# مدل سازی شناختی

# Cognitive Modeling

Presented by: Dr. Maleki, Semnan University, Spring 2016, http://maleki.semnan.ac.ir

# مبحث هشتم:
# میان پرده: محاسبه کردن با توابع

ای نامه ی اسرار الهی که تویی
وی آینه ی جمال شاهی که تویی
بیرون ز تو نیست آنچه در عالم هست
از خود بطلب هر آنچه خواهی که تویی

# فهرست مطالب

# معرفی تابع

**ایده‌ی استفاده از تابع:**

- **جزئی از برنامه در قالبی مجزا نوشته می‌شود که امکان ارزیابی و عیب‌یابی مجزای آن و همچنین استفاده‌ی چندباره از آن را فراهم می‌سازد.**

**ساختار تابع:**

- **ورودی‌ها**
- **نام تابع**
- **خروجی‌ها**

**مثال:**

**می‌خواهیم تابعی با دو متغیر ورودی بنویسیم که مربع مجموع دو عدد ورودی را به عنوان خروجی برگرداند.**

**Listing 8.1 Pseudocode function definition**

```
addNSq = functionDefinition (x,y):
output = (x+y)^2
```

**متغیرهای x و y ، محلی (local)، موقتی (temporary) و جانگهدار (placeholder) هستند.**

# فهرست مطالب

- معرفی تابع
- پیاده سازی مدل هاجکین و هاکسلی در پایتون
- شبیه سازی مدل هاجکین و هاکسلی در متلب
- جمع بندی

# پیاده‌سازی مدل‌های جگین و هاکسلی در پایتون

**Listing 8.2 Python code for Hodgkin and Huxley model**

**وارد کردن کتابخانه ترسیم**
**وارد کردن کتابخانه ریاضیات (برای محاسبه نمایی)**

```
import pylab as pyl
import math as m
```

**تعیین متغیرهای مربوط به مقادیر ثابت**

```
vinit = 0.0
dt = 0.01
ena = 115
gna = 120
ek = -12
gk = 36
el = 10.6
gl = 0.3
```

| $E_{Na}$ | $\bar{g}_{Na}$ | $E_K$ | $\bar{g}_K$ | $E_L$ | $\bar{g}_L$ |
|---|---|---|---|---|---|
| $115\ mV$ | $120\ mS/cm^2$ | $-12\ mV$ | $36\ mS/cm^2$ | $10.6\ mV$ | $0.3\ mS/cm^2$ |

**معرفی تابع قاعده‌ی بروزرسانی**

```
def upd (x,dlta_x):
    return (x + dlta_x * dt)
```

**مقدار جدید = مقدار قبلی + نرخ تغییرات مقدار × گام زمانی**

$$\dot{n} = \alpha_n(V)(1-n) - \beta_n(V)n$$

$$\dot{m} = \alpha_m(V)(1-m) - \beta_m(V)m$$

$$\dot{h} = \alpha_h(V)(1-h) - \beta_h(V)h$$

```
def mnh0 (a, b): return (a / (a+b))
```

```
def am(v): return ((2.5 - 0.1*v) / (m.exp(2.5 - 0.1*v) - 1 ))
def bm(v): return (4 * m.exp((-1)* v / 18))
def an(v): return ((0.1 - 0.01*v) / (m.exp(1 - (0.1*v)) - 1))
def bn(v): return (0.125 / m.exp ((-1)*v/80))
def ah(v): return (0.07 * m.exp ((-1)* v / 20 ))
def bh(v): return (1/(m.exp(3 - (0.1)*v)+1))
```

```
am0 = am(0)
bm0 = bm(0)
an0 = an(0)
bn0 = bn(0)
ah0 = ah(0)
bh0 = bh(0)
```

$$\alpha_m(V) = \frac{2.5 - 0.1V}{e^{2.5-0.1V} - 1} \qquad \beta_m(V) = 4e^{-\frac{V}{18}}$$

$$\alpha_n(V) = \frac{0.1 - 0.01V}{e^{1-0.1V} - 1} \qquad \beta_n(V) = 0.125e^{-\frac{V}{80}}$$

$$\alpha_h(V) = 0.07e^{-\frac{V}{20}} \qquad \beta_h(V) = \frac{1}{e^{3-0.1V} + 1}$$

```
m0 = mnh0(am0,bm0)
n0 = mnh0(an0,bn0)
h0 = mnh0(ah0,bh0)
```

```
def ina (m, h , v): return ( gna * (m ** 3) * h * ( v - ena ) )
def ik (n, v): return ( gk * (n ** 4) * (v - ek) )
def il (v): return ( gl * (v - el) )
```

$$C\frac{dV(t)}{dt} = I_{injected}(t) - \left[\bar{g}_{Na}m^3\,h\,(V(t) - E_{Na}) + \bar{g}_K n^4 (V(t) - E_K) + \bar{g}_L (V(t) - E_L)\right]$$

