NEURON 软件教程

Songting Li **August 2014**

我们将在本文中详细介绍如何使用 NEURON 软件模拟神经元和神经网络. 我们将通过从模拟 HH 点神经元开始, 到模拟由多个具有空间树突结构神经元构成的神经网络, 来介绍关于 NEURON 软件的基本语法, 函数, 以及数据的存储等内容. 相信读者在学习完本部分内容之后, 就可以开始写自己的 NEURON project 以及愉快地玩耍别人的代码了.

本文将分五部分介绍如何使用 NEURON 软件. 在第一部分中, 我们将介绍如何模拟点神经元模型; 在第二部分中, 我们将介绍如何模拟有空间结构的神经元模型; 在第三部分中, 我们将介绍如何使用模板来模拟神经元网络; 在第四部分中, 我们将介绍如何模拟自定义的离子通道动力学; 在第五部分中, 我们将介绍 NEURON 中图形界面的使用方法.

注: 本文内容主要参考自 Andrew Gillies 和 David Sterratt 的 NEURON 教程, 其链接如下 http://www.anc.ed.ac.uk/school/neuron/.

第一部分: 点神经元模型

● 空间结构

在本章节中, 我们首先考虑用 NEURON 软件模拟只有神经元细胞体的点神经元. 虽然大多数神经元的细胞体形状较接近圆球型, 然而在 NEURON 软件中, 描述神经元几何结构时通常是将神经元离散化分割成许多小段不同长度和直径的圆柱体连接而成. 我们将这些圆柱体称之为 sections (在分析的时候我们称它们为 compartments). 因此我们通常用一段圆柱体来模拟点神经元的细胞体 (当然也可以将一个圆球分割成不同直径的圆柱体串联来模拟一个更真实的胞体). 在描述胞体性质之前, 我们首先要创建一个细胞体:

```
create soma
```

create 命令创建了一个名称为 soma 的新 section. 每个新创建的 section 都有五个默认的属性值，如 section 的长度为 L=100um，直径为 diam=500 um，细胞膜电容为 c=1 uF/cm2，细胞质电阻率为 Ra=35.4 ohm*cm，以及 section 中细分的 segment 个数为 nseg=1.

初学 NEURON 的时候一定要注意区分 segment 和 section 的差别. section 是在重构神经元几何机构时将神经元分割成的每一小段分枝，而 segment 是每一段 section 中进一步离散化的节点个数. NEURON 在进行数值计算时，每一段 section 都会被等间距分成 L/nseg 份，且在每一个 segment 的中点处满足电流守恒定律的差分方程. 因此 segment 划分的越多，数值精度越高. 设置 segment 变量的好处为，section 的名称和个数通常在数值模拟的最开始都已经给定，因此用户可以简单改变 segment 的值来提高数值计算精度而不需要更改 section 变量.

若假设神经元细胞体是均匀的，且电流跨膜电流在细胞体上也为均匀分布，则我们只需要细胞体 section 的 segment 默认为 1 即可. 但是通常神经元细胞体的长度，宽度，以及细胞质电阻率与默认值并不相符，因此我们需要对其进行更改. 虽然我们目前只有一个 section，但是一般的，当 section 数目较多时，每个 section 都有自己的名字，我们在更改 section 参数时告诉 NEURON 我们更改的是哪一个 section. 在 NEURON 中，通常有如下三种方法修改 section 中的参数

A. 在想要修改的 section 属性前声明该 section 名字并用点符号(英文句号)连接 section 名字和属性名字.

```
soma.nseg = 1
soma.diam = 18.8
soma.L = 18.8
soma.Ra = 123.0
```

B. 在想要修改的 section 属性前声明该 section 名字并用空格连接 section 名字和属性名字.

```
soma nseg = 1
soma diam = 18.8
soma L = 18.8
soma Ra = 123.0
```

或者我们可以把属于同一个 section 的属性用大括号括起来

```
soma {
    nseg = 1
    diam = 18.8
    L = 18.8
    Ra = 123.0
}
```

特别需要注意的是，NEURON 中的左括号 { 不能另起一行，否则运行代码时会报错. 这和 C 语言习惯不同.

C. 使用 access 命令来声明当前默认的 section 然后再更改当前 section 的属性.

```
access soma
nseg=1
diam=18.8
L=18.8
Ra=123.0
```

这里 access 命令会告诉 NEURON，当前的默认 section 是 soma，若没有特殊的声明（i.e. 方法 A 和方法 B），任何直接使用的属性均属于默认 section，在这里即为 soma. 在代码中我们最好指定一个默认的 section，一般为细胞体（我们最感兴趣的 section），它在进行图形界面操作的时候需要用到. 但最好不要在代码中指定多于一个默认的 section，这样会降低代码的可读性.

● 离子通道

对于新创建的 section，NEURON 都会给它赋予默认的几何参数值，但是其细胞膜上的离子通道类型需要我们给定. NEURON 包含两种内联的离子通道类型，一种为 HH 型离子通道，另一种为渗漏型（passive leaky）离子通道. 声明 section 上离子通道种类需要用命令 insert

```
insert hh
insert pas
```

需指出，insert 命令可以结合空间结构中所描述的三种方法来关联相应的 section. 如

```
soma {
    nseg = 1
    diam = 18.8
    L = 18.8
    Ra = 123.0
    insert hh
}
```

当加入了新的离子通道之后，NEURON 会赋予离子通道属性的默认参数值. 对于 HH 类型离子通道，有如下参数（标准 HH 模型）

- gnabar_hh: 钠离子电导最大值 [Default value = 0.120 S/cm2]

- gkbar_hh: 钾离子电导最大值 [Default value = 0.036 S/cm2]
- gl_hh: 渗漏电导最大值 [Default value = 0.0003 S/cm2]
- ena: 钠离子通道反转电位 [Default value = 50 mV]
- ek: 钾离子电导反转电位 [Default value = -77 mV]
- el_hh: 渗漏电导反转电位 [Default value = -54.3 mV]

对于渗漏离子通道，有如下参数

- g_pas: 渗漏电导最大值 [Default value = 0.001 S/cm2]
- e_pas: 渗漏电导反转电位 [Default value = -70 mV]

我们知道，真实神经元上可能有上百种不同类型的离子通道，它们的动力学也各不相同，不能很好地用 HH 或者 leak channels 来描述. 我们将在第四部分来具体说明如何自定义离子通道动力学.

