



BRIAN

<http://brian.di.ens.fr>

Dan Goodman & Romain Brette  
Ecole Normale Supérieure  
Projet Odysée

[goodman@di.ens.fr](mailto:goodman@di.ens.fr)  
[brette@di.ens.fr](mailto:brette@di.ens.fr)

# Brian: a pure Python simulator

- Brian is for modelling:
  - Networks of spiking neurons
  - Each neuron is modelled as single compartment
  - Not too many neurons (tens of thousands?)
- Benefits are:
  - Easy to learn and use compared to other software
  - Quick to implement and tweak models

# Outline

- Examples
- Brian in detail
- Interactive session part 1: a simple Brian script
- Short break
- Some more examples
- Brief word on performance issues
- Interactive session part 2: something more complicated

# Leaky I&F with Brian

```
from brian import *

tau = 10*ms
Vr = -70*mV
Vt = -55*mV

model = Model(equations='''
    dV/dt = -(V-Vr)/tau : volt
    ''', threshold=Vt, reset=Vr)
G = NeuronGroup(1,model)

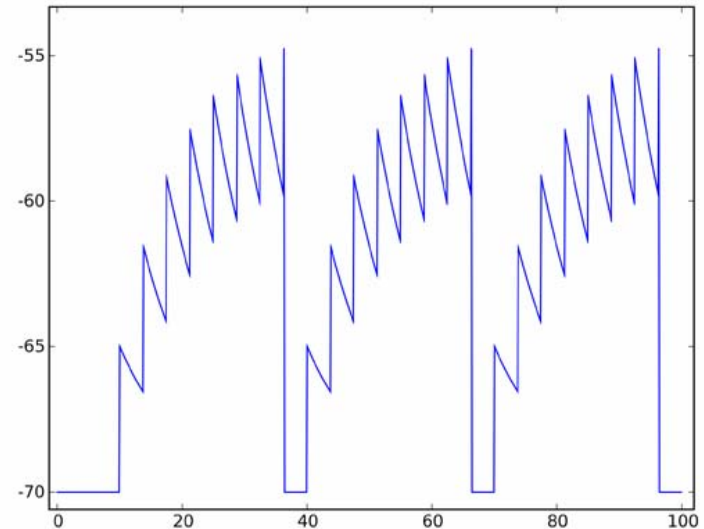
spikes = linspace(10*ms,100*ms,25)
input = MultipleSpikeGeneratorGroup([spikes])

C = Connection(input, G)
C[0,0] = 5*mV

M = StateMonitor(G, 'V', record=True)

G.V = Vr
run(100*ms)
plot(M.times/ms,M[0]/mV)
show()
```

- Equations
- Threshold
- Reset
- Model
- NeuronGroup
- Connection



# Randomly connected network

```
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
'''

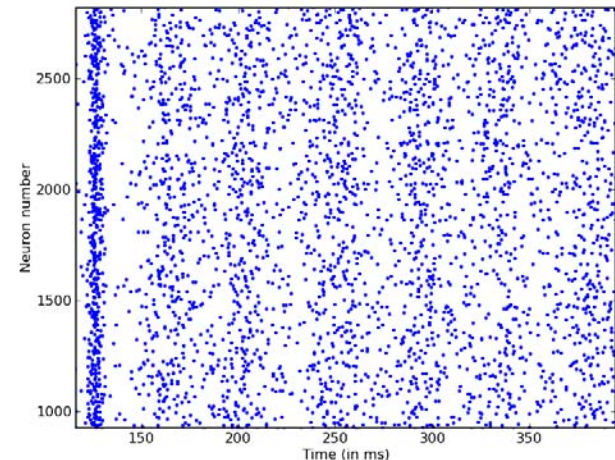
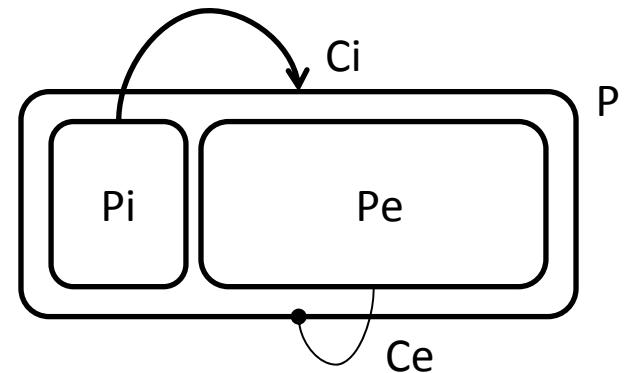
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



Brian in depth

# Anatomy of a Brian script

- Import brian package
- Define neuron model
  - Equations
  - Threshold
  - Reset
- Create groups of neurons
- Create synaptic connections
  - Create a connection
  - Build a synaptic weight matrix
- Create monitors and other operations
- Initialise variables and run the simulation
- Analyse and plot with SciPy, NumPy and PyLab

# First thing: units system

- Standard SI names like volt, amp, etc.
- Standard SI prefixes, mvolt, namp, etc.
- Some short names: mV, ms, nA, etc.
- Quantity class, derived from float
- Arrays with units
  - qarray class, derived from numpy.ndarray
  - Many scipy/numpy functions have Brian versions
  - Implementation is not complete yet
  - Performance issues

# Defining a neuron model: Equations

- Equations, each of one of the following forms:
  - $dx/dt = -x/\tau$  : `volt` (*differential equation*)
  - $y = x*x$  : `volt2` (*equation*)
  - $z = y$  (*alias*)
  - `w` : `unit` (*parameter*)
- See documentation for class `Equations`
- Equations can be a string, or `Equations` object, write `eqs=Equations(stringeqs)`
- Can also get equations from `brian.library`, but won't cover this here (see examples)

# Equations in action

```
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
'''

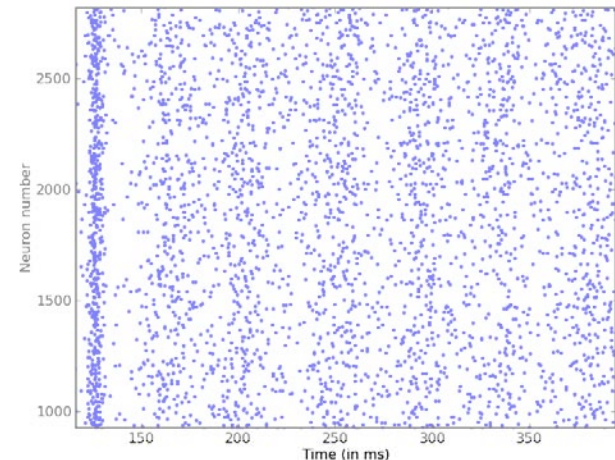
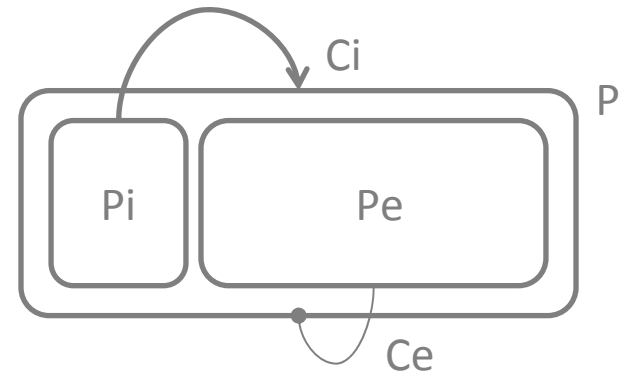
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Defining a neuron model: Model

- `mod=Model(equations=..., threshold=..., reset=...[, refractory=...])`
- Threshold
  - Single value, e.g. `-55*mV`
  - Function, e.g. `lambda V, Vt: V>=Vt`
  - Threshold object, e.g. `EmpiricalThreshold`, see documentation
- Reset
  - Single value, e.g. `-70*mV`
  - Function `f(G, spikes)`, where `G` is a `NeuronGroup` and `spikes` is a list of the indices of the neurons in `G` which have fired a spike, see adaptive threshold example
  - Standard reset object, e.g. `VariableReset`, see documentation
- Refractory, optional refractory period after reset
- Advanced: specify a `StateUpdater` instead of equations

# Model in action (sort of)

```
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
'''

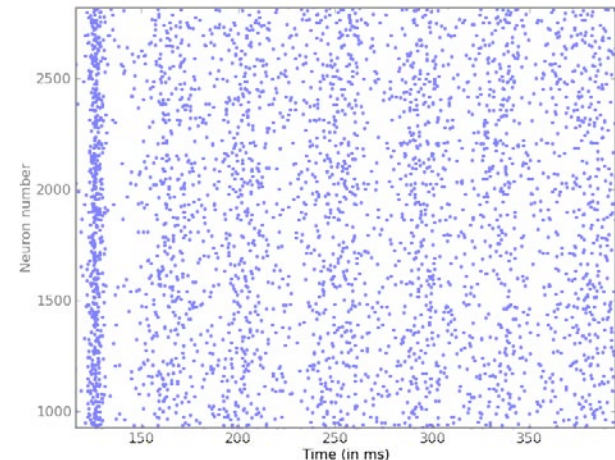
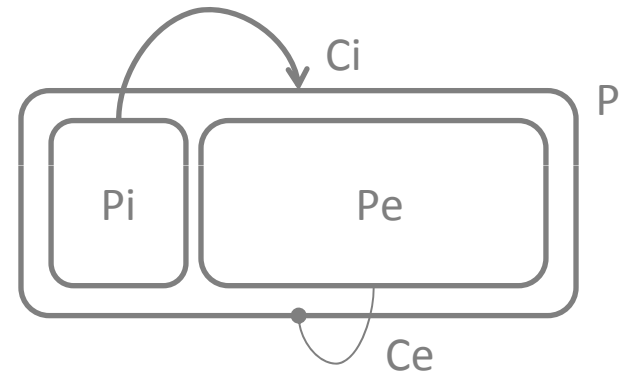
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Create groups of neurons

- `G=NeuronGroup(N, model, [max_delay=...])`
- Subgroups `Ge=G.subgroup(Ne)`,  
`Gi=G.subgroup(Ni)`
- Standard groups:
  - `PoissonGroup(N, rates)` where `rates` is a scalar, array, or function `f(t)` returning a scalar or array
  - `PulsePacket(t, n, sigma)` group of `n` neurons each firing one spike at times distributed `N(t,sigma)`
  - `SpikeGeneratorGroup(N, spiketimes)` group of `N` neurons firing spikes at `spiketimes`, a list of pairs `(i, t)` indicating neuron `i` fires at time `t`
  - `MultipleSpikeGeneratorGroup` (see docs)

# NeuronGroup in action

```
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
'''

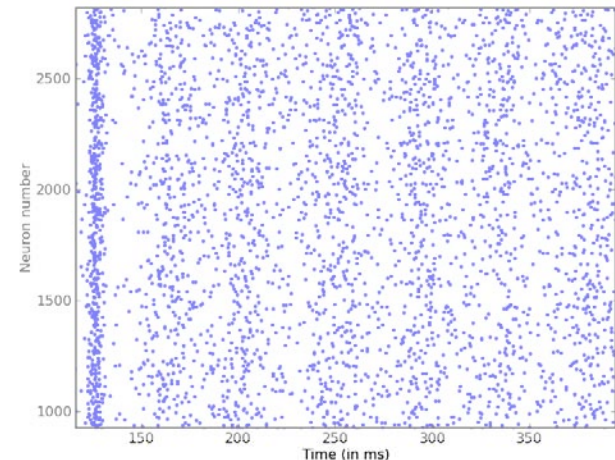
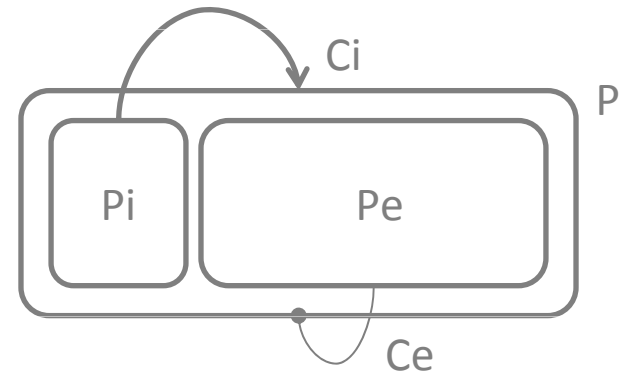
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Create synaptic connections:

## Connection object

- `C=Connection(src, target, statevar[, delay[, modulation[, structure]])`
- When a neuron in `src` fires, the variable `statevar` in `target` is increased by synaptic weight
- `modulation` is for short-term plasticity, variable that scales synaptic weights
- `structure` is matrix structure, can be 'sparse' or 'dense', you need dense for STDP (experimental)

# Connection in action

```
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
'''

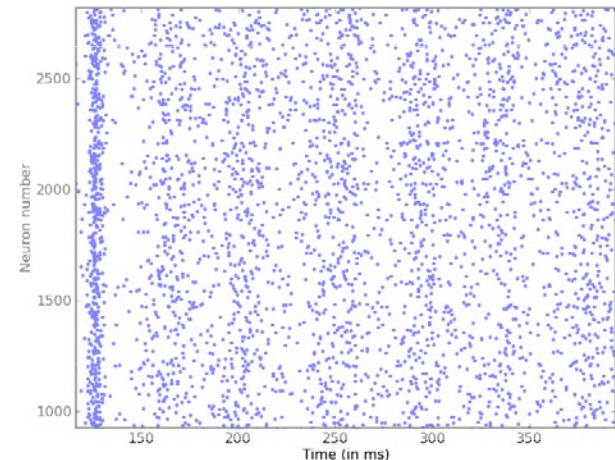
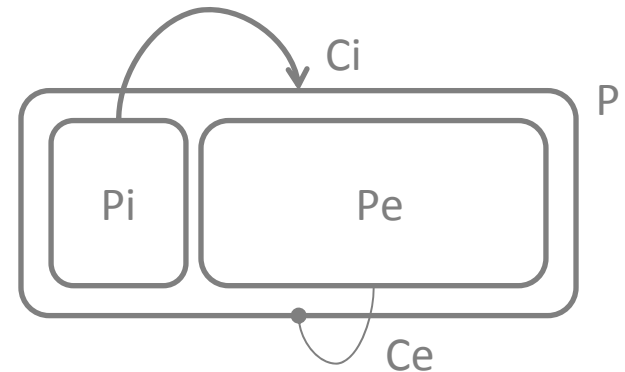
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Create synaptic connections: build weight matrix

- `C.connect_random(src, tgt, p, weight)`
  - `p` is independent probability of connection
  - weight can be a value or a function  $f(i, j)$
- `C.connect_full(src, tgt, weight)`
- `C.connect(src, tgt, W)`
  - `W` is a 2D array
- `C[i, j] = ...`
  - `i` is the source neuron, `j` is the target

# Building weight matrices in action

```
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
'''

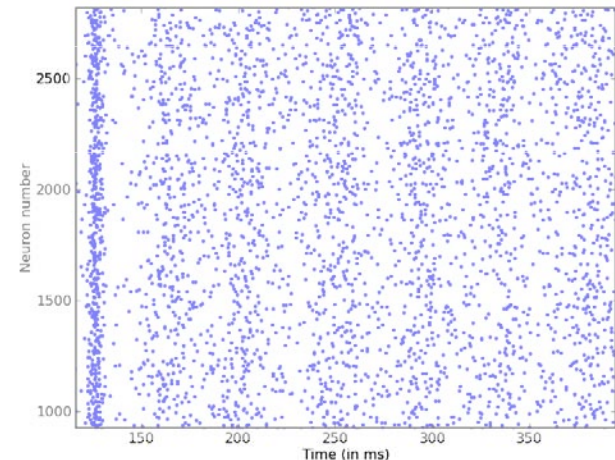
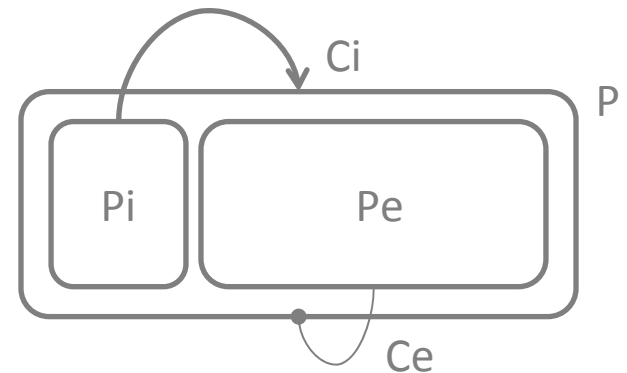
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Create monitors and operations

- Spikes
  - `M=SpikeMonitor(G[,function=...])`
    - function is `f(spikes)` for custom spike monitoring
  - `M=SpikeCounter(G)`
  - `M=PopulationSpikeCounter(G)`
- State variables
  - `M=StateMonitor(G,statevar[,record[,when='...']])`
    - `record=i` to record neuron `i`
    - `record=list` of ints to record neurons in list
    - `record=True` to record all neurons (lots of memory)
    - `when` is when to record during the update step (see docs)
- Other monitors (see docs)
  - `FileSpikeMonitor`
  - `ISIHistogramMonitor`
  - `PopulationRateMonitor`
- Network Operations (see examples)
  - `@network_operation` decorator
  - `@network_operation(when='...')`

# Monitors in action

```
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
'''

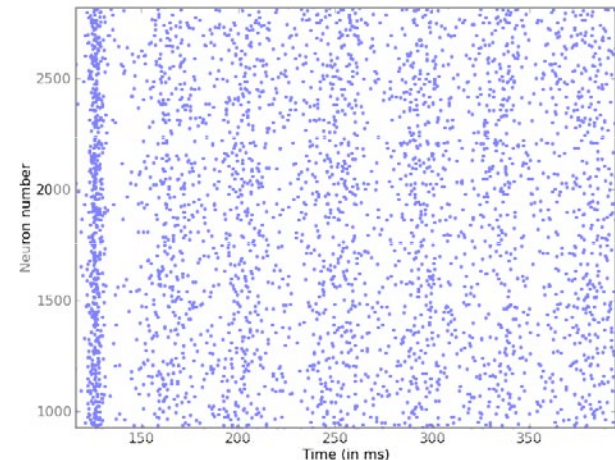
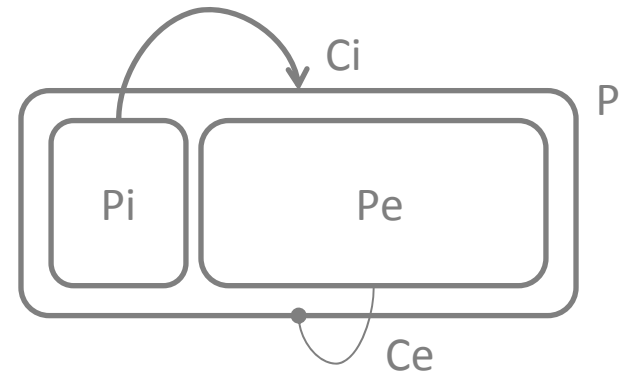
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Initialise and run the simulation

- `G.statevarname = vals, e.g.`
  - `G.V=Vr`
  - `G.V=Vr+(Vt-Vr)*rand(len(G))`
- `run(duration)`
  - Attempts to guess which objects should be included, see docs for 'magic functions'
- **Make a network object**
  - `net=Network(obj1, obj2, ...)`
  - `net.run(duration)`
- Can stop a run with the `stop()` command, or `net.stop()`

# Initialising and running in action

```
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
'''

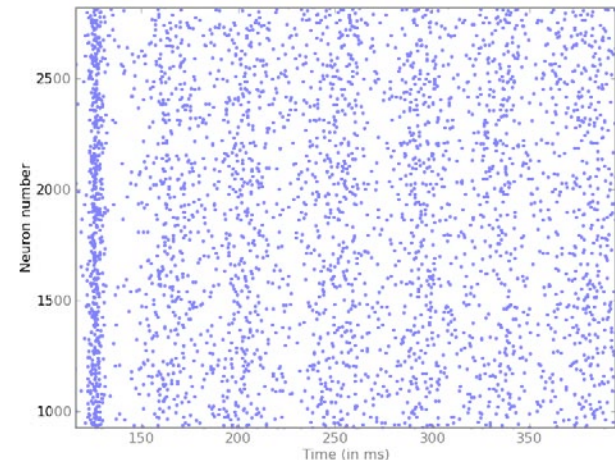
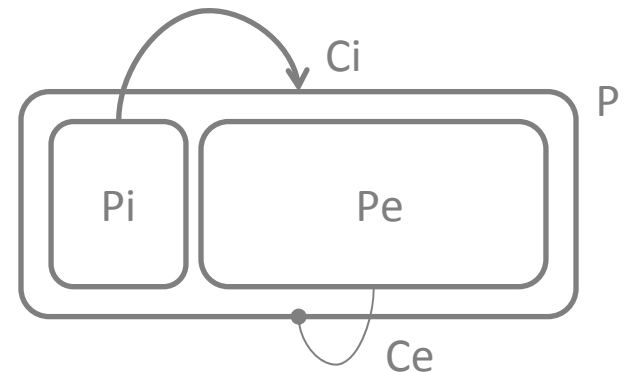
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Analyse and plot: getting data from monitors

- Data from `SpikeMonitor`
  - `M.spikes` list of pairs (i,t)
  - `M.nspikes` total number of spikes
- Data from `SpikeCounter`
  - `M.count` array of spike counts for each neuron
- Data from `StateMonitor`
  - `M.times` array of times at which recordings made
  - `M.mean`, `M.var`, `M.std` array of summary stats for every neuron for the recorded state variable
  - `M[i]` array of values of state variable for neuron i

# Analyse and plot: plotting

- `raster_plot(spikemon)` (see docs for options)
- PyLab
  - Mimics Matlab plotting commands
  - `figure`, `subplot`, `title`, `xlabel`, `ylabel`, `legend` functions like Matlab
  - `plot(xvals, yvals)`
    - e.g. `plot(M.times, M[0])`
  - `imshow(W)` for `W` a 2D array (e.g. weights)

# Plotting in action

```
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
'''

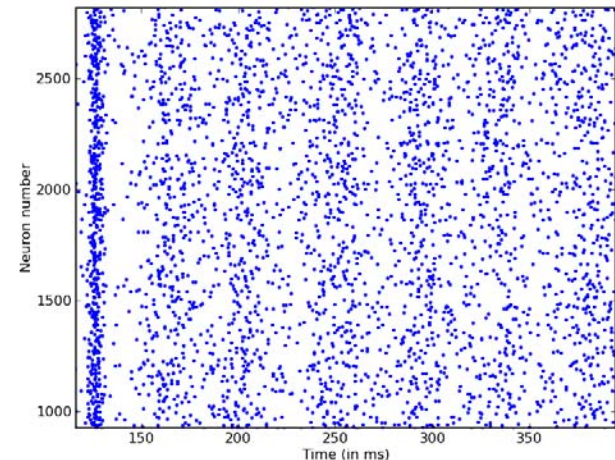
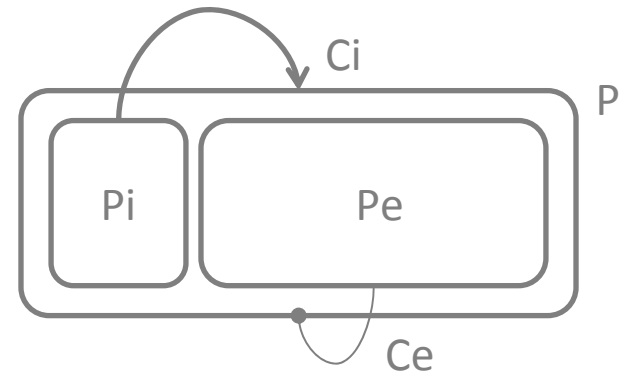
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Analyse and plot: analysis

- Use any function from SciPy or Pylab
- Note on units
  - `M.times`, `M[0]`, etc. return `qarray` objects with units
  - Some things work with units, e.g. `var(M.times)` will have units of `second*second`
  - Other things don't work
    - Convert to standard array by `asarray(x)`
    - Add underscore to attributes of monitor, e.g. `M.times_` is unitless
- Use NeuroTools? (Disclaimer: I haven't got round to trying it yet)

# The whole thing in action

```
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
'''

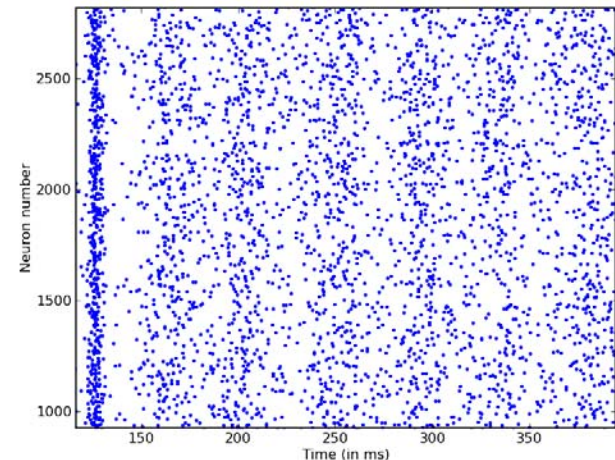
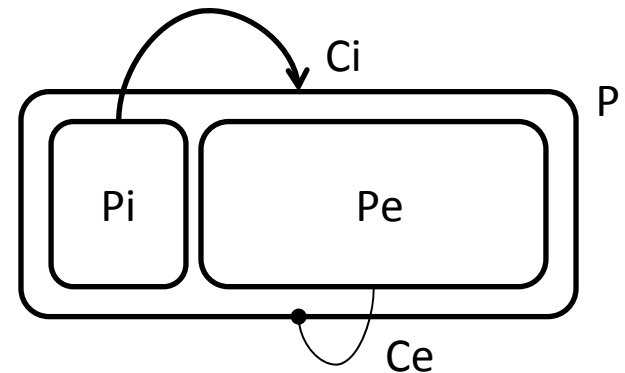
P=NeuronGroup(4000,model=eqs,
              threshold=-50*mV,reset=-60*mV)
P.v=-60*mV+10*mV*rand(len(P))
Pe=P.subgroup(3200)
Pi=P.subgroup(800)

Ce=Connection(Pe,P,'ge')
Ci=Connection(Pi,P,'gi')
Ce.connect_random(Pe, P, 0.02,weight=1.62*mV)
Ci.connect_random(Pi, P, 0.02,weight=-9*mV)

M=SpikeMonitor(P)

run(1*second)
raster_plot(M)
show()
```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$
$$\tau_e \frac{dg_e}{dt} = -g_e$$
$$\tau_i \frac{dg_i}{dt} = -g_i$$



# Questions?

Before we move on to interactive  
session part 1

# Interactive session part 1

- Write a leaky integrate and fire neuron
- Start with no synapses
- Set the reversal potential above threshold to make it produce spikes
- Plot the output
- Connect it to a second leaky integrate and fire neuron with reversal potential below threshold, and a longer time constant
- If you've finished that, try making larger groups with some random parameters and see what behaviour you can get
- Use Brian docs as reference <http://brian.di.ens.fr/docs>
- 20 mins... GO!

# Break

Solutions when we get back

# Solution: leaky IF producing spikes

```
from brian import *

tau = 10*ms
Vt = -55*mV
Vr = -70*mV
El = -54*mV

eqs = '''
    dV/dt = -(V-El)/tau : volt
'''

model = Model(equations=eqs,
              threshold=Vt, reset=Vr)

neuron = NeuronGroup(1, model)

M = StateMonitor(neuron, 'V', record=True)

run(200*ms)

plot(M.times, M[0])
show()
```

# Solution: two leaky IF neurons

```
from brian import *

tau = 10*ms
tau2 = 100*ms
Vt = -55*mV
Vr = -70*mV
El = -54*mV
weight = 5*mV

eqs = '''
    dV/dt = -(V-El)/tau : volt
    '''
eqs2 = '''
    dV/dt = -(V-Vr)/tau2 : volt
    '''

model = Model(equations=eqs,
              threshold=Vt, reset=Vr)
model2 = Model(equations=eqs2,
              threshold=Vt, reset=Vr)

neuron = NeuronGroup(1, model)
neuron2 = NeuronGroup(1, model2)

C = Connection(neuron, neuron2, 'V')
C.connect_full(neuron, neuron2, weight=weight)

M = StateMonitor(neuron, 'V', record=True)
M2 = StateMonitor(neuron2, 'V', record=True)

run(1*second)

plot(M.times, M[0])
plot(M.times, M2[0])
show()
```

# Flexibility: adaptive threshold

```
from brian import *

eqs='''
dv/dt = -v/(10*ms) : volt
dvt/dt = (10*mV-vt)/(10*ms) : volt
'''

def myreset(P, spikes):
    P.v[spikes]=0*mV
    P.vt[spikes]+=3*mV

IF = NeuronGroup(1, model=eqs,
                 reset=myreset,
                 threshold=lambda v, vt: v >= vt)
IF.vt=10*mV
PG = PoissonGroup(1, 500*Hz)

C = Connection(PG, IF, 'v')
C.connect_full(PG, IF, 3*mV)

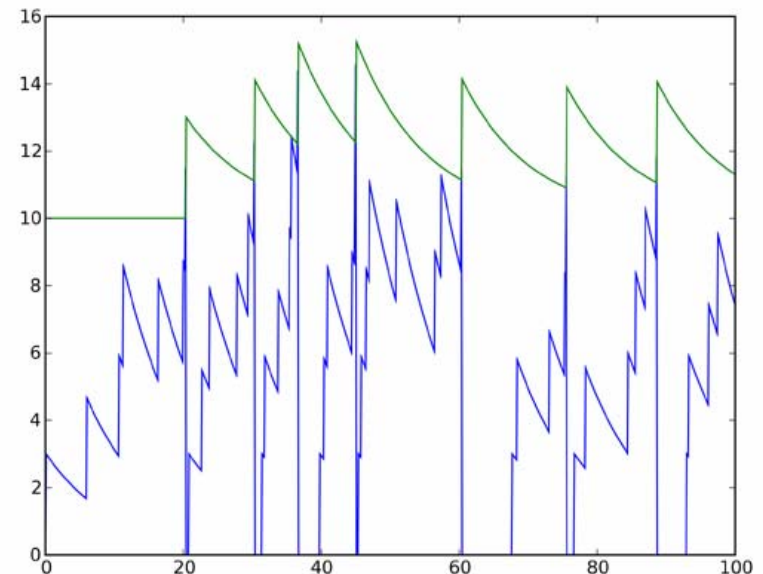
Mv = StateMonitor(IF, 'v', record=True)
Mvt = StateMonitor(IF, 'vt', record=True)

run(100*ms)

plot(Mv.times/ms, Mv[0]/mV)
plot(Mvt.times/ms, Mvt[0]/mV)

show()
```

- Threshold increases when spike arrives and decays
- Implemented as DE and user-defined reset and threshold functions



# Synfire chain (Diesmann et al. 1999)

```

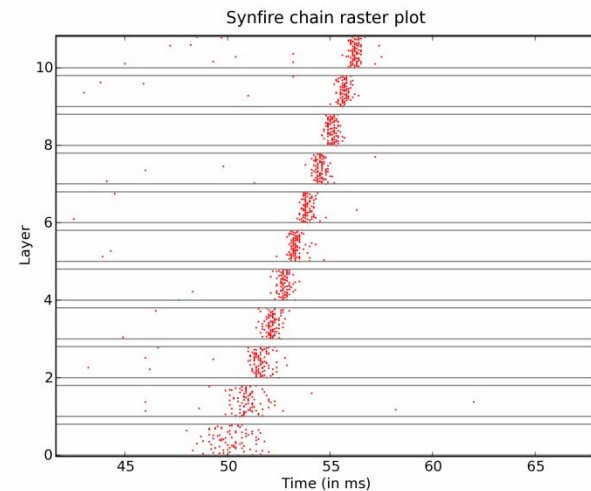
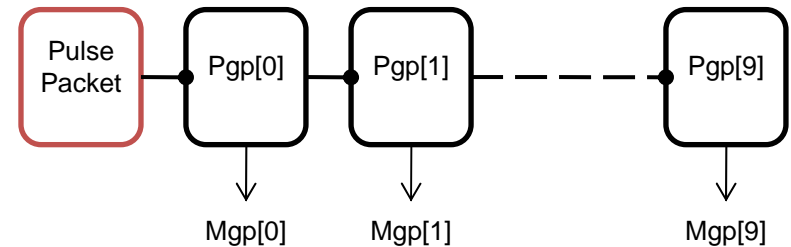
from brian import *
# Neuron model parameters
Vr = -70*mV
Vt = -55*mV
taum = 10*ms
taupsp = 0.325*ms
weight = 4.86 * mV
# Neuron model
eqs=Equations('''
dV/dt=(-(V-Vr)+x)*(1./taum) : volt
dx/dt=(-x+y)*(1./taupsp) : volt
dy/dt=-y*(1./taupsp)+25.27*mV/ms+
(39.24*mV/ms**0.5)*xi : volt
''')
# Neuron groups
P = NeuronGroup(N=1000, model=eqs,
threshold=Vt, reset=Vr, refractory=1*ms)
Pinput = PulsePacket(t=50*ms, n=85, sigma=1*ms)
# The network structure
Pgp = [ P.subgroup(100) for i in range(10) ]
C = Connection(P,P, 'y')
for i in range(9):
    C.connect_full(Pgp[i], Pgp[i+1], weight)
Cinput = Connection(Pinput, P, 'y')
Cinput.connect_full(Pinput, Pgp[0], weight)
# Record the spikes
Mgp = [SpikeMonitor(p, record=True) for p in Pgp]
Minput = SpikeMonitor(Pinput, record=True)
monitors = [Minput]+Mgp
# Setup the network, and run it
P.V = Vr + rand(len(P)) * (Vt-Vr)
run(100*ms)
# Plot result
raster_plot(showgrouplines=True, *monitors)
show()

```

$$\tau_m \frac{dV}{dt} = -(V - V_r) + x$$

$$\tau_{psp} \frac{dx}{dt} = -x + y$$

$$\tau_{psp} \frac{dy}{dt} = -y + \mu + \sigma \xi$$



# Note on performance

- If you have gcc installed, add this line to use some C++ optimised functions:
  - `set_global_preferences(use_weave=True)`
- When building connection matrices, prefer to use functions like `connect_random` and `connect_full` where possible, as they are vectorised
- Turn off the units system by including this as your first line:
  - `import brian_no_units`
- For large networks, should be possible to get close to performance of C++ (60-75% as fast in our tests)

# Interactive session part 2

- Small groups each working on their own project
- We can spend as little or as long as this as we want
- Experiment with examples provided
  - STDP (a bit complicated at the moment)
  - Synfire chains
  - Adaptive threshold
  - Hodgkin-Huxley neurons
  - Topographic map
  - Transient synchronisation
- If anyone has an idea for a network they'd like to try to build, we can do that
- Some other possible things to try
  - Modelling visual stimuli, maybe even something like orientation selectivity (possibly would take too long)
  - Something more complicated with synfire chains, maybe synfire chains with spatial connectivity
- We could also talk more about how to extend Brian to do new things

# Useful info

- Examples shown and slides available at <https://neuralensemble.org/cookbook/wiki/FacetsPythonCourse2008>
- Brian's website: <http://brian.di.ens.fr>
- Brian documentation: <http://brian.di.ens.fr/docs>