$$\dot{m} = \alpha_m(V)(1 - m) - \beta_m(V)m$$

$$\dot{n} = \alpha_n(V)(1 - n) - \beta_n(V)n$$

$$\dot{h} = \alpha_h(V)(1 - h) - \beta_h(V)h$$

```
def newS (v, m, n, h, t):
    if ( t < 5.0 ) or ( t > 6.0 ):
        istim = 0.0
    else:
        istim = 20.0
    dv = ( istim - (ina (m, h, v) + ik(n, v) + il(v)))
    dm = am ( v ) * (1 - m) - bm ( v ) * m
    dn = an ( v ) * (1 - n ) - bn ( v ) * n
    dh = ah ( v ) * (1 - h ) - bh ( v ) * h
    vp = upd ( v , dv )
    tp = t + dt
    mp = upd (m, dm)
    np = upd ( n , dn )
    hp = upd ( h , dh )
    return ( vp ,mp, np , hp , tp )
```

```
vs = []
ms = []
ns = []
hs = []
ts = []
a, b, c, d, e = newS( vinit, m0, n0 , h0, 0.0 )
vs.append (a)
ms.append (b)
ns.append (c)
hs.append (d)
ts.append (e)
for i in ( range(2, 3000) ):
    a, b, c, d, e = newS( vs[-1], ms[-1], ns[-1], hs[-1], ts[-1])
    vs.append (a)
    ms.append (b)
    ns.append (c)
    hs.append (d)
    ts.append (e)

pyl.plot(ts, vs)
pyl.show()
```

با ذخیره کردن این کد با نام handh.py و نوشتن

python2 handh.py

در خط فرمان می‌توان آن را اجرا نمود.

# فهرست مطالب

- ✓ معرفی تابع
- ✓ پیاده سازی مدل هاجکین و هاکسلی در پایتون
- ○ شبیه سازی مدل هاجکین و هاکسلی در متلب
- ○ جمع بندی

# شبیه سازی در متلب

```
% Preparation
clear all, close all, clc,

% Constants
E_Na = 115;          % Sodium reversal voltage
g_Na = 120;          % Sodium conductance
E_K = -12;           % Potassium reversal voltage
g_K = 36;            % Potassium conductance
E_L = 10.6;          % Leak reversal voltage
g_L = 0.3;           % Leak conductance

% time and Input Current
dt = 0.01; t = 0:dt:20;

I_injected = zeros(size(t));

I_injected(100:200)=20;        % Optional
```

```
%% Initial Values
V = 0; V_dot = 0;
Am = ( 2.5 - 0.1*V ) / ( exp(2.5-0.1*V) - 1 );% alpha_m
Bm = 4 * exp(-V/18);                          % beta_m
An = ( 0.1 - 0.01*V ) / ( exp(1-0.1*V) - 1 ); % alpha_n
Bn = 0.125 * exp(-V/80);                      % beta_n
Ah = 0.07 * exp(-V/20);                       % alpha_h
Bh = 1 / ( exp(3-0.1*V) + 1 );                % beta_h

m_dot = 0;
n_dot = 0;
h_dot = 0;
m = Am / (Am + Bm);
n = An / (An + Bn);
h = Ah / (Ah + Bh);

I_Na = g_Na * m^3 * h * (V-E_Na);
I_K = g_K * n^4 * (V-E_K);
I_L = g_L * (V-E_L);
```

$$\dot{m} = \alpha_m(V)(1-m) - \beta_m(V)m$$

$$\dot{n} = \alpha_n(V)(1-n) - \beta_n(V)n$$

$$\dot{h} = \alpha_h(V)(1-h) - \beta_h(V)h$$

$$C\frac{dV(t)}{dt} = I_{injected}(t) - \left[\bar{g}_{Na}m^3\,h\,(V(t)-E_{Na}) + \bar{g}_K n^4(V(t)-E_K) + \bar{g}_L(V(t)-E_L)\right]$$

```
%% Loop
for i=2:length(t),
    Am(i) = ( 2.5 - 0.1*V(i-1) ) / ( exp(2.5-0.1*V(i-1)) - 1 ); % alpha_m
    Bm(i) = 4 * exp(-V(i-1)/18);                               % beta_m
    An(i) = ( 0.1 - 0.01*V(i-1) ) / ( exp(1-0.1*V(i-1)) - 1 );  % alpha_n
    Bn(i) = 0.125 * exp(-V(i-1)/80);                           % beta_n
    Ah(i) = 0.07 * exp(-V(i-1)/20);                            % alpha_h
    Bh(i) = 1 / ( exp(3-0.1*V(i-1)) + 1 );                     % beta_h

    m_dot(i) = Am(i) * (1-m(i-1)) - Bm(i) *m(i-1);
    n_dot(i) = An(i) * (1-n(i-1)) - Bn(i) *n(i-1);
    h_dot(i) = Ah(i) * (1-h(i-1)) - Bh(i) *h(i-1);

    m(i) = m(i-1) + m_dot(i) * dt;
    n(i) = n(i-1) + n_dot(i) * dt;
    h(i) = h(i-1) + h_dot(i) * dt;

    I_Na(i) = g_Na * m(i)^3 * h(i) * (V(i-1)-E_Na);
    I_K(i) = g_K * n(i)^4 * (V(i-1)-E_K);
    I_L(i) = g_L * (V(i-1)-E_L);

    V_dot(i) = I_injected(i) - ( I_Na(i) + I_K(i) + I_L(i) );
    V(i) = V(i-1) + V_dot(i) * dt;
end
```

