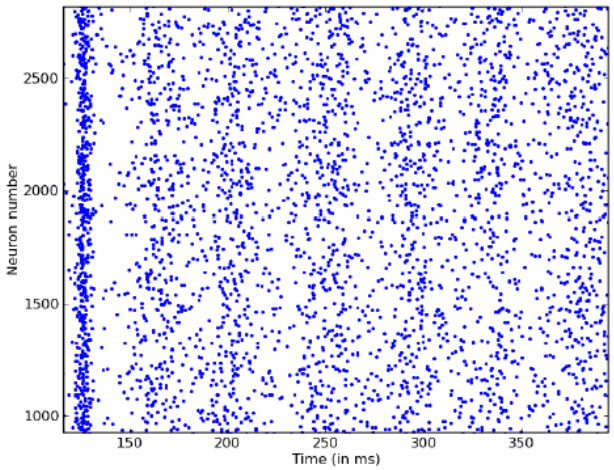
# An example

```
from brian2 import *

tau_m = 20*ms
tau_e = 5*ms
tau_i = 10*ms
V_th = -50*mV
E_L = -60*mV
I_const = 11*mV
eqs = '''
dv/dt = (-(v-E_L) + I_const + g_e + g_i)/tau_m : volt
dg_e/dt = -g_e/tau_e : volt
dg_i/dt = -g_i/tau_i : volt
'''
P = NeuronGroup(4000, model=eqs,
                threshold='v>V_th', reset='v=E_L')
P.v = 'E_L+10*mV*rand()'
Pe = P[:3200]
Pi = P[3200:]

w_e = 1.62*mV
w_i = 9*mV
Ce = Synapses(Pe, P, pre='g_e+=w_e')
Ci = Synapses(Pi, P, pre='g_i-=w_i')
Ce.connect(True, p=0.02)
Ci.connect(True, p=0.02)

M = SpikeMonitor(P)

run(1*second)
plot(M.t/ms, M.i, '.')
xlabel('Time (in ms)'); ylabel('Neuron number')
show()
```

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.

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{\mathrm{d}g_e}{\mathrm{d}t} = -g_e$$
$$\tau_i \frac{\mathrm{d}g_i}{\mathrm{d}t} = -g_i$$



# Standardization issues

Key issue: each simulator has its own language, how to communicate models?

Example 1: PyNEST (Python interface to NEST)

```
SetDefaults("iaf_psc_delta", {"C_m" : 20, "tau_m": 20,
                              "t_ref": 2, "E_L" : 0, "V_th" : 20})
nodes_ex = Create("iaf_psc_delta", NE)
```

*Component-based approach*:
you need to know what the components are exactly
you need to know parameter names
you need to know implicit units

# Standardization issues

Solution: *equation-oriented approach*

Example 2: NineML (Izhikevich model)

Issues:

- can't specify a full simulation (including initialization & stimulus)
- heavy!

```
<?xml version='1.0' encoding='UTF-8'?>
<NineML xmlns="http://nineml.org/9ML/0.1">
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://nineml.org/9ML/0.1 NineML\_v0.2.xsd">

  <ComponentClass name="izhikevichCellNew">

    <Parameter name="a" dimension="none"/>
    <Parameter name="c" dimension="voltage"/>
    <Parameter name="b" dimension="per_time"/>
    <Parameter name="d" dimension="voltage_per_time"/>
    <Parameter name="theta" dimension="voltage"/>

    <AnalogPort name="iSyn" mode="reduce" reduce_op="+" dimension="current"/>
    <AnalogPort name="U" mode="send" dimension="none"/>
    <AnalogPort name="V" mode="send" dimension="voltage"/>
    <EventPort name="spikeOutput" mode="send"/>


    <Dynamics>

        <StateVariable name="V" dimension="voltage"/>
        <StateVariable name="U" dimension="voltage_per_time"/>

        <Alias name="rv" dimension="none">
            <MathInline>V*U</MathInline>
        </Alias>

        <Regime name="subthresholdRegime">

          <TimeDerivative variable="U">
            <MathInline>a*(b*V - U)</MathInline>
          </TimeDerivative>

          <TimeDerivative variable="V">
            <MathInline>0.04*V*V + 5*V + 140.0 - U + iSyn</MathInline>
          </TimeDerivative>


          <OnCondition>
            <Trigger>
              <MathInline>V \> theta </MathInline>
            </Trigger>

            <StateAssignment variable="V" >
              <MathInline>c</MathInline>
            </StateAssignment>

            <StateAssignment variable="U" >
              <MathInline>U+d</MathInline>
            </StateAssignment>

            <EventOut port="spikeOutput" />

          </OnCondition>

        </Regime>
    </Dynamics>

  </ComponentClass>
</NineML>
```

