Supplement to
"A Simplifed Introduction to the NEURON Simulator"

Keiji Imoto
National Institute for Physiological Sciences (NIPS)
National Institutes of Natural Sciences (NINS)
Okazaki 444-8585, Japan

04-Dec-2017

目次

Supplementary materials

1 Python source code

1.1 n01.py

```
#  n01.py:  a simple hh cell

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 200.0
v_init = -65.0

soma = h.Section()
soma.diam = 30.0
soma.L = 30.0
soma.insert('hh')

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

vv = h.Vector()             # membrane potential vector
tv = h.Vector()             # time stamp vector
vv.record(soma(0.5)._ref_v)
tv.record(h._ref_t)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
ax.set_ylim([-80,40])
ax.plot(tv.as_numpy(), vv.as_numpy())
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n01.pdf')
plt.show()
```

1.2 n02.py

```
# n02.py:  a simple hh cell, with depolarized equilibrium potential
#          for leak current
from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 200.0
v_init = -65.0

soma = h.Section()
soma.diam = 30.0
soma.L = 30.0
soma.insert('hh')
soma.el_hh = -30.0  # <== changed

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

vv = h.Vector()              # membrane potential vector
tv = h.Vector()              # time stamp vector
vv.record(soma(0.5)._ref_v)
tv.record(h._ref_t)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
ax.set_ylim([-80,40])
ax.plot(tv.as_numpy(), vv.as_numpy())
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n02.pdf')
plt.show()
```

1.3 n03.py

```
# n03.py:  a simple hh cell,current injection

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 200.0
v_init = -65.0

soma = h.Section()
soma.diam = 30.0
soma.L = 30.0
soma.insert('hh')

ic = h.IClamp(0.5, sec=soma)
ic.delay = 20.0   # ms
ic.dur   = 100.0  # ms
ic.amp   = 0.2     # nA

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

vv = h.Vector()      # membrane potential vector
tv = h.Vector()      # time stamp vector
cv = h.Vector()      # current size
vv.record(soma(0.5)._ref_v)
tv.record(h._ref_t)
cv.record(ic._ref_i)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
ax.set_ylim([-80,40])
ax.plot(tv.as_numpy(), vv.as_numpy())
ax.plot(tv.as_numpy(), 100.0*cv.as_numpy())
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n03.pdf')
plt.show()
```

1.4 n04.py

```python
# n04.py: soma + dendrite

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 200.0
v_init = -65.0

soma = h.Section()
soma.diam = 30.0
soma.L = 30.0
soma.L = 30.0
soma.diam = 30.0
soma.nseg = 1
soma.insert('hh')

ap_dend = h.Section()
ap_dend.L = 500.0
ap_dend.diam = 2.0
ap_dend.nseg = 23
ap_dend.insert('hh')
ap_dend.gnabar_hh = 0.012
ap_dend.gkbar_hh = 0.0036
ap_dend.gl_hh = 0.00003

ap_dend.connect(soma, 1.0, 0.0)

ic = h.IClamp(0.5, sec=soma)
ic.delay = 20.0   # ms
ic.dur   = 100.0  # ms
ic.amp   = 0.5      # nA

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
vs = h.Vector()
va1 = h.Vector()
va2 = h.Vector()
cv = h.Vector()

tv.record(h._ref_t)
vs.record(soma(0.5)._ref_v)
va1.record(ap_dend(0.1)._ref_v)
va2.record(ap_dend(0.9)._ref_v)
cv.record(ic._ref_i)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
#ax.set_xlim([18.0,30.0])
ax.plot(tv.as_numpy(), vs.as_numpy())
```

```
ax.plot(tv.as_numpy(), va1.as_numpy())
ax.plot(tv.as_numpy(), va2.as_numpy())
ax.plot(tv.as_numpy(), 10.0*cv.as_numpy())
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n04a.pdf')
plt.show()
```

1.5 n05.py

```
# n05.py

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 25
v_init = -65.0

soma = h.Section()
soma.L = 30.0
soma.diam = 30.0
soma.nseg = 1
soma.insert('hh')

ap_dend = h.Section()
ap_dend.L = 500.0
ap_dend.diam = 2.0
ap_dend.nseg = 23

ap_dend.insert('hh')
ap_dend.gnabar_hh = 0.012
ap_dend.gkbar_hh = 0.0036
ap_dend.gl_hh = 0.00003

ap_dend.connect(soma, 1.0, 0)

# synaptic input
esyn = h.AlphaSynapse(0.5, sec = ap_dend)
esyn.onset = 5
esyn.tau = 0.1
esyn.gmax = 0.05
esyn.e = 0.0

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
vs = h.Vector()
va1 = h.Vector()
va2 = h.Vector()

tv.record(h._ref_t)
vs.record(soma(0.5)._ref_v)
va1.record(ap_dend(0.1)._ref_v)
va2.record(ap_dend(0.9)._ref_v)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
ax.plot(tv.as_numpy(), vs.as_numpy())
ax.plot(tv.as_numpy(), va1.as_numpy())
ax.plot(tv.as_numpy(), va2.as_numpy())
```

```
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n05.pdf')
plt.show()
```

1.6 n06.py

```python
# n06.py excitatory and inhibitory inputs

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 40.0
h.dt = 0.01
v_init = -65

soma = h.Section()
soma.L = 30.0
soma.diam = 30.0
soma.nseg = 1
soma.insert('hh')

ap_dend = h.Section()
ap_dend.L = 500.0
ap_dend.diam = 2.0
ap_dend.nseg = 23
ap_dend.insert('hh')
ap_dend.gnabar_hh = 0.012
ap_dend.gkbar_hh = 0.0036
ap_dend.gl_hh = 0.00003

ap_dend.connect(soma, 1.0, 0)

esyn = h.AlphaSynapse(0.1, sec = ap_dend)
esyn.onset = 10
esyn.tau   = 0.1
esyn.gmax  = 0.04
esyn.e     = 0.0

isyn_p = h.AlphaSynapse(0.05, sec = ap_dend)
isyn_p.onset = 5
isyn_p.tau = 3.0
isyn_p.e = -70 #v_init

isyn_d = h.AlphaSynapse(0.95, sec = ap_dend)
isyn_d.onset = 5
isyn_d.tau = 3.0
isyn_d.e = -70 #v_init

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
vs = h.Vector()
tv.record(h._ref_t)
vs.record(soma(0.5)._ref_v)

tr = []
tt = []

for i in range(3):
```

```
    if i == 0:
        isyn_p.gmax = 0.0
        isyn_d.gmax = 0.0
    if i == 1:
        isyn_p.gmax = 0.03
        isyn_d.gmax = 0.0
    if i == 2:
        isyn_p.gmax = 0.0
        isyn_d.gmax = 0.03

    h.finitialize(v_init)
    h.fcurrent()
    while h.t < tstop:
        cvode.fixed_step()

    tt.append(np.copy(tv.as_numpy()))
    tr.append(np.copy(vs.as_numpy()))

ax = plt.subplot()
ax.set_ylim([-80,40])
ax.plot(tt[0], tr[0])
ax.plot(tt[1], tr[1])
ax.plot(tt[2], tr[2])
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n06.pdf')
plt.show()
```

1.7 n07.py

```
# n07.py

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 40.0
h.dt = 0.01
v_init = -65

soma = h.Section()
soma.L = 30.0
soma.diam = 30.0
soma.nseg = 1
soma.insert('hh')

ap_dend = h.Section()
ap_dend.L = 500.0
ap_dend.diam = 2.0
ap_dend.nseg = 23
ap_dend.insert('hh')
ap_dend.gnabar_hh = 0.012
ap_dend.gkbar_hh = 0.0036
ap_dend.gl_hh = 0.00003

ap_dend.connect(soma, 1.0, 0)

esyn = h.AlphaSynapse(0.1, sec = ap_dend)
esyn.onset = 10
esyn.tau   = 0.1
esyn.gmax  = 0.04
esyn.e     = 0.0

isyn = [h.AlphaSynapse(0.1*i+0.05, sec = ap_dend) for i in range(10)]
for i in range(10):
    isyn[i].onset = 5
    isyn[i].tau = 3.0
    isyn[i].e = -70 #v_init

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
vs = h.Vector()
tv.record(h._ref_t)
vs.record(soma(0.5)._ref_v)

tr = []
tt = []

# synaptic input
for j in range(-1,10):    # -1 for excitatory input only
    for i in range(10):
        if i == j:
            isyn[i].gmax = 0.02
```

```
        else:
            isyn[i].gmax = 0

    h.finitialize(v_init)
    h.fcurrent()
    while h.t < tstop:
        cvode.fixed_step()

    tt.append(np.copy(tv.as_numpy()))
    tr.append(np.copy(vs.as_numpy()))

ax = plt.subplot()
ax.set_xlim([5.0, 20])
ax.set_ylim([-80,40])
#ax.set_xlim([8,14])
#ax.set_ylim([-70,-50])
for j  in range(11):
    ax.plot(tt[j], tr[j])
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n07.pdf')
#plt.savefig('../figs/n07b.pdf')
plt.show()
```

