Brian 2
An Intuitive & Efficient Neural Simulator

Marcel Stimberg[1], **Romain Brette**[1], **Dan FM Goodman**[2]

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

[2]Department 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…

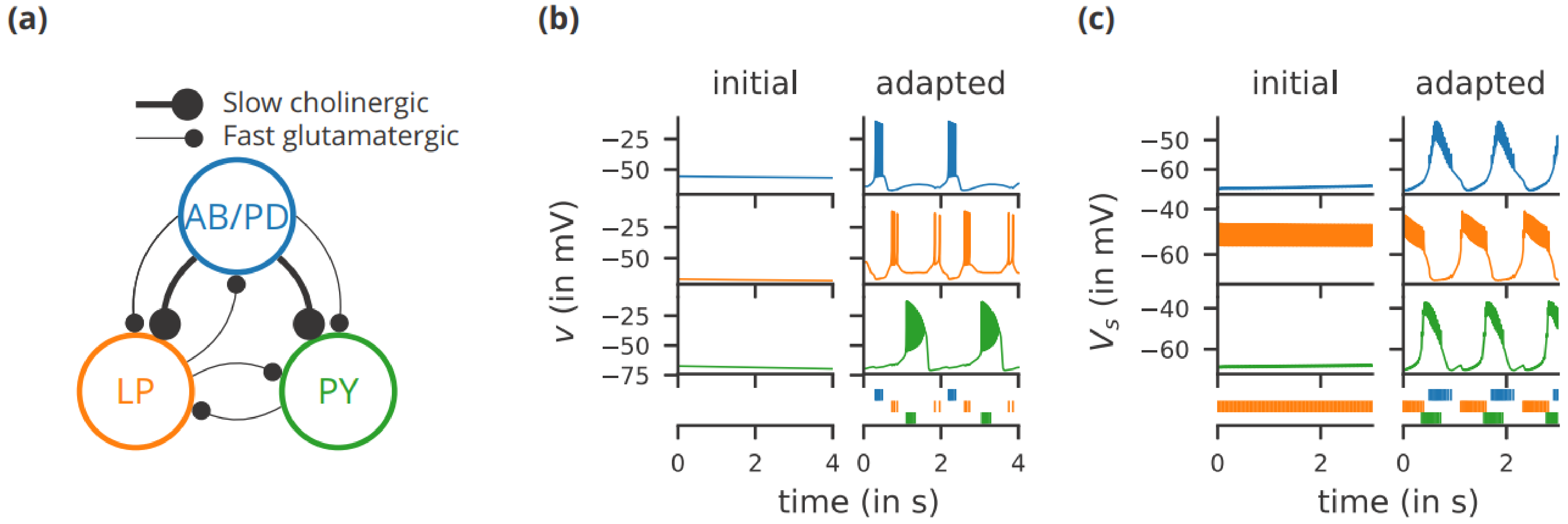
Brian 2 solves this trade-off.

- 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



Generate a stereotypical triphasic motor pattern.

(a) Circuit schematic

(b) Simulated neuron activity

(c) Simulated neuron activity (*biologically detailed*)