# Standardization issues: the  way



Goal: to minimize « language entropy » = uncertainty about syntax and names, given the meaning

*Brette R (2012). On the design of script languages for neural simulation. Network 23(4), 150-156*

Key points:

1) There is already an accepted standard for models: math!
2) If you specify names, units and equations yourself, you don't need to know them in advance
3) Full expressivity requires a programming language

# Standardization issues: the BRIAN way

```
from brian2 import *

tau_m = 20*ms
tau_e = 5*ms
tau_i = 10*ms
V_th = -50*mV
E_L = -60*mV
I_const = 11*mV
eqs = '''
dv/dt = (-(v-E_L) + I_const + g_e + g_i)/tau_m : volt
dg_e/dt = -g_e/tau_e : volt
dg_i/dt = -g_i/tau_i : volt
'''
P = NeuronGroup(4000, model=eqs,
                threshold='v>V_th', reset='v=E_L')
P.v = 'E_L+10*mV*rand()'
Pe = P[:3200]
Pi = P[3200:]

w_e = 1.62*mV
w_i = 9*mV
Ce = Synapses(Pe, P, pre='g_e+=w_e')
Ci = Synapses(Pi, P, pre='g_i-=w_i')
Ce.connect(True, p=0.02)
Ci.connect(True, p=0.02)

M = SpikeMonitor(P)

run(1*second)
plot(M.t/ms, M.i, '.')
xlabel('Time (in ms)'); ylabel('Neuron number')
show()
```

# Standardization issues: the BRIAN way

```
S = Synapses(source_group, target_group,
             '''w : siemens
                dA_source/dt = -A_source/tau_source : siemens (event-driven)
                dA_target/dt = -A_target/tau_target : siemens (event-driven)''',
             pre='''g_post += w
                    A_source += deltaA_source
                    w = clip(w+A_target, 0*siemens, w_max)''',
             post='''A_target += deltaA_target
                     w = clip(w+A_source , 0*siemens, w_max)'''
             )
```

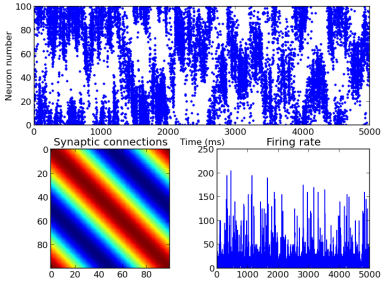
Stimberg M, Goodman DFM, Benichoux V, Brette R (2014).**Equation-oriented specification of neural models for simulations**. Frontiers Neuroinf, doi: 10.3389/fninf.2014.00006

# The future of BRIAN

BRIAN 2 relies on *code generation* to run on multiple targets

```
# Neurons
input = PoissonGroup(N, rates=F)
neurons = NeuronGroup(1, '''dv/dt = (g_e*(E_e-v_r)+E_l-v)/tau_m : volt
                            dg_e/dt = -g_e/tau_e : 1''',
                      threshold='v>vt', reset='v=v_r')
# Synaptic connections
S = Synapses(input, neurons,
             '''w:1
                dApre/dt = -Apre/taupre : 1 (event-driven)
                dApost/dt = -Apost/taupost : 1 (event-driven)''',
             pre='''ge += w
                    Apre += dApre
                    w = clip(w+Apost, 0, gmax)''',
             post='''Apost += dApost
                     w = clip(w+Apre, 0, gmax)''',
             connect=True)
S.w = 'rand()*gmax'
```