1.8 n08.py

```
# n08.py: connecting neurons

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

class HHneuron():  # class definition
    def __init__(self):
        self.soma = h.Section()
        self.ap_dend = h.Section()

        self.soma.L = 30
        self.soma.diam = 30
        self.soma.nseg = 1
        self.soma.insert('hh')

        self.ap_dend.L = 500
        self.ap_dend.diam = 2
        self.ap_dend.nseg = 23
        self.ap_dend.insert('hh')
        self.ap_dend.gnabar_hh = 0.012
        self.ap_dend.gkbar_hh  = 0.0036
        self.ap_dend.gl_hh = 0.00003

        self.ap_dend.connect(self.soma, 1, 0)
        self.esyn = h.Exp2Syn(0.5, sec=self.ap_dend)
        self.esyn.tau1 = 0.5
        self.esyn.tau2 = 1.0
        self.esyn.e = 0

tstop = 100.0
v_init = -65.0

# 2 cells
hh_neuron = [HHneuron() for i in range(2)]

# synapse
stim = h.NetStim(0.5)
stim.interval = 20
stim.number = 3
stim.start = 20
stim.noise = 0

# connections
nclist = []
nclist.append(h.NetCon(stim, hh_neuron[0].esyn, 0.0, 0, 0.02))
nclist.append(h.NetCon(hh_neuron[0].soma(0.5)._ref_v, hh_neuron[1].esyn, 10, 1, 0.02))

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
vs0 = h.Vector()
vs1 = h.Vector()

```

```
tv.record(h._ref_t)
vs0.record(hh_neuron[0].soma(0.5)._ref_v)
vs1.record(hh_neuron[1].soma(0.5)._ref_v)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
ax.plot(tv.as_numpy(), vs0.as_numpy())
ax.plot(tv.as_numpy(), vs1.as_numpy())
plt.xlabel('Time (ms)')
plt.ylabel('Potential (mV)')
plt.savefig('../figs/n08.pdf')
plt.show()
```

1.9 n09.py

```
#  n09.py: voltage clamp

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop = 200.0
h.dt = 0.01
v_init = -65.0

soma = h.Section()
soma.L = 30.0
soma.diam = 30.0
soma.nseg = 1
soma.insert('hh')

esyn = h.Exp2Syn(0.5, sec=soma)
vcl = h.VClamp(0.5, sec=soma)

stim = h.NetStim(0.5)
stim.interval = 50
stim.number = 2
stim.start = 50
stim.noise = 0

vcl.dur[0] = 10.0
vcl.dur[1] = 10.0
vcl.dur[2] = 180.0
vcl.amp[0] = v_init
vcl.amp[1] = v_init
vcl.amp[2] = v_init
vcl.gain = 1000.0
vcl.tau1 = 0.01
vcl.tau2 = 0.01

nclist = []
nclist.append(h.NetCon(stim, esyn, 0.0, 0, 0.005))

tv = h.Vector()
vv = h.Vector()
cv = h.Vector()

tv.record(h._ref_t)
vv.record(soma(0.5)._ref_v)
cv.record(vcl._ref_i)

h.finitialize(v_init)
neuron.run(tstop)

ax = plt.subplot()
ax.set_ylim([-400,100])
ax.plot(tv.as_numpy(), 1000.0*cv.as_numpy())
ax.plot(tv.as_numpy(),vv.as_numpy())
plt.xlabel('Time (ms)')
plt.ylabel('Current')
plt.savefig('../figs/n09.pdf')
```

```
plt.show()
```

1.10 n10.py

```
# n10.py
# iv in voltage-clamp mode

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

tstop  = 50.0
h.dt   = 0.01
vstart = -100.0
vstep  = 10.0
ntrace = 20
npts   = int(tstop/h.dt)+1

soma = h.Section()
soma.diam = 30.0
soma.L = 30.0
soma.nseg = 1
soma.insert('pas')

vc = h.VClamp(0.5, sec=soma)
vc.dur[0] = 10.0
vc.dur[1] = 10.0
vc.dur[2] = 30.0

syn = h.Exp2Syn(0.5, sec=soma)
syn.e = 0

ns = h.NetStim()
ns.number = 1
ns.start = 10.0
ns.noise = 0

nc = []
nc.append(h.NetCon(ns, syn, 0, 0, 0.001))   # 1 nS

tv = h.Vector()
tv.record(h._ref_t)
cv = h.Vector()
cv.record(vc._ref_i)

taxes = []
traces = []
for i in range(ntrace):
    v_init = vstart + vstep * i
    vc.amp[0]=v_init
    vc.amp[1]=v_init
    vc.amp[2]=v_init
    soma.e_pas = v_init

    h.finitialize(v_init)
    h.fcurrent()
    neuron.run(tstop)
    traces.append(np.copy(cv.as_numpy()))
    taxes.append(np.copy(tv.as_numpy()))

```

```
#tt = tv.as_numpy()

ax = plt.subplot()
ax.set_ylim([-0.10,0.10])
for i in range(ntrace):
    ax.plot(taxes[i], traces[i])
#    ax.plot(tt, traces[i])
plt.xlabel('Time (ms)')
plt.ylabel('Current')

plt.savefig('../figs/n10.pdf')
plt.show()
```

1.11 n11.py

```
#  n11.py
import subprocess
import numpy as np
import neuron
from neuron import h
import matplotlib.pyplot as plt

tstop = 200.0
h.dt = 0.01
v_init = -65.0

soma = h.Section()
dend = [h.Section() for i in range(10)]

soma.diam = 30.0
soma.L = 30.0
soma.Ra = 100.0
soma.nseg = 1
soma.insert('hh')
vcl = h.VClamp(0.5, sec=soma)

for i in range(10):
    dend[i].L = 300.0
    dend[i].Ra = 100.0
    dend[i].nseg = 21
    dend[i].diam = 2.0
    dend[i].insert('hh')
    dend[i].gl_hh *= 2.3
    dend[i].connect(soma, 1.0, 0.0)

esyn = h.Exp2Syn(0.75, sec=dend[0])

# stimulation
stim = h.NetStim(0.5)
stim.interval = 50.0
stim.number = 2
stim.start = 50.0
stim.noise = 0

# synaptic connections
nclist = []
nclist.append(h.NetCon(stim, esyn, 0.0, 0, 0.001))

vcl.dur[0] = 10.0
vcl.dur[1] = 10.0
vcl.dur[2] = 180.0
vcl.amp[0] = v_init
vcl.amp[1] = v_init
vcl.amp[2] = v_init
vcl.gain = 1000.0
vcl.tau1 = 0.1
vcl.tau2 = 0.1

tv = h.Vector()
vs = h.Vector()
vd = h.Vector()
cv = h.Vector()
```

```

tv.record(h._ref_t)
vs.record(soma(0.5)._ref_v)
vd.record(dend[0](0.75)._ref_v)
cv.record(vcl._ref_i)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
ax.set_ylim([-70.0, -35.0])
ax.plot(tv.as_numpy(), 100*cv.as_numpy())
ax.plot(tv.as_numpy(), vs.as_numpy())
ax.plot(tv.as_numpy(), vd.as_numpy())
plt.xlabel('Time (ms)')
plt.ylabel('Current')
plt.savefig('../figs/n11.pdf')
plt.show()
```

1.12 n12.py

```python
#  n12.py

import math
import numpy as np
import neuron
from neuron import h
import matplotlib.pyplot as plt

h.load_file("nrngui.hoc")
h.load_file("stdlib.hoc")
h.load_file("../models/tc200.geo")

nSyn  = 30
tstop = 100.0
dt    = 0.01
v_init= -65.0

# Properties of dendrites
dendritic = []
for sec in h.allsec():
    sec.insert('hh')
    sec.Ra = 100
    if sec.name() !=  'soma':
        dendritic.append(sec)
print('number of dend =', len(dendritic))

for sec in dendritic:
    d = 1.0e5 * math.sqrt(sec.diam/(4.0*h.PI*100.0*sec.Ra*sec.cm))
    sec.nseg = int((int((sec.L/(0.1*d))+0.9)/2)*2+1)

# total dendritic length
totalLength = 0.0
for sec in dendritic:
    totalLength += sec.L
print('Tolal dendrite length = ', totalLength)

# Shape Plot
sh = h.Shape(1)
sh.size(-150, 150, -150, 150)
# stimulation
stim = h.NetStim(0.5)
stim.interval = 50.0
stim.number = 2
stim.start = 20.0
stim.noise = 0

nclist = []
locvec = np.random.uniform(0.0, totalLength, nSyn)
esyn = []
nclist = []
for k in range(nSyn):
    l=0
    l1 = 0
    found = 0
    for sec in dendritic:
        if found==0:
            l1 += sec.L
```

```
            if locvec[k] < l1:
                lx = (locvec[k]-l)/sec.L
                e = h.Exp2Syn(lx, sec=sec)
                esyn.append(e)
                sh.color(2, sec=sec)
                sh.point_mark(esyn[k],3,'O',2)
                nclist.append( h.NetCon(stim, esyn[k], 0.0, 0, 0.001))
                found = 1
                break
            l = l1
sh.flush()
h.doEvents()
sh.printfile('../figs/n12shape.eps')

# main loop
cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
vs = h.Vector()
tv.record(h._ref_t)
vs.record(h.soma(0.5)._ref_v)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
#ax.set_ylim([-70.0, -35.0])
ax.plot(tv.as_numpy(), vs.as_numpy())
plt.savefig('../figs/n12.pdf')
plt.show()
```

