Redes Neuronales y Control Difuso

Guia1
Martin Noblía
Marcos Panizzo
Damián Presti

April 9, 2014

1 Ejercicio 1

En este ejercicio trabajaremos con una neurona I&F con los siguientes parámetros:

```
tau = 20.  # time constant(ms)
R = 10.  # input Resistance(MOhms)
C = tau / R  # capacidad
v_thres = 40  # thresold voltage(mV)
```

a) Programar la Neurona cuando se aplica un pulso de corrient constante (I_{max})

b) Usar el método de Euler forward y backward. Mostrar que el método forward se vuelve inestable si $dt > 2\tau$

c) Calcular la curva teórica que predice la frecuencia en función de la corriente aplicada.

d) Verificar la predicción por medio de simulaciones. O sea, simular distintos valores de corrientes y medir la frecuencia de emisión de spikes. Solapar sus mediciones computacionales con la curva teórica en c).

1.0.1 a)

1.1 Neuronas Integrate and Fire:

En este modelo suponemos que debajo de un potencial umbral v_{thres} de disparo la neurona se comporta como un circuito RC y cuando sobrepasa v_{thres} dispara un spike.

Como sabemos la ecuación diferencial que modela un circuito RC es:

$$C\frac{dv}{dt} = \frac{-v}{R} + I$$

o equivalentemente:

$$\frac{dv}{dt} = \frac{-v}{RC} + \frac{I}{C}$$

Para resolver numéricamente esta ecuación diferencial lineal se nos proponen los métodos de Euler(*Forward* y *Backward*) que consisten en aproximar la derivada como $\frac{V^j - V^{j-1}}{dt}$, donde dt es el paso de discretización y en el caso *forward* la iteración de referencia es la $(j-1)$ por ello la ecuación diferencial discretizada queda:

$$\frac{V^j - V^{j-1}}{dt} + \frac{V^{j-1}}{\tau} = f^{j-1}$$

o equivalentemente:

$$V^j = (1 - \frac{dt}{\tau})V^{j-1} + dt f^{j-1}$$

Para el caso del método *Backward* la iteración de referencia es la (j) por ello la ecuación diferencial discretizada queda:

$$\frac{V^j - V^{j-1}}{dt} + \frac{V^j}{\tau} = f^j$$

o equivalentemente:

$$V^j = \frac{V^{j-1} + dt f^j}{1 + \frac{dt}{\tau}}$$

Modelamos a la neurona de acuerdo a la siguiente clase que tiene un metodo de simulación pedido

In [29]:
```python
import numpy as np

class NeuronaIF:
    """
    Clase NeuronaIF que modela una neurona Integrate and fire

    """

    #Constructor de la clase
    def __init__(self, tau, R, v_thres):
        self.tau = tau
        self.R = R
        self.v_thres = v_thres
        self.C = tau / R

    # Metodo de simulacion
    def simulate(self, dt=1, method='forward', t_max=1000, I_max=.2, n_sims=1):
        """Metodo para simular la neurona"""
        self.n_sims = n_sims
```

```
        self.I_max = I_max
        self.dt = dt
        self.t_max = t_max
        self.method = method

        t_v = np.arange(0,self.t_max - self.dt + 1, dt,dtype=float)
        nt_v = len(t_v)

        dt_tau = self.dt / self.tau
        one_dt_tau = 1 - dt_tau # forward Euler
        m1_dt_tau = 1 / (1 + dt_tau) # backward Euler
        Nt = len(t_v)
        i_v_i = np.zeros_like(t_v) # nA

        #i_v_i[(Nt*3/4.):(Nt*3/4.))
        i_v_i[0:(Nt*3/4)] = I_max # nA

        for j in xrange(0, self.n_sims):


            i_v_tot = i_v_i

            v_v = np.zeros_like(t_v)
            s_v = np.zeros_like(t_v)

            for i in xrange(1,nt_v):
                # forward Euler
                if (self.method == 'forward'):

                    v_v[i] =((1 - self.dt / self.tau) * v_v[i-1] )+ (self.dt * i_v_tot[i-1] / s

                # backward Euler
                elif (self.method == 'backward'):

                    v_v[i] = (m1_dt_tau * v_v[i-1] )+ (self.dt * (i_v_tot[i] / self.C))

                if (v_v[i] >= self.v_thres):

                    v_v[i] = 0.
                    s_v[i] = 1.

        vteo = self.I_max*self.R*(1 - np.exp(-t_v / self.tau))

        return t_v, v_v, i_v_i, vteo

    def freq(self, t_ref):

        self.t_ref = t_ref
        # Corriente minima
        I_min = self.v_thres / self.R

        I = np.linspace(I_min,20)

        f = 1 / (self.t_ref - self.tau * np.log(1 - self.v_thres/ (I * self.R)))
```

```
            return I, f

        # Metodo para imprimir los objetos
        def __str__(self):

            return '[NeuronaIF: tau:%s R:%s v_thres:%s]' % (self.tau, self.R, self.v_thres)
```

```
In [30]: #Objeto de la clase NeuronaIF
         neurona_1 = NeuronaIF(20., 10., 40.)

In [31]: #Simulamos la neurona con los parametros default
         t_v, v_v, i_v_i, vteo = neurona_1.simulate()

In [32]: print neurona_1

[NeuronaIF: tau:20.0 R:10.0 v_thres:40.0]

In [33]: %matplotlib inline

In [34]: import matplotlib.pyplot as plt
         plt.rcParams['figure.figsize'] = 8,6

In [35]: fig, axes = plt.subplots()
         axes.plot(t_v, v_v,'b')
         axes.plot(t_v, i_v_i, 'r')
         axes.plot(t_v, vteo, 'k')

         axes.set_xlabel('$t$', fontsize=20)
         axes.set_title('Simulacion de una Neurona IF', fontsize=20)
         fig.savefig('plot1.pdf')
```