Simulation on PC, clusters, GPU




Interface with robots

Embedded simulation on Android smartphones


# Documentation

**briansimulator.org**


The Brian spiking neural network simulator

Posts Comments



## News

- Brian tutorial at CNS 2014
- New Brian paper: Equation-oriented specification of neural models for simulations
- Brian tutorial at CNS 2013
- New book on IPython
- About the new website

## Releases

- New alpha version of Brian 2
- Yet another alpha version of Brian 2.0
- New alpha version of Brian 2.0

# About

*** News: A testable alpha version of Brian 2.0 has been released ***

Brian is a simulator for spiking neural networks available on almost all platforms. The motivation for this project is that a simulator should not only save the time of processors, but also the time of scientists.

Brian is easy to learn and use, highly flexible and easily extensible. The Brian package itself and simulations using it are all written in the Python programming language, which is an easy, concise and highly developed language with many advanced features and development tools, excellent documentation and a large community of users providing support and extension packages.

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 brian 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=-50*mV, reset=-60*mV)
P.v = -60*mV
Pe = P.subgroup(3200)
Pi = P.subgroup(800)
Ce = Connection(Pe, P, 'ge', weight=1.62*mV, sparseness=0.02)
Ci = Connection(Pi, P, 'gi', weight=-9*mV, sparseness=0.02)
M = SpikeMonitor(P)
run(1*second)
raster_plot(M)
show()
```

The efficiency of Brian relies on vectorised computations (using NumPy), so that the code above is only about 25% slower than C.

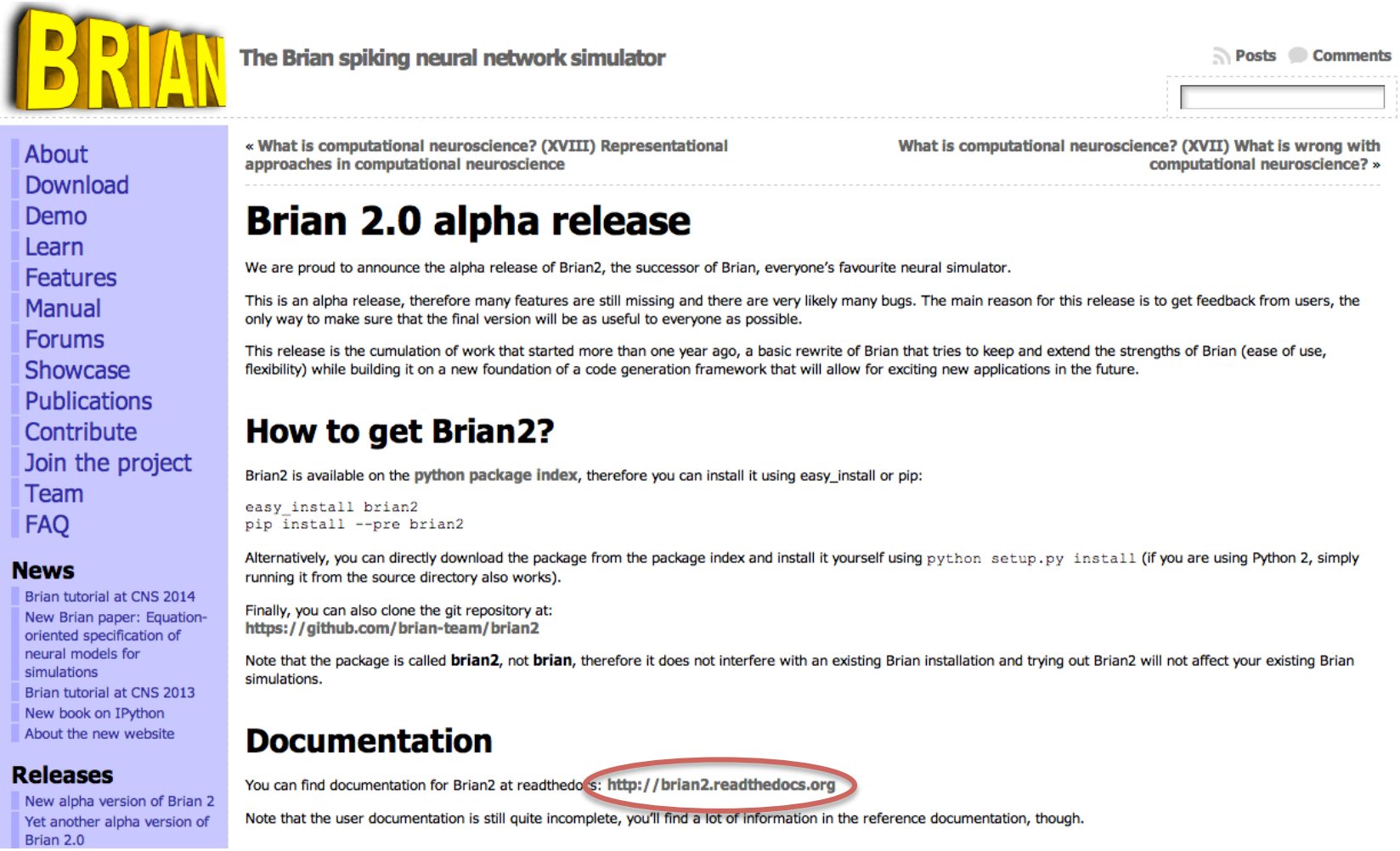