1.13 n13.py

```python
#  n13.py
import numpy as np
import neuron
from neuron import h
import matplotlib.pyplot as plt

h.load_file('nrngui.hoc')
h.load_file("stdlib.hoc")
h.load_file('../models/tc200.geo')

tstop = 200.0
dt = 0.01
v_init = -65.0

nSyn=100      # number of synapses
mdel = 10.0   # mean of synaptic delay
sdel = 1.0    # sd of synaptic delay
w = 0.001     # synaptic weight
br = 5

def measureDist(sec):
    dx = 0.0
    sec1 = sec
    while True:
        sr = h.SectionRef(sec=sec1)
        if sr.has_parent() < 1.0:
            break
        dx += sec1.L
        sec1 = h.SectionRef(sec=sec1).parent
    return dx

def countBranch(sec):
    bx = 0
    sec1 = sec
    while True:
        sr = h.SectionRef(sec=sec1)
        if sr.has_parent() < 1.0:
            break
        bx += 1
        print(sec1.name())
        sec1 = h.SectionRef(sec=sec1).parent
    return bx

#--------------------------------
# Properties of dendrites

dendritic = []
for sec in h.allsec():
    sec.insert('hh')
    sec.Ra = 100.0
    if sec.name() != 'soma':
        dendritic.append(sec)

for sec in dendritic:
    d = h.lambda_f(100,sec=sec)
    sec.nseg = int(int((sec.L/(0.1*d))+0.9)/2*2+1)

```

```
dendriticN = []
total_length=0
for sec in dendritic:
    if countBranch(sec) >= br:
        dendriticN.append(sec)
        total_length += sec.L

print('total length of ">=', br, 'th-branch" dendrites = ', total_length)

# Shape Plot
sh = h.Shape(1)

# stimulation
stim = h.NetStim(0.5)
stim.interval = 50.0
stim.number = 2
stim.start = 50.0
stim.noise = 0

# random generator
locvec = np.random.uniform(0, total_length, nSyn)
delvec = np.random.normal(mdel, sdel, nSyn)
nc = []
esyn = []
for k in range(nSyn-1):
    l = 0
    l1 = 0
    found = 0
    for sec in dendriticN:
        if found == 0:
            l1 += sec.L
            if l1 > locvec[k]:
                lx = (locvec[k]-l)/sec.L
                e = h.Exp2Syn(lx, sec=sec)
                esyn.append(e)
                sh.color(2, sec=sec)
                sh.point_mark(esyn[k],3,'O',2)
                nc.append(h.NetCon(stim, esyn[k], 0.0, delvec[k], w))
                found = 1
                break
            l = l1
sh.flush()
h.doEvents()
sh.printfile('../figs/n13shape.eps')
# main loop

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
vs = h.Vector()
tv.record(h._ref_t)
vs.record(h.soma(0.5)._ref_v)

h.finitialize(v_init)
h.fcurrent()

print('Simulation Start')
neuron.run(tstop)
```

```
ax = plt.subplot()
ax.plot(tv.as_numpy, vs.as_numpy)
plt.savefig('../figs/n13.pdf')
plt.show()
```

1.14 n14.py

```
# n14.py

import numpy as np
import neuron
from neuron import h
import matplotlib.pyplot as plt

tstop = 40.0

soma = h.Section()
soma.insert('m01')

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

tv = h.Vector()
zv = h.Vector()
tv.record(h._ref_t)
zv.record(soma(0.5)._ref_z_m01)

h.finitialize()
#h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
ax.plot(tv.as_numpy(), zv.as_numpy())
plt.xlabel('Time')
plt.ylabel('Z')
plt.savefig('../figs/n14.pdf')
plt.show()
```

1.15 n15.py

```
# n15.py  Markov model of Na channel

import numpy as np
import neuron
from neuron import h
import matplotlib.pyplot as plt

nstates = 9

tstop = 10.0
v_init = -100.0

c = h.Section()
c.insert('mna')
h.dt = 0.05

vc = h.VClamp(0.5, sec=c)
vc.dur[0] = 2.0
vc.dur[1] = 5.0
vc.dur[2] = 3.0
vc.amp[0] = v_init
vc.amp[1] = 0.0
vc.amp[2] = v_init

tv = h.Vector()
zv = [h.Vector() for i in range(nstates)]

tv.record(h._ref_t)
for i in range(nstates):
    s = 'zv[%1d].record(c(0.5)._ref_z%1d_mna)'% (i, i)
    exec(s)

h.finitialize(v_init)
h.fcurrent()
neuron.run(tstop)

ax = plt.subplot()
for i in range(nstates):
    ax.plot(tv.as_numpy(), zv[i].as_numpy())

plt.xlabel('Time')
plt.ylabel('Probability')
plt.savefig('../figs/n15.pdf')
plt.show()
```

1.16 n16.py

```
# n16.py  Markov model of Na channel
#         Voltage-dependent activation

import numpy as np
import neuron
from neuron import h
import matplotlib.pyplot as plt

nstates = 9

tstop  = 20.0
h.dt   = 0.01

v_init = -100.0
vstart = -100.0
vstep  = 10.0
ntraces = 20
npts   = int(tstop/h.dt)+1
ena = 50.0

c = h.Section()
c.insert('mna')

vc = h.VClamp(0.5, sec=c)
vc.dur[0] = 5.0
vc.dur[1] = 10.0
vc.dur[2] = 5.0
vc.amp[0] = v_init
vc.amp[1] = v_init
vc.amp[2] = v_init

tv = h.Vector()
tv.record(h._ref_t)
gv = h.Vector()
gv.record(c(0.5)._ref_z3_mna)
vv = h.Vector()
vv.record(c(0.5)._ref_v)

traces = []
vtra = []
for i in range(ntraces):
    vc.amp[1] = vstart + vstep * i
    h.finitialize(v_init)
    h.fcurrent()
    neuron.run(tstop)

    traces.append(np.copy(gv.as_numpy()))
    vtra.append(np.copy(vv.as_numpy()))

tt = tv.as_numpy()

f, ax = plt.subplots(2, sharex=True)
for i in range(ntraces):
    ax[0].plot(tt, traces[i])
ax[0].set_ylabel('Open probability')

for i in range(ntraces):
```

```
    ax[1].plot(tt, traces[i]*(vtra[i]-ena))
ax[1].set_xlabel('Time (ms)')
ax[1].set_ylabel('Current')

plt.savefig('../figs/n16.pdf')
plt.show()
```

1.17 n17.py

```
# n17.py

import neuron
from neuron import h
import numpy as np
import matplotlib.pyplot as plt

#h.load_file ("nrngui.hoc")
h.load_file ("../models/cellModel.hoc")

tstop = 100
v_init = -65
ncells = 20

cells = []
for i in range(ncells):
    cells.append(h.BallStickCell( ))

nclist = []
for i in range(len(cells)):
    target = cells[(i+1) % ncells]   # next cell
    nc = cells[i].connect2target(target.synlist[0])
    nclist.append(nc)
    nc.delay = 1
    nc.weight[0] = 0.01

#external stimulus to syn of cells[0]
stim = h.NetStim()
stim.number = 1
stim.start = 0
ncstim = h.NetCon(stim, cells[0].synlist[0])
ncstim.delay=0
ncstim.weight[0] = 0.01

tvec  = h.Vector( )
idvec = h.Vector( )
for i in range( len(cells) ):
    nc = cells[i].connect2target(None)
    nc.record (tvec, idvec, i)

# Simulation control
h.finitialize(v_init)
neuron.run(tstop)

print ("\ntime\t cell\n")
for i in range(int(tvec.size())):
    print (i, tvec.x[i], idvec.x[i])
print("tvec.size=",tvec.size())

ax1 = plt.subplot()
ax1.plot(tvec.as_numpy(), idvec.as_numpy(),'|')
plt.xlabel('Time (ms)')
plt.ylabel('Cell number')
plt.savefig('../figs/n17.pdf')
plt.show()

"""
```

```
spikes = [np.empty(0) for i in range(ncells)]
for i in range(int(tvec.size())):
    j = int(idvec.x[i])
    spikes[j] = np.append(spikes[j],tvec.x[i])
ax1 = plt.subplot()
ax1.eventplot(spikes, colors=[[1.0, 0, 0]])
plt.show()
"""
```

1.18 n18.py

```
# n18.py  Random network

import neuron
from neuron import h
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

h.load_file ("../utils/ranstream.hoc")
h.load_file('../models/cellModel.hoc')

tstop = 200
h.dt = 0.01
vinit = -65
NCELL = 200  # total number of cells
C_E = 3      # # of excitatory connections received by each cell

connect_random_low_start_ = 1 # low seed for mcell_ran4_init ( )
cells = []
ranlist = []   # list of the random number sequences
h.mcell_ran4_init(connect_random_low_start_)

# create cells
for i in range(NCELL):        # target (postsynaptic) cell
    cells.append(h.BallStickCell( ))
    ranlist.append(h.RandomStream(i))

# make connection
#   no self-connection
#   only one connection from any source
nclist = []
u = np.zeros(NCELL, dtype=int)
for i in range(NCELL): # target cell
    syn = cells[i].synlist.object(0)
    rs = ranlist[i]
    rs.start ( )
    rs.r.discunif(0, NCELL-1) # returns int in range 0..NCELL-1
    u[:] = 0   # u.x[i]==1 means spike source i has already been chosen
    nsyn = 0
    while nsyn < C_E:
        r = int(rs.repick( ))
        if (r != i) and (u[r] == 0):
            nc = cells[r].connect2target(syn)
            nclist.append (nc)
            nc.delay = 1
            nc.weight[0] = 0.01
            u[r] = 1
            nsyn += 1
        #    print("cell %2d -> cell %2d (syn %d)" % (r, i, 0))

print(len(nclist), " connections")

stim = h.NetStim()
stim.number = 1
stim.start = 0
ncstim = h.NetCon (stim, cells[0].synlist[0])
ncstim.delay = 0
```

```
ncstim.weight[0] = 0.01

tvec = h.Vector( )
idvec = h.Vector( )
for i in range(len(cells)):
    nc = cells[i].connect2target(None)
    nc.record (tvec, idvec, i)
    nc.record (tvec, idvec, i)

#for trace
ttvec = h.Vector( )
vvec = h.Vector()
ttvec.record(h._ref_t)
vvec.record(cells[0].soma(0.5)._ref_v)

# Simulation control
h.finitialize(vinit)
neuron.run(tstop )

# Report simulation results
def spikeout ( ):
    print("tvec size = ", int(tvec.size()))
    print("\ntime\t cell\n")
    for i in range(int(tvec.size())):
        print(i, tvec.x[i], idvec.x[i])

# spikeout ( )

spikes = [np.empty(0) for i in range(NCELL)]
for i in range(int(tvec.size())):
    j = int(idvec.x[i])
    spikes[j] = np.append(spikes[j],tvec.x[i])

ax1 = plt.subplot()
ax1.eventplot(spikes, colors=[[1.0, 0, 0]])
plt.ylabel('Cell number')
plt.xlabel('Time (ms)')
plt.savefig('../figs/n18.pdf')
#plt.show(block=False)
#plt.pause(3)
plt.show()
```

