## Tutorial 1: Simple Neuron

We are going to start with PyHH basic classes.

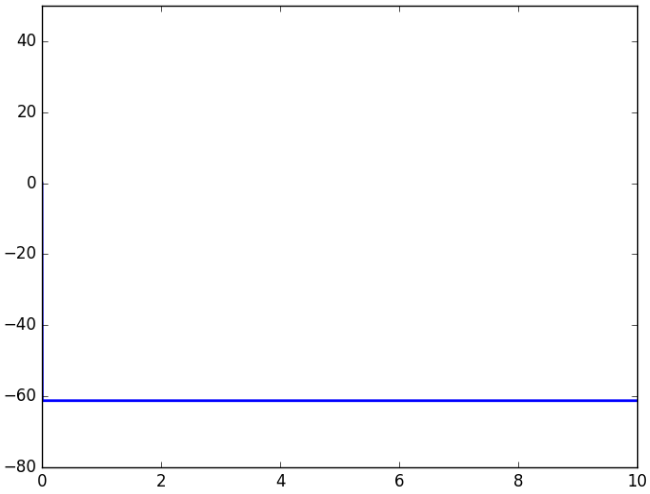
```
1  from pyhh import *
2  cpm = Compartment(diameter = 1.5, length = 100)
3  cpm.add_channels(NaC, KDR, gL)
```

Line 1 import PyHH module. Line 2 creates a compartment (cpm) from the Compartment class. It is a cylinder with diameter = 1.5, length = 100 µm. Line 3 adds predefined channels NaC, KDR and gL to the compartment. NaC, KDR and gL are sodium, potassium and leak channels, respectively.

You will learn how to define the conductances later. Now feed the compartment to an experiment:

```
4  xp = Experiment(cpm)
5  xp.run(t=10, dt=0.005)
6  cpm.plot()
```

Line 4 defines an experiment, which gathers information from the compartment for integration of equations. Line 5 makes a 10-ms simulation run with time step = 0.005 ms. Line 6 plots the membrane potential of the compartment. The time is from 0 to 10 ms. You will see something like

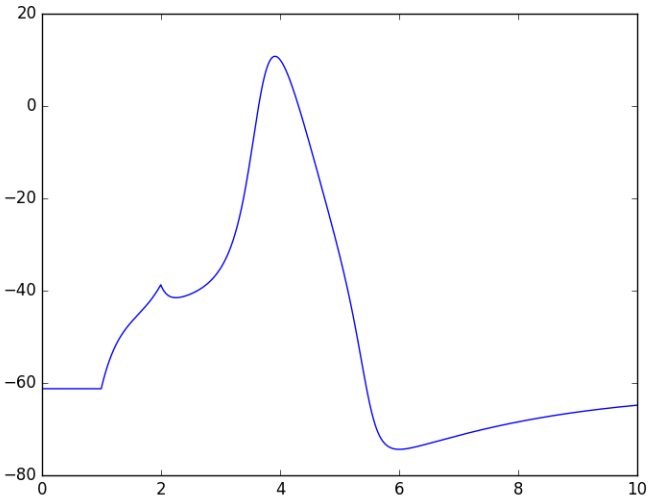



Now, we are going to define a clamper for current injection.

```
7  clp = IClamper()          # create a current clamper
8  clp.Waveform = Rect(delay=1,width=1,amplitude=0.55)
9  clp.connect(cpm)          # connect to the compartment
10 xp.run(10,0.005)          # run simulation again
```

```
11 cpm.plot()                # plot it
```

Line 7 creates a current clampr for current injection. Line 8 set the current waveform (Rect is rectangular impulse), then connect the clamper to the compartment (line 9), finally run the experiment (line 10) and plot the result (line 11). Now you see something different:




Please note the current amplitude is in the unit of pA/μm$^2$. This means that the amplitude is current density. The absolute current values can be easily calculated by

```
print clp.Waveform.Amplitude * cpm.Surface
```

```
261.12524992125003
```

Namely, about 0.261 nA.

