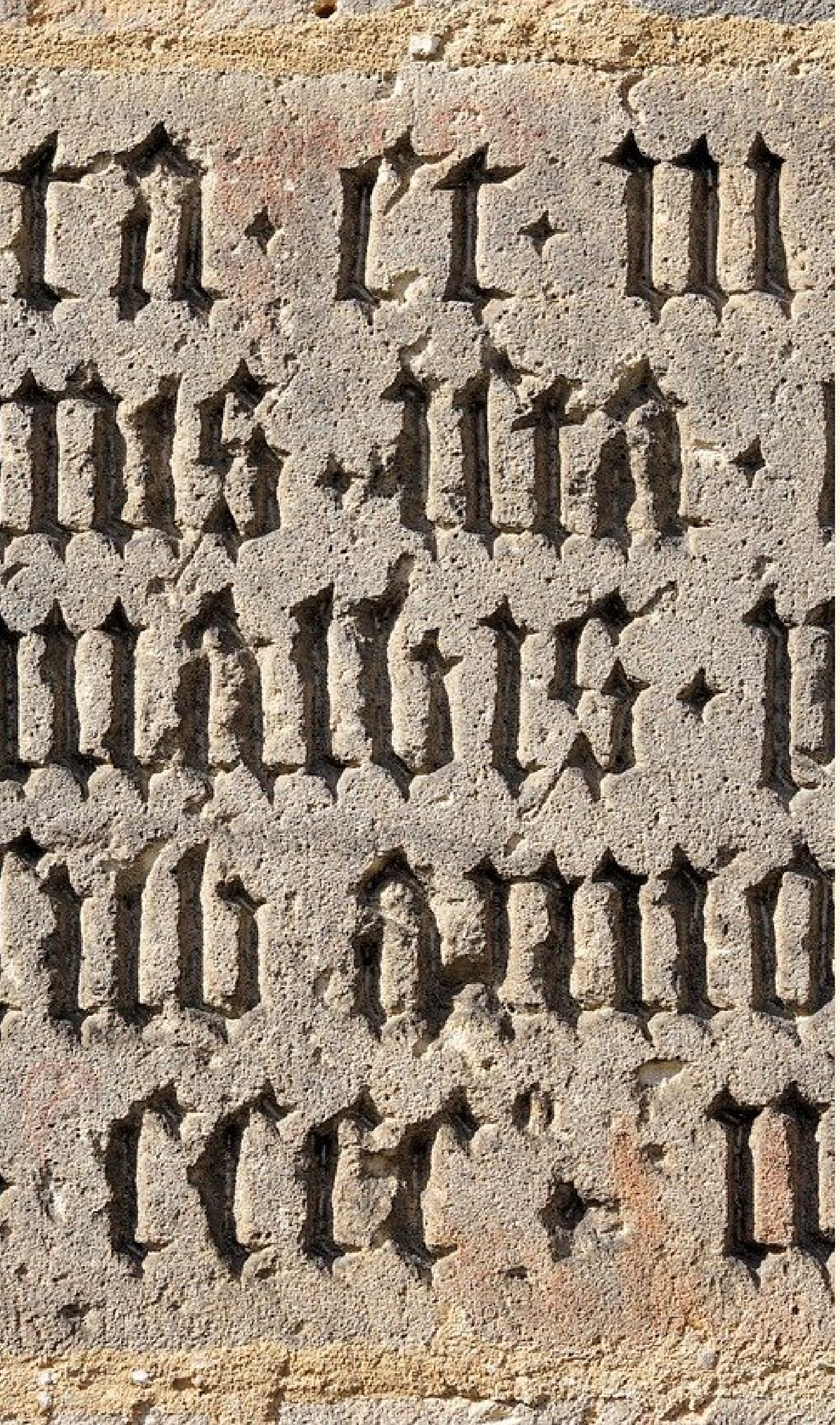
The Brian simulator

NeurotechEU autumn school

Marcel Stimberg

Research Engineer, Brian lead developer
(ISIR/Sorbonne Université)

Oct 19th 2023



Brian's history

Dan Goodman
(Imperial College London)