Continued…

```python
from brian2 import *
defaultclock.dt = 0.01*ms;
Delta_T = 17.5*mV         ; v_T = -40*mV       ; tau = 2*ms          ; tau_adapt = .02*second
tau_Ca = 150*ms           ; tau_x = 2*second   ; v_r = -68*mV        ; tau_z = 5*second
a = 1/Delta_T**3          ; b = 3/Delta_T**2   ; c = 1.2*nA          ; d = 2.5*nA/Delta_T**2
C = 60*pF                 ; S = 2*nA/Delta_T   ; G = 28.5*nS
eqs = '''
dv/dt = (Delta_T*g*(-a*(v - v_T)**3 + b*(v - v_T)**2) + w - x - I_fast - I_slow)/C : volt
dw/dt = (c - d*(v - v_T)**2 - w)/tau : amp
dx/dt = (s*(v - v_r) - x)/tau_x : amp
s = S*(1 - tanh(z)) : siemens
g = G*(1 + tanh(z)) : siemens
dCa/dt = -Ca/tau_Ca : 1
dz/dt = tanh(Ca - Ca_target)/tau_z : 1
I_fast : amp
I_slow : amp
Ca_target : 1 (constant)
label : integer (constant)
'''
ABPD, LP, PY = 0, 1, 2
circuit = NeuronGroup(3, eqs, threshold='v>-20*mV', refractory='v>-20*mV', reset='Ca += 0.1',
                      method='rk2')
circuit.label = [ABPD, LP, PY]
circuit.v = v_r
circuit.w = '-5*nA*rand()'
circuit.z = 'rand()*0.2 - 0.1'
circuit.Ca_target = [0.048, 0.0384, 0.06]

s_fast = 0.2/mV; V_fast = -50*mV; E_syn = -75*mV
eqs_fast = '''
g_fast : siemens (constant)
I_fast_post = g_fast*(v_post - E_syn)/(1+exp(s_fast*(V_fast-v_pre))) : amp (summed)
'''
fast_synapses = Synapses(circuit, circuit, model=eqs_fast)
fast_synapses.connect('label_pre != label_post and not (label_pre == PY and label_post == ABPD)')
fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
fast_synapses.g_fast['label_pre == LP and label_post == PY']   = 0.02*uS
fast_synapses.g_fast['label_pre == PY and label_post == LP']   = 0.005*uS

s_slow = 1/mV; V_slow = -55*mV; k_1 = 1/ms
eqs_slow = '''
k_2 : 1/second (constant)
g_slow : siemens (constant)
I_slow_post = g_slow*m_slow*(v_post-E_syn) : amp (summed)
dm_slow/dt = k_1*(1-m_slow)/(1+exp(s_slow*(V_slow-v_pre))) - k_2*m_slow : 1 (clock-driven)
'''
slow_synapses = Synapses(circuit, circuit, model=eqs_slow, method='exact')
slow_synapses.connect('label_pre == ABPD and label_post != ABPD')
slow_synapses.g_slow['label_post == LP'] = 0.025*uS
slow_synapses.k_2['label_post == LP']    = 0.03/ms
slow_synapses.g_slow['label_post == PY'] = 0.015*uS
slow_synapses.k_2['label_post == PY']    = 0.008/ms

run(59.5*second)
```

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

Neuron model is written in differential equation.

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

Calcium trace decays.

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

Calcium trace increases at each spike.

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

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

Conductances are regulated by trace difference.

Continued…

```
from brian2 import *
defaultclock.dt = 0.01*ms;
Delta_T = 17.5*mV        ; v_T = -40*mV      ; tau = 2*ms         ; tau_adapt = .02*second
tau_Ca = 150*ms          ; tau_x = 2*second  ; v_r = -68*mV       ; tau_z = 5*second
a = 1/Delta_T**3         ; b = 3/Delta_T**2  ; c = 1.2*nA         ; d = 2.5*nA/Delta_T**2
C = 60*pF                ; S = 2*nA/Delta_T  ; G = 28.5*nS
eqs = '''
dv/dt = (Delta_T*g*(-a*(v - v_T)**3 + b*(v - v_T)**2) + w - x - I_fast - I_slow)/C : volt
dw/dt = (c - d*(v - v_T)**2 - w)/tau : amp
dx/dt = (s*(v - v_r) - x)/tau_x : amp
s = S*(1 - tanh(z)) : siemens
g = G*(1 + tanh(z)) : siemens
dCa/dt = -Ca/tau_Ca : 1
dz/dt = tanh(Ca - Ca_target)/tau_z : 1
I_fast : amp
I_slow : amp
Ca_target : 1 (constant)
label : integer (constant)
'''
ABPD, LP, PY = 0, 1, 2
circuit = NeuronGroup(3, eqs, threshold='v>-20*mV', refractory='v>-20*mV', reset='Ca += 0.1',
                      method='rk2')
circuit.label = [ABPD, LP, PY]
circuit.v = v_r
circuit.w = '-5*nA*rand()'
circuit.z = 'rand()*0.2 - 0.1'
circuit.Ca_target = [0.048, 0.0384, 0.06]

s_fast = 0.2/mV; V_fast = -50*mV; E_syn = -75*mV
eqs_fast = '''
g_fast : siemens (constant)
I_fast_post = g_fast*(v_post - E_syn)/(1+exp(s_fast*(V_fast-v_pre))) : amp (summed)
'''
fast_synapses = Synapses(circuit, circuit, model=eqs_fast)
fast_synapses.connect('label_pre != label_post and not (label_pre == PY and label_post == ABPD)')
fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
fast_synapses.g_fast['label_pre == LP and label_post == PY']   = 0.02*uS
fast_synapses.g_fast['label_pre == PY and label_post == LP']   = 0.005*uS

s_slow = 1/mV; V_slow = -55*mV; k_1 = 1/ms
eqs_slow = '''
k_2 : 1/second (constant)
g_slow : siemens (constant)
I_slow_post = g_slow*m_slow*(v_post-E_syn) : amp (summed)
dm_slow/dt = k_1*(1-m_slow)/(1+exp(s_slow*(V_slow-v_pre))) - k_2*m_slow : 1 (clock-driven)
'''
slow_synapses = Synapses(circuit, circuit, model=eqs_slow, method='exact')
slow_synapses.connect('label_pre == ABPD and label_post != ABPD')
slow_synapses.g_slow['label_post == LP'] = 0.025*uS
slow_synapses.k_2['label_post == LP']    = 0.03/ms
slow_synapses.g_slow['label_post == PY'] = 0.015*uS
slow_synapses.k_2['label_post == PY']    = 0.008/ms

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')
```

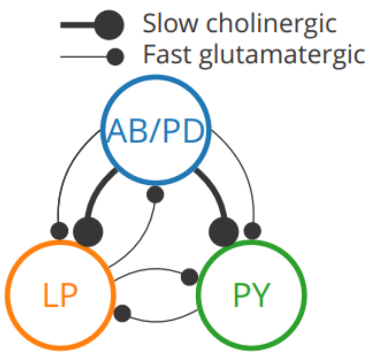

Nonlinear (graded) slow synapse

Continued...

```python
from brian2 import *
defaultclock.dt = 0.01*ms;
Delta_T = 17.5*mV          ; v_T = -40*mV        ; tau = 2*ms           ; tau_adapt = .02*second
tau_Ca = 150*ms            ; tau_x = 2*second    ; v_r = -68*mV         ; tau_z = 5*second
a = 1/Delta_T**3           ; b = 3/Delta_T**2    ; c = 1.2*nA           ; d = 2.5*nA/Delta_T**2
C = 60*pF                  ; S = 2*nA/Delta_T    ; G = 28.5*nS
eqs = '''
dv/dt = (Delta_T*g*(-a*(v - v_T)**3 + b*(v - v_T)**2) + w - x - I_fast - I_slow)/C : volt
dw/dt = (c - d*(v - v_T)**2 - w)/tau : amp
dx/dt = (s*(v - v_r) - x)/tau_x : amp
s = S*(1 - tanh(z)) : siemens
g = G*(1 + tanh(z)) : siemens
dCa/dt = -Ca/tau_Ca : 1
dz/dt = tanh(Ca - Ca_target)/tau_z : 1
I_fast : amp
I_slow : amp
Ca_target : 1 (constant)
label : integer (constant)
'''
ABPD, LP, PY = 0, 1, 2
circuit = NeuronGroup(3, eqs, threshold='v>-20*mV', refractory='v>-20*mV', reset='Ca += 0.1',
                      method='rk2')
circuit.label = [ABPD, LP, PY]
circuit.v = v_r
circuit.w = '-5*nA*rand()'
circuit.z = 'rand()*0.2 - 0.1'
circuit.Ca_target = [0.048, 0.0384, 0.06]

s_fast = 0.2/mV; V_fast = -50*mV; E_syn = -75*mV
eqs_fast = '''
g_fast : siemens (constant)
I_fast_post = g_fast*(v_post - E_syn)/(1+exp(s_fast*(V_fast-v_pre))) : amp (summed)
'''
fast_synapses = Synapses(circuit, circuit, model=eqs_fast)
fast_synapses.connect('label_pre != label_post and not (label_pre == PY and label_post == ABPD)')
fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
fast_synapses.g_fast['label_pre == LP and label_post == PY']   = 0.02*uS
fast_synapses.g_fast['label_pre == PY and label_post == LP']   = 0.005*uS

s_slow = 1/mV; V_slow = -55*mV; k_1 = 1/ms
eqs_slow = '''
k_2 : 1/second (constant)
g_slow : siemens (constant)
I_slow_post = g_slow*m_slow*(v_post-E_syn) : amp (summed)
dm_slow/dt = k_1*(1-m_slow)/(1+exp(s_slow*(V_slow-v_pre))) - k_2*m_slow : 1 (clock-driven)
'''
slow_synapses = Synapses(circuit, circuit, model=eqs_slow, method='exact')
slow_synapses.connect('label_pre == ABPD and label_post != ABPD')
slow_synapses.g_slow['label_post == LP'] = 0.025*uS
slow_synapses.k_2['label_post == LP']    = 0.03/ms
slow_synapses.g_slow['label_post == PY'] = 0.015*uS
slow_synapses.k_2['label_post == PY']    = 0.008/ms

run(59.5*second)
```


```python
fast_synapses.g_fast['label_pre == ABPD and label_post == LP'] = 0.015*uS
fast_synapses.g_fast['label_pre == ABPD and label_post == PY'] = 0.005*uS
fast_synapses.g_fast['label_pre == LP and label_post == ABPD'] = 0.01*uS
fast_synapses.g_fast['label_pre == LP and label_post == PY']   = 0.02*uS
fast_synapses.g_fast['label_pre == PY and label_post == LP']   = 0.005*uS

slow_synapses.g_slow['label_post == LP'] = 0.025*uS
slow_synapses.k_2['label_post == LP']    = 0.03/ms
slow_synapses.g_slow['label_post == PY'] = 0.015*uS
slow_synapses.k_2['label_post == PY']    = 0.008/ms
```


Set up initial values.




