

Supplement to  
“A Simplified 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

1	Python source code	1
1.1	n01.py . . . . .	1
1.2	n02.py . . . . .	2
1.3	n03.py . . . . .	3
1.4	n04.py . . . . .	4
1.5	n05.py . . . . .	6
1.6	n06.py . . . . .	8
1.7	n07.py . . . . .	10
1.8	n08.py . . . . .	12
1.9	n09.py . . . . .	14
1.10	n10.py . . . . .	16
1.11	n11.py . . . . .	18
1.12	n12.py . . . . .	20
1.13	n13.py . . . . .	22
1.14	n14.py . . . . .	25
1.15	n15.py . . . . .	26
1.16	n16.py . . . . .	27
1.17	n17.py . . . . .	29
1.18	n18.py . . . . .	31
1.19	n19.py . . . . .	33
1.20	n20.py . . . . .	35
1.21	n21.py . . . . .	38
2	hoc source code	40
2.1	n01.hoc . . . . .	40
2.2	n02.hoc . . . . .	41
2.3	n03.hoc . . . . .	42
2.4	n04.hoc . . . . .	43
2.5	n05.hoc . . . . .	45
2.6	n06.hoc . . . . .	47
2.7	n07.hoc . . . . .	49
2.8	n08.hoc . . . . .	51
2.9	n09.hoc . . . . .	53
2.10	n10.hoc . . . . .	55
2.11	n11.hoc . . . . .	57
2.12	n12.hoc . . . . .	59
2.13	n13.hoc . . . . .	62

2.14	n14.hoc . . . . .	65
2.15	n15.hoc . . . . .	66
2.16	n16.hoc . . . . .	67
2.17	n17.hoc . . . . .	69
2.18	n17.hoc . . . . .	70
3	models, mod files	71
3.1	cellModel.hoc . . . . .	71
3.2	cellModel.py . . . . .	73
3.3	m01.mod . . . . .	75
3.4	exp2syn.mod . . . . .	76
3.5	gsyn.mod . . . . .	78
3.6	NMDAR.mod . . . . .	80
3.7	itvlfire.mod . . . . .	82

# Supplementary materials

## 1 Python source code

### 1.1 n01.py

```
1 # n01.py: a simple hh cell
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 200.0
9 v_init = -65.0
10
11 soma = h.Section()
12 soma.diam = 30.0
13 soma.L = 30.0
14 soma.insert('hh')
15
16 cnode = h.CVode()
17 cnode.active(1)
18 cnode.atol(1.0e-5)
19
20 vv = h.Vector()           # membrane potential vector
21 tv = h.Vector()           # time stamp vector
22 vv.record(soma(0.5)._ref_v)
23 tv.record(h._ref_t)
24
25 h.finitialize(v_init)
26 h.fcurrent()
27 neuron.run(tstop)
28
29 ax = plt.subplot()
30 ax.set_ylim([-80,40])
31 ax.plot(tv.as_numpy(), vv.as_numpy())
32 plt.xlabel('Time (ms)')
33 plt.ylabel('Potential (mV)')
34 plt.savefig('../figs/n01.pdf')
35 plt.show()
```

## 1.2 n02.py

```
1 # n02.py: a simple hh cell, with depolarized equilibrium potential
2 #         for leak current
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 200.0
9 v_init = -65.0
10
11 soma = h.Section()
12 soma.diam = 30.0
13 soma.L = 30.0
14 soma.insert('hh')
15 soma.el_hh = -30.0 # <== changed
16
17 cvode = h.CVode()
18 cvode.active(1)
19 cvode.atol(1.0e-5)
20
21 vv = h.Vector() # membrane potential vector
22 tv = h.Vector() # time stamp vector
23 vv.record(soma(0.5)._ref_v)
24 tv.record(h._ref_t)
25
26 h.finitialize(v_init)
27 h.fcurrent()
28 neuron.run(tstop)
29
30 ax = plt.subplot()
31 ax.set_ylim([-80,40])
32 ax.plot(tv.as_numpy(), vv.as_numpy())
33 plt.xlabel('Time (ms)')
34 plt.ylabel('Potential (mV)')
35 plt.savefig('../figs/n02.pdf')
36 plt.show()
```

### 1.3 n03.py

```
1 # n03.py: a simple hh cell,current injection
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 200.0
9 v_init = -65.0
10
11 soma = h.Section()
12 soma.diam = 30.0
13 soma.L = 30.0
14 soma.insert('hh')
15
16 ic = h.IClamp(0.5, sec=soma)
17 ic.delay = 20.0 # ms
18 ic.dur = 100.0 # ms
19 ic.amp = 0.2 # nA
20
21 cvode = h.CVode()
22 cvode.active(1)
23 cvode.atol(1.0e-5)
24
25 vv = h.Vector() # membrane potential vector
26 tv = h.Vector() # time stamp vector
27 cv = h.Vector() # current size
28 vv.record(soma(0.5)._ref_v)
29 tv.record(h._ref_t)
30 cv.record(ic._ref_i)
31
32 h.finitialize(v_init)
33 h.fcurrent()
34 neuron.run(tstop)
35
36 ax = plt.subplot()
37 ax.set_ylim([-80,40])
38 ax.plot(tv.as_numpy(), vv.as_numpy())
39 ax.plot(tv.as_numpy(), 100.0*cv.as_numpy())
40 plt.xlabel('Time (ms)')
41 plt.ylabel('Potential (mV)')
42 plt.savefig('../figs/n03.pdf')
43 plt.show()
```

## 1.4 n04.py

```
1 # n04.py: soma + dendrite
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 200.0
9 v_init = -65.0
10
11 soma = h.Section()
12 soma.diam = 30.0
13 soma.L = 30.0
14 soma.L = 30.0
15 soma.diam = 30.0
16 soma.nseg = 1
17 soma.insert('hh')
18
19 ap_dend = h.Section()
20 ap_dend.L = 500.0
21 ap_dend.diam = 2.0
22 ap_dend.nseg = 23
23 ap_dend.insert('hh')
24 ap_dend.gnabar_hh = 0.012
25 ap_dend.gkbar_hh = 0.0036
26 ap_dend.gl_hh = 0.00003
27
28 ap_dend.connect(soma, 1.0, 0.0)
29
30 ic = h.IClamp(0.5, sec=soma)
31 ic.delay = 20.0 # ms
32 ic.dur = 100.0 # ms
33 ic.amp = 0.5 # nA
34
35 ccode = h.CCode()
36 ccode.active(1)
37 ccode.atol(1.0e-5)
38
39 tv = h.Vector()
40 vs = h.Vector()
41 va1 = h.Vector()
42 va2 = h.Vector()
43 cv = h.Vector()
44
45 tv.record(h._ref_t)
46 vs.record(soma(0.5)._ref_v)
47 va1.record(ap_dend(0.1)._ref_v)
48 va2.record(ap_dend(0.9)._ref_v)
49 cv.record(ic._ref_i)
50
51 h.finitialize(v_init)
52 h.fcurrent()
53 neuron.run(tstop)
54
55 ax = plt.subplot()
56 #ax.set_xlim([18.0, 30.0])
57 ax.plot(tv.as_numpy(), vs.as_numpy())
```

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



## 1.5 n05.py

```
1 # n05.py
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 25
9 v_init = -65.0
10
11 soma = h.Section()
12 soma.L = 30.0
13 soma.diam = 30.0
14 soma.nseg = 1
15 soma.insert('hh')
16
17 ap_dend = h.Section()
18 ap_dend.L = 500.0
19 ap_dend.diam = 2.0
20 ap_dend.nseg = 23
21
22 ap_dend.insert('hh')
23 ap_dend.gnabar_hh = 0.012
24 ap_dend.gkbar_hh = 0.0036
25 ap_dend.gl_hh = 0.00003
26
27 ap_dend.connect(soma, 1.0, 0)
28
29 # synaptic input
30 esyn = h.AlphaSynapse(0.5, sec = ap_dend)
31 esyn.onset = 5
32 esyn.tau = 0.1
33 esyn.gmax = 0.05
34 esyn.e = 0.0
35
36 cvode = h.CVode()
37 cvode.active(1)
38 cvode.atol(1.0e-5)
39
40 tv = h.Vector()
41 vs = h.Vector()
42 va1 = h.Vector()
43 va2 = h.Vector()
44
45 tv.record(h._ref_t)
46 vs.record(soma(0.5)._ref_v)
47 va1.record(ap_dend(0.1)._ref_v)
48 va2.record(ap_dend(0.9)._ref_v)
49
50 h.finitialize(v_init)
51 h.fcurrent()
52 neuron.run(tstop)
53
54 ax = plt.subplot()
55 ax.plot(tv.as_numpy(), vs.as_numpy())
56 ax.plot(tv.as_numpy(), va1.as_numpy())
57 ax.plot(tv.as_numpy(), va2.as_numpy())
```

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

## 1.6 n06.py

```
1 # n06.py excitatory and inhibitory inputs
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 40.0
9 h.dt = 0.01
10 v_init = -65
11
12 soma = h.Section()
13 soma.L = 30.0
14 soma.diam = 30.0
15 soma.nseg = 1
16 soma.insert('hh')
17
18 ap_dend = h.Section()
19 ap_dend.L = 500.0
20 ap_dend.diam = 2.0
21 ap_dend.nseg = 23
22 ap_dend.insert('hh')
23 ap_dend.gnabar_hh = 0.012
24 ap_dend.gkbar_hh = 0.0036
25 ap_dend.gl_hh = 0.00003
26
27 ap_dend.connect(soma, 1.0, 0)
28
29 esyn = h.AlphaSynapse(0.1, sec = ap_dend)
30 esyn.onset = 10
31 esyn.tau = 0.1
32 esyn.gmax = 0.04
33 esyn.e = 0.0
34
35 isyn_p = h.AlphaSynapse(0.05, sec = ap_dend)
36 isyn_p.onset = 5
37 isyn_p.tau = 3.0
38 isyn_p.e = -70 #v_init
39
40 isyn_d = h.AlphaSynapse(0.95, sec = ap_dend)
41 isyn_d.onset = 5
42 isyn_d.tau = 3.0
43 isyn_d.e = -70 #v_init
44
45 cvode = h.CVode()
46 cvode.active(1)
47 cvode.atol(1.0e-5)
48
49 tv = h.Vector()
50 vs = h.Vector()
51 tv.record(h._ref_t)
52 vs.record(soma(0.5)._ref_v)
53
54 tr = []
55 tt = []
56
57 for i in range(3):
```

```

58     if i == 0:
59         isyn_p.gmax = 0.0
60         isyn_d.gmax = 0.0
61     if i == 1:
62         isyn_p.gmax = 0.03
63         isyn_d.gmax = 0.0
64     if i == 2:
65         isyn_p.gmax = 0.0
66         isyn_d.gmax = 0.03
67
68     h.finitialize(v_init)
69     h.fcurrent()
70     while h.t < tstop:
71         ccode.fixed_step()
72
73     tt.append(np.copy(tv.as_numpy()))
74     tr.append(np.copy(vs.as_numpy()))
75
76 ax = plt.subplot()
77 ax.set_ylim([-80,40])
78 ax.plot(tt[0], tr[0])
79 ax.plot(tt[1], tr[1])
80 ax.plot(tt[2], tr[2])
81 plt.xlabel('Time (ms)')
82 plt.ylabel('Potential (mV)')
83 plt.savefig('../figs/n06.pdf')
84 plt.show()