● 外部刺激

以上我们已经完成了对点神经元的代码描述. 接下来我们可以在神经元上加入电流刺激. NEURON 中对电流刺激的描述与离子通道的描述有所不同，主要在于离子通道属于所谓的 mechanism，其单位是单位面积上的数值大小. 而电流刺激属于所谓的 point process，其单位是绝对的数值大小 (默认的电流单位为 nA 而不是 nA/cm2). 另外，对于离子通道或其它 mechanism，在一个 section 中只可以存在一个，而对于电流刺激或其它 point processes，在一个 segment 中都可以有很多个.

point process 在 NEURON 中被看成是一种对象(object)，在创建前需要先对其进行声明

```
objectvar stim
```

上述命令声明了一个叫做 stim 的对象，现在我们需要通过 new 命令给 stim 进行赋值. 对于电流刺激以及一般的 point process，需要在赋值时指定刺激的位置，比如在我们的模型中刺激给在 soma 的中心

```
soma stim = new IClamp(0.5)
```

对于给定的 section，这里是 soma，它上面的空间位置是用[0,1]的局部坐标来描述的. 0.5 表示在 soma 的正中心位置.

对于一般的 0<x<1,

```
secX stim = new IClamp(x)
```

若 SecX 有多个 segments，且 x 不是某个 segment 的中点，那么刺激的位置将会默认给在 x 所在的 segment 的中点处. 这是因为 NEURON 在进行数值求解模型时，采用的是联立求解每个 segment 中点处的电流守恒离散差分方程.

在 NEURON 中，有几种内联的刺激函数可供选择，包括 IClamp (电流钳)，VClamp (电压钳)，以及 ExpSyn (指数函数模拟突触输入)等. 关于它们的说明可参见 Programmer's Reference. 我们也可以用 NMODL 语言自定义需要的刺激函数. 和离子通道一样，每个刺激函数也具有默认的属性参数. 对于电流钳，我们需要设置一下属性

IClamp:

- del: 开始模拟后过多少时间给刺激 (单位 ms)
- dur: 刺激的持续时间 (单位 ms)
- amp: 刺激的幅度 (单位 nA)

代码为

```
stim.del = 100
stim.dur = 100
stim.amp = 0.1
```

于是我们在模拟开始后的 100ms 之后给细胞体注入了大小为 0.1nA 的持续时间为 100ms 的电流刺激. 注意到对于电流钳以及其它点过程，不同于 section. 对于多个 section，有默认的 section，因此对于没有指定 section 的属性变量均指向默认 section. 而对于多个点过程，NEURON 中无法定义默认点过程. 因此点过程参数

需要指定是属于那一个点过程的，方法为点过程名称和参数之间用英语句号连接.

- **运行程序**

设置好了神经元模型和刺激输入之后，我们需要给 NEURON 指令让它开始运行我们的模型. 正如之前提到，在 NEURON 中我们不需要去处理模型背后的数值计算细节，因为 NEURON 为我们提供了一个标准函数库，使我们可以很容易的开始运行我们的模型以及其它一些功能.

若想要调用标准库中的函数，首先我们需要在呆码的最开始处加载函数库

```
load_file("nrngui.hoc")
```

然后我们在代码中模型和刺激描述之后设置数值计算的时间步长 dt (默认为 0.025ms)，终止时间 tstop (默认为 5ms) 并调用 run() 函数

```
dt=0.001
tstop=300
run()
```

就可以完成模型的数值计算了. 正如在《NEURON 软件简介》, NEURON 中默认的数值格式为一阶精度的向后 Euler 格式.

- **数据存储**

我们可以用 print 命令将细胞体处的电压值打印在交互界面上 (注意这里的电压输出是模型运行结束时候的电压值)

```
print "soma_v=", soma.v
```

print 命令可以打印多个字符串或者变量，中间用逗号分开，字符串前后需要加上双引号.

另一方面，我们可以直接将数据写入文件中再导入到我们熟悉的 matlab 中进行

处理. 我们首先需要将感兴趣的数据存在数组中, 然后再将该数组写入到文件. 在 NEURON 中这一过程可以通过 Vector 和 File 对象完成, 我们下面对它进行详细说明. 比如我们想记录下电压随着时间变化的数据, 那么我们首先需要声明和创建两个 Vector 对象来分别记录时间点和相应的电压值数据

```
objref rect, recv
rect = new Vector()
recv = new Vector()
```

随后我们可以利用 Vector 对象中的 record 函数告诉 NEURON 我们想要记录的变量是什么

```
rect.record(&t)
recv.record(&soma.v(0.5))
```

这里的&符号是必须的, 且括号中的参数为我们想要记录的值, 即每个时间节点和每个时间点胞体中点处的电压值. 虽然在我们的例子中只有一个 segment, 胞体任何一点的电压值都等于中点处的值, 但我们仍需指定特定的位置记录数据. 非常重要的是, 这段代码要放在 run()函数之前, 这样在 run 的过程中数据会不断被记录下来.

在 run() 函数结束了之后, 我们可以将数据写入到文件中. 这时我们需要创建一个新的文件用来保存数据 (推荐保存成 dat 格式)

```
objref savdata
savdata = new File()
savdata.wopen("partA.dat")
```

通常我们会用 File 对象的 printf 函数在文件开头写入一些数据的相关信息, 如变量名称和数据的大小等

```
savdata.printf("t soma.v(0.5)\n")
savdata.printf("%d\n",rect.size())
```

这里 printf 的使用方法和 C 语言中的语法基本相同 (见 Programmer's Reference). 同理, %d 表示整数变量, \n 表示换行. 而 size 函数从属于 Vector 对象, 其返回值为 vector 的大小. 随后我们可以用 for 循环将数据写入到文件中

```
for i=0,rect.size()-1 {
    savdata.printf("%g %g\n", rect.x(i), recv.x(i))
}
```

关于 for 循环的用法如下

for *var* = *start*, *end* {

command

}

其中 var 是循环的变量名,start 和 end 是变量的上下界, command 是每次循环中你想要干的事儿. 另外, 在我们的例子中, 每次 for 循环里面 Vector 对象中的 x 函数返回它的元素值, 同样第一个元素的指标从 0 开始. 最后, 在程序结尾用如下命令结束我们的文件写入,

```
savdata.close()
```

以上是用 Vector 格式存储数据. 此外 NEURON 还提供给我们一种 Matrix 对象来高效存储数据, 它可以避免使用 for 循环. 关于 Matrix 的用法具体可参见 Programmer's Reference. 实现的代码如下

```
objref mat
mat = new Matrix()
mat.resize(recv.size(),2)
mat.setcol(0, rect)
mat.setcol(1, recv)
mat.fprint(savdata, " %g /t")
savdata.close()
```