1.19 n19.py

```
# n19.py

import neuron
from neuron import h
import numpy as np
import matplotlib.pyplot as plt

class InvlFire(object):
    def __init__(self):
        self.cell = h.Section()
        self.IF = h.IntervalFire(0.5, sec=self.cell)
        self.IF.tau  = 5.0
        self.IF.invl = 10.0
        self.synlist = h.List()


tstop = 100.0
h.dt = 0.1

ce = InvlFire()

# external stimulation
stim = h.NetStim()
stim.number = 40
stim.start = 25
stim.interval = 1

ncstim = h.NetCon(stim, ce.IF)
ncstim.delay= 0
ncstim.weight[0] = 0.07

# record for spikes
nc = h.NetCon(ce.IF, None)
stvec = h.Vector( )
nc.record(stvec)

# run control
h.finitialize()
tt = []
mm = []
MM = []
while h.t < tstop:
    tt.append(h.t)
    mm.append(ce.IF.m)
    MM.append(ce.IF.M())
    h.fadvance()

f, ax = plt.subplots(2, sharex=True)
ax[0].plot(tt, mm)
ax[0].plot(tt, MM)
ax[0].set_ylabel('Values of m and M')

ax[1].eventplot(stvec.as_numpy())
ax[1].set_xlabel('Time (ms)')
ax[1].set_ylabel('Events')

plt.savefig('../figs/n19.pdf')
```

```
plt.show()
```

1.20 n20.py

```
# n20.py

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt

class InvlFire(object):
    def __init__(self):
        self.cell = h.Section()
        self.IF = h.IntervalFire(0.5, sec=self.cell)
        self.IF.tau  = 5.0
        self.IF.invl = 10.0

tstop = 200.0
ncells =600
exRatio  = 0.8
inhRatio = 1.0 - exRatio
exCells = int(exRatio*ncells)
inhCells = ncells - exCells
conRateE = 0.005
conRateI = 0.05
synWeightE = 0.01
synWeightI = -0.18

tbin = 2.0
nbin = int((tstop/tbin))+1

print("excitatory cells: ", exCells)
print("inhibitory cells: ", inhCells)

print("estimated excitatory synapses: ", exCells * ncells * conRateE)
print("estimated inhibitory synapses: ", inhCells * ncells * conRateI)


cells = [InvlFire() for i in range(ncells)]
print(len(cells))

"""
# clock-wise connections
nclist = []
for i in range(ncells):
    target = cells[(i+1) % ncells]   # next cell
    nc = h.NetCon(cells[i].IF, target.IF)
    nclist.append(nc)
    nc.delay = 5
    nc.weight[0] = 0.5
"""
# random connections
nclist = []
used = np.zeros(ncells)
for i in range(ncells):
    if(i < exCells):
        maxConnection = int(ncells*conRateE)
        synWeight = synWeightE
    else:
        maxConnection = int(ncells*conRateI)
```

```python
        synWeight = synWeightI

    ncon = 0
    j = 0
    used[:] = 0
    trand = np.random.randint(0, ncells, size=ncells)
    while ncon < maxConnection:
        k = trand[j]
        j += 1
        if(used[k]==0):
#            if(i < excCells):
#                print(i," ==> ", k)
#            else:
#                print(i," --> ", k)

            nc = h.NetCon(cells[i].IF, cells[k].IF)
            nclist.append(nc)
            nc.delay = 3
            nc.weight[0] = synWeight
            used[k] = 1
            ncon = ncon+1

# randomize initial state
stim = h.NetStim()
stim.number   = 1
stim.start    = 0
stim.interval = 1
ncini = []
mr = np.random.uniform(0, 1.0, ncells)
for i in range(ncells):
    ncstim = h.NetCon(stim, cells[i].IF)
    ncstim.delay= 0
    ncstim.weight[0] = mr[i]
    ncini.append(ncstim)

# record for spikes
stvec  = h.Vector( )
idvec = h.Vector( )
for i in range( len(cells) ):
    nc = h.NetCon(cells[i].IF, None)
    nc.record (stvec, idvec, i)

# Simulation control
h.finitialize()
neuron.run(tstop)

#print ("\ntime\t cell\n")
#for i in range(int(tvec.size())):
#    print (i, tvec.x[i], idvec.x[i])
#print("tvec.size=",tvec.size())
hg = np.zeros(nbin, dtype=int)
hge = np.zeros(nbin, dtype=int)
hgi = np.zeros(nbin, dtype=int)
ht = np.linspace(0, tstop, nbin)
for i in range(int(stvec.size())):
    ix = int(stvec.x[i]/tbin)
    hg[ix] = hg[ix] + 1
    if(idvec.x[i]<excCells):
        hge[ix] = hge[ix] + 1
    else:
        hgi[ix] = hgi[ix] + 1
```

```
hge = hge.astype(float)/exCells
hgi = hgi.astype(float)/inhCells

f, ax = plt.subplots(3, sharex=True)
ax[0].plot(stvec.as_numpy(), idvec.as_numpy(),'.', markersize=2)
ax[0].set_ylabel('Cell number')
ax[1].plot(ht, hg)
ax[1].set_ylabel('No of spikes')
ax[2].plot(ht, hge)
ax[2].plot(ht, hgi)
ax[2].set_xlabel('Time (ms)')
ax[2].set_ylabel('Probability')

plt.savefig('../figs/n20.pdf')
plt.show()


"""
spikes = [np.empty(0) for i in range(ncells)]
for i in range(int(stvec.size())):
    j = int(idvec.x[i])
    spikes[j] = np.append(spikes[j],stvec.x[i])
ax1 = plt.subplot()
ax1.eventplot(spikes, colors=[[1.0, 0, 0]])
plt.show()
"""
```

1.21 n21.py

```

# n21.py collision experiment

from neuron import h
import neuron
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

nsegs = 10000
ulen = 5.0    # unit length in micro-meter
totalLength = ulen * nsegs
npts = nsegs + 1
h.dt = 0.025
dl = 1.0/float(nsegs)

fCollision = False #True

v_init = -65.0
tstop = 100.0

axon = h.Section()
axon.L = totalLength
axon.diam = 2.0
axon.nseg = npts
axon.insert("hh")

cvode = h.CVode()
cvode.active(1)
cvode.atol(1.0e-5)

ic1 = h.IClamp(0.0, sec=axon)
ic1.delay = 1.0 # ms ic.dur = 100.0 # ms
ic1.amp = 0.3  # nA
ic1.dur = 1.0

if fCollision == True:
    ic2 = h.IClamp(1.0, sec=axon)
    ic2.delay = 1.0 # ms ic.dur = 100.0 # ms
    ic2.amp = 0.3  # nA
    ic2.dur = 1.0


yy = []
h.finitialize(v_init)
h.fcurrent()

yv = np.zeros(npts)

while h.t < tstop:
    for i in range(npts):
        yv[i] = axon(dl*i).v
    yy.append(np.copy(yv))
    cvode.fixed_step()

ntraces = len(yy)

```

```python
fig, ax = plt.subplots(figsize=(10,2))
xv = np.linspace(0.0, totalLength, npts)
yv = np.zeros(npts)
line, = ax.plot(xv,yv)
ax.set_ylim(-100.0,50.0)
nframes = 5
def update(i):
    ydata = yy[nframes*i]
    line.set_ydata(ydata)
    return line,

ani = animation.FuncAnimation(fig, update, frames = int(ntraces/nframes), interval=30)
plt.show()
```

2 hoc source code

2.1 n01.hoc

```
//  n01.hoc:  a simple hh cell

create soma
soma insert hh

objref cvode
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)

objref vv, tv
vv = new Vector()              // membrane potential vector
tv = new Vector()              // time stamp vector
vv.record(&soma.v(0.5))
tv.record(&t)

tstop = 200.0
v_init = -65.0
finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
vv.plot(g, tv, 2, 1)
g.flush()
```

2.2 n02.hoc

```
//  n02.hoc:  a simple hh cell

create soma
soma {
    insert hh
        el_hh = -30.0  // <== changed
}

objref cvode
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)

objref vv, tv
vv = new Vector()              // membrane potential vector
tv = new Vector()              // time stamp vector
vv.record(&soma.v(0.5))
tv.record(&t)

tstop = 200.0
v_init = -65.0
finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
vv.plot(g, tv, 2, 1)
g.flush()
```