```

## 1.7 n07.py

```
1 # n07.py
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 40.0
9 h.dt = 0.01
10 v_init = -65
11
12 soma = h.Section()
13 soma.L = 30.0
14 soma.diam = 30.0
15 soma.nseg = 1
16 soma.insert('hh')
17
18 ap_dend = h.Section()
19 ap_dend.L = 500.0
20 ap_dend.diam = 2.0
21 ap_dend.nseg = 23
22 ap_dend.insert('hh')
23 ap_dend.gnabar_hh = 0.012
24 ap_dend.gkbar_hh = 0.0036
25 ap_dend.gl_hh = 0.00003
26
27 ap_dend.connect(soma, 1.0, 0)
28
29 esyn = h.AlphaSynapse(0.1, sec = ap_dend)
30 esyn.onset = 10
31 esyn.tau = 0.1
32 esyn.gmax = 0.04
33 esyn.e = 0.0
34
35 isyn = [h.AlphaSynapse(0.1*i+0.05, sec = ap_dend) for i in range(10)]
36 for i in range(10):
37     isyn[i].onset = 5
38     isyn[i].tau = 3.0
39     isyn[i].e = -70 #v_init
40
41 cvode = h.CVode()
42 cvode.active(1)
43 cvode.atol(1.0e-5)
44
45 tv = h.Vector()
46 vs = h.Vector()
47 tv.record(h._ref_t)
48 vs.record(soma(0.5)._ref_v)
49
50 tr = []
51 tt = []
52
53 # synaptic input
54 for j in range(-1,10): # -1 for excitatory input only
55     for i in range(10):
56         if i == j:
57             isyn[i].gmax = 0.02
```

```

58         else:
59             isyn[i].gmax = 0
60
61         h.finitialize(v_init)
62         h.fcurrent()
63         while h.t < tstop:
64             ccode.fixed_step()
65
66         tt.append(np.copy(tv.as_numpy()))
67         tr.append(np.copy(vs.as_numpy()))
68
69 ax = plt.subplot()
70 ax.set_xlim([5.0, 20])
71 ax.set_ylim([-80,40])
72 #ax.set_xlim([8,14])
73 #ax.set_ylim([-70,-50])
74 for j in range(11):
75     ax.plot(tt[j], tr[j])
76 plt.xlabel('Time (ms)')
77 plt.ylabel('Potential (mV)')
78 plt.savefig('../figs/n07.pdf')
79 #plt.savefig('../figs/n07b.pdf')
80 plt.show()

```

## 1.8 n08.py

```
1 # n08.py: connecting neurons
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 class HHneuron(): # class definition
9     def __init__(self):
10         self.soma = h.Section()
11         self.ap_dend = h.Section()
12
13         self.soma.L = 30
14         self.soma.diam = 30
15         self.soma.nseg = 1
16         self.soma.insert('hh')
17
18         self.ap_dend.L = 500
19         self.ap_dend.diam = 2
20         self.ap_dend.nseg = 23
21         self.ap_dend.insert('hh')
22         self.ap_dend.gnabar_hh = 0.012
23         self.ap_dend.gkbar_hh = 0.0036
24         self.ap_dend.gl_hh = 0.00003
25
26         self.ap_dend.connect(self.soma, 1, 0)
27         self.esyn = h.Exp2Syn(0.5, sec=self.ap_dend)
28         self.esyn.tau1 = 0.5
29         self.esyn.tau2 = 1.0
30         self.esyn.e = 0
31
32 tstop = 100.0
33 v_init = -65.0
34
35 # 2 cells
36 hh_neuron = [HHneuron() for i in range(2)]
37
38 # synapse
39 stim = h.NetStim(0.5)
40 stim.interval = 20
41 stim.number = 3
42 stim.start = 20
43 stim.noise = 0
44
45 # connections
46 nclist = []
47 nclist.append(h.NetCon(stim, hh_neuron[0].esyn, 0.0, 0, 0.02))
48 nclist.append(h.NetCon(hh_neuron[0].soma(0.5)._ref_v, hh_neuron[1].esyn, 10, 1, 0.02))
49
50 ccode = h.CVode()
51 ccode.active(1)
52 ccode.atol(1.0e-5)
53
54 tv = h.Vector()
55 vs0 = h.Vector()
56 vs1 = h.Vector()
57
```

```

58 tv.record(h._ref_t)
59 vs0.record(hh_neuron[0].soma(0.5)._ref_v)
60 vs1.record(hh_neuron[1].soma(0.5)._ref_v)
61
62 h.finitialize(v_init)
63 h.fcurrent()
64 neuron.run(tstop)
65
66 ax = plt.subplot()
67 ax.plot(tv.as_numpy(), vs0.as_numpy())
68 ax.plot(tv.as_numpy(), vs1.as_numpy())
69 plt.xlabel('Time (ms)')
70 plt.ylabel('Potential (mV)')
71 plt.savefig('../figs/n08.pdf')
72 plt.show()

```



## 1.9 n09.py

```
1 # n09.py: voltage clamp
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 tstop = 200.0
9 h.dt = 0.01
10 v_init = -65.0
11
12 soma = h.Section()
13 soma.L = 30.0
14 soma.diam = 30.0
15 soma.nseg = 1
16 soma.insert('hh')
17
18 esyn = h.Exp2Syn(0.5, sec=soma)
19 vcl = h.VClamp(0.5, sec=soma)
20
21 stim = h.NetStim(0.5)
22 stim.interval = 50
23 stim.number = 2
24 stim.start = 50
25 stim.noise = 0
26
27 vcl.dur[0] = 10.0
28 vcl.dur[1] = 10.0
29 vcl.dur[2] = 180.0
30 vcl.amp[0] = v_init
31 vcl.amp[1] = v_init
32 vcl.amp[2] = v_init
33 vcl.gain = 1000.0
34 vcl.tau1 = 0.01
35 vcl.tau2 = 0.01
36
37 nclist = []
38 nclist.append(h.NetCon(stim, esyn, 0.0, 0, 0.005))
39
40 tv = h.Vector()
41 vv = h.Vector()
42 cv = h.Vector()
43
44 tv.record(h._ref_t)
45 vv.record(soma(0.5)._ref_v)
46 cv.record(vcl._ref_i)
47
48 h.finitialize(v_init)
49 neuron.run(tstop)
50
51 ax = plt.subplot()
52 ax.set_ylim([-400,100])
53 ax.plot(tv.as_numpy(), 1000.0*cv.as_numpy())
54 ax.plot(tv.as_numpy(), vv.as_numpy())
55 plt.xlabel('Time (ms)')
56 plt.ylabel('Current')
57 plt.savefig('../figs/n09.pdf')
```

```
58 plt.show()
```

## 1.10 n10.py

```
1 # n10.py
2 # iv in voltage-clamp mode
3
4 from neuron import h
5 import neuron
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 tstop = 50.0
10 h.dt = 0.01
11 vstart = -100.0
12 vstep = 10.0
13 ntrace = 20
14 npts = int(tstop/h.dt)+1
15
16 soma = h.Section()
17 soma.diam = 30.0
18 soma.L = 30.0
19 soma.nseg = 1
20 soma.insert('pas')
21
22 vc = h.VClamp(0.5, sec=soma)
23 vc.dur[0] = 10.0
24 vc.dur[1] = 10.0
25 vc.dur[2] = 30.0
26
27 syn = h.Exp2Syn(0.5, sec=soma)
28 syn.e = 0
29
30 ns = h.NetStim()
31 ns.number = 1
32 ns.start = 10.0
33 ns.noise = 0
34
35 nc = []
36 nc.append(h.NetCon(ns, syn, 0, 0, 0.001)) # 1 nS
37
38 tv = h.Vector()
39 tv.record(h._ref_t)
40 cv = h.Vector()
41 cv.record(vc._ref_i)
42
43 taxes = []
44 traces = []
45 for i in range(ntrace):
46     v_init = vstart + vstep * i
47     vc.amp[0]=v_init
48     vc.amp[1]=v_init
49     vc.amp[2]=v_init
50     soma.e_pas = v_init
51
52     h.finitialize(v_init)
53     h.fcurrent()
54     neuron.run(tstop)
55     traces.append(np.copy(cv.as_numpy()))
56     taxes.append(np.copy(tv.as_numpy()))
57
```

```
58 #tt = tv.as_numpy()
59
60 ax = plt.subplot()
61 ax.set_ylim([-0.10,0.10])
62 for i in range(ntrace):
63     ax.plot(taxes[i], traces[i])
64 #     ax.plot(tt, traces[i])
65 plt.xlabel('Time (ms)')
66 plt.ylabel('Current')
67
68 plt.savefig('../figs/n10.pdf')
69 plt.show()
```

## 1.11 n11.py

```
1 # n11.py
2 import subprocess
3 import numpy as np
4 import neuron
5 from neuron import h
6 import matplotlib.pyplot as plt
7
8 tstop = 200.0
9 h.dt = 0.01
10 v_init = -65.0
11
12 soma = h.Section()
13 dend = [h.Section() for i in range(10)]
14
15 soma.diam = 30.0
16 soma.L = 30.0
17 soma.Ra = 100.0
18 soma.nseg = 1
19 soma.insert('hh')
20 vcl = h.VClamp(0.5, sec=soma)
21
22 for i in range(10):
23     dend[i].L = 300.0
24     dend[i].Ra = 100.0
25     dend[i].nseg = 21
26     dend[i].diam = 2.0
27     dend[i].insert('hh')
28     dend[i].gl_hh *= 2.3
29     dend[i].connect(soma, 1.0, 0.0)
30
31 esyn = h.Exp2Syn(0.75, sec=dend[0])
32
33 # stimulation
34 stim = h.NetStim(0.5)
35 stim.interval = 50.0
36 stim.number = 2
37 stim.start = 50.0
38 stim.noise = 0
39
40 # synaptic connections
41 nclist = []
42 nclist.append(h.NetCon(stim, esyn, 0.0, 0, 0.001))
43
44 vcl.dur[0] = 10.0
45 vcl.dur[1] = 10.0
46 vcl.dur[2] = 180.0
47 vcl.amp[0] = v_init
48 vcl.amp[1] = v_init
49 vcl.amp[2] = v_init
50 vcl.gain = 1000.0
51 vcl.tau1 = 0.1
52 vcl.tau2 = 0.1
53
54 tv = h.Vector()
55 vs = h.Vector()
56 vd = h.Vector()
57 cv = h.Vector()
```

```

58
59 tv.record(h._ref_t)
60 vs.record(soma(0.5)._ref_v)
61 vd.record(dend[0](0.75)._ref_v)
62 cv.record(vcl._ref_i)
63
64 h.finitialize(v_init)
65 h.fcurrent()
66 neuron.run(tstop)
67
68 ax = plt.subplot()
69 ax.set_ylim([-70.0, -35.0])
70 ax.plot(tv.as_numpy(), 100*cv.as_numpy())
71 ax.plot(tv.as_numpy(), vs.as_numpy())
72 ax.plot(tv.as_numpy(), vd.as_numpy())
73 plt.xlabel('Time (ms)')
74 plt.ylabel('Current')
75 plt.savefig('../figs/n11.pdf')
76 plt.show()