这样我们先告诉NEURON我们需要一个行数为recv.size()列数为2的矩阵，然后将时间点写入矩阵第一列，电压值写入矩阵第二列．最后将矩阵写入 savdata 文件中．打开保存的文件时，第一行为这个 matrix 的大小，在数据处理时要注意．

这样我们就完成了关于 HH 点神经元模型的代码．读者可以尝试将以上代码写到PSPad中并保存成partA.hoc文件，然后右键单击用NEURON打开，最后用matlab查看结果．

第二部分：空间神经元模型

在第一部分中，我们实现了点神经元模型的模拟．然而，真实的神经元具有树突，胞体和轴突三部分组成，具有复杂的空间几何结构．因此，在本章节中，我们将介绍如何模拟具有空间结构的神经元．

● 创建 section

回顾第一部分中我们提到，NEURON 中的神经元模型都是离散成许多段圆柱体拼接而成．因此我们只需要创建多个圆柱体 section 来表示神经元的树突和轴突结构即可，

```
create soma, dend[2]
```

通过上述命令我们分别创建了一个叫做somasection，以及一个叫做dend的数组，一个表示树突，一个表示轴突 (虽然是叫做 dend 的数组中的元素，但名字并不重要，你可以起任何的名字，只要你喜欢)．理论上我们也可以创建两个叫做dend1和dend2的section，但是真实的神经元有很多分叉结构，且长度和直径各处都不太相同，因此需要创建几十甚至上百的的section，此时数组结构就会显得十分方便．NEURON 中的数组结构和 C 语言中数组相似，数组的第一个元素指标是从0开始而并非1开始．因此第一个元素为dend[0]，第二个元素为dend[1]．这里数组的参数 2 也可以是一个事先定义好的变量如下

```
ndend = 2
create soma, dend[ndend]
```

接下来我们就要重复第一部分中的步骤给每个 section 的几何属性赋值如下(这里用的是方法 B) 并修改离子通道的参数使得更接近一个真实的神经元 (参数值仅供参考, 这里重点在于说明语法)

```
soma {
    nseg = 1
    diam = 18.8
    L = 18.8
    Ra = 123.0
    insert hh
    gnabar_hh=0.25
    gl_hh = .0001666
    el_hh = -60.0
}
dend[0] {
    nseg = 5
    diam = 3.18
    L = 701.9
    Ra = 123
    insert pas
    g_pas = .0001667
    e_pas = -60.0
}
```

```
dend[1] {
    nseg = 5
    diam = 2.0
    L = 549.1
    Ra = 123
    insert pas
    g_pas = .0001667
    e_pas = -60.0
}
```

在我们的模型中，由于树突和轴突的长度很长，但是我们分别只定义了一个 section 去描述它们，似乎过于简化. 但是正如第一部分提到，我们可以通过设置 nseg 的数目来提高空间分辨率，描述膜电位沿着树突和轴突的空间传递. 从计算数学的角度来说，nseg 的个数就是对电缆方程 (Rall, cable theory) 在 [0, L] 上的离散化的点个数 (注意方程离散的点在每个 segment 的中点处而不是端点处). 显然 nseg 越大，模拟越精确，但所需要的计算时间也越长. 另外，一个好的习惯是，nseg 通常设置为奇数. 这样可以保证：当刺激位置给在 section 的中点 0.5 处时确实是给在中点，且对于 section 中点的电压等数据的读取也确实是来自中点.

- **连接 section**

接下来我们将连接树突轴突和胞体. 由于每个 section 都是一个圆柱体，有两个端点，因此，特别地，我们将指定一个 section 的哪一端和另一个 section 的哪一端相连. 这可以用命令 connect 实现 (图 1)

```
connect dend[0](0), soma(0)
connect dend[1](0), soma(1)
```

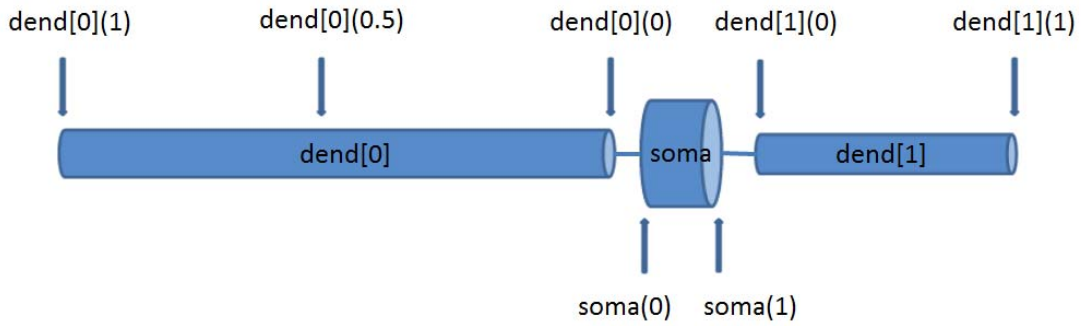

图 1

其中 dend[0](0)表示 dend[0]的起始端, dend[0](1)表示 dend[0]的末端. 我们之前在设置电流刺激位置的时候提到过, 一个 section 上的位置采用的是[0,1]上的局部坐标, 再如 dend[0](0.5)表示 dend[0]上距离起始端 50%处的点, 即中点. 在 NEURON 中, 似乎只允许两个 section 的两个端点相连接.

随后我们可以添加外部电流刺激或者其他刺激, 然后运行程序, 保存数据. 如同第一部分的操作, 我们在这里略去, 读者可自行完成这部分内容. 同时, 读者可以探索: (1) 电流刺激给在树突不同位置时在胞体处的电压波形如何变化, 即树突的 passive filtering effect (2) 膜电位如何在树突中轴向传递.