2.3 n03.hoc

```
//  n03.hoc:  a simple hh cell,current injection

create soma
soma insert hh

objref ic
soma ic = new IClamp(0.5)
ic.del    = 20.0    // ms
ic.dur    = 100.0   // ms
ic.amp    = 10      // nA

objref cvode
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)

objref vv, tv, cv
vv = new Vector()             // membrane potential vector
tv = new Vector()             // time stamp vector
cv = new Vector()
vv.record(&soma.v(0.5))
tv.record(&t)
cv.record(&ic.i)

tstop = 200.0
v_init = -65.0
finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
vv.plot(g, tv, 2, 1)
cv.plot(g, tv, 3, 1)
g.flush()
```

2.4 n04.hoc

```
// n04.hoc: soma + dendrite

load_file("nrngui.hoc")
tstop = 200.0
v_init = -65.0

create soma
soma {
    L = 30.0
    diam = 30.0
    nseg = 1
    insert hh
}
create ap_dend
ap_dend {
    L = 500.0
    diam = 2.0
    nseg = 23
    insert hh
    gnabar_hh = 0.012
    gkbar_hh = 0.0036
    gl_hh = 0.00003
}
connect ap_dend(0.0), soma(1.0)

objref ic
soma ic = new IClamp(0.5)
ic.del  = 20.0   // ms
ic.dur  = 100.0  // ms
ic.amp  = 1      // nA

objref cvode
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)

objref tv, vs, va1, va2, cv
tv = new Vector()
vs = new Vector()
va1 = new Vector()
va2 = new Vector()
cv = new Vector()

tv.record(&t)
vs.record(&soma.v(0.5))
va1.record(&ap_dend.v(0.1))
va2.record(&ap_dend.v(0.9))
cv.record(&ic.i)

finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
vs.plot(g, tv, 2, 1)
```

```
va1.plot(g, tv, 3, 1)
va2.plot(g, tv, 4, 1)
cv.mul(10).plot(g,tv,5,1)
g.flush()
```

2.5 n05.hoc

```
// n05.hoc

tstop = 25
v_init = -65.0

create soma
soma {
    L = 30.0
    diam = 30.0
    nseg = 1
    insert hh
}

create ap_dend
ap_dend {
        L = 500.0
    diam = 2.0
    nseg = 23
    insert hh
    gnabar_hh = 0.012
    gkbar_hh = 0.0036
    gl_hh = 0.00003
}

connect ap_dend(0), soma(1.0)

// synaptic input
objref esyn
ap_dend esyn = new AlphaSynapse(0.5)
esyn.onset = 5
esyn.tau = 0.1
esyn.gmax = 0.05
esyn.e = 0.0

objref tv, vs, va1, va2
tv  = new Vector()
vs  = new Vector()
va1 = new Vector()
va2 = new Vector()

tv.record(&t)
vs.record(&soma.v(0.5))
va1.record(&ap_dend.v(0.1))
va2.record(&ap_dend.v(0.9))

objref cvode
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)

finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
```

```
vs.plot(g, tv, 2, 1)
va1.plot(g, tv, 3, 1)
va2.plot(g, tv, 4, 1)
g.flush()
```

2.6 n06.hoc

```
// n06.py excitatory and inhibitory inputs

tstop = 40.0
dt = 0.01
v_init = -65

create soma
soma {
    L = 30
    diam = 30
    nseg = 1
    insert hh
}

create ap_dend
ap_dend {
    L = 500
    diam = 2
    nseg = 23
    insert hh
    gnabar_hh = 0.012
    gkbar_hh = 0.0036
    gl_hh = 0.00003
}

connect ap_dend(0), soma(1)

ap_dend objref esyn
esyn = new AlphaSynapse(0.1)
esyn.onset = 10
esyn.tau   = 0.1
esyn.gmax  = 0.04
esyn.e     = 0.0

objref isyn_p
ap_dend isyn_p = new AlphaSynapse(0.05)
isyn_p.onset = 5
isyn_p.tau = 3.0
isyn_p.e = -70    // v_init

objref isyn_d
isyn_d = new AlphaSynapse(0.95)
isyn_d.onset = 5
isyn_d.tau = 3.0
isyn_d.e = -70    // v_init

objref tv, vs
tv = new Vector()
vs = new Vector()
tv.record(&t)
vs.record(&soma.v(0.5))

objref tr, tt
tr = new List()
tt = new List()

objref cvode
```

```
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)


for i = 0, 2 {
    if(i == 0){
        isyn_p.gmax = 0.0
        isyn_d.gmax = 0.0
    }
    if(i == 1){
        isyn_p.gmax = 0.03
        isyn_d.gmax = 0.0
    }
    if(i == 2){
        isyn_p.gmax = 0.0
        isyn_d.gmax = 0.03
    }
    finitialize(v_init)
    fcurrent()
    while(t < tstop){
        cvode.fixed_step()
    }
    tt.append(tv.c())
    tr.append(vs.c())
}

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
tr.o(0).plot(g, tt.o(0), 2, 1)
tr.o(1).plot(g, tt.o(1), 3, 1)
tr.o(2).plot(g, tt.o(2), 4, 1)
g.flush()
```

2.7 n07.hoc

```
// n07.py

tstop = 40.0
dt = 0.01
v_init = -65

create soma
soma {
    L = 30
    diam = 30
    nseg = 1
    insert hh
}

create ap_dend
ap_dend {
    L = 500
    diam = 2
    nseg = 23
    insert hh
    gnabar_hh = 0.012
    gkbar_hh = 0.0036
    gl_hh = 0.00003
}
connect ap_dend(0), soma(1)

objref esyn
ap_dend esyn = new AlphaSynapse(0.1)
esyn.onset = 10
esyn.tau   = 0.1
esyn.gmax  = 0.04
esyn.e     = 0.0


objref isyn, isyns
isyns = new List()
for i = 0, 9 {
    ap_dend isyn = new AlphaSynapse(0.1*i+0.05)
    isyn.onset = 5
    isyn.tau = 3.0
    isyn.e = -70
    isyns.append(isyn)
}

objref tv, vs
tv = new Vector()
vs = new Vector()
tv.record(&t)
vs.record(&soma.v(0.5))

objref tr, tt
tr = new List()
tt = new List()

cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)
```

```
// synaptic input
for j = -1, 9 {            // -1 for excitatory input only
    for i = 0, 9 {
        if(i == j){
            isyns.o(i).gmax = 0.02
        }else{
            isyns.o(i).gmax = 0
        }
    }

    finitialize(v_init)
    fcurrent()
    while(t < tstop){
        cvode.fixed_step()
    }
    tt.append(tv.c())
    tr.append(vs.c())
}

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
for i = 0, 10 {
    cl = (i % 8) + 1
    tr.o(i).plot(g, tt.o(i), cl, 1)
}
g.flush()
```

2.8 n08.hoc

```
// n08.hoc: connecting neurons

begintemplate HHneuron
    public soma, ap_dend, esyn
    create soma, ap_dend
    objref esyn

    proc init() {
        soma {
            L = 30
            diam = 30
            nseg = 1
            insert hh
        }
        ap_dend {
            L = 500
            diam = 2
            nseg = 23
            insert hh
            gnabar_hh = 0.012
            gkbar_hh  = 0.0036
            gl_hh = 0.00003
        }

        connect ap_dend(0), soma(1)
        ap_dend esyn = new Exp2Syn(0.5)
        esyn.tau1 = 0.5
        esyn.tau2 = 1.0
        esyn.e = 0
    }
endtemplate HHneuron


// 2 cells
objref hh_neuron[2]
hh_neuron[0] = new HHneuron()
hh_neuron[1] = new HHneuron()

// synapse
objref stim
stim = new NetStim(0.5)
stim.interval = 20
stim.number = 3
stim.start = 20
stim.noise = 0

// connections
objref nclist
nclist = new List()
nclist.append(new NetCon(stim, hh_neuron[0].esyn, 0.0, 0, 0.02))
hh_neuron[0].soma nclist.append(new NetCon(&v(0.5), hh_neuron[1].esyn, 10, 1, 0.02))

tstop = 100.0
v_init = -65.0

objref cvode
cvode = new CVode()
```

```
cvode.active(1)
cvode.atol(1.0e-5)

objref tv, vs0, vs1
tv = new Vector()
vs0 = new Vector()
vs1 = new Vector()

tv.record(&t)
vs0.record(&hh_neuron[0].soma.v(0.5))
vs1.record(&hh_neuron[1].soma.v(0.5))

finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
vs0.plot(g, tv, 2, 1)
vs1.plot(g, tv, 3, 1)
g.flush()
```

2.9 n09.hoc

```
//  n09.hoc: voltage clamp


create soma
soma {
    L = 30.0
    diam = 30.0
    nseg = 1
    insert hh
}
objref esyn, vcl
soma esyn = new Exp2Syn(0.5)
soma vcl = new VClamp(0.5)

objref stim
stim = new NetStim(0.5)
stim.interval = 50
stim.number = 2
stim.start = 50
stim.noise = 0

objref nclist
nclist = new List()
nclist.append( new NetCon(stim, esyn, 0.0, 0, 0.005))

tstop = 200.0
dt = 0.01
v_init = -65.0

vcl.dur[0] = 10.0
vcl.dur[1] = 10.0
vcl.dur[2] = 180.0
vcl.amp[0] = v_init
vcl.amp[1] = v_init
vcl.amp[2] = v_init
vcl.gain = 1000.0
vcl.tau1 = 0.01
vcl.tau2 = 0.01

objref tv, vv, cv
tv = new Vector()
vv = new Vector()
cv = new Vector()

tv.record(&t)
vv.record(&soma.v(0.5))
cv.record(&vcl.i)

finitialize(v_init)
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -300, 20)
vv.plot(g, tv, 2, 1)
cv = cv.mul(1000)
cv.plot(g, tv, 3, 1)
```

```
g.flush()
```