Romain Brette
(ISIR, Sorbonne Université)

- Started in **2007** at ENS Paris by Romain and Dan
- Widely used for **research and teaching**

> “ Defining new neural models should be no more difficult than writing down their equations ”

- No built-in library, tools to describe **"any" model**
- Big rewrite in 2014 (**code generation**)
- **Free-and-open-source** since the start

Brian's philosophy

Methods

Network model

We modelled the L2/3 network associated with a single barrel column—one of the most extensively studied cortical microcircuits[29,30]—as a network of leaky integrate-and-fire neurons[31]. The dynamics of each neuron were governed by:

$$\tau \frac{\mathrm{d}V_i}{\mathrm{d}t} = V_r - V(t) + R[I_i^{\mathrm{exc}}(t) + I_i^{\mathrm{inh}}(t) + I_i^{\mathrm{ext}}(t)] \quad (1)$$

where V is the membrane potential, V_r is the rest/reset potential, τ is the membrane time constant, R is the input resistance, I^{exc} and I^{inh} are excitatory and inhibitory synaptic currents, I^{ext} is a current representing sensory stimulus drive (for example, from layer 4 inputs), and i indexes the neurons in the network. When the membrane potential reaches the spiking threshold, V_{th}, a spike is emitted, the membrane potential is reset to V_r, and the dynamics of the neuron are frozen for a short refractory period, t_{ref}. The synaptic currents follow kick-and-decay dynamics:

$$\tau_{\mathrm{syn}} \frac{\mathrm{d}I_{\mathrm{i}}^{\mathrm{syn}}}{\mathrm{d}t} = -I_{\mathrm{i}}^{\mathrm{syn}} + \tau_{\mathrm{syn}} \sum_{\mathrm{j},k} w_{\mathrm{ij}} \delta(t - t_{\mathrm{i}}^{\mathrm{syn}} - t_{\mathrm{d}}) \quad (2)$$

where 'syn' denotes the type of synapse (either excitatory or inhibitor), τ_{syn} is the synaptic time constant, w_{ij} is a matrix of synaptic strengths from neuron j to neuron i, $t_{\mathrm{j}k}$ is the time of the kth spike of neuron j, and t_{d} is the spike transmission delay. The sum over j is over all neurons, while the sum over k is over all spikes from that neuron.

```
model = '''
dV/dt = (V_r- V + R*(IsynE + IsynI +
            Iswitch*Iext))/tau: 1 (unless refractory)
dIsynE/dt = -IsynE/tauSynE: 1
dIsynI/dt = -IsynI/tauSynI: 1
'''
group = NeuronGroup(Ntotal, model,
                    threshold='V>V_th', reset='V=Vr',
                    refractory=2*ms)
```

- Use the same language to describe models that we use in scientific publications: **equations**
- Built-in system for **physical units**
  Dimensional quantities are used everywhere, consistency is checked/enforced

```
>>> Cm = 200*pF; Rm = 100*Mohm
>>> tau = Cm * Rm
>>> print(tau)
20. ms
```

- Written in **Python** and making best use of it (overwritten operators for unit system, indexing, helpful error messages...)
- Friendly **community**, extensive **documentation** and helpful forums 😺

Paper on the left:
Peron et al. Recurrent interactions in local cortical circuits. Nature (2020) [10.1038/s41586-020-2062-x](10.1038/s41586-020-2062-x)

Brian's technology: code generation

(code simplified for clarity)

Brian code

```
group = NeuronGroup(1, 'dv/dt = -v / tau : 1')
```

"Abstract code" (internal pseudo-code representation)

```
v_new = v*exp(-dt/tau)
```

Brian's technology: code generation

C++ code snippet (once-per-group code)

```
const double value_1 = exp(-dt/tau);
```

C++ code snippet (once-per-neuron code)

```
double v = neurongroup_v[_idx];
const double v_new = value_1*v;
neurongroup_v[_idx] = v_new;
```

Brian's technology: code generation

Final C++ code after insertion in template

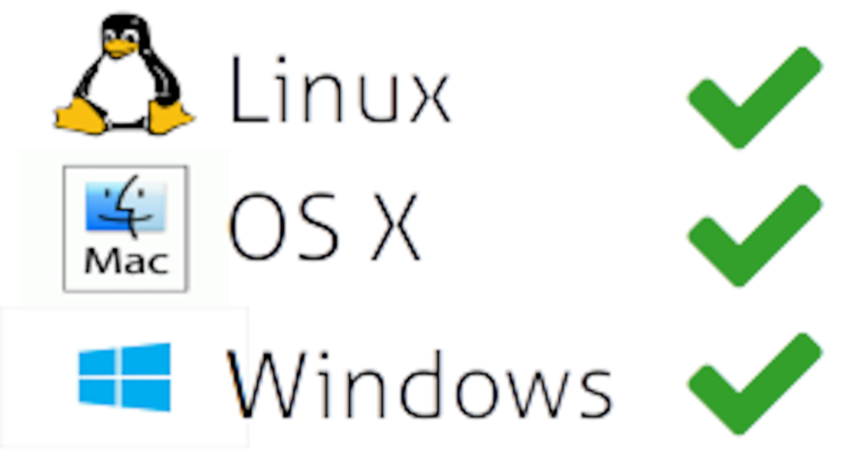
```
const double value_1 = exp(-dt/tau);

for(int _idx=0; _idx<_N; idx++)
{
  double v = neurongroup_v[_idx];
  const double v_new = value_1*v;
  neurongroup_v[_idx] = v_new;
}
```

Brian's technology: execution modes

- **Runtime mode**
  Simulation loop runs in **Python**, calls compiled blocks to do the work
  👍 Very flexible, can be combined with **arbitrary Python code**
  👎 **Overhead** from running Python, especially for small networks

- **Standalone mode**
  Everything (models + connection definitions, initializations) written out to a **standalone C++ project**
  Compiled binary executes full run and stores results to disk
  👍 **Fast**, no Python overhead
  👍 Can be **tailored** to other platforms
  👎 **Less flexible**, no Python interaction during run



Brian's domain

- **integrate-and-fire** models

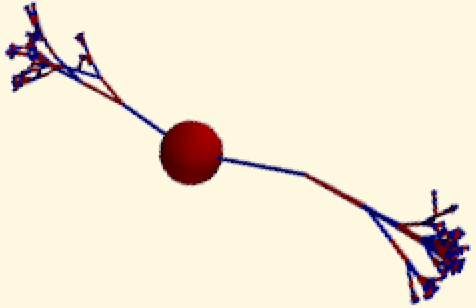
$$C_m dv/dt = g_L(E_L - v) + I$$

If $v > v_{th}$: emit spike and set $v \leftarrow v_{reset}$

- non-linear channel dynamics (e.g. **Hodgkin-Huxley**)

$$C_m dv/dt = (g_L \left(E_L - v\right) + m^3 n g_{Na} \left(E_{Na} - v\right) + n^4 \left(E_K - v\right)$$

(complex morphologies not yet very convenient to use)



Interactive jupyter notebook tutorial

We will discuss the following examples:

- Generating the f/I curve of a *leaky integrate-and-fire neuron*
  Example: IF_curve_LIF — Brian 2 documentation
- Simulating a sparsely connected *random network*
  Example: CUBA — Brian 2 documentation
- Modelling synapses with *spike-timing-dependent plasticity*
  Example: STDP — Brian 2 documentation

Example: neuron model

Leaky integrate-and-fire neuron with stimulus current

$$C\frac{dV}{dt} = I_{\text{stim}} + g_L(V_{\text{rest}} - V)$$

If $V(t) > V_{\text{threshold}}$ → emit spike and set $V(t) = V_{\text{reset}}$

```
N_neurons = 100
C = 200*pF
g_L = 10*nS
V_rest = -70*mV
V_threshold = -50*mV
V_reset = V_rest
eqs = 'dV/dt = (I_stim + g_L*(V_rest - V)) / C : volt'
group = NeuronGroup(N_neurons, eqs,
                    threshold='V > V_threshold', reset='V = V_reset')
```

Example: synapse model

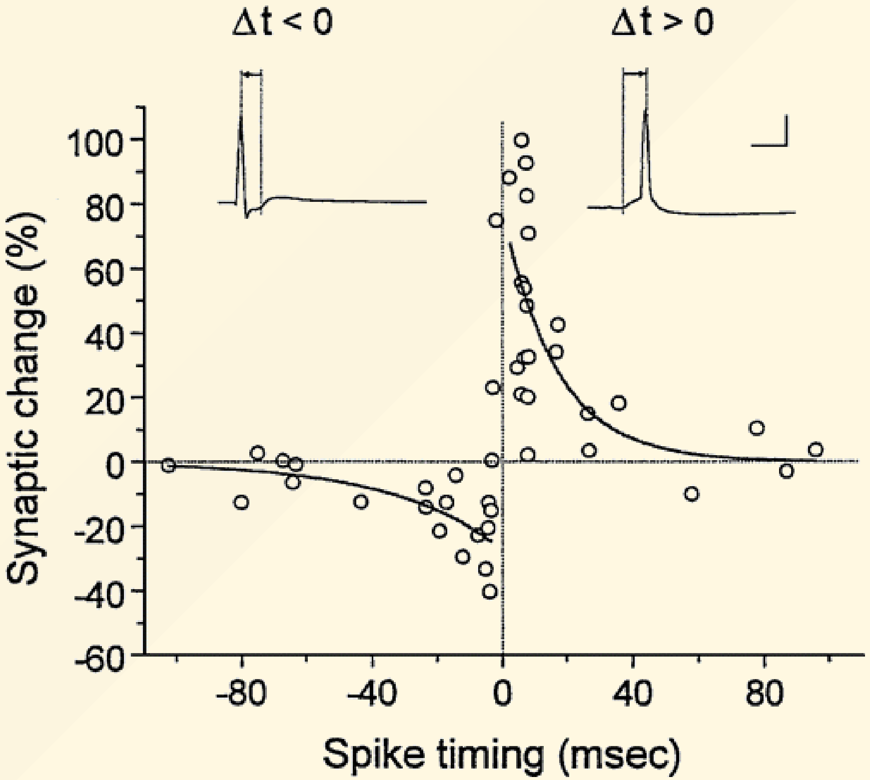
current-based synapse (simple exponential)

- In between spikes: $\frac{dI_{\text{syn}}}{dt} = \frac{-I_{\text{syn}}}{\tau_{\text{syn}}}$
- when a spike arrives, increase I_{syn} by 0.1 nA

```
eqs = '''dV/dt = (I_syn + g_L*(V_rest - V)) / C : volt
         dI_syn/dt = -I_syn/tau_syn : amp'''
group = NeuronGroup(N_neurons, eqs,
                    threshold='V > V_threshold', reset='V = V_reset')
synapses = Synapses(…, neurons, on_pre='I_syn += 0.1*nA')
synapses.connect(…)
```

Example: synaptic plasticity

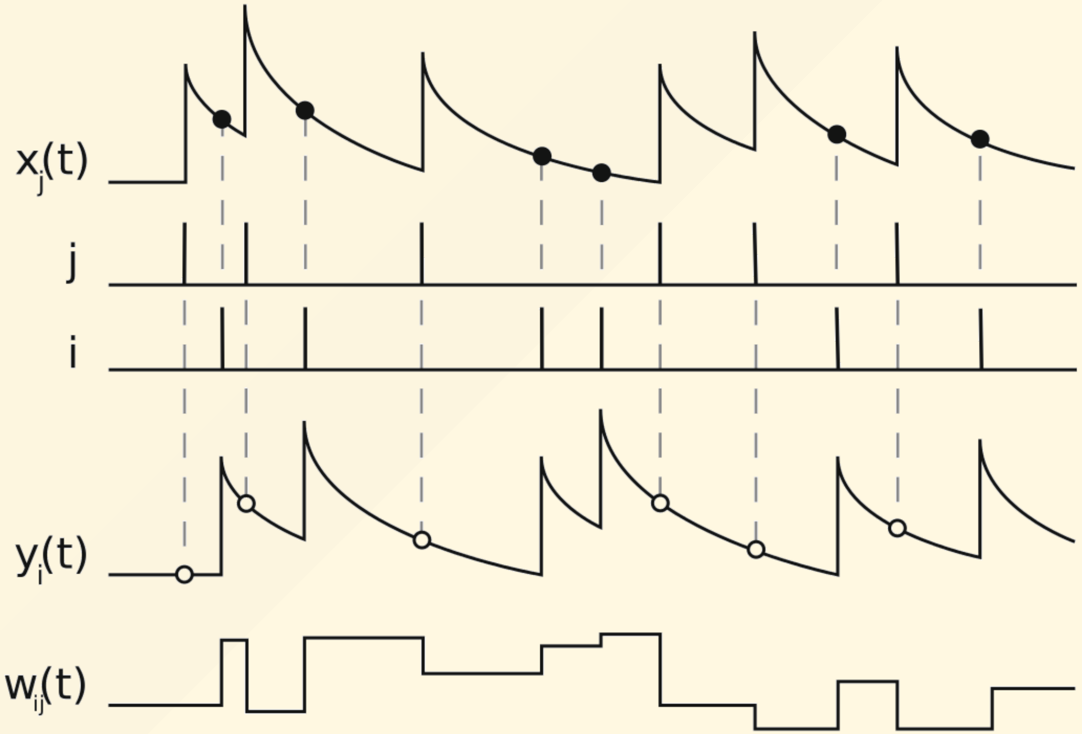
Spike-timing dependent plasticity



Bi and Poo. Synaptic Modifications in Cultured Hippocampal Neurons: [...]. J Neurosci (1998) 10.1523/JNEUROSCI.18-24-10464

Example: synaptic plasticity

Online implementation



Morrison et al. Phenomenological models of synaptic plasticity based on spike timing. Biol Cybern (2008) 10.1007/s00422-008-0233-1

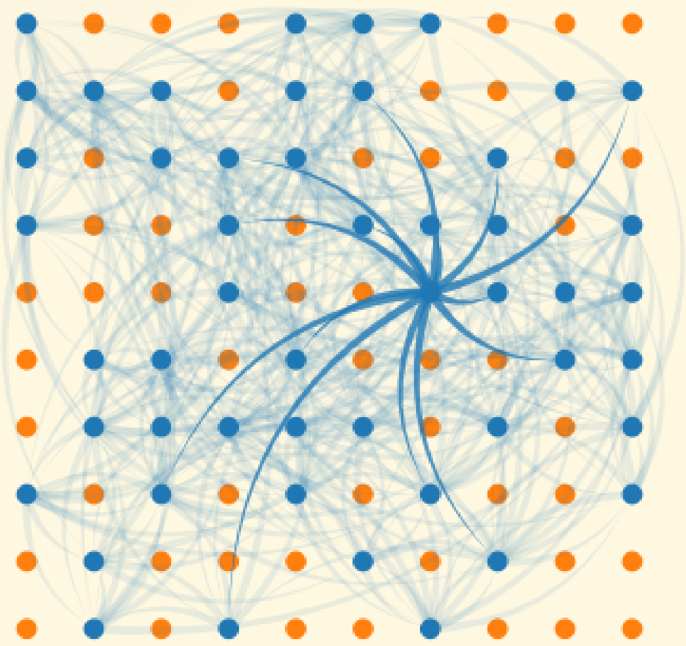
Showcase example 1

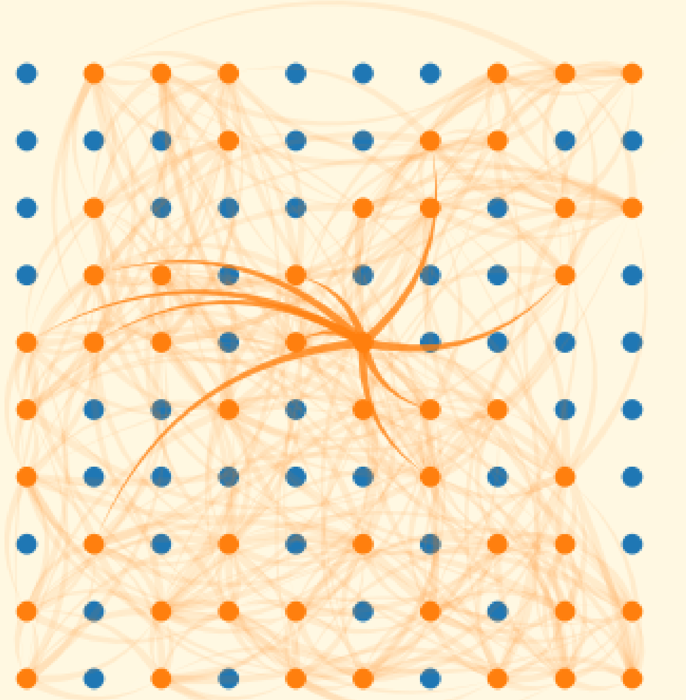
Synaptic connections

- Expressive **connection syntax**
- arbitrary **labels/properties** in models
  → **descriptive** connection declaration

```
neuron = NeuronGroup(100, model="""# ... actual neuron model
                     neuron_type : integer (constant)
                     x : meter (constant)
                     y : meter (constant)""")
neuron.neuron_type = 'int(rand() * 2)'  # two types
neuron.x = '(i % 10) * 50*um'  # Neurons arranged in a grid
neuron.y = '(i // 10) * 50*um'

synapses = Synapses(neuron, neuron)  # do-nothing synapse just for illustration
synapses.connect('neuron_type_pre == neuron_type_post',  # only if same type
                 # Probability = negative exponential of distance
                 p='1.5*exp(-sqrt(((x_pre - x_post)**2 +'
                   '              (y_pre - y_post)**2))/(100*um))')
```





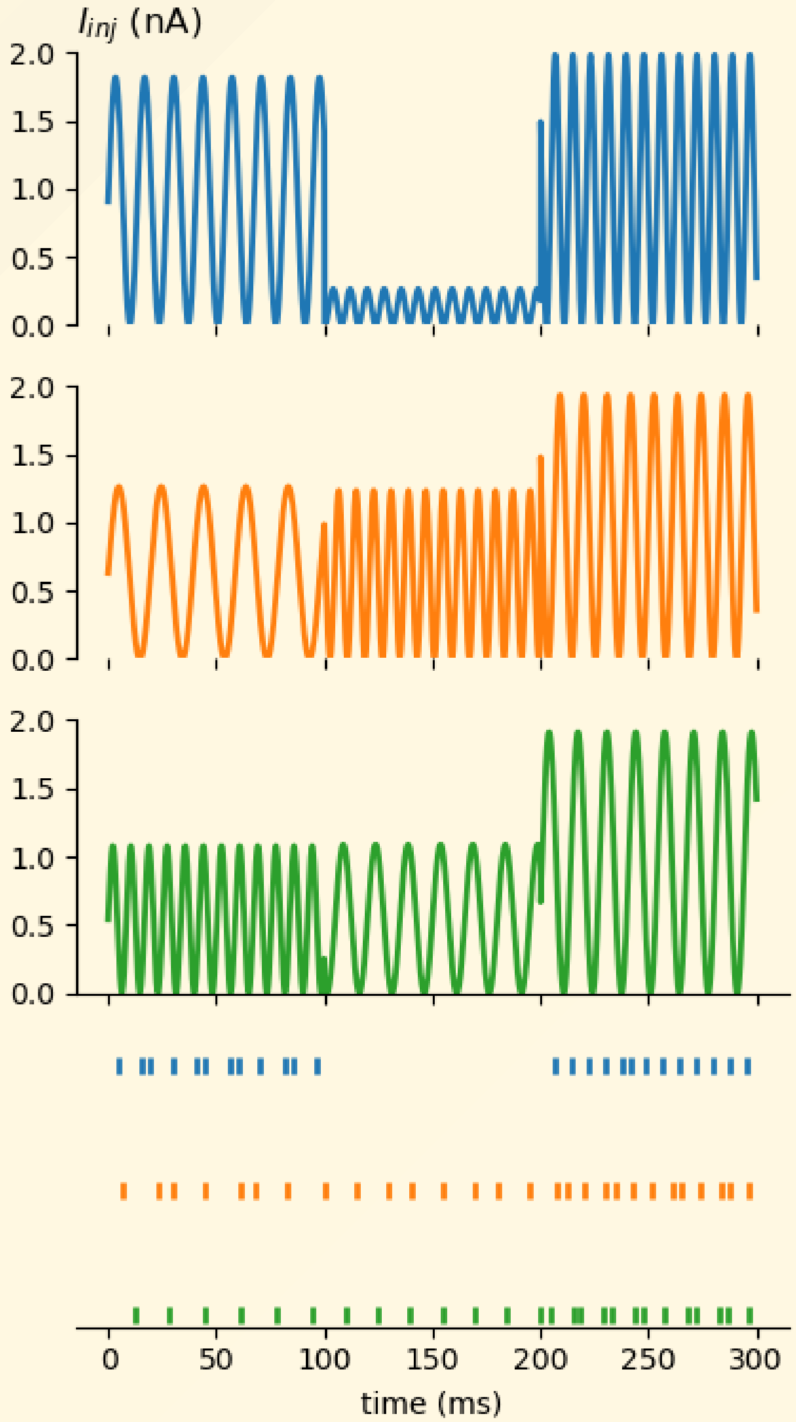
Full example code

Showcase example 2

Stimulus protocols

- Flexible description of e.g. **stimuli**
- parameters can **change over time**

```
g_L = 10*nS; E_L = -70*mV; C_m = 200*pF
neuron = NeuronGroup(3, model="""
                     dv/dt = (g_L*(E_L - v) + I_inj)/C_m : volt
                     I_inj = amplitude * (0.5 + 0.5 * sin(2*pi*freq*t)) : amp
                     amplitude : amp
                     freq : Hz""",
                     threshold='v > -40*mV', reset='v = E_L')
neuron.run_regularly("""amplitude = rand() * 2*nA
                        freq = 50*Hz + 100*Hz*rand()""",
                     dt=100*ms)  # change injected current every 100ms
```



[Full example code]

Where to learn more about Brian

Website: briansimulator.org

Documentation: brian2.readthedocs.io

Discussion forum: brian.discourse.group

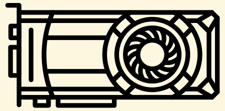
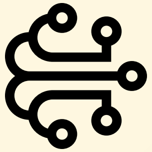
Development repository: github.com/brian-team/brian2

Stimberg, M., Brette, R., & Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. ELife, 8, e47314. 10.7554/eLife.47314

Stimberg, M., Goodman, D. F. M., Brette, R., & Pittà, M. D. (2019). Modeling Neuron–Glia Interactions with the Brian 2 Simulator. In M. De Pittà & H. Berry (Eds.), Computational Glioscience (pp. 471–505). Springer International Publishing. 10.1007/978-3-030-00817-8_18

Stimberg, M., Goodman, D. F. M., Benichoux, V., & Brette, R. (2014). Equation-oriented specification of neural models for simulations. Frontiers in Neuroinformatics, 8. 10.3389/fninf.2014.00006

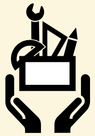
The Brian ecosystem

Brian2GeNN (code generation for GeNN)
brian2genn.readthedocs.io
With: *T. Nowotny*

Brian2CUDA (code generation for CUDA)
github.com/brian-team/brian2cuda/
D. Alevi, M. Augustin

Brian2Lava (code generation for Loihi 2) 🚧
gitlab.com/brian2lava/brian2lava
Tetzlaff lab



dendrify
(simplified multi-compartmental models)
M. Pagkalos, S. Chavlis

brian2tools
(visualization, NeuroML import/export)
With: *Vigneswaran C, K. Kumar, D. Krzemiński*

brian2modelfitting
(fitting models to experimental data)
With: *R. Brette, A. Teska, A.L. Kapetanović*