● 神经元模型数据库

以上我们只考虑一个树突和一个轴突的理想神经元模型. 然而对于上百个树突 section 来说, 用上述方法一个个手动输入太过繁琐而且易出错. 接下来我们将介绍如何建立更加真实的单个神经元模型. 值得一提的是, 现在的成像技术可以使得我们很好地重构一个神经元的几何结构, 并且科学家已经建立了有多数据库来共享这些数据, 供我们下载进行研究. 这里给出一个比较好的数据库 NeuroMorpho 如下

http://neuromorpho.org/

我们可以从 NeuroMorpho 中下载不同区域不同类型的神经元几何结构,然后导入到 NEURON 中. 举例来说, 我们可以下载一个大鼠小脑中的 Purkinje cell, 保存成 NEURON 格式, 然后将后缀名添上 hoc (如 Purkinje cell.hoc). 最后在我们的代码中将它用如下命令加载进来即可.

```
load_file("Purkinje cell.hoc")
```

因此我们可以得到一个真实几何形状的 purkinje 神经元，它的几何形状如图 2 所示．在第五部分我们将介绍如何在 NEURON 中查看神经元模型的几何形状．需要注意的是，这个数据库中只包含了神经元的几何结构，而离子通道等生物物理特性都是 NEURON 的默认值，需要自己重新设置．关于离子通道的设置通常是根据文献报道的离子通道随着距离的分布，且离子通道通常不是简单的 HH 类型，需要读者自定义它们的动力学．这一部分内容将在第四部分进行介绍．

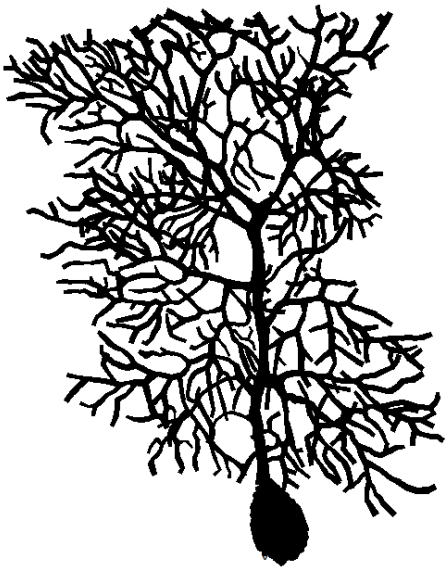

图 2

另外一个推荐的数据库就是 NEURON 自己的数据库．

http://senselab.med.yale.edu/modeldb/

在这个数据库中，读者可以发现，通常每个 project 有一个单独的 hoc 文件来存放神经元的空间几何数据．同样的，我们可以将它用 load 命令加载进来，结合到自己的 project 中．

如果查看从数据库导出的 Purkinje cell.hoc 文件，我们会发现它里面对树突和胞体的几何设置和我们之前介绍的并不相同．它用的是 pt3dadd 函数．这是 NEURON 的第二种定义几何结构的方法，将每个 section 都放置到三维空间中，这

对于实验重构神经元几何方法来说更加自然. 对于每个 section, 我们首先用 pt3dclear()命令清空关于该section的任何几何结构, 然后用pt3dadd()命令定义几何结构. pt3dadd(x,y,z,diam) 函数有四个参数, 分别为空间位置的 x,y,z 三个坐标, 以及在这个坐标点上这个 section 的直径。因此每个 section 的几何不再是一个圆柱体,而是几个圆台拼接而成的图形如图 3 所示.

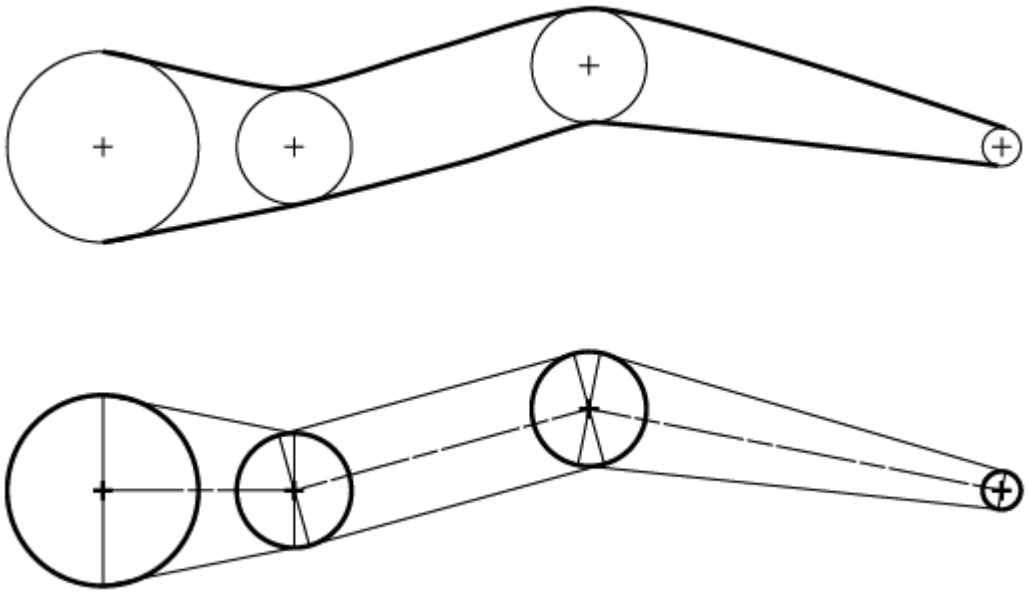

图 3

最后需要注意的是, 当一个神经元有多个 section 的时候, 在开始一段时间细胞膜电位可能并没有处在其静息状态. 这是因为在不同的 section 上有不同的离子通道类型(而且在有些复杂模型中离子通道可以空间分布不均匀), 因此尽管在每个 section 上局部细胞膜电位都是在其平衡态, 但是它们的值未必相等. 因此通常我们在模拟的时候在前面几毫秒内不给任何刺激, 使得神经元各个部分的膜电位都到达其平衡静息态(排除初值的影响), 然后再给予刺激进行模拟.

第三部分：神经元网络模型

在第二部分我们介绍了如何建立单个具有几何结构的神经元模型. 在本章节中, 我们将介绍如何建立神经元网络模型. 在我们的大脑中, 通常一个局部区域的神经元种类相对均一, 因此我们可以利用 NEURON 的模板 templates 功能来复制一个神经元模板得到多个完全相同的神经元 (当然随后我们也可以修改它们), 最

后我们通过 NetCon 函数将它们连接成一个神经网络.

● 模板创建

模板的创建语法如下

begintemplate *name*
code
endtemplate *name*

模板需要以 begintemplate 开始并以 endtemplate 结尾. 其中 name 是我们需要创建的模板名字, code 是我们的模板代码. 比如若我们想要用第二部分所创建的神经元作为模板, 则可以用如下代码

```
begintemplate cell
public soma, dend
create soma, dend[1]
proc init() {
    ndend = 2
    create soma, dend[ndend]
    soma {
      nseg = 1
      diam = 18.8
      L = 18.8
      Ra = 123.0
      insert hh
      gnabar_hh=0.25
      gl_hh = .0001666
```

```
    el_hh = -60.0
  }
  dend[0] {
    nseg = 5
    diam = 3.18
    L = 701.9
    Ra = 123
    insert pas
    g_pas = .0001666
    e_pas = -60.0
  }
  dend[1] {
    nseg = 5
    diam = 2.0
    L = 549.1
    Ra = 123
    insert pas
    g_pas = .0001666
    e_pas = -60.0
  }
  connect dend[0](0), soma(0)
  connect dend[1](0), soma(1)
```

```
}
endtemplate cell
```

函数 init()

在创建模板的过程中，我们定义了一个初始化函数 (procedure) 叫做 init(). 它会在每次模板被调用时运行，因此会自动创建出 sections 以及它们之间的连接关系.