```

## 1.12 n12.py

```
1 # n12.py
2
3 import math
4 import numpy as np
5 import neuron
6 from neuron import h
7 import matplotlib.pyplot as plt
8
9 h.load_file("nrngui.hoc")
10 h.load_file("stdlib.hoc")
11 h.load_file("../models/tc200.geo")
12
13 nSyn = 30
14 tstop = 100.0
15 dt = 0.01
16 v_init = -65.0
17
18 # Properties of dendrites
19 dendritic = []
20 for sec in h.allsec():
21     sec.insert('hh')
22     sec.Ra = 100
23     if sec.name() != 'soma':
24         dendritic.append(sec)
25 print('number of dend = ', len(dendritic))
26
27 for sec in dendritic:
28     d = 1.0e5 * math.sqrt(sec.diam/(4.0*h.PI*100.0*sec.Ra*sec.cm))
29     sec.nseg = int((int((sec.L/(0.1*d))+0.9)/2)*2+1)
30
31 # total dendritic length
32 totalLength = 0.0
33 for sec in dendritic:
34     totalLength += sec.L
35 print('Total dendrite length = ', totalLength)
36
37 # Shape Plot
38 sh = h.Shape(1)
39 sh.size(-150, 150, -150, 150)
40 # stimulation
41 stim = h.NetStim(0.5)
42 stim.interval = 50.0
43 stim.number = 2
44 stim.start = 20.0
45 stim.noise = 0
46
47 nclist = []
48 locvec = np.random.uniform(0.0, totalLength, nSyn)
49 esyn = []
50 nclist = []
51 for k in range(nSyn):
52     l=0
53     l1 = 0
54     found = 0
55     for sec in dendritic:
56         if found==0:
57             l1 += sec.L
```

```

58         if locvec[k] < 11:
59             lx = (locvec[k]-1)/sec.L
60             e = h.Exp2Syn(lx, sec=sec)
61             esyn.append(e)
62             sh.color(2, sec=sec)
63             sh.point_mark(esyn[k],3,'0',2)
64             nclist.append( h.NetCon(stim, esyn[k], 0.0, 0, 0.001))
65             found = 1
66             break
67         l = 11
68 sh.flush()
69 h.doEvents()
70 sh.printfile(' ../figs/n12shape.eps')
71
72 # main loop
73 cvode = h.CVode()
74 cvode.active(1)
75 cvode.atol(1.0e-5)
76
77 tv = h.Vector()
78 vs = h.Vector()
79 tv.record(h._ref_t)
80 vs.record(h.soma(0.5)._ref_v)
81
82 h.finitialize(v_init)
83 h.fcurrent()
84 neuron.run(tstop)
85
86 ax = plt.subplot()
87 #ax.set_ylim([-70.0, -35.0])
88 ax.plot(tv.as_numpy(), vs.as_numpy())
89 plt.savefig(' ../figs/n12.pdf')
90 plt.show()

```



## 1.13 n13.py

```
1 # n13.py
2 import numpy as np
3 import neuron
4 from neuron import h
5 import matplotlib.pyplot as plt
6
7 h.load_file('nrngui.hoc')
8 h.load_file("stdlib.hoc")
9 h.load_file('../models/tc200.geo')
10
11 tstop = 200.0
12 dt = 0.01
13 v_init = -65.0
14
15 nSyn=100      # number of synapses
16 mdel = 10.0   # mean of synaptic delay
17 sdel = 1.0    # sd of synaptic delay
18 w = 0.001     # synaptic weight
19 br = 5
20
21 def measureDist(sec):
22     dx = 0.0
23     sec1 = sec
24     while True:
25         sr = h.SectionRef(sec=sec1)
26         if sr.has_parent() < 1.0:
27             break
28         dx += sec1.L
29         sec1 = h.SectionRef(sec=sec1).parent
30     return dx
31
32 def countBranch(sec):
33     bx = 0
34     sec1 = sec
35     while True:
36         sr = h.SectionRef(sec=sec1)
37         if sr.has_parent() < 1.0:
38             break
39         bx += 1
40         print(sec1.name())
41         sec1 = h.SectionRef(sec=sec1).parent
42     return bx
43
44 #-----
45 # Properties of dendrites
46
47 dendritic = []
48 for sec in h.allsec():
49     sec.insert('hh')
50     sec.Ra = 100.0
51     if sec.name() != 'soma':
52         dendritic.append(sec)
53
54 for sec in dendritic:
55     d = h.lambda_f(100, sec=sec)
56     sec.nseg = int(int((sec.L/(0.1*d))+0.9)/2*2+1)
57
```

```

58 dendriticN = []
59 total_length=0
60 for sec in dendritic:
61     if countBranch(sec) >= br:
62         dendriticN.append(sec)
63         total_length += sec.L
64
65 print('total length of ">=', br, 'th-branch" dendrites = ', total_length)
66
67 # Shape Plot
68 sh = h.Shape(1)
69
70 # stimulation
71 stim = h.NetStim(0.5)
72 stim.interval = 50.0
73 stim.number = 2
74 stim.start = 50.0
75 stim.noise = 0
76
77 # random generator
78 locvec = np.random.uniform(0, total_length, nSyn)
79 delvec = np.random.normal(mdel, sdel, nSyn)
80 nc = []
81 esyn = []
82 for k in range(nSyn-1):
83     l = 0
84     l1 = 0
85     found = 0
86     for sec in dendriticN:
87         if found == 0:
88             l1 += sec.L
89             if l1 > locvec[k]:
90                 lx = (locvec[k]-l)/sec.L
91                 e = h.Exp2Syn(lx, sec=sec)
92                 esyn.append(e)
93                 sh.color(2, sec=sec)
94                 sh.point_mark(esyn[k],3,'0',2)
95                 nc.append(h.NetCon(stim, esyn[k], 0.0, delvec[k], w))
96                 found = 1
97                 break
98     l = l1
99 sh.flush()
100 h.doEvents()
101 sh.printfile('../figs/n13shape.eps')
102 # main loop
103
104 cvode = h.CVode()
105 cvode.active(1)
106 cvode.atol(1.0e-5)
107
108 tv = h.Vector()
109 vs = h.Vector()
110 tv.record(h._ref_t)
111 vs.record(h.soma(0.5)._ref_v)
112
113 h.finitialize(v_init)
114 h.fcurrent()
115
116 print('Simulation Start')
117 neuron.run(tstop)
118

```

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

## 1.14 n14.py

```
1 # n14.py
2
3 import numpy as np
4 import neuron
5 from neuron import h
6 import matplotlib.pyplot as plt
7
8 tstop = 40.0
9
10 soma = h.Section()
11 soma.insert('m01')
12
13 cnode = h.CVode()
14 cnode.active(1)
15 cnode.atol(1.0e-5)
16
17 tv = h.Vector()
18 zv = h.Vector()
19 tv.record(h._ref_t)
20 zv.record(soma(0.5)._ref_z_m01)
21
22 h.finitialize()
23 #h.fcurrent()
24 neuron.run(tstop)
25
26 ax = plt.subplot()
27 ax.plot(tv.as_numpy(), zv.as_numpy())
28 plt.xlabel('Time')
29 plt.ylabel('Z')
30 plt.savefig('../figs/n14.pdf')
31 plt.show()
```

## 1.15 n15.py

```
1 # n15.py Markov model of Na channel
2
3 import numpy as np
4 import neuron
5 from neuron import h
6 import matplotlib.pyplot as plt
7
8 nstates = 9
9
10 tstop = 10.0
11 v_init = -100.0
12
13 c = h.Section()
14 c.insert('mna')
15 h.dt = 0.05
16
17 vc = h.VClamp(0.5, sec=c)
18 vc.dur[0] = 2.0
19 vc.dur[1] = 5.0
20 vc.dur[2] = 3.0
21 vc.amp[0] = v_init
22 vc.amp[1] = 0.0
23 vc.amp[2] = v_init
24
25 tv = h.Vector()
26 zv = [h.Vector() for i in range(nstates)]
27
28 tv.record(h._ref_t)
29 for i in range(nstates):
30     s = 'zv[%1d].record(c(0.5)._ref_z%1d_mna)'% (i, i)
31     exec(s)
32
33 h.finitialize(v_init)
34 h.fcurrent()
35 neuron.run(tstop)
36
37 ax = plt.subplot()
38 for i in range(nstates):
39     ax.plot(tv.as_numpy(), zv[i].as_numpy())
40
41 plt.xlabel('Time')
42 plt.ylabel('Probability')
43 plt.savefig('../figs/n15.pdf')
44 plt.show()
```

## 1.16 n16.py

```
1 # n16.py  Markov model of Na channel
2 #          Voltage-dependent activation
3
4 import numpy as np
5 import neuron
6 from neuron import h
7 import matplotlib.pyplot as plt
8
9 nstates = 9
10
11 tstop = 20.0
12 h.dt = 0.01
13
14 v_init = -100.0
15 vstart = -100.0
16 vstep = 10.0
17 ntraces = 20
18 npts = int(tstop/h.dt)+1
19 ena = 50.0
20
21 c = h.Section()
22 c.insert('mna')
23
24 vc = h.VClamp(0.5, sec=c)
25 vc.dur[0] = 5.0
26 vc.dur[1] = 10.0
27 vc.dur[2] = 5.0
28 vc.amp[0] = v_init
29 vc.amp[1] = v_init
30 vc.amp[2] = v_init
31
32 tv = h.Vector()
33 tv.record(h._ref_t)
34 gv = h.Vector()
35 gv.record(c(0.5)._ref_z3_mna)
36 vv = h.Vector()
37 vv.record(c(0.5)._ref_v)
38
39 traces = []
40 vtra = []
41 for i in range(ntraces):
42     vc.amp[1] = vstart + vstep * i
43     h.finitialize(v_init)
44     h.fcurrent()
45     neuron.run(tstop)
46
47     traces.append(np.copy(gv.as_numpy()))
48     vtra.append(np.copy(vv.as_numpy()))
49
50 tt = tv.as_numpy()
51
52 f, ax = plt.subplots(2, sharex=True)
53 for i in range(ntraces):
54     ax[0].plot(tt, traces[i])
55 ax[0].set_ylabel('Open probability')
56
57 for i in range(ntraces):
```

```
58     ax[1].plot(tt, traces[i]*(vtra[i]-ena))
59 ax[1].set_xlabel('Time (ms)')
60 ax[1].set_ylabel('Current')
61
62 plt.savefig('../figs/n16.pdf')
63 plt.show()
```

## 1.17 n17.py

```
1 # n17.py
2
3 import neuron
4 from neuron import h
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 #h.load_file ("nrngui.hoc")
9 h.load_file ("../models/cellModel.hoc")
10
11 tstop = 100
12 v_init = -65
13 ncells = 20
14
15 cells = []
16 for i in range(ncells):
17     cells.append(h.BallStickCell( ))
18
19 nclist = []
20 for i in range(len(cells)):
21     target = cells[(i+1) % ncells] # next cell
22     nc = cells[i].connect2target(target.synlist[0])
23     nclist.append(nc)
24     nc.delay = 1
25     nc.weight[0] = 0.01
26
27 #external stimulus to syn of cells[0]
28 stim = h.NetStim()
29 stim.number = 1
30 stim.start = 0
31 ncstim = h.NetCon(stim, cells[0].synlist[0])
32 ncstim.delay=0
33 ncstim.weight[0] = 0.01
34
35 tvec = h.Vector( )
36 idvec = h.Vector( )
37 for i in range( len(cells) ):
38     nc = cells[i].connect2target(None)
39     nc.record (tvec, idvec, i)
40
41 # Simulation control
42 h.finitialize(v_init)
43 neuron.run(tstop)
44
45 print ("\ntime\t cell\n")
46 for i in range(int(tvec.size())):
47     print (i, tvec.x[i], idvec.x[i])
48 print ("tvec.size=",tvec.size())
49
50 ax1 = plt.subplot()
51 ax1.plot(tvec.as_numpy(), idvec.as_numpy(), '|')
52 plt.xlabel('Time (ms)')
53 plt.ylabel('Cell number')
54 plt.savefig('../figs/n17.pdf')
55 plt.show()
56
57 """
```



```
58 spikes = [np.empty(0) for i in range(ncells)]
59 for i in range(int(tvec.size())):
60     j = int(idvec.x[i])
61     spikes[j] = np.append(spikes[j],tvec.x[i])
62 ax1 = plt.subplot()
63 ax1.eventplot(spikes, colors=[[1.0, 0, 0]])
64 plt.show()
65 """
```

## 1.18 n18.py

```
1 # n18.py Random network
2
3 import neuron
4 from neuron import h
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import matplotlib
8
9 h.load_file ("../utils/ranstream.hoc")
10 h.load_file('../models/cellModel.hoc')
11
12 tstop = 200
13 h.dt = 0.01
14 vinit = -65
15 NCELL = 200 # total number of cells
16 C_E = 3 # # of excitatory connections received by each cell
17
18 connect_random_low_start_ = 1 # low seed for mcell_ran4_init ( )
19 cells = []
20 ranlist = [] # list of the random number sequences
21 h.mcell_ran4_init(connect_random_low_start_)
22
23 # create cells
24 for i in range(NCELL): # target (postsynaptic) cell
25     cells.append(h.BallStickCell( ))
26     ranlist.append(h.RandomStream(i))
27
28 # make connection
29 # no self-connection
30 # only one connection from any source
31 nclist = []
32 u = np.zeros(NCELL, dtype=int)
33 for i in range(NCELL): # target cell
34     syn = cells[i].synlist.object(0)
35     rs = ranlist[i]
36     rs.start ( )
37     rs.r.discunif(0, NCELL-1) # returns int in range 0..NCELL-1
38     u[:] = 0 # u.x[i]==1 means spike source i has already been chosen
39     nsyn = 0
40     while nsyn < C_E:
41         r = int(rs.repick( ))
42         if (r != i) and (u[r] == 0):
43             nc = cells[r].connect2target(syn)
44             nclist.append (nc)
45             nc.delay = 1
46             nc.weight[0] = 0.01
47             u[r] = 1
48             nsyn += 1
49             # print("cell %2d -> cell %2d (syn %d)" % (r, i, 0))
50
51 print(len(nclist), " connections")
52
53 stim = h.NetStim()
54 stim.number = 1
55 stim.start = 0
56 ncstim = h.NetCon (stim, cells[0].synlist[0])
57 ncstim.delay = 0
```

```

58 ncstim.weight[0] = 0.01
59
60 tvec = h.Vector( )
61 idvec = h.Vector( )
62 for i in range(len(cells)):
63     nc = cells[i].connect2target(None)
64     nc.record(tvec, idvec, i)
65     nc.record(tvec, idvec, i)
66
67 #for trace
68 ttvec = h.Vector( )
69 vvec = h.Vector()
70 ttvec.record(h._ref_t)
71 vvec.record(cells[0].soma(0.5)._ref_v)
72
73 # Simulation control
74 h.finitialize(vinit)
75 neuron.run(tstop )
76
77 # Report simulation results
78 def spikeout ( ):
79     print("tvec size = ", int(tvec.size()))
80     print("\ntime\t cell\n")
81     for i in range(int(tvec.size())):
82         print(i, tvec.x[i], idvec.x[i])
83
84 # spikeout ( )
85
86 spikes = [np.empty(0) for i in range(NCELL)]
87 for i in range(int(tvec.size())):
88     j = int(idvec.x[i])
89     spikes[j] = np.append(spikes[j],tvec.x[i])
90
91 ax1 = plt.subplot()
92 ax1.eventplot(spikes, colors=[[1.0, 0, 0]])
93 plt.ylabel('Cell number')
94 plt.xlabel('Time (ms)')
95 plt.savefig('../figs/n18.pdf')
96 #plt.show(block=False)
97 #plt.pause(3)
98 plt.show()

```

## 1.19 n19.py

```
1 # n19.py
2
3 import neuron
4 from neuron import h
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 class InvlFire(object):
9     def __init__(self):
10         self.cell = h.Section()
11         self.IF = h.IntervalFire(0.5, sec=self.cell)
12         self.IF.tau = 5.0
13         self.IF.invl = 10.0
14         self.synlist = h.List()
15
16
17 tstop = 100.0
18 h.dt = 0.1
19
20 ce = InvlFire()
21
22 # external stimulation
23 stim = h.NetStim()
24 stim.number = 40
25 stim.start = 25
26 stim.interval = 1
27
28 ncstim = h.NetCon(stim, ce.IF)
29 ncstim.delay = 0
30 ncstim.weight[0] = 0.07
31
32 # record for spikes
33 nc = h.NetCon(ce.IF, None)
34 stvec = h.Vector()
35 nc.record(stvec)
36
37 # run control
38 h.finitialize()
39 tt = []
40 mm = []
41 MM = []
42 while h.t < tstop:
43     tt.append(h.t)
44     mm.append(ce.IF.m)
45     MM.append(ce.IF.M())
46     h.fadvance()
47
48 f, ax = plt.subplots(2, sharex=True)
49 ax[0].plot(tt, mm)
50 ax[0].plot(tt, MM)
51 ax[0].set_ylabel('Values of m and M')
52
53 ax[1].eventplot(stvec.as_numpy())
54 ax[1].set_xlabel('Time (ms)')
55 ax[1].set_ylabel('Events')
56
57 plt.savefig('../figs/n19.pdf')
```

```
58 plt.show()
```

## 1.20 n20.py

```
1 # n20.py
2
3 from neuron import h
4 import neuron
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 class InvlFire(object):
9     def __init__(self):
10         self.cell = h.Section()
11         self.IF = h.IntervalFire(0.5, sec=self.cell)
12         self.IF.tau = 5.0
13         self.IF.invl = 10.0
14
15 tstop = 200.0
16 ncells = 600
17 exRatio = 0.8
18 inhRatio = 1.0 - exRatio
19 exCells = int(exRatio*ncells)
20 inhCells = ncells - exCells
21 conRateE = 0.005
22 conRateI = 0.05
23 synWeightE = 0.01
24 synWeightI = -0.18
25
26 tbin = 2.0
27 nbin = int((tstop/tbin))+1
28
29 print("excitatory cells: ", exCells)
30 print("inhibitory cells: ", inhCells)
31
32 print("estimated excitatory synapses: ", exCells * ncells * conRateE)
33 print("estimated inhibitory synapses: ", inhCells * ncells * conRateI)
34
35
36 cells = [InvlFire() for i in range(ncells)]
37 print(len(cells))
38
39 """
40 # clock-wise connections
41 nclist = []
42 for i in range(ncells):
43     target = cells[(i+1) % ncells] # next cell
44     nc = h.NetCon(cells[i].IF, target.IF)
45     nclist.append(nc)
46     nc.delay = 5
47     nc.weight[0] = 0.5
48 """
49 # random connections
50 nclist = []
51 used = np.zeros(ncells)
52 for i in range(ncells):
53     if(i < exCells):
54         maxConnection = int(ncells*conRateE)
55         synWeight = synWeightE
56     else:
57         maxConnection = int(ncells*conRateI)
```

```

58         synWeight = synWeightI
59
60     ncon = 0
61     j = 0
62     used[:] = 0
63     trand = np.random.randint(0, ncells, size=ncells)
64     while ncon < maxConnection:
65         k = trand[j]
66         j += 1
67         if(used[k]==0):
68 #             if(i < excCells):
69 #                 print(i," ==> ", k)
70 #             else:
71 #                 print(i," --> ", k)
72
73         nc = h.NetCon(cells[i].IF, cells[k].IF)
74         nc.list.append(nc)
75         nc.delay = 3
76         nc.weight[0] = synWeight
77         used[k] = 1
78         ncon = ncon+1
79
80 # randomize initial state
81 stim = h.NetStim()
82 stim.number = 1
83 stim.start = 0
84 stim.interval = 1
85 ncini = []
86 mr = np.random.uniform(0, 1.0, ncells)
87 for i in range(ncells):
88     ncstim = h.NetCon(stim, cells[i].IF)
89     ncstim.delay= 0
90     ncstim.weight[0] = mr[i]
91     ncini.append(ncstim)
92
93 # record for spikes
94 stvec = h.Vector( )
95 idvec = h.Vector( )
96 for i in range( len(cells) ):
97     nc = h.NetCon(cells[i].IF, None)
98     nc.record (stvec, idvec, i)
99
100 # Simulation control
101 h.finitialize()
102 neuron.run(tstop)
103
104 #print ("\ntime\t cell\n")
105 #for i in range(int(tvec.size())):
106 #    print (i, tvec.x[i], idvec.x[i])
107 #print("tvec.size=",tvec.size())
108 hg = np.zeros(nbin, dtype=int)
109 hge = np.zeros(nbin, dtype=int)
110 hgi = np.zeros(nbin, dtype=int)
111 ht = np.linspace(0, tstop, nbin)
112 for i in range(int(stvec.size())):
113     ix = int(stvec.x[i]/tbin)
114     hg[ix] = hg[ix] + 1
115     if(idvec.x[i]<exCells):
116         hge[ix] = hge[ix] + 1
117     else:
118         hgi[ix] = hgi[ix] + 1

```

```

119 hge = hge.astype(float)/exCells
120 hgi = hgi.astype(float)/inhCells
121
122 f, ax = plt.subplots(3, sharex=True)
123 ax[0].plot(stvec.as_numpy(), idvec.as_numpy(), '.', markersize=2)
124 ax[0].set_ylabel('Cell number')
125 ax[1].plot(ht, hg)
126 ax[1].set_ylabel('No of spikes')
127 ax[2].plot(ht, hge)
128 ax[2].plot(ht, hgi)
129 ax[2].set_xlabel('Time (ms)')
130 ax[2].set_ylabel('Probability')
131
132 plt.savefig('../figs/n20.pdf')
133 plt.show()
134
135
136 """
137 spikes = [np.empty(0) for i in range(ncells)]
138 for i in range(int(stvec.size())):
139     j = int(idvec.x[i])
140     spikes[j] = np.append(spikes[j], stvec.x[i])
141 ax1 = plt.subplot()
142 ax1.eventplot(spikes, colors=[[1.0, 0, 0]])
143 plt.show()
144 """

```



## 1.21 n21.py

```
1
2 # n21.py collision experiment
3
4 from neuron import h
5 import neuron
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import matplotlib.animation as animation
9
10 nsegs = 10000
11 ulen = 5.0      # unit length in micro-meter
12 totalLength = ulen * nsegs
13 npts = nsegs + 1
14 h.dt = 0.025
15 dl = 1.0/float(nsegs)
16
17 fCollision = False #True
18
19 v_init = -65.0
20 tstop = 100.0
21
22 axon = h.Section()
23 axon.L = totalLength
24 axon.diam = 2.0
25 axon.nseg = npts
26 axon.insert("hh")
27
28 ccode = h.CCode()
29 ccode.active(1)
30 ccode.atol(1.0e-5)
31
32 ic1 = h.IClamp(0.0, sec=axon)
33 ic1.delay = 1.0 # ms ic.dur = 100.0 # ms
34 ic1.amp = 0.3 # nA
35 ic1.dur = 1.0
36
37 if fCollision == True:
38     ic2 = h.IClamp(1.0, sec=axon)
39     ic2.delay = 1.0 # ms ic.dur = 100.0 # ms
40     ic2.amp = 0.3 # nA
41     ic2.dur = 1.0
42
43
44 yy = []
45 h.finitialize(v_init)
46 h.fcurrent()
47
48 yv = np.zeros(npts)
49
50 while h.t < tstop:
51     for i in range(npts):
52         yv[i] = axon(dl*i).v
53     yy.append(np.copy(yv))
54     ccode.fixed_step()
55
56 ntraces = len(yy)
57
```

```

58 fig, ax = plt.subplots(figsize=(10,2))
59 xv = np.linspace(0.0, totalLength, npts)
60 yv = np.zeros(npts)
61 line, = ax.plot(xv,yv)
62 ax.set_ylim(-100.0,50.0)
63 nframes = 5
64 def update(i):
65     ydata = yy[nframes*i]
66     line.set_ydata(ydata)
67     return line,
68
69 ani = animation.FuncAnimation(fig, update, frames = int(ntraces/nframes), interval=30)
70 plt.show()

```

## 2 hoc source code

### 2.1 n01.hoc

```
1 // n01.hoc: a simple hh cell
2
3 create soma
4 soma insert hh
5
6 objref cvode
7 cvode = new CVode()
8 cvode.active(1)
9 cvode.atol(1.0e-5)
10
11 objref vv, tv
12 vv = new Vector()           // membrane potential vector
13 tv = new Vector()           // time stamp vector
14 vv.record(&soma.v(0.5))
15 tv.record(&t)
16
17 tstop = 200.0
18 v_init = -65.0
19 finitialize(v_init)
20 fcurrent()
21 run(tstop)
22
23 objref g
24 g = new Graph()
25 g.size(0, tstop, -100, 50)
26 vv.plot(g, tv, 2, 1)
27 g.flush()
```

## 2.2 n02.hoc

```
1 // n02.hoc: a simple hh cell
2
3 create soma
4 soma {
5     insert hh
6     el_hh = -30.0 // <= changed
7 }
8
9 objref cvode
10 cvode = new CVode()
11 cvode.active(1)
12 cvode.atol(1.0e-5)
13
14 objref vv, tv
15 vv = new Vector() // membrane potential vector
16 tv = new Vector() // time stamp vector
17 vv.record(&soma.v(0.5))
18 tv.record(&t)
19
20 tstop = 200.0
21 v_init = -65.0
22 finitialize(v_init)
23 fcurrent()
24 run(tstop)
25
26 objref g
27 g = new Graph()
28 g.size(0, tstop, -100, 50)
29 vv.plot(g, tv, 2, 1)
30 g.flush()
```

## 2.3 n03.hoc

```
1 // n03.hoc: a simple hh cell,current injection
2
3 create soma
4 soma insert hh
5
6 objref ic
7 soma ic = new IClamp(0.5)
8 ic.del = 20.0 // ms
9 ic.dur = 100.0 // ms
10 ic.amp = 10 // nA
11
12 objref cnode
13 cnode = new CVode()
14 cnode.active(1)
15 cnode.atol(1.0e-5)
16
17 objref vv, tv, cv
18 vv = new Vector() // membrane potential vector
19 tv = new Vector() // time stamp vector
20 cv = new Vector()
21 vv.record(&soma.v(0.5))
22 tv.record(&t)
23 cv.record(&ic.i)
24
25 tstop = 200.0
26 v_init = -65.0
27 finitialize(v_init)
28 fcurrent()
29 run(tstop)
30
31 objref g
32 g = new Graph()
33 g.size(0, tstop, -100, 50)
34 vv.plot(g, tv, 2, 1)
35 cv.plot(g, tv, 3, 1)
36 g.flush()
```

## 2.4 n04.hoc

```
1 // n04.hoc: soma + dendrite
2
3 load_file("nrngui.hoc")
4 tstop = 200.0
5 v_init = -65.0
6
7 create soma
8 soma {
9     L = 30.0
10    diam = 30.0
11    nseg = 1
12    insert hh
13 }
14 create ap_dend
15 ap_dend {
16     L = 500.0
17     diam = 2.0
18     nseg = 23
19     insert hh
20     gnabar_hh = 0.012
21     gkbar_hh = 0.0036
22     gl_hh = 0.00003
23 }
24 connect ap_dend(0.0), soma(1.0)
25
26 objref ic
27 soma ic = new IClamp(0.5)
28 ic.del = 20.0 // ms
29 ic.dur = 100.0 // ms
30 ic.amp = 1 // nA
31
32 objref cvode
33 cvode = new CVode()
34 cvode.active(1)
35 cvode.atol(1.0e-5)
36
37 objref tv, vs, va1, va2, cv
38 tv = new Vector()
39 vs = new Vector()
40 va1 = new Vector()
41 va2 = new Vector()
42 cv = new Vector()
43
44 tv.record(&t)
45 vs.record(&soma.v(0.5))
46 va1.record(&ap_dend.v(0.1))
47 va2.record(&ap_dend.v(0.9))
48 cv.record(&ic.i)
49
50 finitialize(v_init)
51 fcurrent()
52 run(tstop)
53
54 objref g
55 g = new Graph()
56 g.size(0, tstop, -100, 50)
57 vs.plot(g, tv, 2, 1)
```

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

## 2.5 n05.hoc

```
1 // n05.hoc
2
3 tstop = 25
4 v_init = -65.0
5
6 create soma
7 soma {
8     L = 30.0
9     diam = 30.0
10    nseg = 1
11    insert hh
12 }
13
14 create ap_dend
15 ap_dend {
16     L = 500.0
17     diam = 2.0
18     nseg = 23
19     insert hh
20     gnabar_hh = 0.012
21     gkbar_hh = 0.0036
22     gl_hh = 0.00003
23 }
24
25 connect ap_dend(0), soma(1.0)
26
27 // synaptic input
28 objref esyn
29 ap_dend esyn = new AlphaSynapse(0.5)
30 esyn.onset = 5
31 esyn.tau = 0.1
32 esyn.gmax = 0.05
33 esyn.e = 0.0
34
35 objref tv, vs, va1, va2
36 tv = new Vector()
37 vs = new Vector()
38 va1 = new Vector()
39 va2 = new Vector()
40
41 tv.record(&t)
42 vs.record(&soma.v(0.5))
43 va1.record(&ap_dend.v(0.1))
44 va2.record(&ap_dend.v(0.9))
45
46 objref ccode
47 ccode = new CCode()
48 ccode.active(1)
49 ccode.atol(1.0e-5)
50
51 finitialize(v_init)
52 fcurrent()
53 run(tstop)
54
55 objref g
56 g = new Graph()
57 g.size(0, tstop, -100, 50)
```



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

## 2.6 n06.hoc

```
1 // n06.py excitatory and inhibitory inputs
2
3 tstop = 40.0
4 dt = 0.01
5 v_init = -65
6
7 create soma
8 soma {
9     L = 30
10    diam = 30
11    nseg = 1
12    insert hh
13 }
14
15 create ap_dend
16 ap_dend {
17     L = 500
18     diam = 2
19     nseg = 23
20     insert hh
21     gnabar_hh = 0.012
22     gkbar_hh = 0.0036
23     gl_hh = 0.00003
24 }
25
26 connect ap_dend(0), soma(1)
27
28 ap_dend objref esyn
29 esyn = new AlphaSynapse(0.1)
30 esyn.onset = 10
31 esyn.tau = 0.1
32 esyn.gmax = 0.04
33 esyn.e = 0.0
34
35 objref isyn_p
36 ap_dend isyn_p = new AlphaSynapse(0.05)
37 isyn_p.onset = 5
38 isyn_p.tau = 3.0
39 isyn_p.e = -70 // v_init
40
41 objref isyn_d
42 isyn_d = new AlphaSynapse(0.95)
43 isyn_d.onset = 5
44 isyn_d.tau = 3.0
45 isyn_d.e = -70 // v_init
46
47 objref tv, vs
48 tv = new Vector()
49 vs = new Vector()
50 tv.record(&t)
51 vs.record(&soma.v(0.5))
52
53 objref tr, tt
54 tr = new List()
55 tt = new List()
56
57 objref ccode
```

```

58 cvode = new CVode()
59 cvode.active(1)
60 cvode.atol(1.0e-5)
61
62
63 for i = 0, 2 {
64     if(i == 0){
65         isyn_p.gmax = 0.0
66         isyn_d.gmax = 0.0
67     }
68     if(i == 1){
69         isyn_p.gmax = 0.03
70         isyn_d.gmax = 0.0
71     }
72     if(i == 2){
73         isyn_p.gmax = 0.0
74         isyn_d.gmax = 0.03
75     }
76     finitialize(v_init)
77     fcurrent()
78     while(t < tstop){
79         cvode.fixed_step()
80     }
81     tt.append(tv.c())
82     tr.append(vs.c())
83 }
84
85 objref g
86 g = new Graph()
87 g.size(0, tstop, -100, 50)
88 tr.o(0).plot(g, tt.o(0), 2, 1)
89 tr.o(1).plot(g, tt.o(1), 3, 1)
90 tr.o(2).plot(g, tt.o(2), 4, 1)
91 g.flush()

```

## 2.7 n07.hoc

```
1 // n07.py
2
3 tstop = 40.0
4 dt = 0.01
5 v_init = -65
6
7 create soma
8 soma {
9     L = 30
10    diam = 30
11    nseg = 1
12    insert hh
13 }
14
15 create ap_dend
16 ap_dend {
17     L = 500
18     diam = 2
19     nseg = 23
20     insert hh
21     gnabar_hh = 0.012
22     gkbar_hh = 0.0036
23     gl_hh = 0.00003
24 }
25 connect ap_dend(0), soma(1)
26
27 objref esyn
28 ap_dend esyn = new AlphaSynapse(0.1)
29 esyn.onset = 10
30 esyn.tau = 0.1
31 esyn.gmax = 0.04
32 esyn.e = 0.0
33
34
35 objref isyn, isyns
36 isyns = new List()
37 for i = 0, 9 {
38     ap_dend isyn = new AlphaSynapse(0.1*i+0.05)
39     isyn.onset = 5
40     isyn.tau = 3.0
41     isyn.e = -70
42     isyns.append(isyn)
43 }
44
45 objref tv, vs
46 tv = new Vector()
47 vs = new Vector()
48 tv.record(&t)
49 vs.record(&soma.v(0.5))
50
51 objref tr, tt
52 tr = new List()
53 tt = new List()
54
55 ccode = new CCode()
56 ccode.active(1)
57 ccode.atol(1.0e-5)
```

```

58
59 // synaptic input
60 for j = -1, 9 {                      // -1 for excitatory input only
61     for i = 0, 9 {
62         if(i == j){
63             isyns.o(i).gmax = 0.02
64         }else{
65             isyns.o(i).gmax = 0
66         }
67     }
68
69     finitialize(v_init)
70     fcurrent()
71     while(t < tstop){
72         ccode.fixed_step()
73     }
74     tt.append(tv.c())
75     tr.append(vs.c())
76 }
77
78 objref g
79 g = new Graph()
80 g.size(0, tstop, -100, 50)
81 for i = 0, 10 {
82     cl = (i % 8) + 1
83     tr.o(i).plot(g, tt.o(i), cl, 1)
84 }
85 g.flush()

```

## 2.8 n08.hoc

```
1 // n08.hoc: connecting neurons
2
3 begintemplate HHneuron
4     public soma, ap_dend, esyn
5     create soma, ap_dend
6     objref esyn
7
8     proc init() {
9         soma {
10             L = 30
11             diam = 30
12             nseg = 1
13             insert hh
14         }
15         ap_dend {
16             L = 500
17             diam = 2
18             nseg = 23
19             insert hh
20             gnabar_hh = 0.012
21             gkbar_hh = 0.0036
22             gl_hh = 0.00003
23         }
24
25         connect ap_dend(0), soma(1)
26         ap_dend esyn = new Exp2Syn(0.5)
27         esyn.tau1 = 0.5
28         esyn.tau2 = 1.0
29         esyn.e = 0
30     }
31 endtemplate HHneuron
32
33
34 // 2 cells
35 objref hh_neuron[2]
36 hh_neuron[0] = new HHneuron()
37 hh_neuron[1] = new HHneuron()
38
39 // synapse
40 objref stim
41 stim = new NetStim(0.5)
42 stim.interval = 20
43 stim.number = 3
44 stim.start = 20
45 stim.noise = 0
46
47 // connections
48 objref nclist
49 nclist = new List()
50 nclist.append(new NetCon(stim, hh_neuron[0].esyn, 0.0, 0, 0.02))
51 hh_neuron[0].soma nclist.append(new NetCon(&v(0.5), hh_neuron[1].esyn, 10, 1, 0.02))
52
53 tstop = 100.0
54 v_init = -65.0
55
56 objref ccode
57 ccode = new CCode()
```

```

58 ccode.active(1)
59 ccode.atol(1.0e-5)
60
61 objref tv, vs0, vs1
62 tv = new Vector()
63 vs0 = new Vector()
64 vs1 = new Vector()
65
66 tv.record(&t)
67 vs0.record(&hh_neuron[0].soma.v(0.5))
68 vs1.record(&hh_neuron[1].soma.v(0.5))
69
70 finitialize(v_init)
71 fcurrent()
72 run(tstop)
73
74 objref g
75 g = new Graph()
76 g.size(0, tstop, -100, 50)
77 vs0.plot(g, tv, 2, 1)
78 vs1.plot(g, tv, 3, 1)
79 g.flush()

