Practica

ALTERNATIVAS - Modelo de Hodgkin-Huxley

Luca Martino

Modelo de Fitzhugh-Nagumo

FitzHugh [20] y por separado Nagumo, Arimoto y Yoshizawa [21], redujeron el modelo de HH e introdujeron un modelo de dos dimensiones más sencillo de tratar analíticamente. La observación básica es la escala de tiempos entre las variables, considerando una variable rápida y una lenta. De acuerdo a las simulaciones se observa que el tiempo característico de activación del canal de sodio es bastante rápido comparado con las otras ecuaciones con lo que se puede considerar a m como constante y eliminando la variable. También se observó que $h+n \approx 0.8$, es decir, esencialmente una constante durante el potencial de acción, con lo cual se puede eliminar otra ecuación, quedando así el modelo de FitzHugh que se aprecia en las ecuaciones (2.43) el cual después fuera representado con circuitos eléctricos por Nagumo.

$$\frac{dx}{dt} = x - \frac{x^3}{3} - y + I_{ext} \tag{2.43a}$$

$$\tau \frac{dy}{dt} = x + a - by \tag{2.43b}$$

El sistema de ecuaciones (2.43) es la forma de sistemas excitables y generalmente son conocidos como variables de excitación y de recuperación. La variable de excitación gobierna la subida al estado excitado, mientras que la variable de recuperación causa el regreso al estado de reposo.

Modelo de Fitzhugh-Nagumo

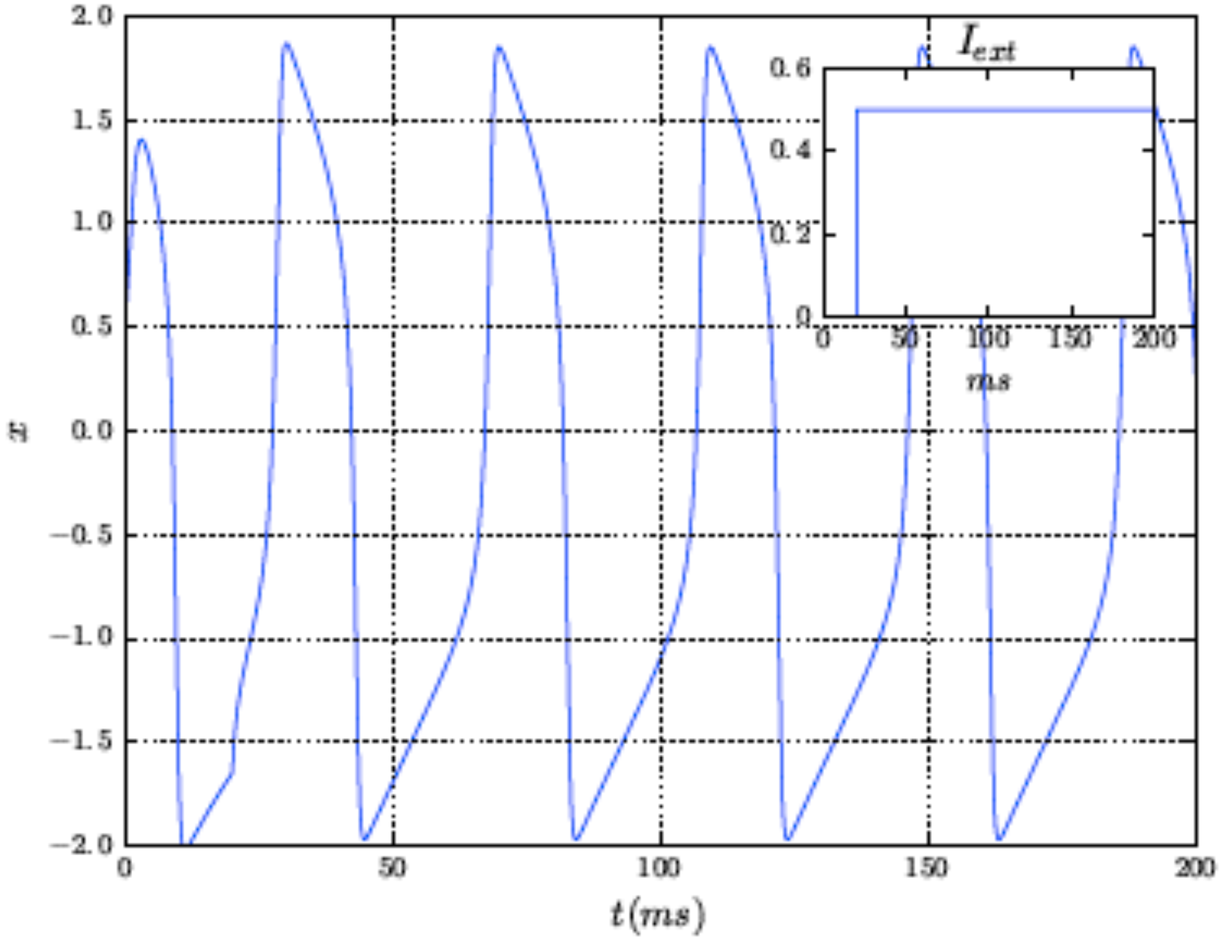

Figura 2.11: Simulación numérica ecuaciones de Fitzhugh-Nagumo con entrada de corriente de excitación de 0.5

Modelo de Fitzhugh-Nagumo

Sin embargo, estas ecuaciones no proporcionan una representación realística de los disparos, los cuales son una característica esencial en la transmisión del impulso nervioso. Por otro lado, la aparición de fenómenos mas complejos como comportamientos caóticos no pueden aparecer al ser propios de dimensiones mayores y de estructuras dinámicas más complejas.

Codigo Python del Modelo de Fitzhugh-Nagumo

El siguiente código es la función de las ecuaciones de Fitzhugh Nagumo cuyos argumentos de entrada son la corriente de excitación externa y el tiempo de simulación y argumentos de salida son los vectores de estado x, y, la corriente y el tiempo.

```
def FN(i_ext,tsim):
    import math
    import numpy as np
    dt=0.001
    loop=int(tsim/dt)
    #a=1
    a=0.7
    #b=0.05
    b=0.8
    #c=0.1
    c=0.08
    t=np.arange(0,loop)*dt
    x=np.zeros((loop,1))
    y=np.zeros((loop,1))
    I=np.zeros((loop,1))