函数定义给编程会带来很大的方便，且一个好的编程习惯是我们通常需要将代码分解成一个个函数，从而增加可读性，且更易修改. 用 proc 定义的函数没有返回值，但是可以有参数传递. 若用户需要定义一个可返回参数的函数，则需要用 func 定义之. 关于它们的详细介绍可参见 Programmer's Reference. 特别地，这些自定义函数的参数格式和 C 语言完全不同，需要特别加以注意 ($1, $2, …)! !

public 变量

在模板中我们声明了 soma 和 dend 为 public 变量，它告诉 NEURON 我们可以在模板定义之外使用它们. 比如说我们想要在用模板创建的神经元胞体上加入电流刺激，如果不对 soma 进行 public 声明，则我们无法调用之. 关于 public 的作用我们还会在第四部分中碰到.

对于一般的对象一样，我们需要首先对其进行声明

```
ncells = 2
objectvar cells[ncells]
```

然后用 for 循环定义 2 个神经元如下

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

- **网络连接**

目前我们用模板创建的两个神经元没有连接，相互独立．接下来我们将用 NetcCon 命令将它们通过突触连接．当任何突触前神经元胞体处膜电位超过我们设置的阈值时，突触后神经元的树突上会产生突触电流，当然我们可以设置任意的电流的形式(指数衰减，双指数差等等).

同上面所介绍的，我们利用模板定义两个神经元 cells[0]和 cells[1]，并将 cells[1]设置成突触后神经元．于是我们需要在其树突上定义突触．假设第一个神经元 cells[0]的轴突末端连接到 cells[1]的树突 dend[1]上位于相对位置 0.7 处的突触，由于突触也是对象，则我们先声明这个突触随后我们定义这两个突触类型为指数衰减型 (即接收输入时电导为 delta 函数跳跃)

```
objectvar syn
cells[1].dend[1] syn = new ExpSyn(0.7)
```

现在我们可以定义它们的连接

```
objectvar netcon
netcon = new NetCon(&cells[0].soma.v(0.5), syn, -20,
1, 0.5))
```

上述命令将 cells[0]和 cells[1]连接起来，只要 cells[0]的 soma 处电压值超过阈值 -20mV，则定义在 cells[1]上的突触 syn 将会产生一个指数衰减的突触电流，且从 cells[0]超过阈值的时间到突触电流产生的时间之间的延时为 1ms，电流强度为 0.5．一般的，NetCon 对象的语法为(参考 Programmer's Reference)

new NetCon(&*source_v*, *synapse*, *threshold*, *delay*, *weight*)

接下来请读者给 cells[0]从 100ms 到 200ms 一个电流刺激，记录 cells[0]和 cells[1]的细胞膜电位，并实现多个神经元多个突触的网络连接.

当一个神经元接收到多个突触输入时，我们可以将所有的突触输入对象存储在 List 变量中，组成一个突触数组．List 数组的好处是我们可以通过 for 循环方便地

修改所有的突触电流参数，如输入强度以及传递延时等．我们在这里不做介绍，关于 List 数据类型有需要的读者可参考 Programmer's Reference.

● 大尺度复杂神经元网络

在模拟大尺度复杂神经元网络的时候，NEURUON 为我们提供了一种更加简便的方法，即通过 Channel Builder 和 Cell Builder 来创建和保存模板神经元的 hoc 文件，然后通过 Network Builder 来创建神经元网络，保存成 hoc 文件．然后在 project 中调用这些 hoc 文件，进行所需要的数值实验．关于这一部分内容可以参考 The NEURON Book 的第十二章，以及 http://www.neuron.yale.edu/neuron/docs 中的 cell builder tutorials, channel builder tutorials, 和 network builder tutorials.

第四部分：离子通道动力学

在 NEURON 中，我们只能够选择有限的离子通道类型，即 HH 型和渗漏型离子通道．然而，在真实的神经元中有许多不同种类的离子通道，因此对它们的模拟十分重要．不同于 hoc 语言，NEURON 为我们提供了模拟离子通道动力学的语言，NEURON Model Description Language (NMODL)．我们在本章节中将从模拟低阈值的钙离子通道(T-type calcium channel)切入，介绍 NMODL 的语法格式.

● 钙离子通道动力学

关于低阈值钙离子通道的动力学描述最早在 Wang et.al. 1991 中给出.

$$\mathrm{I_T} = g_{\mathrm{T}} r^3 s(v - E_{Ca})$$

其中g_{T} 为钙离子通道电导最大值. r 为激活的门控变量, s 为失活的门控变量 (类比 HH 模型中钠离子的 m 和 h 门控变量)．E_{Ca} 为钙离子反转电位. r 和 s 的动力学如图 4 所示.

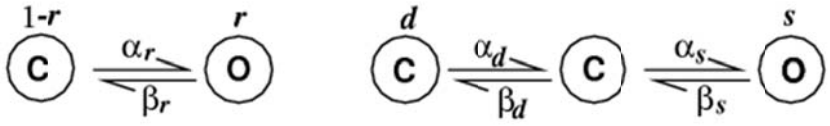

图 4

其中 r 有打开和关闭两个状态, s 有一个打开状态和两个关闭状态，我们把其中一

个关闭状态记为 d，它们满足动力学如下

$$\dot{r} = \alpha_r(1-r) - \beta_r r$$
$$\dot{s} = \alpha_s(1-d-s) - \beta_s s$$
$$\dot{d} = \alpha_d(1-d-s) - \beta_d d$$

接下来我们将在 NEURON 中用 NMODL 语言模拟该离子通道动力学. 简单来说，我们先用 PSPad 新建一个空白文件，然后编写描述离子通道动力学的代码，最后保存成.mod 格式，然后再在 hoc 文件中调用它既可. 接下来我们将详细说明中间最关键的一步，如何编写描述离子通道动力学的代码.