See the **demo** and the **manual** (also available at **brian.readthedocs.org**) for more examples.

***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 (2008) **Brian: a simulator for spiking neural networks in Python**. *Front. Neuroinform.* doi:10.3389/neuro.11.005.2008; or (2) Goodman DF and Brette R (2009). **The Brian simulator**. *Front Neurosci* doi:10.3389/neuro.01.026.2009. You can also **download our logo** for posters and presentations.

*Brian is being developed by **Romain Brette** and **Dan Goodman** (see the **team page** for full credits).* It is released under the **CeCILL license**.

Follow Brian on **Twitter** and **Facebook** !

# Documentation



BRIAN

The Brian spiking neural network simulator

Posts Comments




« What is computational neuroscience? (XVIII) Representational approaches in computational neuroscience

What is computational neuroscience? (XVII) What is wrong with computational neuroscience? »


## Brian 2.0 alpha release

We are proud to announce the alpha release of Brian2, the successor of Brian, everyone's favourite neural simulator.

This is an alpha release, therefore many features are still missing and there are very likely many bugs. The main reason for this release is to get feedback from users, the only way to make sure that the final version will be as useful to everyone as possible.

This release is the cumulation of work that started more than one year ago, a basic rewrite of Brian that tries to keep and extend the strengths of Brian (ease of use, flexibility) while building it on a new foundation of a code generation framework that will allow for exciting new applications in the future.

## How to get Brian2?

Brian2 is available on the **python package index**, therefore you can install it using easy_install or pip:

```
easy_install brian2
pip install --pre brian2
```

Alternatively, you can directly download the package from the package index and install it yourself using `python setup.py install` (if you are using Python 2, simply running it from the source directory also works).

Finally, you can also clone the git repository at:
**https://github.com/brian-team/brian2**

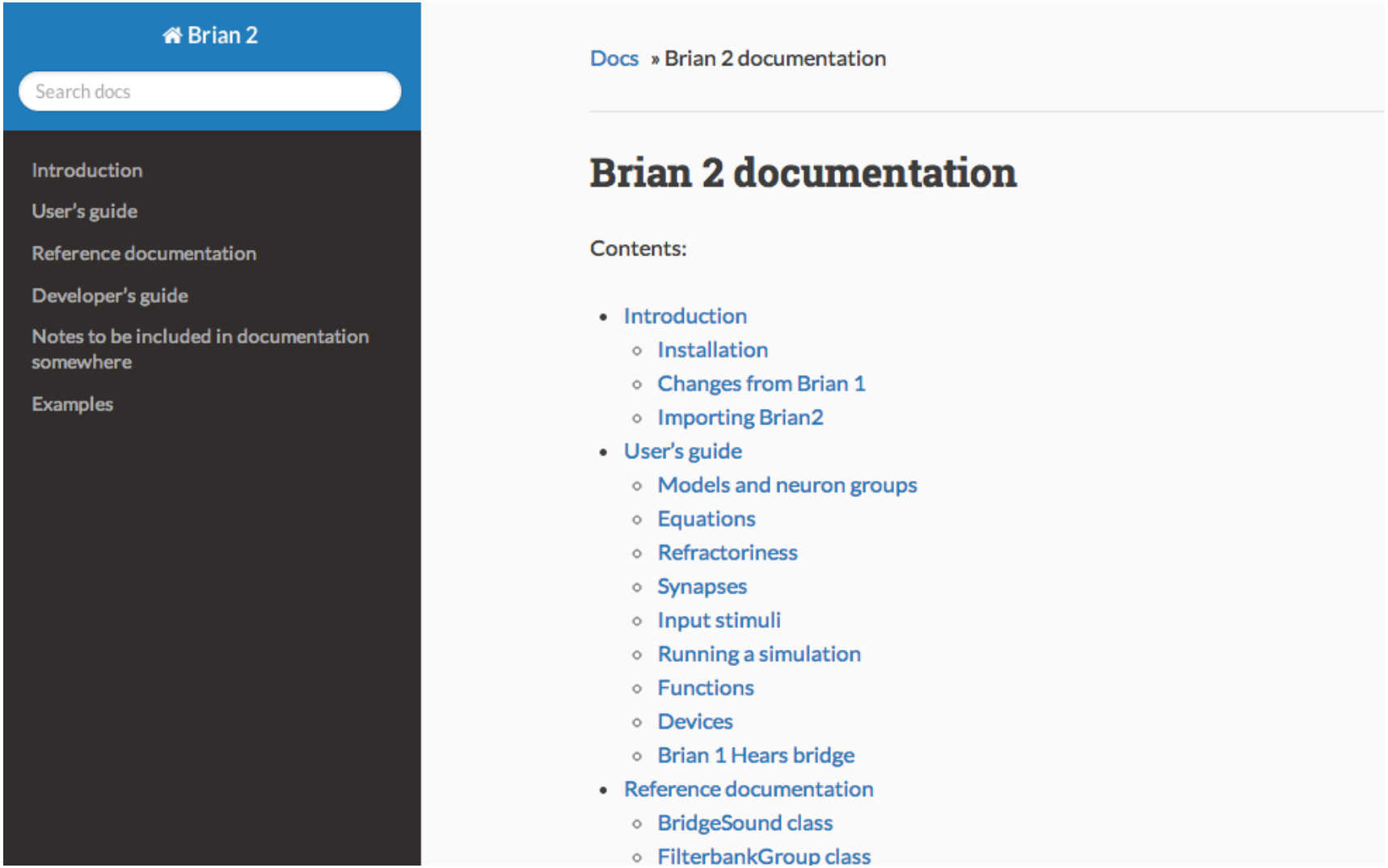
Note that the package is called **brian2**, not **brian**, therefore it does not interfere with an existing Brian installation and trying out Brian2 will not affect your existing Brian simulations.

## Documentation

You can find documentation for Brian2 at readthedocs: **http://brian2.readthedocs.org**

Note that the user documentation is still quite incomplete, you'll find a lot of information in the reference documentation, though.

# Documentation



# Forums


The Brian spiking neural network simulator

Posts Comments



## News

- Brian tutorial at CNS 2014
- New Brian paper: Equation-oriented specification of neural models for simulations
- Brian tutorial at CNS 2013
- New book on IPython
- About the new website

## Releases

- New alpha version of Brian 2
- Yet another alpha version of Brian 2.0
- New alpha version of Brian 2.0

# About

*** News: A testable alpha version of Brian 2.0 has been released ***

Brian is a simulator for spiking neural networks available on almost all platforms. The motivation for this project is that a simulator should not only save the time of processors, but also the time of scientists.

Brian is easy to learn and use, highly flexible and easily extensible. The Brian package itself and simulations using it are all written in the Python programming language, which is an easy, concise and highly developed language with many advanced features and development tools, excellent documentation and a large community of users providing support and extension packages.

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 brian 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=-50*mV, reset=-60*mV)
P.v = -60*mV
Pe = P.subgroup(3200)
Pi = P.subgroup(800)
Ce = Connection(Pe, P, 'ge', weight=1.62*mV, sparseness=0.02)
Ci = Connection(Pi, P, 'gi', weight=-9*mV, sparseness=0.02)
M = SpikeMonitor(P)
run(1*second)
raster_plot(M)
show()
```