Note: PyHH provides an alternative way of defining and connecting the clamper:

```
cpm.add_iclamper()
cpm.iClamper.Waveform = Rect(delay=1, width=1, amplitude=0.55)
```

The built-in plot is limited in functionality and if you want flexibility, make your own plot. The membrane potential is stored in **cpm.Vm**, and the time series in **xp.T** (also in **cpm.T**):

```
12 import pylab as plt       # import the tool
13 plt.figure()              # create a figure
14 plt.plot(xp.T, cpm.Vm)    # plot it
15 plt.ylim([-80,50])        # set y-axis range
16 plt.show()                # show the plot
```

For complete codes, see tutorial-1.py and run tutorial-1.py in a terminal, but first, quit Python by typing

```
quit()
```

And press enter. Then type:

```
python tutorial-1.py
```

### Note: Spherical Compartment

The cell body is better modeled as a sphere than as a cylinder. A sphere has no length. In PyHH you can define a sphere by

```
cpm = Compartment(diameter = 50, length = None)
```

PyHH will treat the compartment as a sphere. Try a spherical compartment in the above example.

### Two Compartment Model with tutorial 2

From this example, we are going to model a neuron with multiple compartments.

```
1  from pyhh import *
2  soma = Compartment(50, length=None)
3  dend = Compartment(1.5, length=100)
4  soma.connect(dend)
```

Note: you cannot do dend.connect(soma). Connection is directed in PyHH.

```
5  soma.add_channels(NaC, KDR, gL)
6  dend.add_channels(NaC, KDR, gL)
7  soma.add_iclamper()
8  soma.iClamper.Waveform = Rect(delay=2, width=0.5, amplitude=1.5)
```

Do experiment as before and plot the results:

```
9  xp = Experiment([soma,dend])
10 xp.run(10, 0.002)
```

You can plot the results like:

```
soma.plot()
dend.plot()
```

but PyHH provides a quick way:

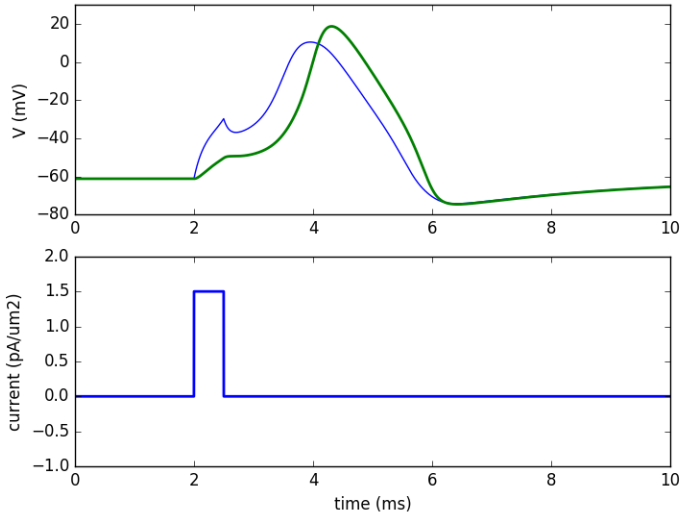
```
11 xp.plot()
```

which plots potential from both compartments. It is better to make your own plot, since you have more control over the look of the plot:

```
12 import pylab as plt
13 plt.figure()
14 plt.subplot(2,1,1)
```

```
15 plt.plot(xp.T, soma.Vm)
16 plt.plot(xp.T, dend.Vm, linewidth=2.0)
17 plt.ylim([-80,30])
18 plt.ylabel('V (mV)')
19 plt.subplot(2,1,2)
20 plt.plot(xp.T, soma.iClamper.Command, linewidth=2.0)
21 plt.ylim([-1,2])
22 plt.xlabel('time (ms)')
23 plt.ylabel('current (pA/um2)')
24 plt.show()
```

With your plot, you will see something like:




If you want to continue the run, just type:

```
xp.run(10, 0.005)
xp.plot()
```

You will see that the run start from 10 to 20 ms, and the starting potential is the continuation of the previous curve. If you want to change the time frame, you can do

```
xp.Clock = 0
```

then

```
xp.run(10,0.005)
xp.plot()
```

you will find the time start from zero again. Check tutorial-2.py for the complete codes.