```

## 2.9 n09.hoc

```
1 // n09.hoc: voltage clamp
2
3
4 create soma
5 soma {
6     L = 30.0
7     diam = 30.0
8     nseg = 1
9     insert hh
10 }
11 objref esyn, vcl
12 soma esyn = new Exp2Syn(0.5)
13 soma vcl = new VClamp(0.5)
14
15 objref stim
16 stim = new NetStim(0.5)
17 stim.interval = 50
18 stim.number = 2
19 stim.start = 50
20 stim.noise = 0
21
22 objref nclist
23 nclist = new List()
24 nclist.append( new NetCon(stim, esyn, 0.0, 0, 0.005))
25
26 tstop = 200.0
27 dt = 0.01
28 v_init = -65.0
29
30 vcl.dur[0] = 10.0
31 vcl.dur[1] = 10.0
32 vcl.dur[2] = 180.0
33 vcl.amp[0] = v_init
34 vcl.amp[1] = v_init
35 vcl.amp[2] = v_init
36 vcl.gain = 1000.0
37 vcl.tau1 = 0.01
38 vcl.tau2 = 0.01
39
40 objref tv, vv, cv
41 tv = new Vector()
42 vv = new Vector()
43 cv = new Vector()
44
45 tv.record(&t)
46 vv.record(&soma.v(0.5))
47 cv.record(&vcl.i)
48
49 finitialize(v_init)
50 run(tstop)
51
52 objref g
53 g = new Graph()
54 g.size(0, tstop, -300, 20)
55 vv.plot(g, tv, 2, 1)
56 cv = cv.mul(1000)
57 cv.plot(g, tv, 3, 1)
```



58 `g.flush()`

## 2.10 n10.hoc

```
1 // n10.hoc
2 // iv in voltage-clamp mode
3
4
5 tstop = 50.0
6 dt = 0.01
7 vstart = -100.0
8 vstep = 10.0
9 ntrace = 20
10 npts = int(tstop/dt)+1
11
12 create soma
13 soma {
14     diam = 30.0
15     L = 30.0
16     nseg = 1
17     insert pas
18 }
19
20 objref vc
21 soma vc = new VClamp(0.5)
22 vc.dur[0] = 10.0
23 vc.dur[1] = 10.0
24 vc.dur[2] = 30.0
25
26 objref syn
27 soma syn = new Exp2Syn(0.5)
28 syn.e = 0
29
30 objref ns
31 ns = new NetStim()
32 ns.number = 1
33 ns.start = 10.0
34 ns.noise = 0
35
36 objref nc
37 nc = new List()
38 nc.append(new NetCon(ns, syn, 0, 0, 0.001)) // 1 nS
39
40 objref tv, cv
41 tv = new Vector()
42 tv.record(&t)
43 cv = new Vector()
44 cv.record(&vc.i)
45
46 objref traces[ntrace]
47 for i = 0, ntrace-1 {
48     v_init = vstart + vstep * i
49     vc.amp[0]=v_init
50     vc.amp[1]=v_init
51     vc.amp[2]=v_init
52     soma.e_pas = v_init
53
54     finitialize(v_init)
55
56     fcurrent()
57     run(tstop)
```

```
58     traces[i] = cv.c()
59 }
60
61 objref g
62 g = new Graph()
63 g.size(0, tstop, -0.1, 0.1)
64 for i = 0, ntrace-1 {
65     j = (i % 8) + 1
66     traces[i].plot(g, tv, j, 1)
67 }
68 g.flush()
```

## 2.11 n11.hoc

```
1 // n11.hoc
2
3 create soma
4 soma {
5     L = 30.0
6     Ra = 100.0
7     nseg = 1
8     diam = 30.0
9     insert hh
10 }
11
12 create dend[10]
13 for i = 0, 9 {
14     dend[i] {
15         L = 300.0
16         Ra = 100.0
17         nseg = 21
18         diam = 2.0
19         insert hh
20         gl_hh *= 2.3
21     }
22     connect dend[i](0), soma(1.0)
23 }
24
25 objref vcl
26 soma vcl = new VClamp(0.5)
27
28 objref esyn
29 dend[0] esyn = new Exp2Syn(0.75)
30
31 // stimulation
32 objref stim
33 stim = new NetStim(0.5)
34 stim.interval = 50.0
35 stim.number = 2
36 stim.start = 50.0
37 stim.noise = 0
38
39 // synaptic connections
40 objref nclist
41 nclist = new List()
42 nclist.append(new NetCon(stim, esyn, 0.0, 0, 0.001))
43
44 tstop = 200.0
45 dt = 0.01
46 v_init = -65.0
47
48 vcl.dur[0] = 10.0
49 vcl.dur[1] = 10.0
50 vcl.dur[2] = 180.0
51 vcl.amp[0] = v_init
52 vcl.amp[1] = v_init
53 vcl.amp[2] = v_init
54 vcl.gain = 1000.0
55 vcl.tau1 = 0.1
56 vcl.tau2 = 0.1
57
```

```

58 objref tv, vs, vd, cv
59 tv = new Vector()
60 vs = new Vector()
61 vd = new Vector()
62 cv = new Vector()
63
64 tv.record(&t)
65 vs.record(&soma.v(0.5))
66 vd.record(&dend[0].v(0.75))
67 cv.record(&vcl.i)
68
69 finitialize(v_init)
70 fcurrent()
71 run(tstop)
72
73 objref g
74 g = new Graph()
75 g.size(0, tstop, -70, -35)
76 vs.plot(g, tv, 2, 1)
77 vd.plot(g, tv, 3, 1)
78 cv = cv.mul(100)
79 cv.plot(g, tv, 4, 1)
80 g.flush()

```

## 2.12 n12.hoc

```
1 // n12.hoc
2
3 load_file("stdlib.hoc")
4 load_file("../models/tc200.geo")
5
6 nSyn = 30
7 tstop = 100.0
8 dt = 0.01
9 v_init = -65.0
10
11
12 begintemplate TimeClock
13     public getSec
14     double jx[3]
15     strdef datetime
16     func getSec(){ // returns time in sec
17         system("date ' +%H %M %' S", datetime)
18         sscanf(datetime,"%d%d%d", &jx[0], &jx[1], &jx[2])
19         return (jx[0]*60 + jx[1])*60 + jx[2]
20     }
21 endtemplate TimeClock
22
23
24 func measureDist(){local dx localobj sr
25     dx = 0
26     sr = new SectionRef()
27     while(sr.has_parent){
28         dx += L
29         access sr.parent
30         sr = new SectionRef()
31     }
32     return dx
33 }
34
35 // Properties of dendrites
36
37 objref dendritic
38 dendritic = new SectionList()
39 forall{
40     insert hh
41     Ra = 100
42     dendritic.append()
43 }
44 soma dendritic.remove()
45
46 forsec dendritic {
47     d = lambda_f(100)
48     nseg = int((L/(0.1*d))+0.9)/2*2+1
49 }
50
51 // total dendritic length
52 totalLength = 0.0
53 forsec dendritic {
54     totalLength += L
55 }
56 print "Total dendrite length = ", totalLength
57
```

```

58 // Shape Plot
59 objref sh
60 sh = new Shape(1)
61 sh.size(-150, 150, -150, 150)
62
63 // stimulation
64 objref stim
65 stim = new NetStim(0.5)
66 stim.interval = 50.0
67 stim.number = 2
68 stim.start = 20.0
69 stim.noise = 0
70
71 objref nclist
72 nclist = new List()
73
74 // random generator
75 objref tx, r, locvec
76 tx = new TimeClock()
77 r = new Random(tx.getSec())
78 r.uniform(0, totalLength)
79 locvec = new Vector(nSyn)
80 locvec.setrand(r)
81
82
83 objref sref, esyn[nSyn]
84 for k = 0, nSyn-1 {
85     l=0
86     l1 = 0
87     found = 0
88     forsec dendritic {
89         if(found==0){
90             l1 += L
91             if(l1>locvec.x(k)){
92                 lx = (locvec.x(k)-1)/L
93                 sref = new SectionRef()
94                 print secname(), measureDist()
95                 sref.sec {
96                     esyn[k] = new Exp2Syn(lx)
97                     sh.color(2)
98                 }
99                 sh.point_mark(esyn[k],3)
100                 nclist.append( new NetCon(stim, esyn[k], 0.0, 0, 0.001))
101                 found = 1
102                 break
103             }
104             l = l1
105         }
106     }
107 }
108 sh.flush()
109 doEvents()
110
111 for k =0, nSyn-1 {
112     l = 0
113     l1 = 0
114     found = 0
115     forsec dendritic {
116         if(found==0){
117             l1 += L
118             if(l1>locvec.x(k)){

```

```

119         lx = (locvec.x(k)-1)/L
120         sref = new SectionRef()
121         print secname(), measureDist()
122         sref.sec {
123             esyn[k] = new Exp2Syn(lx)
124             sh.color(2)
125         }
126         sh.point_mark(esyn[k],3)
127         nclist.append( new NetCon(stim, esyn[k], 0.0, 0, 0.001))
128         found = 1
129         break
130     }
131     l = l1
132 }
133 }
134 }
135 sh.flush()
136 doEvents()
137
138
139 // main loop
140 objref cnode
141 cnode = new CNode()
142 cnode.active(1)
143 cnode.atol(1.0e-5)
144
145 objref tv, vs
146 tv = new Vector()
147 vs = new Vector()
148 tv.record(&t)
149 vs.record(&soma.v(0.5))
150
151 finitialize(v_init)
152 fcurrent()
153 run(tstop)
154
155 objref g
156 g = new Graph()
157 g.size(0, tstop, -100, 50)
158 vs.plot(g, tv, 2, 1)
159 g.flush()

```



## 2.13 n13.hoc

```
1 // n13.hoc
2
3
4 load_file("../models/tc200.geo")
5
6 tstop = 200.0
7 dt = 0.01
8 v_init = -65.0
9
10 nSyn=100 // number of synapses
11 mdel = 10.0 // mean of synaptic delay
12 sdel = 1.0 // sd of synaptic delay
13 w = 0.001 // synaptic weight
14 br = 5
15
16 objref dendritic, dendriticN
17 objref sh
18 objref r, vloc, vdel, tx
19 objref esyn[nSyn]
20 objref sref
21 strdef s
22
23 begintemplate TimeClock
24     public getSec
25     double jx[3]
26     strdef datetime
27     func getSec(){ // returns time in sec
28         system("date ' +%H %M %' S", datetime)
29         sscanf(datetime,"%d%d%d", &jx[0], &jx[1], &jx[2])
30         return (jx[0]*60 + jx[1])*60 + jx[2]
31     }
32 endtemplate TimeClock
33
34 func measureDist(){local dx localobj sr, srsave
35     dx = 0
36     srsave = new SectionRef()
37     sr = new SectionRef()
38     while(sr.has_parent){
39         dx += L
40         access sr.parent
41         sr = new SectionRef()
42     }
43     access srsave.sec
44     return dx
45 }
46
47
48 func countBranch(){local bx localobj sr, srsave
49     bx = 0
50     srsave = new SectionRef()
51     sr = new SectionRef()
52     while(sr.has_parent){
53         if(sr.nchild==2){
54             bx += 1
55         }
56         access sr.parent
57         sr = new SectionRef()
```

```

58     }
59     access srsave.sec
60     return bx
61 }
62
63
64
65 // -----
66 // Properties of dendrites
67
68 dendritic = new SectionList()
69 forall {
70     insert hh
71     Ra = 100
72     dendritic.append()
73 }
74 soma dendritic.remove()
75
76 forsec dendritic {
77     d = lambda_f(100)
78     nseg = int((L/(0.1*d))+0.9)/2*2+1
79 }
80
81 dendriticN = new SectionList()
82 total_length=0
83 forall {
84     if(countBranch()>=br){
85         dendriticN.append()
86         total_length += L
87     }
88 }
89 print "total length of \">=", br, "-branch\" dendrites = ", total_length
90
91
92
93
94 // Shape Plot
95
96 sh = new Shape(1)
97 //-----
98
99 // stimulation
100 objref stim
101 stim = new NetStim(0.5)
102 stim.interval = 50
103 stim.number = 2
104 stim.start = 50
105 stim.noise = 0
106 //-----
107 // random generator
108 tx = new TimeClock()
109 r = new Random(tx.getSec())
110 r.uniform(0, total_length)
111 vloc = new Vector(nSyn)
112 vloc.setrand(r)
113 r.normal(mdel,sdel*sdel)
114 vdel = new Vector(nSyn)
115 vdel.setrand(r)
116
117 objref nc
118 nc = new List()