$$\alpha_m(V) = \frac{2.5 - 0.1V}{e^{2.5-0.1V} - 1}$$

$$\beta_m(V) = 4e^{-\frac{V}{18}}$$

$$\alpha_n(V) = \frac{0.1 - 0.01V}{e^{1-0.1V} - 1}$$

$$\beta_n(V) = 0.125e^{-\frac{V}{80}}$$

$$\alpha_h(V) = 0.07e^{-\frac{V}{20}}$$

$$\beta_h(V) = \frac{1}{e^{3-0.1V} + 1}$$

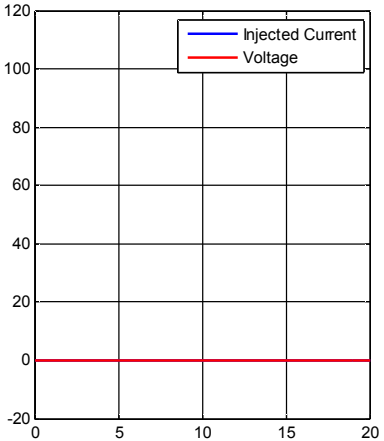
$$\dot{m} = \alpha_m(V)(1-m) - \beta_m(V)m$$

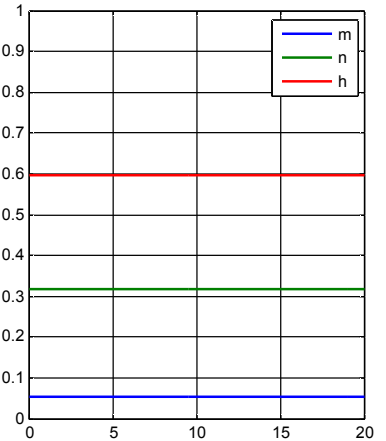
$$\dot{n} = \alpha_n(V)(1-n) - \beta_n(V)n$$

$$\dot{h} = \alpha_h(V)(1-h) - \beta_h(V)h$$

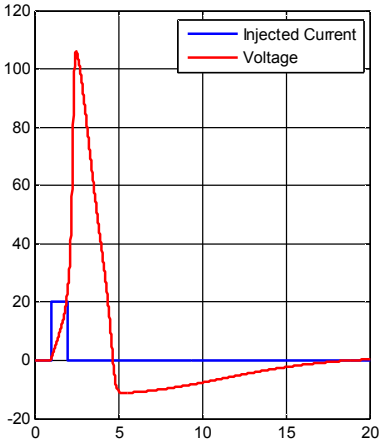
$$C\frac{dV(t)}{dt} = I_{injected}(t) - \left[\bar{g}_{Na}m^3\,h\,(V(t) - E_{Na}) + \bar{g}_K n^4 (V(t) - E_K) + \bar{g}_L (V(t) - E_L)\right]$$

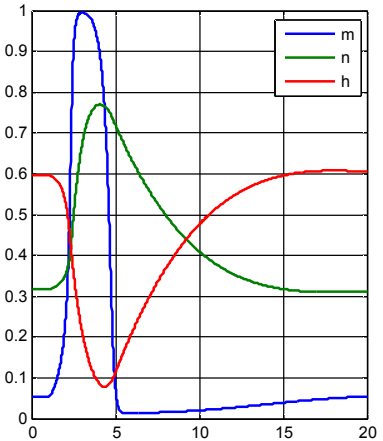
```
%% Generating plots
figure, set(gcf,'Position',[300 100 800 400])
subplot(1,2,1),
plot(t,I_injected,'b',t,V,'r','LineWidth',2), ylim([-20 120]), grid,
legend('Injected Current','Voltage'),
subplot(1,2,2), plot(t,m,t,n,t,h,'LineWidth',2), ylim([0 1]), grid,
legend('m','n','h'),
```


Injected Current
Voltage
120
100
80
60
40
20
0
-20
0
5
10
15
20


m
n
h
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20

# شبیه سازی در متلب



# فهرست مطالب

- معرفی تابع
- پیاده سازی مدل هاجکین و هاکسلی در پایتون
- شبیه سازی مدل هاجکین و هاکسلی در متلب
- جمع بندی

# فهرست مطالب

- معرفی تابع
- پیاده سازی مدل هاجکین و هاکسلی در پایتون
- شبیه سازی مدل هاجکین و هاکسلی در متلب
- جمع بندی


اگر آماده اشتباه کردن نباشید
هیچوقت فکر نابی به ذهنتان نخواهد رسید...
کن رابینسون
www.Bahrampoor.com آموزش سخنرانی و فن بیان