PraticoNeuro

Parte 1 – Neurônio individual

Exercício 01

Vamos começar com o seguinte código:

```
1   from brian2 import *
2
3   tau = 10 * ms
4   vrest = -70.0 * mV
5   R = 100 * Mohm
6   eqs = '''
7       dv/dt = (-(v-vrest) + R*I)/tau : volt (unless refractory)
8       I : amp
9   '''
10
11  neurons = NeuronGroup(1, eqs, threshold = 'v > -60.0*mV', reset = 'v = vrest',
12                        refractory = 5*ms, method='linear')
13  neurons.v = vrest
14  neurons.I = 100*pA
15
16  state_mon = StateMonitor(neurons, 'v', record = True)
17
18  run (100*ms)
19
20  plot(state_mon.t/ms, state_mon.v[0]/mV)
21  xlabel('Tempo (ms)')
22  ylabel('Voltagem (mV)')
23  show()
```

Obs: caso esteja usando o Jupyter Notebook, adicione na linha 2 o comando: %matplotlib inline

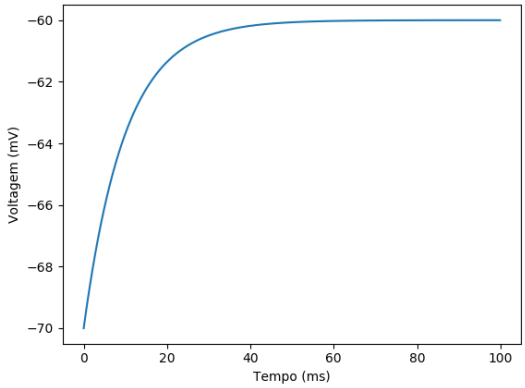
Passo 1. Digite este código em um editor de texto. Salve-o com o nome de sua preferência, porém com a extensão de arquivo do Python (Ex: "example01.py")

Passo 2. Abra o terminal (Linux) ou prompt de comando (Windows) na mesma pasta na qual o arquivo example01.py foi criado.

Passo 3. Rode o arquivo no terminal através do comando:

python example01.py
Ou use: ipython notebook

Compare seu resultado ao abaixo:



Vamos entender o código:

```
from brian2 import *
%matplotlib inline
```

A primeira linha importa as classes e funções do *BRIAN2* e as demais funcionalidades do pacote *pylab*.

A segunda linha permite apresentar os resultados gráficos na mesma janela de execução do programa. Obs: caso esteja rodando os códigos diretamente via terminal (Linux) ou prompt de comando (Windows), esta linha não é necessária, porém, para visualizar o gráfico deve-se adicionar ao fim do código a linha com o comando: show( )

Linha 3 a 5:

```
tau = 10 * ms
vrest = -70.0 * mV
R = 100 * Mohm
```

Define nome de constantes e variáveis com unidade. Para tal basta multiplicar o escalar ou vetor pela unidade física.

Atente-se sempre para as unidades! Caso você tente realizar alguma operação matemática que não faça sentido para as unidades, o código retornará ERROR. Exemplo: **tau + R** retornará erro, pois não é possível somar uma variável em unidade de tempo com outra variável com unidade de resistência (ohm).

Linhas 6 a 9:

```
eqs = '''
    dv/dt = (-(v-vrest) + R*I)/tau : volt (unless refractory)
```

```
        I : amp
    '''
```

eqs é uma equação. Ela calcula o novo valor das variáveis (neste caso: **v**) para cada passo de interação (neste caso: **dt**). Por padrão, **dt** é 0.1*ms.

Observe que, diferente de **tau**, **R** e **vrest**, a variável **I** está sendo definida dentro da equação. Veremos o motivo mais adiante.

Linhas 11 e 12:

```
    neurons = NeuronGroup(1, eqs, threshold = 'v > -60.0*mV', reset = 'v = vrest', refractory = 5*ms, method='linear')
```

O ponto principal do simulador BRIAN é a classe "*NeuronGroup*", uma função que cria um grupo de neurônios que compartilham as mesmas definições e propriedades.