**Test More Compartments: tutorial-3**

We are going to try 50 compartments. This will test how fast PyHH is. By using Python list data type and for loops, it is easy to define these compartments. A Python list is a sequence of items separated by commas, and all the items are between square brackets. Python lists will appear in the example scripts repeatedly. Check https://docs.python.org/3/tutorial/introduction.html#lists for more information. A python list can be created like

```
L = []
```

This creates an empty list called L. You can add items to the list like

```
L.append(1)
L.append(2)
L.append(3)
```

now,

```
L = [1,2,3]
```

You can access the items by the index like

```
P1[0], P1[1], P1[2]
```

Python has some function for building special list. The most important one is range. range(5) will create a list of [0, 1, 2, 3, 4]. When combined with the for loops, lists are very powerful. In tutorial-3.py, we are building a list of 50 compartment. As usual, we import necessary modules

```
1  from pyhh import *
2  N = 50   # number of compartment to be created
3  D = 1.5  # diameter
4  L = 50   # length
5  P = []   # define a list
6  for i in range(N):     # loops 50 times
7    a = Compartment(D, L) # create a compartment
8    P.append(a)           # add it to P
```

Lines 6, 7 and 8 can merge into two lines:

```
for i in range(N):     # loops 50 times
  P.append(Compartment(D,L))           # add it to P
```

If you know list comprehension, lines 5, 6, 7 and 8 can be replaced by just one line:

```
P = [ Compartment(D, L) for i in range(N) ]
```

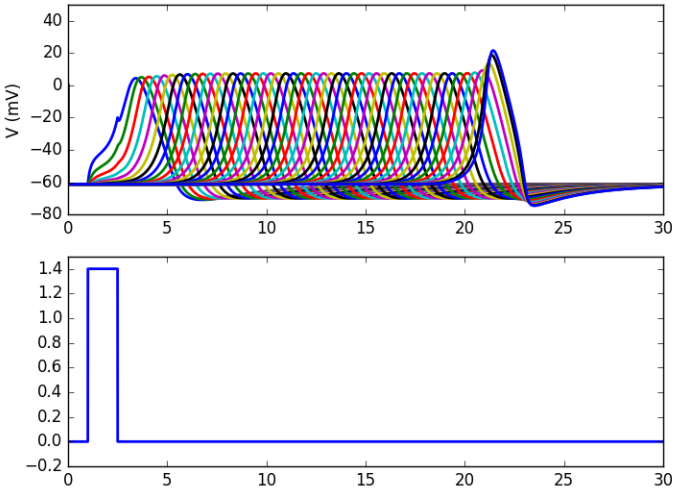
Now connect compartments and add channels using for loops:

```
9   for i in range(N-1):
10    P[i].connect(P[i+1])
11  for cpm in P:
12    cpm.add_channels(NaC, KDR, gL)
```

We add a current clamper to the first compartment:

```
13  P[0].add_iclamper()
14  P[0].iClamper.Waveform = Rect(delay=1.5, width=1., amplitude=1.65)
```

Done! Now you can define an experiment as before. Check the codes of tutorial-3.py for details. You will get something like:




On my Windows system, integration took 14.24 sec, on Ubuntu, it took 10.86 sec. Do you know why the last compartment has the biggest action potential?

## Voltage Clamp: tutorial-4

We are going to build a single-compartment neuron to see how the voltage clamp works. First, we import pyhh and pylab as usual, and prepare our cell:

```
1   from pyhh import *
2   soma = Compartment(diameter = 50, length=None)
3   soma.add_channels(NaC, KDR, gL)
```

Now we are going to define a voltage clamper from VClamper:

```
4   clp = VClamper()
5   clp.Waveform = Rect(delay=1, width=5, amplitude=40)
```

```
6  clp.connect(soma)
```

The default baseline is -60 mV. If you don't like this baseline value, for example, you can set a new one by clp.set_baseline(-70). Now we just use the default value. Alternatively, you can just add the clamper like:

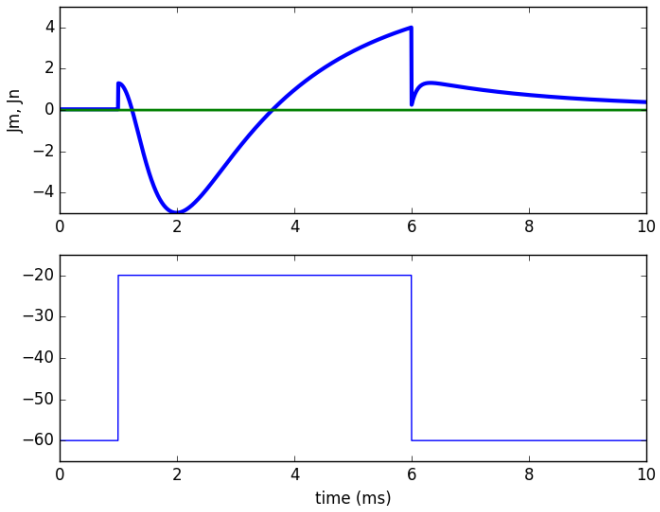
```
soma.add_vclamper()
soma.vClamper.Waveform = Rect(delay=1, width=5, amplitude=40)
```

It's time to run an experiment and see what happens:

```
7  xp = Experiment(soma)
8  xp.run(t=10,dt=0.005)
9  clp.plot() # default
```

In PyHH, a voltage clamper stores two variables, the transmembrane current (Jm), the cytoplasmic current (Jn). Both are normalized to the compartment surface area. So line 9 will plot both traces. Line 11 will only plots Jm. If you want to see the sum of Jm and Jn, set show_Jp flag as 1 as in line 12. Please not we ignore the capacitive current (Jc). During real patch clamp experiments, we want to measure the transmembrane current (Jm), and we always suppose that Jp = Jm. Actually even for ideal voltage clampers and ideal pipets, Jp is not equal to Jm. Jp is composed of at least two components, the transmembrane capacity current (Jc) and the transmembrane ionic current (Jm). If space clamp is not good, the cytosolic current (Jn) also contributes to Jp. If  Jn is small, we can expect Jp ≈ Jm. In real experiment, we normally see Jp, not the components of Jp. See tutorial-4.py for details.

You will see something like:



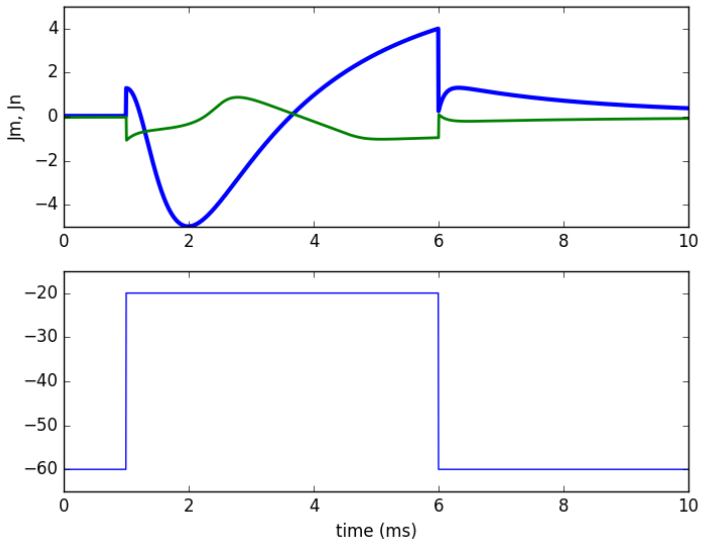
In next example, we will see recording from a multi-compartment neuron will be more complicated due to space clamp problem.