1.2 b)

De acuerdo a la ecuación de discretización del método *forward* si llamamos $a = 1 - \frac{dt}{\tau}$ y evaluamos vemos que se puede generalizar la formula a la siguiente expresión:

$$V^j = a^{j-1}V^1 + dt\sum_{i=1}^{j-1} a^{j-i-1}f^i$$

Al ser una serie geométrica $|a| < 1$ y además para asegurar convergencia $dt < 2\tau$
Vamos a simular nuestra neurona con un $dt = 41$ que no cumpliría la condición de convergencia

```
In [36]: # Simulacion de una neurona con un dt=41
         t_v, v_v, i_v_i, vteo = neurona_1.simulate(41)

In [37]: fig, axes = plt.subplots()
         axes.plot(t_v, v_v,'b')
         axes.plot(t_v, i_v_i, 'r')
         axes.plot(t_v, vteo, 'k')

         axes.set_xlabel('$t$', fontsize=20)
         axes.set_title('Simulacion de una Neurona IF', fontsize=20)

Out[37]: <matplotlib.text.Text at 0x346d790>
```

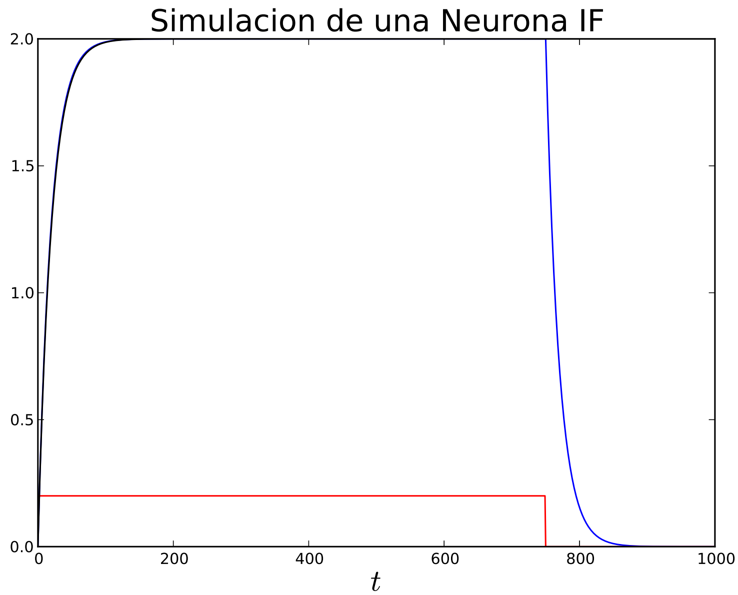

Figure 1:

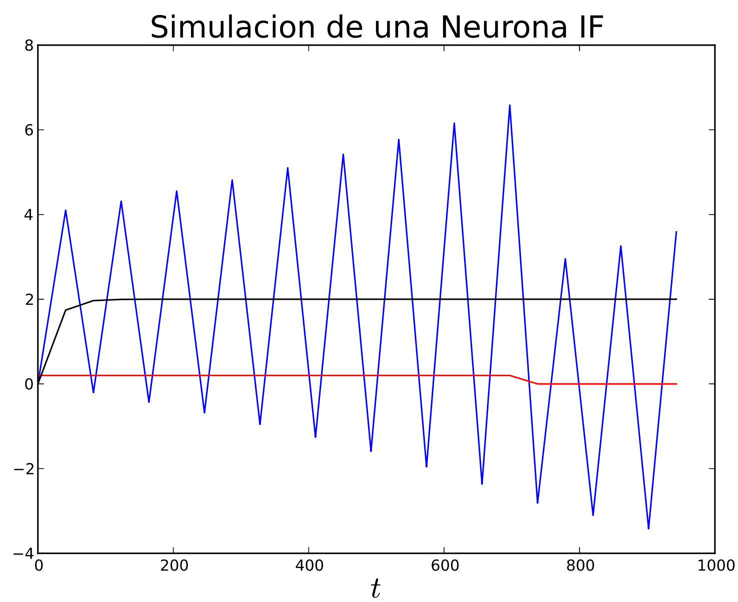

Figure 2:

1.3 c) - d)