```

```

119 for k = 0, nSyn-1 {
120     l=0
121     l1 = 0
122     found = 0
123     forsec dendriticN {
124         if(found==0){
125             l1 += L
126             if(l1>vloc.x(k)){
127                 lx = (vloc.x(k)-l)/L
128                 sref = new SectionRef()
129                 print secname(), ", dist=", measureDist(), \
130                     ", br=", countBranch(), ", delay=", vdel.x(k)
131                 sref.sec {
132                     esyn[k] = new Exp2Syn(lx)
133                     sh.color(2)
134                 }
135                 sh.point_mark(esyn[k],3)
136                 nc.append(new NetCon(stim, esyn[k], 0.0, vdel.x(k), w))
137                 found = 1
138                 break
139             }
140             l = l1
141         }
142     }
143 }
144 sh.flush()
145 doEvents()
146 //-----
147 // main loop
148
149 objref tv, vs
150 tv = new Vector()
151 vs = new Vector()
152 tv.record(&t)
153 vs.record(&soma.v(0.5))
154
155 finitialize(v_init)
156 fcurrent()
157 run(tstop)
158
159 objref g
160 g = new Graph()
161 g.size(0, tstop, -100, 50)
162 vs.plot(g, tv, 2, 1)
163 g.flush()

```

## 2.14 n14.hoc

```
1 // n14.py
2
3 create soma
4 soma insert m01
5
6 tstop = 40.0
7 v_init = -65.0
8
9 objref cvode
10 cvode = new CVode()
11 cvode.active(1)
12 cvode.atol(1.0e-5)
13
14 objref tv, zv
15 tv = new Vector()
16 zv = new Vector()
17 tv.record(&t)
18 zv.record(&soma.z_m01(0.5))
19
20 finitialize(v_init)
21 //h.fcurrent()
22 run(tstop)
23
24 objref g
25 g = new Graph()
26 g.size(0, tstop, -1, 1)
27 zv.plot(g, tv, 2, 1)
28 g.flush()
```

## 2.15 n15.hoc

```
1 // n15.hoc Markov model of Na channel
2
3 nstates = 9
4
5 tstop = 10.0
6 v_init = -100.0
7 dt = 0.05
8
9 create c
10 c insert mna
11
12 objref vc
13 c vc = new VClamp(0.5)
14 vc.dur[0] = 2.0
15 vc.dur[1] = 5.0
16 vc.dur[2] = 3.0
17 vc.amp[0] = v_init
18 vc.amp[1] = 0.0
19 vc.amp[2] = v_init
20
21 objref tv, zv
22 tv = new Vector()
23 zv = new List()
24 for i = 0, nstates-1 {
25     zv.append(new Vector())
26 }
27
28 tv.record(&t)
29 strdef s
30 for i = 0, nstates-1 {
31     sprintf(s, "zv.object(%1d).record(&c.z%1d_mna(0.5))", i, i)
32     execute(s)
33 }
34
35 finitialize(v_init)
36 fcurrent()
37 run(tstop)
38
39 objref g
40 g = new Graph()
41 g.size(0, tstop, 0, 1)
42 for i = 0, nstates-1 {
43     zv.object(i).plot(g, tv, i+2, 1)
44 }
45 g.flush()
```

## 2.16 n16.hoc

```
1 // n16.hoc  Markov model of Na channel
2 //          Voltage-dependent activation
3
4 nstates = 9
5
6 tstop = 20.0
7 dt = 0.01
8
9 v_init = -100.0
10 vstart = -100.0
11 vstep = 10.0
12 ntraces = 20
13 npts = int(tstop/dt)+1
14 eqna = 50.0
15
16 create c
17 c insert mna
18
19 objref vc
20 c vc = new VClamp(0.5)
21 vc.dur[0] = 5.0
22 vc.dur[1] = 10.0
23 vc.dur[2] = 5.0
24 vc.amp[0] = v_init
25 vc.amp[1] = v_init
26 vc.amp[2] = v_init
27
28 objref tv, gv, vv
29 tv = new Vector()
30 tv.record(&t)
31 gv = new Vector()
32 gv.record(&c.z3_mna(0.5))
33 vv = new Vector()
34 vv.record(&c.v(0.5))
35
36 objref traces, vtra
37 traces = new List()
38 vtra = new List()
39 for i = 0, ntraces-1 {
40     vc.amp[1] = vstart + vstep * i
41     finitialize(v_init)
42     fcurrent()
43     run(tstop)
44
45     traces.append(gv.c())
46     vtra.append(vv.c())
47 }
48
49 // open probability
50 objref g0, g1
51 g0 = new Graph()
52 g0.size(0, tstop, 0, 0.3)
53 for i = 0, ntraces-1 {
54     traces.o(i).plot(g0, tv, 2, 1)
55 }
56 g0.flush()
57
```

```
58 // current-voltage
59 objref cv
60 cv = new List()
61 g1 = new Graph()
62 g1.size(0, tstop, -15, 2)
63 for i = 0, ntraces-1 {
64     cv.append(vtra.o(i).sub(eqna).mul(traces.o(i)))
65     cv.object(i).plot(g1, tv, 2, 1)
66 }
67 g1.flush()
```

## 2.17 n17.hoc

```
1 // n15.hoc
2
3
4 load_file ("../models/cellModel.hoc")
5
6 ncells = 20
7 objref cell[ncells]
8
9 for i = 0, ncells-1 {
10     cell[i] = new BallStickCell( )
11 }
12
13 objref nclist, target, nc
14 nclist = new List()
15 for i = 0, ncells-1 {
16     target = cell[(i+1) % ncells] // next cell
17     nc = cell[i].connect2target(target.synlist.o(0))
18     nclist.append(nc)
19     nc.delay = 1
20     nc.weight[0] = 0.01
21 }
22 }
23
24 //external stimulus to syn of cell[0]
25 objref stim, ncstim, nil
26 stim = new NetStim()
27 stim.number = 1
28 stim.start = 0
29 ncstim = new NetCon(stim, cell[0].synlist.o(0))
30 ncstim.delay=0
31 ncstim.weight[0] = 0.01
32
33 objref tvec, idvec
34 tvec = new Vector( )
35 idvec = new Vector( )
36 for i = 0, ncells-1 {
37     nc = cell[i].connect2target(nil)
38     nc.record (tvec, idvec, i)
39 }
40
41 // Simulation control
42 tstop = 100
43 v_init = -65
44 finitialize(v_init)
45 run(tstop)
46
47 objref g
48 g = new Graph()
49 g.size(0, tstop, 0, ncells)
50 idvec.mark(g, tvec, "|", 4, 2, 2)
51 g.flush()
```



## 2.18 n17.hoc

```
1 // n15.hoc
2
3
4 load_file ("../models/cellModel.hoc")
5
6 ncells = 20
7 objref cell[ncells]
8
9 for i = 0, ncells-1 {
10     cell[i] = new BallStickCell( )
11 }
12
13 objref nclist, target, nc
14 nclist = new List()
15 for i = 0, ncells-1 {
16     target = cell[(i+1) % ncells] // next cell
17     nc = cell[i].connect2target(target.synlist.o(0))
18     nclist.append(nc)
19     nc.delay = 1
20     nc.weight[0] = 0.01
21 }
22 }
23
24 //external stimulus to syn of cell[0]
25 objref stim, ncstim, nil
26 stim = new NetStim()
27 stim.number = 1
28 stim.start = 0
29 ncstim = new NetCon(stim, cell[0].synlist.o(0))
30 ncstim.delay=0
31 ncstim.weight[0] = 0.01
32
33 objref tvec, idvec
34 tvec = new Vector( )
35 idvec = new Vector( )
36 for i = 0, ncells-1 {
37     nc = cell[i].connect2target(nil)
38     nc.record (tvec, idvec, i)
39 }
40
41 // Simulation control
42 tstop = 100
43 v_init = -65
44 finitialize(v_init)
45 run(tstop)
46
47 objref g
48 g = new Graph()
49 g.size(0, tstop, 0, ncells)
50 idvec.mark(g, tvec, "|", 4, 2, 2)
51 g.flush()
```

## 3 models, mod files

### 3.1 cellModel.hoc

```
1
2 // M.L. Hines1 and N.T. Carneva
3 // Translating network models to parallel hardware in NEURON
4 // J Neurosci Methods. 2008 Apr 30; 169(2): - 425455.
5
6 begintemplate BallStickCell
7     public is_art
8     public init, topol, basic_shape, subsets, geom, biophys, geom_nseg, \
9     biophys_inhomo
10    public synlist, x, y, z, position, connect2target
11    public soma, dend
12    public all
13
14    objref synlist
15    proc init ( ) {
16        topol ( )
17        subsets ( )
18        geom ( )
19        biophys ( )
20        geom_nseg ( )
21        synlist = new List ( )
22        synapses ( )
23        x = y = z = 0 // only change via position
24    }
25    create soma, dend
26    proc topol ( ) { local i
27        connect dend (0), soma (1)
28        basic_shape ( )
29    }
30    proc basic_shape ( ) {
31        soma {pt3dclear ( ) pt3dadd (0, 0, 0, 1) pt3dadd (15, 0, 0, 1)}
32        dend {pt3dclear ( ) pt3dadd (15, 0, 0, 1) pt3dadd (105, 0, 0, 1)}
33    }
34    objref all
35    proc subsets ( ) { local i
36        objref all
37        all = new SectionList ( )
38        soma all.append ( )
39        dend all.append ( )
40    }
41    proc geom ( ) {
42        forsec all { }
43        soma { L = diam = 12.6157 } // *area = 500
44        dend { L = 200 diam = 1 }
45    }
46
47    //external lambda_f
48    proc geom_nseg ( ) {
49        forsec all {
50            lf = 1e5*sqrt(diam/(4*PI*100*Ra*cm))
51            nseg = int ((L/(0.1*lf)+.9)/2)*2 + 1
52        }
53    }
54
```

```

55     proc biophys ( ) {
56         forsec all {
57             Ra = 100
58             cm = 1
59         }
60         soma {
61             insert hh
62             gnabar_hh = 0.12
63             gkbar_hh = 0.036
64             gl_hh = 0.0003
65             el_hh = -54.3
66         }
67         dend {
68             insert pas
69             g_pas = 0.001
70             e_pas = -65
71         }
72     }
73
74     proc biophys_inhomo ( ){}
75
76     proc position ( ) { local i
77         soma for i = 0, n3d( )-1 {
78             pt3dchange (i, $1-x+x3d (i), $2-y+y3d (i), $3-z+z3d (i), diam3d (i))
79         }
80         x = $1 y = $2 z = $3
81     }
82
83     obfunc connect2target ( ) { localobj nc // $o1 target point process,
84                                     // optional $o2 returned NetCon
85         soma nc = new NetCon (&v(0.5), $o1)
86         nc.threshold = 10
87         return nc
88     }
89
90     objref syn_
91     proc synapses ( ) {
92         /* E0 */
93         dend syn_ = new ExpSyn (0.8)
94         synlist.append (syn_)
95         syn_.tau = 2
96         /* I1 */
97         dend syn_ = new ExpSyn (0.1)
98         synlist.append (syn_)
99         syn_.tau = 5
100        syn_.e = -80
101    }
102
103    func is_art ( ) {
104        return 0
105    }
106 endtemplate BallStickCell