● NMODL 语言

NMODL 语言的一个明显特点为其代码由很多模块组成. 每个模块都以一个关键词开始，随后在一对大括号{}中填写该模块相应的代码. 以下我们来逐一解释每个模块. 这些模块在写其它离子通道时也都需要用到，具有一般性.

1. The TITLE block

```
TITLE Calcium T channel
```

Title block 是对这个 mod 文件的说明，尽管它不是必要的，可以省略.

2. The UNITS block

```
UNITS {
    (mV) = (millivolt)
    (mA) = (milliamp)
}
```

UNITS block 定义了在 mod 文件中需要用到的单位的缩写. 在上述例子中,它告诉 NEURON 我们将用 mV 表示毫伏, mA 表示毫安. 这个说明是必须的，因为 NEURON 进行模型的数值计算时需要统一单位.

3. The NEURON block

NEURON block 声明和 NEURON 接口时的信息. 在 hoc 文件调用 mod 文件时, 它告诉 hoc 文件哪些是公共变量可以查看或者修改的. 代码如下

```
NEURON {
    SUFFIX CaT
    USEION ca READ eca WRITE ica
    RANGE gmax
}
```

首先我们通过 SUFFIX 来命名这个 mod 文件所描述的离子通道. 这里我们把这个钙离子通道称为 CaT, 正如在默认的 HH 通道中, 其 SUFFIX 为 hh 一样. 因此在 hoc 文件中,我们可以直接用命令 insert CaT 且用 var_CaT 来查看它其中的变量(如门控变量的值). 这种 Var_Channel 的格式是 NEURON 的默认格式, 请读者习惯之.

USEION 描述改离子通道是通透哪一种离子的. 在 NEURON 中有三种离子可供选择, 分别为 Na, K, Ca 离子. 当然我们也可以自定义其它的离子,这里不做讨论.

READ 中列出了在计算离子电流时所需要的变量, 通常为反转电位 eca, 而 WRITE 中列出了在 NEURON 中调用 mod 文件时所需计算的变量, 通常为离子电流 ica.

RANGE 中列出了哪些变量在 hoc 文件调用时可以被查看或者被修改. 这里我们设置钙离子电导最大值 gmax 可以在 hoc 文件中被修改.

另外, NEURON block 中还可以有一个 GLOBAL 的声明, 它声明在调用 mod 文件中, 永远不会被改变的变量. 这里我们并不需要 GLOBAL 变量.

4. The PARAMETER block

PARAMETER 声明了可以在 hoc 文件中设置的参数, 这里为 gmax,我们需要给出它

的值以及单位(用括号括上).

```
PARAMETER {
    gmax  = 0.002 (mho/cm2)
}
```

随后我们就可以在 hoc 文件中修改这个参数, 如

```
soma gmax_CaT = 0.001
```

5. The ASSIGNED block

ASSIGN 列出了那些会在 NEURON 中被不断计算更新的变量, 这里有如下一些

```
ASSIGNED {
    v     (mV)
    eca   (mV)
    ica   (mA/cm2)
    ralpha (/ms)
    rbeta (/ms)
    salpha (/ms)
    sbeta (/ms)
    dalpha (/ms)
    dbeta (/ms)
}
```

这里要注意 eca 是钙离子的平衡电位,在一般的模拟中,我们都默认为常数. 但是在 NEURON 计算中, 它**可以**时时计算膜内外的离子浓度差, 因此 eca 可以随着计算时间而改变. 我们也可以在定义的 section 中修改 eca 的值.

```
soma {
insert CaT
```

```
eca = 126.1
}
```

6. The STATE block

状态变量即门控变量,

```
STATE {
    r s d
}
```

7. The heart of the mechanism

接下来我们将介绍求解关于 r,s,d 动力学方程的代码. 首先我们要创建一个 PROCEDURE 写出关于 r,s,d 关于 v 的函数,

```
PROCEDURE settables(v (mV)) {
    LOCAL  bd
    TABLE ralpha, rbeta, salpha, sbeta, dalpha, dbeta
        FROM -100 TO 100 WITH 200

    ralpha = 1.0/(1.7+exp(-(v+28.2)/13.5))
    rbeta  =
    exp(-(v+63.0)/7.8)/(exp(-(v+28.8)/13.1)+1.7)

    salpha = exp(-(v+160.3)/17.8)
    sbeta  = (sqrt(0.25+exp((v+83.5)/6.3))-0.5) *
                    (exp(-(v+160.3)/17.8))
```

```
    bd      = sqrt(0.25+exp((v+83.5)/6.3))
    dalpha =
    (1.0+exp((v+37.4)/30.0))/(240.0*(0.5+bd))
    dbeta   = (bd-0.5)*dalpha
}
```

注意到其中 TABLE 一行的代码. 由于这些函数都会随着时间改变而改变(因为电压是时间的函数), 单位了提高效率我们可以创建一个 library 可以用来线性差值(虽然精度略差..)得到对应某个 v 的函数值. 我们可以用 TABLE 命令实现它, 这里我们创建了一个 library, v 从-100 到 100,中间一共 200 个点.

8. The DERIVATIVE block

现在我们需要计算关于 r,s,d 随时间的演化.

```
DERIVATIVE states {
    settables(v)
    r' = ((ralpha*(1-r)) - (rbeta*r))
    d' = ((dbeta*(1-s-d)) - (dalpha*d))
    s' = ((salpha*(1-s-d)) - (sbeta*s))
}
```

第一行我们先调用 settables(v)函数得到相应的 alpha 和 beta 函数值, 然后计算 r,d,s 的微分方程.

9. The BREAKPOINT block

BREAKPOINT 设置了方程求解的信息

```
BREAKPOINT {
```

```
    SOLVE states METHOD cnexp
    ica = gmax*r*r*r*s*(v-eca)
}
```

如该方程是求解状态变量，方法为 cnexp. 我没有找到这个方法的具体介绍，只知道它是 NEURON 中的默认方法，具有二阶精度，非常适用于求解 hh 类型的自治系统. 最后给出了关于离子电流的计算公式.