Como sabemos la solución a a la ecuación diferencial es:

$$v(t) = IR(1 - exp(\frac{-t}{\tau}))$$

cuando $v(t) = v_{thres}$ tenemos:

$$v_{thres} = IR(1 - exp(\frac{-t_{thres}}{\tau}))$$

Por ello si despejamos t_{thres} nos queda:

$$t_{thres} = -\tau \, log(\frac{1 - v_{thres}}{IR})$$

Luego como la frecuencia es la inversa del tiempo y además teniendo en cuenta que existe un tiempo muerto(t_{ref}) donde la neurona no puede ser exitada

$$f = \frac{1}{t_{ref} + t_{thres}} = \frac{1}{t_{ref} - \tau \, log(1 - \frac{v_{thres}}{IR})}$$

```
In [38]: t_v, v_v, i_v_i, vteo = neurona_1.simulate(1)
```

```
In [39]: I,f = neurona_1.freq(.01)  # el parametro que utilizamos es t_ref = 0.1 ms
```

```
In [40]: f
```

```
Out[40]: array([ 0.        ,  0.01934612,  0.02545683,  0.03074147,  0.03565575,
                 0.04035922,  0.04492744,  0.04940219,  0.05380896,  0.0581644 ,
                 0.06247993,  0.06676371,  0.07102172,  0.0752585 ,  0.07947753,
                 0.08368154,  0.08787273,  0.09205285,  0.09622334,  0.1003854 ,
                 0.10454001,  0.10868801,  0.11283011,  0.11696691,  0.12109893,
                 0.12522662,  0.12935037,  0.13347051,  0.13758735,  0.14170115,
                 0.14581214,  0.14992053,  0.1540265 ,  0.15813021,  0.16223182,
                 0.16633146,  0.17042924,  0.17452528,  0.17861966,  0.1827125 ,
                 0.18680385,  0.1908938 ,  0.19498242,  0.19906977,  0.2031559 ,
                 0.20724087,  0.21132473,  0.21540752,  0.21948928,  0.22357005])
```

```
In [41]: I
```

```
Out[41]: array([  4.        ,   4.32653061,   4.65306122,   4.97959184,
                  5.30612245,   5.63265306,   5.95918367,   6.28571429,
                  6.6122449 ,   6.93877551,   7.26530612,   7.59183673,
                  7.91836735,   8.24489796,   8.57142857,   8.89795918,
                  9.2244898 ,   9.55102041,   9.87755102,  10.20408163,
                 10.53061224,  10.85714286,  11.18367347,  11.51020408,
                 11.83673469,  12.16326531,  12.48979592,  12.81632653,
                 13.14285714,  13.46938776,  13.79591837,  14.12244898,
                 14.44897959,  14.7755102 ,  15.10204082,  15.42857143,
                 15.75510204,  16.08163265,  16.40816327,  16.73469388,
                 17.06122449,  17.3877551 ,  17.71428571,  18.04081633,
                 18.36734694,  18.69387755,  19.02040816,  19.34693878,
                 19.67346939,  20.        ])
```

```
In [44]: # Plots de las frecuencias de spiking
         plt.plot(I,f,'o')
         plt.xlabel('I[nA]', fontsize=20)
         plt.ylabel('frequency', fontsize=20)
         plt.title('Curva teorica de frecuencias', fontsize=20)
```

```
Out[44]: <matplotlib.text.Text at 0x3ba5b10>
```

```
In []:
```

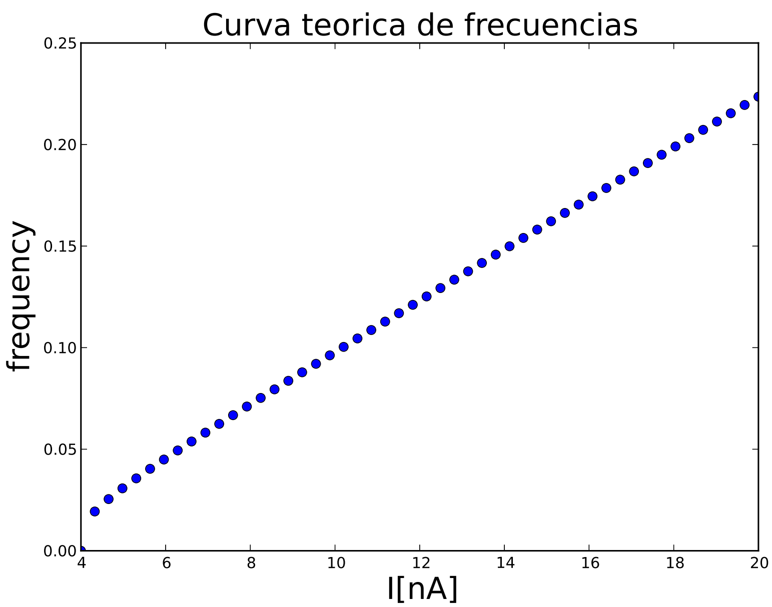
Curva teorica de frecuencias
0.25
0.20
0.15
0.10
0.05
0.00
frequency
4
6
8
10
12
14
16
18
20
I[nA]

Figure 3: