

Brian 2

An Intuitive & Efficient Neural Simulator

Marcel Stimberg¹, Romain Brette¹, Dan FM Goodman²

¹Sorbonne Université, INSERM, CNRS, Institut de la Vision, Paris, France

²Department of Electrical and Electronic Engineering, Imperial College London, UK

Overview

- Introduction
- Design implementation
- Case study 1: Pyloric network
- Case study 2: Ocular model
- Case study 3: Threshold finding
- Case study 4: Real-time audio
- Drawbacks

Introduction

Want your simulator to attract as many neuro-researchers as possible.

Performance

Is it *benefiting* from:

- Vectorization techniques
- Pre-compiled models

Flexibility

Is it *easy* to define:

- Non-standard models
- Arbitrary protocols

Continued...

Brian 2 solves this trade-off.

➤ Flexibility: User-written Python script

- No limit on the experiment structure

➤ Performance: Code generation

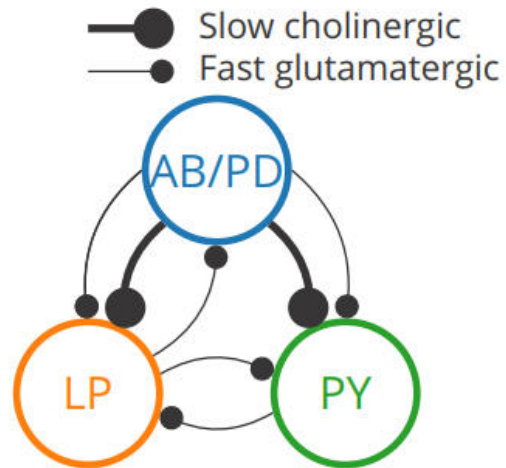
- Transform high-level model into low-level executable code

Design Implementation

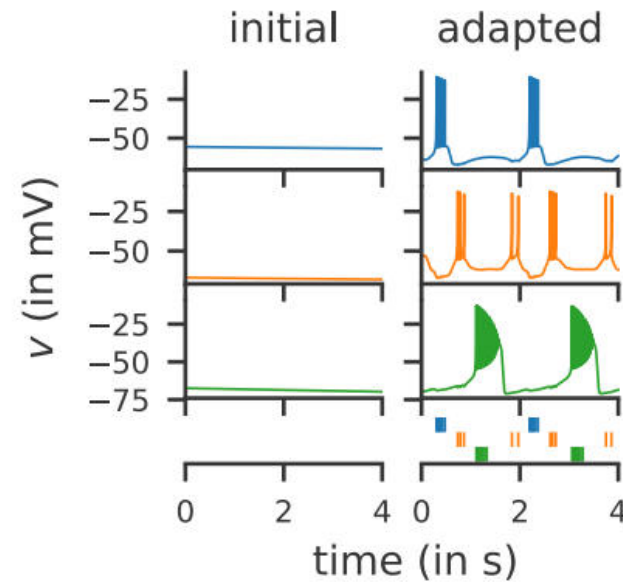
1. Non-standard models
 - Mathematical equations should be explicitly written.
2. Complete computational experiment
 - Models must interact with general control flow.
3. Efficient code generation
 - Generated code can integrate into the simulation flow.
4. Extensibility of code
 - Code can be extended either at high- or low-level.

Case Study 1: Pyloric Network

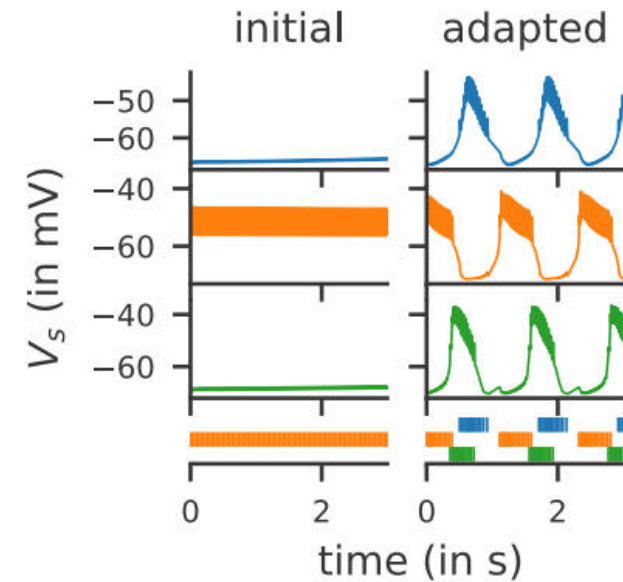
(a)



(b)



(c)



Generate a stereotypical triphasic motor pattern.

- (a) Circuit schematic
- (b) Simulated neuron activity
- (c) Simulated neuron activity (*biologically detailed*)

Continued...

```
1 from brian2 import *
2 defaultclock.dt = 0.01*ms;
3 Delta_T = 17.5*mV ; v_T = -40*mV ; tau = 2*ms ; tau_adapt = .02*second
4 tau_Ca = 150*ms ; tau_x = 2*second ; v_r = -68*mV ; tau_z = 5*second
5 a = 1/Delta_T**3 ; b = 3/Delta_T**2 ; c = 1.2*nA ; d = 2.5*nA/Delta_T**2
6 C = 60*pF ; S = 2*nA/Delta_T ; G = 28.5*nS
7 eqs = '''
8 dv/dt = (Delta_T*g*(-a*(v - v_T)**3 + b*(v - v_T)**2) + w - x - I_fast - I_slow)/C : volt
9 dw/dt = (c - d*(v - v_T)**2 - w)/tau : amp
10 dx/dt = (s*(v - v_r) - x)/tau_x : amp
11 s = S*(1 - tanh(z)) : siemens
12 g = G*(1 + tanh(z)) : siemens
13 dCa/dt = -Ca/tau_Ca : 1
14 dz/dt = tanh(Ca - Ca_target)/tau_z : 1
15 I_fast : amp
16 I_slow : amp
17 Ca_target : 1 (constant)
18 label : integer (constant)
19 '''
20 ABPD, LP, PY = 0, 1, 2
21 circuit = NeuronGroup(3, eqs, threshold='v>-20*mV', refractory='v>-20*mV', reset='Ca += 0.1',
22 method='rk2')
23 circuit.label = [ABPD, LP, PY]
24 circuit.v = v_r
25 circuit.w = '-5*nA*rand()'
26 circuit.z = 'rand()*0.2 - 0.1'
27 circuit.Ca_target = [0.048, 0.0384, 0.06]
28
29 s_fast = 0.2/mV; V_fast = -50*mV; E_syn = -75*mV
30 eqs_fast = '''
31 g_fast : siemens (constant)
32 I_fast_post = g_fast*(v_post - E_syn)/(1+exp(s_fast*(V_fast-v_pre))) : amp (summed)
33 '''
34 fast_synapses = Synapses(circuit, circuit, model=eqs_fast)
35 fast_synapses.connect('label_pre != label_post and not (label_pre == PY and label_post == ABPD)')
36 fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
37 fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
38 fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
39 fast_synapses.g_fast['label_pre == LP and label_post == PY'] = 0.02*uS
40 fast_synapses.g_fast['label_pre == PY and label_post == LP'] = 0.005*uS
41
42 s_slow = 1/mV; V_slow = -55*mV; k_1 = 1/ms
43 eqs_slow = '''
44 k_2 : 1/second (constant)
45 g_slow : siemens (constant)
46 I_slow_post = g_slow*m_slow*(v_post-E_syn) : amp (summed)
47 dm_slow/dt = k_1*(1-m_slow)/(1+exp(s_slow*(V_slow-v_pre))) - k_2*m_slow : 1 (clock-driven)
48 '''
49 slow_synapses = Synapses(circuit, circuit, model=eqs_slow, method='exact')
50 slow_synapses.connect('label_pre == ABPD and label_post != ABPD')
51 slow_synapses.g_slow['label_post == LP'] = 0.025*uS
52 slow_synapses.k_2['label_post == LP'] = 0.03/ms
53 slow_synapses.g_slow['label_post == PY'] = 0.015*uS
54 slow_synapses.k_2['label_post == PY'] = 0.008/ms
55
56 run(59.5*second)
```

```
8 dv/dt = (Delta_T*g*(-a*(v - v_T)**3 + b*(v - v_T)**2) + w - x - I_fast - I_slow)/C : volt
9 dw/dt = (c - d*(v - v_T)**2 - w)/tau : amp
10 dx/dt = (s*(v - v_r) - x)/tau_x : amp
```

Neuron model is written in differential equation.

```
13 dCa/dt = -Ca/tau_Ca : 1
```

Calcium trace decays.

```
21 circuit = NeuronGroup(3, eqs, threshold='v>-20*mV', refractory='v>-20*mV', reset='Ca += 0.1',
22 method='rk2')
```

Calcium trace increases at each spike.

```
11 s = S*(1 - tanh(z)) : siemens
12 g = G*(1 + tanh(z)) : siemens

14 dz/dt = tanh(Ca - Ca_target)/tau_z : 1
```

Conductances are regulated by trace difference.

Continued...

```
1 from brian2 import *
2 defaultclock.dt = 0.01*ms;
3 Delta_T = 17.5*mV ; v_T = -40*mV ; tau = 2*ms ; tau_adapt = .02*second
4 tau_Ca = 150*ms ; tau_x = 2*second ; v_r = -68*mV ; tau_z = 5*second
5 a = 1/Delta_T**3 ; b = 3/Delta_T**2 ; c = 1.2*nA ; d = 2.5*nA/Delta_T**2
6 C = 60*pF ; S = 2*nA/Delta_T ; G = 28.5*nS
7 eqs = '''
8 dv/dt = (Delta_T*g*(-a*(v - v_T)**3 + b*(v - v_T)**2) + w - x - I_fast - I_slow)/C : volt
9 dw/dt = (c - d*(v - v_T)**2 - w)/tau : amp
10 dx/dt = (s*(v - v_r) - x)/tau_x : amp
11 s = S*(1 - tanh(z)) : siemens
12 g = G*(1 + tanh(z)) : siemens
13 dCa/dt = -Ca/tau_Ca : 1
14 dz/dt = tanh(Ca - Ca_target)/tau_z : 1
15 I_fast : amp
16 I_slow : amp
17 Ca_target : 1 (constant)
18 label : integer (constant)
19 '''
20 ABPD, LP, PY = 0, 1, 2
21 circuit = NeuronGroup(3, eqs, threshold='v>-20*mV', refractory='v>-20*mV', reset='Ca += 0.1',
22 method='rk2')
23 circuit.label = [ABPD, LP, PY]
24 circuit.v = v_r
25 circuit.w = '-5*nA*rand()'
26 circuit.z = 'rand()*0.2 - 0.1'
27 circuit.Ca_target = [0.048, 0.0384, 0.06]
28
29 s_fast = 0.2/mV; V_fast = -50*mV; E_syn = -75*mV
30 eqs_fast = '''
31 g_fast : siemens (constant)
32 I_fast_post = g_fast*(v_post - E_syn)/(1+exp(s_fast*(V_fast-v_pre))) : amp (summed)
33 '''
34 fast_synapses = Synapses(circuit, circuit, model=eqs_fast)
35
36 fast_synapses.connect('label_pre != label_post and not (label_pre == PY and label_post == ABPD)')
37 fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
38 fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
39 fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
40 fast_synapses.g_fast['label_pre == LP and label_post == PY'] = 0.02*uS
41 fast_synapses.g_fast['label_pre == PY and label_post == LP'] = 0.005*uS
42
43 s_slow = 1/mV; V_slow = -55*mV; k_1 = 1/ms
44 eqs_slow = '''
45 k_2 : 1/second (constant)
46 g_slow : siemens (constant)
47 I_slow_post = g_slow*m_slow*(v_post-E_syn) : amp (summed)
48 dm_slow/dt = k_1*(1-m_slow)/(1+exp(s_slow*(V_slow-v_pre))) - k_2*m_slow : 1 (clock-driven)
49 '''
50 slow_synapses = Synapses(circuit, circuit, model=eqs_slow, method='exact')
51 slow_synapses.connect('label_pre == ABPD and label_post != ABPD')
52 slow_synapses.g_slow['label_post == LP'] = 0.025*uS
53 slow_synapses.k_2['label_post == LP'] = 0.03/ms
54 slow_synapses.g_slow['label_post == PY'] = 0.015*uS
55 slow_synapses.k_2['label_post == PY'] = 0.008/ms
56
57 run(59.5*second)
```

```
29 s_fast = 0.2/mV; V_fast = -50*mV; E_syn = -75*mV
30 eqs_fast = '''
31 g_fast : siemens (constant)
32 I_fast_post = g_fast*(v_post - E_syn)/(1+exp(s_fast*(V_fast-v_pre))) : amp (summed)
33 '''
34 fast_synapses = Synapses(circuit, circuit, model=eqs_fast)
```

Nonlinear (graded) fast synapse

```
42 s_slow = 1/mV; V_slow = -55*mV; k_1 = 1/ms
43 eqs_slow = '''
44 k_2 : 1/second (constant)
45 g_slow : siemens (constant)
46 I_slow_post = g_slow*m_slow*(v_post-E_syn) : amp (summed)
47 dm_slow/dt = k_1*(1-m_slow)/(1+exp(s_slow*(V_slow-v_pre))) - k_2*m_slow : 1 (clock-driven)
48 '''
49 slow_synapses = Synapses(circuit, circuit, model=eqs_slow, method='exact')
```

Nonlinear (graded) slow synapse

Continued...

```

1  from brian2 import *
2  defaultclock.dt = 0.01*ms;
3  Delta_T = 17.5*mV; v_T = -40*mV; tau = 2*ms; tau_adapt = .02*second
4  tau_Ca = 150*ms; tau_x = 2*second; v_r = -68*mV; tau_z = 5*second
5  a = 1/Delta_T**3; b = 3/Delta_T**2; c = 1.2*nA; d = 2.5*nA/Delta_T**2
6  C = 60*pF; S = 2*nA/Delta_T; G = 28.5*nS
7  eqs = '''
8  dv/dt = (Delta_T*g*(-a*(v - v_T)**3 + b*(v - v_T)**2) + w - x - I_fast - I_slow)/C : volt
9  dw/dt = (c - d*(v - v_T)**2 - w)/tau : amp
10 dx/dt = (s*(v - v_r) - x)/tau_x : amp
11 s = S*(1 - tanh(z)) : siemens
12 g = G*(1 + tanh(z)) : siemens
13 dCa/dt = -Ca/tau_Ca : 1
14 dz/dt = tanh(Ca - Ca_target)/tau_z : 1
15 I_fast : amp
16 I_slow : amp
17 Ca_target : 1 (constant)
18 label : integer (constant)
19 '''
20 ABPD, LP, PY = 0, 1, 2
21 circuit = NeuronGroup(3, eqs, threshold='v>-20*mV', refractory='v>-20*mV', reset='Ca += 0.1',
22                      method='rk2')
23 circuit.label = [ABPD, LP, PY]
24 circuit.v = v_r
25 circuit.w = '-5*nA*rand()'
26 circuit.z = 'rand()*0.2 - 0.1'
27 circuit.Ca_target = [0.048, 0.0384, 0.06]
28
29 s_fast = 0.2/mV; V_fast = -50*mV; E_syn = -75*mV
30 eqs_fast = '''
31 g_fast : siemens (constant)
32 I_fast_post = g_fast*(v_post - E_syn)/(1+exp(s_fast*(V_fast-v_pre))) : amp (summed)
33 '''
34 fast_synapses = Synapses(circuit, circuit, model=eqs_fast)
35 fast_synapses.connect('label_pre != label_post and not (label_pre == PY and label_post == ABPD)')
36 fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
37 fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
38 fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
39 fast_synapses.g_fast['label_pre == LP and label_post == PY'] = 0.02*uS
40 fast_synapses.g_fast['label_pre == PY and label_post == LP'] = 0.005*uS
41
42 s_slow = 1/mV; V_slow = -55*mV; k_1 = 1/ms
43 eqs_slow = '''
44 k_2 : 1/second (constant)
45 g_slow : siemens (constant)
46 I_slow_post = g_slow*m_slow*(v_post-E_syn) : amp (summed)
47 dm_slow/dt = k_1*(1-m_slow)/(1+exp(s_slow*(V_slow-v_pre))) - k_2*m_slow : 1 (clock-driven)
48 '''
49 slow_synapses = Synapses(circuit, circuit, model=eqs_slow, method='exact')
50 slow_synapses.connect('label_pre == ABPD and label_post != ABPD')
51 slow_synapses.g_slow['label_post == LP'] = 0.025*uS
52 slow_synapses.k_2['label_post == LP'] = 0.03/ms
53 slow_synapses.g_slow['label_post == PY'] = 0.015*uS
54 slow_synapses.k_2['label_post == PY'] = 0.008/ms
55
56 run(59.5*second)

```

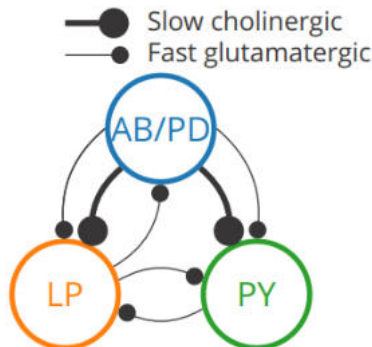
```

36 fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
37 fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
38 fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
39 fast_synapses.g_fast['label_pre == LP and label_post == PY'] = 0.02*uS
40 fast_synapses.g_fast['label_pre == PY and label_post == LP'] = 0.005*uS

51 slow_synapses.g_slow['label_post == LP'] = 0.025*uS
52 slow_synapses.k_2['label_post == LP'] = 0.03/ms
53 slow_synapses.g_slow['label_post == PY'] = 0.015*uS
54 slow_synapses.k_2['label_post == PY'] = 0.008/ms

```

Set up initial values.



```

35 fast_synapses.connect('label_pre != label_post and not (label_pre == PY and label_post == ABPD)')

50 slow_synapses.connect('label_pre == ABPD and label_post != ABPD')

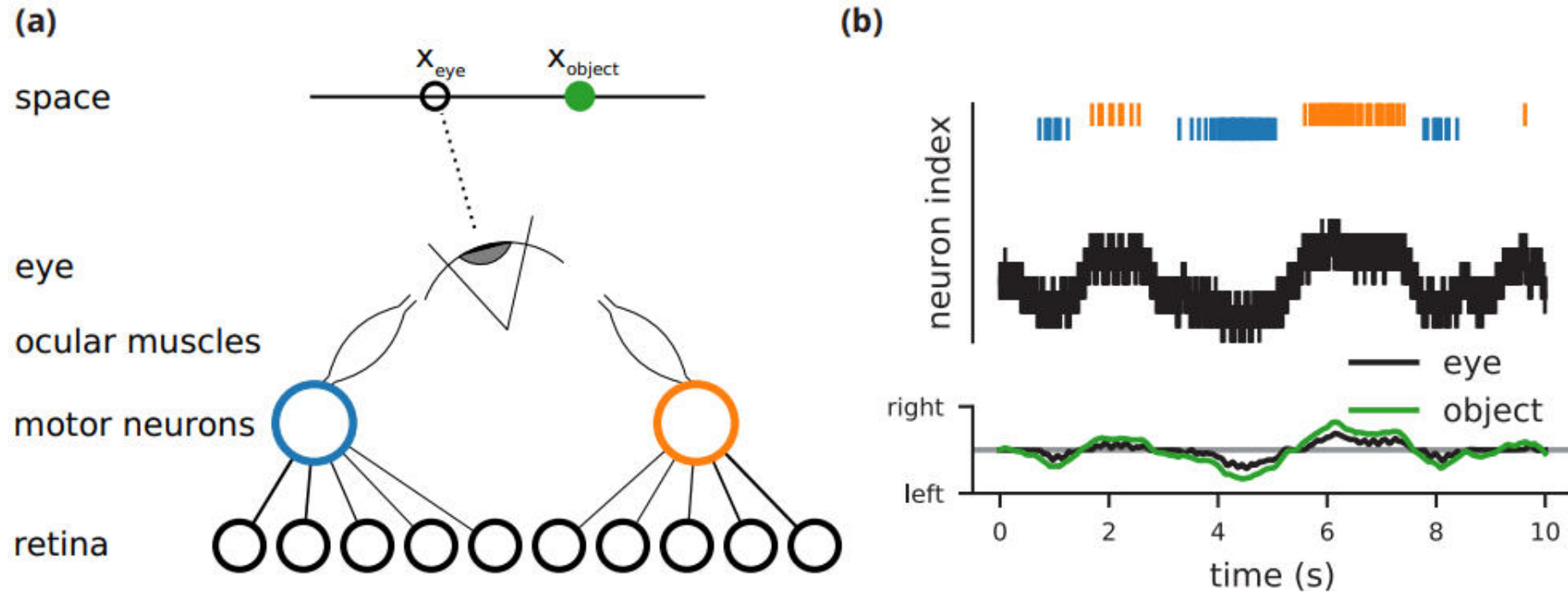
```

Connectivity pattern

Comparison to other approaches

- Implement in a language like C++?
 - Requires significant technical skill
 - Justified if iteration is done thousands of times
 - Difficult to adapt for other purposes
- Use description language such as LEMS / NeuroML2?
 - Like Brian2, but somewhat more verbose
- Use NMODL language in NEURON simulator?
 - Requires learning new language
- Use NESTML language in NEST simulator?
 - Doesn't support graded synapse

Case Study 2: Ocular Model



Two antagonistic muscles are modelled mechanically as elastic spring with friction.

(a) Circuit schematic

(b) Simulated activity of sensory neurons (black) & motor neurons (blue, orange)

Continued...

```
1 from brian2 import *
2
3 alpha = (1/(50*ms))**2; beta = 1/(50*ms); tau_muscle = 20*ms; tau_object = 500*ms
4 eqs_eye = '''dx/dt = velocity : 1
5             dvelocity/dt = alpha*(x0-x)-beta*velocity : 1/second
6             dx0/dt = -x0/tau_muscle : 1
7             dx_object/dt = (noise - x_object)/tau_object: 1
8             dnoise/dt = -noise/tau_object + tau_object**(-0.5)*xi : 1'''
9 eye = NeuronGroup(1, model=eqs_eye, method='euler')
10
11 taum = 20*ms
12 motoneurons = NeuronGroup(2, model='dv/dt = -v/taum : 1', threshold='v>1', reset='v=0',
13                             refractory=5*ms, method='exact')
14
15 motosynapses = Synapses(motoneurons, eye, model='w : 1', on_pre='x0_post += w')
16 motosynapses.connect() # connects all motoneurons to the eye
17 motosynapses.w = [-0.5, 0.5]
18
19 N = 20; width = 2./N; gain = 4.
20 eqs_retina = '''dv/dt = (I-(1+gs)*v)/taum : 1
21                I = gain*exp(-(x_object-x_eye-x_neuron)/width)**2) : 1
22                x_neuron : 1 (constant)
23                x_object : 1 (linked) # position of the object
24                x_eye : 1 (linked) # position of the eye
25                gs : 1 # total synaptic conductance'''
26 retina = NeuronGroup(N, model=eqs_retina, threshold='v>1', reset='v=0', method='exact')
27 retina.v = 'rand()'
28 retina.x_eye = linked_var(eye, 'x')
29 retina.x_object = linked_var(eye, 'x_object')
30 retina.x_neuron = '-1.0 + 2.0*i/(N-1)'
31
32 sensorimotor_synapses = Synapses(retina, motoneurons, model='w : 1 (constant)',
33                                 on_pre='v_post += w')
34 sensorimotor_synapses.connect(j='int(x_neuron_pre > 0)')
35 # Strength scales with eccentricity:
36 sensorimotor_synapses.w = '20*abs(x_neuron_pre)/N_pre'
37
38 run(10*second)
```

```
4 eqs_eye = '''dx/dt = velocity : 1
5             dvelocity/dt = alpha*(x0-x)-beta*velocity : 1/second
6             dx0/dt = -x0/tau_muscle : 1
```

Position of eye follows 2nd-order differential equation.

```
7             dx_object/dt = (noise - x_object)/tau_object: 1
8             dnoise/dt = -noise/tau_object + tau_object**(-0.5)*xi : 1'''
```

Stimulus moves in a stochastic process.

```
11 taum = 20*ms
12 motoneurons = NeuronGroup(2, model='dv/dt = -v/taum : 1', threshold='v>1', reset='v=0',
13                             refractory=5*ms, method='exact')
```

Muscles are controlled by two motoneurons.

Continued...

```
1 from brian2 import *
2
3 alpha = (1/(50*ms))**2; beta = 1/(50*ms); tau_muscle = 20*ms; tau_object = 500*ms
4 eqs_eye = '''dx/dt = velocity : 1
5             dvelocity/dt = alpha*(x0-x)-beta*velocity : 1/second
6             dx0/dt = -x0/tau_muscle : 1
7             dx_object/dt = (noise - x_object)/tau_object: 1
8             dnoise/dt = -noise/tau_object + tau_object**-0.5*xi : 1'''
9 eye = NeuronGroup(1, model=eqs_eye, method='euler')
10
11 taum = 20*ms
12 motoneurons = NeuronGroup(2, model='dv/dt = -v/taum : 1', threshold='v>1', reset='v=0',
13                             refractory=5*ms, method='exact')
14
15 motosynapses = Synapses(motoneurons, eye, model='w : 1', on_pre='x0_post += w')
16 motosynapses.connect() # connects all motoneurons to the eye
17 motosynapses.w = [-0.5, 0.5]
18
19 N = 20; width = 2./N; gain = 4.
20 eqs_retina = '''dv/dt = (I-(1+gs)*v)/taum : 1
21                I = gain*exp(-((x_object-x_eye-x_neuron)/width)**2) : 1
22                x_neuron : 1 (constant)
23                x_object : 1 (linked) # position of the object
24                x_eye : 1 (linked) # position of the eye
25                gs : 1 # total synaptic conductance'''
26 retina = NeuronGroup(N, model=eqs_retina, threshold='v>1', reset='v=0', method='exact')
27 retina.v = 'rand()'
28 retina.x_eye = linked_var(eye, 'x')
29 retina.x_object = linked_var(eye, 'x_object')
30 retina.x_neuron = '-1.0 + 2.0*i/(N-1)'
31
32 sensorimotor_synapses = Synapses(retina, motoneurons, model='w : 1 (constant)',
33                                 on_pre='v_post += w')
34 sensorimotor_synapses.connect(j='int(x_neuron_pre > 0)')
35 # Strength scales with eccentricity:
36 sensorimotor_synapses.w = '20*abs(x_neuron_pre)/N_pre'
37
38 run(10*second)
```

```
20 eqs_retina = '''dv/dt = (I-(1+gs)*v)/taum : 1
21                I = gain*exp(-((x_object-x_eye-x_neuron)/width)**2) : 1
```

Retinal neurons receive visual input.

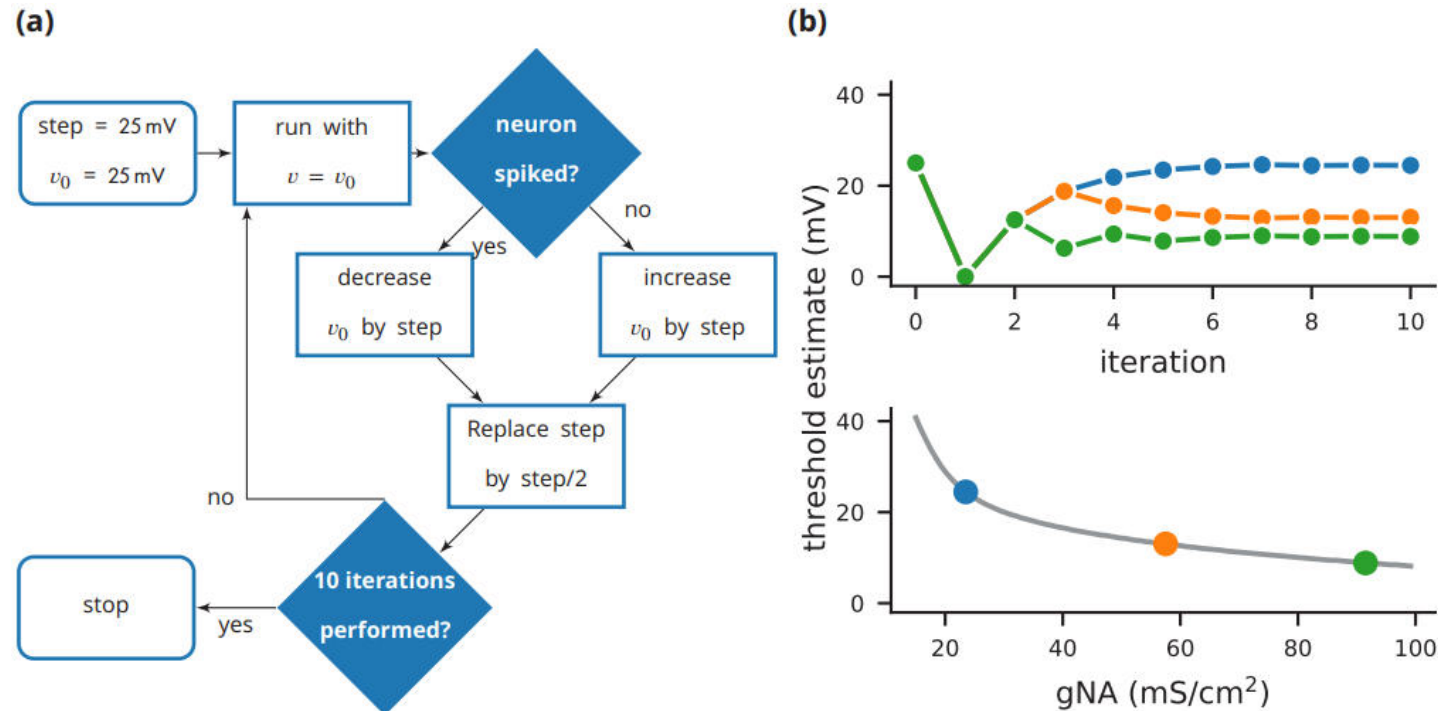
```
32 sensorimotor_synapses = Synapses(retina, motoneurons, model='w : 1 (constant)',
33                                 on_pre='v_post += w')
```

Retinal neurons project on motoneuron to control muscle.

Comparison to other approaches

- Use language such as LEMS / NeuroML2 / NMODL?
 - Same as “Case Study 1”
- Use NESTML language in NEST simulator?
 - Doesn't support continuous interaction between single environment & multiple neurons

Case Study 3: Threshold Finding



Determine the voltage firing threshold of a neuron.

(a) Flowchart of bisection algorithm

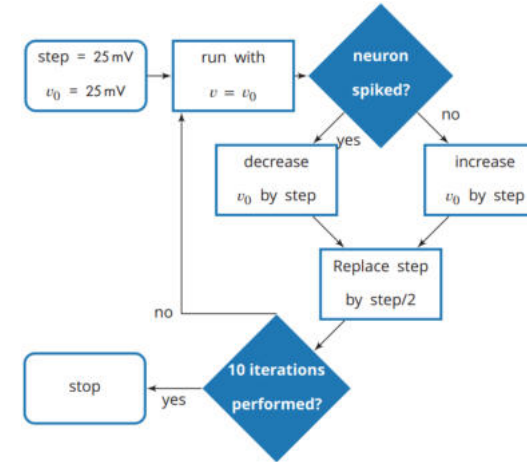
(b) Refinement of threshold over iterations (three different Na densities)

Continued...

```
1 from brian2 import *
2 defaultclock.dt = 0.01*ms
3
4 El = 10.613*mV; ENa = 115*mV; EK = -12*mV
5 gl = 0.3*mS/cm**2; gK = 36*mS/cm**2; C = 1*uF/cm**2
6 gNa_min = 15*mS/cm**2; gNa_max = 100*mS/cm**2
7
8 eqs = '''dv/dt = (gl*(El - v) + gNa*m**3*h*(ENa - v) + gK*n**4*(EK - v)) / C : volt
9         gNa : siemens/meter**2
10        dm/dt = alphan*(1 - m) - betam*m : 1
11        dn/dt = alphan*(1 - n) - betan*n : 1
12        dh/dt = alphah*(1 - h) - betah*h : 1
13        alphan = (0.1/mV)*(-v + 25*mV)/(exp((-v + 25*mV)/(10*mV)) - 1)/ms : Hz
14        betam = 4 * exp(-v/(18*mV))/ms : Hz
15        alphah = 0.07 * exp(-v/(20*mV))/ms : Hz
16        betah = 1/(exp((-v+30*mV) / (10*mV)) + 1)/ms : Hz
17        alphan = (0.01/mV) * (-v+10*mV) / (exp((-v+10*mV) / (10*mV)) - 1)/ms : Hz
18        betan = 0.125*exp(-v/(80*mV))/ms : Hz'''
19 neurons = NeuronGroup(100, eqs, threshold='v > 50*mV', method='exponential_euler')
20 neurons.gNa = 'gNa_min + (gNa_max - gNa_min)*1.0*i/N'
21 neurons.v = 0*mV
22 neurons.m = '1/(1 + betam/alphan)'
23 neurons.n = '1/(1 + betan/alphan)'
24 neurons.h = '1/(1 + betah/alphah)'
25 S = SpikeMonitor(neurons)
26
27 store()
28
29 # We locate the threshold by bisection
30 v0 = 25*mV*ones(len(neurons))
31 step = 25*mV
32
33 for i in range(10):
34     restore()
35     neurons.v = v0
36     run(20*ms)
37     v0[S.count == 0] += step
38     v0[S.count > 0] -= step
39     step /= 2.0
```

```
30 v0 = 25*mV*ones(len(neurons))
31 step = 25*mV
```

Set initial estimate and step width.



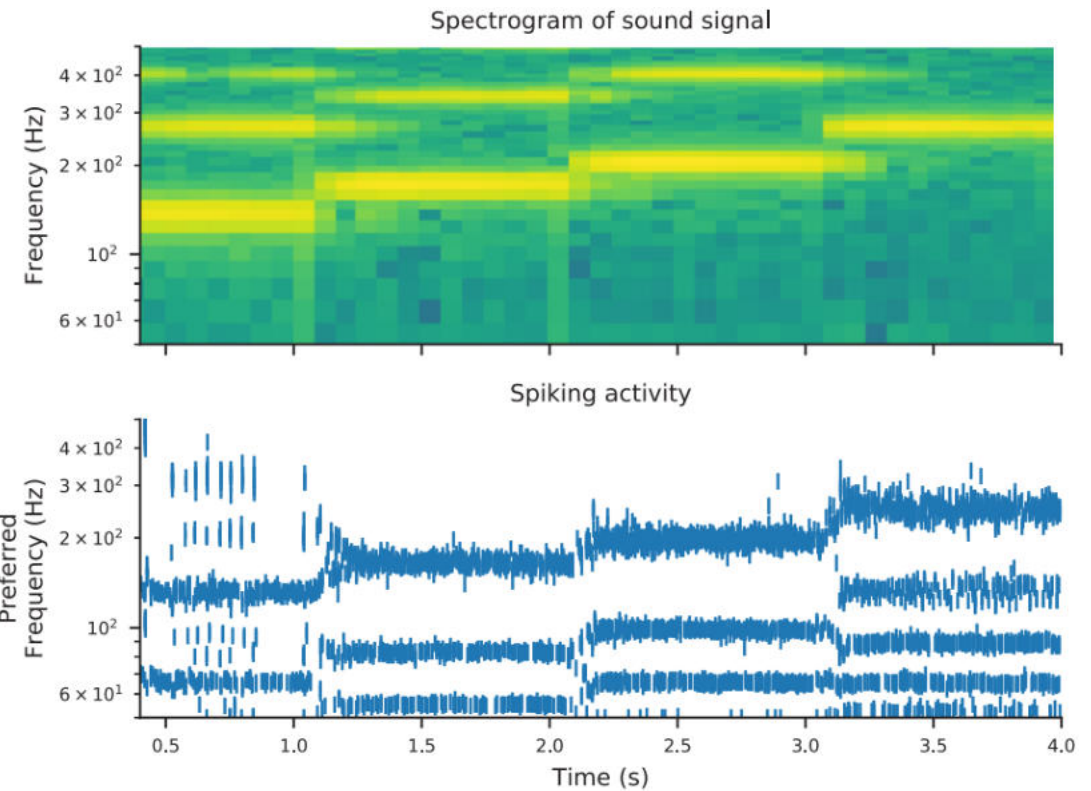
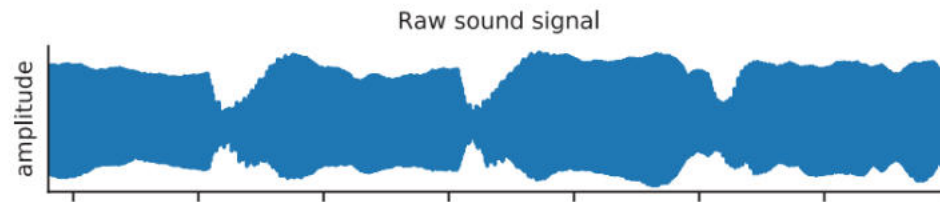
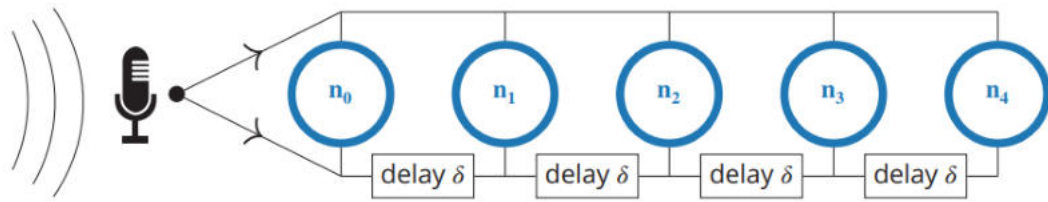
```
33 for i in range(10):
34     restore()
35     neurons.v = v0
36     run(20*ms)
37     v0[S.count == 0] += step
38     v0[S.count > 0] -= step
39     step /= 2.0
```

Perform the bisection for a certain #iteration.

Comparison to other approaches

- Use language such as LEMS / NeuroML2?
 - Can only specify duration and step size
- Use NEST simulator?
 - Like Brian2; use SLI or Python
- Use NEURON simulator?
 - Like Brian2; use HOC or Python

Case Study 4: Real-time Audio



Detect the pitch based on autocorrelation of a signal.

Continued...

```
1 from brian2 import *
2 import os
3 set_device('cpp_standalone')
4
5 sample_rate = 48kHz; buffer_size = 128; defaultclock.dt = 1/sample_rate
6 max_delay = 20*ms; tau_ear = 1*ms; tau_th = 5*ms
7 min_freq = 50*Hz; max_freq = 1000*Hz; num_neurons = 300; tau = 1*ms; sigma = .1
8
9 @implementation('cpp', '')
10 PaStream *_init_stream() {
11     PaStream* stream;
12     Pa_Initialize();
13     Pa_OpenDefaultStream(&stream, 1, 0, paFloat32, SAMPLE_RATE, BUFFER_SIZE, NULL, NULL);
14     Pa_StartStream(stream);
15     return stream;
16 }
17
18 float get_sample(const double t) {
19     static PaStream* stream = _init_stream();
20     static float buffer[BUFFER_SIZE];
21     static int next_sample = BUFFER_SIZE;
22
23     if (next_sample >= BUFFER_SIZE)
24     {
25         Pa_ReadStream(stream, buffer, BUFFER_SIZE);
26         next_sample = 0;
27     }
28     return buffer[next_sample++];
29 }''' , libraries=['portaudio'], headers=['<portaudio.h>'],
30     define_macros=[('BUFFER_SIZE', buffer_size),
31                    ('SAMPLE_RATE', sample_rate)])
32 @check_units(t=second, result=1)
33 def get_sample(t):
34     raise NotImplementedError('Use a C++-based code generation target.')
35
36 eqs_ear = '''dx/dt = (sound - x)/tau_ear : 1 (unless refractory)
37             dth/dt = (0.1*x - th)/tau_th : 1
38             sound = clip(get_sample(t), 0, inf) : 1 (constant over dt)'''
39 receptors = NeuronGroup(1, eqs_ear, threshold='x>th',
40                         reset='x=0; th = th*2.5 + 0.01',
41                         refractory=2*ms, method='exact')
42 receptors.th = 1
43
44 eqs_neurons = '''dv/dt = -v/tau+sigma*(2./tau)**.5*xi : 1
45                 freq : Hz (constant)'''
46 neurons = NeuronGroup(num_neurons, eqs_neurons, threshold='v>1', reset='v=0', method='euler')
47 neurons.freq = 'exp(log(min_freq/Hz)+(i+1.0/(num_neurons-1))*log(max_freq/min_freq))*Hz'
48
49 synapses = Synapses(receptors, neurons, on_pre='v += 0.5', multisynaptic_index='k')
50 synapses.connect(n=2) # one synapse without delay; one with delay
51 synapses.delay['k == 1'] = '1/freq_post'
52
53 run(10*second)
```

Two modes:

(1) Runtime mode:

- Python controls overall simulation.
- It calls compiled code objects to do heavy lifting.
- Overhead: Repeated switching from Python to another language
 - Justified if flexibility is preferred

(2) Standalone mode:

- Low-level code is generated.
- It controls overall simulation.
- Able to generate code for target platform (GPU)

Comparison to other approaches

- Use language such as LEMS / NeuroML2?
 - Not possible
- Use NEST simulator?
 - Utilize MUSIC framework to couple multiple simulators
 - Cannot apply continuous-valued inputs
- Use NEURON simulator?
 - Can include user-written C code
 - No documented mechanism to link external libraries

Drawbacks

1. Explicit model definitions

- Difficult to design tools to programmatically inspect a model
 - Rebuttal: Reduce risk of difference between implementation & description

2. Tightly integrated simulation flow

- Difficult to reuse or programmatically compare a model
 - Rebuttal: Reduce complexity & chance of errors

3. No scaling up

- Lack of support for running large networks over multiple machines
 - Rebuttal: Most people use smaller networks for parameter exploration.

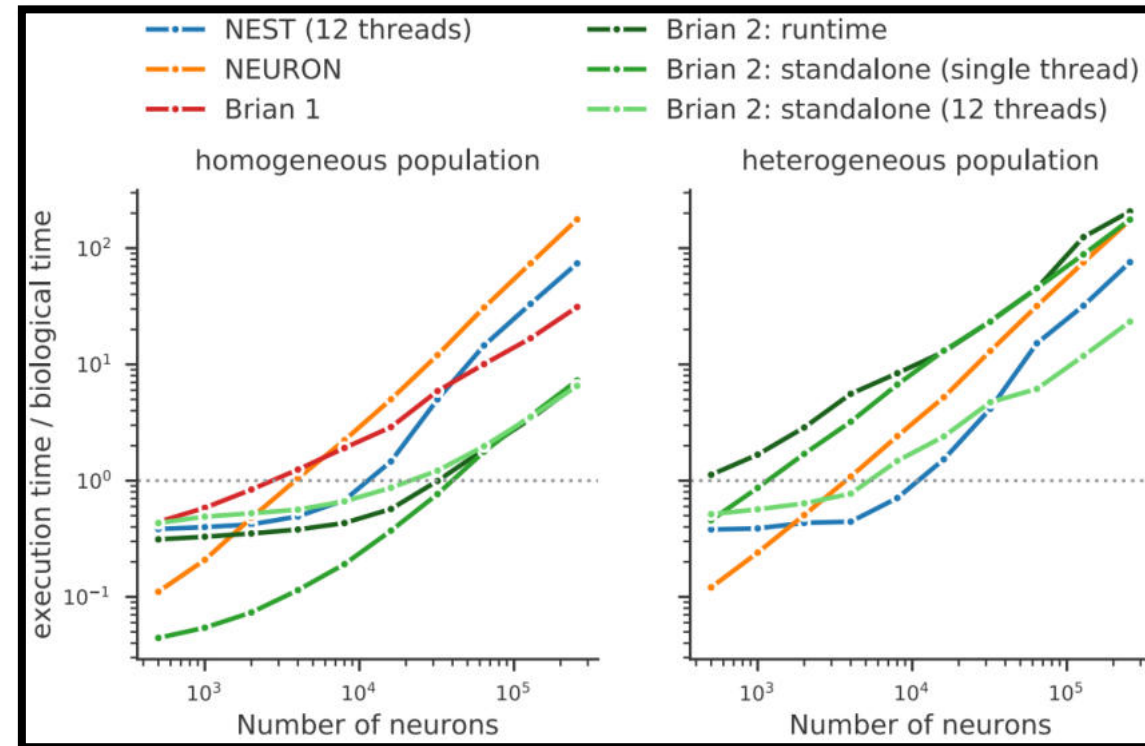
Continued...

4. Rudimentary multi-compartmental models

- Not as mature as NEURON or GENESIS simulator

5. Automated optimization techniques

- Generate optimization for specialization of models



CUBA benchmark



Questions?
Comments?
Concerns?