A especificação mínima que a *NeuronGroup* necessita é: número de neurônios (primeiro elemento) e a descrição do modelo de neurônio na forma de uma equação diferencial (segundo elemento, neste caso: **eqs**).

threshold define a condição do limiar de disparo do neurônio. Ou seja, se o potencial de membrana (**v**) ultrapassar certo valor, então é considerado que o neurônio teve um potencial de ação. (Outros modelos de neurônio podem não precisar dessa condição).

reset define o valor que o potencial de membrana (**v**) retornará após a ocorrência de um disparo.

refractory define o tempo refratário absoluto entre disparos de um neurônio.

method refere-se ao método numérico que integrará a equação diferencial do modelo de neurônio. Alguns outros métodos estão implementados no BRIAN, por exemplo: Runge-Kutta de 2ª ('rk2') e 4ª ordem ('rk4'); Euler ('euler'); entre outros.

Linhas 13 e 14:

```
    neurons.v = vrest
    neurons.I = 100*pA
```

Inicialização das variáveis. Neste caso, v será iniciado com seu valor de repouso e será dada uma corrente constante de **I = 100*pA**.

Estes valores são suficientes para que o potencial de membrana aumente em 10 mV a partir do repouso.

Linha 16:

```
    state_mon = StateMonitor(neurons, 'v', record = True)
```

A função “StateMonitor”, como o seu nome sugere, serve para monitorar o estado das variáveis do programa. Neste caso, ela está sendo usada para gravar os valores que a variável **v** assume longo do tempo.

Vamos gravar o valor que **v** assume para cada incremento de **dt** e então plotar tensão (**v)** *versus* tempo (**t)**.

Linha 18:

```
run (100*ms)
```

Rode a simulação pelo tempo definido. Lembre-se de colocar um valor dentro da função como 100*ms por exemplo. “run()” causará um erro ou *looping* infinito.

Opcionalmente, você pode usar o comando: run (100*ms, report = ‘stdout’). O elemento adicionado reportará em seu terminal o andamento da simulação. Faça o teste!

Linhas 20 a 23:

```
plot(state_mon.t/ms, state_mon.v[0]/mV)
xlabel('Tempo (ms)')
ylabel('Voltagem (mV)')
show()
```

O comando “plot” plotará os dados salvos pelo monitor que criamos anteriormente. Em geral, a função “plot” aceitará no mínimo dois argumentos, o primeiro são os pontos no eixo x e o segundo são os pontos no eixo y.

“xlabel” e “ylabel” são funções para a adição de legendas nos eixos x e y respectivamente.

Observe o gráfico. A evolução de **v** ao longo do tempo parece uma exponencial?

- Ela é coerente com a equação colocada em **eqs**?

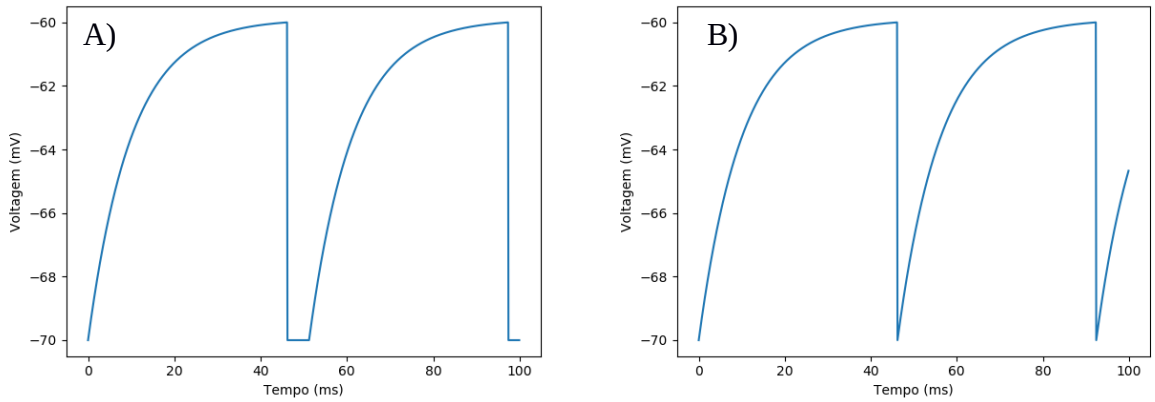
- Altere o valor da corrente **I** para 0.0*pA e veja o que acontece. Note que agora estamos simulando um neurônio isolado na ausência de qualquer estímulo externo ou de rede.

- Com **I = 0.0*pA**, altere também o valor inicial de voltagem na linha 13 para **neurons.v = -80.0*mV**. Para que valor tende **v** quando **t** vai para infinito? Onde definimos este valor? Teste também para: **neurons.v = - 60.0*mV**.

Exercício 02

Utilizando o mesmo código do exercício anterior, agora vamos alterar o valor da corrente injetada **I** (linha 14) para **I = 101.0*pA**.

- Que resultado esperamos encontrar?
- Qual resultado vamos obter, A ou B (vide gráfico a seguir)?
- Há duas maneiras de se obter o outro resultado. Quais são elas? Qual a diferença conceitual e prática entre elas?



****

Note que para o modelo integra-e-dispara com vazamento, os potenciais de ação não são desenhados graficamente, isto se dá, pois, a condição de disparo está descrita na linha 11 em **v > -60.0*mV**. Isto é, quando **v** ultrapassa um limiar, automaticamente o seu valor é levado para o repouso.

Apenas por fins estéticos e didáticos vamos adicionar os potenciais de ação na mão utilizando os instantes que o neurônio disparou. Para isto, primeiro adicionaremos uma função do BRIAN que monitora os instantes dos spikes: “SpikeMonitor( )”. Esta função deverá ser adicionada juntamente com o outro monitor que criamos previamente.

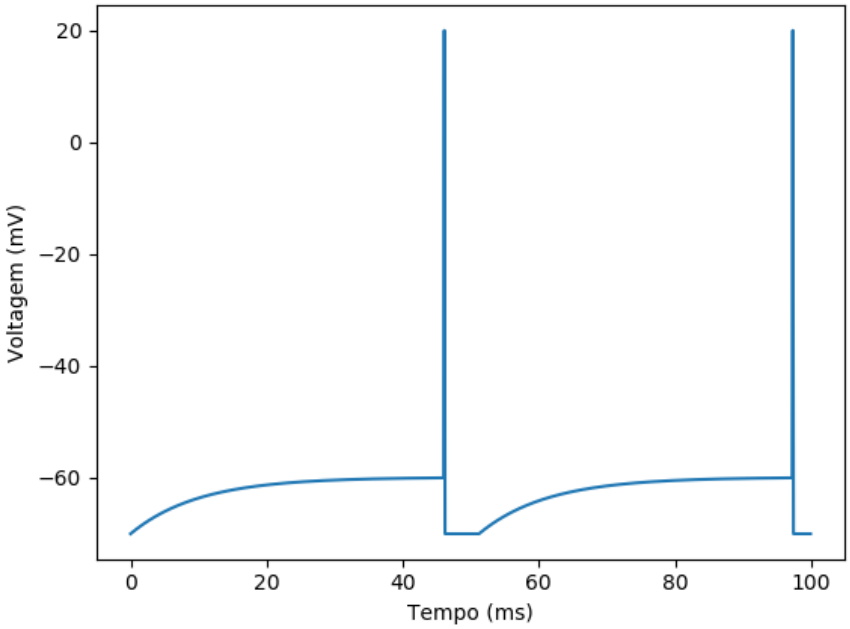
```
spike_mon = SpikeMonitor(neurons)
```

Após o comando “run ( )”, modifique o final do código e substitua pelos comandos a seguir:

```
vm = state_mon.v[0]
for t in spike_mon.spike_trains()[0]:
    i = int(t / defaultclock.dt)
    vm[i] = 20*mV

plot(state_mon.t/ms, vm/mV)
xlabel('Tempo (ms)')
ylabel('Voltagem (mV)')
show()
```

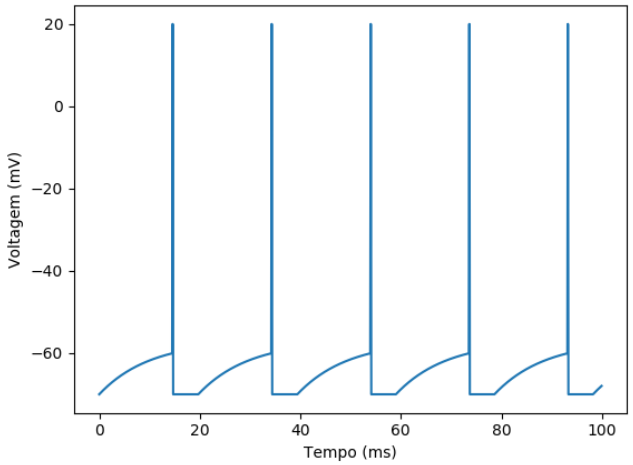
O gráfico A acima redesenhado terá a seguinte aparência:



****

Exercício 03

Vamos alterar uma única linha do programa para obter ao menos 5 spikes (disparos) em 100 ms como na figura abaixo.



a) Qual valor de corrente você utilizou? Qual o valor da frequência de disparos desse neurônio em Hertz (número de disparos por segundo)?

b) Qual o valor de corrente para que o mesmo neurônio dispare com o dobro da frequência do item a? Para dobrar a frequência o valor de corrente foi dobrado?

c) Qual a relação entre a frequência de disparos desse neurônio em função da corrente contínua injetada?

Dica: o monitor "spike_mon" definido no exercício anterior guarda os instantes e a contagem dos disparos do neurônio.

Exercício 04

Antes de respondermos o item c do exercício 3 de uma maneira mais eficiente, vamos primeiro aprender como criar grupos de neurônios.

Vamos retornar ao código do ex 1:

```
1   from brian2 import *
2
3   tau = 10 * ms
4   vrest = -70.0 * mV
5   R = 100 * Mohm
6   eqs = '''
7       dv/dt = (-(v-vrest) + R*I)/tau : volt (unless refractory)
8       I : amp
9   '''
10
11  neurons = NeuronGroup(1, eqs, threshold = 'v > -60.0*mV', reset = 'v = vrest',
12                        refractory = 5*ms, method='linear')
13  neurons.v = vrest
14  neurons.I = 100*pA
15
16  state_mon = StateMonitor(neurons, 'v', record = True)
17
18  run (100*ms)
19
20  plot(state_mon.t/ms, state_mon.v[0]/mV)
21  xlabel('Tempo (ms)')
22  ylabel('Voltagem (mV)')
23  show()
```

Iremos, então, criar 3 neurônios de modo que cada um deles tenham os mesmos parâmetros, porém, recebam valores distintos de corrente injetada.

Na linha 2 defina uma variável "n" referente ao número de neurônios que serão simulados:

```
n = 3
```

Já na linha 11, substitua o número “1” pela variável “**n**”. E, em seguida, defina o valor da corrente **I** na linha 14 da seguinte forma:

```
neurons.I = '50.0*pA + i*60*pA'
```

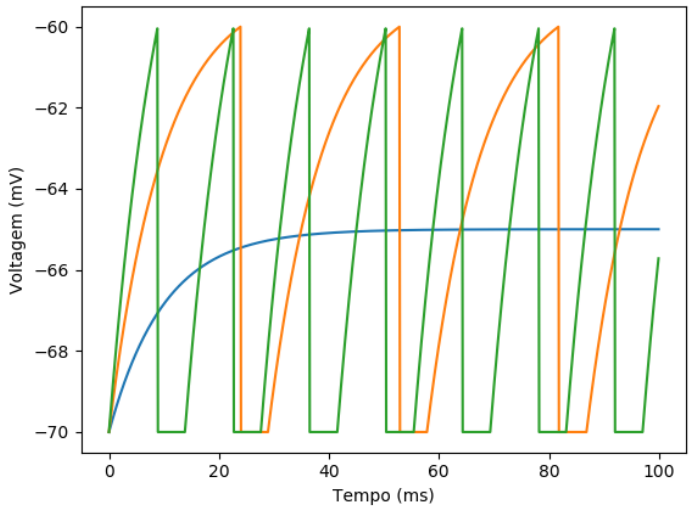
ou

```
neurons.I = [50.0, 110.0, 170.0]*pA
```

Para conferir se cada neurônio recebeu, de fato, um valor de corrente diferente, plote-os adicionando as seguintes linhas após a linha 20:

```
plot(state_mon.t/ms, state_mon.v[1]/mV)
plot(state_mon.t/ms, state_mon.v[2]/mV)
```

Obtenha o seguinte resultado:



Observe que três curvas foram plotadas, a curva do neurônio 0 (state_mon.v[0]), a do neurônio 1 (state_mon.v[1]) e a do neurônio 2 (state_mon.v[2]).

Apenas olhando o gráfico obtido, qual cor/curva (azul, verde, vermelho) corresponde a qual neurônio (0,1,2)?

Caso queira fazer o mesmo, mas com outro parâmetro, por exemplo o **R**, é necessário que o parâmetro seja declarado dentro de “eqs”:

```
eqs = '''
    dv/dt = (-(v-vrest) + R*I)/tau : volt (unless refractory)
    I : amp
    R : ohm
'''
```

E então alterá-lo da mesma forma como foi feito com a corrente **I**.

Exercício 05

Podemos facilmente obter a resposta do item c) do exercício anterior utilizando o BRIAN. Note que para essa tarefa não é necessário que saibamos todos os valores dos potenciais de membrana do neurônio, mas sim, o número de vezes que ele disparou. Portanto, utilizaremos o monitor SpikeMonitor (), definido anteriormente como:

```
spike_mon = SpikeMonitor (neurons)
```

Ao final do código, digite:

```
print(spike_mon.count[0])
```

Este último comando lhe dará a contagem de disparos do neurônio de índice 0. Caso o seu grupo "neurons" tivesse 2 neurônios, então os índices desses neurônios seriam 0 e 1. Portanto, para ver a contagem do 2º neurônio, seria apenas necessário visualizar a variável: spike_mon.count[1].

Com base nessas informações:

a) Construa o gráfico da curva F-I para esse neurônio. Para isso, aumente o tempo de simulação para 1000 ms e simule 1000 neurônios na mesma simulação, porém, cada um recebendo um valor diferente de corrente.

b) Qual o valor máximo de frequência de disparos que esse neurônio alcança?

c) Qual a relação do tempo refratário com esse valor máximo? Altere o valor do tempo refratário e verifique sua influência.

Exercício extra

Compreendendo bem os códigos feitos até aqui, você pode realizar simulações da maioria dos modelos abstratos disponíveis na literatura.

Para exemplificar, podemos construir o código do modelo AdEx, que é o modelo integra-e-dispara exponencial com adaptação. Com este modelo é possível obter outras classes eletrofisiológicas mais complexas que a do modelo que usamos até aqui.

Seguindo o raciocínio montado até agora, tudo o que precisamos saber são as equações diferenciais envolvidas e os parâmetros pertencentes a ela. E então substituir de forma adequada em nosso código inicial.

O modelo AdEx contém os seguintes parâmetros:

```
S = 20000 * um**2
Cm = S*1*uF/cm**2
gl = S*0.05*msiemens/cm**2
El = -60.0*mV
delta = 2.5*mV
vt = -50.0*mV
vrest = -60.0*mV
vcut = -50.0*mV + 5*2.5*mV
tau_w = 600*ms
```

E, diferente do integra-e-dispara, este modelo tem duas equações diferencias acopladas:

```
eqs = '''
    dv/dt = (-gl*(v-El) + gl*delta*exp((v-vt)/delta)- u + I)/Cm : volt (unless refractory)
    du/dt = (1/tau_w)*(a*(v-El)-u) : amp

    I : amp
    a : siemens
    b : amp
'''
neurons = NeuronGroup(1, eqs, threshold = 'v >= vcut', reset = reset_adex,
                      refractory = 5*ms, method='euler')
```

Estes são os itens necessários para alterar o modelo. Note que fixamos a maior parte dos parâmetros e mantivemos como variável apenas o “a” e o “b”. Isto, porque nesse modelo esses dois parâmetros são suficientes para obtermos diversos tipos de disparos diferentes.

- Defina:

neurons.a = 0.001*uS

neurons.b = 0.005*nA

E rode o programa.

- Rode o mesmo programa, porém, varie o valor de b aumentando-o aos poucos. O que você observou?

Códigos:

Exercício 1 a 3:

Abaixo está uma possível resolução baseada no exercício 1.

```
from brian2 import *

tau = 10 * ms
vrest = -70.0 * mV
R = 100 * Mohm
eqs = '''
    dv/dt = (-(v-vrest) + R*I)/tau : volt (unless refractory)
    I : amp
'''

neurons = NeuronGroup(1, eqs, threshold = 'v > -60.0*mV', reset = 'v = vrest',
                      refractory = 5*ms, method='linear')
neurons.v = vrest
neurons.I = 100*pA

state_mon = StateMonitor(neurons, 'v', record = True)

run (100*ms)

plot(state_mon.t/ms, state_mon.v[0]/mV)
xlabel('Tempo (ms)')
ylabel('Voltagem (mV)')
show()
```

Exercício 4:

```
from brian2 import *
n = 3
tau = 10 * ms
vrest = -70.0 * mV
R = 100 * Mohm
eqs = '''
    dv/dt = (-(v-vrest) + R*I)/tau : volt (unless refractory)
    I : amp
'''

neurons = NeuronGroup(n, eqs, threshold = 'v > -60.0*mV', reset = 'v = vrest',
                      refractory = 5*ms, method='linear')
neurons.v = vrest
neurons.I = '50.0*pA + i*60*pA'

state_mon = StateMonitor(neurons, 'v', record = True)

run (100*ms, report = 'stdout')

plot(state_mon.t/ms, state_mon.v[0]/mV)
plot(state_mon.t/ms, state_mon.v[1]/mV)
plot(state_mon.t/ms, state_mon.v[2]/mV)

xlabel('Tempo (ms)')
ylabel('Voltagem (mV)')
show()
```

Exercício 5:

```
from brian2 import *

n = 1000
tau = 10 * ms
vrest = -70.0 * mV
R = 100 * Mohm

eqs = '''
    dv/dt = (-(v-vrest) + R*I)/tau : volt (unless refractory)
    I : amp
'''

neurons = NeuronGroup(n, eqs, threshold = 'v > -60.0*mV', reset = 'v = vrest',
                      refractory = 5*ms, method='linear')
neurons.v = vrest
neurons.I = '300*pA*i/(n-1)'

state_mon = StateMonitor(neurons, 'v', record = True)
spike_mon = SpikeMonitor(neurons)

run (1000*ms, report = 'stdout')

plot(neurons.I/pA, spike_mon.count)
xlabel('I (pA)')
ylabel('Frequencia de disparo (Hz)')
show()
```

Exercício extra:

```
from brian2 import *

S = 20000 * um**2
Cm = S*1*uF/cm**2
gl = S*0.05*msiemens/cm**2
El = -60.0*mV
delta = 2.5*mV
vt = -50.0*mV
vrest = -60.0*mV
vcut = -50.0*mV + 5*2.5*mV
tau_w = 600*ms

eqs = '''
    dv/dt = (-gl*(v-El) + gl*delta*exp((v-vt)/delta)- u + I)/Cm : volt (unless
    refractory)
    du/dt = (1/tau_w)*(a*(v-El)-u) : amp

    I : amp
    a : siemens
    b : amp
'''
reset_adex = '''
    u += b
    v = vrest
    '''

neurons = NeuronGroup(1, eqs, threshold = 'v >= vcut', reset = reset_adex,
                      refractory = 5*ms, method='euler')
neurons.v = vrest
neurons.I = 100.0*pA
neurons.a = 0.001*uS
neurons.b = 0.005*nA

state_mon = StateMonitor(neurons, 'v', record = True)
spike_mon = SpikeMonitor(neurons)

run (1000*ms, report = 'stdout')

vm = state_mon.v[0]
for t in spike_mon.spike_trains()[0]:
    i = int(t / defaultclock.dt)
    vm[i] = 20*mV

plot(state_mon.t/ms, vm/mV)
xlabel('Tempo (ms)')
ylabel('Voltagem (mV)')
show()
```