2.10 n10.hoc

```
// n10.hoc
// iv in voltage-clamp mode


tstop  = 50.0
dt    = 0.01
vstart = -100.0
vstep  = 10.0
ntrace = 20
npts   = int(tstop/dt)+1

create soma
soma {
    diam = 30.0
    L = 30.0
    nseg = 1
    insert pas
}

objref vc
soma vc = new VClamp(0.5)
vc.dur[0] = 10.0
vc.dur[1] = 10.0
vc.dur[2] = 30.0

objref syn
soma syn = new Exp2Syn(0.5)
syn.e = 0

objref ns
ns = new NetStim()
ns.number = 1
ns.start = 10.0
ns.noise = 0

objref nc
nc = new List()
nc.append(new NetCon(ns, syn, 0, 0, 0.001))   // 1 nS

objref tv, cv
tv = new Vector()
tv.record(&t)
cv = new Vector()
cv.record(&vc.i)

objref traces[ntrace]
for i =0, ntrace-1 {
    v_init = vstart + vstep * i
    vc.amp[0]=v_init
    vc.amp[1]=v_init
    vc.amp[2]=v_init
    soma.e_pas = v_init

    finitialize(v_init)

    fcurrent()
    run(tstop)
```

```
    traces[i] = cv.c()
}

objref g
g = new Graph()
g.size(0, tstop, -0.1, 0.1)
for i = 0, ntrace-1 {
    j = (i % 8) + 1
    traces[i].plot(g, tv, j, 1)
}
g.flush()
```

2.11 n11.hoc

```
//  n11.hoc

create soma
soma {
    L = 30.0
    Ra = 100.0
    nseg = 1
    diam = 30.0
    insert hh
}

create dend[10]
for i = 0, 9 {
    dend[i] {
        L = 300.0
        Ra = 100.0
        nseg = 21
        diam = 2.0
        insert hh
        gl_hh *= 2.3
    }
    connect dend[i](0), soma(1.0)
}

objref vcl
soma vcl = new VClamp(0.5)

objref esyn
dend[0] esyn = new Exp2Syn(0.75)

// stimulation
objref stim
stim = new NetStim(0.5)
stim.interval = 50.0
stim.number = 2
stim.start = 50.0
stim.noise = 0

// synaptic connections
objref nclist
nclist = new List()
nclist.append(new NetCon(stim, esyn, 0.0, 0, 0.001))

tstop = 200.0
dt = 0.01
v_init = -65.0

vcl.dur[0] = 10.0
vcl.dur[1] = 10.0
vcl.dur[2] = 180.0
vcl.amp[0] = v_init
vcl.amp[1] = v_init
vcl.amp[2] = v_init
vcl.gain = 1000.0
vcl.tau1 = 0.1
vcl.tau2 = 0.1

```

```
objref tv, vs, vd, cv
tv = new Vector()
vs = new Vector()
vd = new Vector()
cv = new Vector()

tv.record(&t)
vs.record(&soma.v(0.5))
vd.record(&dend[0].v(0.75))
cv.record(&vcl.i)

finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -70, -35)
vs.plot(g, tv, 2, 1)
vd.plot(g, tv, 3, 1)
cv = cv.mul(100)
cv.plot(g, tv, 4, 1)
g.flush()
```

2.12 n12.hoc

```
//  n12.hoc

load_file("stdlib.hoc")
load_file("../models/tc200.geo")

nSyn  = 30
tstop = 100.0
dt    = 0.01
v_init= -65.0


begintemplate TimeClock
    public getSec
    double jx[3]
    strdef datetime
    func getSec(){  // returns time in sec
        system("date ' +%H %M %' S", datetime)
        sscanf(datetime,"%d%d%d", &jx[0], &jx[1], &jx[2])
        return (jx[0]*60 + jx[1])*60 + jx[2]
    }
endtemplate TimeClock


func measureDist(){local dx localobj sr
    dx = 0
    sr = new SectionRef()
    while(sr.has_parent){
        dx += L
        access sr.parent
        sr = new SectionRef()
    }
    return dx
}

// Properties of dendrites

objref dendritic
dendritic = new SectionList()
forall{
        insert hh
        Ra = 100
        dendritic.append()
}
soma dendritic.remove()

forsec dendritic {
    d = lambda_f(100)
    nseg = int((L/(0.1*d))+0.9)/2*2+1
}

// total dendritic length
totalLength = 0.0
forsec dendritic {
    totalLength += L
}
print "Tolal dendrite length = ", totalLength

```

```
// Shape Plot
objref sh
sh = new Shape(1)
sh.size(-150, 150, -150, 150)

// stimulation
objref stim
stim = new NetStim(0.5)
stim.interval = 50.0
stim.number = 2
stim.start = 20.0
stim.noise = 0

objref nclist
nclist = new List()

// random generator
objref tx, r, locvec
tx = new TimeClock()
r = new Random(tx.getSec())
r.uniform(0, totalLength)
locvec = new Vector(nSyn)
locvec.setrand(r)


objref sref, esyn[nSyn]
for k = 0, nSyn-1 {
    l=0
    l1 = 0
    found = 0
    forsec dendritic {
        if(found==0){
            l1 += L
            if(l1>locvec.x(k)){
                lx = (locvec.x(k)-l)/L
                sref = new SectionRef()
                print secname(), measureDist()
                sref.sec {
                    esyn[k] = new Exp2Syn(lx)
                    sh.color(2)
                }
                sh.point_mark(esyn[k],3)
                nclist.append( new NetCon(stim, esyn[k], 0.0, 0, 0.001))
                found = 1
                break
            }
            l = l1
        }
    }
}
sh.flush()
doEvents()

for k =0, nSyn-1 {
    l = 0
    l1 = 0
    found = 0
    forsec dendritic {
        if(found==0){
            l1 += L
            if(l1>locvec.x(k)){
```

```
                lx = (locvec.x(k)-l)/L
                sref = new SectionRef()
                print secname(), measureDist()
                sref.sec {
                    esyn[k] = new Exp2Syn(lx)
                    sh.color(2)
                }
                sh.point_mark(esyn[k],3)
                nclist.append( new NetCon(stim, esyn[k], 0.0, 0, 0.001))
                found = 1
                break
            }
            l = l1
        }
    }
}
sh.flush()
doEvents()


// main loop
objref cvode
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)

objref tv, vs
tv = new Vector()
vs = new Vector()
tv.record(&t)
vs.record(&soma.v(0.5))

finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
vs.plot(g, tv, 2, 1)
g.flush()
```

2.13 n13.hoc

```
//  n13.hoc


load_file("../models/tc200.geo")

tstop = 200.0
dt = 0.01
v_init = -65.0

nSyn=100       // number of synapses
mdel = 10.0    // mean of synaptic delay
sdel = 1.0     // sd of synaptic delay
w = 0.001      // synaptic weight
br = 5

objref dendritic, dendriticN
objref sh
objref r, vloc, vdel, tx
objref esyn[nSyn]
objref sref
strdef s

begintemplate TimeClock
    public getSec
    double jx[3]
    strdef datetime
    func getSec(){  // returns time in sec
        system("date '+%H %M %'S", datetime)
        sscanf(datetime,"%d%d%d", &jx[0], &jx[1], &jx[2])
        return (jx[0]*60 + jx[1])*60 + jx[2]
    }
endtemplate TimeClock

func measureDist(){local dx localobj sr, srsave
    dx = 0
    srsave = new SectionRef()
    sr = new SectionRef()
    while(sr.has_parent){
        dx += L
        access sr.parent
        sr = new SectionRef()
    }
    access srsave.sec
    return dx
}


func countBranch(){local bx localobj sr, srsave
    bx = 0
    srsave = new SectionRef()
    sr = new SectionRef()
    while(sr.has_parent){
        if(sr.nchild==2){
            bx += 1
        }
        access sr.parent
        sr = new SectionRef()
```

```
    }
    access srsave.sec
    return bx
}



// ------------------------------
//  Properties of dendrites

dendritic = new SectionList()
forall {
    insert hh
    Ra = 100
    dendritic.append()
}
soma dendritic.remove()

forsec dendritic {
    d = lambda_f(100)
    nseg = int((L/(0.1*d))+0.9)/2*2+1
}

dendriticN = new SectionList()
total_length=0
forall {
    if(countBranch()>=br){
        dendriticN.append()
        total_length += L
    }
}
print "total length of \">=", br, "-branch\" dendrites = ", total_length




// Shape Plot

sh = new Shape(1)
//------------------------------

// stimulation
objref stim
stim = new NetStim(0.5)
stim.interval = 50
stim.number = 2
stim.start = 50
stim.noise = 0
//------------------------------
// random generator
tx = new TimeClock()
r = new Random(tx.getSec())
r.uniform(0, total_length)
vloc = new Vector(nSyn)
vloc.setrand(r)
r.normal(mdel,sdel*sdel)
vdel = new Vector(nSyn)
vdel.setrand(r)

objref nc
nc = new List()
```

```
for k = 0, nSyn-1 {
    l=0
    l1 = 0
    found = 0
    forsec dendriticN {
        if(found==0){
            l1 += L
            if(l1>vloc.x(k)){
                lx = (vloc.x(k)-l)/L
                sref = new SectionRef()
                print secname(), ", dist=", measureDist(), \
                    ", br=", countBranch(), ", delay=", vdel.x(k)
                sref.sec {
                    esyn[k] = new Exp2Syn(lx)
                    sh.color(2)
                }
                sh.point_mark(esyn[k],3)
                nc.append(new NetCon(stim, esyn[k], 0.0, vdel.x(k), w))
                found = 1
                break
            }
            l = l1
        }
    }
}
sh.flush()
doEvents()
//------------------------------
// main loop

objref tv, vs
tv = new Vector()
vs = new Vector()
tv.record(&t)
vs.record(&soma.v(0.5))

finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -100, 50)
vs.plot(g, tv, 2, 1)
g.flush()
```