10. The INITIAL block

INITIAL 给出了状态变量平衡态的值作为其初始态.

```
INITIAL {
    settables(v)
    r = ralpha/(ralpha+rbeta)
    s = (salpha*(dbeta+dalpha) - (salpha*dbeta))/
            ((salpha+sbeta)*(dalpha+dbeta) -
(salpha*dbeta))
    d = (dbeta*(salpha+sbeta) - (salpha*dbeta))/
            ((salpha+sbeta)*(dalpha+dbeta) -
(salpha*dbeta))
}
```

- **编译和调用**

最后我们只要将该文件保存成.mod 格式,然后在 hoc 文件中直接调用即可. 如直接在 section 中 insert CaT, 用法同默认的 hh 通道. 在使用之前,我们需要对 mod 文件进行编译,方法即打开 mknrndll 选择 mod 文件所在的文件夹即可.以上我们通

过一个例子讲解了如何使用 NMODL 编写自定义的离子通道动力学. 由于篇幅的过,不可能讲到每个细节,重在让读者了解它可以做什么,关于这一章节想要了解的更多的读者可以参见 The NEURON Book 的第九章.

第五部分：图形界面

图形界面能做的事情比较有限，比如我们不能够通过它实现复杂的代码，但是有些功能十分重要，且较适合初学者上手. 这里我们简单介绍它的一些基本功能. 因为作者不偏爱它的图形界面，因此详细的说明请参见 The NEURON Book.

当你点击 nrngui 或者在 hoc 文件中加载了 nrngui.hoc 时且运行完程序后，都会看到图 5 和图 6

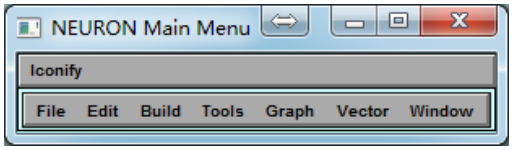

图 5

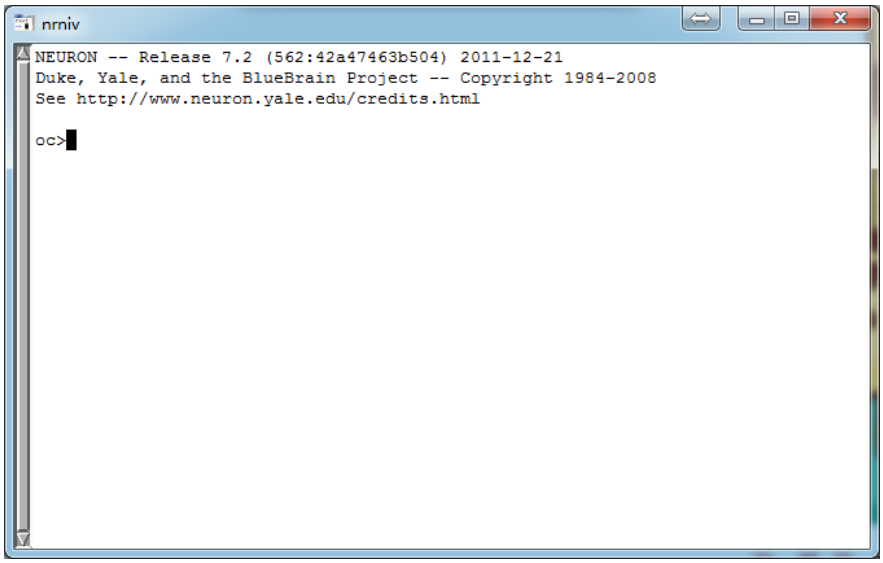

图 6

其中图 5 为菜单栏(Main Menu)，图 6 为命令窗口. 我们先来介绍菜单栏，再来介绍命令窗口.

- **菜单栏**

在 File 菜单中我们可以加载和保存文件.

在 Build 菜单中我们可以创建单个神经元，离子通道和神经元网络. 关于这一部分内容可以参考 The NEURON Book 的第十二章，以及 http://www.neuron.yale.edu/neuron/docs 中的 cell builder tutorials, channel builder tutorials, 和 network builder tutorials.

在 Tools 菜单下的 RunControl 中我们可以设置运行 code 的代码，在 Point Processes 中可以添加电流等点过程刺激，在 Distributed mechanisms 中可以添加离子通道等机制.

在 Graph 中,我们可以直接将数据进行画图. 用户可以选择画图的变量，如电压，电流，状态变量以及神经元的几何结构. 比如我们甚至我们可以进行 phase plane 分析. 比如我们加载(或者直接运行)了 partC.hoc 之后，我们可以打开 Graph 中的 Voltage axis，右上方的 v(.5)表示在 hoc 文件中我们 access 的默认 section 处电压值(在我们的代码中应是 cells[0].soma)，这也是我们需要在 hoc 文件中制定一个默认 section 的原因 (画图需要). 然后我们鼠标左键点击左上方的白色小框，