## Tutorial-5. Space Clamp

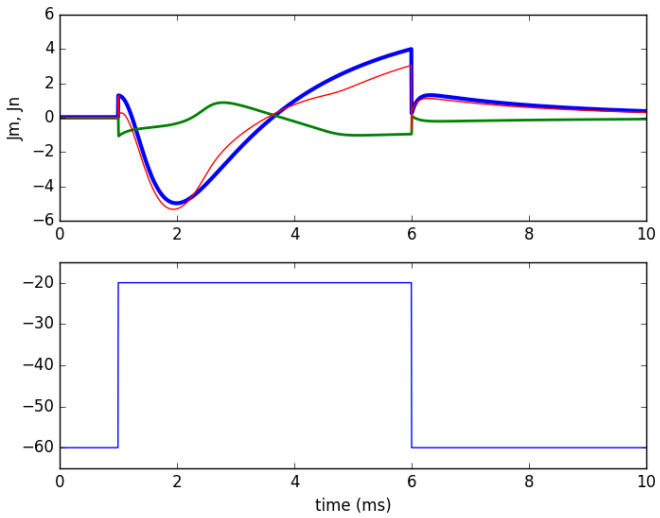
In this example, we are going to learn something about space clamp, which has been ignored by many patch clampers. The example codes are very similar to the previous one, but we use two compartments.

```
1  from pyhh import *

2  soma = Compartment(diameter = 50, length=None)
3  soma.add_channels(NaC, KDR, gL)
4  dend = Compartment(diameter = 1.5, length = 100)
5  dend.add_channels(NaC, KDR, gL)
6  soma.connect(dend)

7  soma.add_vclamper()
8  soma.vClamper.Waveform = Rect(delay=1, width=5, amplitude=40)

9  xp = Experiment([soma,dend])
10 xp.run(t=10,dt=0.005)

11 soma.vClamper.plot()
12 soma.vClamper.plot(show_Jm=1,show_Jn=1,show_Jp=1)
13 soma.vClamper.plot(show_Jm=1,show_Jn=0) # show Jm only
```

Now you see things are quite different. Line 11 produces:



The blue line is the transmembrane current (Jm), the green one is the current between the two compartments. Line 12 produces:



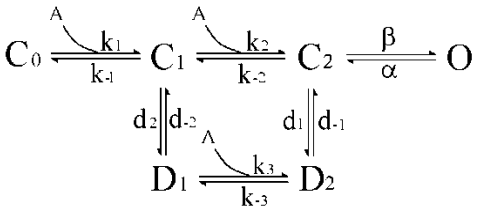

The red curve is the sum of the blue one and the green one. In real experiments, our pipet tips and amplifiers do not distinguish the current components. In other words, we see Jp more than Jm, but we normally treat them as the same thing, but they are not.

What happens if we also clamp the potential of the other compartment (run tutorial-5a.py):

```
dend.add_vclamper()
dend.vClamper.Waveform = Rect(delay=1, width=5, amplitude=40)
```

## Tutorial-6: the Ligand-Gated Ion Channel

In this example, I will use AMPA receptors to illustrate the simulation of LGICs based on Markov transition schemes. There are many transition schemes available, currently I use the scheme from Krampfl et al., 2002  in Eur. J. Neurosci. 15:51-62.




In this scheme, C represents closed states, O represents open states, and D represents deactivation states. k, α, β and d are rate constants.

Before defining the AMPA receptor, we have to define glutamate, the ligand of the AMPAR (you will know why soon):

```
Glu = Ligand()
```

I use Python dictionaries to represent the transition graph:

```
>>> ampar_transit = {
'C0': {'C1': 4.0},
'C1': {'C0': 2.0,   'D1': 0.15, 'C2': 2.0},
'C2': {'C1': 4.0,   'D2': 0.70, 'O': 20.0},
'D1': {'C1': 0.015, 'D2': 2.0},
'D2': {'C2': 0.002, 'D1': 0.875},
'O':  {'C2': 8.0}
}
```

You will understand this dictionary if you compare it with the transition scheme. In addition, we have to tell where the agonist (ligand) bind, and I also use Python dictionaries for this purpose:

```
>>> ampar_binding = {Glu:
                     {'C0': 'C1',
                      'C1': 'C2',
                      'D1': 'D2'
                     }
                    }
```

Now we can define the AMPA receptor:

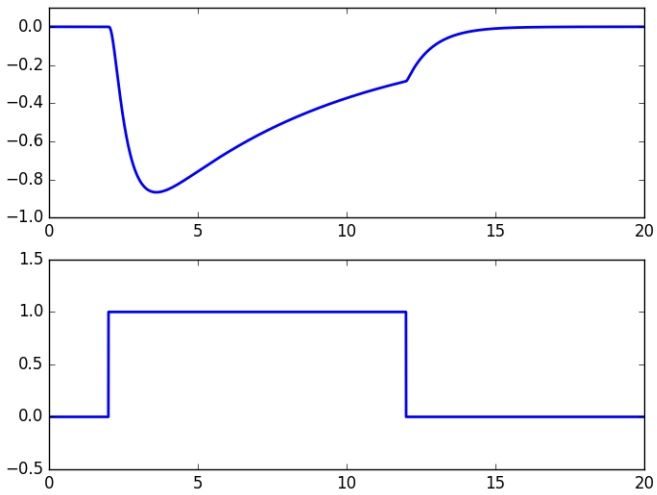
```
AMPAR = LGIC(ampar_transit, ampar_binding, gMax = 0.05, ER = 0)
```

The complete codes are as follows:

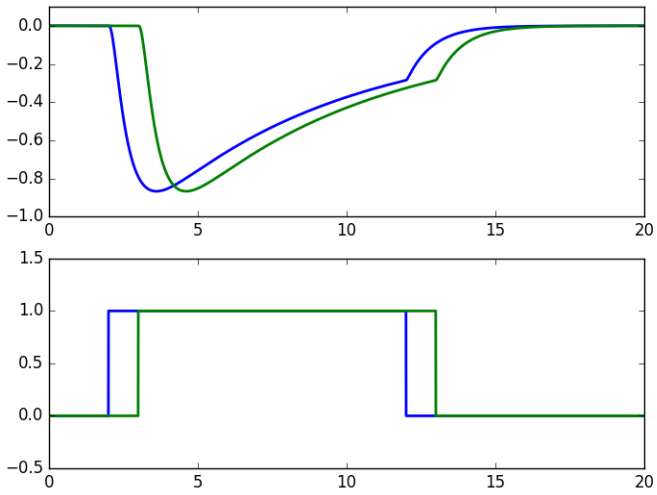
```
>>> from Pyhh import *
>>> import pylab as plt

>>> Glu = Ligand()
>>> ampar_transit = {
'C0': {'C1': 4.0},
'C1': {'C0': 2.0,   'D1': 0.15, 'C2': 2.0},
'C2': {'C1': 4.0,   'D2': 0.70, 'O': 20.0},
'D1': {'C1': 0.015, 'D2': 2.0},
'D2': {'C2': 0.002, 'D1': 0.875},
'O':  {'C2': 8.0}
}

>>> ampar_binding = {Glu:
{'C0': 'C1',
'C1': 'C2',
'D1': 'D2'
}
}
>>> AMPAR = LGIC(ampar_transit, ampar_binding, gMax = 0.05, ER = 0)
>>> cpm = Compartment(diameter = 1.5, length = 100)
>>> ampar = cpm.add_channels(AMPAR)
>>> cpm.add_channels(gL)
>>> vclp = VClamper()
>>> vclp.Waveform = Rect(delay=0, width=150, amplitude=0)
>>> vclp.connect(cpm)
>>> deliver = CClamper(ampar.Ligand)
```

```
>>> deliver.Waveform = Rect(delay=2, width=10, amplitude=1)
>>> deliver.connect(cpm)
>>> xp = Experiment(cpm)
>>> xp.run(20,dt=0.005)

>>> plt.figure()
>>> plt.subplot(2,1,1)
>>> plt.plot(xp.T, vclp.Jm, linewidth=2.0)
>>> plt.ylim([-1,0.1])
>>> plt.subplot(2,1,2)
>>> plt.plot(xp.T, deliver.Command, linewidth=2.0)
>>> plt.ylim([-0.5,1.5])
>>> plt.show()
```

You will see something like:




The file tutorial-7.py provides another example for AMPAR modeling with two compartments. Run the script and you will see something like:




Next, I will go back to the voltage-gated ion channels and explain how NaC, KDR and gL are defined.