2.14 n14.hoc

```
// n14.py

create soma
soma insert m01

tstop = 40.0
v_init = -65.0

objref cvode
cvode = new CVode()
cvode.active(1)
cvode.atol(1.0e-5)

objref tv, zv
tv = new Vector()
zv = new Vector()
tv.record(&t)
zv.record(&soma.z_m01(0.5))

finitialize(v_init)
//h.fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, -1, 1)
zv.plot(g, tv, 2, 1)
g.flush()
```

2.15 n15.hoc

```
// n15.hoc  Markov model of Na channel

nstates = 9

tstop = 10.0
v_init = -100.0
dt = 0.05

create c
c insert mna

objref vc
c vc = new VClamp(0.5)
vc.dur[0] = 2.0
vc.dur[1] = 5.0
vc.dur[2] = 3.0
vc.amp[0] = v_init
vc.amp[1] = 0.0
vc.amp[2] = v_init

objref tv, zv
tv = new Vector()
zv = new List()
for i = 0, nstates-1 {
    zv.append(new Vector())
}

tv.record(&t)
strdef s
for i = 0, nstates-1{
    sprint(s, "zv.object(%1d).record(&c.z%1d_mna(0.5))", i, i)
    execute(s)
}

finitialize(v_init)
fcurrent()
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, 0, 1)
for i = 0, nstates-1 {
    zv.object(i).plot(g, tv, i+2, 1)
}
g.flush()
```

2.16 n16.hoc

```
// n16.hoc   Markov model of Na channel
//           Voltage-dependent activation

nstates = 9

tstop   = 20.0
dt    = 0.01

v_init = -100.0
vstart = -100.0
vstep   = 10.0
ntraces = 20
npts    = int(tstop/dt)+1
eqna = 50.0

create c
c insert mna

objref vc
c vc = new VClamp(0.5)
vc.dur[0] = 5.0
vc.dur[1] = 10.0
vc.dur[2] = 5.0
vc.amp[0] = v_init
vc.amp[1] = v_init
vc.amp[2] = v_init

objref tv, gv, vv
tv = new Vector()
tv.record(&t)
gv = new Vector()
gv.record(&c.z3_mna(0.5))
vv = new Vector()
vv.record(&c.v(0.5))

objref traces, vtra
traces = new List()
vtra = new List()
for i = 0, ntraces-1 {
    vc.amp[1] = vstart + vstep * i
    finitialize(v_init)
    fcurrent()
    run(tstop)

    traces.append(gv.c())
    vtra.append(vv.c())
}

// open probability
objref g0, g1
g0 = new Graph()
g0.size(0, tstop, 0, 0.3)
for i = 0, ntraces-1 {
    traces.o(i).plot(g0, tv, 2, 1)
}
g0.flush()
```

```
// current-voltage
objref cv
cv = new List()
g1 = new Graph()
g1.size(0, tstop, -15, 2)
for i = 0, ntraces-1 {
    cv.append(vtra.o(i).sub(eqna).mul(traces.o(i)))
    cv.object(i).plot(g1, tv, 2, 1)
}
g1.flush()
```

2.17 n17.hoc

```
// n15.hoc


load_file ("../models/cellModel.hoc")

ncells = 20
objref cell[ncells]

for i = 0, ncells-1 {
    cell[i] = new BallStickCell( )
}

objref nclist, target, nc
nclist = new List()
for i = 0, ncells-1 {
    target = cell[(i+1) % ncells]   // next cell
    nc = cell[i].connect2target(target.synlist.o(0))
    nclist.append(nc)
    nc.delay = 1
    nc.weight[0] = 0.01

}

//external stimulus to syn of cell[0]
objref stim, ncstim, nil
stim = new NetStim()
stim.number = 1
stim.start = 0
ncstim = new NetCon(stim, cell[0].synlist.o(0))
ncstim.delay=0
ncstim.weight[0] = 0.01

objref tvec, idvec
tvec  = new Vector( )
idvec = new Vector( )
for i = 0, ncells-1 {
    nc = cell[i].connect2target(nil)
    nc.record (tvec, idvec, i)
}

// Simulation control
tstop = 100
v_init = -65
finitialize(v_init)
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, 0, ncells)
idvec.mark(g, tvec, "|", 4, 2, 2)
g.flush()
```

2.18 n17.hoc

```
// n15.hoc


load_file ("../models/cellModel.hoc")

ncells = 20
objref cell[ncells]

for i = 0, ncells-1 {
    cell[i] = new BallStickCell( )
}

objref nclist, target, nc
nclist = new List()
for i = 0, ncells-1 {
    target = cell[(i+1) % ncells]   // next cell
    nc = cell[i].connect2target(target.synlist.o(0))
    nclist.append(nc)
    nc.delay = 1
    nc.weight[0] = 0.01

}

//external stimulus to syn of cell[0]
objref stim, ncstim, nil
stim = new NetStim()
stim.number = 1
stim.start = 0
ncstim = new NetCon(stim, cell[0].synlist.o(0))
ncstim.delay=0
ncstim.weight[0] = 0.01

objref tvec, idvec
tvec  = new Vector( )
idvec = new Vector( )
for i = 0, ncells-1 {
    nc = cell[i].connect2target(nil)
    nc.record (tvec, idvec, i)
}

// Simulation control
tstop = 100
v_init = -65
finitialize(v_init)
run(tstop)

objref g
g = new Graph()
g.size(0, tstop, 0, ncells)
idvec.mark(g, tvec, "|", 4, 2, 2)
g.flush()
```

3 models, mod files

3.1 cellModel.hoc

```

// M.L. Hines1 and N.T. Carneva
// Translating network models to parallel hardware in NEURON
// J Neurosci Methods. 2008 Apr 30; 169(2): - 425455.

begintemplate BallStickCell
        public is_art
        public init, topol, basic_shape, subsets, geom, biophys, geom_nseg, \
        biophys_inhomo
        public synlist, x, y, z, position, connect2target
        public soma, dend
        public all

        objref synlist
        proc init ( ) {
                topol ( )
                subsets ( )
                geom ( )
                biophys ( )
                geom_nseg ( )
                synlist = new List ( )
                synapses ( )
                x = y = z = 0 // only change via position
        }
        create soma, dend
        proc topol ( ) { local i
                connect dend (0), soma (1)
                basic_shape ( )
        }
        proc basic_shape ( ) {
                soma {pt3dclear ( ) pt3dadd (0, 0, 0, 1) pt3dadd (15, 0, 0, 1)}
                dend {pt3dclear ( ) pt3dadd (15, 0, 0, 1) pt3dadd (105, 0, 0, 1)}
        }
        objref all
        proc subsets ( ) { local i
                objref all
                all = new SectionList ( )
                soma all.append ( )
                dend all.append ( )
        }
        proc geom ( ) {
                forsec all { }
                soma { L = diam = 12.6157 } // *area = 500
                dend { L = 200 diam = 1 }
        }

        //external lambda_f
        proc geom_nseg ( ) {
                forsec all {
                        lf = 1e5*sqrt(diam/(4*PI*100*Ra*cm))
                        nseg = int ((L/(0.1*lf)+.9)/2)*2 + 1
                }
        }

```

```
        proc biophys ( ) {
                forsec all {
                        Ra = 100
                        cm = 1
                }
                soma {
                        insert hh
                        gnabar_hh = 0.12
                        gkbar_hh = 0.036
                        gl_hh = 0.0003
                        el_hh = -54.3
                }
                dend {
                        insert pas
                        g_pas = 0.001
                        e_pas = -65
                }
        }

        proc biophys_inhomo ( ){}

        proc position ( ) { local i
                soma for i = 0, n3d( )-1 {
                        pt3dchange (i, $1-x+x3d (i), $2-y+y3d (i), $3-z+z3d (i), diam3d (i))
                }
                x = $1 y = $2 z = $3
        }

        obfunc connect2target ( ) { localobj nc // $o1 target point process,
                                            // optional $o2 returned NetCon
                soma nc = new NetCon (&v(0.5), $o1)
                nc.threshold = 10
                return nc
        }

        objref syn_
        proc synapses ( ) {
                /* E0 */
                dend syn_ = new ExpSyn (0.8)
                synlist.append (syn_)
                syn_.tau = 2
                /* I1 */
                dend syn_ = new ExpSyn (0.1)
                synlist.append (syn_)
                syn_.tau = 5
                syn_.e = -80
        }

        func is_art ( ) {
                return 0
        }
endtemplate BallStickCell
```

3.2 cellModel.py

```python
# cellModel.py : Bann-and-Stick model

from neuron import h
import math

class BallStickCell(object):

    def __init__(self):
        self.x = self.y = self.z = 0
        self.soma = h.Section(name='soma', cell=self)
        self.dend = h.Section(name='dend', cell=self)
        self.synlist = h.List()
        self.topol()
        self.subsets()
        self.geom()
        self.biophys()
        self.geom_nseg()
        self.synapses()

    def topol(self):
        self.dend.connect(self.soma(1))

    def subsets(self):
        self.all = h.SectionList()
        self.all.append(sec=self.soma)
        self.all.append(sec=self.dend)

    def geom(self):
        self.soma.L = self.soma.diam = 12.6157 # microns
        self.dend.L = 200                      # microns
        self.dend.diam = 1                     # microns

    def geom_nseg(self):
        for sec in self.all:
            sec.nseg = int((sec.L/(0.1*h.lambda_f(100))+.9)/2)*2 + 1

    def biophys(self):
        for sec in self.all: # 'all' defined in build_subsets
            sec.Ra = 100    # Axial resistance in Ohm * cm
            sec.cm = 1      # Membrane capacitance in micro Farads / cm^2

        # Insert active Hodgkin-Huxley current in the soma
        self.soma.insert('hh')
        self.soma.gnabar_hh = 0.12  # Sodium conductance in S/cm2
        self.soma.gkbar_hh = 0.036  # Potassium conductance in S/cm2
        self.soma.gl_hh = 0.0003    # Leak conductance in S/cm2
        self.soma.el_hh = -54.3     # Reversal potential in mV
        # Insert passive current in the dendrite
        self.dend.insert('pas')
        self.dend.g_pas = 0.001  # Passive conductance in S/cm2
        self.dend.e_pas = -65    # Leak reversal potential mV

    def connect2target(self, pp):
        nc = h.NetCon(self.soma(0.5)._ref_v, pp, sec=self.soma)
        nc.threshold = 10
        return nc
```

```
    def synapses (self):
        self.synlist.append(h.ExpSyn(0.8,sec=self.dend))
        self.synlist.object(0).tau = 2

        self.synlist.append(h.ExpSyn(0.1,sec=self.dend))
        self.synlist.object(1).tau = 5
        self.synlist.object(1).e = -80.0
```

3.3 m01.mod

```
: m01.mod: a simple ODE y'' = -y

NEURON {
    SUFFIX m01
    RANGE z
}

STATE {z z1}

INITIAL {
    z  = 0.0
    z1 = 1.0
}

BREAKPOINT{
    SOLVE zstates METHOD cnexp
}

DERIVATIVE zstates {
    z'  =  z1
    z1' = -z
}
```

3.4 exp2syn.mod

```
:   nrn-6.1src/nrnoc/exp2syn.mod

NEURON {
    POINT_PROCESS Exp2Syn
    RANGE tau1, tau2, e, i
    NONSPECIFIC_CURRENT i
    RANGE g
    GLOBAL total
}
UNITS {
    (nA) = (nanoamp)
    (mV) = (millivolt)
    (uS) = (microsiemens)
}
PARAMETER {
    tau1= 0.1 (ms) <1e-9,1e9>
    tau2 = 10 (ms) <1e-9,1e9>
    e=0 (mV)
}
ASSIGNED {
    v (mV)
    i (nA)
    g (uS)
    factor
    total (uS)
}
STATE {
    A (uS)
    B (uS)
}
INITIAL {
    LOCAL tp
    total = 0
    if (tau1/tau2 > .9999) {  : avoid tau1==tau2
        tau1 = .9999*tau2
    }
    A=0
    B=0
    tp = (tau1*tau2)/(tau2 - tau1) * log(tau2/tau1)
    factor = -exp(-tp/tau1) + exp(-tp/tau2)
    factor = 1/factor
}

BREAKPOINT {
    SOLVE state METHOD cnexp
    g = B-A
    i = g * (v - e)
}

DERIVATIVE state {
    A' = -A/tau1
    B' = -B/tau2
}
NET_RECEIVE(weight (uS)) {
    A = A + weight*factor
    B = B + weight*factor
    total = total+weight
```

```
}
```

3.5 gsyn.mod

```
: gsyn.mod    The NEURON Book pp. 281-282

NEURON {
    POINT_PROCESS GSyn
    RANGE tau1, tau2, e, i
    RANGE Gtau1, Gtau2, Ginc
    NONSPECIFIC_CURRENT i
    RANGE g
}
UNITS {
    (nA)    =  (nanoamp)
    (mV)    =  (millivolt)
    (umho)  =  (micromho)
}
PARAMETER {
    tau1    = 1     (ms)
    tau2    = 1.05  (ms)
    Gtau1   = 20    (ms)
    Gtau2   = 21    (ms)
    Ginc    = 1
    e       = 0     (mV)
}
ASSIGNED {
    v   (mv)
    i   (nA)
    g   (umho)
    factor
    Gfactor
}
STATE {
    A (umho)
    B (umho)
}
INITIAL {
    LOCAL tp
    A = 0
    B = 0
    tp = (tau1*tau2)/(tau2-tau1) * log(tau2/tau1)
    factor = -exp(-tp/tau1) + exp(-tp/tau2)
    factor = 1/factor
    tp = (Gtau1*Gtau2)/(Gtau2-Gtau1) * log(Gtau2/Gtau1)
    Gfactor = -exp(-tp/Gtau1) + exp(-tp/Gtau2)
    Gfactor = 1/Gfactor
}
BREAKPOINT {
    SOLVE state METHOD cnexp
    g = B - A
    i = g * (v -e)
:    i = g * (v - e)
}
DERIVATIVE state {
    A' = -A/tau1
    B' = -B/tau2
}
NET_RECEIVE (weight (umho), w, G1, G2, t0 (ms)){
    G1 = G1*exp(-(t-t0)/Gtau1)
    G2 = G2*exp(-(t-t0)/Gtau2)
```

```
    G1 = G1 + Ginc * Gfactor
    G2 = G2 + Ginc * Gfactor
    t0 = t
    w = weight * (1 + G2 - G1)
    A = A + w*factor
    B = B + w*factor
}
```

3.6 NMDAR.mod

```
: NMDAR.mod   NMDA receptor channel
NEURON {
    POINT_PROCESS nmdanet
    RANGE R, g, mg
    NONSPECIFIC_CURRENT i
    GLOBAL Cdur, Alpha, Beta, Erev, Rinf, Rtau
}
UNITS {
    (nA) = (nanoamp)
    (mV) = (millivolt)
    (umho) = (micromho)
    (mM) = (milli/liter)
}
PARAMETER {
    Cdur    = 1   (ms)      : transmitter duration (rising phase)
    Alpha   = 0.35 (/ms)    : forward (binding) rate
    Beta    = 0.035 (/ms)   : backward (unbinding) rate
    Erev    = 0 (mV)        : reversal potential
    mg      = 1 (mM)        : external magnesium concentration
}
ASSIGNED {
    v       (mV)            : postsynaptic voltage
    i       (nA)            : current = g*(v - Erev)
    g       (umho)          : conductance
    Rinf                    : steady state channels open
    Rtau    (ms)            : time constant of channel binding
    synon
}
STATE {Ron Roff}
INITIAL {
    Rinf = Alpha / (Alpha + Beta)
    Rtau = 1 / (Alpha + Beta)
    synon = 0
}

BREAKPOINT {
    SOLVE release METHOD cnexp
    g = mgblock(v)*(Ron + Roff)*1(umho)
    i = g*(v - Erev)
}

DERIVATIVE release {
    Ron' = (synon*Rinf - Ron)/Rtau
    Roff' = -Beta*Roff
}
FUNCTION mgblock(v(mV)) {
    TABLE
    DEPEND mg
    FROM -140 TO 80 WITH 1000
    mgblock = 1 / (1 + exp(0.062 (/mV) * -v) * (mg / 3.57 (mM)))
}
NET_RECEIVE(weight, on, nspike, r0, t0 (ms)) {
    if (flag == 0) { : a spike, so turn on if not already in a Cdur pulse
        nspike = nspike + 1
        if (!on) {
            r0 = r0*exp(-Beta*(t - t0))
            t0 = t
```

```
            on = 1
            synon = synon + weight
            Ron =  Ron + r0
            Roff = Roff - r0
        }
        net_send(Cdur, nspike)
    }
    if (flag == nspike) { : if this associated with last spike then turn off
        r0 = weight*Rinf + (r0 - weight*Rinf)*exp(-(t - t0)/Rtau)
        t0 = t
        synon = synon - weight
        Ron = Ron - r0
        Roff = Roff + r0
        on = 0
    }
}
```

3.7 itvlfire.mod

```
: InvlFire.mod  The NEURON Book pp. 309- 310

NEURON {
    ARTIFICIAL_CELL IntervalFire  : name of the module
    RANGE tau, m, invl            : accessible variables
}
PARAMETER {
    tau  = 5 (ms)   <1e-9,1e9>
    invl = 10 (ms) <1e-9,1e9>
}
ASSIGNED {                        : non-accessible variables
    m
    minf
    t0 (ms)
}
INITIAL {
    minf = 1/(1 - exp(-invl/tau)) : so natural spike interval is invl
    m = 0
    t0 = 0
    net_send(firetime(), 1)       : set a self-event
                                  : after time period of firetime()
}
NET_RECEIVE (w) {
    m = M()
    t0 = t
    if (flag == 0) {              : *** event triggered by others ***
        m = m + w
        if (m > 1) {
            m = 0
            net_event(t)          : issue event
        }
        net_move(t+firetime())    : move the next event to t + firetime()
    }else{                        : *** self-triggered event ***
        net_event(t)              : issue event
        m = 0
        net_send(firetime(), 1)   : next self-event
    }
}
FUNCTION firetime()(ms) {         : m < 1 and minf > 1
    firetime = tau*log((minf-m)/(minf - 1))
}
FUNCTION M() {                    : to monitor m-value
    M = minf + (m - minf)*exp(-(t - t0)/tau)
}
```