```

## 3.2 cellModel.py

```
1 # cellModel.py : Bann-and-Stick model
2
3 from neuron import h
4 import math
5
6 class BallStickCell(object):
7
8     def __init__(self):
9         self.x = self.y = self.z = 0
10        self.soma = h.Section(name='soma', cell=self)
11        self.dend = h.Section(name='dend', cell=self)
12        self.synlist = h.List()
13        self.topol()
14        self.subsets()
15        self.geom()
16        self.biophys()
17        self.geom_nseg()
18        self.synapses()
19
20    def topol(self):
21        self.dend.connect(self.soma(1))
22
23    def subsets(self):
24        self.all = h.SectionList()
25        self.all.append(sec=self.soma)
26        self.all.append(sec=self.dend)
27
28    def geom(self):
29        self.soma.L = self.soma.diam = 12.6157 # microns
30        self.dend.L = 200 # microns
31        self.dend.diam = 1 # microns
32
33    def geom_nseg(self):
34        for sec in self.all:
35            sec.nseg = int((sec.L/(0.1*h.lambda_f(100))+.9)/2)*2 + 1
36
37    def biophys(self):
38        for sec in self.all: # 'all' defined in build_subsets
39            sec.Ra = 100 # Axial resistance in Ohm * cm
40            sec.cm = 1 # Membrane capacitance in micro Farads / cm^2
41
42        # Insert active Hodgkin-Huxley current in the soma
43        self.soma.insert('hh')
44        self.soma.gnabar_hh = 0.12 # Sodium conductance in S/cm2
45        self.soma.gkbar_hh = 0.036 # Potassium conductance in S/cm2
46        self.soma.gl_hh = 0.0003 # Leak conductance in S/cm2
47        self.soma.el_hh = -54.3 # Reversal potential in mV
48        # Insert passive current in the dendrite
49        self.dend.insert('pas')
50        self.dend.g_pas = 0.001 # Passive conductance in S/cm2
51        self.dend.e_pas = -65 # Leak reversal potential mV
52
53    def connect2target(self, pp):
54        nc = h.NetCon(self.soma(0.5)._ref_v, pp, sec=self.soma)
55        nc.threshold = 10
56        return nc
57
```

```
58     def synapses (self):
59         self.synlist.append(h.ExpSyn(0.8,sec=self.dend))
60         self.synlist.object(0).tau = 2
61
62         self.synlist.append(h.ExpSyn(0.1,sec=self.dend))
63         self.synlist.object(1).tau = 5
64         self.synlist.object(1).e = -80.0
```

### 3.3 m01.mod

```
1 : m01.mod: a simple ODE  $y'' = -y$ 
2
3 NEURON {
4     SUFFIX m01
5     RANGE z
6 }
7
8 STATE {z z1}
9
10 INITIAL {
11     z = 0.0
12     z1 = 1.0
13 }
14
15 BREAKPOINT{
16     SOLVE zstates METHOD cnexp
17 }
18
19 DERIVATIVE zstates {
20     z' = z1
21     z1' = -z
22 }
```

### 3.4 exp2syn.mod

```
1 :   nrn-6.1src/nrnoc/exp2syn.mod
2
3 NEURON {
4     POINT_PROCESS Exp2Syn
5     RANGE tau1, tau2, e, i
6     NONSPECIFIC_CURRENT i
7     RANGE g
8     GLOBAL total
9 }
10 UNITS {
11     (nA) = (nanoamp)
12     (mV) = (millivolt)
13     (uS) = (microsiemens)
14 }
15 PARAMETER {
16     tau1= 0.1 (ms) <1e-9,1e9>
17     tau2 = 10 (ms) <1e-9,1e9>
18     e=0 (mV)
19 }
20 ASSIGNED {
21     v (mV)
22     i (nA)
23     g (uS)
24     factor
25     total (uS)
26 }
27 STATE {
28     A (uS)
29     B (uS)
30 }
31 INITIAL {
32     LOCAL tp
33     total = 0
34     if (tau1/tau2 > .9999) { : avoid tau1==tau2
35         tau1 = .9999*tau2
36     }
37     A=0
38     B=0
39     tp = (tau1*tau2)/(tau2 - tau1) * log(tau2/tau1)
40     factor = -exp(-tp/tau1) + exp(-tp/tau2)
41     factor = 1/factor
42 }
43
44 BREAKPOINT {
45     SOLVE state METHOD cnexp
46     g = B-A
47     i = g * (v - e)
48 }
49
50 DERIVATIVE state {
51     A' = -A/tau1
52     B' = -B/tau2
53 }
54 NET_RECEIVE(weight (uS)) {
55     A = A + weight*factor
56     B = B + weight*factor
57     total = total+weight
```





### 3.5 gsyn.mod

```
1 : gsyn.mod      The NEURON Book pp. 281-282
2
3 NEURON {
4     POINT_PROCESS GSyn
5     RANGE tau1, tau2, e, i
6     RANGE Gtau1, Gtau2, Ginc
7     NONSPECIFIC_CURRENT i
8     RANGE g
9 }
10 UNITS {
11     (nA)      = (nanoamp)
12     (mV)      = (millivolt)
13     (umho)    = (micromho)
14 }
15 PARAMETER {
16     tau1      = 1      (ms)
17     tau2      = 1.05  (ms)
18     Gtau1     = 20     (ms)
19     Gtau2     = 21     (ms)
20     Ginc      = 1
21     e         = 0      (mV)
22 }
23 ASSIGNED {
24     v      (mv)
25     i      (nA)
26     g      (umho)
27     factor
28     Gfactor
29 }
30 STATE {
31     A (umho)
32     B (umho)
33 }
34 INITIAL {
35     LOCAL tp
36     A = 0
37     B = 0
38     tp = (tau1*tau2)/(tau2-tau1) * log(tau2/tau1)
39     factor = -exp(-tp/tau1) + exp(-tp/tau2)
40     factor = 1/factor
41     tp = (Gtau1*Gtau2)/(Gtau2-Gtau1) * log(Gtau2/Gtau1)
42     Gfactor = -exp(-tp/Gtau1) + exp(-tp/Gtau2)
43     Gfactor = 1/Gfactor
44 }
45 BREAKPOINT {
46     SOLVE state METHOD cnexp
47     g = B - A
48     i = g * (v - e)
49 :     i = g * (v - e)
50 }
51 DERIVATIVE state {
52     A' = -A/tau1
53     B' = -B/tau2
54 }
55 NET_RECEIVE (weight (umho), w, G1, G2, t0 (ms)){
56     G1 = G1*exp(-(t-t0)/Gtau1)
57     G2 = G2*exp(-(t-t0)/Gtau2)
```

```
58     G1 = G1 + Ginc * Gfactor
59     G2 = G2 + Ginc * Gfactor
60     t0 = t
61     w = weight * (1 + G2 - G1)
62     A = A + w*factor
63     B = B + w*factor
64 }
```

### 3.6 NMDAR.mod

```

1 : NMDAR.mod    NMDA receptor channel
2 NEURON {
3     POINT_PROCESS nmdanet
4     RANGE R, g, mg
5     NONSPECIFIC_CURRENT i
6     GLOBAL Cdur, Alpha, Beta, Erev, Rinf, Rtau
7 }
8 UNITS {
9     (nA) = (nanoamp)
10    (mV) = (millivolt)
11    (umho) = (micromho)
12    (mM) = (milli/liter)
13 }
14 PARAMETER {
15     Cdur    = 1 (ms)      : transmitter duration (rising phase)
16     Alpha   = 0.35 (/ms) : forward (binding) rate
17     Beta    = 0.035 (/ms) : backward (unbinding) rate
18     Erev    = 0 (mV)      : reversal potential
19     mg      = 1 (mM)      : external magnesium concentration
20 }
21 ASSIGNED {
22     v      (mV)          : postsynaptic voltage
23     i      (nA)          : current = g*(v - Erev)
24     g      (umho)        : conductance
25     Rinf   (ms)          : steady state channels open
26     Rtau   (ms)          : time constant of channel binding
27     synon
28 }
29 STATE {Ron Roff}
30 INITIAL {
31     Rinf = Alpha / (Alpha + Beta)
32     Rtau = 1 / (Alpha + Beta)
33     synon = 0
34 }
35
36 BREAKPOINT {
37     SOLVE release METHOD cnexp
38     g = mgblock(v)*(Ron + Roff)*1(umho)
39     i = g*(v - Erev)
40 }
41
42 DERIVATIVE release {
43     Ron' = (synon*Rinf - Ron)/Rtau
44     Roff' = -Beta*Roff
45 }
46 FUNCTION mgblock(v(mV)) {
47     TABLE
48     DEPEND mg
49     FROM -140 TO 80 WITH 1000
50     mgblock = 1 / (1 + exp(0.062 (/mV) * -v) * (mg / 3.57 (mM)))
51 }
52 NET_RECEIVE(weight, on, nspike, r0, t0 (ms)) {
53     if (flag == 0) { : a spike, so turn on if not already in a Cdur pulse
54         nspike = nspike + 1
55         if (!on) {
56             r0 = r0*exp(-Beta*(t - t0))
57             t0 = t

```

```

58         on = 1
59         synon = synon + weight
60         Ron = Ron + r0
61         Roff = Roff - r0
62     }
63     net_send(Cdur, nspike)
64 }
65 if (flag == nspike) { : if this associated with last spike then turn off
66     r0 = weight*Rinf + (r0 - weight*Rinf)*exp(-(t - t0)/Rtau)
67     t0 = t
68     synon = synon - weight
69     Ron = Ron - r0
70     Roff = Roff + r0
71     on = 0
72 }
73 }

```

### 3.7 itvlfire.mod

```

1 : InvlFire.mod   The NEURON Book pp. 309- 310
2
3 NEURON {
4     ARTIFICIAL_CELL IntervalFire : name of the module
5     RANGE tau, m, invl           : accessible variables
6 }
7 PARAMETER {
8     tau = 5 (ms) <1e-9,1e9>
9     invl = 10 (ms) <1e-9,1e9>
10 }
11 ASSIGNED {
12     m
13     minf
14     t0 (ms)
15 }
16 INITIAL {
17     minf = 1/(1 - exp(-invl/tau)) : so natural spike interval is invl
18     m = 0
19     t0 = 0
20     net_send(firetime(), 1)       : set a self-event
21                                   : after time period of firetime()
22 }
23 NET_RECEIVE (w) {
24     m = M()
25     t0 = t
26     if (flag == 0) {               : *** event triggered by others ***
27         m = m + w
28         if (m > 1) {
29             m = 0
30             net_event(t)           : issue event
31         }
32         net_move(t+firetime())     : move the next event to t + firetime()
33     }else{                         : *** self-triggered event ***
34         net_event(t)               : issue event
35         m = 0
36         net_send(firetime(), 1)   : next self-event
37     }
38 }
39 FUNCTION firetime()(ms) {          : m < 1 and minf > 1
40     firetime = tau*log((minf-m)/(minf - 1))
41 }
42 FUNCTION M() {                    : to monitor m-value
43     M = minf + (m - minf)*exp(-(t - t0)/tau)
44 }

```