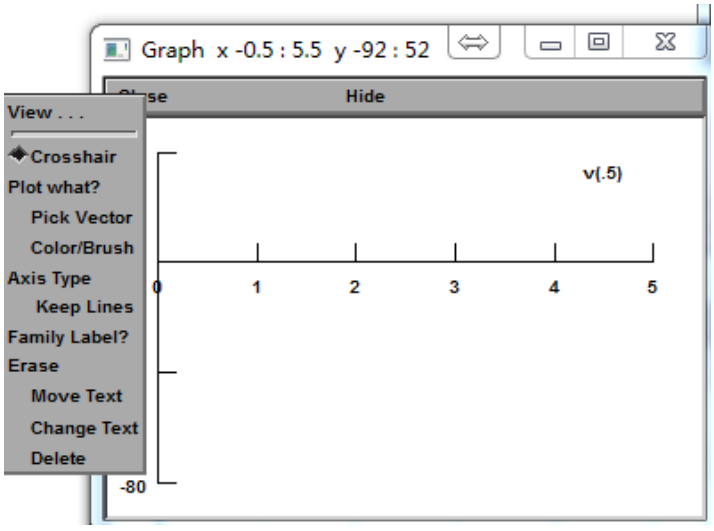

图 7

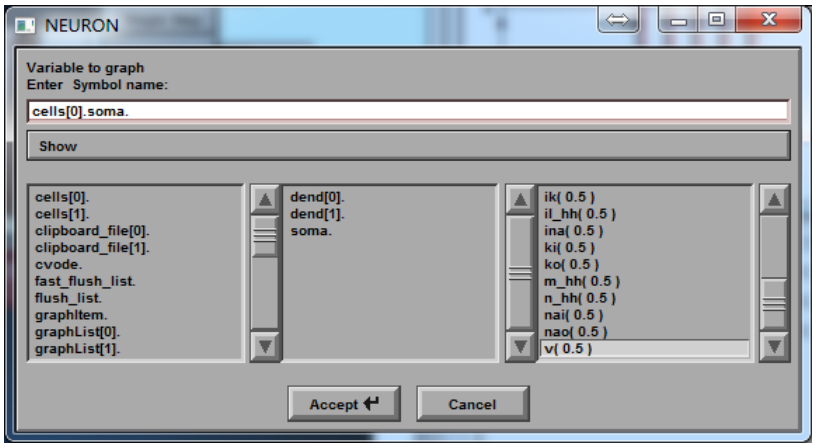

图 8

选中 plot what，然后在 show 中选择你要画的变量．比如我们想要同时画出两个神经元的细胞体膜电位，则我们选中 show—object refs—cells[0]---soma---v(0.5)，再点击 accept (可以不选，因为这是默认的变量，会自动画上)．然后重复以上过程选中 show—object refs—cells[1]---soma---v(0.5)，就可以了．最后我们要在 Tools 中的 Run Control 中点击 init & Run，就可以看到两个电压值的曲线了．但是两个曲线的默认颜色都是黑色的，因此为了区分二者，我们可以在左上角的小框中选中 Color/Brush，设置曲线的颜色．具体方法是先选中颜色，再选中曲线(或其右上角的名称)．

如果想要把电压数据保存下来，则我们先选中图 7 中的 pick vector，然后选中想要保存的曲线，最后在图 5 的 Vector 菜单下选中 Save to File 即可．

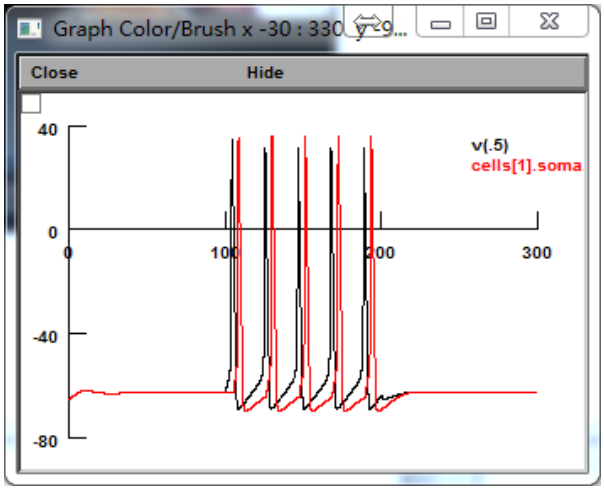

图 9

读者可以试着查看 cells[0]soma 上的钠离子钾离子电流和状态变量随时间的变化. 如果想对图像进行放大或缩小，图 7 中的菜单第一行 view 中可以选择放大或者缩小图片, view=plot 选项会给你最优的视角.

最后需要介绍的是 Shape Plot 会画出模型的三维空间结构. 在这里我们是两个 3-compartment neuron，所以几何如下图 10 所示. 同样的左上角白方格菜单可以让你有不同的 3-D 视角查看图像，且可以选择不同的 Shape Style (如显示神经元各处的直径还是显示单位直径的示意图).

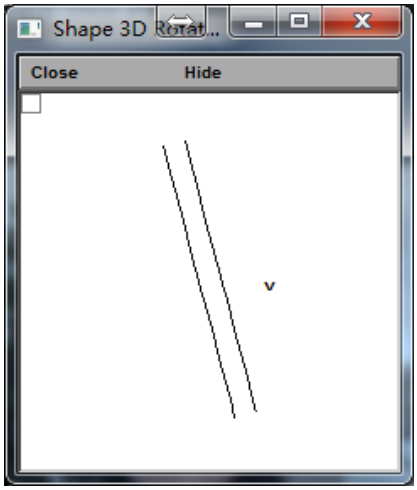

图 10

如果我们想把所有的图的结果保存下来，我们可以通过图 5 中 Window 菜单中的 Print & File Window Manager 保存下来.

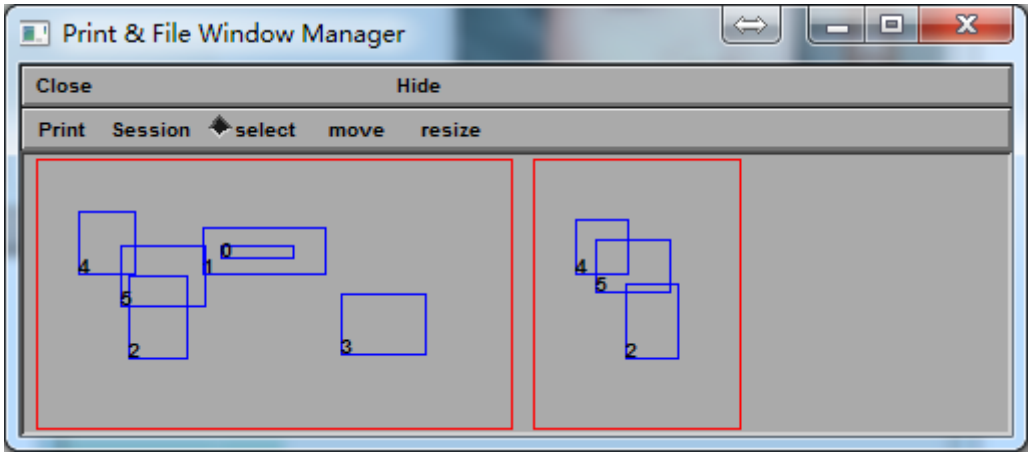

图 11

在图 11 中，左边的红框表示你的电脑显示屏幕，蓝框表示屏幕上 NEURON 的所有窗口，保持它们在屏幕上的相对位置不变. 因此我们可以知道每个蓝框对应的

是哪个窗口. 最后我们可以选中蓝框打印我们想要的窗口, 蓝框就会自动跳到右边红框中. 右边红框表示我们打印的 A4 纸. 最后点击 Print 命令就可以完成打印, 推荐保存成 pdf 格式. 或者我们可以点击 Session 命令保存成.ses 格式,在下次研究时加载它 (如 load 和 xopen 命令), 选中的窗口就会再次出现.

- **命令窗口**

这里命令的窗口和 matlab 的命令窗口概念类似, 可以方便地和 NEURON 进行交互, 以及将 hoc 文件中的输出结果打印到它上面.

结束语

以上就是关于 NEURON 使用方法的一个简单介绍, 目的在于让读者了解到它的基本功能以及可以开始读写简单代码. 对于初学 NEURON 的用户来说, 最好的方法就是从阅读别人的代码开始, 对于不清楚的函数查询 Porgrammer's Reference, 进一步熟悉 NEURON. 附件中给出了一个关于树突整合 project 的代码(Hao et.al. PNAS, 2009), 供大家阅读练习.

最后希望本教程能够对大家的科学研究有所帮助!