**Neuroscience, Learning, Memory, Cognition**

**Sharif university of technology**

**Dr. Karbalaie**

**Soraya charkas 99101387**

**Homework 1**

## Part one-

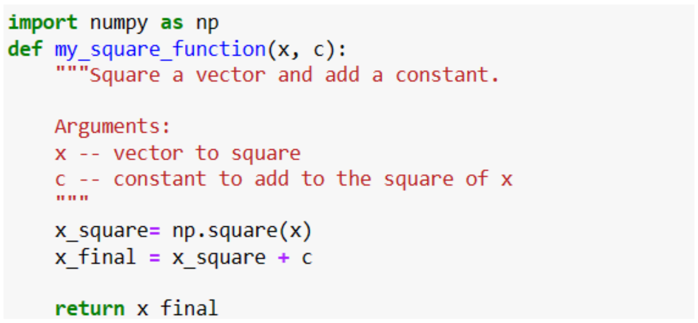
## 1-Working with arrays using PyLab

Q-Search about pylab package and describe it shortly?

Pylab is a module that provides a Matlab like namespace by  importing functions from the modules Numpy and Matplotlib ,Numpy is a module that provides efficient numerical vector calculations based on underlying Fortran and C binary libraries. Matplotlib contains functions to create visualization of data.

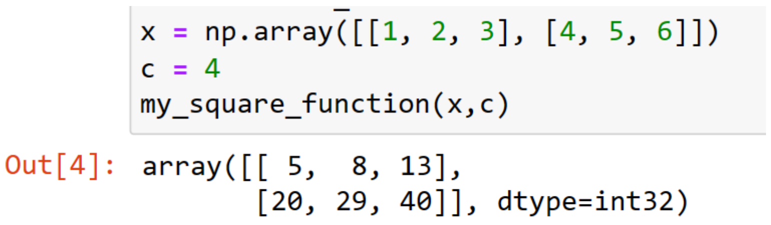
## 2-Defining a new function

We can easily use numpy predefined functions to solve this problem. We can use the “square” function in numpy to square a vector and the add the constant to receive the desired output:



```
import numpy as np
def my_square_function(x, c):
    """Square a vector and add a constant.

    Arguments:
    x -- vector to square
    c -- constant to add to the square of x
    """
    x_square= np.square(x)
    x_final = x_square + c

    return x final
```

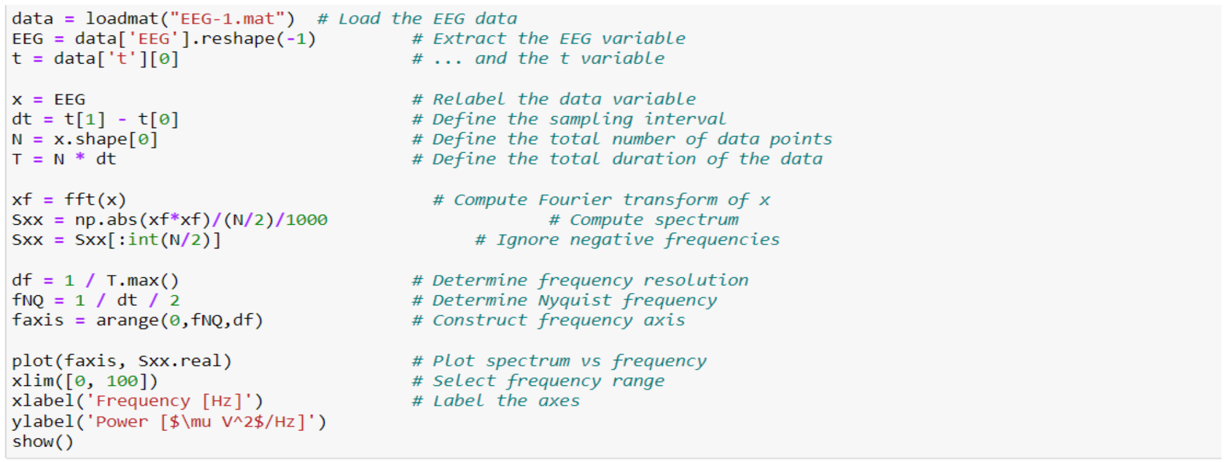
The function is written as above (the written function is available in Jupiter notebook as well). the output for an arbitrary input is as below:



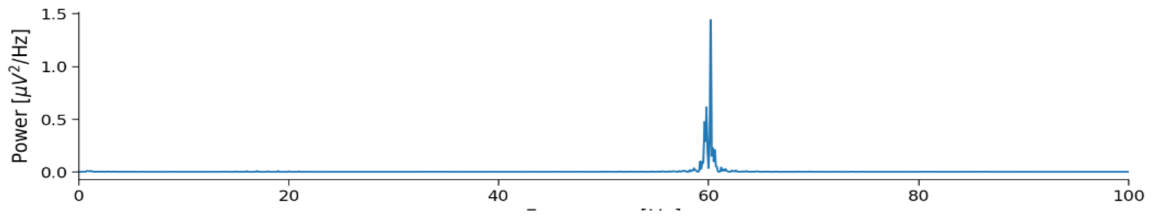
```
x = np.array([[1, 2, 3], [4, 5, 6]])
c = 4
my_square_function(x,c)

Out[4]: array([[ 5,  8, 13],
               [20, 29, 40]], dtype=int32)
```

## 3-loading matlab data and plot power spectrum

We use the predefined fft function to find thee Fourier transform of the given data. Then we compute the spectrum using the theories we already know and ignore the negative data's.



```
data = loadmat("EEG-1.mat")  # Load the EEG data
EEG = data['EEG'].reshape(-1)         # Extract the EEG variable
t = data['t'][0]                      # ... and the t variable

x = EEG                               # Relabel the data variable
dt = t[1] - t[0]                      # Define the sampling interval
N = x.shape[0]                        # Define the total number of data points
T = N * dt                            # Define the total duration of the data

xf = fft(x)                             # Compute Fourier transform of x
Sxx = np.abs(xf*xf)/(N/2)/1000                      # Compute spectrum
Sxx = Sxx[:int(N/2)]                        # Ignore negative frequencies

df = 1 / T.max()                      # Determine frequency resolution
fNQ = 1 / dt / 2                      # Determine Nyquist frequency
faxis = arange(0,fNQ,df)              # Construct frequency axis

plot(faxis, Sxx.real)                 # Plot spectrum vs frequency
xlim([0, 100])                        # Select frequency range
xlabel('Frequency [Hz]')              # Label the axes
ylabel('Power [$\mu V^2$/Hz]')
show()
```



## Part two-

## -Coding exercise 1: python code to simulate LIF neuron

In this part we are trying to simulate the equation below:

$$\tau_m \frac{dV}{dt} = -(V - E_L) + \frac{I}{g_L}$$

To calculate the increment of the membrane potential, we use the equation above to implement it:

```
# Calculate the increment of the membrane potential
dv = (-(v[it] - E_L) + Iinj[it] / g_L) * (dt / tau_m)
```

To update the membrane potential, we add V[it] to dv.

```
# Update the membrane potential
v[it + 1] = v[it] + dv
```

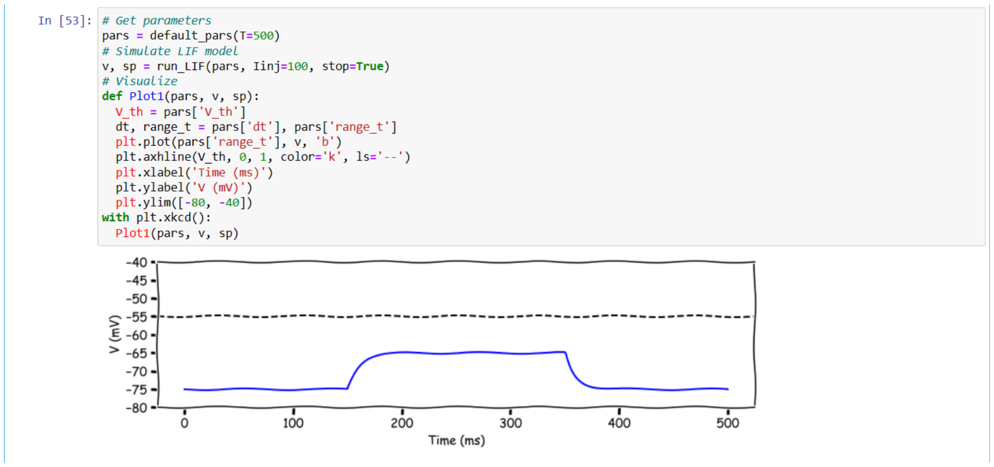
At the beginning, we initialize our parameters which is already written. After that we initialize the voltage by generating an all zero vector. Note that the v[0] should equal to V_init which we have already set before. Next, we generate an all one vector to use as our input current. Next part of our code is a loop to check whether we had a pulse or not and if so, set beginning and ending as 0.the rec_spike array counts the number of spikes.

```
rec_spikes = []  # record spike times
tr = 0.  # the count for refractory duration

for it in range(Lt - 1):

  if tr > 0:  # check if in refractory period
    v[it] = V_reset  # set voltage to reset
    tr = tr - 1 # reduce running counter of refractory period

  elif v[it] >= V_th:  # if voltage over threshold
    rec_spikes.append(it)  # record spike event
    v[it] = V_reset  # reset voltage
    tr = tref / dt  # set refractory time
```

In this loop, we have two states. If tr>0 which means, we are in refractory period we set the v[it] as v_reset which is already initialized and reduce tr and repeat the loop. And if not, if v[it] is already equal or greater than v_th which means we have passed the threshold, besides setting v[it] as v_reset, we increase the number of spikes.

The next two parts were the parts which we were supposed to fill and I have already explained them before.

For plotting, we used the xkcd package which is already used to plot the example figure.



```
In [53]: # Get parameters
         pars = default_pars(T=500)
         # Simulate LIF model
         v, sp = run_LIF(pars, Iinj=100, stop=True)
         # Visualize
         def Plot1(pars, v, sp):
           V_th = pars['V_th']
           dt, range_t = pars['dt'], pars['range_t']
           plt.plot(pars['range_t'], v, 'b')
           plt.axhline(V_th, 0, 1, color='k', ls='--')
           plt.xlabel('Time (ms)')
           plt.ylabel('V (mV)')
           plt.ylim([-80, -40])
         with plt.xkcd():
           Plot1(pars, v, sp)
```

## -Response of an LIF model to different types of input currents

### -Parameter exploration of Direct current (DC) input amplitude

Q-

A-How much DC is needed to reach the threshold (rheobase current)?

By increasing the current, around 215 pA we achieve the desired threshold.

B-How does the membrane time constant affect the frequency of the neuron?

The relation between membrane time constant and firing rate is opposite, as the time constant increases we can declare that the firing rate is decreasing because the membrane is taking more time to reach the threshold.

## -Gaussian white noise (GWN) current

### -LIF neuron explorer for noisy input

Q-

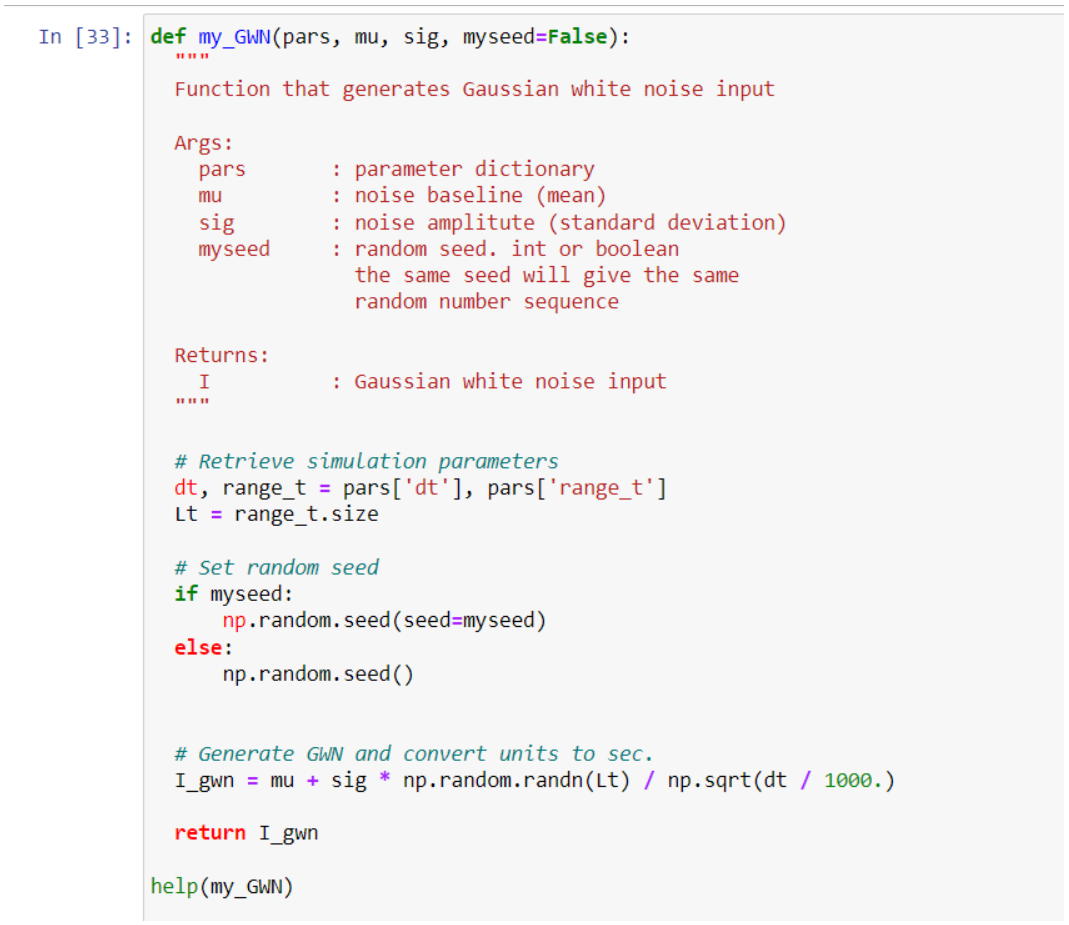
1-How does the minimum input (i.e. $\mu$)needed to make a neuron spike change with increase $\sigma$ ?

-by increasing sigma, the minimum input needed to make a neuron spike becomes lower because the fluctuations make the threshold voltage become lower.

2-how does the spike regularity change with increase in $\sigma$

--$\sigma$ determines the level of disorderliness of spikes. The relation between them is direct, the more the $\sigma$ is, the level of disorderliness of spikes is higher.

The filled function is as below:



```
In [33]: def my_GWN(pars, mu, sig, myseed=False):
           """
           Function that generates Gaussian white noise input

           Args:
             pars       : parameter dictionary
             mu         : noise baseline (mean)
             sig        : noise amplitute (standard deviation)
             myseed     : random seed. int or boolean
                          the same seed will give the same
                          random number sequence

           Returns:
             I          : Gaussian white noise input
           """

           # Retrieve simulation parameters
           dt, range_t = pars['dt'], pars['range_t']
           Lt = range_t.size

           # Set random seed
           if myseed:
               np.random.seed(seed=myseed)
           else:
               np.random.seed()


           # Generate GWN and convert units to sec.
           I_gwn = mu + sig * np.random.randn(Lt) / np.sqrt(dt / 1000.)

           return I_gwn

         help(my_GWN)
```

## -Analyzing GWN Effects on Spiking

- As we increase the input average ($\mu$) or the input fluctuation ($\sigma$) the spike count changes. How much can we increase the spike count, and what might be the relationship between GWN mean/std or DC value and spike count?
- As we increase the GWN mean, we can see that we get closer to the desired DC input so we can assume that the relation between them is direct.
- We have seen above that when we inject DC, the neuron spikes in a regular manner (clock-like), and this regularity is reduced when GWN is injected. The question is, how irregular can we make the neurons spiking by changing the parameters of the GWN?
-

  we can assume that the threshold of how we can make the spike irregular depends on the DC input and it cannot effect more than a certain amount.

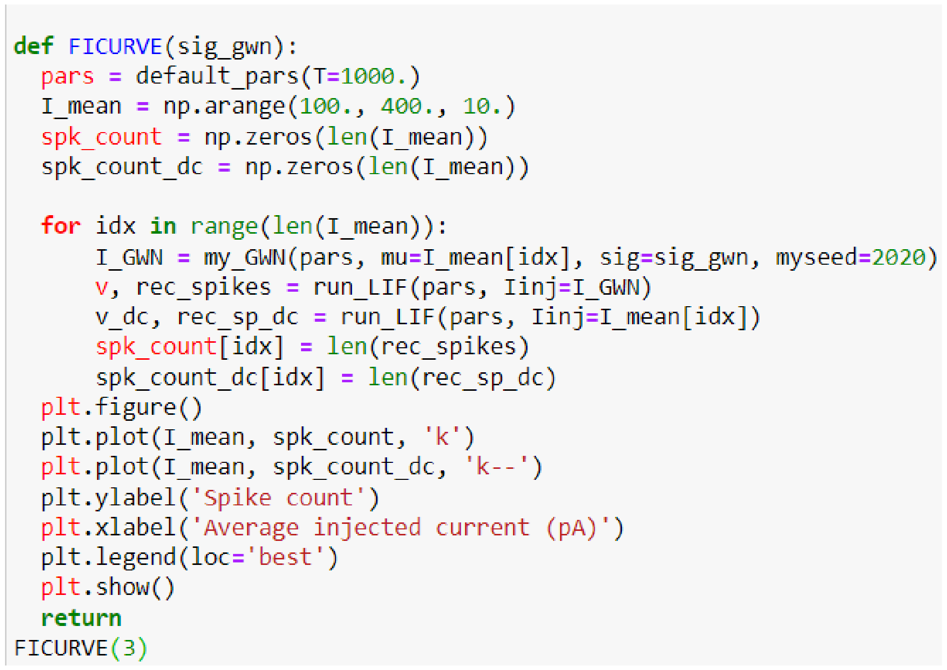
## The interspike interval (ISI)

The interspike interval is the time between subsequent action potentials of a neuron. Action potentials are propagated along the axons of a neuron to reach the nerve terminals where they can trigger the release of chemical messengers to affect other neurons.

## Firing rate and spike time irregularity

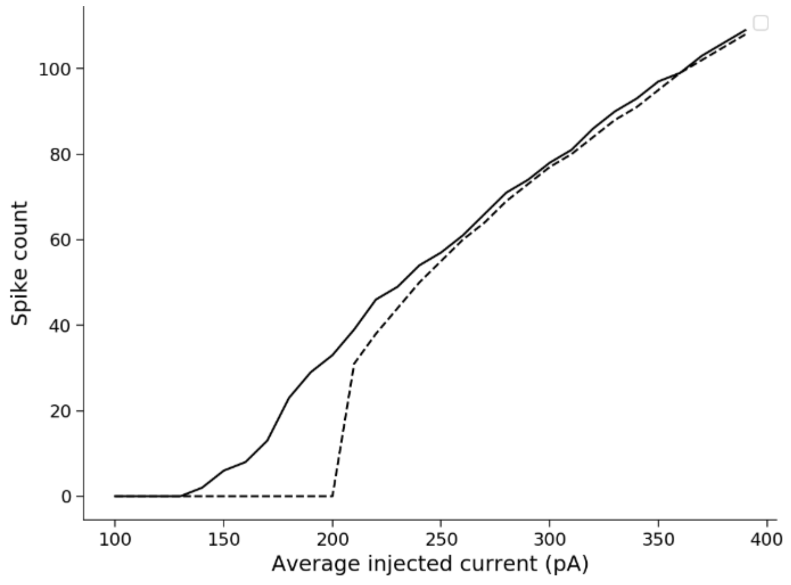
### F-I Explorer for different sig_gwn

Here is a simple code to show the curve with different $\sigma$.

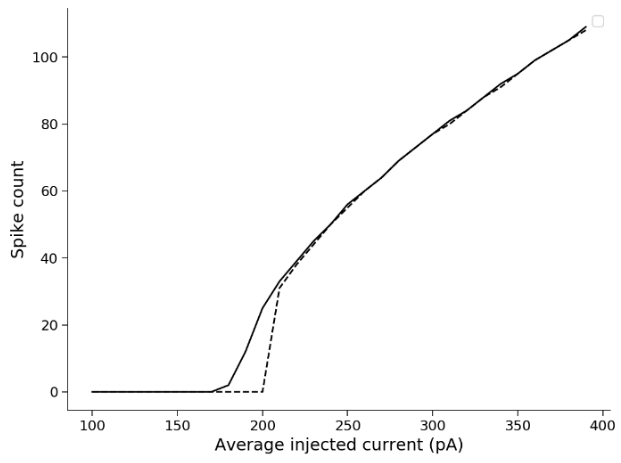


```
def FICURVE(sig_gwn):
  pars = default_pars(T=1000.)
  I_mean = np.arange(100., 400., 10.)
  spk_count = np.zeros(len(I_mean))
  spk_count_dc = np.zeros(len(I_mean))

  for idx in range(len(I_mean)):
      I_GWN = my_GWN(pars, mu=I_mean[idx], sig=sig_gwn, myseed=2020)
      v, rec_spikes = run_LIF(pars, Iinj=I_GWN)
      v_dc, rec_sp_dc = run_LIF(pars, Iinj=I_mean[idx])
      spk_count[idx] = len(rec_spikes)
      spk_count_dc[idx] = len(rec_sp_dc)
  plt.figure()
  plt.plot(I_mean, spk_count, 'k')
  plt.plot(I_mean, spk_count_dc, 'k--')
  plt.ylabel('Spike count')
  plt.xlabel('Average injected current (pA)')
  plt.legend(loc='best')
  plt.show()
  return
FICURVE(3)
```

For example, for sigma=3 we  will get:



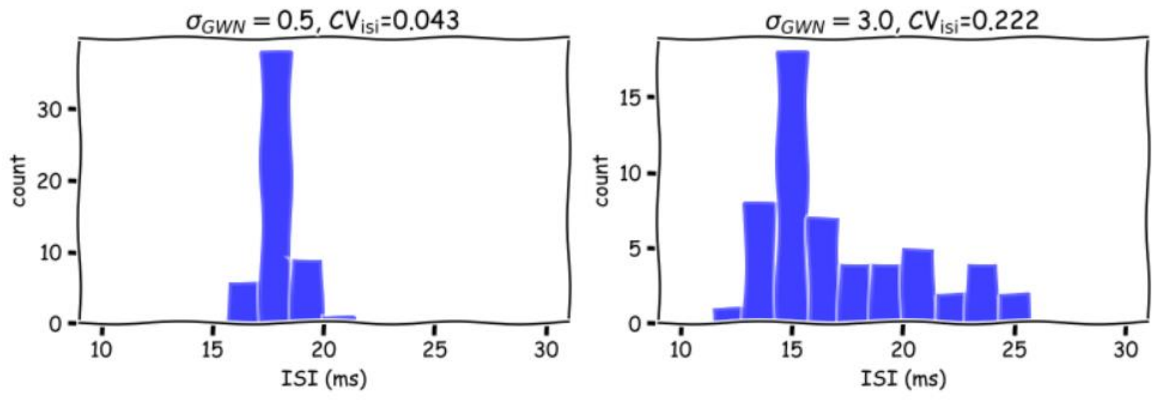
And for sigma=1 we have:



as we can see, by decreasing  sigma we are getting closer to our DC input.as we increase sigma, we are getting away from the average DC input.

## Compute $CV_{ISI}$ values

After filling the missing parts, the output plots will be as below:



## Spike irregularity explorer for different `sig_gwn`

How different levels of fluctuation $\sigma$ affect the CVs for different average injected currents ($\mu$)?

1. Does the standard deviation of the injected current affect the F-I curve in any qualitative manner?

Yes.as we saw; we have a nonlinear relation with DC input. But by adding GWN and increasing sigma, we can decrease this nonlinearity.in fact, the noise smooths the diagram.

2. Why does increasing the mean of GWN reduce the CVISIISI?

By increasing the mean of gwn, the effective input mean is above spike threshold at some points. When the input is high, the neuron charges up to the spike threshold. Which causes the regular spiking.

3. If you plot spike count (or rate) vs. CVISIISI, should there be a relationship between the two? Try out yourself

High firing rate causes high GWN mean. The relation between GWN mean and firing rate is direct and opposite with CV_ISI.so by increasing firing rate spike irregularity decreases and the reason is the spike threshold.

## Ornstein-Uhlenbeck Process

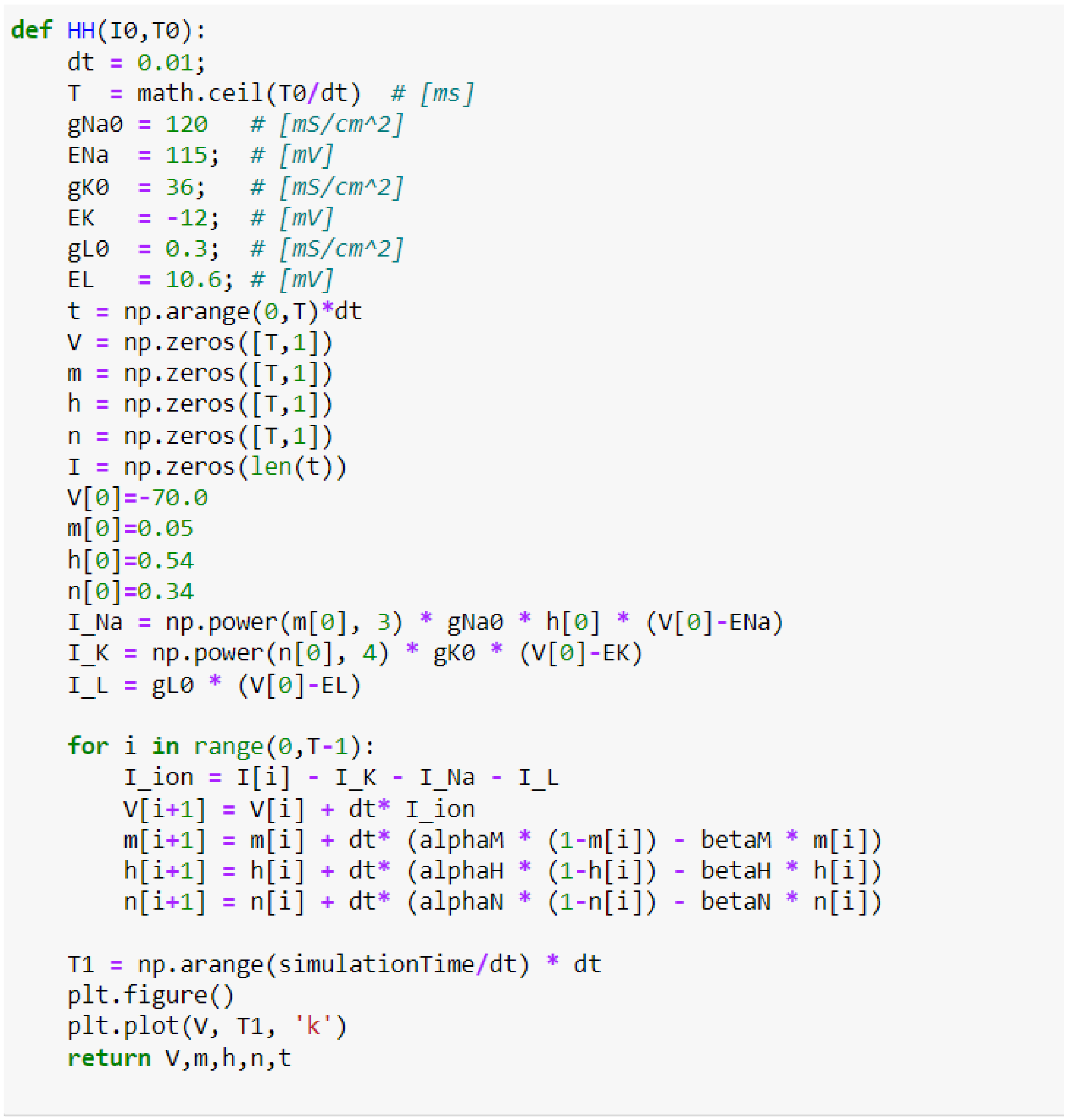
- How does the OU type input change neuron responsiveness?
- What do you think will happen to the spike pattern and rate if you increased or decreased the time constant of the OU process?
- We expect the firing rate decrease and the spiking be more regular when the time constant of the OU process is very long and the input current is very low. And by increasing the time constant we expect firing rate and CV_ISI to decrease. As we know, membrane time constant is also important and we can conclude that the same result will happen for the membrane as well.

## Extensions to Integrate-and-Fire models

The integrate and fire neuron model is a widely used method to analyze a neural system.it describes the membrane potential of a neuron in terms of synaptic inputs and the injected current that it receives.

## The Hodgkin-Huxley model

In this section, we add some extra inputs like current to our code:



```
def HH(I0,T0):
    dt = 0.01;
    T  = math.ceil(T0/dt)  # [ms]
    gNa0 = 120   # [mS/cm^2]
    ENa  = 115;  # [mV]
    gK0  = 36;   # [mS/cm^2]
    EK   = -12;  # [mV]
    gL0  = 0.3;  # [mS/cm^2]
    EL   = 10.6; # [mV]
    t = np.arange(0,T)*dt
    V = np.zeros([T,1])
    m = np.zeros([T,1])
    h = np.zeros([T,1])
    n = np.zeros([T,1])
    I = np.zeros(len(t))
    V[0]=-70.0
    m[0]=0.05
    h[0]=0.54
    n[0]=0.34
    I_Na = np.power(m[0], 3) * gNa0 * h[0] * (V[0]-ENa)
    I_K = np.power(n[0], 4) * gK0 * (V[0]-EK)
    I_L = gL0 * (V[0]-EL)

    for i in range(0,T-1):
        I_ion = I[i] - I_K - I_Na - I_L
        V[i+1] = V[i] + dt* I_ion
        m[i+1] = m[i] + dt* (alphaM * (1-m[i]) - betaM * m[i])
        h[i+1] = h[i] + dt* (alphaH * (1-h[i]) - betaH * h[i])
        n[i+1] = n[i] + dt* (alphaN * (1-n[i]) - betaN * n[i])

    T1 = np.arange(simulationTime/dt) * dt
    plt.figure()
    plt.plot(V, T1, 'k')
    return V,m,h,n,t
```