The efficiency of Brian relies on vectorised computations (using NumPy), so that the code above is only about 25% slower than C.

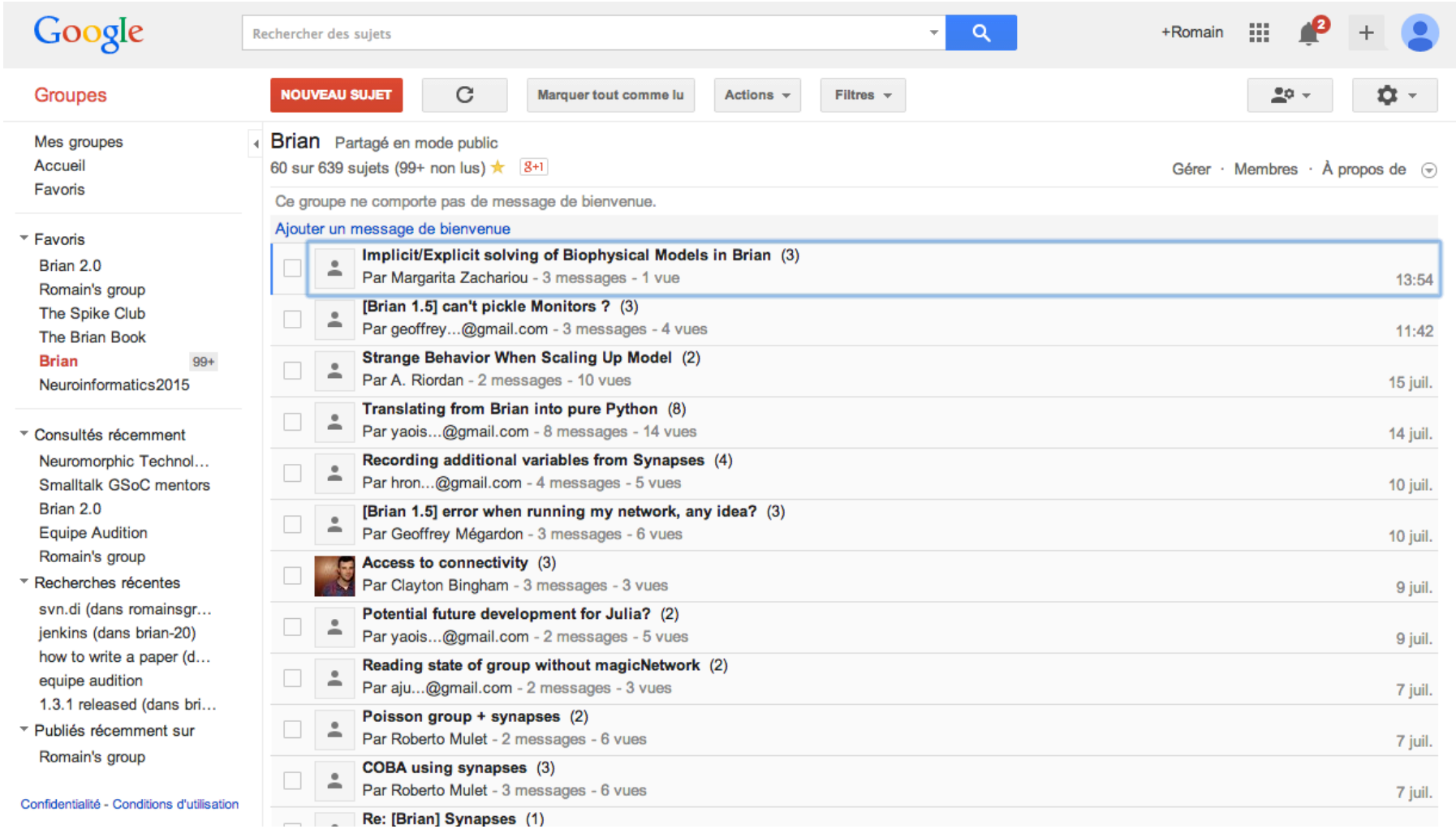
See the **demo** and the **manual** (also available at **brian.readthedocs.org**) for more examples.

***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 (2008) **Brian: a simulator for spiking neural networks in Python**. *Front. Neuroinform.* doi:10.3389/neuro.11.005.2008; or (2) Goodman DF and Brette R (2009). **The Brian simulator**. *Front Neurosci* doi:10.3389/neuro.01.026.2009. You can also **download our logo** for posters and presentations.

*Brian is being developed by **Romain Brette** and **Dan Goodman** (see the **team page** for full credits).* It is released under the **CeCILL license**.

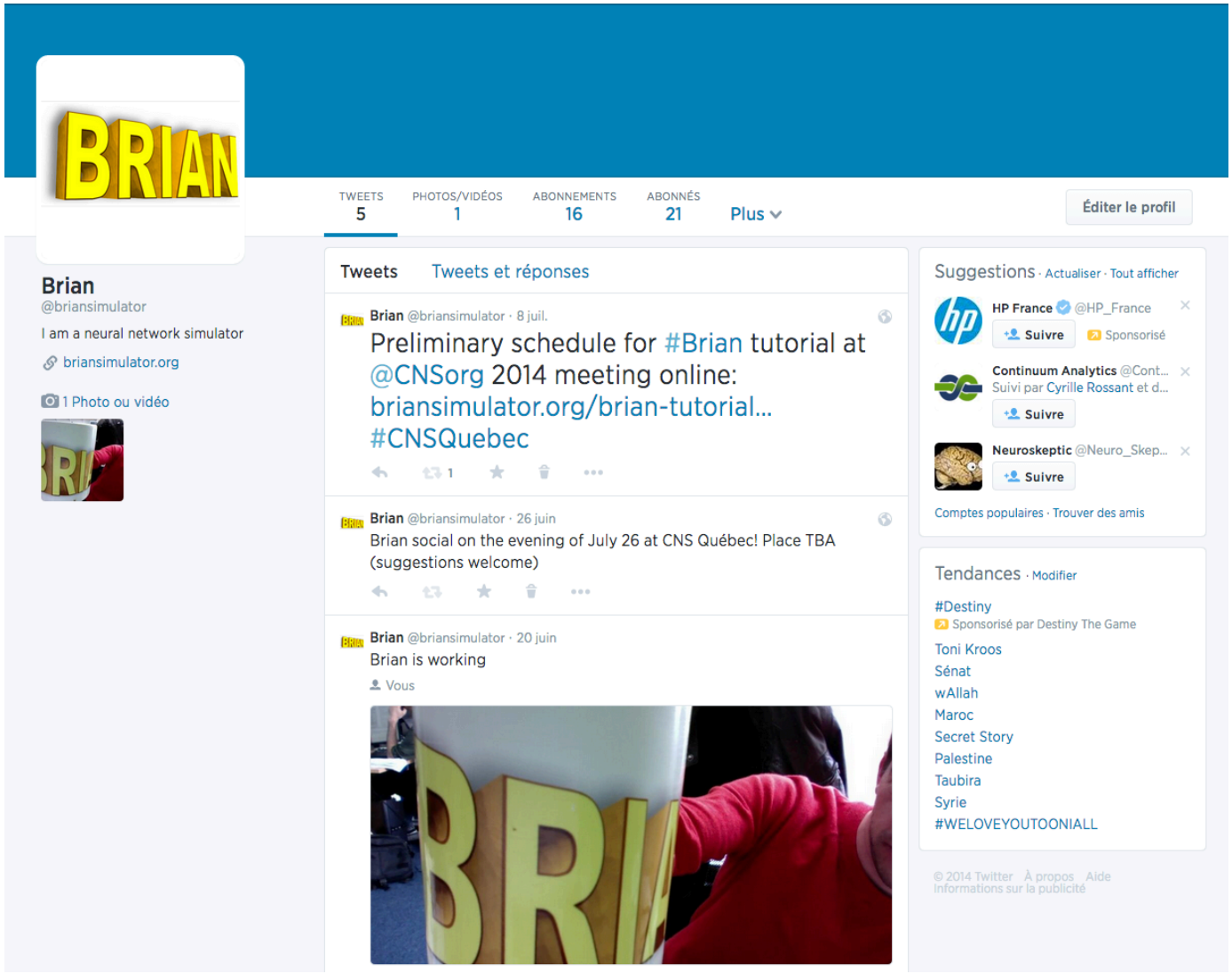
Follow Brian on **Twitter** and **Facebook** !

# Forums


Google
Rechercher des sujets
+Romain
Groupes
NOUVEAU SUJET
Marquer tout comme lu
Actions
Filtres
Mes groupes
Accueil
Favoris
Favoris
Brian 2.0
Romain's group
The Spike Club
The Brian Book
Brian 99+
Neuroinformatics2015
Consultés récemment
Neuromorphic Technol…
Smalltalk GSoC mentors
Brian 2.0
Equipe Audition
Romain's group
Recherches récentes
svn.di (dans romainsgr…
jenkins (dans brian-20)
how to write a paper (d…
equipe audition
1.3.1 released (dans bri…
Publiés récemment sur
Romain's group
Confidentialité - Conditions d'utilisation
Brian Partagé en mode public
60 sur 639 sujets (99+ non lus)
Gérer · Membres · À propos de
Ce groupe ne comporte pas de message de bienvenue.
Ajouter un message de bienvenue
Implicit/Explicit solving of Biophysical Models in Brian (3)
Par Margarita Zachariou - 3 messages - 1 vue 13:54
[Brian 1.5] can't pickle Monitors ? (3)
Par geoffrey...@gmail.com - 3 messages - 4 vues 11:42
Strange Behavior When Scaling Up Model (2)
Par A. Riordan - 2 messages - 10 vues 15 juil.
Translating from Brian into pure Python (8)
Par yaois...@gmail.com - 8 messages - 14 vues 14 juil.
Recording additional variables from Synapses (4)
Par hron...@gmail.com - 4 messages - 5 vues 10 juil.
[Brian 1.5] error when running my network, any idea? (3)
Par Geoffrey Mégardon - 3 messages - 6 vues 10 juil.
Access to connectivity (3)
Par Clayton Bingham - 3 messages - 3 vues 9 juil.
Potential future development for Julia? (2)
Par yaois...@gmail.com - 2 messages - 5 vues 9 juil.
Reading state of group without magicNetwork (2)
Par aju...@gmail.com - 2 messages - 3 vues 7 juil.
Poisson group + synapses (2)
Par Roberto Mulet - 2 messages - 6 vues 7 juil.
COBA using synapses (3)
Par Roberto Mulet - 3 messages - 6 vues 7 juil.
Re: [Brian] Synapses (1)

# Twitter

twitter.com/briansimulator



# Facebook

facebook.com/briansimulator


# user groups (BUG)

Groups for local communities
= a self-managed subdomain of briansimulator.org
(e.g. germany.briansimulator.org)
+ a mailing-list

- post announcements, organize local tutorials
- share code
- support in local language

Interested in creating a BUG? *email romain.brette@inserm.fr*

Any idea welcome!

# Today

| | | |
|---|---|---|
| Now – 10.10 | **Core concepts of Brian 2** | *Marcel Stimberg* |
| 10.10 – 10.40 | Coffee break | |
| 10.40 – 12.00 | **Tutorial** | *Pierre Yger* |
| 12.00 – 13.30 | Lunch | |
| 13.30 – 13.45 | **Going from Brian 1 to Brian 2** | *Marcel Stimberg* |
| 13.45 – 14.50 | **Advanced Brian 2** - Part I | *Dan Goodman* |
| 14.50 – 15.20 | Coffee break | |
| 15.20 – 16.30 | **Advanced Brian 2** - Part II | *Marcel Stimberg* |
| 19.00 – late | ***Brian social @ Pub Saint-Patrick (1200 rue Saint-Jean)*** | |