    x[0]=0.5
    y[0]=0.1
    for i in range (0,loop-1):
        if t[i]>20:
            I[i]=i_ext
        else:
            I[i]=0
        x[i+1] = x[i] + dt*(x[i]-(1/3*x[i]**3)-y[i]+I[i])
        y[i+1] = y[i] + dt*(c*(x[i]+a-b*y[i]))
    return x,y,t,I
```

Para poder obtener la gráfica de la Figura 2.11 se utilizó el siguiente código que llama a la función "FN" definida con anterioridad.

Codigo Python del Modelo de Fitzhugh-Nagumo

```
import matplotlib.pyplot as plt
import numpy as np
(x,y,t,I)=FN(0.5,200)
plt.grid(True)
plt.plot(t,x)
plt.xlabel(r"$t␣(ms)$", fontsize = 16)
plt.ylabel(r"$x$", fontsize = 16)
plt.xticks([0,50,100,150,200],[r'$0$',r'$50$',r'$100$',r'$150$',r'$200$'
    ])
plt.yticks([-2,-1.5,-1,-0.5,0,0.5,1,1.5,2],[r'$-2.0$',r'$-1.5$',r'$-1.0$'
    ,r'$-0.5$',r'$0.0$',r'$0.5$',r'$1.0$',r'$1.5$',r'$2.0$'])
a = plt.axes([.63, .61, .24, .24])
plt.plot(t,I)
plt.title('$I_{ext}$', fontsize = 18)
plt.xlabel(r"$ms$", fontsize = 14)
plt.xticks([0,50,100,150,200],[r'$0$',r'$50$',r'$100$',r'$150$',r'$200$'
    ])
plt.yticks([0,0.2,0.4,0.6],[r'$0$',r'$0.2$',r'$0.4$',r'$0.6$'])
plt.show()
```

Modelo de Hindmarsh-Rose

Hindmarsh y Rose modificaron el modelo de FitzHugh y Nagumo (FN) al remplazar la función lineal por una cuadrática, este cambio hizo al modelo capaz de reflejar disparos rápidos que tienen lugar en un intervalo amplio en el tiempo. Posteriormente en [22], modificaron su modelo para obtener un sistema de ecuaciones diferenciales no lineales tridimensional. Este modelo es conocido por ser capaz de demostrar casi todas las características genéricas que el modelo de Hodgkin y Huxley. Las ecuaciones del modelo son:

$$\frac{dx}{dt} = I_{ext} - ax^3 + bx^2 + y - z \tag{2.44a}$$

$$\frac{dy}{dt} = c - dx^2 - \beta y \tag{2.44b}$$

$$\frac{dz}{dt} = \varepsilon(s(x - x_R) - z) \tag{2.44c}$$

Modelo de Hindmarsh-Rose

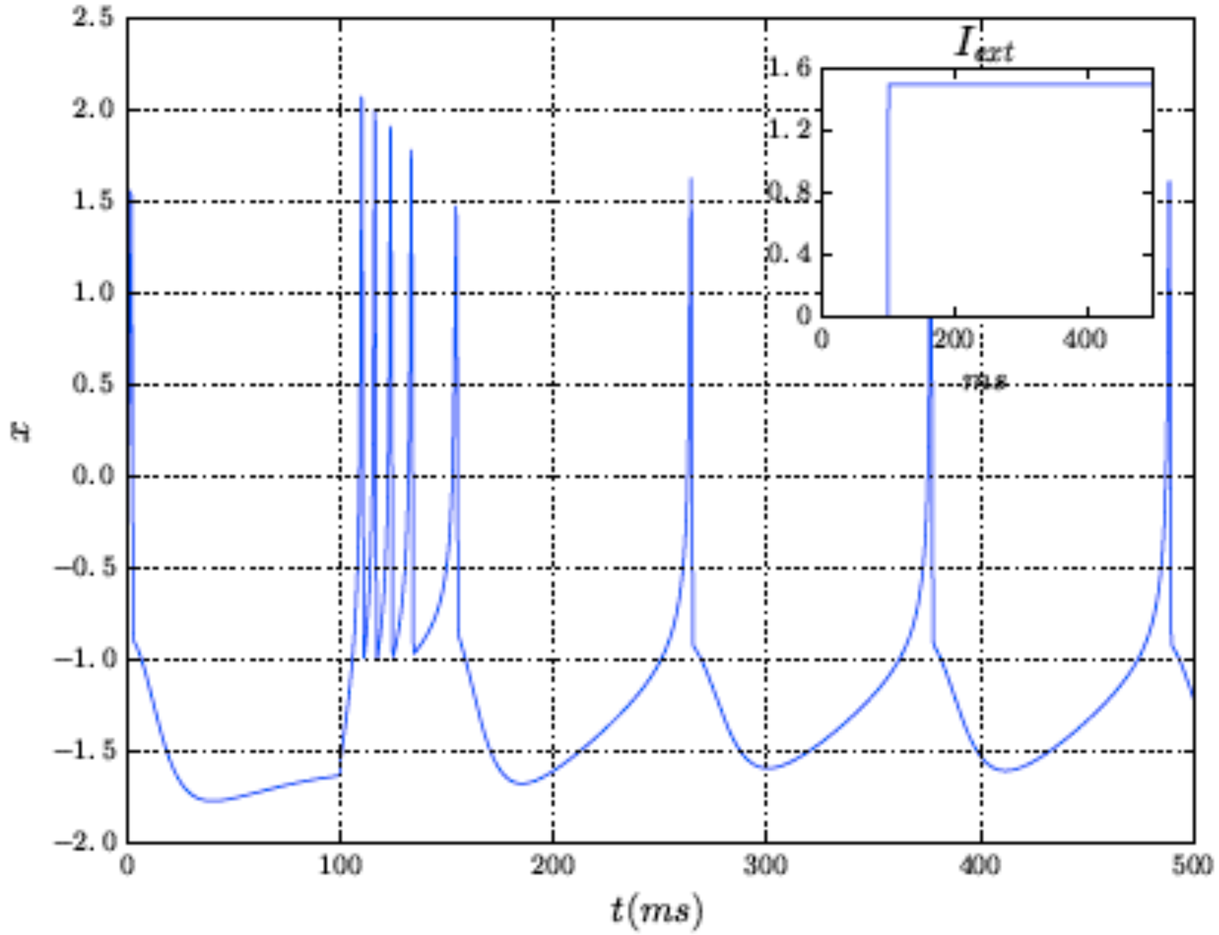

Figura 2.12: Simulación numérica ecuaciones de Hindmarsh-Rose

Modelo de Hindmarsh-Rose

De las ecuaciones (2.44) la variable x representa el voltaje que cruza a través de la membrana celular, mientras que la variable y está asociada a la apertura de canales, y por útimo la variable z describe la activación o inactivación de algunas corrientes. El parámetro constante I_{ext} determina el modo de salida del modelo, que en el caso de la elección de parámetros puede ser potencial de reposo, bursting o spiking, además de que para ciertos parámetros puede presentar un comportamiento caótico.

Con los valores de $a = 1, b = 3, c = 1, d = 5, \beta = 1, \varepsilon = 0.01, s = 4, x_R = 1.6$ y entrada $I_{ext} = 1.5$, el comportamiento mediante simulación numérica obtenida a partir del programa presentado en A.4 es el que se observa en la Figura 2.12. Con el parámetro $b = 2.82$ el comportamiento es el de la Figura 2.13

Modelo de Hindmarsh-Rose

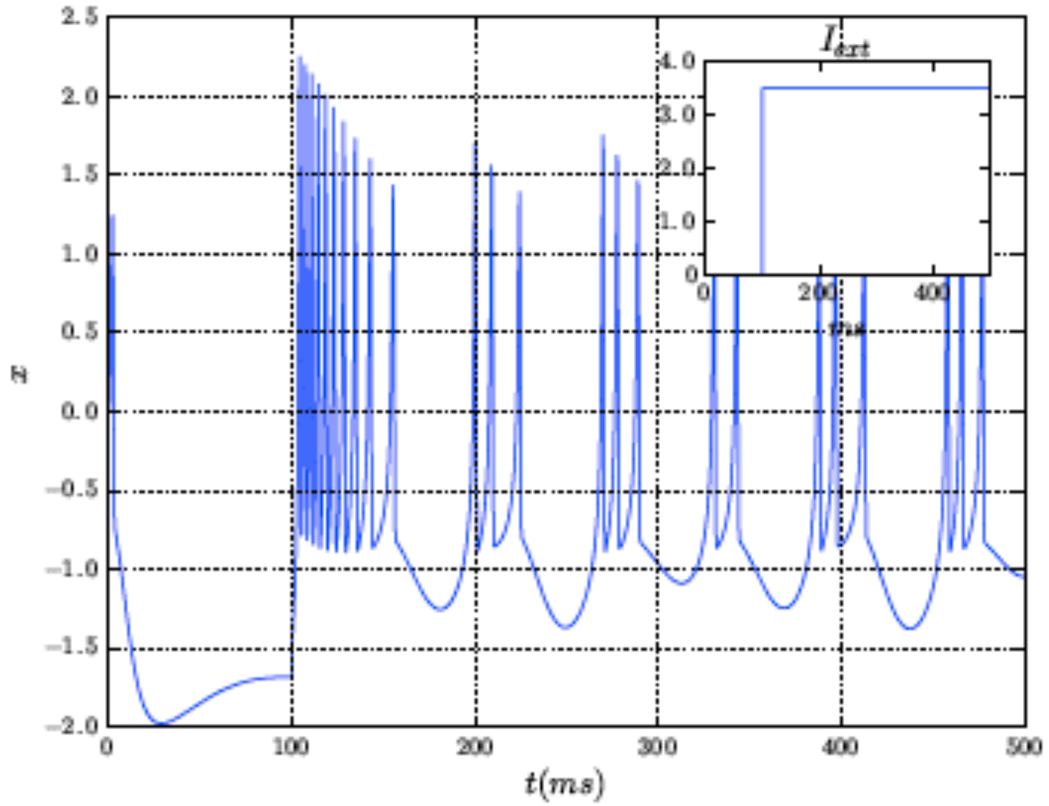

Figura 2.13: Simulación numérica de las ecuaciones de Hindmarsh-Rose con variación del parámetro de bifurcación ($b = 2.82$)

Python code of Modelo de Hindmarsh-Rose

El siguiente código es la función de las ecuaciones de Hindmarsh Rose cuyos argumentos de entrada son los parámetros α, b, c, d, r y s, la corriente de excitación externa y el tiempo de simulación y argumentos de salida son los estados x, y y z la corriente y el tiempo.

```
def HR(a,b,c,d,r,s,i_ext,tsim):
    import math
    import numpy as np
    dt=0.001
    x_R=1.6
    loop=int(tsim/dt)
    t=np.arange(0,loop)*dt
    x=np.zeros((loop,1))
    y=np.zeros((loop,1))
    z=np.zeros((loop,1))
    I=np.zeros((loop,1))
    for i in range (0,loop-1):
        if t[i]>100:
            I[i]=i_ext
        else:
            I[i]=0
        x[i+1] = x[i] + dt*(y[i]-a*x[i]**3+b*x[i]**2-z[i]+I[i])
        y[i+1] = y[i] + dt*(c-d*x[i]**2-y[i])
        z[i+1] = z[i] + dt*(r*(s*(x[i]+x_R)-z[i]))
    return x,y,z,t,I
```

Python code of Modelo de Hindmarsh-Rose

Para poder obtener la gráfica la Figura 2.12 se utilizó el siguiente código que llama a la función "HR" definida con anterioridad.

```
import matplotlib.pyplot as plt
import numpy as np
(x,y,z,t,I)=HR(1,3,1,5,0.01,4,1.5,500)
#(x,y,z,t,I)=HR(1,2.82,1,5,0.02,4,3.5,500)
plt.grid(True)
plt.plot(t,x)
plt.xlabel(r"$t\ (ms)$", fontsize = 16)
plt.ylabel(r"$x$", fontsize = 16)
plt.xticks([0,100,200,300,400,500],[r'$0$',r'$100$',r'$200$',r'$300$',r'
    $400$',r'$500$'])
plt.yticks([-2,-1.5,-1,-0.5,0,0.5,1,1.5,2,2.5],[r'$-2.0$',r'$-1.5$',r'$
    -1.0$',r'$-0.5$',r'$0.0$',r'$0.5$',r'$1.0$',r'$1.5$',r'$2.0$',r'$2.5$'
    ])
a = plt.axes([.63, .61, .24, .24])
plt.plot(t,I)
plt.title('$I_{ext}$', fontsize = 18)
plt.xlabel(r"$ms$", fontsize = 14)
plt.xticks([0,200,400],[r'$0$',r'$200$',r'$400$'])
plt.yticks([0,1,2,3,4],[r'$0$',r'$1.0$',r'$2.0$',r'$3.0$',r'$4.0$'])
plt.show()
```

Para obtener el comportamiento caótico de la Figura 2.13 se cambió el valor del parámetro de entrada *b*, con el programa se obtiene la figura comentando la segunda línea y descomentando la tercera.

Modelo de Morris-Lecar

El modelo de Morris y Lecar (ML) [23] fue formulado en el contexto de la actividad eléctrica de la fibra muscular de los cirrípedos (comúnmente llamados percebes) por lo que es un modelo biológicamente inspirado. En este modelo, los canales de sodio son remplazados por canales de calcio con ω como la variable de activación y sin variable de inactivación. La dinámica está dada por:

$$C_{ML}\frac{dV}{dt} = I_{ext} - \bar{g}_{Ca(ML)} m_{\infty}(V)(V - E_{Ca(ML)}) - \bar{g}_{K(ML)}\omega(V - E_{K(ML)}) - \bar{g}_{L(ML)}(V - E_{L(ML)}) \quad (2.41a)$$

Modelo de Morris-Lecar

$$\frac{d\omega}{dt} = \eta(V)(\omega_\infty(V) - \omega) \tag{2.41b}$$

Donde

$$m_\infty(V) = \frac{1}{2}\left[1 + \tanh\left(\frac{V - V_1}{V_2}\right)\right] \tag{2.42a}$$

$$\omega_\infty(V) = \frac{1}{2}\left[1 + \tanh\left(\frac{V - V_3}{V_4}\right)\right] \tag{2.42b}$$

$$\eta(V) = \frac{1}{\bar{\eta}}\cosh\left(\frac{V - V_3}{2V_4}\right) \tag{2.42c}$$

Con los valores de parámetros $\bar{g}_{L(ML)} = 2$, $\bar{g}_{Ca(ML)} = 4$, $\bar{g}_{K(ML)} = 8$, $E_{L(ML)} = -50mV$, $E_{Ca(ML)} = 100mV$, $V_{K(ML)} = -70$, $V_1 = -1$, $V_2 = 15$, $V_3 = 10$, $V_4 = 14.5$, $I_{ext} = 0.1mA$ y $C_{ML} = 20$ se obtiene la la Figura 2.10 obtenida por el programa presentado en A.2.

Modelo de Morris-Lecar

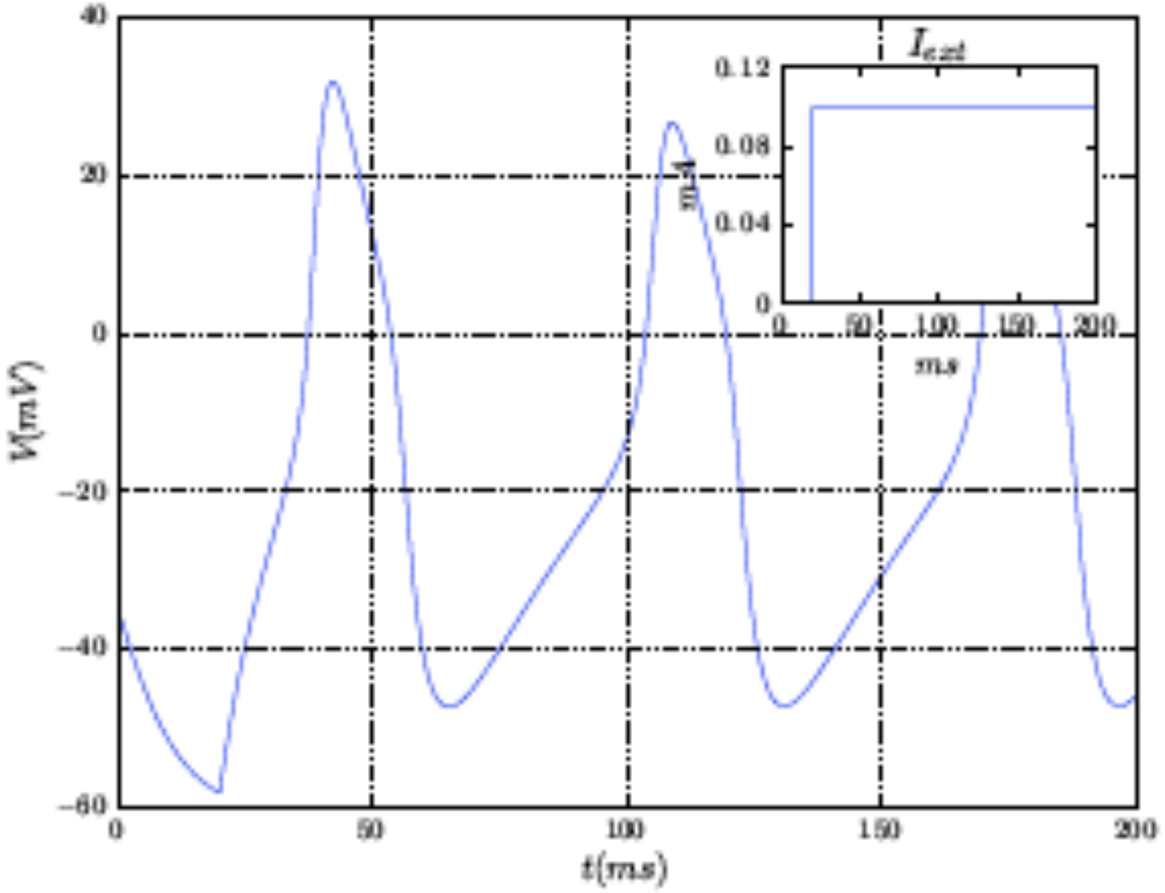

Figura 2.10: Simulación numérica de las ecuaciones de Morris-Lecar

Python code del Modelo de Morris-Lecar

El siguiente código es la función de las ecuaciones de Morris Lecar cuyos argumentos de entrada son el voltaje inicial, la corriente de excitación externa y el tiempo de simulación y argumentos de salida son los vectores de voltaje la corriente y el tiempo.

```
def ML(v0,tsim,i_ext):
    import math
    import numpy as np
    dt=0.001
    loop=int(tsim/dt)
    g_Ca =4.4
    e_Ca=120
    g_K=8
    e_K=-84
    g_L=2
    e_L=-60
    C_m=20
    v1=-1.2
    v2=18
    v3=2
    v4=30
    T0=15
    t=np.arange(1,loop+1)*dt
    V=np.zeros((loop,1))
    W=np.zeros((loop,1))
    I=np.zeros((loop,1))
    V[0]=v0
    W[0]=0.0149
    def M_inf (V):
        x=0.5+0.5*math.tanh((V-v1)/v2)
        return x
    def W_inf (V):
        x=0.5+0.5*math.tanh((V-v3)/v4)
        return x
    def t_W (V):
        x=T0/(math.cosh((V-v3)/(2*v4)))
        return x
    for i in range (0,loop-1):
        if t[i]>20:
            I[i]=i_ext
        else:
            I[i]=0
        V[i+1] = V[i] + dt*((I[i]+(-V[i]+e_Ca)*g_Ca*M_inf(V[i])+(-V[i]+
            e_K)*g_K*W[i]+(-V[i]+e_L)*g_L)/C_m)
        W[i+1] = W[i] + dt*((W_inf(V[i])-W[i])/(t_W(V[i])))
    return V,t,I
```

Python code of Modelo de Morris-Lecar

Para poder obtener la gráfica la Figura 2.10 se utilizó el siguiente código que llama a la función "ML" definida con anterioridad.

```
import matplotlib.pyplot as plt
import numpy as np
(V,t,I)=ML(-35,200,100)
plt.grid(True)
plt.plot(t,V)
plt.xlabel(r"$t\ (ms)$", fontsize = 16)
plt.ylabel(r"$V\ (mV)$", fontsize = 16)
plt.xticks([0,50,100,150,200],[r'$0$',r'$50$',r'$100$',r'$150$',r'$200$'
    ])
plt.yticks([-60,-40,-20,0,20,40],[r'$-60$',r'$-40$',r'$-20$',r'$0$',r'
    $20$',r'$40$'])
a = plt.axes([.63, .61, .24, .24])
plt.plot(t,I)
plt.title('$I_{ext}$', fontsize = 18)
plt.xlabel(r"$ms$", fontsize = 14)
plt.ylabel(r"$m_mA$", fontsize = 14)
plt.xticks([0,50,100,150,200],[r'$0$',r'$50$',r'$100$',r'$150$',r'$200$'
    ])
plt.yticks([0,40,80,120],[r'$0$',r'$0.04$',r'$0.08$',r'$0.12$'])
plt.show()
```