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

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


Connectivity pattern

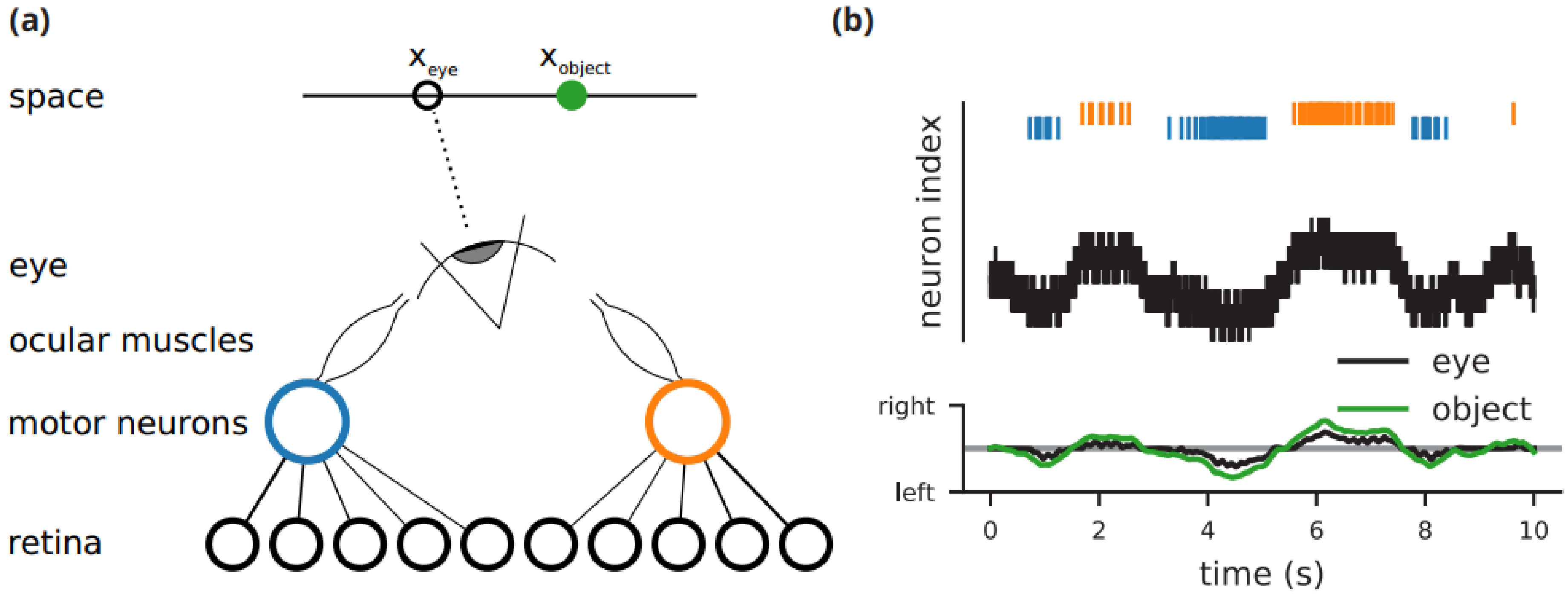
Continued…

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

[define non-standard models]

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…

```
from brian2 import *

alpha = (1/(50*ms))**2; beta = 1/(50*ms); tau_muscle = 20*ms; tau_object = 500*ms
eqs_eye = '''dx/dt = velocity : 1
          dvelocity/dt = alpha*(x0-x)-beta*velocity : 1/second
          dx0/dt = -x0/tau_muscle : 1
          dx_object/dt = (noise - x_object)/tau_object:  1
          dnoise/dt = -noise/tau_object + tau_object**-0.5*xi : 1'''
eye = NeuronGroup(1, model=eqs_eye, method='euler')

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

motosynapses = Synapses(motoneurons, eye, model='w : 1', on_pre='x0_post += w')
motosynapses.connect() # connects all motoneurons to the eye
motosynapses.w = [-0.5, 0.5]

N = 20; width = 2./N; gain = 4.
eqs_retina = '''dv/dt = (I-(1+gs)*v)/taum : 1
             I = gain*exp(-((x_object-x_eye-x_neuron)/width)**2) : 1
             x_neuron : 1 (constant)
             x_object : 1 (linked) # position of the object
             x_eye : 1 (linked) # position of the eye
             gs : 1 # total synaptic conductance'''
retina = NeuronGroup(N, model=eqs_retina, threshold='v>1', reset='v=0', method='exact')
retina.v = 'rand()'
retina.x_eye = linked_var(eye, 'x')
retina.x_object = linked_var(eye, 'x_object')
retina.x_neuron = '-1.0 + 2.0*i/(N-1)'

sensorimotor_synapses = Synapses(retina, motoneurons, model='w : 1 (constant)',
                                 on_pre='v_post += w')
sensorimotor_synapses.connect(j='int(x_neuron_pre > 0)')
# Strength scales with eccentricity:
sensorimotor_synapses.w = '20*abs(x_neuron_pre)/N_pre'

run(10*second)
```


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


Position of eye follows 2nd-order differential equation.


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


Stimulus moves in a stochastic process.


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


Muscles are controlled by two motoneurons.

Continued…

```python
from brian2 import *

alpha = (1/(50*ms))**2; beta = 1/(50*ms); tau_muscle = 20*ms; tau_object = 500*ms
eqs_eye = '''dx/dt = velocity : 1
             dvelocity/dt = alpha*(x0-x)-beta*velocity : 1/second
             dx0/dt = -x0/tau_muscle : 1
             dx_object/dt = (noise - x_object)/tau_object:  1
             dnoise/dt = -noise/tau_object + tau_object**-0.5*xi : 1'''
eye = NeuronGroup(1, model=eqs_eye, method='euler')

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

motosynapses = Synapses(motoneurons, eye, model='w : 1', on_pre='x0_post += w')
motosynapses.connect() # connects all motoneurons to the eye
motosynapses.w = [-0.5, 0.5]

N = 20; width = 2./N; gain = 4.
eqs_retina = '''dv/dt = (I-(1+gs)*v)/taum : 1
                I = gain*exp(-((x_object-x_eye-x_neuron)/width)**2) : 1
                x_neuron : 1 (constant)
                x_object : 1 (linked) # position of the object
                x_eye : 1 (linked) # position of the eye
                gs : 1 # total synaptic conductance'''
retina = NeuronGroup(N, model=eqs_retina, threshold='v>1', reset='v=0', method='exact')
retina.v = 'rand()'
retina.x_eye = linked_var(eye, 'x')
retina.x_object = linked_var(eye, 'x_object')
retina.x_neuron = '-1.0 + 2.0*i/(N-1)'

sensorimotor_synapses = Synapses(retina, motoneurons, model='w : 1 (constant)',
                                 on_pre='v_post += w')
sensorimotor_synapses.connect(j='int(x_neuron_pre > 0)')
# Strength scales with eccentricity:
sensorimotor_synapses.w = '20*abs(x_neuron_pre)/N_pre'

run(10*second)
```

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

Retinal neurons receive visual input.

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

Retinal neurons project on motoneuron to control muscle.

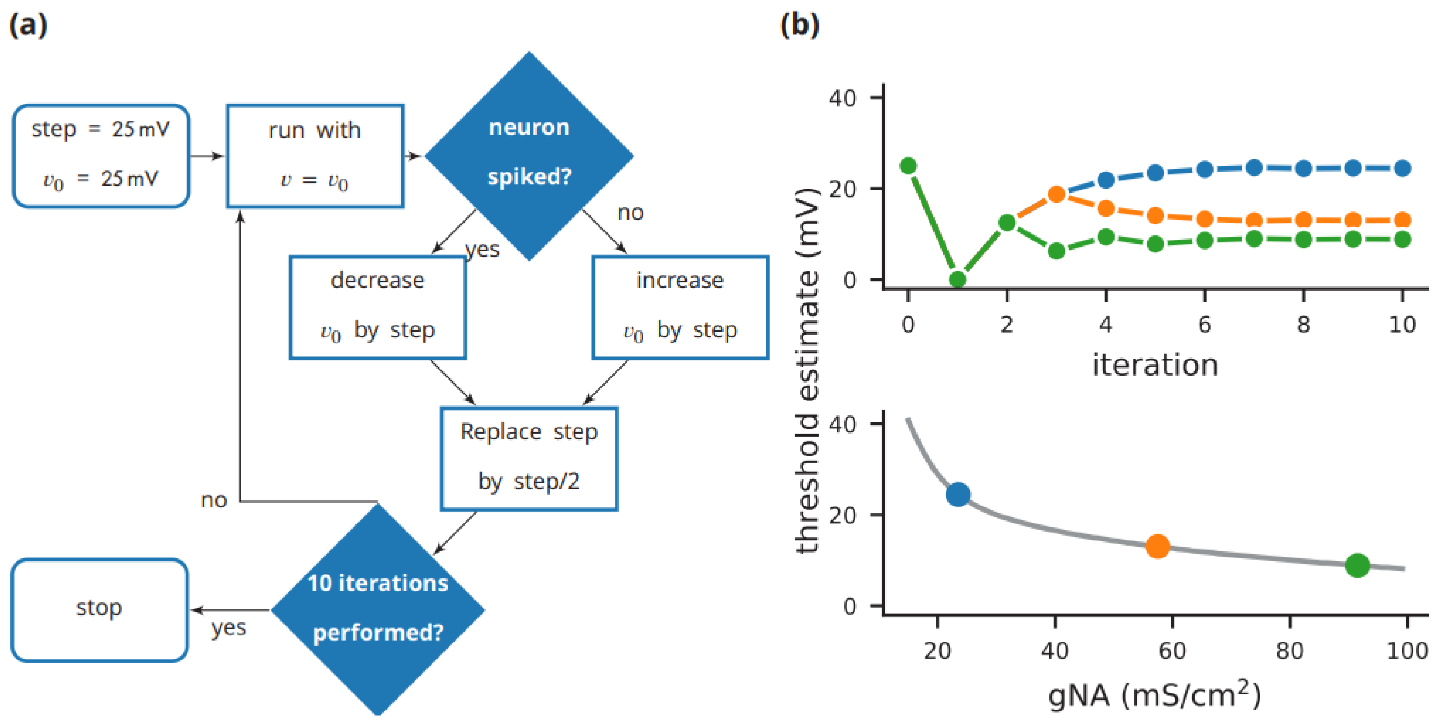
Continued…

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

[define dynamics of stimulus]

Case Study 3: Threshold Finding



Determine the voltage firing threshold of a neuron.

(a) Flowchart of bisection algorithm

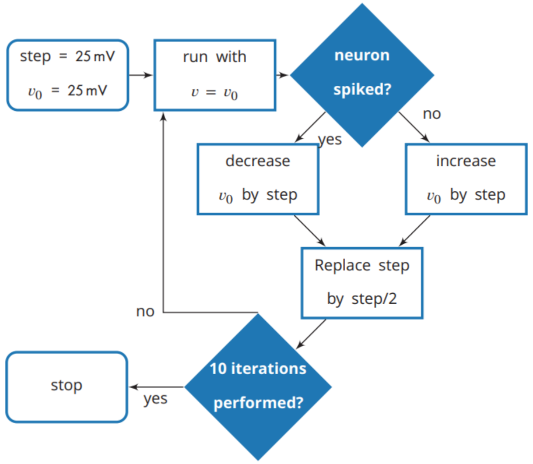
(b) Refinement of threshold over iterations (three different Na densities)

Continued…

```python
from brian2 import *
defaultclock.dt = 0.01*ms

El = 10.613*mV; ENa = 115*mV; EK = -12*mV
gl = 0.3*mS/cm**2; gK = 36*mS/cm**2; C = 1*uF/cm**2
gNa_min = 15*mS/cm**2; gNa_max = 100*mS/cm**2

eqs = '''dv/dt = (gl*(El - v) + gNa*m**3*h*(ENa - v) + gK*n**4*(EK - v)) / C : volt
         gNa : siemens/meter**2
         dm/dt = alpham*(1 - m) - betam*m : 1
         dn/dt = alphan*(1 - n) - betan*n : 1
         dh/dt = alphah*(1 - h) - betah*h : 1
         alpham = (0.1/mV)*(-v + 25*mV)/(exp((-v + 25*mV)/(10*mV)) - 1)/ms : Hz
         betam = 4 * exp(-v/(18*mV))/ms : Hz
         alphah = 0.07 * exp(-v/(20*mV))/ms : Hz
         betah = 1/(exp((-v+30*mV) / (10*mV)) + 1)/ms : Hz
         alphan = (0.01/mV) * (-v+10*mV) / (exp((-v+10*mV) / (10*mV)) - 1)/ms : Hz
         betan = 0.125*exp(-v/(80*mV))/ms : Hz'''
neurons = NeuronGroup(100, eqs, threshold='v > 50*mV', method='exponential_euler')
neurons.gNa = 'gNa_min + (gNa_max - gNa_min)*1.0*i/N'
neurons.v = 0*mV
neurons.m = '1/(1 + betam/alpham)'
neurons.n = '1/(1 + betan/alphan)'
neurons.h = '1/(1 + betah/alphah)'
S = SpikeMonitor(neurons)

store()

# We locate the threshold by bisection
v0 = 25*mV*ones(len(neurons))
step = 25*mV

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


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


Set initial estimate and step width.




```python
for i in range(10):
    restore()
    neurons.v = v0
    run(20*ms)
    v0[S.count == 0] += step
    v0[S.count > 0] -= step
    step /= 2.0
```


Perform the bisection for a certain #iteration.

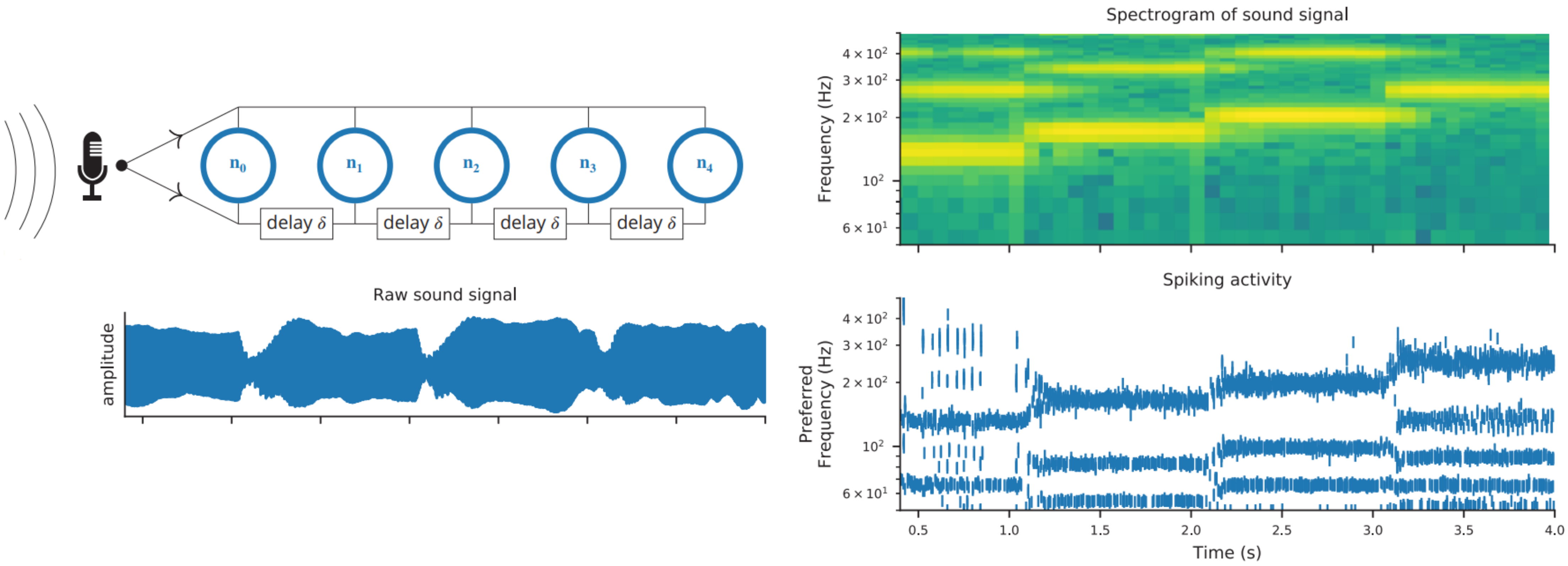
Continued…

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

[define arbitrary protocols]

Case Study 4: Real-time Audio



Detect the pitch based on autocorrelation of a signal.

Continued…

```python
from brian2 import *
import os
set_device('cpp_standalone')

sample_rate = 48*kHz; buffer_size = 128; defaultclock.dt = 1/sample_rate
max_delay = 20*ms; tau_ear = 1*ms; tau_th = 5*ms
min_freq = 50*Hz; max_freq = 1000*Hz; num_neurons = 300; tau = 1*ms; sigma = .1

@implementation('cpp','''
PaStream *_init_stream() {
    PaStream* stream;
    Pa_Initialize();
    Pa_OpenDefaultStream(&stream, 1, 0, paFloat32, SAMPLE_RATE, BUFFER_SIZE, NULL, NULL);
    Pa_StartStream(stream);
    return stream;
}

float get_sample(const double t) {
    static PaStream* stream = _init_stream();
    static float buffer[BUFFER_SIZE];
    static int next_sample = BUFFER_SIZE;

    if (next_sample >= BUFFER_SIZE)
    {
        Pa_ReadStream(stream, buffer, BUFFER_SIZE);
        next_sample = 0;
    }
    return buffer[next_sample++];
}''', libraries=['portaudio'], headers=['<portaudio.h>'],
      define_macros=[('BUFFER_SIZE', buffer_size),
                     ('SAMPLE_RATE', sample_rate)])
@check_units(t=second, result=1)
def get_sample(t):
    raise NotImplementedError('Use a C++-based code generation target.')

eqs_ear = '''dx/dt = (sound - x)/tau_ear: 1 (unless refractory)
             dth/dt = (0.1*x - th)/tau_th : 1
             sound = clip(get_sample(t), 0, inf) : 1 (constant over dt)'''
receptors = NeuronGroup(1, eqs_ear, threshold='x>th',
                        reset='x=0; th = th*2.5 + 0.01',
                        refractory=2*ms, method='exact')
receptors.th = 1

eqs_neurons = '''dv/dt = -v/tau+sigma*(2./tau)**.5*xi : 1
                 freq : Hz (constant)'''
neurons = NeuronGroup(num_neurons, eqs_neurons, threshold='v>1', reset='v=0', method='euler')
neurons.freq = 'exp(log(min_freq/Hz)+(i*1.0/(num_neurons-1))*log(max_freq/min_freq))*Hz'

synapses = Synapses(receptors, neurons, on_pre='v += 0.5', multisynaptic_index='k')
synapses.connect(n=2)  # one synapse without delay; one with delay
synapses.delay['k == 1'] = '1/freq_post'

run(10*second)
```

Two modes:

(1) <u>Runtime mode:</u>

- 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) <u>Standalone mode:</u>

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

Continued…

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

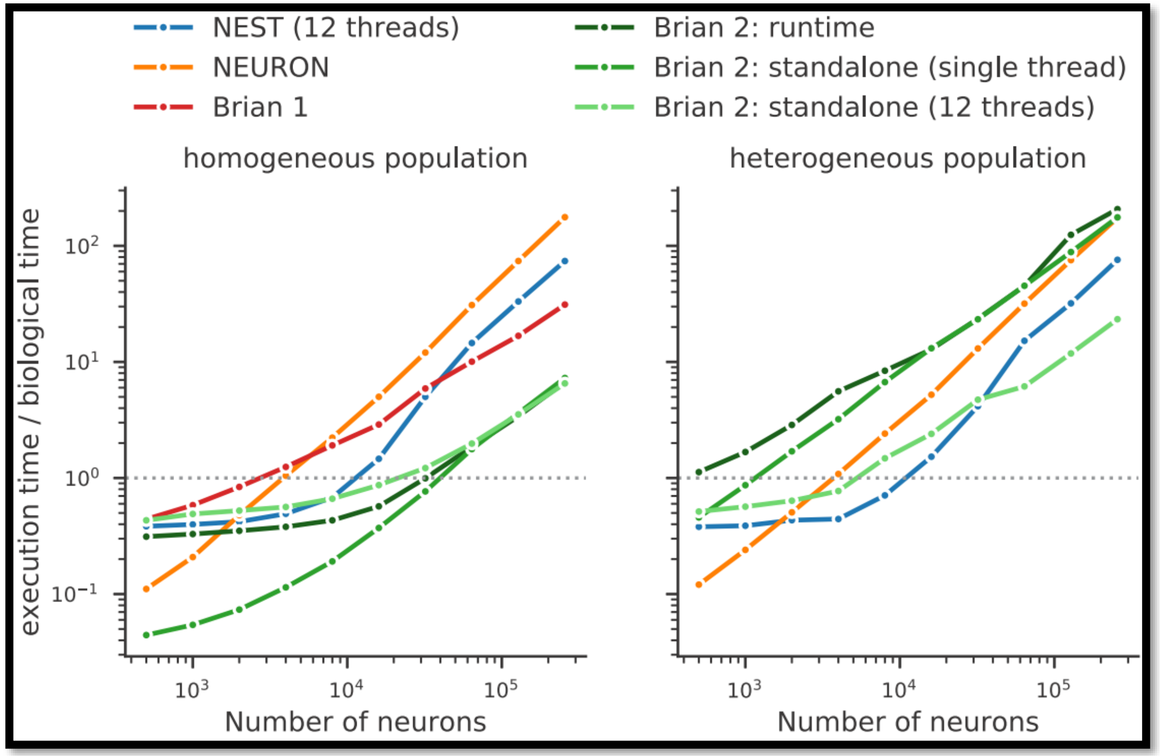
[define interface with other language]

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?